

Huu Truc Nguyen
21310174

Rapport pour le projet AutoScaling et IaC

Introduction

Ce projet a pour but de mettre en place les conteneurs redis, NodeJS et React et de les surveiller grâce à Prometheus et Grafana.

Il va être divisé en trois parties, la partie avec Kubernetes, la partie avec Prometheus/ Grafana et finalement, les problèmes rencontrés durant la création du projet

Partie I

La première partie est en rapport avec la création du conteneur redis, nodeJS et frontend. Pour cela, nous avons créé les fichiers YAML de déploiement et de service pour les conteneurs.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: truckat-redis-deployment
  labels:
    app: redis
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
        - name: redis
          image: redis:latest
          ports:
            - containerPort: 6379
```

```
apiVersion: v1
kind: Service
metadata:
  name: truckat-redis-service
spec:
  selector:
    app: redis
  ports:
    - protocol: TCP
      port: 6379
      targetPort: 6379
  type: ClusterIP
```

```

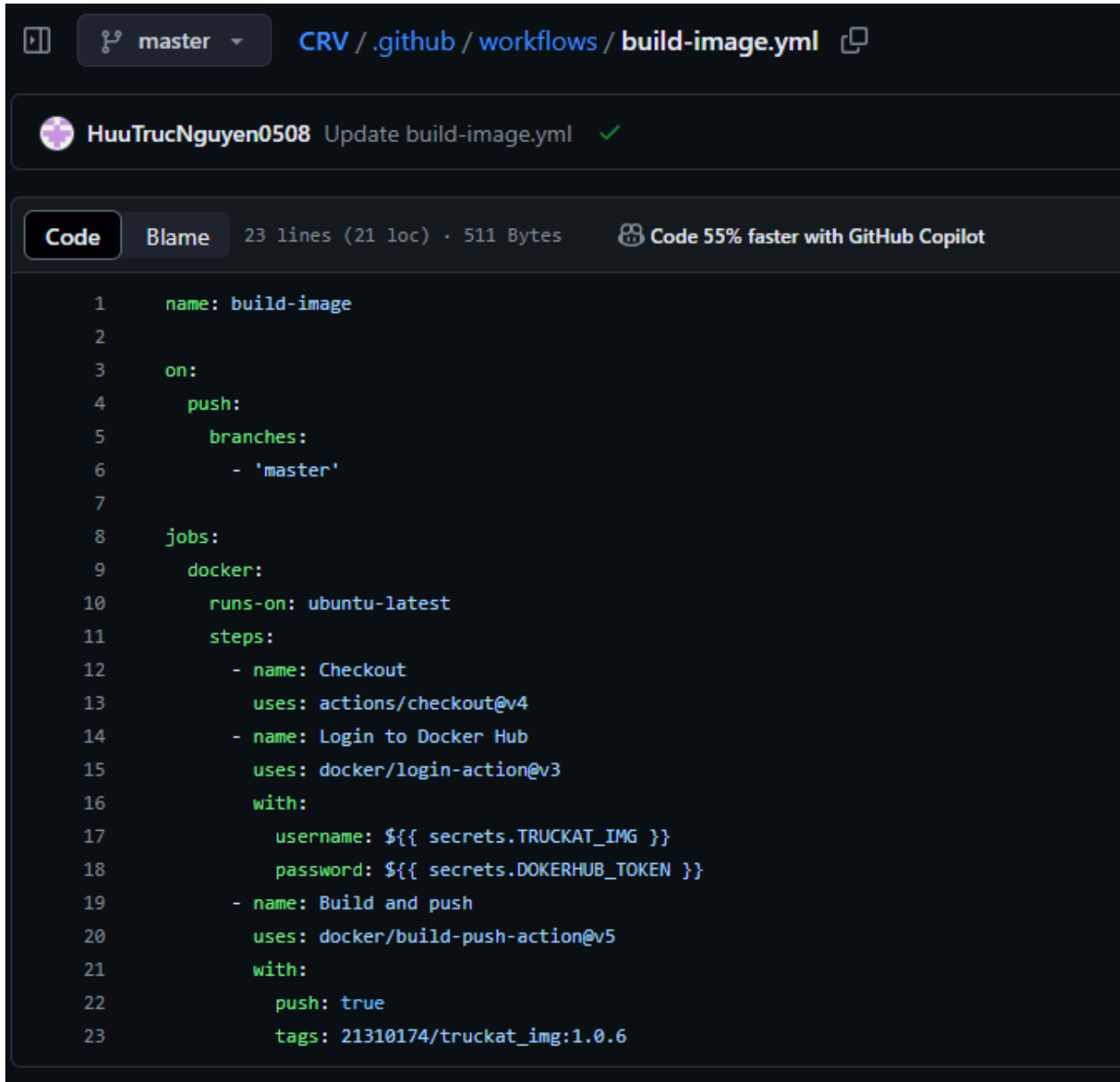
apiVersion: apps/v1
kind: Deployment
metadata:
  name: truckat-node-redis-deployment
  labels:
    app: truckat-node-redis
spec:
  replicas: 1
  selector:
    matchLabels:
      app: truckat-node-redis
  template:
    metadata:
      labels:
        app: truckat-node-redis
    spec:
      containers:
        - name: truckat-node-redis
          imagePullPolicy: Always
          image: arthurescrou/node-redis:1.0.5
          resources:
            requests:
              cpu: "200m"
          ports:
            - containerPort: 8080
          env:
            - name: PORT
              value: '8080'
            - name: REDIS_URL
              value: redis://truckat-redis-service.default.svc.cluster.local:6379
            - name: REDIS_REPLICAS_URL
              value: redis://truckat-redis-service.default.svc.cluster.local:6379

