



Projet : Infrastructure as Code et AutoScaling

Cloud et Réseaux Virtuels

Ahcene Loubar

Master 1 Informatique – Parcours Réseaux
Sorbonne Université

Encadrant : Arthur Escriou

6 avril 2025

Table des matières

1	Introduction	4
2	Installation des prérequis	5
2.1	Nettoyage de l'environnement Docker	5
2.2	Installation de Docker	6
2.3	Installation de kubectl et Minikube	7
3	Retour d'expérience sur l'installation : erreurs et ajustements	9
3.1	Nettoyage Docker	9
3.2	Téléchargement et configuration de Docker	9
3.3	Installation de kubectl	9
3.4	Minikube	10
3.5	Authentification DockerHub	10
3.6	Conclusion sur les difficultés	10
4	Déploiement des composants	11
4.1	Déploiement de Redis (Master/Replica)	11
4.1.1	Fichiers YAML utilisés	11
4.1.2	Déploiement	11
4.1.3	Test de la réplication	12
4.2	Déploiement du Backend Node.js	13
4.2.1	Création des fichiers de déploiement Kubernetes	13

4.2.2	Configuration de l'auto-scaling du backend	13
4.2.3	Construction et publication de l'image Docker	14
4.2.4	Déploiement Kubernetes	14
4.2.5	Vérification de l'exécution du backend	14
4.3	Déploiement du Frontend React	14
4.3.1	Construction et publication	15
4.3.2	Déploiement sur le cluster	15
4.3.3	Configuration et déploiement du frontend	15
5	Monitoring avec Prometheus et Grafana	17
5.1	Choix de la stratégie de monitoring	17
5.2	Configuration de Prometheus	17
5.2.1	Création des fichiers Prometheus	17
5.2.2	Fichier de configuration : <code>prometheus-cfg.yaml</code>	18
5.2.3	Déploiement de Prometheus	18
5.3	Déploiement de Grafana	19
5.3.1	Déploiement sur le cluster	19
5.3.2	Création d'un dashboard	20
6	Versionnement, Reproductibilité et Conclusion	21
6.1	Mise en place du dépôt GitHub	21
6.1.1	Commandes utilisées	22
6.2	Reproductibilité du projet	22
6.3	Bilan et conclusion	22
7	Annexe : Détails supplémentaires et observations sur les captures	24
7.1	Initialisation de Minikube	24
7.2	Déploiement de Redis	24

7.3	Création et déploiement du backend Node.js	25
7.4	Publication d'une image sur Docker Hub	25
7.5	Déploiement du frontend React	26
7.6	Monitoring avec Prometheus et Grafana	26

1. Introduction

L'objectif de ce projet est de concevoir, déployer et superviser une infrastructure logicielle moderne reposant sur les principes de l'Infrastructure as Code (IaC) et du scaling automatique. Le tout est orchestré dans un cluster Kubernetes local à l'aide de Minikube.

L'infrastructure cible est composée de plusieurs composants interdépendants :

- Une base de données Redis en mode maître/réplicas, optimisée pour la lecture/écriture.
- Un serveur backend stateless développé en Node.js, capable de monter à l'échelle horizontalement.
- Un frontend React servant d'interface de test.
- Un outil de monitoring basé sur Prometheus et Grafana pour observer en temps réel l'état du système.

Tout le projet suit une approche déclarative, où chaque ressource est définie sous forme de fichiers YAML, assurant ainsi la reproductibilité du déploiement.

Ce rapport détaille les étapes suivies, les choix techniques réalisés, les difficultés rencontrées, ainsi que les mécanismes de montée en charge et de supervision mis en place.

2. Installation des prérequis

Avant de pouvoir déployer l'infrastructure sur Kubernetes, il est nécessaire de préparer un environnement local adapté. Cela comprend l'installation de Docker, de l'outil de gestion de clusters Kubernetes `kubectl`, ainsi que Minikube qui permet de simuler un cluster Kubernetes local.

Les objectifs de cette phase sont :

- Nettoyer l'environnement pour éviter les conflits de versions.
- Installer une version stable et récente de Docker.
- Déployer les outils nécessaires à la gestion et à la simulation du cluster Kubernetes.

2.1 Nettoyage de l'environnement Docker

La première étape consiste à supprimer tout composant Docker ou containerd précédemment installé afin d'éviter des conflits lors de l'installation propre.

```
for pkg in docker.io docker-doc docker-compose-v2 podman-  
docker containerd runc; do sudo apt-get remove $pkg; done
```

```

root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/backend# docker login

USING WEB-BASED LOGIN

Info → To sign in with credentials on the command line, use 'docker login -u <username>'

Your one-time device confirmation code is: DTPM-PBCM
Press ENTER to open your browser or submit your device code here: https://login.docker.com/activate

Waiting for authentication in the browser...
^Clogin canceled
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/backend# docker login

USING WEB-BASED LOGIN

Info → To sign in with credentials on the command line, use 'docker login -u <username>'

Your one-time device confirmation code is: XWLV-GTRG
Press ENTER to open your browser or submit your device code here: https://login.docker.com/activate

Waiting for authentication in the browser...

WARNING! Your credentials are stored unencrypted in '/root/.docker/config.json'.
Configure a credential helper to remove this warning. See
https://docs.docker.com/go/credential-store/

```

FIGURE 2.1 – Suppression des anciens paquets Docker

2.2 Installation de Docker

L'installation de Docker se fait en plusieurs étapes :

1. Ajout des clés GPG et du dépôt officiel Docker.
2. Mise à jour des paquets.
3. Installation du moteur Docker, de Docker Compose et de plugins utiles.

```

sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /
  etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

```

Ajout de la source APT :

```

echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/
  apt/keyrings/docker.asc] https://download.docker.com/linux
  /ubuntu \
$(. /etc/os-release && echo "$UBUNTU_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

```

Installation :

```

sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
  docker-buildx-plugin docker-compose-plugin

