TD 8

Introduction

Après avoir créé de faux clients afin de simuler un trafic sur notre application, nous avons commencé par créer une instance Prometheus collectant des données sur notre Swarm et les enregistrant sous forme de séries temporelles.

Nous allons poursuivre le monitoring de notre infrastructure en acquérissant plus de métriques par l'intermédiaire de nouveaux services. On va incorporer ces derniers à Prometheus pour les afficher sommairement. Ils seront représentés par la suite par des graphiques à l'aide de Grafana.

Partie I – Retour sur notre galerie

1)

Nous cherchons ici à savoir quels sont les goulots d'étranglement de notre application. Cela peut être lié à l'espace disque (tous les programmes installés sur les VM), ou encore lié au réseau : dû à la présence de multiples conteneurs. La fréquence d'accès au disque est notamment à prendre en compte.

En ayant créé des clients émulés, on est en mesure de créer plusieurs utilisateurs se connectant de façon concurrentes à notre application. Pour un service exécutant nos utilisateurs, on peut augmenter son nombre de répliques grâce à la commande : *« docker service scale service_name=nb_repliques»*.

Pour notre application, ce sont les téléchargements de photos qui va prendre le plus de place. Une solution consiste à stocker les photos les plus demandées dans un cache. Toute photo qui est regardé passe dans le cache ; les photos moins utilisés sont poussés en fin de cache puis éjectés.

Partie II – Collecte et affichage de données

Les métriques fournies par Docker avec Prometheus sont nombreuses mais pas toutes pertinentes. Nous allons voir quelques outils de collecte de métriques s'interfaçant avec Docker.

On commence par créer une nouvelle stack nommée « monitor ».
Pour cela, en premier lieu on crée un nouveau réseau pour notre stack :

```
vagrant@manager:/$ sudo docker network create --driver overlay --attachable net-monitoring
xb95n91lu4t242jp1bzrn1vea
```

On crée d'abord un fichier monitor-stack.yml dans lequel on va mettre cAdvisor et Prometheus :

monitor-stack.yml docker-compose.yml

```yaml
version: '3'


networks:
  net-monitoring:

services:
  cadvisor:
    image: google/cadvisor:v0.29.0
    links:
      - influxdb:influxsrv
    command: -storage_driver=influxdb -storage_driver_db=cadvisor -storage_dri
    restart: always
    ports:
      - "8080:8080"
    networks:
      - net-monitoring
    volumes:
      - /:/rootfs:ro
      - /var/run:/var/run:rw
      - /sys:/sys:ro
      - /var/lib/docker/:/var/lib/docker:ro

  prom:
    image: quay.io/prometheus/prometheus:v2.0.0
    volumes:
     - ../../tmp/prometheus.yml:/etc/prometheus/prometheus.yml
    command: "--config.file=/etc/prometheus/prometheus.yml --storage.tsdb.path
    ports:
     - "9090:9090"
    networks:
      - net-monitoring
```

Il est en suite possible de deploy la stack :

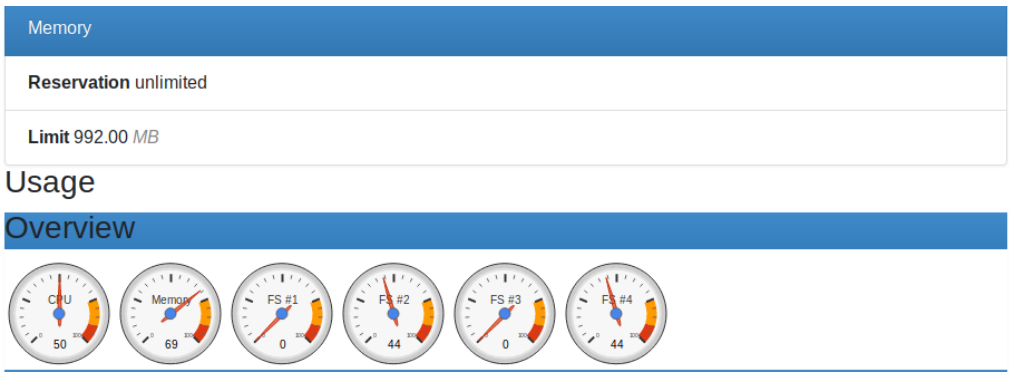
```
vagrant@manager:/vagrant/myhbsapp$ sudo docker stack deploy --compose-file monitor-stack.yml monitor
Ignoring unsupported options: links, restart

Creating service monitor_cadvisor
Creating service monitor_prom
```

On a donc 2 stack : une pour notre application et une autre pour le monitoring.