---

apiVersion: v1
kind: Service
metadata:
  name: truckat-node-redis-service
spec:
  selector:
    app: truckat-node-redis
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: LoadBalancer

```

Puisque l'image du frontend n'a pas été donnée, nous avons cloné le github qui contient l'app frontend et rajouté le github workflow pour le compiler en image et le mettre dans docker hub



```
1  name: build-image
2
3  on:
4    push:
5      branches:
6        - 'master'
7
8  jobs:
9    docker:
10     runs-on: ubuntu-latest
11     steps:
12       - name: Checkout
13         uses: actions/checkout@v4
14       - name: Login to Docker Hub
15         uses: docker/login-action@v3
16         with:
17           username: ${ secrets.TRUCKAT_IMG }
18           password: ${ secrets.DOKERHUB_TOKEN }
19       - name: Build and push
20         uses: docker/build-push-action@v5
21         with:
22           push: true
23           tags: 21310174/truckat_img:1.0.6
```

Avec ce code, lorsque nous avons maintenant un image du frontend et pouvons alors écrire le déploiement et service:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: truckat-frontend-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
          image: 21310174/truckat_img:1.0.6
          ports:
            - containerPort: 7654

---

# frontend-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: truckat-frontend-service
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 7654
      targetPort: 7654
  type: LoadBalancer

```

Après avoir fait les principaux conteneurs, il faut maintenant écrire les services et déploiements pour les répliques du redis et l'auto scaling:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: truckat-redis-replica-deployment
  labels:
    app: truckat-redis-replica
spec:
  replicas: 3
  selector:
    matchLabels:
      app: truckat-redis-replica
  template:
    metadata:
      labels:
        app: truckat-redis-replica
    spec:
      containers:
        - name: truckat-redis-replica
          image: redis:latest
          resources:
            requests:
              cpu: "200m"
          ports:
            - containerPort: 6379
          command: ["redis-server", "--slaveof", "redis", "6379", "--port", "6379"]
---

apiVersion: v1
kind: Service
metadata:
  name: truckat-redis-replica-service
spec:
  type: ClusterIP
  ports:
    - protocol: TCP
      port: 6379
      targetPort: 6379
  selector:
    app: truckat-redis-replica

```

Ce code va créer des repliement esclaves pour le conteneur redis

```

apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: truckat-node-redis-autoscaler
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: truckat-node-redis-deployment
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
---
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: truckat-redis-replica-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: truckat-redis-replica-deployment
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50

```

Ces codes vont permettre de déployer automatiquement un nouveau réplique si le taux d'utilisation du CPU atteint les 50%

En mettant tous les codes ensemble, nous avons un seul YAML qui va tout déployer lorsque nous faisons “kubectl apply -f fichier.yaml”.

```

PS C:\Users\nguye\Documents\CRV\Test> kubectl apply -f .\Trucat-project.yaml
deployment.apps/truckat-redis-deployment created
service/truckat-redis-service created
deployment.apps/truckat-node-redis-deployment created
service/truckat-node-redis-service created
deployment.apps/truckat-frontend-deployment created
service/truckat-frontend-service created
deployment.apps/truckat-redis-replica-deployment created
service/truckat-redis-replica-service created
horizontalpodautoscaler.autoscaling/truckat-node-redis-autoscaler created
horizontalpodautoscaler.autoscaling/truckat-redis-replica-hpa created

```

Avec “kubectl get all”, nous aurons:

```
PS C:\Users\nguye> kubectl get all

NAME                                     READY   STATUS    RESTARTS   AGE
pod/truckat-frontend-deployment-5cd7df7ff-smnj9   1/1     Running   4 (14h ago)  19h
pod/truckat-node-redis-deployment-78f7bc6c57-sbdf   1/1     Running   1 (12m ago)  15h
pod/truckat-redis-deployment-6db6bc49cf-gd4ft     1/1     Running   4 (12m ago)  21h
pod/truckat-redis-replica-deployment-849b6846f4-n4ssx 1/1     Running   1 (12m ago)  15h