```

Vérification de l'installation :

```
docker run hello-world
```

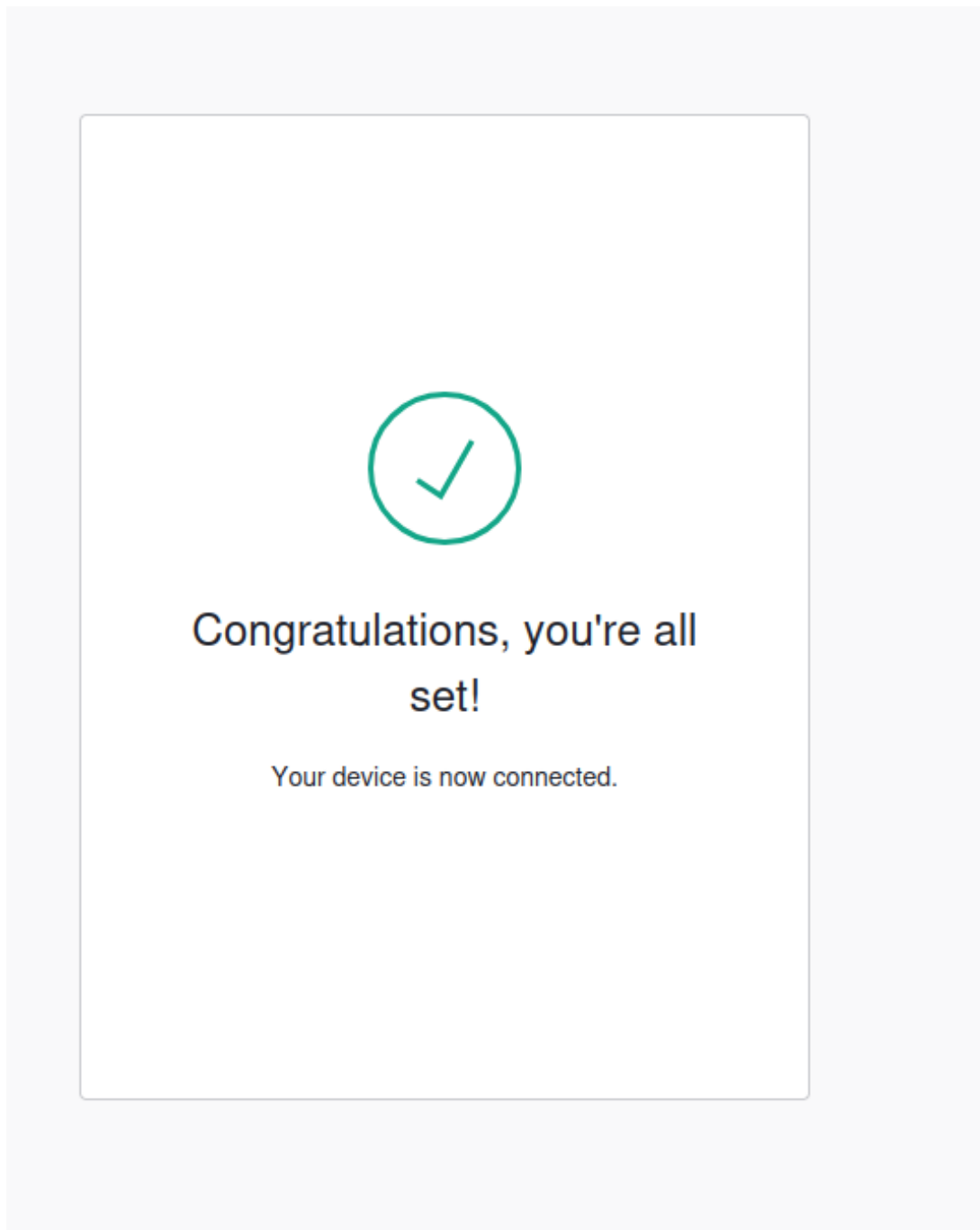


FIGURE 2.2 – Docker installé avec succès

2.3 Installation de kubectl et Minikube

Une fois Docker opérationnel, on installe les outils nécessaires à la gestion et à la simulation du cluster Kubernetes :

- `kubectl` : client CLI pour Kubernetes
- `minikube` : environnement local de type cluster

Installation de `kubectl` :

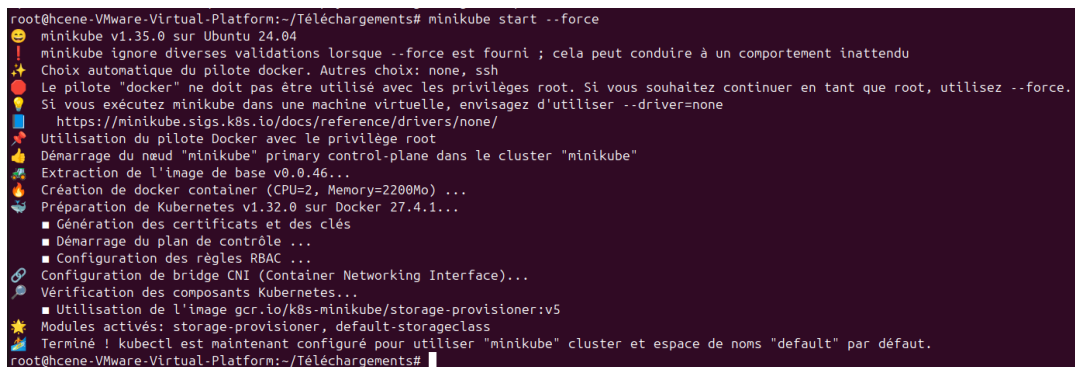
```
curl -LO https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
```

Installation de `minikube` :

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
mv minikube-linux-amd64 minikube
chmod +x minikube
sudo mv minikube /usr/local/bin/
```

Démarrage du cluster local :

```
minikube start --force
```



```
root@hcene-VMware-Virtual-Platform:~/Téléchargements# minikube start --force
🐹 minikube v1.35.0 sur Ubuntu 24.04
! minikube ignore diverses validations lorsque --force est fourni ; cela peut conduire à un comportement inattendu
🌟 Choix automatique du pilote docker. Autres choix: none, ssh
🔴 Le pilote "docker" ne doit pas être utilisé avec les privilèges root. Si vous souhaitez continuer en tant que root, utilisez --force.
💡 Si vous exécutez minikube dans une machine virtuelle, envisagez d'utiliser --driver=none
🔗 https://minikube.sigs.k8s.io/docs/reference/drivers/none/
🔴 Utilisation du pilote Docker avec le privilège root
🔴 Démarrage du nœud "minikube" primary control-plane dans le cluster "minikube"
📦 Extraction de l'image de base v0.0.46...
🔥 Création de docker container (CPU=2, Memory=2200Mo) ...
🔧 Préparation de Kubernetes v1.32.0 sur Docker 27.4.1...
  ■ Génération des certificats et des clés
  ■ Démarrage du plan de contrôle ...
  ■ Configuration des règles RBAC ...
🔗 Configuration de bridge CNI (Container Networking Interface)...
🔍 Vérification des composants Kubernetes...
  ■ Utilisation de l'image gcr.io/k8s-minikube/storage-provisioner:v5
🌟 Modules activés: storage-provisioner, default-storageclass
🔥 Terminé ! kubectl est maintenant configuré pour utiliser "minikube" cluster et espace de noms "default" par défaut.
root@hcene-VMware-Virtual-Platform:~/Téléchargements#
```

FIGURE 2.3 – Minikube démarré

3. Retour d'expérience sur l'installation : erreurs et ajustements

L'installation des prérequis a été semée d'embûches. Ce chapitre documente l'ensemble des commandes réellement exécutées, les erreurs rencontrées, ainsi que les solutions apportées pour parvenir à un environnement fonctionnel.

3.1 Nettoyage Docker

```
for pkg in docker.io docker-doc docker-compose-v2 podman-  
docker containerd runc; do sudo apt-get remove $pkg; done
```

Les retours système indiquaient que certains paquets n'étaient pas installés, ou introuvables (`contained` au lieu de `containerd` par exemple). Il a fallu ajuster les noms et relancer les commandes.

3.2 Téléchargement et configuration de Docker

Suite à l'ajout du dépôt Docker, l'installation s'est déroulée correctement, bien que plusieurs dépendances supplémentaires aient été installées (comme Git, pigz, etc.).

...

3.3 Installation de kubectl

L'installation manuelle du binaire a échoué plusieurs fois à cause de commandes mal formatées, de fichiers absents ou de permissions manquantes. Voici un extrait des erreurs rencontrées :

```
cat: kubectl.sha256: Aucun fichier ou dossier de ce nom  
sha256sum: 'entr e standard' : n'a trouv aucune ligne de  
somme de contr le correctement format e
```

```
chmod: impossible d'accéder à 'kubectl': Aucun fichier ou dossier de ce nom
```

Après plusieurs essais et corrections, la commande suivante a finalement permis de récupérer une version fonctionnelle :

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"  
chmod +x kubectl  
mv kubectl /usr/local/bin/
```

3.4 Minikube

La configuration avec `-force` a permis de contourner l'exécution en tant que root dans un environnement VM, bien que le message d'avertissement recommande d'utiliser `-driver=none`.

