



Sorbonne Université Faculté des Sciences et Ingénierie

Master 1 informatique parcours Réseaux

## **Projet AutoScaling et IaC**

Module : Cloud et réseaux virtuels

**Auteur :**

SOAL Achouak Sabrina 21310586

Ce rapport décrit l'architecture et les décisions prises pour la mise à l'échelle d'une application web utilisant Redis, Node.js et React. L'application est composée de différentes parties :

**Base de données Redis** : une base de données Redis principale avec des répliques pour la lecture.

**Serveur Node.js** : un serveur stateless qui utilise Redis pour stocker et récupérer des données.

**Client React** : un client web qui utilise le serveur Node.js pour communiquer avec Redis.

**Prometheus/Grafana** : Un ensemble d'outils de monitoring pour surveiller les performances de l'application.

## I. Redis :

On a choisi d'implémenter un modèle Main/Replicas où une base de données Redis principale accepte toutes les opérations tandis que les replicas recopient les données de cette base pour permettre une lecture en parallèle.

- **Installation et configuration de Redis avec un pattern main/replicas (2 replicas)**

### 1- Fichiers à créer :

**redis-deployment.yaml** (déploiement de la base principale)

**redis-deployment-replica.yaml** (déploiement des replicas)

**redis-service.yaml** (service pour la base principale)

**redis-replica-service.yaml** (service pour les replicas)

**redis-deployment.yaml** et **redis-replica-deployment-replica.yaml** : Ces fichiers définissent les déploiements pour les pods Redis maître et replica. Ils lancent les instances Redis et les gèrent en cas de panne.

**redis-service.yaml** et **redis-replica-service.yaml** : Ces fichiers définissent les services pour les pods Redis. Ils exposent les pods à l'extérieur du cluster Kubernetes et permettent aux applications de s'y connecter.

### 2- Étapes :

#### 2.1- Déploiement :

- Création du déploiement Redis Main :

On crée un fichier nommé **redis-deployment.yaml** :

Avec la commande : **nano redis-deployment.yaml**

Ce fichier de configuration crée un déploiement Redis principal avec une seule réplique.

```
GNU nano 7.2 redis-deploiement.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-main
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis-main
  template:
    metadata:
      labels:
        app: redis-main
    spec:
      containers:
        - name: redis
          image: redis:latest
          ports:
            - containerPort: 6379
          command: ["redis-server", "--appendonly", "yes"]
```

- Création du déploiement Redis Replica :

On crée un fichier nommé redis-deployment-replica.yaml :

Ce fichier de configuration crée un déploiement Redis replica avec deux répliques.

```
GNU nano 7.2 redis-deployment-replica.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-replica1
  labels:
    app: redis-replica1
spec:
  replicas: 2
  selector:
    matchLabels:
      app: redis-replica1
  template:
    metadata:
      labels:
        app: redis-replica1
    spec:
      containers:
        - name: redis
          image: redis:latest
          ports:
            - containerPort: 6379
          command: ["redis-server", "--replicaof", "redis-main", "6379"]
```

### 2.2- Service :

- Création du service Redis Main :

On crée un fichier nommé redis-service.yaml

Ce fichier de configuration crée un service Redis principal avec une seule réplique.

```
GNU nano 7.2 redis-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: redis-main
spec:
  selector:
    app: redis-main
  ports:
    - protocol: TCP
      port: 6379
      targetPort: 6379
  type: ClusterIP
```

- Création du service Redis Replica :

On crée un fichier nommé redis-replica-service.yaml

Ce fichier de configuration crée un service Redis replica avec deux répliques.

```
GNU nano 7.2 redis-replica-service.yaml *
apiVersion: v1
kind: Service
metadata:
  name: redis-replicat1
spec:
  selector:
    app: redis-replicat1
  ports:
    - protocol: TCP
      port: 6379
      targetPort: 6379
  type: ClusterIP
```

### 3- Appliquer les configurations :

On utilise la commande `kubectl apply -f <nom_du_fichier.yaml>` pour déployer les ressources Kubernetes à partir des fichiers YAML créés.

```
achouak@achouak:~/projet/back-end/redis-node-master$ kubectl apply -f redis-deployment.yaml
deployment.apps/redis-main unchanged
achouak@achouak:~/projet/back-end/redis-node-master$ kubectl apply -f redis-deployment-replica.yaml
deployment.apps/redis-replicat1 unchanged
achouak@achouak:~/projet/back-end/redis-node-master$ kubectl apply -f redis-service.yaml
service/redis-main unchanged
achouak@achouak:~/projet/back-end/redis-node-master$ kubectl apply -f redis-replica-service.yaml
service/redis-replicat1 unchanged
achouak@achouak:~/projet/back-end/redis-node-master$
```

### 4- Vérifier l'installation :

- La commande `kubectl get pods` montre que tous les pods sont en cours d'exécution et prêts.
- La commande `kubectl get service` montre que tous les services sont disponibles.
- La commande `kubectl get deployment` montre que tous les déploiements sont à jour et ont le nombre de répliques souhaité.

```
pod/redis-main-55cd4489f4-zzhfj          1/1      Running   0          53m
pod/redis-replicat1-68f95fbc6d-52rpk     1/1      Running   0          52m
pod/redis-replicat1-68f95fbc6d-78q8x     1/1      Running   0          52m
```

```
service/redis-main          ClusterIP   10.111.25.70   <none>   6379/TCP   52m
service/redis-replicat1     ClusterIP   10.105.118.210 <none>   6379/TCP   52m
```

```
deployment.apps/redis-main          1/1      1          1          53m
deployment.apps/redis-replicat1     2/2      2          2          52m
```

### 5- Autoscalling :

- Création du fichier Redis hpa :

On crée un fichier nommé `redis-replica-hpa.yaml`

Le fichier `redis-replica-hpa.yaml` permet de configurer l'autoscaling horizontal pour les pods Redis replica.

```

GNU nano 7.2 redis-replica-hpa.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: redis-replica-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: redis-replicat1
  minReplicas: 1
  maxReplicas: 2
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 80

```

- Appliquer le fichier :

```

achouak@achouak:~/projet/back-end/redis-node-master$ kubectl apply -f redis-repl
ica-hpa.yaml
horizontalpodautoscaler.autoscaling/redis-replica-hpa unchanged
achouak@achouak:~/projet/back-end/redis-node-master$

```

- Vérification :

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
horizontalpodautoscaler.autoscaling/backend-hpa	Deployment/backend-deployment	<unknown>/80%	1	2	2	39m
horizontalpodautoscaler.autoscaling/php-apache	Deployment/php-apache	0%/50%	1	10	1	4d23h
horizontalpodautoscaler.autoscaling/redis-replica-hpa	Deployment/redis-replicat1	<unknown>/80%	1	2	2	4d23h

Cela indique que l'HPA est actif et surveille les répliques du déploiement "redis-replicat1" pour s'assurer que l'utilisation de la ressource reste à un niveau acceptable, et il prendra des mesures pour scaler le nombre de répliques en conséquence si nécessaire.

## 6- Tester la configuration :

Pour tester la configuration avec Redis CLI, on peut suivre ces étapes une fois que le déploiement et service Redis sont opérationnels :

- Redis CLI doit être installé sur la machine.
- On utilise Redis CLI pour se connecter au service Redis.
- Une fois connecté, on peut utiliser les commandes Redis normales pour interagir avec la base de données Redis. Par exemple, on peut tester en définissant une clé et en récupérant sa valeur :

```

achouak@achouak:~/projet/back-end/redis-node-master$ redis-cli
127.0.0.1:6379> ping redis-replicat1
"redis-replicat1"
127.0.0.1:6379> SET mykey myvalue
OK
127.0.0.1:6379> GET mykey
"myvalue"
127.0.0.1:6379> KEYS *
1) "mykey"
127.0.0.1:6379>

```

On peut définir et récupérer des valeurs avec succès, cela signifie que la configuration Redis avec Kubernetes fonctionne correctement.

