

TD 9

Introduction

Nous réaliserons ce TP au cours des 2 dernières séances. Aujourd'hui notre cluster est doté d'instruments de monitoring performants qui permettent entre autres de visualiser des données sur tous les aspects techniques de notre swarm. Il manque toutefois les données sur l'utilisateur. Il faut en effet savoir : combien de pages ne s'affichent pas correctement ? Combien de temps l'utilisateur met pour accéder aux pages qu'il demande ?

Nous allons essayer d'assurer une qualité de service spécifiée par le SLA (Service-Level Agreement) suivant :

- temps de réponse doit être inférieur à 200ms
- taux d'erreurs 50x doit rester inférieur à 1 %

Partie I – Installation de Traefik

Traefik est un reverse proxy et load balancer qui reçoit toutes les connexions entrantes de notre swarm, les fait suivre au bon service (selon l'URL demandée) et dispatche la charge sur les différentes répliques.

Traefik nous donnera les métriques de temps de réponse et le taux d'erreurs.

1)

Traefik collecte les métriques en étant pointé sur notre Swarm. Il écoute directement ce dernier et génère tous les chemins par lesquels nos microservices sont connectés au monde extérieur.

2)

Nous devons choisir un nom de domaine pour notre application. Sur notre hôte, dans `/etc/hosts` on ajoute « 12.0.0.2 myhsapp.com » aux lignes suivantes :

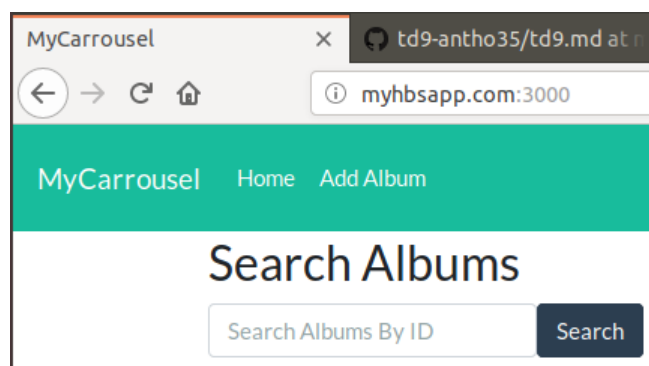
```
Fichier Édition Affichage Rechercher Terminal Aide
GNU nano 2.9.3 hosts

127.0.0.1    localhost
127.0.1.1    orain-TM1701

# The following lines are desirable for IPv6 capable ho
::1        ip6-localhost ip6-loopback
fe00::0    ip6-localnet
ff00::0    ip6-mcastprefix
ff02::1    ip6-allnodes
ff02::2    ip6-allrouters

12.0.0.2    myhsapp.com
```

La modification a bien été prise en compte :



3)

Nous allons à présent installer Traefik. Dans la stack liée à notre galerie, on va modifier le docker-compose.yml en conséquence.

Voici les lignes que nous ajoutons au docker-compose.yml :

```
reverse-proxy:
  image: traefik # The official Traefik docker image
  command: --web --docker --docker.swarmmode --docker.domain=myhbsapp.com --docker.watch --
  ports:
    - "80:80" # The HTTP port
    - "8080:8080" # The Web UI (enabled by --api)
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock # So that Traefik can listen to the Docker
```

Je me suis rendu compte que mon service myhbsapp_puppeter n'était pas fonctionnel pour les TP précédents. Je dois rajouter un volume supplémentaire pour que mon fichier puppeteer.js soit bien pris en compte :

```
puppeter:
  image: 127.0.0.1:5000/puppeter
  build: ./puppeter
  ports:
    - "4000:4000"
  volumes:
    - "./puppeter/screenshots:/screenshots"
    - "./puppeter:/puppeter"
```

L'ajout de la ligne « - ./puppeter:/puppeter » a été effectué dans les 2 fichiers : docker-compose.yml et stack-compose.yml.

Nous devons également modifier notre fichier puppeteer.js de façon à ce que l'on soit attentif au nom de domaine dans le navigateur. Ce n'est plus l'adresse IP de notre manager qui est indiqué. J'ai également enlevé la suppression d'album car cela engendrait un soucis de synchronisation. Dans ce client émulé je navigue sur le site et j'ajoute des albums.

On arrête notre stack « myhbsapp », on met à jour notre docker-compose puis on push et enfin on deploy :

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker-compose up -d
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on the current node.

To deploy your application across the swarm, use the bundle feature of the Docker experimental build.

More info:
https://docs.docker.com/compose/bundles

Creating network "myhbsapp_default" with the default driver
Creating myhbsapp_minio_1
Creating myhbsapp_reverse-proxy_1
Creating myhbsapp_web_1
Creating myhbsapp_redis_1
Creating myhbsapp_puppeter_1
```

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker-compose push
Pushing web (127.0.0.1:5000/myhbsapp:latest)...
The push refers to repository [127.0.0.1:5000/myhbsapp]
399ae2021616: Pushed
1a531ce01f01: Pushed
19f9d75329f8: Pushed
ffbb143619acd: Pushed
```

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker stack deploy --compose-file stack-compose.yml myhbsapp
Ignoring unsupported options: build

Creating network myhbsapp_net
Creating service myhbsapp_web
Creating service myhbsapp_redis
Creating service myhbsapp_reverse-proxy
Creating service myhbsapp_puppeteer
Creating service myhbsapp_minio
```

On peut accéder au Traefik via le port 8080

Pour le moment il n'y a ni frontend ni backend. Nous allons ajouter des services que nous allons gérer avec Traefik.

Dans stack-compose.yml, on ajoute les labels au service lié à notre app :