NAME                                     TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
service/kubernetes                       ClusterIP     10.96.0.1    <none>        443/TCP          48h
service/truckat-frontend-service         LoadBalancer 10.109.113.57 127.0.0.1     7654:31497/TCP   19h
service/truckat-node-redis-service       LoadBalancer 10.105.102.96 127.0.0.1     8080:32367/TCP   37h
service/truckat-redis-replica-service    ClusterIP     10.103.45.168 <none>        6379/TCP         38h
service/truckat-redis-service            ClusterIP     10.105.130.254 <none>        6379/TCP         37h

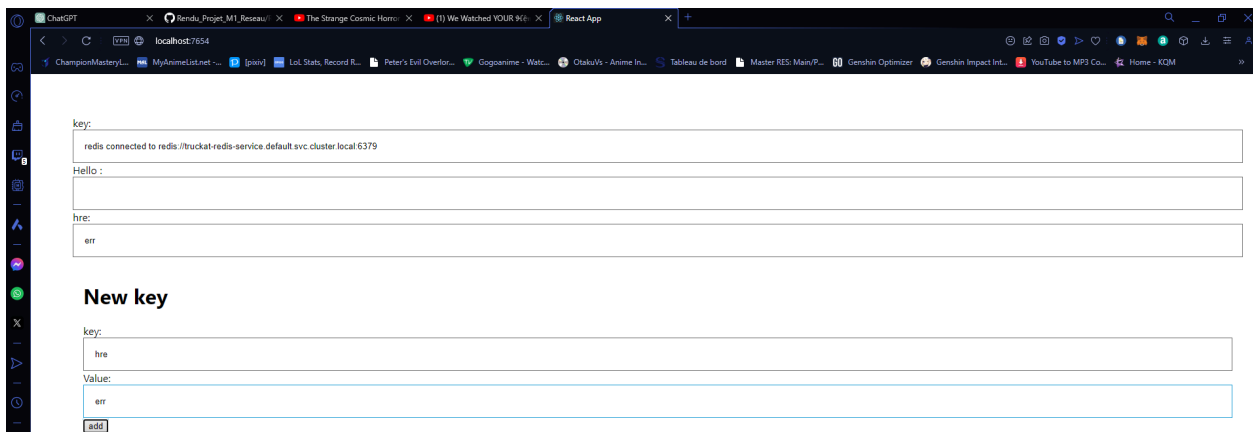
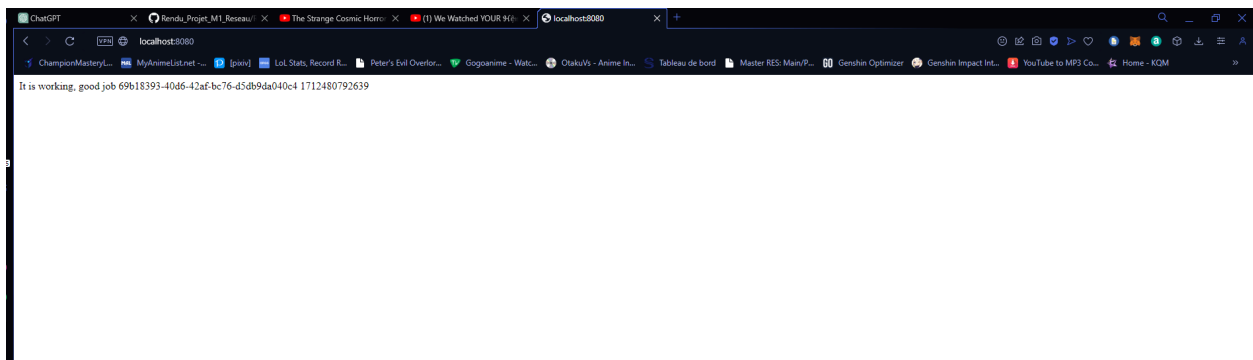
NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/truckat-frontend-deployment 1/1     1             1           19h
deployment.apps/truckat-node-redis-deployment 1/1     1             1           37h
deployment.apps/truckat-redis-deployment     1/1     1             1           37h
deployment.apps/truckat-redis-replica-deployment 1/1     1             1           15h

NAME                                     DESIRED   CURRENT   READY   AGE
replicaset.apps/truckat-frontend-deployment-5cd7df7ff 1         1         1       19h
replicaset.apps/truckat-node-redis-deployment-699d4d6598 0         0         0       37h
replicaset.apps/truckat-node-redis-deployment-78f7bc6c57 1         1         1       15h
replicaset.apps/truckat-redis-deployment-6db6bc49cf 1         1         1       21h
replicaset.apps/truckat-redis-deployment-79b469995f 0         0         0       37h
replicaset.apps/truckat-redis-replica-deployment-849b6846f4 1         1         1       15h

