

Rapport de Synthèse FinOps Kubernetes avec Kubecost

Ce rapport présente notre démarche pour optimiser les coûts d'une application MERN déployée sur Kubernetes, en utilisant les principes FinOps et les outils Kubecost, Prometheus et Grafana. Nous explorerons la mise en place, le déploiement, et l'analyse des métriques pour une gestion financière et technique efficace.

 *Projet réalisé par :*

- *BOUCHAREB Imad*
- *HALLOUCHE Walid*
- *LOUNISSI Asma*



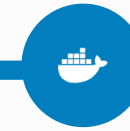
Mise en place de l'environnement

Pour simuler un environnement cloud de manière locale et maîtriser nos coûts, nous avons mis en place plusieurs outils essentiels :



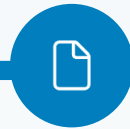
WSL2 + Ubuntu

Simulation d'un environnement Linux sous Windows.



Docker

Création des conteneurs (frontend, backend, MongoDB).




Minikube & kubectl

Simulation et pilotage du cluster Kubernetes localement.



Helm

Installation simplifiée de nos outils de surveillance.

✅  **Apport FinOps** : Cette configuration nous a permis de simuler précisément les coûts cloud, tout en mesurant l'utilisation des ressources sans engendrer de frais réels.

Dockerisation de l'application MERN

L'application source MERN (MongoDB, Express.js, React, Node.js) a été dockerisée pour garantir la portabilité et la cohérence de l'environnement.

Frontend

Application React conteneurisée.

Backend

Application Express.js + Node.js conteneurisée.

MongoDB

Utilisation de l'image Docker officielle avec volumes persistants.

La configuration inclut l'exposition des ports, la gestion des variables d'environnement et l'utilisation de volumes persistants pour MongoDB, assurant ainsi la persistance des données.



📄 **Apport FinOps** : La dockerisation standardise le déploiement, permet une maîtrise accrue de la consommation (taille d'image, CPU/RAM) et offre une meilleure observabilité.

Déploiement Kubernetes

Le déploiement de l'application MERN sur Kubernetes a été réalisé via des fichiers YAML, essentiels pour orchestrer les différents composants.

1

Deployment

Gère le cycle de vie des pods, assurant la scalabilité.

2


Service

Fournit un accès stable et constant aux pods de l'application.

3

Ingress

Route le trafic HTTP externe vers les services appropriés.

✔  **Apport FinOps** : Le déploiement Kubernetes permet une gestion fine des ressources via `resources.requests` et `resources.limits` et{" "}
`resources.limits`, facilitant les ajustements basés sur les métriques observées.

Installation de KubeCost

KubeCost a été installé via Helm pour une analyse détaillée des coûts du cluster Kubernetes.

```
helm repo add kubecost https://kubecost.github.io/cost-analyzer/helm install kubecost kubecost/cost-analyzer --namespace kubecost --create-namespace kubectl port-forward -n kubecost deployment/kubecost-cost-analyzer 9090
```

Problèmes rencontrés et solutions :

Pod CrashLoopBackOff



Réinstallation et vérification du namespace.

Erreur nginx (Grafana)

Installation manquante de Grafana.

Accès au port 9090

Redirection via `kubectl port-forward`.




  **Apport FinOps** : KubeCost offre une analyse granulaire des coûts par pod, namespace et workload, et fournit des suggestions précieuses pour l'optimisation des dépenses.

Analyse Kubecost – Résultats Actualisés



Suite à l'intégration de Prometheus et Grafana, les coûts ont évolué, révélant des opportunités d'optimisation.

Kubernetes Costs	0,03 \$US	4,79 \$US
Total Costs	0,03 \$US	4,79 \$US
Monthly Savings	1,81 \$US/mo	1,96 \$US/mo
Cluster Efficiency	0%	0%

L'augmentation des coûts est due à l'ajout de composants de monitoring.

- **Recommandations de Savings :**
-  **Right-size** : Ajuster les requêtes CPU/RAM pour une meilleure adéquation aux besoins réels.
-  **Suppression** : Éliminer les workloads et volumes persistants inutilisés.
-  **Réduction** : Diminuer la taille des disques quand cela est possible.

La vue Allocations permet une analyse granulaire par Namespace, Controller, et Pod, révélant la consommation précise par conteneur.

  **Apport FinOps** : Ces vues permettent une visualisation granulaire des dépenses et une identification ciblée des ressources sous-utilisées.