```
services:
  web:
    image: 127.0.0.1:5000/myhbsapp
    deploy:
      labels:
        - traefik.enable=true
        - traefik.port=3000
        - traefik.frontend.rule=Host:myhbsapp.com
        - traefik.docker.network=myhbsapp_net
        - traefik.backend=web
```

The screenshot shows the Traefik dashboard at `myhbsapp.com:8080/dashboard/`. The interface includes a search bar, tabs for `PROVIDERS` and `HEALTH`, and a version indicator `V1.7.8 / MAROILLES` with a `DOCUMENTATION` link.

Under the `docker` provider, there are two sections:

- 1 FRONTENDS**: Shows a configuration for `frontend-Host-myhbsapp-com-0`. The `Main` tab displays:
 - Route Rule**: `Host:myhbsapp.com`
 - Entry Points**: `http`
 - Backend**: `backend-web`
- 1 BACKENDS**: Shows a configuration for `backend-web`. The `Main` tab displays:

Server	Weight
<code>http://10.0.4.3:3000</code>	1

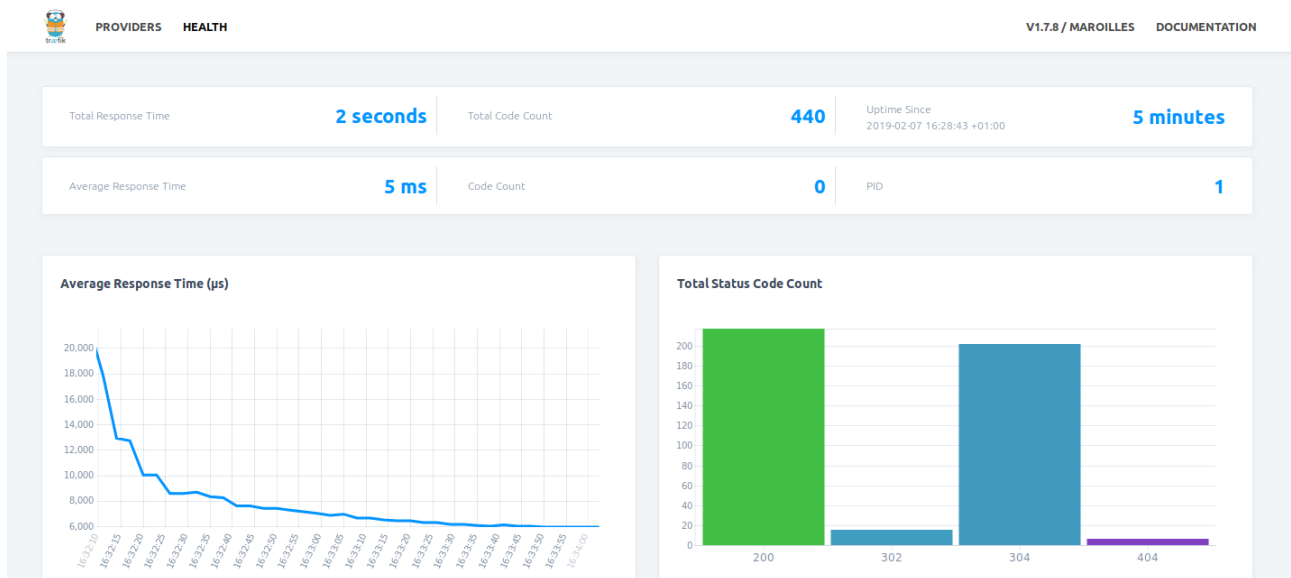
4)

Pour que l'application fonctionne correctement j'avais besoin de lancer 2 fois ma commande :
« `sudo docker stack deploy --compose-file stack-compose.yml myhbsapp` »

On se rend dans l'onglet Health.

Même en ayant fait quelques modifications pour mon service `myhbsapp_puppeteer`. Aucun impact n'était observé sur le graphique lorsque je lançais plusieurs clients.

En dehors de ma VM j'ai donc lancé chacun leur tour une dizaine de clients émulsés :



J'ai un temps de réponse moyen de 5ms. Notre service respecte donc bien le SLA (inférieur à 200ms). Ce temps de réponse est faible mais ce n'est pas étonnant compte tenu des performances de mon PC.

Partie II – Incorporation à la stack de monitoring

Nous souhaitons à présent intégrer toutes les données précédentes dans Grafana.

1)

Je ne suis pas parvenu à intégrer traefik au swarmprom, j'ai par conséquent repris mon ancienne stack : monitor.

J'ai tout d'abord changé le port de mon service cadvisor (8082:8080) puisqu'il avait les mêmes que le service traefik.

Endpoint	State	Labels	Last Scrape	Error
cadvisor (1/1 up)				
http://12.0.0.2:8082/metrics	UP	instance="12.0.0.2:8082"	12.26s ago	
prometheus (1/1 up)				
http://12.0.0.2:9090/metrics	UP	instance="12.0.0.2:9090"	10.447s ago	

Dans notre fichier de configuration `/tmp/prometheus.yml` on ajoute les lignes suivantes de façon à intégrer traefik dans prometheus.

```
- job_name: 'traefik'

static_configs:
  - targets: ['12.0.0.2:8080']
```

A chaque redémarrage de la VM il est nécessaire de remplir à nouveau ce fichier de configuration.

On supprime la stack monitor et on la redémarre :

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker stack deploy -c monitor-stack.yml monitor
Ignoring unsupported options: links, restart

Creating network monitor_net-monitoring
Creating service monitor_prom
Creating service monitor_grafana
Creating service monitor_cadvisor
```