3.5 Authentification DockerHub

L'authentification initiale avec mot de passe a échoué, car GitHub a désactivé cette méthode. Il a donc fallu créer une clé SSH pour pouvoir effectuer le push vers le dépôt distant.

```
ssh-keygen -t ed25519 -C "hcneloubar@gmail.com"
```

3.6 Conclusion sur les difficultés

Ces erreurs, bien que nombreuses, ont permis de mieux comprendre le fonctionnement bas niveau des outils utilisés. Elles ont également mis en évidence l'importance de la rigueur et de la documentation dans les projets d'Infrastructure as Code.

4. Déploiement des composants

L'objectif de cette étape est de déployer les différents services de notre infrastructure sur Kubernetes. L'architecture repose sur 3 briques principales :

- **Redis** : base de données clé-valeur avec un maître (write) et des répliques (read).
- **Backend Node.js** : API REST stateless qui interagit avec Redis.
- **Frontend React** : application cliente pour tester le fonctionnement global.

4.1 Déploiement de Redis (Master/Replica)

Redis est déployé en deux entités distinctes :

- **redis-maitre** : unique, accepte les écritures.
- **redis-esclave** : répliques en lecture seule avec montée en charge automatique.

4.1.1 Fichiers YAML utilisés

Les fichiers utilisés :

- `redis-maitre-deployment.yaml`
- `redis-maitre-service.yaml`
- `redis-esclave-deployment.yaml`
- `redis-esclave-service.yaml`
- `redis-esclave-auto-scaling.yaml`

4.1.2 Déploiement

```
kubectl apply -f redis-maitre-deployment.yaml
kubectl apply -f redis-maitre-service.yaml
kubectl apply -f redis-esclave-deployment.yaml
kubectl apply -f redis-esclave-service.yaml
kubectl apply -f redis-esclave-auto-scaling.yaml
```

```

root@hcene-VMware-Virtual-Platform:~# cd Documents
root@hcene-VMware-Virtual-Platform:~/Documents# cd CRV-projet/fichiers.yaml
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers.yaml# kubectl apply -f redis-maitre-deployment.yaml
kubectl apply -f redis-maitre-service.yaml
kubectl apply -f redis-esclave-deployment.yaml
kubectl apply -f redis-esclave-service.yaml
kubectl apply -f redis-esclave-auto-scaling.yaml
deployment.apps/redis-maitre created
service/redis-maitre created
deployment.apps/redis-esclave created
service/redis-esclave created
horizontalpodautoscaler.autoscaling/redis-esclave-auto-scaling created
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers.yaml# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
redis-esclave-7b448c7f7b-rcsd1     1/1     Running   0           22s
redis-maitre-794c58957c-gv869     1/1     Running   0           23s
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers.yaml#

```

FIGURE 4.1 – Redis master et réplicas déployés

4.1.3 Test de la réplication

Afin de s'assurer que les données insérées dans le maître Redis sont bien propagées aux réplicas, un test simple est réalisé.

Dans le pod maître :

```

kubectl exec -it deploy/redis-maitre -- redis-cli
> set test "Ahcene LOUBAR"

```

Puis dans un des pods réplicas, on exécute une boucle de lecture continue :

```

kubectl exec -it deploy/redis-esclave -- sh
> while true; do redis-cli get test; done

```

Cette commande affiche en continu la valeur stockée, prouvant que la réplication fonctionne en temps réel.

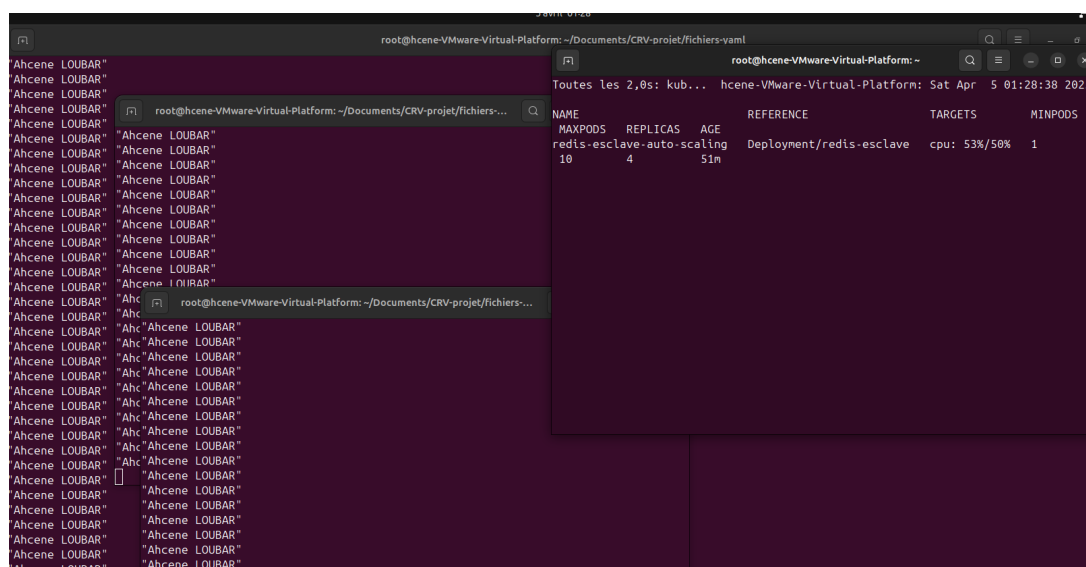


FIGURE 4.2 – Résultat du test de réplication entre le maître et les réplicas Redis

Juste après cette opération, il est possible de vérifier à nouveau l'état des pods Redis, notamment les ressources consommées ou les événements associés.

```
Toutes les 2,0s: kub... hcene-VMware-Virtual-Platform: Sat Apr 5 01:21:23 202
```

NAME	REFERENCE	TARGETS	MINPODS
redis-esclave-auto-scaling	Deployment/redis-esclave	cpu: 70%/50%	1
MAXPODS			
10			
REPLICAS			
4			
AGE			
44m			

FIGURE 4.3 – Vérification de l'état et des détails des pods Redis après la réplication

4.2 Déploiement du Backend Node.js

Le serveur Node.js étant stateless, il est duplicable facilement. Il sert d'interface entre les clients frontend et la base Redis (master ou réplica).