NAME                                     REFERENCE                                     TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
horizontalpodautoscaler.autoscaling/truckat-node-redis-autoscaler  Deployment/truckat-node-redis-deployment  2%/59%   1         10        1          15h
horizontalpodautoscaler.autoscaling/truckat-redis-replica-hpa      Deployment/truckat-redis-replica-deployment  6%/59%   1         10        1          14h
PS C:\Users\nguye>
```

A noter ici que l'image des conteneurs est pris directement d'internet

Avec Minikube tunnel, il est possible d'accéder au nodeJS et frontend a travers localhost:8080 et localhost:7654



Et tous les changements faites sur l'app React sera écrit dans le logs du NodeJS


```

PS C:\Users\nguye> kubectl logs truckat-node-redis-deployment-78f7bc6c57-sbdff
Sun, 07 Apr 2024 09:04:43 GMT: redis connected to redis://truckat-redis-service.default.svc.cluster.local:6379
Sun, 07 Apr 2024 09:04:45 GMT: redis replicas connected to redis://truckat-redis-service.default.svc.cluster.local:6379
Sun, 07 Apr 2024 09:04:45 GMT: Set "key" value to "redis connected to redis://truckat-redis-service.default.svc.cluster.local:6379"
Sun, 07 Apr 2024 09:04:45 GMT: listening at http://localhost:8080 server 69b18393-40d6-42af-bc76-d5db9da040c4
Sun, 07 Apr 2024 09:06:32 GMT: It is working, good job 69b18393-40d6-42af-bc76-d5db9da040c4 1712480792639
Sun, 07 Apr 2024 09:06:45 GMT: get items
Sun, 07 Apr 2024 09:06:45 GMT: get items
Sun, 07 Apr 2024 09:06:45 GMT: get item key
Sun, 07 Apr 2024 09:06:45 GMT: get item key
Sun, 07 Apr 2024 09:07:01 GMT: post item Hello there
Sun, 07 Apr 2024 09:07:01 GMT: get item Hello
Sun, 07 Apr 2024 09:07:01 GMT: get item Hello
Sun, 07 Apr 2024 09:07:06 GMT: post item Hello there
Sun, 07 Apr 2024 09:07:06 GMT: get item Hello
Sun, 07 Apr 2024 09:07:06 GMT: get item Hello
Sun, 07 Apr 2024 09:07:10 GMT: post item Hello yo
Sun, 07 Apr 2024 09:07:10 GMT: get item Hello
Sun, 07 Apr 2024 09:07:10 GMT: get item Hello
Sun, 07 Apr 2024 09:07:11 GMT: get items
Sun, 07 Apr 2024 09:07:11 GMT: get items
Sun, 07 Apr 2024 09:07:11 GMT: get item key
Sun, 07 Apr 2024 09:07:11 GMT: get item Hello
Sun, 07 Apr 2024 09:07:11 GMT: get item key
Sun, 07 Apr 2024 09:07:11 GMT: get item Hello
Sun, 07 Apr 2024 09:07:12 GMT: get items
Sun, 07 Apr 2024 09:07:12 GMT: get items
Sun, 07 Apr 2024 09:07:12 GMT: get item Hello
Sun, 07 Apr 2024 09:07:12 GMT: get item key
Sun, 07 Apr 2024 09:07:12 GMT: get item Hello
Sun, 07 Apr 2024 09:07:12 GMT: get item key
Sun, 07 Apr 2024 09:07:48 GMT: get items
Sun, 07 Apr 2024 09:07:48 GMT: get items
Sun, 07 Apr 2024 09:07:48 GMT: get item key
Sun, 07 Apr 2024 09:07:48 GMT: get item Hello
Sun, 07 Apr 2024 09:07:48 GMT: get item key
Sun, 07 Apr 2024 09:07:48 GMT: get item Hello
Sun, 07 Apr 2024 09:07:48 GMT: get items
Sun, 07 Apr 2024 09:07:48 GMT: get items
Sun, 07 Apr 2024 09:07:48 GMT: get item key
Sun, 07 Apr 2024 09:07:48 GMT: get item Hello
Sun, 07 Apr 2024 09:07:48 GMT: get item key
Sun, 07 Apr 2024 09:07:49 GMT: get items
Sun, 07 Apr 2024 09:07:49 GMT: get items
Sun, 07 Apr 2024 09:07:49 GMT: get item key
Sun, 07 Apr 2024 09:07:49 GMT: get item Hello
Sun, 07 Apr 2024 09:07:49 GMT: get item key
Sun, 07 Apr 2024 09:07:49 GMT: get item Hello
Sun, 07 Apr 2024 09:07:49 GMT: get items
Sun, 07 Apr 2024 09:07:49 GMT: get items
Sun, 07 Apr 2024 09:07:49 GMT: get item Hello
Sun, 07 Apr 2024 09:07:49 GMT: get item key

```

Partie II

La partie concerne la mise en place de Prometheus et Grafana. Pour cela, j'ai utilisé la ligne de commande qui permette de créer un conteneur Prometheus et Grafana:

