Rapport CRV : Projet AutoScaling & Infrastructure as Code

MU4IN019 – Cloud et Réseaux Virtuels
21118889 BENYAHIA Ilyas

1. Introduction

Ce projet a pour objectif de concevoir et de déployer une infrastructure scalable sur Kubernetes. L'architecture repose sur plusieurs composants interdépendants, notamment un backend en Node.js, une base de données Redis, un frontend React ainsi qu'un système de monitoring avec Prometheus et Grafana. L'ensemble est conçu pour être **automatiquement scalable** et **reproductible**, grâce à des fichiers de configuration Kubernetes en YAML et une gestion dynamique des ressources via les **Horizontal Pod Autoscalers (HPA)**.

2. Architecture et Technologies

L'infrastructure repose sur plusieurs composants essentiels :

- **Backend** : Application Node.js stateless permettant d'interagir avec Redis.
- **Base de données** : Redis en mode **single instance**, avec un exportateur de métriques pour Prometheus.
- **Frontend** : Application React servie par Nginx, déployée sous forme de container sur Kubernetes.
- **Monitoring** : Prometheus pour la collecte des métriques et Grafana pour la visualisation.
- **Montée à l'échelle** : Autoscaling horizontal basé sur l'utilisation CPU des pods.

Chaque composant est déployé sous forme de conteneur basé sur les images suivantes :

Composant	Image utilisée
Backend Node.js	`arthurescriou/node-redis:1.0.5`
Base de données Redis	`redis:latest`
Exportateur de métriques Redis	`oliver006/redis_exporter`
Frontend React (hébergé par Nginx)	`cedricxxx/redis-react:latest`
Prometheus	`prom/prometheus:latest`
Grafana	`grafana/grafana:latest`

Chaque image a été choisie pour garantir compatibilité, stabilité et performance dans l'environnement Kubernetes.

3. Déploiement Kubernetes

3.1 Organisation des ressources

L'infrastructure est entièrement définie à travers des fichiers **YAML**, ce qui permet une **reproductibilité complète** du déploiement sur n'importe quel cluster Kubernetes.

Les principaux objets Kubernetes utilisés :

1. **Deployments** :
    - Déploiement de Redis, du backend Node.js et du frontend React.
    - Utilisation d’un **Horizontal Pod Autoscaler (HPA)** pour Redis et le backend Node.js.
2. **Services** :
    - Redis et Prometheus sont exposés en **ClusterIP** (accès interne au cluster).
    - Grafana et le frontend React sont exposés en **LoadBalancer** pour un accès externe.
3. **ConfigMaps** :
    - Utilisés pour stocker les configurations de Prometheus et Grafana.

3.2 Déploiement détaillé

3.2.1 Base de données Redis

Redis est déployé avec un **exportateur de métriques** permettant à Prometheus de collecter des statistiques sur l’utilisation de Redis. Un **HPA (Horizontal Pod Autoscaler)** ajuste dynamiquement le nombre de pods Redis entre **3 et 10** en fonction de l'utilisation CPU.

3.2.2 Backend Node.js

L’application backend est conçue pour être **stateless**, ce qui permet son **scaling horizontal** sans problème. Un **HPA** ajuste dynamiquement le nombre de pods entre **2 et 10** en fonction de la charge CPU.

3.2.3 Monitoring avec Prometheus et Grafana

- **Prometheus** collecte les métriques des services Redis, du backend et du frontend.
- **Grafana** permet de visualiser ces données via des dashboards.

4. Stratégies de montée à l’échelle

Le système est conçu pour s’adapter dynamiquement à la charge via Kubernetes **Horizontal Pod Autoscaler (HPA)**.

4.1 Montée à l’échelle du backend Node.js

Lorsqu'un pic de charge CPU est détecté, Kubernetes augmente le nombre de pods backend. Cela permet de gérer une augmentation soudaine du trafic sans interruption et améliore la tolérance aux pannes en cas de surcharge.

4.2 Montée à l’échelle de Redis

Redis peut voir sa capacité de lecture étendue en augmentant le nombre de réplicas. Cela permet d’absorber plus de requêtes sans impacter les performances d'écriture et permet une gestion efficace des ressources en fonction des besoins.

5. Conclusion

L’un des points clés de ce projet est sa **reproductibilité** grâce à une infrastructure définie **entièrement en YAML**. Le déploiement **facilement réplicable** sur n'importe quel cluster Kubernetes. On peut aussi avoir une gestion **versionnée** de l’infrastructure via Git.

Ce projet montre comment concevoir une **infrastructure scalable** sur Kubernetes en combinant **Node.js, Redis, React, Prometheus et Grafana**. **Autoscaling dynamique** pour le backend et Redis. **Monitoring avancé** avec Prometheus et Grafana. **Infrastructure as Code** pour un **déploiement reproductible et optimisé**.