## II. NodeJs :

On a décidé d'utiliser un serveur Node.js stateless, ce qui signifie qu'il ne conserve pas d'état persistant. Cela permet de dupliquer facilement le serveur sans modifier le comportement de

l'application, ce qui facilite la montée en échelle. Le serveur Node.js communique avec la base Redis principale ainsi qu'avec ses replicas pour effectuer des opérations de lecture. Les replicas sont utilisés pour permettre une lecture en parallèle et une montée en charge.

- **Installation et configuration de NodeJs**

## 1- Fichiers à créer :

### Dockerfile

**backend-deployment.yaml** (déploiement de le backend)

**backend-service.yaml** (service pour le backend)

**Dockerfile** : Le fichier Dockerfile est utilisé pour construire une image Docker de l'application backend Node.js.

**backend-deployment.yaml** : Ce fichier YAML est utilisé pour déployer l'application backend Node.js dans un cluster Kubernetes

**backend-service.yaml** : Ce fichier YAML est utilisé pour créer un service Kubernetes pour l'application backend Node.js.

## 2- Etapes :

### 2.1- Création du fichier Dockerfile :



```
GNU nano 7.2 dockerfile
FROM node:latest

COPY . .

RUN npm install -g npm@latest

RUN npm cache clean --force

RUN npm install --force

CMD ["node", "main.mjs"]
```

- Construire l'image Docker :

On exécute la commande suivante pour construire l'image Docker :

**docker build -t soalachouak/node-redis .**

```
achouak@achouak:~/projet/back-end/redis-node-master$ docker build -t soalachouak/node-redis .
Sending build context to Docker daemon 154.6kB
Step 1/6 : FROM node:latest
--> c3978d05bc68
Step 2/6 : COPY . .
--> 2fa54f16a325
Step 3/6 : RUN npm install -g npm@latest
--> Running in f30d2609e5fe

changed 37 packages in 1m

25 packages are looking for funding
  run `npm fund` for details
Removing intermediate container f30d2609e5fe
--> 85cab79f899e
Step 4/6 : RUN npm cache clean --force
--> Running in 83b509310de1
npm WARN using --force Recommended protections disabled.
Removing intermediate container 83b509310de1
--> 17d3f8c2f29e
Step 5/6 : RUN npm install --force
--> Running in d7a48ced67f4
```

```
added 377 packages, and audited 378 packages in 9m

40 packages are looking for funding
  run `npm fund` for details

1 moderate severity vulnerability

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
Removing intermediate container d7a48ced67f4
--> 4f9a61c3ef2a
Step 6/6 : CMD ["node", "main.mjs"]
--> Running in d2b8b9276e3e
Removing intermediate container d2b8b9276e3e
--> f3abe68d25ed
Successfully built f3abe68d25ed
Successfully tagged soalachouak/node-redis:latest
```


- Pousser l'image Docker vers un registre :

Pour partager l'image Docker, on doit la pousser vers un registre. On doit se connecter à un registre (docker hub) et on exécute la commande suivante :


**docker push soalachouak/node-redis:latest**

```
achouak@achouak:~/projet/back-end/redis-node-master$ docker push soalachouak/node-redis:latest
The push refers to repository [docker.io/soalachouak/node-redis]
6cf1991f7121: Pushed
77bbd6fc0b1e: Pushed
571dd29386ee: Pushed
290c91af576d: Pushed
3053dc82b7d3: Mounted from soalachouak/redis-react
9b7a370e94a7: Mounted from soalachouak/redis-react
5996a891ea4c: Mounted from soalachouak/redis-react
5358370f44ab: Mounted from soalachouak/redis-react
21e1c4948146: Mounted from soalachouak/redis-react
68866beb2ed2: Mounted from soalachouak/redis-react
e6e2ab10dba6: Mounted from soalachouak/redis-react
0238a1790324: Mounted from soalachouak/redis-react
latest: digest: sha256:3abb481b295bd9f4b909abb2ca2bed496a19476e5c3005f704ae4aa27
9e3c2ab size: 2843
achouak@achouak:~/projet/back-end/redis-node-master$ kubect1 get all
```

La capture d'écran de la page Web montre les informations sur le dépôt Docker sur Docker Hub. Le dépôt s'appelle soalachouak/redis-node.



soalachouak/node-redis 

Updated about 16 hours ago

This repository does not have a description 

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
 latest		Image	an hour ago	16 hours ago

[See all](#)

## 2-3 Déploiement :

- Création du déploiement NodeJs :

On crée un fichier nommé backend-deployment.yaml:

On remplace soalachouak/redis-node:latest par le nom et la version de l'image Docker.

```
GNU nano 7.2 backend-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend
          imagePullPolicy: Always
          image: soalachouak/node-redis:latest
          ports:
            - containerPort: 8080 #Port exposé par l'application Node.js
          env:
            - name: PORT
              value: '8080'
            - name: REDIS_URL
              value: redis://redis-main.default.svc.cluster.local:6379
            - name: REDIS_REPLICAS_URL
              value: redis://redis-replica1.default.svc.cluster.local:6379
```

## 2-4 Service :

- Création du service NodeJs :

On crée un fichier nommé backend-service.yaml:



```
GNU nano 7.2 backend-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app: backend
  ports:
  - protocol: TCP
    port: 8080 # Port externe pour accéder au service
    targetPort: 8080 # Port interne de l'application Node.js
  type: LoadBalancer
```

### 3- Déployer l'application sur Kubernetes :

On exécute la commande suivante pour déployer l'application sur Kubernetes :

```
kubectl apply -f backend-deployment.yaml
```

```
kubectl apply -f backend-service.yaml
```

```
achouak@achouak:~/projet/back-end/redis-node-master$ kubectl apply -f backend-deployment.yaml
deployment.apps/backend-deployment unchanged
achouak@achouak:~/projet/back-end/redis-node-master$ kubectl apply -f backend-service.yaml
service/backend-service unchanged
achouak@achouak:~/projet/back-end/redis-node-master$
```

### 4- Vérifier l'installation :

- La commande `kubectl get pods` montre que tous les pods sont en cours d'exécution et **prêts (Ready)**.
- La commande `kubectl get service` montre que tous les services sont **disponibles**.
- La commande `kubectl get deployment` montre que tous les déploiements sont à jour et ont le nombre de **répliques souhaité**.

NAME	READY	STATUS	RESTARTS	AGE
pod/backend-deployment-6d68bbb7c6-ggxfp	1/1	Running	0	51m
pod/backend-deployment-6d68bbb7c6-xrlr9	1/1	Running	0	51m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/backend-service	LoadBalancer	10.102.108.157	<pending>	8080:32452/TCP	14h

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/backend-deployment	2/2	2	2	51m

### 5- Autoscalling :

- Création du fichier backend hpa :

On crée un fichier nommé backend-hpa.yaml

Le fichier backend-hpa.yaml permet de configurer l'autoscaling horizontal pour les pods backend replica.

```
GNU nano 7.2 backend-hpa.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: backend-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: backend-deployment
  minReplicas: 1
  maxReplicas: 2
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 80
```

- Appliquer le fichier :

```
achouak@achouak:~/projet/back-end/redis-node-master$ kubectl apply -f backend-hpa.yaml
horizontalpodautoscaler.autoscaling/backend-hpa unchanged
achouak@achouak:~/projet/back-end/redis-node-master$
```

- Vérification :

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
horizontalpodautoscaler.autoscaling/backend-hpa	Deployment/backend-deployment	<unknown>/80%	1	2	2	39m
horizontalpodautoscaler.autoscaling/php-apache	Deployment/php-apache	0%/50%	1	10	1	4d23h
horizontalpodautoscaler.autoscaling/redis-replica-hpa	Deployment/redis-replicat	<unknown>/80%	1	2	2	4d23h

Cela indique que l'HPA est actif et surveille les répliques du déploiement "backend-replica" pour s'assurer que l'utilisation de la ressource reste à un niveau acceptable, et il prendra des mesures pour scaler le nombre de répliques en conséquence si nécessaire.