```
docker run --name prometheus -d -p 127.0.0.1:9090:9090 prom/prometheus
```

```
docker run -d --name=grafana -p 3000:3000 grafana/grafana
```

Ces deux lignes de commande permettent vont prendre l'image de Prometheus et Grafana, créer pour chacun un conteneur et l'expose au port 9090 et 3000 respectivement

Images [Give feedback](#)

Local **Hub**

1.89 GB / 4.53 GB in use 3 Images Last refresh: 23 hours ago

Search

Name	Tag	Status	Created	Size	Actions
prom/prometheus e350b167c4fa	latest	In use	10 days ago	261.54 MB	▶ ⋮ 🗑
grafana/grafana ccb72d0beb64	latest	In use	17 days ago	429.9 MB	▶ ⋮ 🗑
gcr.io/k8s-minikube/kicbase dbc548475405	v0.0.42	In use	5 months ago	1.19 GB	▶ ⋮ 🗑

Containers [Give feedback](#)

Container CPU usage ⓘ Container memory usage ⓘ Show charts

No containers are running. No containers are running.

Search Only show running containers

Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
grafana bebb6849e066	grafana/grafana	Exited	N/A	3000:3000	3 hours ago	▶ ⋮ 🗑
prometheus b473e51242ec	prom/prometheus	Exited	N/A	9090:9090	3 hours ago	▶ ⋮ 🗑
minikube e1790d4bccd2	gcr.io/k8s-minikube/kicbase:v0.0.42	Exited (137)	N/A	0:22 Show all ports (5)	3 hours ago	▶ ⋮ 🗑

On peut alors accéder grâce à localhost:9090 et localhost:3000

localhost:9090/graph

Prometheus Alerts Graph Status Help

☐ Use local time
 ☐ Enable query history
 ☒ Enable autocomplete
 ☒ Enable highlighting
 ☒ Enable linter

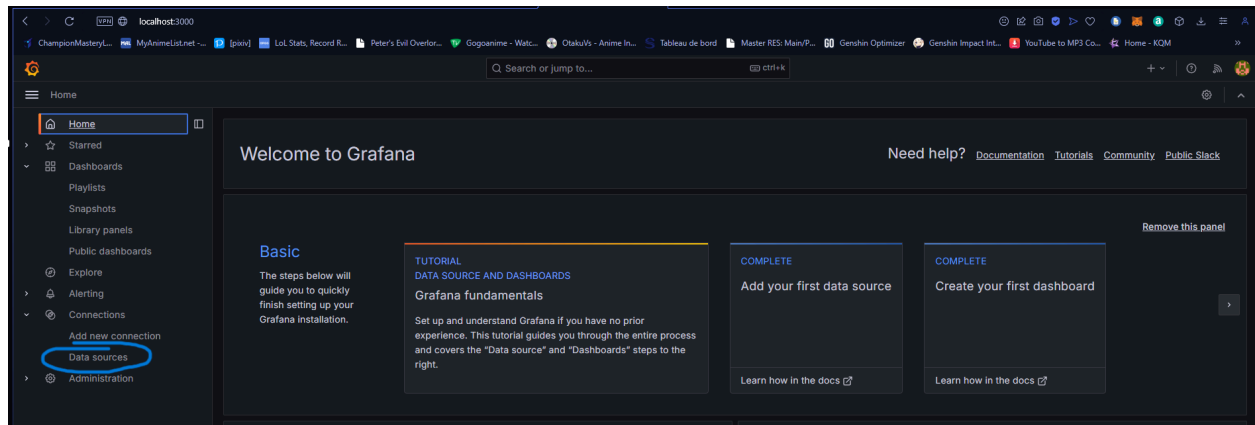
Search Expression (press Shift+Enter for newlines) Execute

Table Graph

← Evaluation time →

No data queried yet

Add Panel Remove Panel



Cependant, Prometheus ne va en ce moment voir lui-même car nous avons pas rajouter les ports qu'il doit voir. Il faut donc aller dans ce fichier prometheus.yml et ajouter les scrap_configs avec "docker exec -it prometheus /bin/sh -c 'cd /etc/prometheus && vi prometheus.yml && kill -HUP 1' ".

A l'intérieur, il faut rajouter:

- job_name: "WMI Exporter"
static_configs:
 - targets: ["host.docker.internal:9182"]

- job_name: "Grafana"
static_configs:
 - targets: ["host.docker.internal:3000"]

- job_name: "node-redis"
static_configs:
 - targets: ["host.docker.internal:8080"]

A la fin du fichier tout en gardant les mêmes indentations, sauvegarder et quitter le fichier. Nous devons avoir le contenu présent:

```

# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
      - targets:
        # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label 'job=<job_name>' to any timeseries scraped from this config.
  - job_name: "prometheus"

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ["localhost:9090"]

  - job_name: "WMI Exporter"
    static_configs:
      - targets: ["host.docker.internal:9182"]