4.2.1 Création des fichiers de déploiement Kubernetes

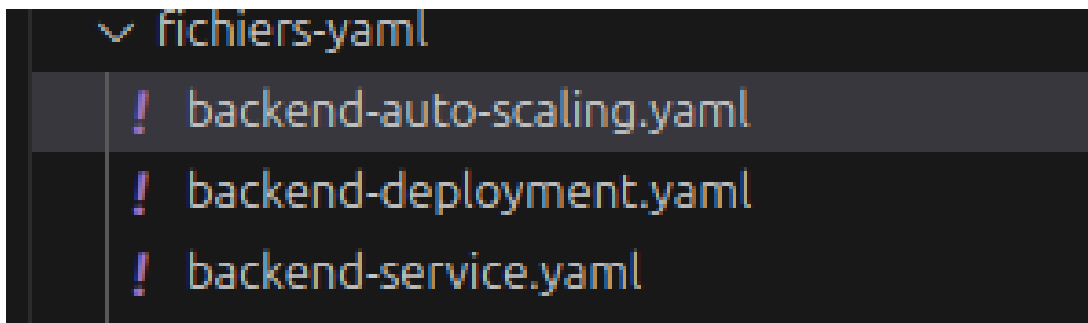


FIGURE 4.4 – Création des fichiers YAML pour le backend Node.js

4.2.2 Configuration de l'auto-scaling du backend

```
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet# cd fichiers-yaml
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers-yaml# kubectl apply -f backend-deployment.yaml
kubectl apply -f backend-service.yaml
kubectl apply -f backend-auto-scaling.yaml
deployment.apps/backend created
service/backend created
horizontalpodautoscaler.autoscaling/backend-auto-scaling created
```

FIGURE 4.5 – Définition du HPA pour le backend Node.js

4.2.3 Construction et publication de l'image Docker

L'image Docker est créée à partir du dépôt <https://github.com/arthurescriou/redis-node>.

```
docker build -t backend-image .  
docker login  
docker tag backend-image hcene/backend-image  
docker push hcene/backend-image
```

```
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/backend# ^C  
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/backend# docker build -t backend-image .  
[+] Building 56.4s (10/10) FINISHED
```

FIGURE 4.6 – Construction de l'image Docker pour le backend

4.2.4 Déploiement Kubernetes

```
kubectl apply -f backend-deployment.yaml  
kubectl apply -f backend-service.yaml  
kubectl apply -f backend-auto-scaling.yaml
```

4.2.5 Vérification de l'exécution du backend

```
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers-yaml# minikube service backend --url  
http://192.168.49.2:30540  
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers-yaml#
```

FIGURE 4.7 – Le backend Node.js fonctionne dans le cluster Kubernetes



The screenshot shows a web browser window with the address bar displaying '192.168.49.2:30540'. The page content reads: 'It is working, good job a3490f9a-9134-4032-a3d4-e642c57f09d8 1743858501194'. This indicates that the backend service is running correctly in the Kubernetes cluster.

FIGURE 4.8 – Déploiement du backend Node.js

4.3 Déploiement du Frontend React

Le frontend est utilisé pour tester l'infrastructure. Il est basé sur le dépôt <https://github.com/arthurescriou/redis-react>, compilé en une app statique.

4.3.1 Construction et publication

```
docker build -t frontend-image .
docker tag frontend-image hcene/frontend-image
docker push hcene/frontend-image
```

4.3.2 Déploiement sur le cluster

```
kubectl apply -f frontend-deployment.yaml
kubectl apply -f frontend-service.yaml
```

4.3.3 Configuration et déploiement du frontend

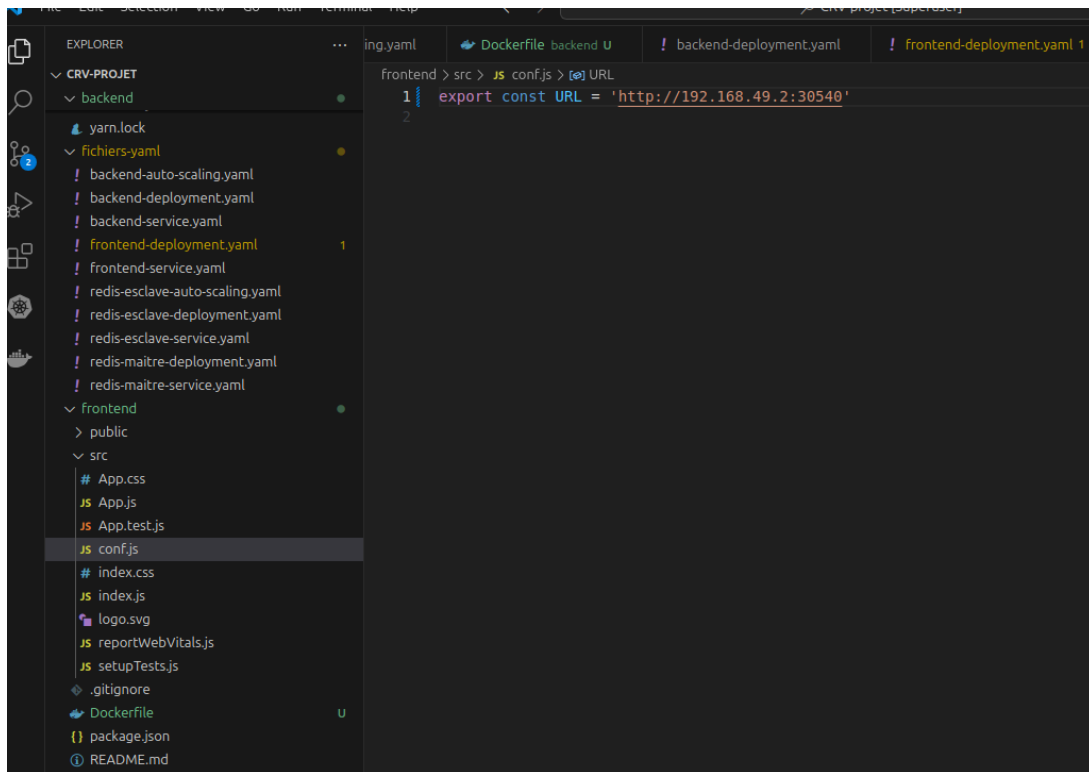


FIGURE 4.9 – Fichiers YAML de déploiement du frontend React

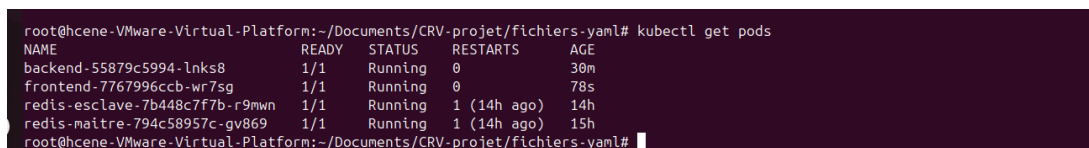


FIGURE 4.10 – Affichage des pods frontend via kubectl


```
root@ahcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers-yaml# minikube service frontend --url  
http://192.168.49.2:30888  
root@ahcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers-yaml#
```

FIGURE 4.11 – Accès au frontend depuis un navigateur via le NodePort

key:
redis connected to redis://redis-maitre:6379

New key

key:
Value:
add

FIGURE 4.12 – Interface React fonctionnelle via le service Kubernetes

Ahcene:
LOUBAR

Mortada :
(empty)

6:
7

Hocine:
LOUBAR

key:
redis connected to redis://redis-maitre:6379

New key

key:
Value:
add

ointer inside or press Ctrl+G.