## 6- Accéder à l'application :

On doit obtenir l'adresse IP du cluster Minikube pour accéder à l'application, on utilise la commande : **minikube ip**

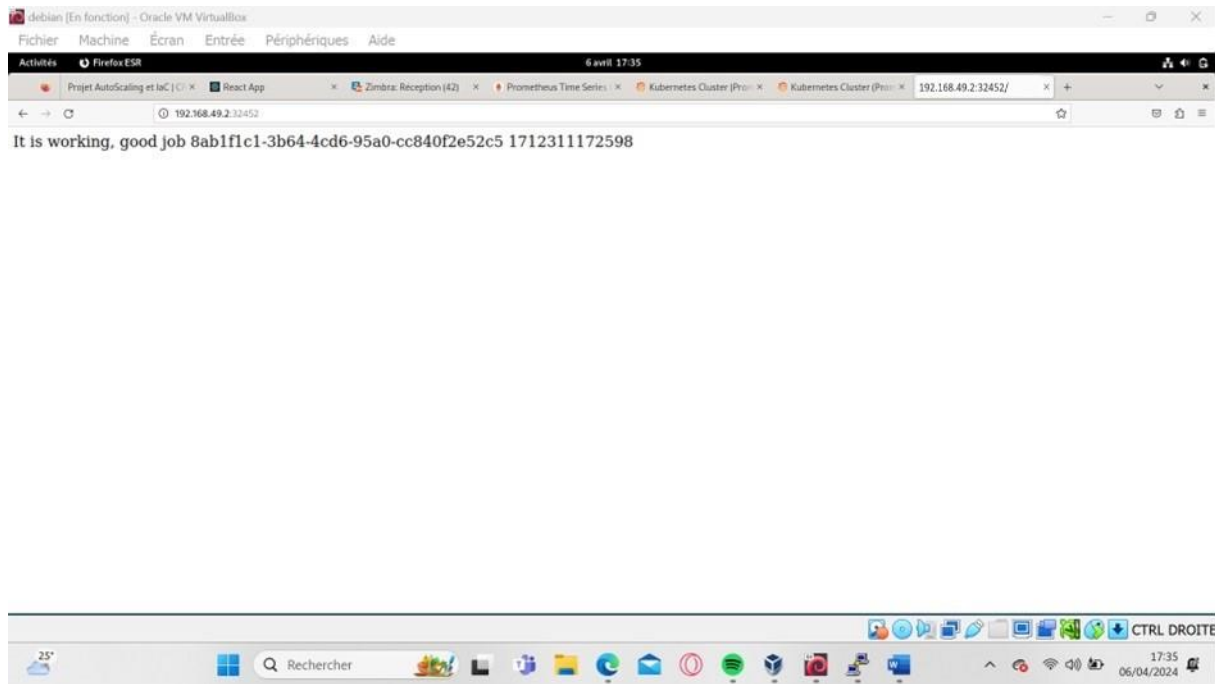
```
achouak@achouak:~/projet/front-end/redis-react-master$ minikube ip
192.168.49.2
```

Pour accéder à l'interface web de Nodejs, on doit connaître le numéro de port assigné par le service backend-service-ext. On peut l'obtenir en exécutant la commande suivante :

**minikube service backend-service-ext --url**

```
achouak@achouak:~/projet/back-end/redis-node-master$ minikube service backend-service --url
http://192.168.49.2:32452
```

L'application est accessible sur le port 32452 de le cluster Kubernetes. On peut utiliser un navigateur web pour accéder à l'application.



### III. React :

On a choisi d'utiliser le framework React pour le développement du frontend de l'application. Le projet frontend React communiquera avec le serveur Node.js pour récupérer les données nécessaires à son fonctionnement.

- **Installation et configuration de React**

#### 1- Fichiers à créer :

**Dockerfile**

**Frontend-node-deployment.yaml** (déploiement de le frontend)

**Frontend-node -service.yaml** (service pour le frontend)

**Dockerfile** : Le fichier Dockerfile est utilisé pour construire une image Docker de l'application frontend React.

**Frontend-node-deployment.yaml**: Ce fichier YAML est utilisé pour déployer l'application frontend React dans un cluster Kubernetes

**Frontend-node -service.yaml**: Ce fichier YAML est utilisé pour créer un service Kubernetes pour l'application frontend React.

#### 2- Etapes :

##### 2.1- Création du fichier Dockerfile :

```
GNU nano 7.2                                     dockerfile
FROM node:latest

COPY package.json ./

RUN npm install -g npm@latest

RUN npm cache clean --force

RUN npm install --force

COPY . .

EXPOSE 3000

CMD ["npm", "start"]
```

- Construire l'image Docker :

On exécute la commande suivante pour construire l'image Docker :

```
docker build -t soalachouak/redis-react .
```

```
soalachouak@soalachouak:~/projet/front-end/redis-react-master$ docker build -t soalachouak/redis-react .
Sending build context to Docker daemon 458.2kB
Step 1/8 : FROM node:latest
--> c3978d05bc68
Step 2/8 : COPY package.json ./
--> Using cache
--> da783a479e2d
Step 3/8 : RUN npm install -g npm@latest
--> Using cache
--> 4b7c9dc5eda9
Step 4/8 : RUN npm cache clean --force
--> Using cache
--> d70b5eef636f
Step 5/8 : RUN npm install --force
--> Using cache
--> 01bb90de3467
Step 6/8 : COPY . .
--> b4dc3ceb3698
Step 7/8 : EXPOSE 3000
--> Running in d74f0223cfe8
Removing intermediate container d74f0223cfe8
--> dca18cdcafec
Step 8/8 : CMD ["npm", "start"]
--> Running in 4f512cb61fea
Removing intermediate container 4f512cb61fea
--> 95a564ccc606
Successfully built 95a564ccc606
Successfully tagged soalachouak/redis-react:latest
```


- Pousser l'image Docker vers un registre :

Pour partager l'image Docker, on doit la pousser vers un registre. On doit se connecter à un registre (docker hub) et on exécute la commande suivante :


```
docker push soalachouak/redis-react :latest
```

```
achouak@achouak:~/projet/front-end/redis-react-master$ docker push soalachouak/redis-react:latest
The push refers to repository [docker.io/soalachouak/redis-react]
710delb859b8: Pushed
0fb790ce91ca: Layer already exists
d7f793c8deal: Layer already exists
c42a83107b63: Layer already exists
a9affd57a3e6: Layer already exists
3053dc82b7d3: Layer already exists
9b7a370e94a7: Layer already exists
5996a891ea4c: Layer already exists
5358370f44ab: Layer already exists
21e1c4948146: Layer already exists
68866beb2ed2: Layer already exists
e6e2ab10dba6: Layer already exists
0238a1790324: Layer already exists
latest: digest: sha256:ff6a6301e1321ca5a3490e79a2af17c5b740a1b3739800cf1aaa220f342bfe8a size: 3052
achouak@achouak:~/projet/front-end/redis-react-master$
```

La capture d'écran de la page Web montre les informations sur le dépôt Docker sur Docker Hub. Le dépôt s'appelle soalachouak/redis-react.



**soalachouak/redis-react** 

Updated less than a minute ago

This repository does not have a description 

**Tags**

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
 latest		Image	---	a few seconds ago

[See all](#)

### 2-3 Déploiement :

- Création du déploiement React :

Créez un fichier nommé frontend-node-deployment.yaml:

Remplacez soalachouak/redis-react:latest par le nom et la version de votre image Docker.

```
GNU nano 7.2 frontend-node-deployment.yaml *
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-node-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend-node
  template:
    metadata:
      labels:
        app: frontend-node
    spec:
      containers:
        - name: frontend-node
          image: soalachouak/redis-react:latest
          ports:
            - containerPort: 3000 #Port exposé par votre application Node.js
```

## 2-4 Déploiement :

- Création du service React :

Créez un fichier nommé frontend-node-service.yaml:

```
GNU nano 7.2 frontend-node-service.yaml *
apiVersion: v1
kind: Service
metadata:
  name: frontend-node-service
spec:
  selector:
    app: frontend-node
  ports:
  - protocol: TCP
    port: 8080 # Port externe pour accéder au service
    targetPort: 3000 # Port interne de l'application Node.js
  type: LoadBalancer
```

## 3- Déployer l'application sur Kubernetes :

On exécute la commande suivante pour déployer l'application sur Kubernetes :

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

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

```
achouak@achouak:~/projet/front-end/redis-react-master$ kubectl apply -f frontend-node-deployment.yaml
deployment.apps/frontend-node-deployment unchanged
achouak@achouak:~/projet/front-end/redis-react-master$ kubectl apply -f frontend-node-service.yaml
service/frontend-node-service configured
```

## 4- Vérifier l'installation :

- La commande `kubectl get pods` montre que tous les pods sont en cours d'exécution et **prêts**.
- La commande `kubectl get service` montre que tous les services sont **disponibles**.
- La commande `kubectl get deployment` montre que tous les déploiements sont à jour et ont le nombre de **répliques souhaité**.