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker stack services myhbsapp
ID                  NAME                MODE                REPLICAS            IMAGE                                PORTS
0iiuunbd4bx9        myhbsapp_minio      replicated          1/1                 minio/minio:latest                   *:9000->9000/tcp
3qncob5bv7v7        myhbsapp_redis      replicated          1/1                 redis:alpine
l7f3ebqr4n5v        myhbsapp_web        replicated          1/1                 127.0.0.1:5000/myhbsapp:latest       *:3000->3000/tcp
yqdrinroxgsj        myhbsapp_puppeteer  replicated          1/1                 127.0.0.1:5000/puppeteer:latest      *:4000->4000/tcp
```

```
vagrant@manager:/vagrant/myhbsapp$ sudo docker stack services monitor
ID                  NAME                MODE                REPLICAS            IMAGE                                PORTS
3vzcf9n5y57e        monitor_prom        replicated          1/1                 quay.io/prometheus/prometheus:v2.0.0 *:9090->9090/tcp
ahmehbf5p39w        monitor_cadvisor    replicated          1/1                 google/cadvisor:v0.29.0              *:8080->8080/tcp
```

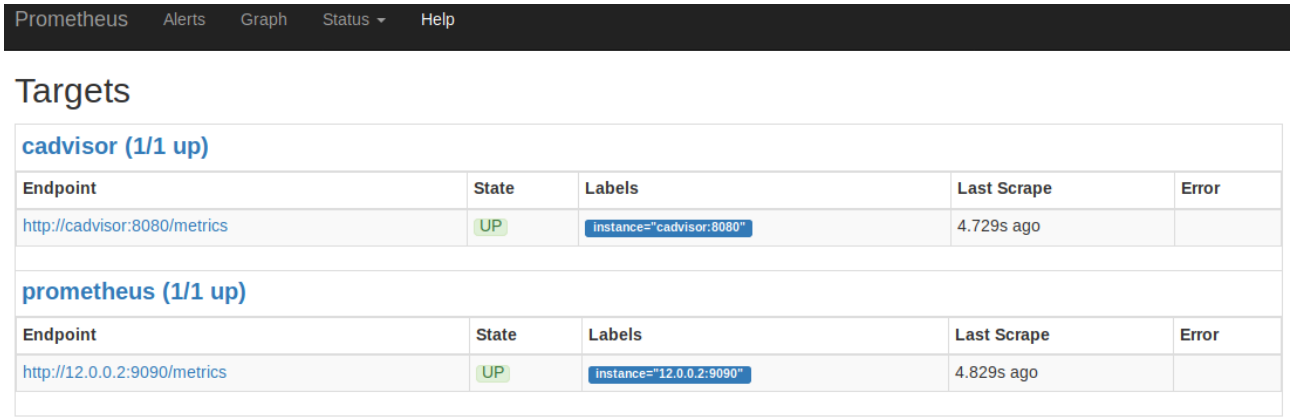
On peut tester cAdvisor à l'adresse 12.0.0.2:8080 :



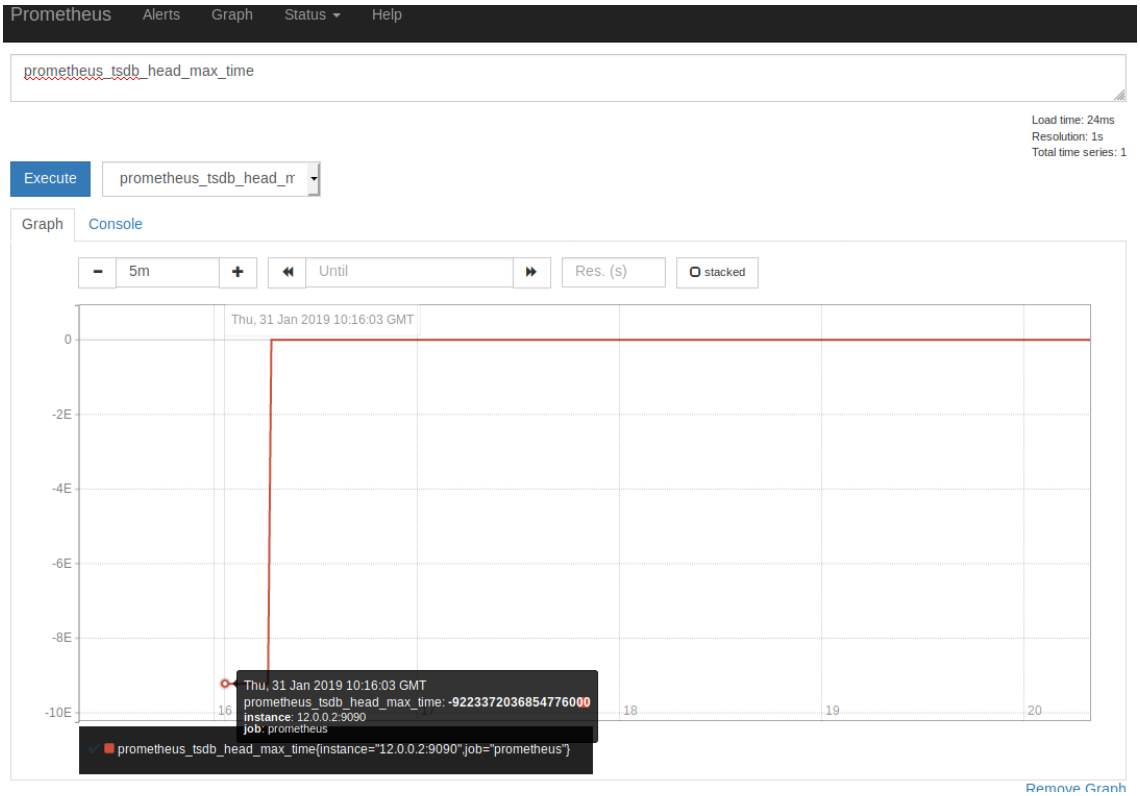
On modifie notre fichier de configuration prometheus.yml afin d'ajouter cadvisor en lieu et place du docker :