  - job_name: "Grafana"
    static_configs:
      - targets: ["host.docker.internal:3000"]

  - job_name: "node-redis"
    static_configs:
      - targets: ["host.docker.internal:8080"]

```

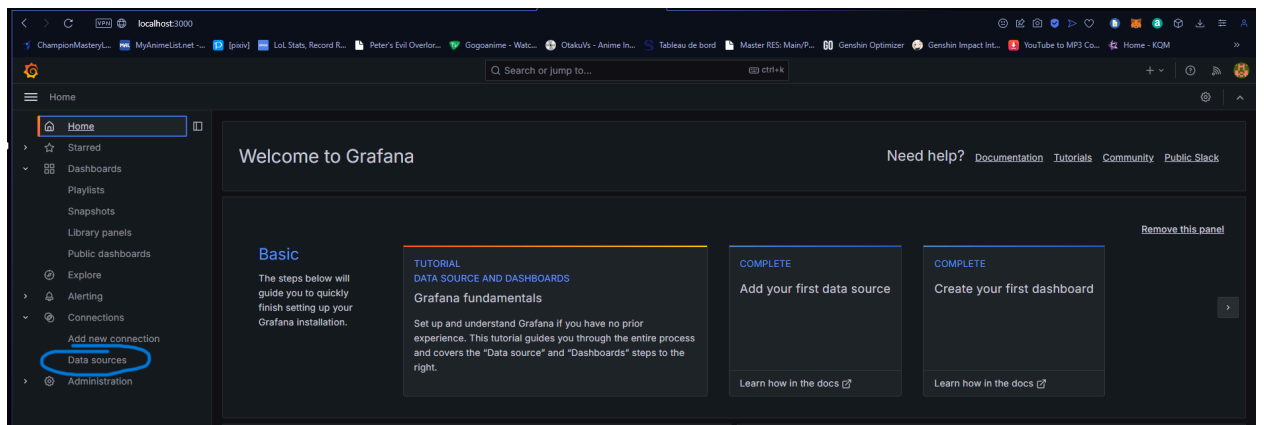
Ajouter ces contenus vont permettre à Prometheus de voir les ports du NodeJS (notre objectif principal) et donc quand nous rafraichissons la page Prometheus et allons dans Status -> Targets, nous devrons avoir:

The screenshot shows the Prometheus Targets page with a table of scraped targets. The table has columns for Endpoint, State, Labels, Last Scrape, Scrape Duration, and Error. The targets listed are Grafana, WMI Exporter, node-red, and Prometheus.

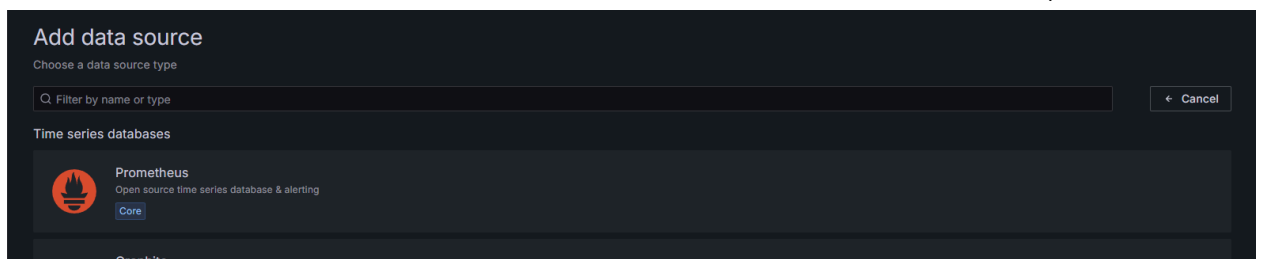
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
Grafana (1/1 up)					
http://host.docker.internal:3000/metrics	UP	instance="host.docker.internal:3000" job="Grafana"	1.603s ago	7.300ms	
WMI Exporter (1/1 up)					
http://host.docker.internal:9182/metrics	UP	instance="host.docker.internal:9182" job="WMI Exporter"	1.445s ago	772.704ms	
node-red (1/1 up)					
http://host.docker.internal:8080/metrics	UP	instance="host.docker.internal:8080" job="node-red"	7.862s ago	9.712ms	
prometheus (1/1 up)					
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	11.819s ago	14.127ms	

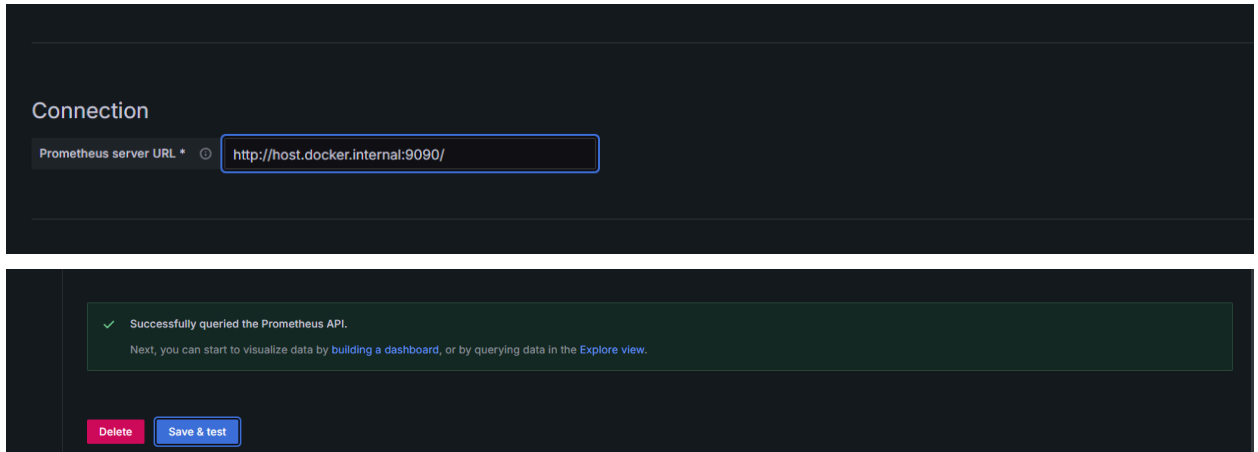
Du côté de Grafana, il faut d'abord créer un data source:

- Aller dans l'option datasources qui se trouve à gauche

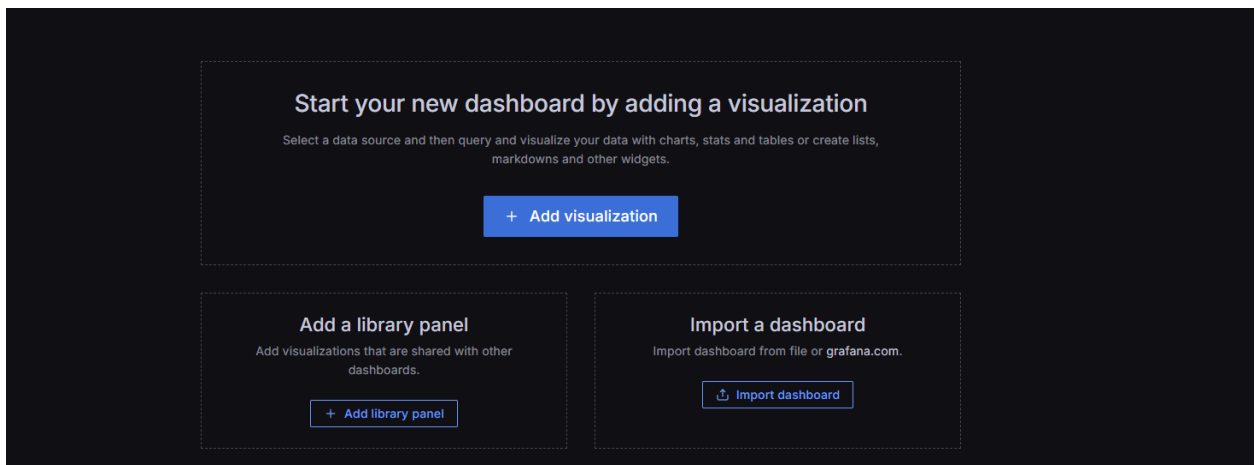


- Et choisissez Prometheus comme type de data source a creer
- Ensuite, il faut mettre l'URL de Prometheus (ATTENTION, vu que nous nous travaillons dans des conteneurs docker, localhost:9090 deviendra host.docker.internal:9090)






- Maintenant aller dans Dashboard et choisissez l'option d'import de dashboard:



- Mettez ensuite 11159 dans la barre ID et lorsque vous appuyez sur le bouton load, Grafana va vous dire de choisir le data source. Vous choisissez celui que nous avons créé (le défaut) et appuyez sur "Import"

Import dashboard

Import dashboard from file or Grafana.com


Upload dashboard JSON file
Drag and drop here or click to browse
Accepted file types: .json, .txt

Find and import dashboards for common applications at grafana.com/dashboards

Import via dashboard JSON model