FIGURE 4.13 – Affichage des entrées provenant de Redis après rafraîchissement

5. Monitoring avec Prometheus et Grafana

Pour observer l'état de santé des composants de notre infrastructure et identifier les éventuels goulots d'étranglement, un système de monitoring a été mis en place. Ce système repose sur deux outils largement utilisés dans les environnements Kubernetes :

- **Prometheus** : outil de collecte de métriques, capable d'interroger des endpoints HTTP exposant des données au format **OpenMetrics**.
- **Grafana** : outil de visualisation des données, qui se connecte à Prometheus pour créer des dashboards.

5.1 Choix de la stratégie de monitoring

L'objectif ici est de monitorer principalement le backend Node.js, en exposant des métriques sur l'endpoint `/metrics` grâce à la bibliothèque `prom-client`. Ce choix permet d'avoir une visibilité directe sur le comportement applicatif (nombre de requêtes, latence, etc.).

Il est aussi possible, en extension du projet, d'ajouter des *exporters* pour Redis et Kubernetes (via `kube-state-metrics`), mais cela n'était pas requis dans les objectifs principaux.

5.2 Configuration de Prometheus

5.2.1 Création des fichiers Prometheus

```
root@ecene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers-yaml# kubectl apply -f prometheus-cfg.yaml
kubectl apply -f prometheus-deployment.yaml
kubectl apply -f prometheus-service.yaml
configmap/prometheus-cfg created
deployment.apps/prometheus created
service/prometheus created
```

FIGURE 5.1 – Fichiers de configuration pour le déploiement de Prometheus

5.2.2 Fichier de configuration : prometheus-cfg.yaml

Le fichier configure Prometheus avec une `scrape_job` pour aller chercher les métriques sur le service backend.

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'node-backend'
    static_configs:
      - targets: ['backend-service:3000']
```

5.2.3 Déploiement de Prometheus

Déploiement des ressources sur le cluster :

```
kubectl apply -f prometheus-cfg.yaml
kubectl apply -f prometheus-deployment.yaml
kubectl apply -f prometheus-service.yaml
```

```
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers.yaml# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
backend-55879c5994-lnks8            1/1     Running   0           50m
frontend-7767996ccb-wr7sg          1/1     Running   0           21m
prometheus-789fbc6cdd-xfqjn        1/1     Running   0           102s
redis-esclave-7b448c7f7b-r9mwn     1/1     Running   1 (14h ago) 14h
redis-maitre-794c58957c-gv869     1/1     Running   1 (14h ago) 15h
```

FIGURE 5.2 – Vérification des pods Prometheus via kubectl

```
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers.yaml# minikube service prometheus --url
http://192.168.49.2:30666
```

FIGURE 5.3 – Accès à l'interface Prometheus via navigateur

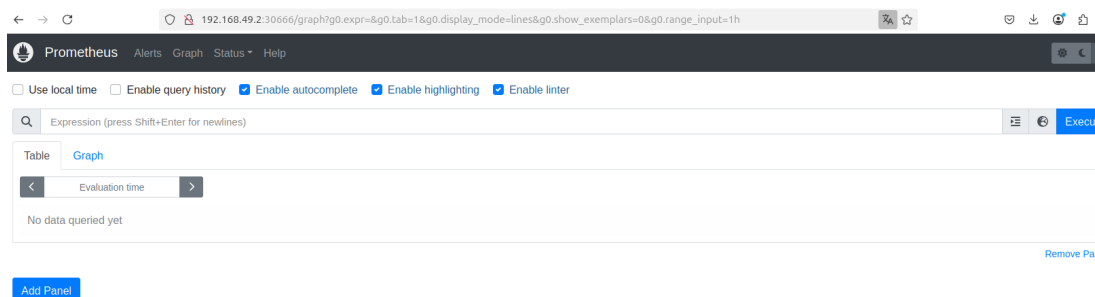


FIGURE 5.4 – Interface Prometheus accessible et fonctionnelle

5.3 Déploiement de Grafana

Grafana permet de créer des dashboards à partir des données collectées par Prometheus.

5.3.1 Déploiement sur le cluster

```
kubectl apply -f grafana-deployment.yaml
kubectl apply -f grafana-service.yaml
```

```
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers-yaml# kubectl apply -f grafana-deployment.yaml
kubectl apply -f grafana-service.yaml
deployment.apps/grafana created
service/grafana created
```

FIGURE 5.5 – Fichiers YAML pour le déploiement de Grafana