NAME	READY	STATUS	RESTARTS	AGE
pod/backend-deployment-6d68bbb7c6-ggxfp	1/1	Running	0	51m
pod/backend-deployment-6d68bbb7c6-xrlr9	1/1	Running	0	51m
pod/frontend-node-deployment-548f8fc49-j22kd	1/1	Running	15 (84m ago)	4d18h

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/backend-service	LoadBalancer	10.102.108.157	<pending>	8080:32452/TCP	14h
service/frontend-node-service	LoadBalancer	10.110.96.131	<pending>	8080:30526/TCP	4d18h

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/backend-deployment	2/2	2	2	51m
deployment.apps/frontend-node-deployment	1/1	1	1	4d18h

## 7- Accéder à l'application :

On doit obtenir l'adresse IP du cluster Minikube pour accéder à l'application, on utilise la commande : `minikube ip`

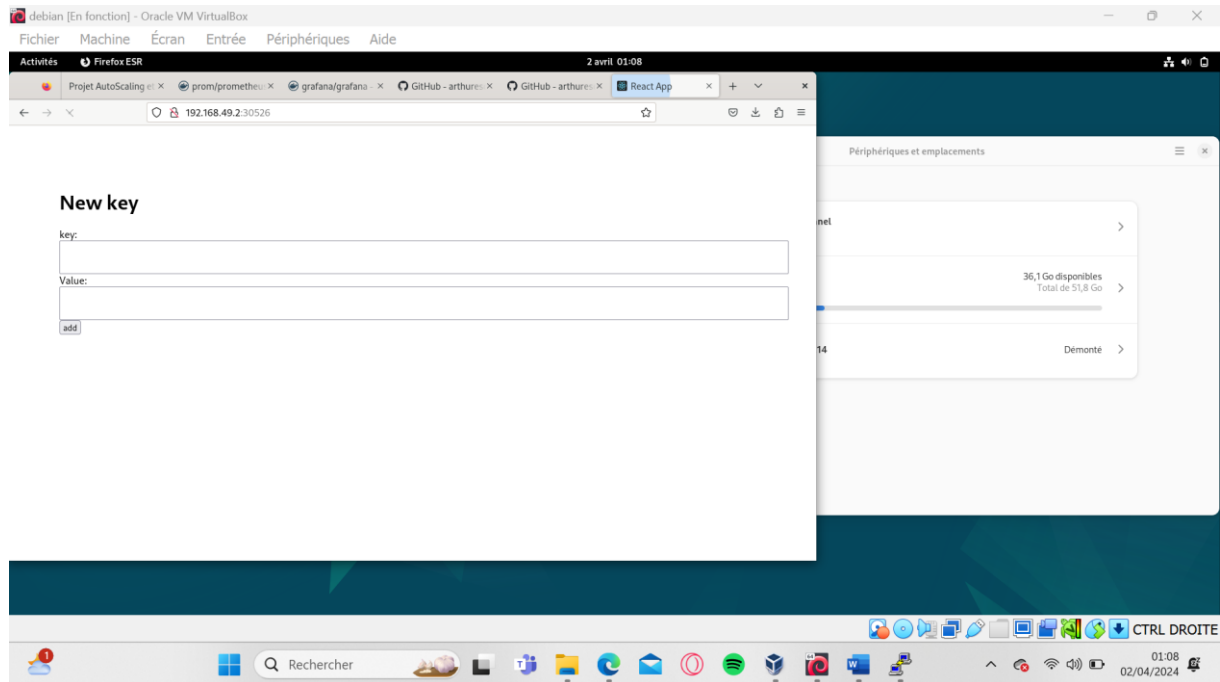
```
achouak@achouak:~/projet/front-end/redis-react-master$ minikube ip
192.168.49.2
```

Pour accéder à l'interface web de React, on doit connaître le numéro de port assigné par le service frontend-node-service-ext. On peut l'obtenir en exécutant la commande suivante :

```
minikube service frontend-node-service-ext --url
```

```
achouak@achouak:~/projet/back-end/redis-node-master$ minikube service frontend-node-service --url  
http://192.168.49.2:30526
```

L'application est accessible sur le port 30526 de le cluster Kubernetes. On peut utiliser un navigateur web pour accéder à l'application.



## IV. Prometheus/grafana :

Déployer Prometheus et Grafana pour surveiller une application Node.js exposant des métriques via l'API /metrics.

- **Installation et configuration de Prometheus/grafana**

### 1- Étapes :

#### 1-1 Installation prometheus avec helm :

On va installer Prometheus en utilisant Helm, un gestionnaire de paquets pour Kubernetes.

- Ajout du dépôt Helm :

Tout d'abord, on ajoute le dépôt de la communauté Prometheus qui contient la chart Prometheus :

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

```
achouak@achouak:~/projet$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts  
"prometheus-community" already exists with the same configuration, skipping  
achouak@achouak:~/projet$
```

- Mise à jour du dépôt :

Ensuite, on met à jour la liste des charts disponibles dans le dépôt :

## helm repo update

```
achouak@achouak:~/projet/prometheus$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "stable" chart repository
Update Complete. ☐Happy Helming!☐
```

- Installation de Prometheus :

Maintenant, on installe Prometheus en utilisant la chart prometheus du dépôt prometheus-community :

## helm install prometheus prometheus-community/prometheus

Cette commande va déployer Prometheus et les composants associés sur le cluster Kubernetes.

```
achouak@achouak:~/projet/prometheus$ helm install prometheus prometheus-community/prometheus
NAME: prometheus
LAST DEPLOYED: Thu Apr 4 04:22:41 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Prometheus server can be accessed via port 80 on the following DNS name from within your cluster:
prometheus-server.default.svc.cluster.local

Get the Prometheus server URL by running these commands in the same shell:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=prometheus,app.kubernetes.io/instance=prometheus" -o jsonpath="{.items[0].metadata.name}")
  kubectl --namespace default port-forward $POD_NAME 9090

The Prometheus alertmanager can be accessed via port 9093 on the following DNS name from within your cluster:
prometheus-alertmanager.default.svc.cluster.local

Get the Alertmanager URL by running these commands in the same shell:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=alertmanager,app.kubernetes.io/instance=prometheus" -o jsonpath="{.items[0].metadata.name}")
  kubectl --namespace default port-forward $POD_NAME 9093
```



```
#####  
####  
#####   WARNING: Pod Security Policy has been disabled by default since   #  
#####  
#####   it deprecated after k8s 1.25+. use                               #  
#####  
#####   (index .Values "prometheus-node-exporter" "rbac"                   #  
#####  
#####   "pspEnabled") with (index .Values                                 #  
#####  
#####   "prometheus-node-exporter" "rbac" "pspAnnotations")               #  
#####  
#####   in case you still need it.                                         #  
#####  
#####  
#####  
  
The Prometheus PushGateway can be accessed via port 9091 on the following DNS  
name from within your cluster:  
prometheus-prometheus-pushgateway.default.svc.cluster.local  
  
Get the PushGateway URL by running these commands in the same shell:  
export POD_NAME=$(kubectl get pods --namespace default -l "app=prometheus-p  
ushgateway,component=pushgateway" -o jsonpath="{.items[0].metadata.name}")  
kubectl --namespace default port-forward $POD_NAME 9091  
  
For more information on running Prometheus, visit:  
https://prometheus.io/
```

- Exposer le service Prometheus :

Pour accéder à l'interface web de Prometheus depuis votre navigateur, il faut exposer le service prometheus-server en dehors du cluster Kubernetes. On va utiliser un service de type NodePort qui assigne un port aléatoire sur chaque noeud du cluster.

```
kubectl expose service prometheus-server --type=NodePort --target-port=9090 --  
name=prometheus-server-ext
```

Cette commande crée un nouveau service nommé prometheus-server-ext de type NodePort. Le service expose le port 9090 de Grafana sur un port aléatoire accessible depuis l'extérieur du cluster.

```
achouak@achouak:~/projet/prometheus$ kubectl expose service prometheus-server  
--type=NodePort --target-port=9090 --name=prometheus-server-ext  
  
service/prometheus-server-ext exposed
```

## 1-2 Installation grafana avec helm :

On va installer Grafana, un outil de visualisation pour les métriques collectées par Prometheus, en utilisant Helm.