```
{
  "title": "Example - Repeating Dictionary variables",
  "uid": "_OHnEoN4z",
  "panels": [...]
  ...
}
```

Published by

Joey Yang

Updated on

2019-11-10 14:39:46

Options


Name


Folder

Unique identifier (UID)

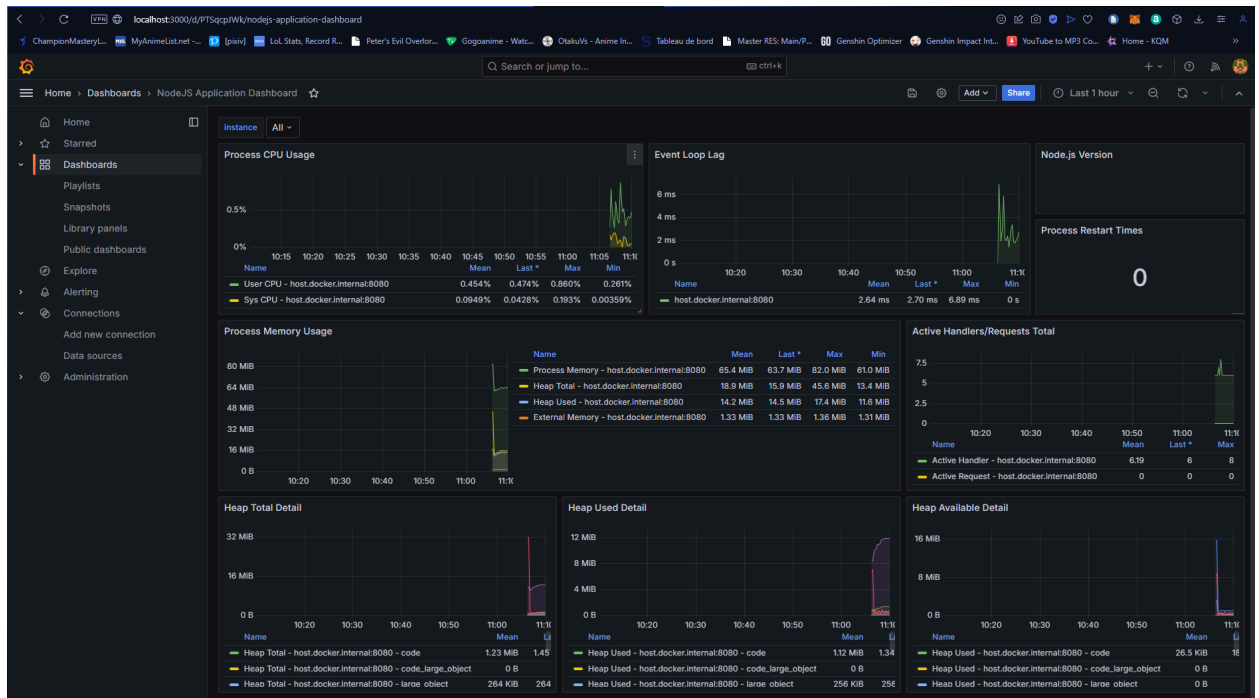
The unique identifier (UID) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The UID allows having consistent URLs for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.

prometheus

 prometheus

 prometheus **default** Prometheus

- Nous avons maintenant un dashboard Grafana qui va surveillez le NodeJS



Partie III

Durant ce projet, nous avons rencontré beaucoup de problèmes, certains fausses pistes et des fichiers incomplets:

- Pour que les conteneur d'auto scaling horizontal puissent fonctionner proprement, nous devons rajouter l'addons metrics ainsi que rajouter l'option ressources dans les spec du contenu. Nous avons dû prendre beaucoup de temps de recherche avant de trouver.