```
- job_name: 'prometheus'

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

  static_configs:
    - targets: ['12.0.0.2:9090']

- job_name: 'cadvisor'
      # metrics_path defaults to '/metrics'
      # scheme defaults to 'http'.

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

On peut alors se rendre sur l'adresse : 12.0.0.2:9090/targets



Il est possible d'afficher quelques graphiques liés aux métriques en se rendant dans l'onglet « Graph ».



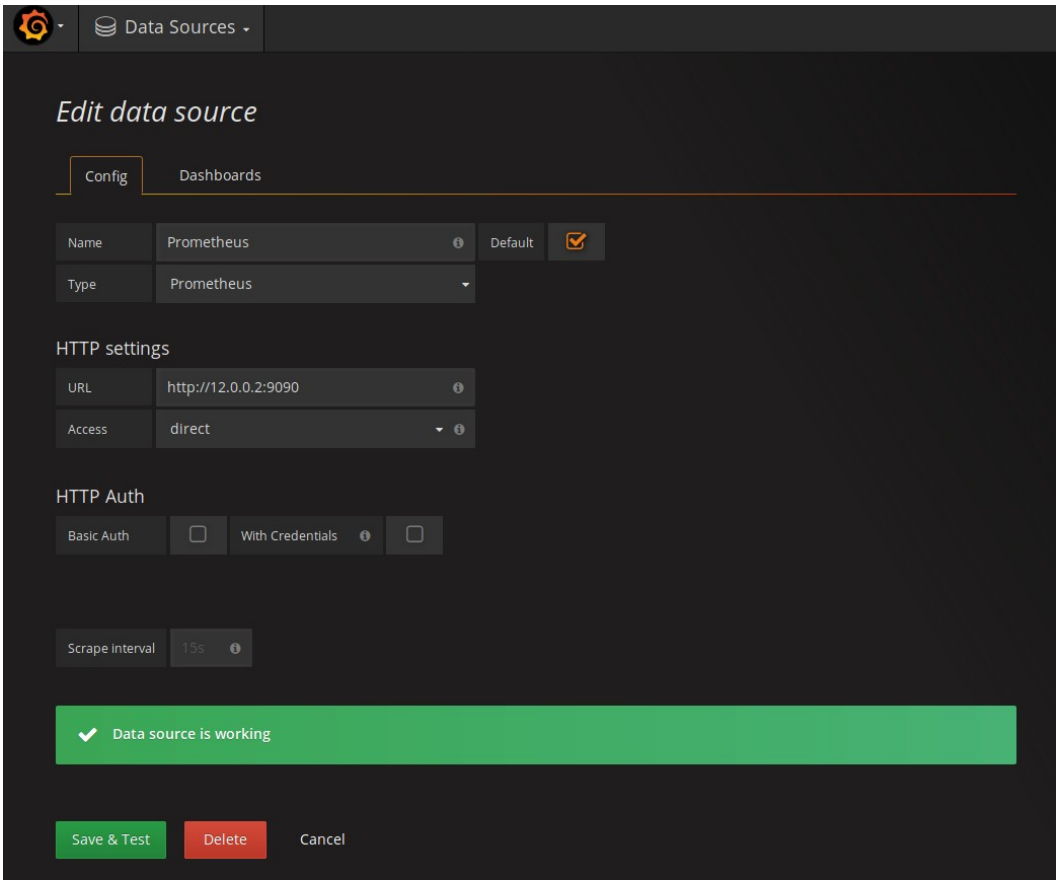
2)
On va ajouter grafana à notre stack.