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker stack services monitor
ID                NAME                MODE                REPLICAS            IMAGE                PORTS
7ee45j9qd2gz      monitor_grafana      replicated          1/1                 grafana/grafana:latest  *:3002->3000/tcp
lrrdi70yo193      monitor_cadvisor     replicated          1/1                 google/cadvisor:v0.29.0  *:8082->8080/tcp
zpxjyswk189y      monitor_prom         replicated          1/1                 quay.io/prometheus/prometheus:v2.0.0  *:9090->9090/tcp
```

On se rend à l'url : « myhbsapp.com:9090/targets

Prometheus Alerts Graph Status ▾ Help			
Targets			
cadvisor (1/1 up)			
Endpoint	State	Labels	Last Scrape
http://12.0.0.2:8082/metrics	UP	instance="12.0.0.2:8082"	2.241s ago
prometheus (1/1 up)			
Endpoint	State	Labels	Last Scrape
http://12.0.0.2:9090/metrics	UP	instance="12.0.0.2:9090"	434ms ago
traefik (1/1 up)			
Endpoint	State	Labels	Last Scrape
http://12.0.0.2:8080/metrics	UP	instance="12.0.0.2:8080"	14.181s ago

Traefik a bien été ajouté à Prometheus. Nous devons à présent configurer grafana pour faire en sorte que ces métriques soient les données sources de grafana.

Settings

Dashboards

Settings

NamePrometheusDefault

HTTP

URLhttp://12.0.0.2:9090

AccessServer (Default)Help

Whitelisted CookiesAdd Name

Auth

Basic AuthWith Credentials

TLS Client AuthWith CA Cert

Skip TLS Verify

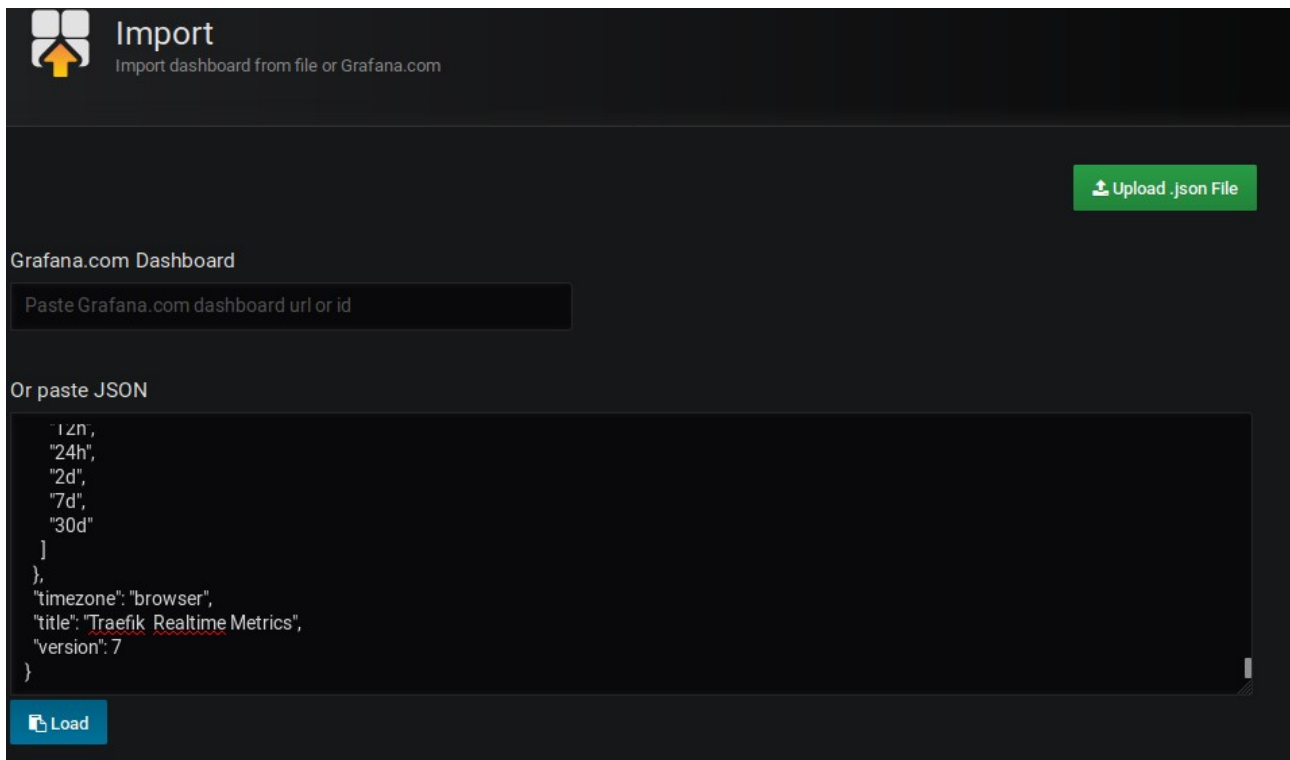
Scrape interval15s

Query timeout60s

HTTP MethodGET

✓ Data source is working

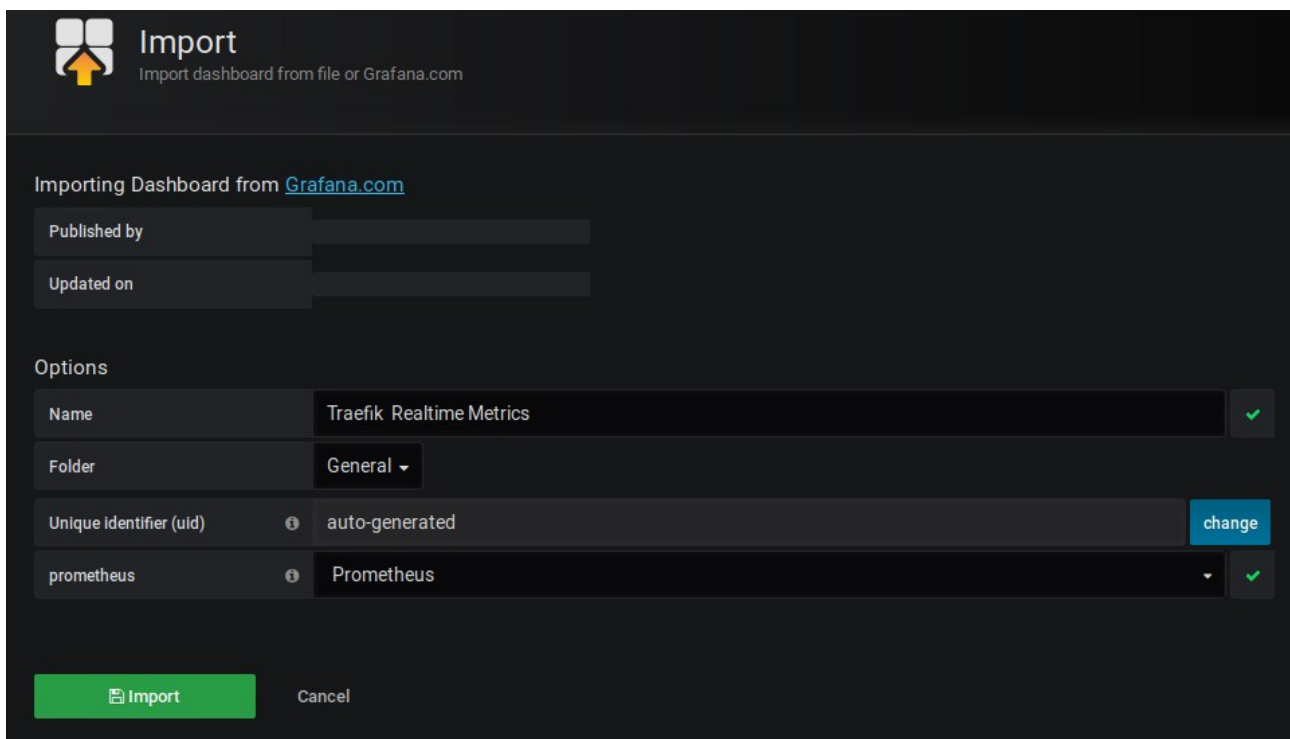
On importe un dashboard depuis grafana. Pour cela on récupère le fichier json correspondant au dashboard affichant les métriques Traefik en temps réel sur le site grafana.com :



The image shows the Grafana 'Import' interface. At the top, there's a header with the Grafana logo and the text 'Import' and 'Import dashboard from file or Grafana.com'. Below this, there's a green button labeled 'Upload .json File'. The main section is titled 'Grafana.com Dashboard' and contains a text input field with the placeholder 'Paste Grafana.com dashboard url or id'. Below this, there's a section titled 'Or paste JSON' with a large text area containing a JSON snippet. At the bottom left, there's a blue button labeled 'Load'.