```
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers-yaml# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
backend-55879c5994-lnks8            1/1     Running   0           70m
frontend-7767996ccb-wr7sg           1/1     Running   0           41m
grafana-85b785d45d-8sbj6            1/1     Running   0           2m10s
prometheus-555886958d-zxmzm         1/1     Running   0           8m10s
redis-esclave-7b448c7f7b-r9mwn      1/1     Running   1 (14h ago)  14h
redis-maitre-794c58957c-gv869       1/1     Running   1 (14h ago)  15h
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers-yaml#
```

FIGURE 5.6 – Affichage du pod Grafana actif

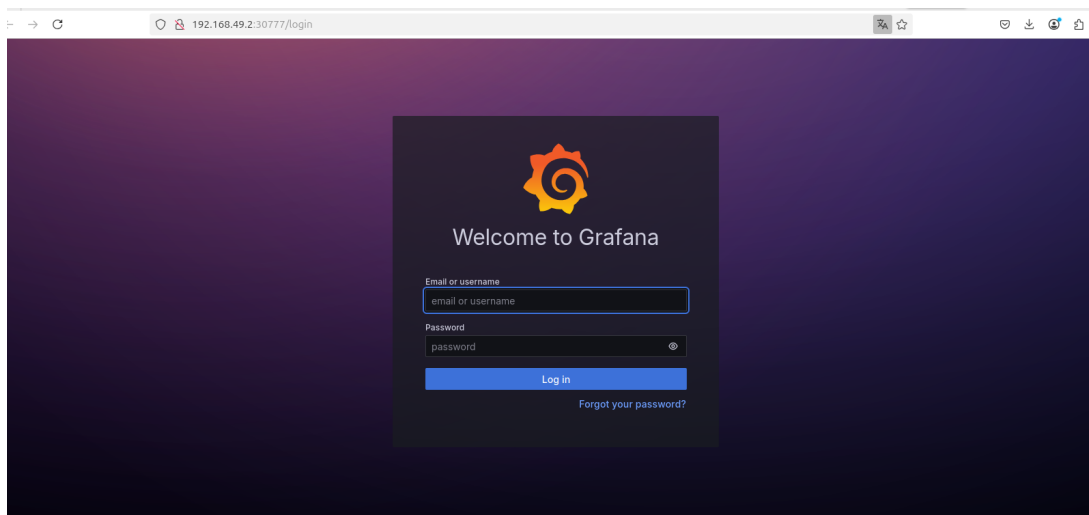


FIGURE 5.7 – Interface Grafana accessible et fonctionnelle

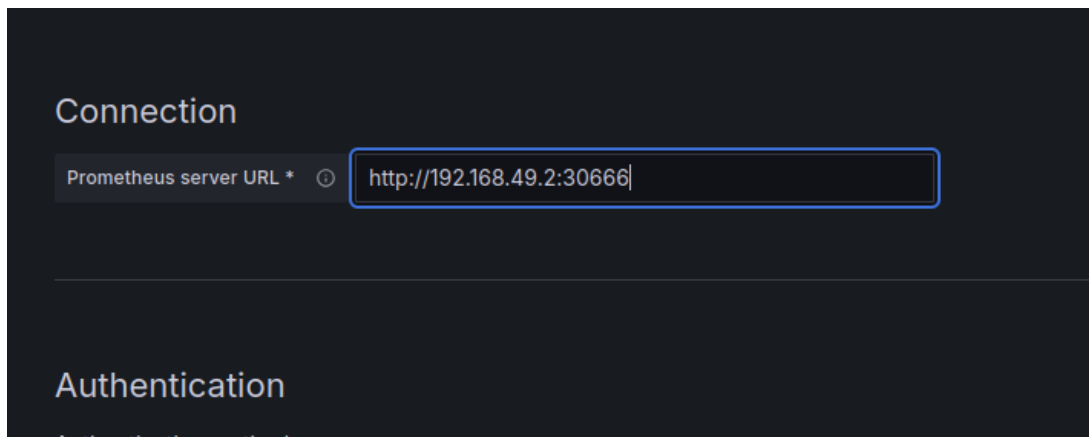
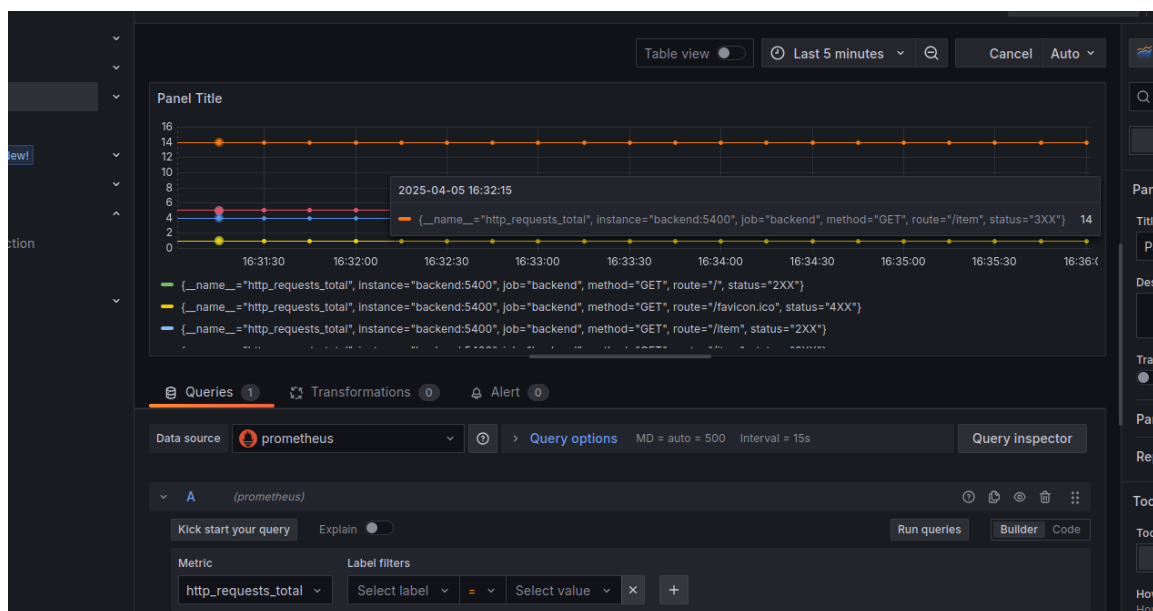


FIGURE 5.8 – Ajout de Prometheus comme source de données dans Grafana

5.3.2 Création d'un dashboard

Une fois connecté à Grafana (via le port-forward ou le NodePort), on peut :

1. Ajouter Prometheus comme datasource (URL : `http://prometheus-service:9090`)
2. Créer un dashboard avec des panels, par exemple :
 - Nombre de requêtes HTTP (compteur personnalisé)
 - Temps de réponse moyen
 - CPU et mémoire du pod backend (si métriques Kubernetes disponibles)

FIGURE 5.9 – Affichage des métriques `http_requests_total` dans un panel Grafana

6. Versionnement, Reproductibilité et Conclusion

6.1 Mise en place du dépôt GitHub

Afin d'assurer un bon suivi du projet et de faciliter sa collaboration ou sa reprise, l'ensemble des fichiers nécessaires au déploiement de l'infrastructure ont été versionnés dans un dépôt Git. Ce dépôt contient :

- Tous les fichiers `.yaml` de déploiement Kubernetes.
- Les Dockerfiles pour le backend et le frontend.
- Les scripts éventuels d'automatisation.
- Ce rapport au format \LaTeX .

Cependant, comme les dossiers `backend` et `frontend` ont été clonés depuis leurs dépôts GitHub respectifs (<https://github.com/arthurescriou/redis-node> et <https://github.com/arthurescriou/redis-node>), ils contenaient chacun un sous-répertoire `.git`, ce qui empêchait leur versionnement dans le dépôt principal.

Pour contourner ce problème, la méthode suivante a été utilisée en ligne de commande :

1. Renommer les dossiers clonés :

```
mv backend backend-old
mv frontend frontend-old
```

2. Créer de nouveaux dossiers vides :

```
mkdir backend
mkdir frontend
```

3. Copier le contenu des anciens dossiers (sans leur historique Git) vers les nouveaux :

```
cp -r backend-old/* backend/
cp -r frontend-old/* frontend/
```

4. Supprimer les anciens dossiers :

```
rm -rf backend-old frontend-old
```

Cette manipulation a permis de nettoyer les répertoires des métadonnées Git et de les intégrer proprement dans le dépôt principal.

6.1.1 Commandes utilisées