```
spec:
  containers:
    - name: truckat-redis-replica
      image: redis:latest
      resources:
        requests:
          cpu: "200m"
```

Avant ça, lorsque nous faisons "kubectl get all", nous avons:

```
Deployment/truckat-redis-replica-deployment <unknown>/50%
```


- La même chose s'est produite lorsque nous voulions exposer le frontend. En effet, puisque le port dans le Dockerfile était différent du port du frontend, à part la méthode de "port-forward", il était impossible d'accéder au frontend



```
1 FROM node:14
2
3 WORKDIR /src
4
5 COPY . .
6
7 RUN yarn install
8
9 ENV PORT=7654
10
11 CMD ["yarn", "start"]
```

- Finalement, le problème qui nous a pris le plus de temps était la mise en place de Prometheus et Grafana. Au début, nous avons voulu installer Prometheus et Grafana dans Minikube (une possibilité) mais étions incapables de le faire. De plus, il fallut beaucoup de recherches avant de comprendre comment faire pour que Prometheus puisse voir les ports. Après avoir découvert la possibilité d'installer Prometheus et Grafana à travers Docker, il ne restait plus qu'ajouter les ports dans le fichier prometheus.yml tout en changeant le port de localhost:8080 en host.docker.internal:8080 (documentation de Docker pour qu'un conteneur puisse voir le port d'extérieur). Et ainsi, Prometheus voit maintenant tous les localhost: que nous voulons.

Ces problèmes viennent surtout d'un manque de connaissance et il fallait passer beaucoup de temps pour rechercher sur Internet avant de trouver la solution

Conclusion

En conclusion, ce projet a été très intéressant, il nous a permis d'avoir un aperçu dans le monde du devOPS et le résultat a été très satisfaisant.

Un grand remerciement à notre chargé de TD pour nous avoir aidé le plus possible et nous a appris comment utiliser le github workflow.