```
{
  "id": "1234",
  "uid": "1234",
  "type": "dashboard",
  "name": "Traefik Realtime Metrics",
  "slug": "traefik-realtime-metrics",
  "tags": [
    "traefik",
    "realtime",
    "metrics"
  ],
  "timezone": "browser",
  "version": 7
}
```

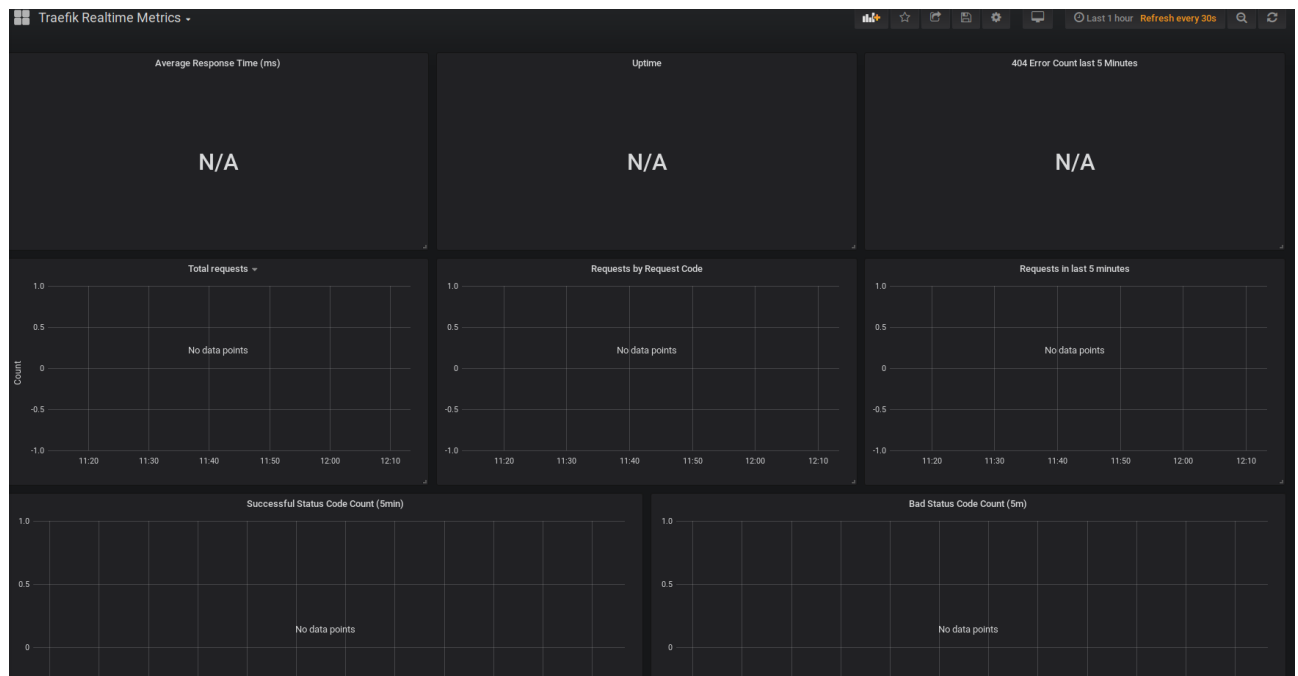
Puis on met prometheus en temps que sources de données pour ce dashboard :



The image shows the Grafana 'Import' interface for configuring the dashboard. At the top, there's a header with the Grafana logo and the text 'Import' and 'Import dashboard from file or Grafana.com'. Below this, there's a section titled 'Importing Dashboard from Grafana.com'. This section contains two input fields: 'Published by' and 'Updated on'. Below these, there's a section titled 'Options'. This section contains four rows of configuration options: 'Name' (Traefik Realtime Metrics), 'Folder' (General), 'Unique Identifier (uid)' (auto-generated), and 'prometheus' (Prometheus). Each row has a green checkmark icon on the right. At the bottom left, there's a green button labeled 'Import' and a 'Cancel' button.

Option	Value	Action
Name	Traefik Realtime Metrics	✓
Folder	General	▼
Unique Identifier (uid)	auto-generated	change
prometheus	Prometheus	✓

On se rend donc sur le dashboard que nous venons de créer :



Il semblerait que l'outil soit mal configuré.
On clique sur « average response time » et on edit.

La métrique était la suivante :

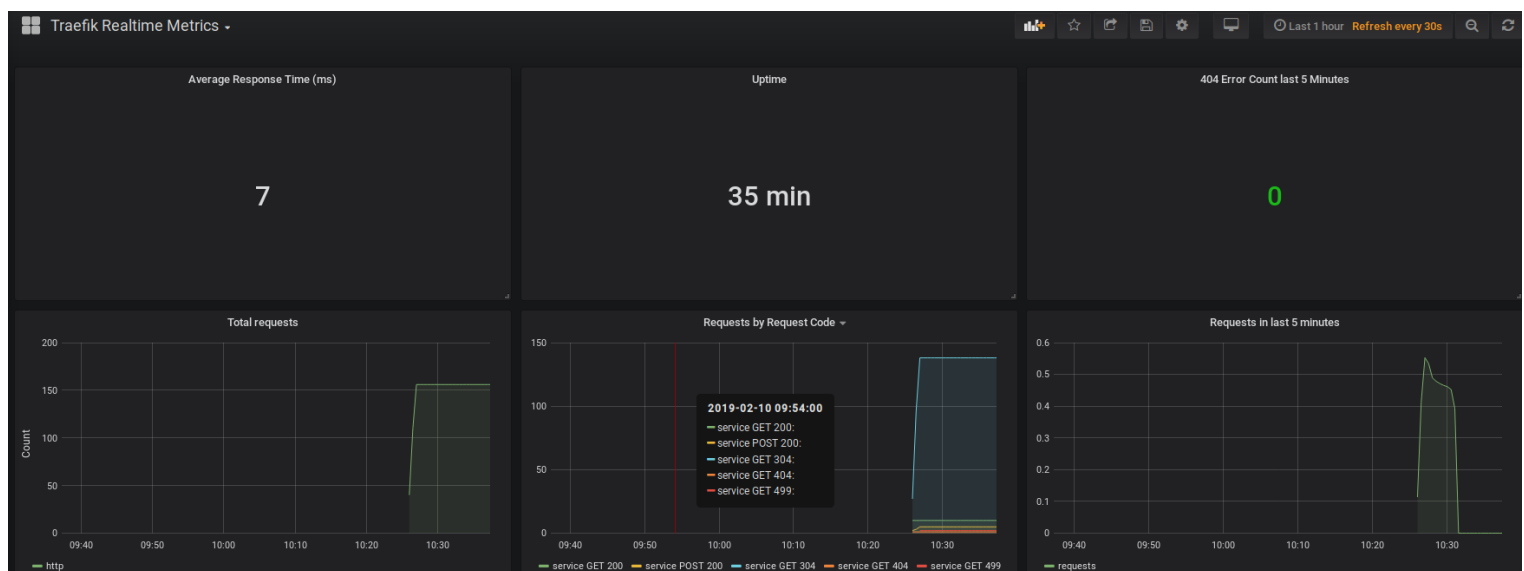
$\text{sum}(\text{traefik_request_duration_seconds_sum}) / \text{sum}(\text{traefik_requests_total}) * 1000$

Les deux interfaces prometheus (12.0.0.2/graph) et grafana ne trouvaient aucun point. Ils suggéraient d'ajouter « entrypoint » ou « backend » après traefik dans la commande.