```
git init
git add .
git commit -m "Initial commit"
git remote add origin git@github.com:somehcene/IaC-and-
  Autoscaling-project.git
git push -u origin master
```

Le projet est disponible publiquement à l'adresse suivante : <https://github.com/somehcene/IaC-and-Autoscaling-project>

6.2 Reproductibilité du projet

L'approche Infrastructure as Code permet une **reproductibilité complète** du déploiement. En clonant le dépôt et en suivant les instructions du fichier `README.md`, toute personne peut :

- Reconstituer l'environnement via Docker et Minikube.
- Déployer l'infrastructure complète (Redis, backend, frontend).
- Obtenir le monitoring Prometheus/Grafana.
- Observer l'auto-scaling des composants répliquables.

Cette reproductibilité est essentielle dans un contexte DevOps ou cloud-native, car elle garantit que l'environnement de production, de test ou de développement peut être reconstruit de manière identique.

6.3 Bilan et conclusion

Ce projet m'a permis de consolider des compétences clés dans le domaine de l'infrastructure cloud-native, notamment :

- Le déploiement et la gestion d'une application sur Kubernetes.
- L'utilisation d'outils de conteneurisation comme Docker.
- L'implémentation de stratégies de montée en charge via les HPA (Horizontal Pod Autoscaler).
- La mise en place d'un monitoring avec Prometheus et Grafana.
- La structuration d'un projet reproductible avec Git et IaC.

Il est désormais possible d'envisager plusieurs pistes d'amélioration, telles que :

- Le passage à une base Redis hautement disponible (Sentinel ou Cluster).
- Le scaling plus intelligent du backend via des métriques personnalisées.
- L'automatisation complète du déploiement avec Helm ou ArgoCD.

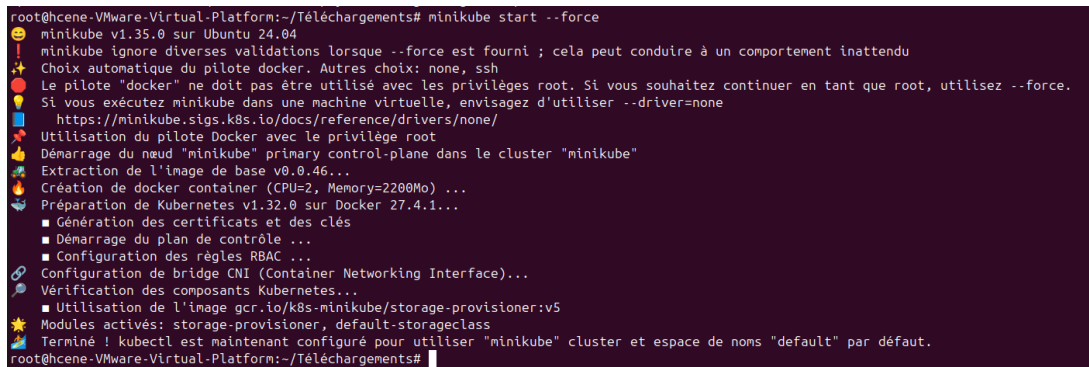
En somme, ce projet représente une mise en pratique complète des principes modernes de déploiement applicatif dans le cloud, en respectant les bonnes pratiques de scalabilité,

d'observabilité et d'automatisation.

7. Annexe : Détails supplémentaires et observations sur les captures

Cette annexe présente un complément visuel et descriptif du déroulé du projet à travers les captures d'écran issues du terminal et de l'interface utilisateur.

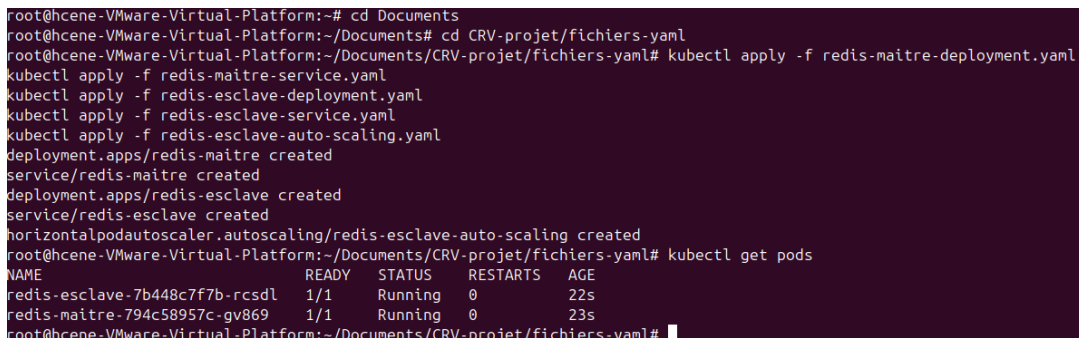
7.1 Initialisation de Minikube



```
root@hcene-VMware-Virtual-Platform:~/Téléchargements# minikube start --force
minikube v1.35.0 sur Ubuntu 24.04
! minikube ignore diverses validations lorsque --force est fourni ; cela peut conduire à un comportement inattendu
* Choix automatique du pilote docker. Autres choix: none, ssh
* Le pilote "docker" ne doit pas être utilisé avec les privilèges root. Si vous souhaitez continuer en tant que root, utilisez --force.
* Si vous exécutez minikube dans une machine virtuelle, envisagez d'utiliser --driver=none
  https://minikube.sigs.k8s.io/docs/reference/drivers/none/
* Utilisation du pilote Docker avec le privilège root
* Démarrage du nœud "minikube" primary control-plane dans le cluster "minikube"
* Extraction de l'image de base v0.0.46...
* Création de docker container (CPU=2, Memory=2200Mo) ...
* Préparation de Kubernetes v1.32.0 sur Docker 27.4.1...
  ■ Génération des certificats et des clés
  ■ Démarrage du plan de contrôle ...
  ■ Configuration des règles RBAC ...
* Configuration de bridge CNI (Container Networking Interface)...
* Vérification des composants Kubernetes...
  ■ Utilisation de l'image gcr.io/k8s-minikube/storage-provisioner:v5
* Modules activés: storage-provisioner, default-storageclass
* Terminé ! kubectl est maintenant configuré pour utiliser "minikube" cluster et espace de noms "default" par défaut.
root@hcene-VMware-Virtual-Platform:~/Téléchargements#
```

FIGURE 7.1 – Démarrage du cluster avec Minikube

7.2 Déploiement de Redis



```
root@hcene-VMware-Virtual-Platform:~# cd Documents
root@hcene-VMware-Virtual-Platform:~/Documents# cd CRV-projet/fichiers-yaml
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers-yaml# kubectl apply -f redis-maitre-deployment.yaml
kubectl apply -f redis-maitre-service.yaml
kubectl apply -f redis-esclave-deployment.yaml
kubectl apply -f redis-esclave-service.yaml
kubectl apply -f redis-esclave-auto-scaling.yaml
deployment.apps/redis-maitre created
service/redis-maitre created
deployment.apps/redis-esclave created
service/redis-esclave created
horizontalpodautoscaler.autoscaling/redis-esclave-auto-scaling created
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers-yaml# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
redis-esclave-7b448c7f7b-rcsdl      1/1     Running   0           22s
redis-maitre-794c58957c-gv869      1/1     Running   0           23s
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers-yaml#
```

FIGURE 7.2 – Redis master et répliqués correctement déployés

7.3 Création et déploiement du backend Node.js

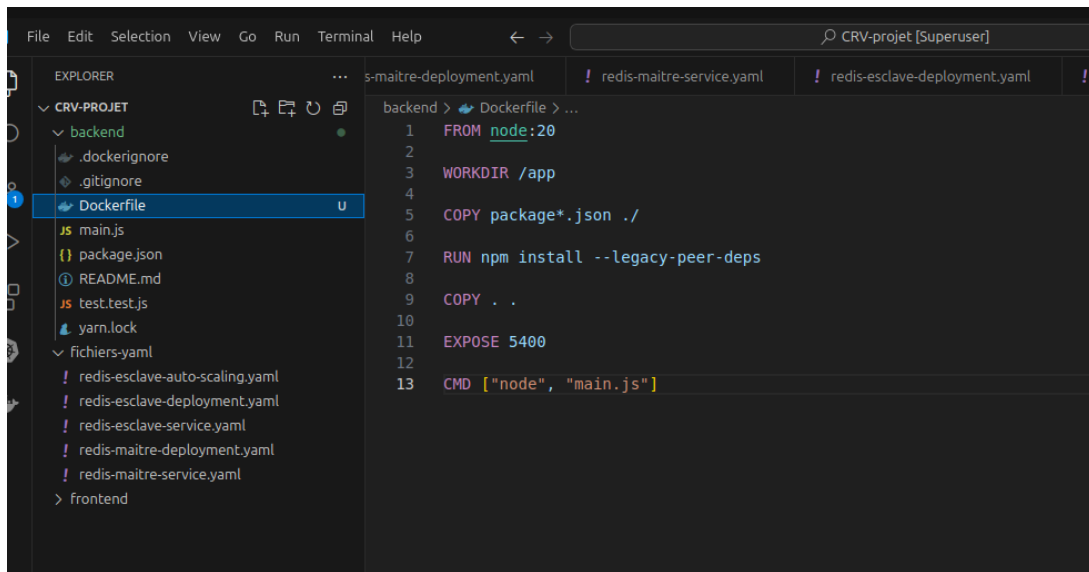


FIGURE 7.3 – Contenu du Dockerfile pour le backend

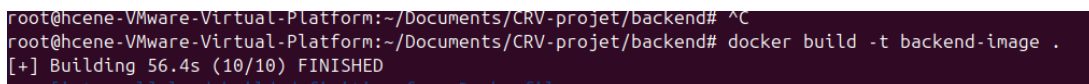


FIGURE 7.4 – Construction de l'image Docker du backend

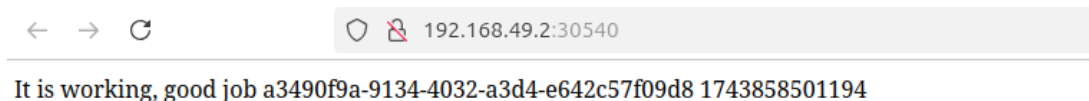


FIGURE 7.5 – Backend Node.js opérationnel

7.4 Publication d'une image sur Docker Hub

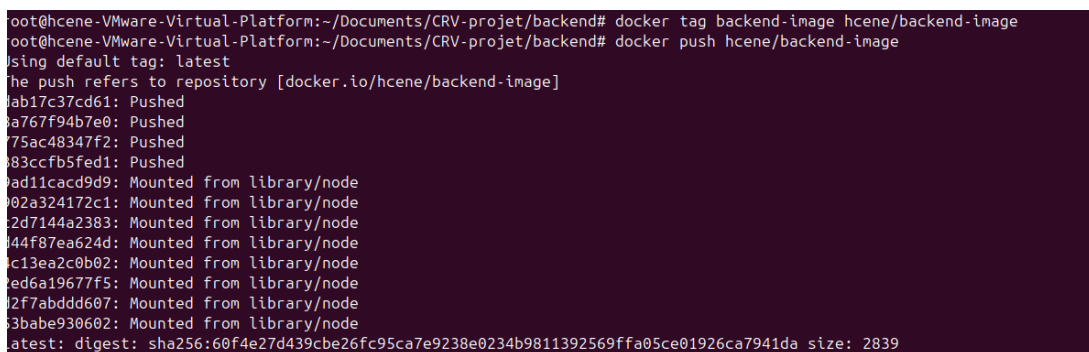


FIGURE 7.6 – Poussée de l'image Docker vers Docker Hub

7.5 Déploiement du frontend React