- Ajout du dépôt Helm :

On ajoute d'abord le dépôt officiel de Grafana qui contient la chart Grafana :

```
helm repo add grafana https://grafana.github.io/helm-charts
```

```
achouak@achouak:~/projet/prometheus$ helm repo add grafana https://grafana.github.io/helm-charts
"grafana" has been added to your repositories
```

- Mise à jour du dépôt :

Ensuite, on met à jour la liste des charts disponibles dans le dépôt :

#### helm repo update

```
achouak@achouak:~/projet/prometheus$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "grafana" chart repository
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "stable" chart repository
Update Complete. ☐Happy Helming!☐
```

- Installation de Grafana :

Maintenant, on installe Grafana en utilisant la chart grafana du dépôt officiel (stable) :

#### helm install grafana stable/grafana

Cette commande va déployer Grafana sur votre cluster Kubernetes.

```
achouak@achouak:~/projet/prometheus$ helm install grafana grafana/grafana
NAME: grafana
LAST DEPLOYED: Thu Apr  4 04:37:26 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get your 'admin' user password by running:

    kubectl get secret --namespace default grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo

2. The Grafana server can be accessed via port 80 on the following DNS name from within your cluster:

    grafana.default.svc.cluster.local

    Get the Grafana URL to visit by running these commands in the same shell:
    export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=grafana" -o jsonpath="{.items[0].metadata.name}")
    kubectl --namespace default port-forward $POD_NAME 3000

3. Login with the password from step 1 and the username: admin
#####
#####
##### WARNING: Persistence is disabled!!! You will lose your data when #####
#####
##### the Grafana pod is terminated. #####
#####
#####
```

- Exposer le service Grafana :

Pour accéder à l'interface web de Grafana depuis votre navigateur, il faut exposer le service grafana en dehors du cluster Kubernetes. On va utiliser un service de type NodePort qui assigne un port aléatoire sur chaque noeud du cluster.

```
kubectl expose service grafana --type=NodePort --target-port=3000 --name=grafana-ext
```

Cette commande crée un nouveau service nommé grafana-ext de type NodePort. Le service expose le port 3000 de Grafana sur un port aléatoire accessible depuis l'extérieur du cluster.

```
achouak@achouak:~/projet/prometheus$ kubectl expose service grafana --type=NodePort --target-port=3000 --name=grafana-ext
service/grafana-ext exposed
```

## 2- Vérifier l'installation :

- La commande `kubectl get pods` montre que tous les pods sont en cours d'exécution et **prêts**.
- La commande `kubectl get service` montre que tous les services sont **disponibles**.
- La commande `kubectl get deployment` montre que tous les déploiements sont à jour et ont le nombre de **répliques souhaité**.

NAME	READY	STATUS	RESTARTS	AGE
pod/backend-deployment-6d68bbb7c6-ggxfp	1/1	Running	0	51m
pod/backend-deployment-6d68bbb7c6-xrlr9	1/1	Running	0	51m
pod/frontend-node-deployment-548f8fc49-j22kd	1/1	Running	15 (84m ago)	4d18h
pod/grafana-57466d69-hzsk8	1/1	Running	14 (84m ago)	2d12h
pod/php-apache-598b474864-rpgkn	1/1	Running	17 (84m ago)	4d23h
pod/prometheus-alertmanager-0	1/1	Running	48	2d12h
pod/prometheus-kube-state-metrics-65468947fb-5wskf	1/1	Running	36 (79m ago)	2d12h
pod/prometheus-prometheus-node-exporter-rztcp	1/1	Running	46 (84m ago)	2d12h
pod/prometheus-prometheus-pushgateway-76976dc66-wcvtm	1/1	Running	17 (84m ago)	2d12h
pod/prometheus-server-5bb9fdbccb-tk4kr	2/2	Running	34 (84m ago)	2d12h
pod/redis-main-55cd4489f4-zzhfj	1/1	Running	0	53m
pod/redis-replical-68f95fbc6d-52rpk	1/1	Running	0	52m
pod/redis-replical-68f95fbc6d-78q8x	1/1	Running	0	52m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/backend-service	LoadBalancer	10.102.108.157	<pending>	8080:32452/TCP	14h
service/frontend-node-service	LoadBalancer	10.110.96.131	<pending>	8080:30526/TCP	4d18h
service/grafana	ClusterIP	10.97.74.83	<none>	80/TCP	2d12h
service/grafana-ext	NodePort	10.107.110.173	<none>	80:30441/TCP	43h
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	4d23h
service/php-apache	ClusterIP	10.110.113.165	<none>	80/TCP	4d23h
service/prometheus	NodePort	10.96.130.185	<none>	9090:30235/TCP	3d17h
service/prometheus-alertmanager	ClusterIP	10.103.150.199	<none>	9093/TCP	2d12h
service/prometheus-alertmanager-headless	ClusterIP	None	<none>	9093/TCP	2d12h
service/prometheus-kube-state-metrics	ClusterIP	10.103.85.77	<none>	8080/TCP	2d12h
service/prometheus-prometheus-node-exporter	ClusterIP	10.97.45.205	<none>	9100/TCP	2d12h
service/prometheus-prometheus-pushgateway	ClusterIP	10.109.224.151	<none>	9091/TCP	2d12h
service/prometheus-server	ClusterIP	10.105.100.246	<none>	80/TCP	2d12h
service/prometheus-server-ext	NodePort	10.110.101.118	<none>	80:31873/TCP	45h
service/redis-main	ClusterIP	10.111.25.70	<none>	6379/TCP	52m
service/redis-replical	ClusterIP	10.105.118.210	<none>	6379/TCP	52m

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/backend-deployment	2/2	2	2	51m
deployment.apps/frontend-node-deployment	1/1	1	1	4d18h
deployment.apps/grafana	1/1	1	1	2d12h
deployment.apps/php-apache	1/1	1	1	4d23h
deployment.apps/prometheus-kube-state-metrics	1/1	1	1	2d12h
deployment.apps/prometheus-prometheus-pushgateway	1/1	1	1	2d12h
deployment.apps/prometheus-server	1/1	1	1	2d12h
deployment.apps/redis-main	1/1	1	1	53m
deployment.apps/redis-replical	2/2	2	2	52m

## 3- Accéder à Prometheus :

Pour accéder à l'interface web de Prometheus, On doit connaître le numéro de port assigné par le service prometheus-server-ext. On doit l'obtenir en exécutant la commande suivante :