```
sum(traefik_entrypoint_request_duration_seconds_sum) / sum(traefik_entrypoint_requests_total) * 1000
```

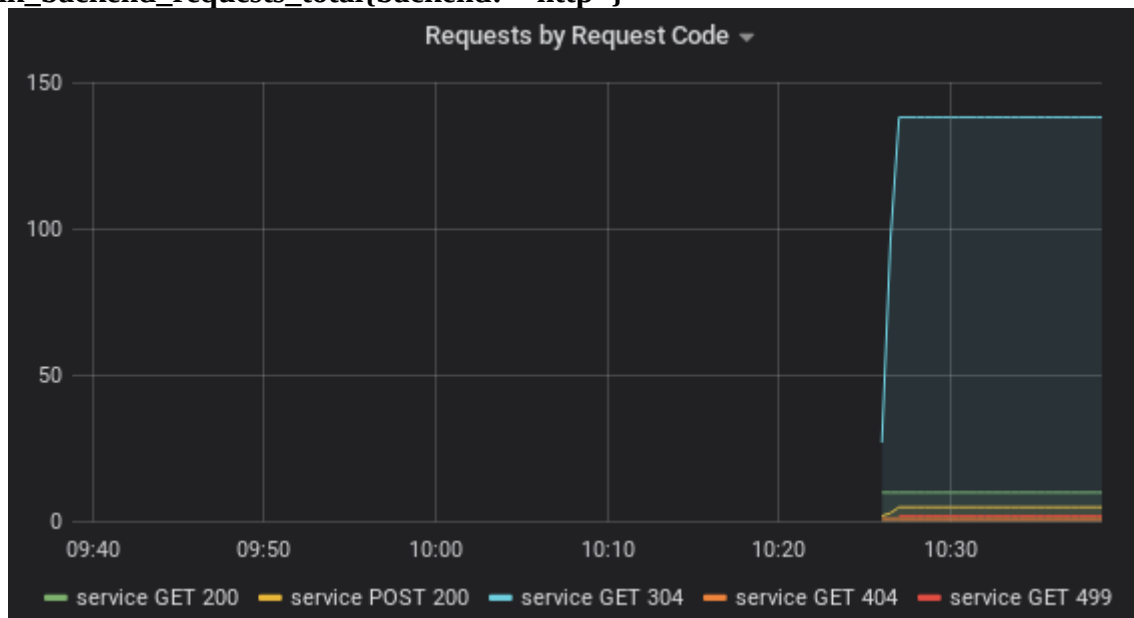
J'ai donc fait de même pour tous les autres graphs.

On retrouve les mêmes valeurs que dans traefik. Le temps de réponse est de 7ms au moment où j'ai pris cette capture, le nombre requêtes est de 153. Il n'y a pas d'erreur 404.

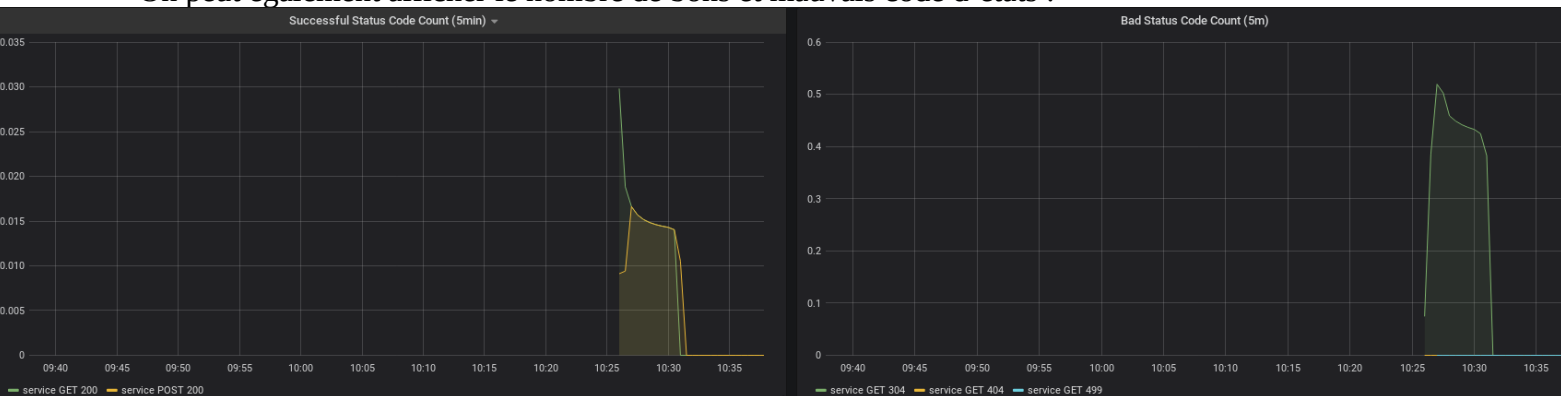


Dans mon service grafana, j'ai rajouté la persistance, chose que je n'avais pas fait lors des précédents TP. L'identifiant est « admin » et le mot de passe « anthocloud ».

Pour afficher le nombre total de code d'état j'ai utilisé la métrique suivante :
traefik_backend_requests_total{backend!~"http"}



On peut également afficher le nombre de bons et mauvais code d'états :



3) Précédemment j'ai fait état d'un soucis lié au redémarrage de la machine virtuelle pour le fichier /tmp/prometheus.yml. Pour remédier à cela dans le répertoire myhbsapp j'ai créé un dossier prometheus dans lequel nous avons le fichier monitor-stack.yml et le fichier de configuration prometheus.yml :

```
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds by default
  evaluation_interval: 15s # Evaluate rules every 15 seconds by default

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

scrape_configs:
  # The job name is added as a label `job=` to any timeseries scraped from this config.
  - job_name: 'prometheus'
    static_configs:
      - targets: ['12.0.0.2:9090']

  - job_name: 'cadvisor'
    static_configs:
      - targets: ['12.0.0.2:8082']

  - job_name: 'traefik'
    static_configs:
      - targets: ['12.0.0.2:8080']
```


Dans le fichier ci-dessus nous avons précisé un rule_file : « traefik.rules.yml ». Dans ce dernier nous allons configurer une alerte lorsque le temps de réponse moyen dépasse notre SLA. Pour la démonstration nous allons fixer le temps de réponse à 5.6ms.

```
groups:
- name: traefik.rules.yml
  rules:
  - alert: average_response_time
    expr: sum(traefik_entrypoint_request_duration_seconds_sum) / sum(traefik_entrypoint_requests_total) * 1000 > 5.6
    for: 1m
    labels:
      severity: warning
    annotations:
      description: average_response_time is above your SLA
      summary: average_response_time
```

Dans le fichier traefik.rules.yml ci-dessus, on indique que l'on souhaite recevoir une alerte lorsque le temps de réponse est supérieur à 5.6ms pendant au moins 1min.

Dans la stack swarmprom il aurait été très facile de tester cette alerte dans la mesure où tous les outils étaient configurés notamment ceux étant liés aux alertes. Il aurait fallu ajouter le code ci-dessus dans le fichier swarm_task.rules.yml du dossier /prometheus/rules du swarmprom par exemple.

Dans notre cas, il faudrait installer AlertManager et le configurer pour nous envoyer des alertes.

Partie III – Chargez

A présent, nous allons observer comment notre système se comporte lorsqu'on lui coupe des membres.

1)

On commence par ajouter le service visualizer.

Dans la stack myhsapp on ajoute le service visualizer :

```
viz:
  image: dockersamples/visualizer
  volumes:
    - "/var/run/docker.sock:/var/run/docker.sock"
  ports:
    - "8084:8080"
```

On relance la stack :

```
vagrant@manager:/vagrant/myhsapp$ sudo docker stack deploy -c stack-compose.yml myhsapp
Ignoring unsupported options: build