```
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/fichiers-yaml# cd ../frontend
root@hcene-VMware-Virtual-Platform:~/Documents/CRV-projet/frontend# docker build -t frontend-image .
docker tag frontend-image hcene/frontend-image
docker push hcene/frontend-image
[+] Building 113.3s (17/17) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 308B
```

FIGURE 7.7 – Création et publication de l'image React

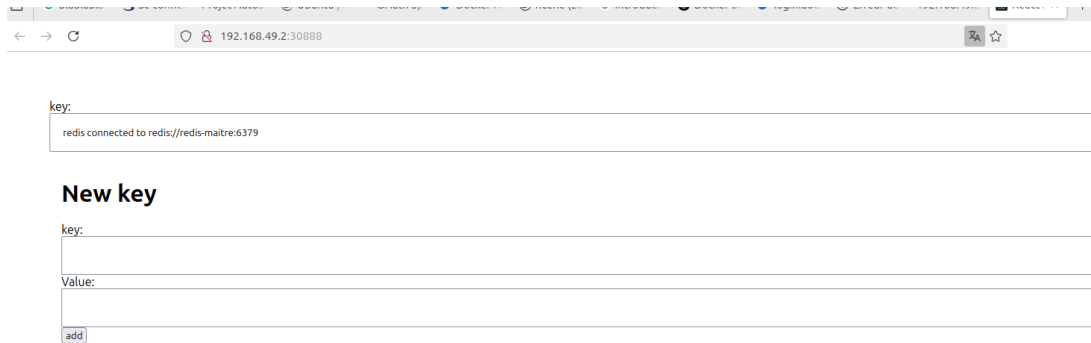


FIGURE 7.8 – Interface React accessible via service Kubernetes

7.6 Monitoring avec Prometheus et Grafana

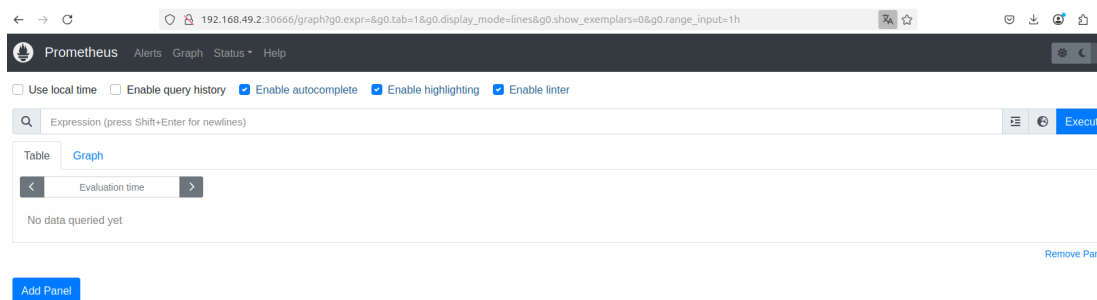


FIGURE 7.9 – Interface Prometheus fonctionnelle

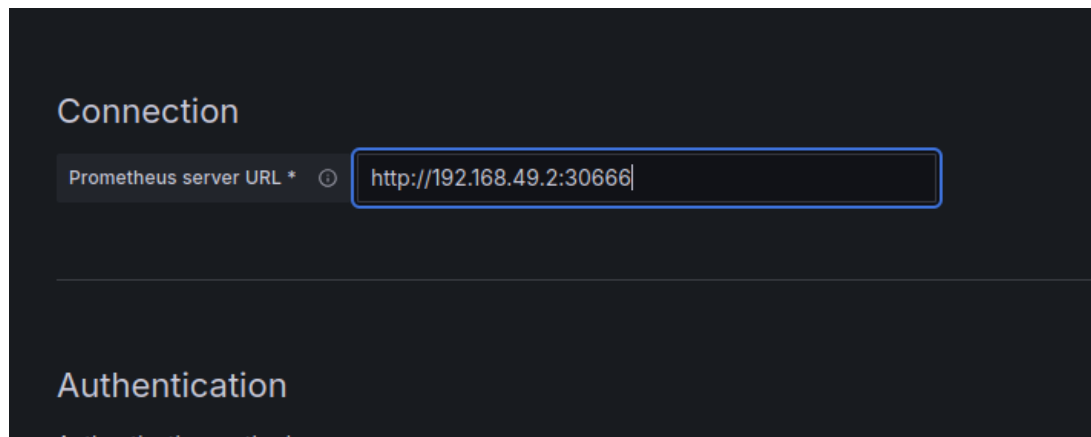


FIGURE 7.10 – Ajout de Prometheus dans Grafana

Ce complément graphique permet d'illustrer plus fidèlement l'exécution des différentes étapes du projet et de valider le bon fonctionnement de l'ensemble de l'architecture.