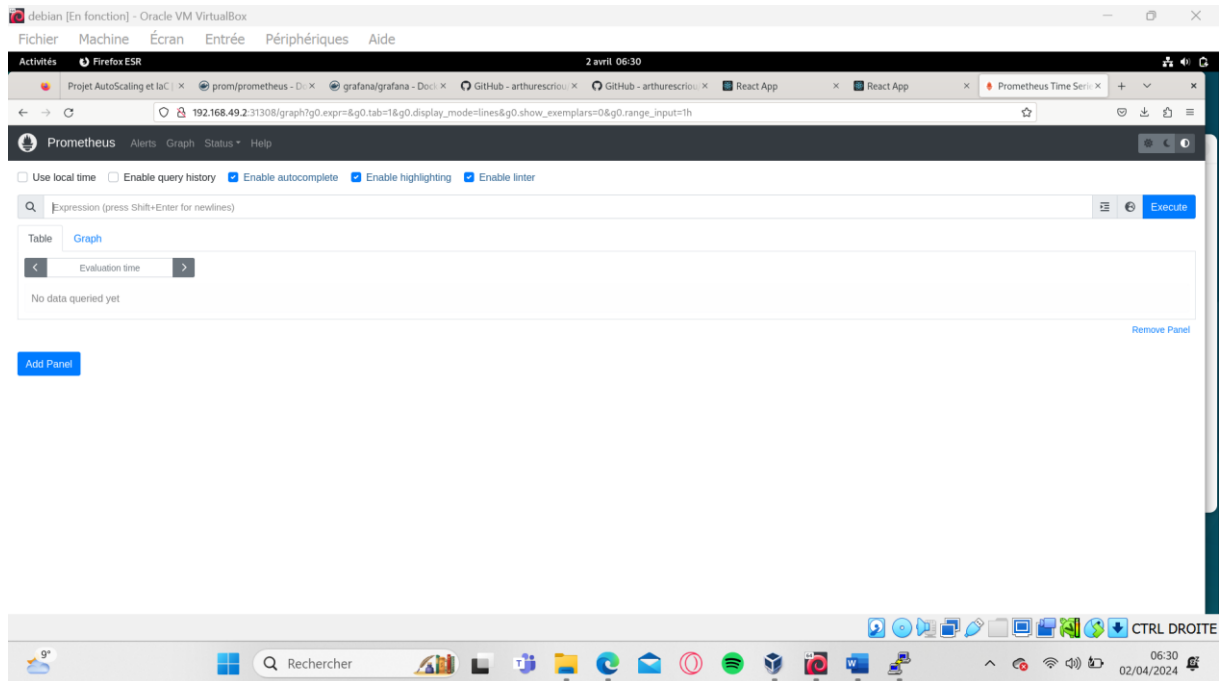
```
minikube service prometheus-server-ext --url
```

## Projet AutoScaling et Iac

NAMESPACE	NAME	TARGET PORT	URL
default	prometheus-server-ext	80	http://192.168.49.2:31873

On peut accéder à Prometheus en ouvrant l'URL suivante :

<http://192.168.49.2:31873>



- Pour vérifier les cibles (cibles) :

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://192.168.49.2:8443/metrics	UP	instance="192.168.49.2:8443" job="kubernetes-apiservers"	25.667s ago	123.683ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://kubernetes.default.svc/api/v1/nodes/minikube/proxy/metrics	UP	beta.kubernetes.io/arch="amd64" beta.kubernetes.io/os="linux" instance="minikube" job="kubernetes-nodes" kubernetes.io/hostname="minikube" kubernetes.io/hostname="minikube" kubernetes.io/hostname="minikube" minikube.k8s.io/commit="f223e6e809a4d797f7d7b14e979705912d" minikube.k8s.io/name="minikube" minikube.k8s.io/primary="true" minikube.k8s.io/updated-at="2024-04-01T17:30:17.0709" minikube.k8s.io/version="v1.32.0"	1m 3s ago	1.305s	

# Projet AutoScaling et Iac

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://kubernetes.default.svc/api/v1/nodes/minikube/proxy/metrics/cadvisor	UP	<a href="#">k8s.kubernetes.io arch="amd64"</a> <a href="#">k8s.kubernetes.io os="linux"</a> <a href="#">instance="minikube"</a> <a href="#">job="kubernetes-nodes-cadvisor"</a> <a href="#">kubernetes.io arch="amd64"</a> <a href="#">kubernetes.io hostname="minikube"</a> <a href="#">kubernetes.io os="linux"</a> <a href="#">minikube.k8s.io kubeid="3220a6a8950a6d79772d7b16c4979f050512d"</a> <a href="#">minikube.k8s.io name="minikube"</a> <a href="#">minikube.k8s.io primary="true"</a> <a href="#">minikube.k8s.io updated.at="2024-04-01T17:39:17.070Z"</a> <a href="#">minikube.k8s.io version="v1.32.0" ▾</a>	56.507s ago	133.108ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.244.0.165:8080/metrics	UP	<a href="#">app.kubernetes.io instance="kube-state-metrics"</a> <a href="#">app.kubernetes.io managed-by="helm"</a> <a href="#">app.kubernetes.io name="kube-state-metrics"</a> <a href="#">helm.sh chart="kube-state-metrics 2.9.0" instance="10.244.0.165-8080"</a> <a href="#">job="kubernetes-service-endpoints" namespace="kube-system"</a> <a href="#">node="minikube" service="kube-state-metrics" ▾</a>	24.36s ago	78.138ms	
http://10.244.0.154:8080/metrics	UP	<a href="#">app.kubernetes.io component="metrics"</a> <a href="#">app.kubernetes.io instance="prometheus"</a> <a href="#">app.kubernetes.io managed-by="helm"</a> <a href="#">app.kubernetes.io name="kube-state-metrics"</a>	8.178s ago	76.021ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.244.0.154:8080/metrics	UP	<a href="#">app.kubernetes.io name="kube-state-metrics"</a> <a href="#">helm.sh chart="kube-state-metrics 2.9.0" instance="10.244.0.165-8080"</a> <a href="#">job="kubernetes-service-endpoints" namespace="kube-system"</a> <a href="#">node="minikube" service="kube-state-metrics" ▾</a>	8.178s ago	76.021ms	
http://10.244.0.154:8080/metrics	UP	<a href="#">app.kubernetes.io component="metrics"</a> <a href="#">app.kubernetes.io instance="prometheus"</a> <a href="#">app.kubernetes.io managed-by="helm"</a> <a href="#">app.kubernetes.io name="kube-state-metrics"</a> <a href="#">app.kubernetes.io part-of="kube-state-metrics"</a> <a href="#">app.kubernetes.io version="v1.11.0"</a> <a href="#">helm.sh chart="kube-state-metrics 5.18.0" instance="10.244.0.154-8080"</a> <a href="#">job="kubernetes-service-endpoints" namespace="default"</a> <a href="#">node="minikube" service="prometheus-kube-state-metrics" ▾</a>	10.37s ago	30.682ms	
http://192.168.49.2:9100/metrics	UP	<a href="#">instance="10.244.0.167-9151" job="kubernetes-service-endpoints"</a> <a href="#">k8s.app="kube-dns" kubernetes.io cluster-service="true"</a> <a href="#">kubernetes.io name="CoreDNS" namespace="kube-system"</a> <a href="#">node="minikube" service="kube-dns" ▾</a>	57.459s ago	59.070ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.244.0.167:9153/metrics	UP	<a href="#">instance="10.244.0.167-9151" job="kubernetes-service-endpoints"</a> <a href="#">k8s.app="kube-dns" kubernetes.io cluster-service="true"</a> <a href="#">kubernetes.io name="CoreDNS" namespace="kube-system"</a> <a href="#">node="minikube" service="kube-dns" ▾</a>	57.459s ago	59.070ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://192.168.49.2:9100/metrics	UP	<a href="#">app.kubernetes.io component="metrics"</a> <a href="#">app.kubernetes.io instance="prometheus"</a> <a href="#">app.kubernetes.io managed-by="helm"</a> <a href="#">app.kubernetes.io name="prometheus-node-exporter"</a> <a href="#">app.kubernetes.io part-of="prometheus-node-exporter"</a> <a href="#">app.kubernetes.io version="v1.7.0"</a> <a href="#">helm.sh chart="prometheus-node-exporter 4.32.0" instance="192.168.49.2-9100" job="kubernetes-service-endpoints" namespace="default" node="minikube" service="prometheus-prometheus-node-exporter" ▾</a>	52.621s ago	597.358ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	<a href="#">instance="localhost:9090" job="prometheus" ▾</a>	58.425s ago	275.860ms	

D'après les images Prometheus fonctionne correctement. On peut voir que toutes les cibles (targets) sont "up", ce qui signifie que Prometheus est capable de collecter des métriques auprès d'elles.