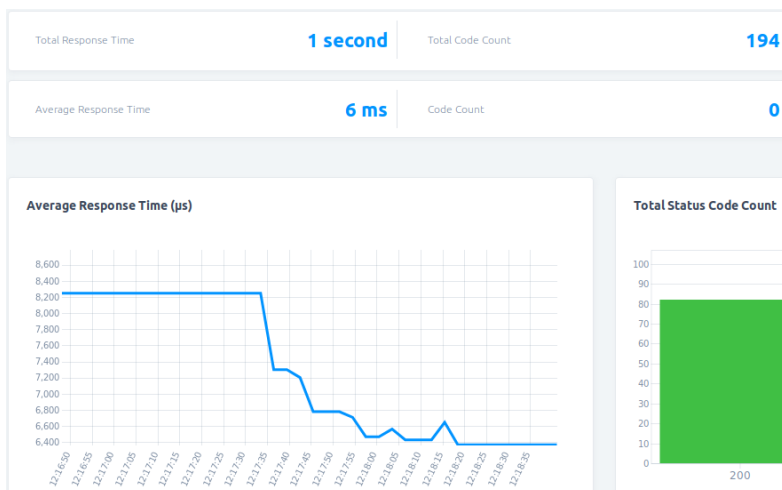
Creating network myhsapp_net
Creating network myhsapp_default
Creating service myhsapp_redis
Creating service myhsapp_reverse-proxy
Creating service myhsapp_puppeteer
Creating service myhsapp_viz
Creating service myhsapp_minio
^[[5~Creating service myhsapp_web
vagrant@manager:/vagrant/myhsapp$ sudo docker stack services myhsapp
ID                NAME                MODE                REPLICAS            IMAGE                PORTS
3w2uzfwyzjzc      myhsapp_puppeteer    replicated           0/1                 127.0.0.1:5000/puppeteer:latest
d04zq7ihpyao      myhsapp_minio        replicated           1/1                 minio/minio:latest  *:9000->9000/tcp
glf0sa4p9cqo      myhsapp_viz          replicated           1/1                 dockersamples/visualizer:latest  *:8084->8080/tcp
nn7wq9mhu83z      myhsapp_redis        replicated           1/1                 redis:alpine
t899zjfw4nq       myhsapp_reverse-proxy replicated           1/1                 traefik:latest      *:80->80/tcp, *:8080->8080/tcp
th1p1tfm67e3      myhsapp_web          replicated           1/1                 127.0.0.1:5000/myhsapp:latest     *:3000->3000/tcp
```

Dans le navigateur, on se rend à l'adresse 12.0.0.2:8084 :



On remarque que certains services tels que `myhbsapp_reverse-proxy`, `myhbsapp_web`, et `myhbsapp_minio` sont bien distribués sur les workers 1 et 2.

Pour notre SLA, on va fixer le temps de réponse à 7ms. Pour le moment, le temps de réponse est de 6ms après avoir lancé quelques clients.



```
orain@orain-TM1701:~/Documents/td9-antho35/td9/3VM/myhbsapp/puppeteer$ node puppeteer.js  
timeout completed  
orain@orain-TM1701:~/Documents/td9-antho35/td9/3VM/myhbsapp/puppeteer$ node puppeteer.js  
timeout completed  
orain@orain-TM1701:~/Documents/td9-antho35/td9/3VM/myhbsapp/puppeteer$ node puppeteer.js  
timeout completed  
orain@orain-TM1701:~/Documents/td9-antho35/td9/3VM/myhbsapp/puppeteer$ node puppeteer.js  
timeout completed  
orain@orain-TM1701:~/Documents/td9-antho35/td9/3VM/myhbsapp/puppeteer$ node puppeteer.js  
timeout completed
```

On éteint le worker-1 et on observe ce qu'il se passe.

```
orain@orain-TM1701:~/Documents/td9-antho35/td9/3VM/myhbsapp$ vagrant halt worker-1  
==> worker-1: Attempting graceful shutdown of VM...
```

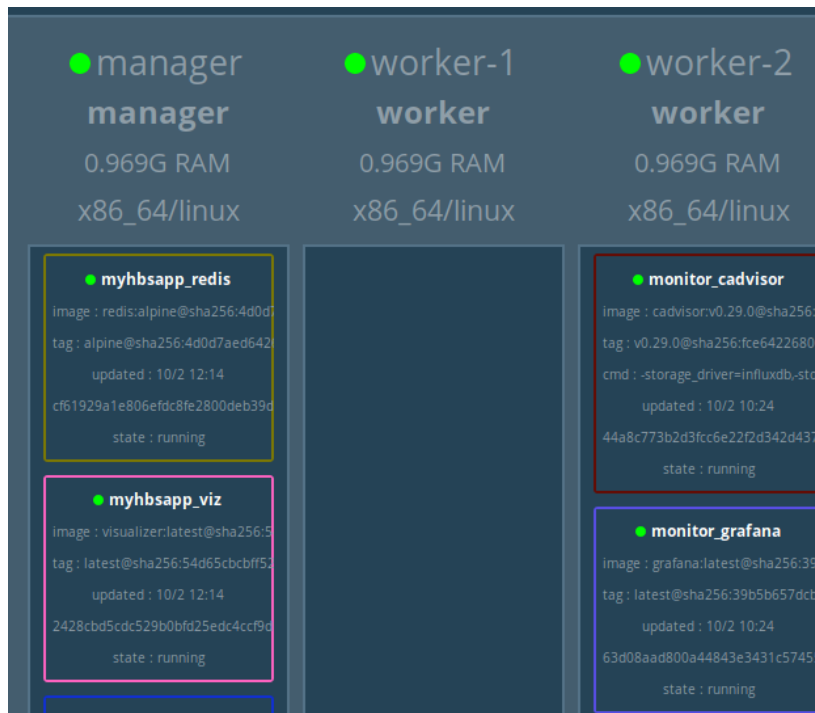
L'ensemble des services présents dans le worker-1 se sont déplacés dans le worker-2. La stack s'est adaptée.

Lorsque l'on lance plusieurs clients, on remarque qu'on a de plus en plus de mal à respecter notre SLA, dû à notre nouvelle configuration.



2)

Lorsque l'on redémarre le worker-1, les services restent sur le worker-2

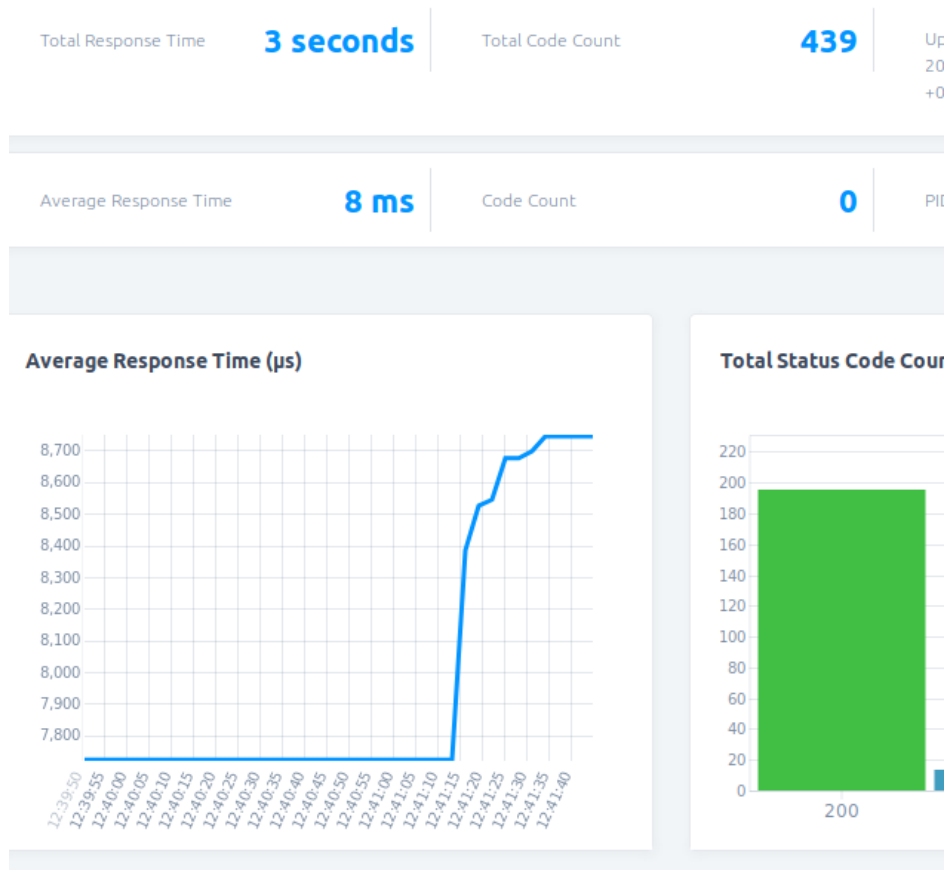


On va éteindre le worker-2 :

