Rapport de projet - Déploiement Kubernetes avec Monitoring

Objectif du projet

Déployer une application composée de :

- **Backend Node.js** connectée à **Redis** pour la gestion des données.
- **Frontend React** pour l'interface utilisateur.
- **Exécution sous Kubernetes** via **Minikube**.
- **Surveillance** avec **Prometheus** et **Grafana**.
- **Autoscaling** avec l'**Horizontal Pod Autoscaler (HPA)** pour ajuster dynamiquement les ressources en fonction de la charge.

Stack technique

- **Base de données** : **Redis** (pour la gestion des données persistantes).
- **Serveur backend** : **Node.js** avec une API REST (et un point /metrics pour Prometheus).
- **Frontend** : **React**.
- **Orchestration** : **Kubernetes via Minikube**.
- **Monitoring** : **Prometheus** et **Grafana**.
- **Autoscaling** : **Metrics Server** et **HPA**.

Etapes initiales avec Docker

Redis :

- Commandes Docker classiques pour démarrer Redis.

Backend Node.js :

1. Clonage du dépôt.
2. Création du **Dockerfile** et construction de l'image.
3. Utilisation de **docker-compose** pour la configuration multi-conteneurs.

Frontend React :

1. Clonage du dépôt et modification du fichier de configuration (src-conf.js).
2. Construction et exécution de l'image React via Docker.

Passage à Kubernetes

Pré-requis :

- Initialisation de Minikube et chargement des images Docker dans le cluster.

Déploiement Kubernetes :

- Déploiement de Redis, Node.js et React via des fichiers YAML pour les **Deployments**, **Services**, et **Horizontal Pod Autoscaler (HPA)**.
- Configuration de **Prometheus** et **Grafana** avec leurs fichiers de déploiement respectifs.

Monitoring : Prometheus & Grafana

Prometheus :

- Configuration via **ConfigMap** pour indiquer où récupérer les métriques des services.
- Accès via l'URL locale de Minikube (http://localhost:9090/targets) pour vérifier les services "UP".

Grafana :

- Configuration de la datasource pour Prometheus.
- Accès via Minikube pour visualiser les métriques dans un tableau de bord.

Autoscaling (HPA)

- **Metrics Server** activé pour collecter les métriques du cluster.
- **HPA** ajustant le nombre de pods du backend Node.js et du frontend React en fonction de l'utilisation CPU et des autres métriques.

Persistance des données

- **Redis**, utilisé comme base de données, est configuré pour assurer la persistance des données en cas de redémarrage des pods. Les données sont stockées sur un **Persistent Volume (PV)**, ce qui garantit leur conservation même si le pod Redis est supprimé ou redémarré.

Notes importantes

- **Minikube tunnel** est essentiel pour exposer les services localement.
- Accès aux services via les URL générées par Minikube (minikube service ...).

Conclusion

- Déploiement complet des composants **backend**, **frontend**, et **base de données** sur Kubernetes.
- Mise en place d'un **monitoring** complet avec **Prometheus** et **Grafana** pour la surveillance des services.
- Implémentation de **l'autoscaling** via **HPA** pour une gestion dynamique des ressources en fonction de la charge.
- La **persistance des données** est assurée grâce à Redis, avec une configuration de **Persistent Volumes**.

Technologies utilisées

- **Minikube** : Cluster Kubernetes local.
- **Kubernetes** : Orchestration des conteneurs.
- **Prometheus** : Collecte des métriques.
- **Grafana** : Visualisation des métriques.
- **Docker** : Conteneurisation des applications.
- **Redis** : Base de données NoSQL avec persistance.