#### 4- Accéder à Grafana et récupérer les identifiants :

Pour accéder à l'interface web de Grafana, on doit connaître le numéro de port assigné par le service grafana-ext. On peut l'obtenir en exécutant la commande suivante :

```
minikube service grafana-ext --url
```

NAMESPACE	NAME	TARGET PORT	URL
default	grafana-ext	80	http://192.168.49.2:30441

On peut accéder à Grafana en ouvrant l'URL suivante :

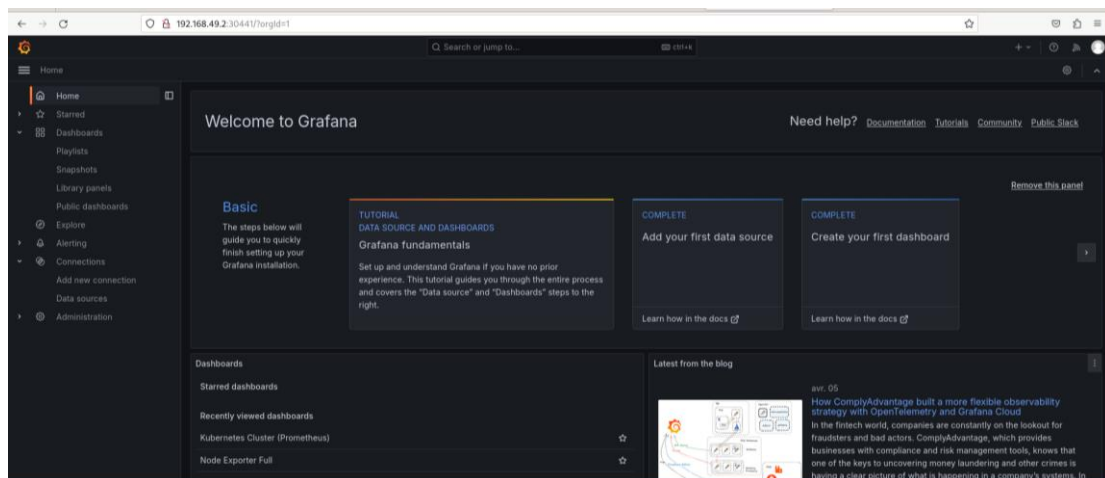
<http://192.168.49.2:30441>

- Récupération du nom d'utilisateur et du mot de passe Grafana :

Par défaut, Grafana génère automatiquement un nom d'utilisateur et un mot de passe stockés de manière sécurisée dans un secret Kubernetes. On peut le récupérer en exécutant la commande suivante :

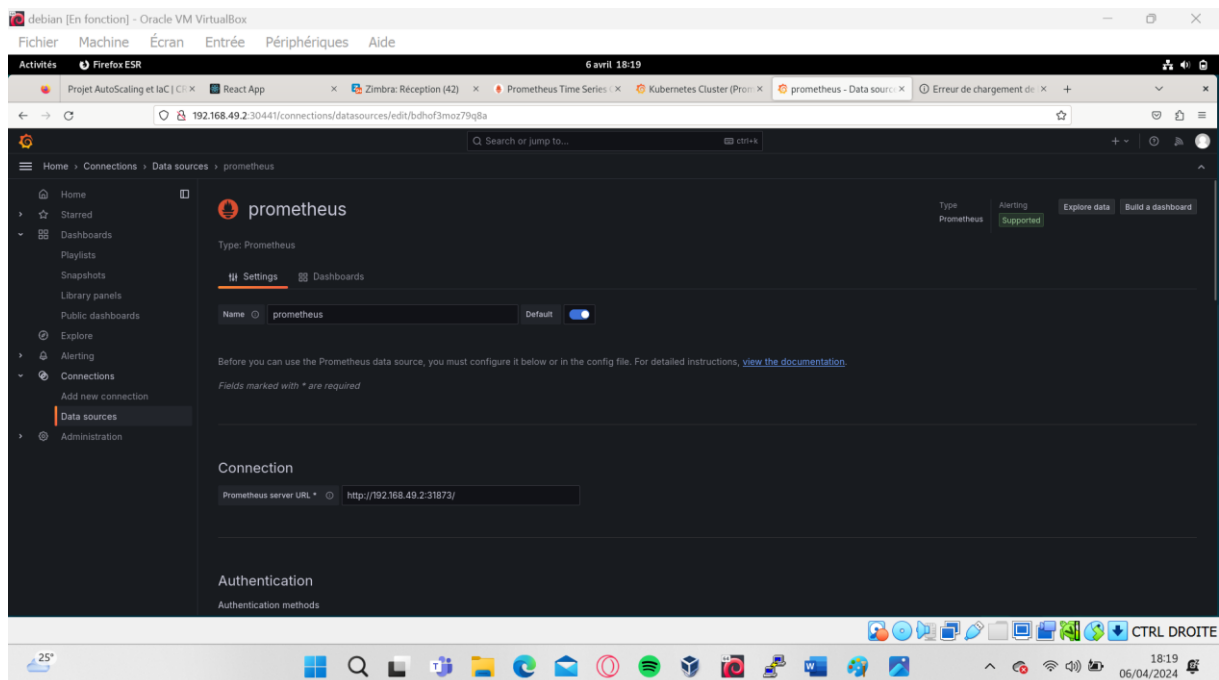
```
kubectl get secret --namespace default grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo
```

```
achouak@achouak:~/projet/prometheus$ kubectl get secret --namespace default grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo  
hGCj81B3zG67YkS6PWx19DWDt8jXWx51QlcM6ceR
```



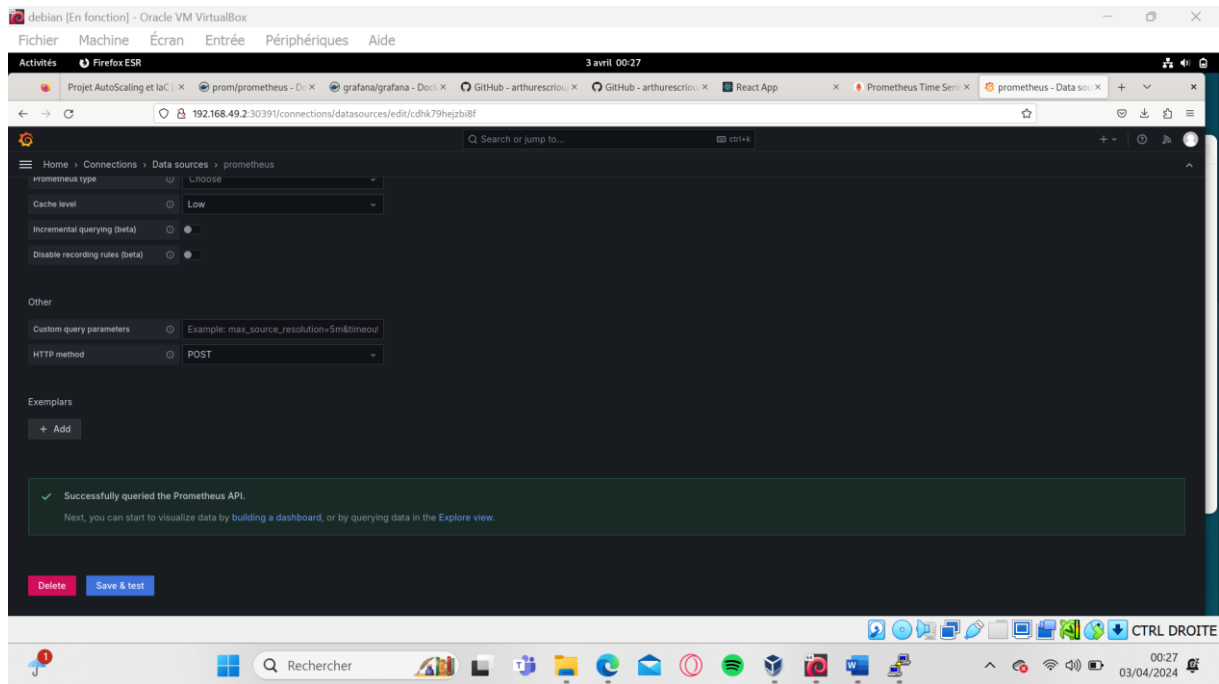
- Création d'une source de données Prometheus dans Grafana :