```
orain@orain-TM1701:~/Documents/td9-antho35/td9/3VM/myhbsapp$ vagrant halt worker-2  
==> worker-2: Attempting graceful shutdown of VM...
```



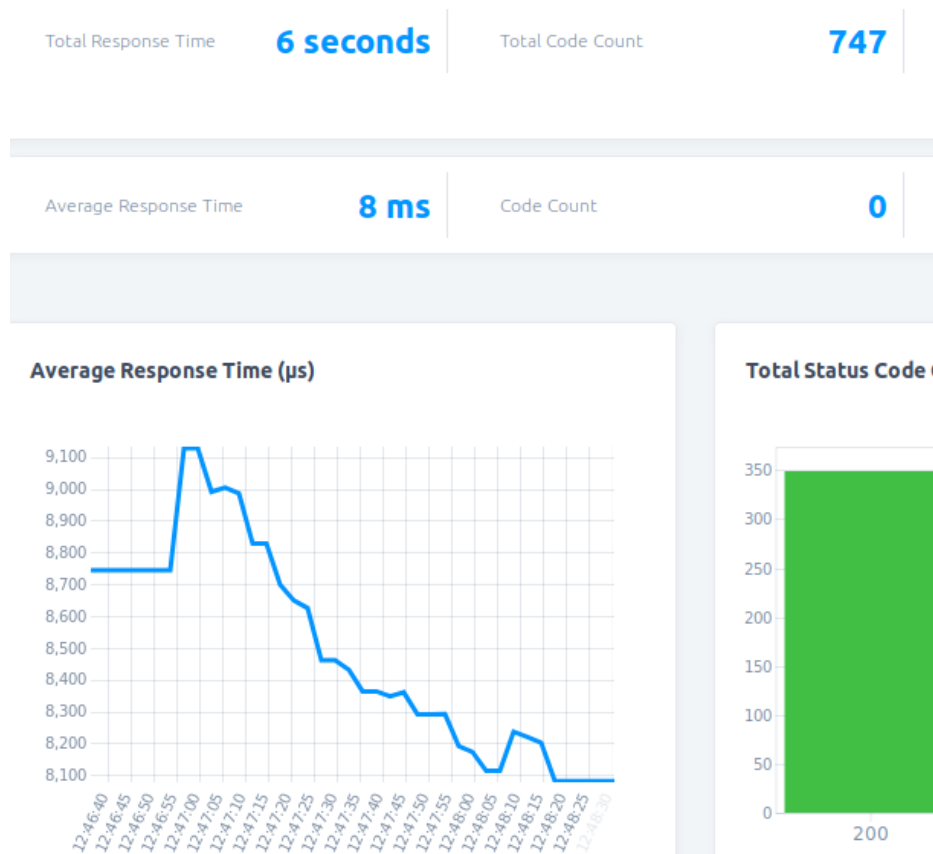
Les services du worker-2 sont bien repassés sur le worker-1. Comme précédemment, notre SLA est vite dépassé en fonctionnant avec un seul worker.



Nous allons couper les 2 workers à présent.
Tous les services sont passés dans le manager.

● manager	● worker-1	● worker-2
manager	worker	worker
0.969G RAM	0.969G RAM	0.969G RAM
x86_64/linux	x86_64/linux	x86_64/linux
<div><div>● monitor_prom</div><div>image : prometheus:v2.0.0@sha256:15008611fb4a78b8995db022d8831</div><div>tag : v2.0.0@sha256:53afe934a8d4</div><div>cmd : --config.file=/etc/prometheus</div><div>updated : 10/2 12:44</div><div>state : running</div></div>		
<div><div>● myhbsapp_minio</div><div>image : minio:latest@sha256:de2ad9c6b68ecd7c30b084f19dd3</div><div>tag : latest@sha256:7439652be65e</div><div>cmd : minio.server/data</div><div>updated : 10/2 12:44</div><div>state : running</div></div>		
<div><div>● monitor_grafana</div><div>image : grafana:latest@sha256:3d8357e60907a56286423bf2f9e91</div><div>tag : latest@sha256:39b5b657dcba</div><div>updated : 10/2 12:44</div><div>state : running</div></div>		

Avec uniquement le manager en place, le fait de lancer plusieurs clients fait augmenter le temps de réponse brièvement puis ne cesse de diminuer par la suite.
Il semblerait que la configuration avec le manager seul soit meilleure que celle avec un worker.



On redémarre les 2 workers : les services restent sur le manager.
On coupe le manager.
L'application web ainsi que tous les autres services ne sont plus accessibles.

3)
Nous reconfigurons notre Swarm de façon à ce que tous les nœuds soient des managers.

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker node ls
```

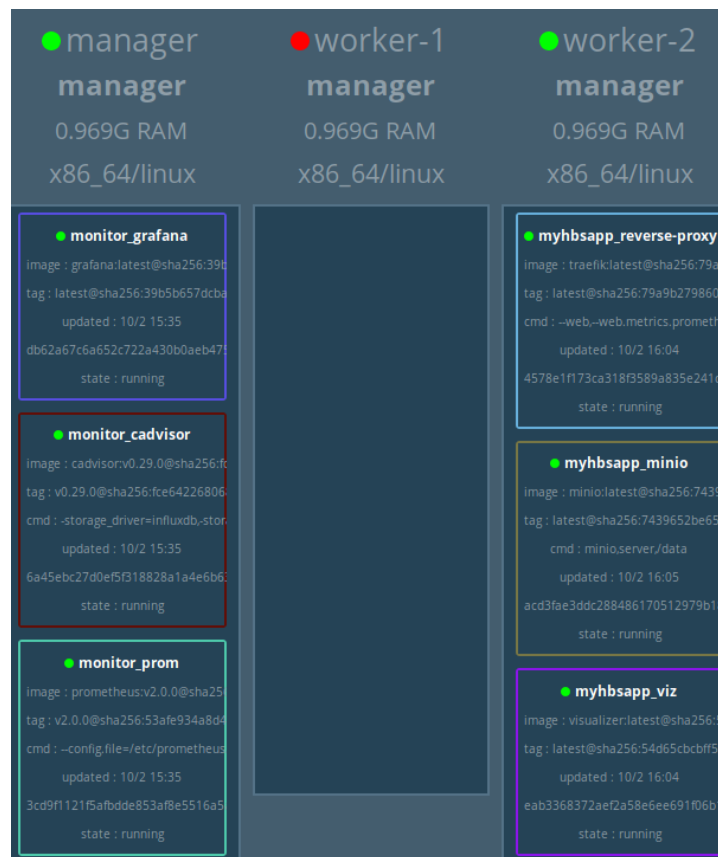
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
d50qk5xspyup10nidn8xfjg6x *	manager	Ready	Active	Leader	18.09.1
vlzn9jigp8qoip0oqkr0vnqs4	worker-1	Ready	Active		18.09.1
yj2dbyj0um8k7g1jxwrurmctz	worker-2	Ready	Active		18.09.1