Intégration Prometheus + Grafana

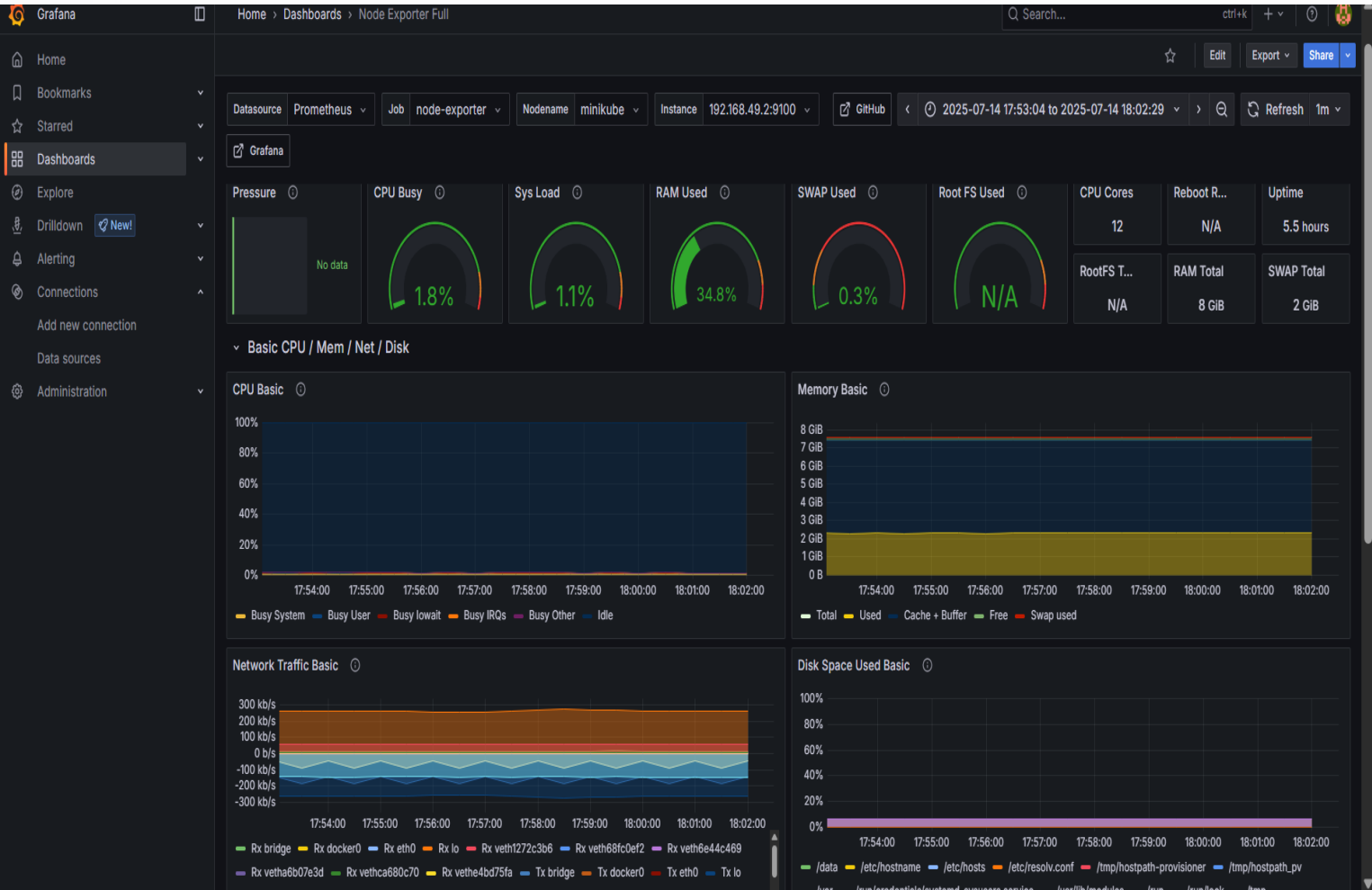
L'intégration de Prometheus et Grafana via Helm offre une observabilité complète des métriques techniques.

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm install prometheus prometheus-community/kube-prometheus-stack --namespace monitoring --create-namespace
```

Récupération du mot de passe Grafana et accès :

```
kubectl -n monitoring get secret prometheus-grafana -o jsonpath="{.data.admin-password}" | base64 -d ; echo
kubectl port-forward svc/prometheus-grafana -n monitoring 3000:80
```

Le dashboard ID [Grafana.com](https://grafana.com/dashboards/1860) : 1860 a été importé, offrant une vue complète sur le CPU, la RAM, les disques et le réseau.



Conclusion Finale

1 Dockerisation & Déploiement

Application MERN complète dockerisée et déployée sur Minikube.

2 Suivi des Coûts

Kubecost pour l'analyse des coûts et les recommandations FinOps.

3 Visualisation des Métriques

Grafana et Prometheus pour le monitoring système en temps réel.

4 Optimisation FinOps

Identification précise des axes d'optimisation des ressources.

Ce projet a démontré l'efficacité de l'approche FinOps combinée à des outils de monitoring robustes pour une gestion optimisée des ressources et des coûts dans un environnement Kubernetes.