Dans le champ URL du serveur Prometheus, on met l'URL de serveur Prometheus. L'URL est <http://192.168.0.49:31873>



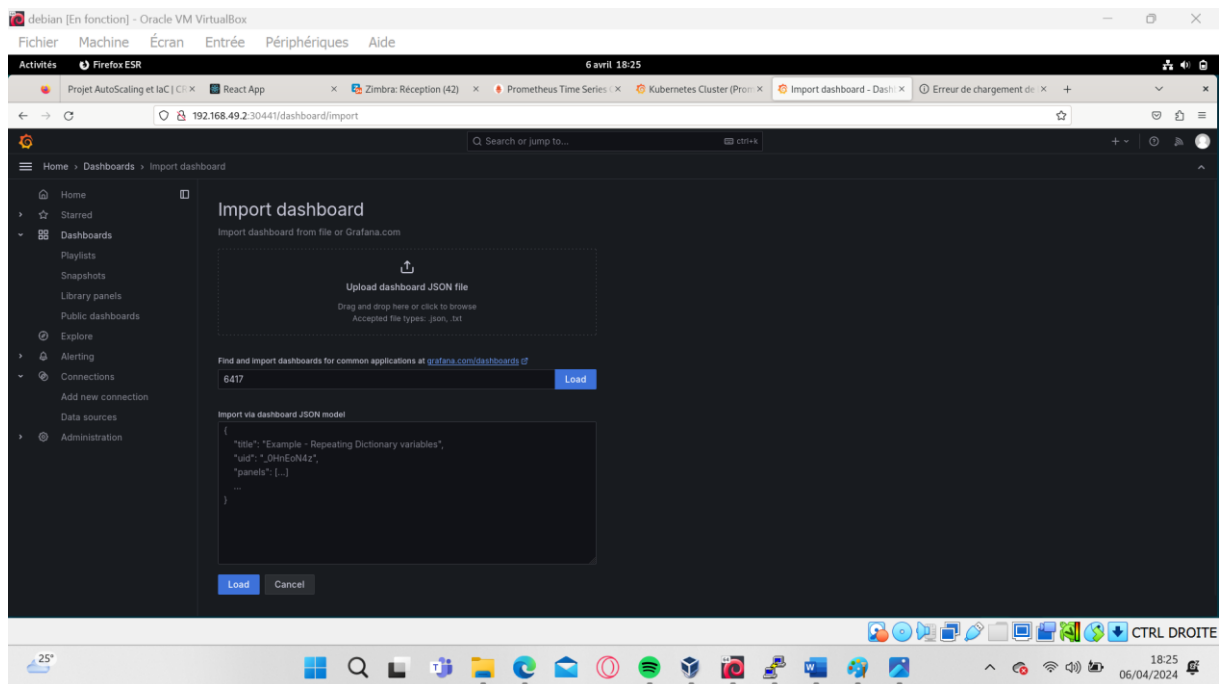
## Projet AutoScaling et Iac

- Test de la source de données :



Le test de la source de données est réussi, le message sur la capture indique que la connexion à Prometheus a réussi.

- Importation d'un tableau de bord : (6417)

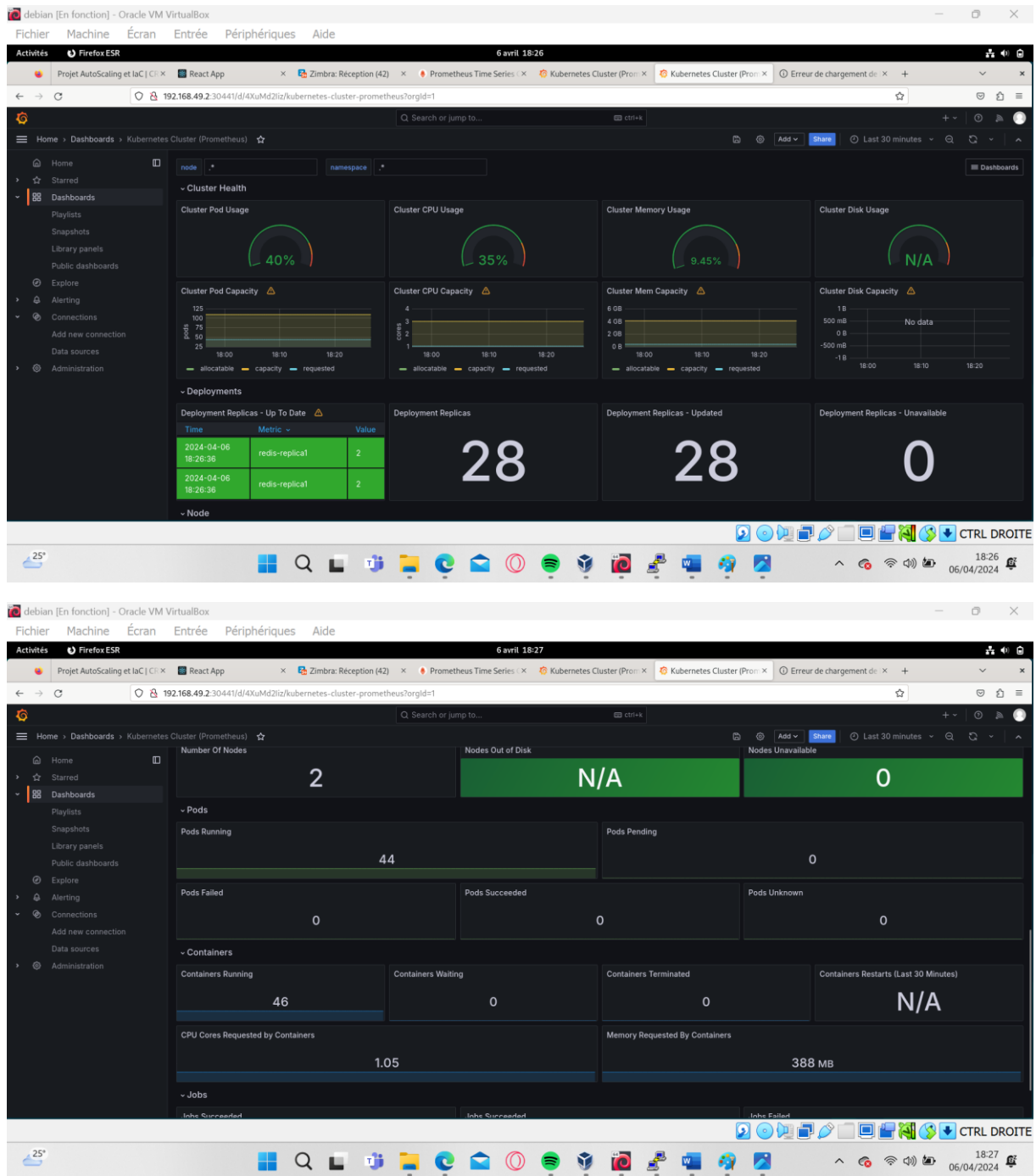


- Vérification de l'importation du tableau de bord :

Les captures montrent un tableau de bord Grafana (6417) qui surveille les performances du cluster Kubernetes. Le tableau de bord est divisé en plusieurs sections, chacune affichant un ensemble de métriques spécifiques.



## Projet AutoScaling et Iac



Prometheus et Grafana ont été déployés avec succès pour surveiller l'application Node.js/React utilisant Redis. La montée à l'échelle automatique et dynamique a été configurée pour garantir que les ressources sont utilisées de manière optimale. L'infrastructure est reproductible et peut être déployée facilement dans d'autres environnements.