On passe le worker-1 et worker-2 en manager :

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker node promote worker-1 worker-2
```

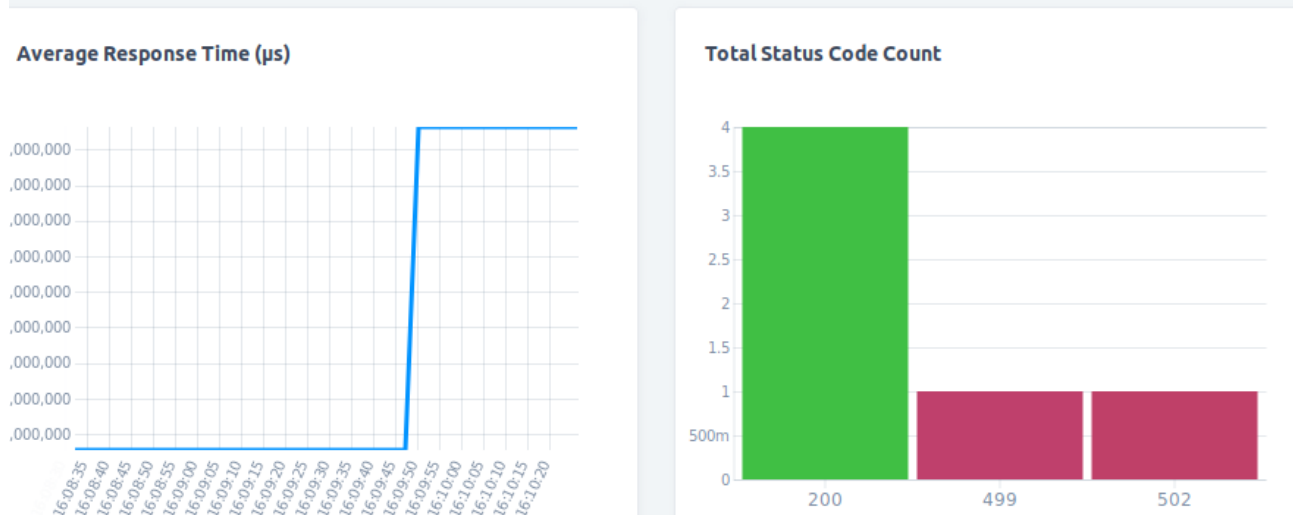
Node worker-1 promoted to a manager in the swarm.
Node worker-2 promoted to a manager in the swarm.

On éteint le worker-1 :
 Le worker 2 prend en charge tous les services du worker-1 (redis, reverse-proxy et minio).



Toutefois, l'application n'est plus fonctionnelle.

Total Response Time	2 minutes	Total Code Count	6	Uptime Since 2019-02-10 16:04:34 +01:00	5 minutes
Average Response Time	29263 ms	Code Count	0	PID	1



On éteint le second worker :

● manager	● worker-1	● worker-2
manager	manager	manager
0.969G RAM	0.969G RAM	0.969G RAM
x86_64/linux	x86_64/linux	x86_64/linux

Cela provoque l'arrêt du premier manager également.

Lorsque tous les nœuds sont en mode manager, il faut qu'ils soient tous actifs pour que l'application soit fonctionnelle.

4)

Je n'ai pas eu le temps de finir la dernière question. En effet, j'ai préféré m'attarder sur le fait de fournir un service puppeteer fonctionnel et donc ça m'a pris du temps.

J'ai apporté les modifications suivantes à mon service puppeteer :

```
puppeteer:
  image: alekzonder/puppeteer:latest
  build: ./puppeteer
  volumes:
    - "./puppeteer/screenshots:/screenshots"
    - "./puppeteer:/app/index.js"
```

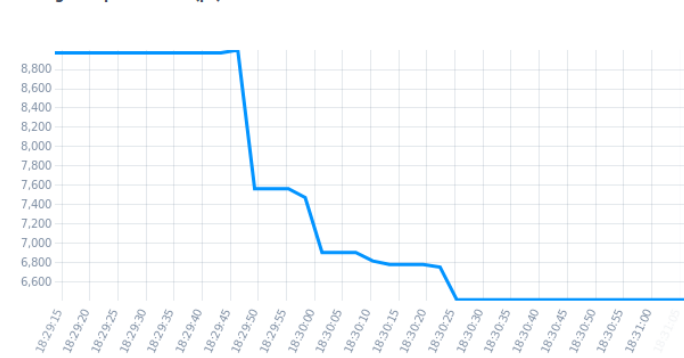
```
vagrant@manager:/vagrant/myhsapp$ sudo docker stack services myhsapp
ID                NAME                MODE                REPLICAS                IMAGE                PORTS
0s5tcmqqlay3     myhsapp_viz         replicated          1/1                     dockersamples/visualizer:latest
pqxmvm0nz9nro    myhsapp_reverse-proxy replicated          1/1                     traefik:latest
>8080/tcp
s7lvk4u958ni     myhsapp_redis       replicated          1/1                     redis:alpine
tinimwvk2h8j     myhsapp_puppeteer   replicated          1/1                     alekzonder/puppeteer:latest
u2fonhx1n5jh     myhsapp_minio        replicated          1/1                     minio/minio:latest
xbctxusvc6a1     myhsapp_web          replicated          1/1                     127.0.0.1:5000/myhsapp:latest
*:8084->8080/tcp
*:80->80/tcp, *:8080-
```

J'ai désormais un service d'émulation de client fonctionnel dans ma VM, j'ai donc pu lancer 3 clients simultanément :

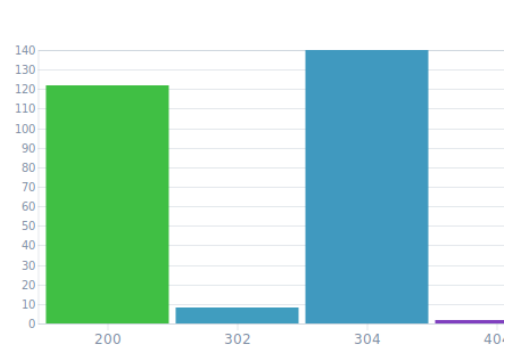
J'ai un temps de réponse moyen de 3ms.

```
vagrant@manager:/vagrant/myhsapp$ sudo docker service scale myhsapp_puppeteer=3
myhsapp_puppeteer scaled to 3
overall progress: 3 out of 3 tasks
1/3: running [=====>]
2/3: running [=====>]
3/3: running [=====>]
verify: Service converged
Average Response Time 6 ms Code Count 0 PID
```

Average Response Time (µs)



Total Status Code Count



Conclusion

Au cours de cet ultime TP j'ai pu notamment corriger quelques petites imperfections dans mon swarm (persistance de grafana, configuration de prometheus et service puppeteer). Ce TP m'a également permis de recueillir des données sur l'état de santé de mon application. L'outil traefik a ainsi été mis en place et ajouté à la stack monitoring. Cela a permis de voir comment se comportait notre application face à différentes configuration de pannes.

Ce module a été très formateur dans le sens où il fallait sans cesse faire face aux imprévus et corriger les bugs survenants. Il nous a montré l'ensemble des outils permettant la conteneurisation d'une application et de son infrastructure. On a aussi pu garantir la pérennité de cette dernière par le biais d'outils de monitoring.