Dans notre répertoire myhbsapp, on crée un nouveau fichier de configuration pour grafana au chemin : */grafana/grafana.ini :*

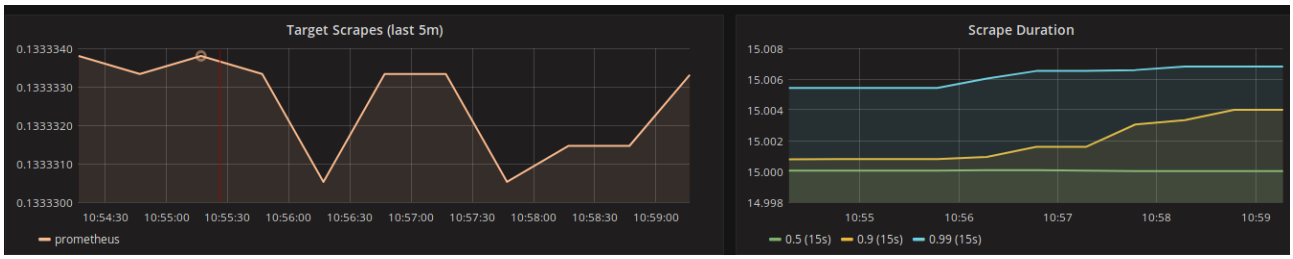
```
# default section
instance_name = ${HOSTNAME}

[security]
admin_user = admin

[auth.google]
client_secret = 0ldS3cretKey
```

Puis dans notre stack-monitor.yml on ajoute les lignes suivantes en créant un volume avec notre fichier de configuration grafana.ini :

```
grafana:
  image: grafana/grafana:4.6.2
  volumes:
    - ./grafana/grafana.ini:/etc/grafana/grafana.ini
  ports:
    - "3002:3000"
  networks:
    - net-monitoring
```



Dans la figure ci-dessus, on a configuré grafana pour qu'elle prenne les données de Prometheus en source.
On peut donc créer un nouveau dashboard et admirer de beaux graphiques :



Partie III – Le bazooka

Pour la suite, nous allons installer tous les outils précédents à l'aide de SwarmProm.

1) Dans un premier temps, on arrête la stack « monitor » tout en laissant tourner la galerie.

On clone le répertoire de stefan prodan : *https://github.com/stefanprodan/swarmprom.git*
Puis on lance une stack :

```
vagrant@manager:/vagrant/myhbsapp/swarmprom$ ADMIN_USER=admin \
> ADMIN_PASSWORD=admin \
> SLACK_URL=https://hooks.slack.com/services/TOKEN \
> SLACK_CHANNEL=devops-alerts \
> SLACK_USER=alertmanager \
> sudo docker stack deploy -c docker-compose.yml mon
Updating service mon_alertmanager (id: agvajfhoevw16ztbia3hrh8pc)
Updating service mon_unsee (id: zc7xsmq3rqd097a5vl8jtxq6n)
Updating service mon_node-exporter (id: hy56gob8tcaqm6porkm9znyo6)
Updating service mon_prometheus (id: hxy3e4rwmp0amxwo3ulblali5)
Creating service mon_caddy
Creating service mon_dockerd-exporter
Creating service mon_cadvisor
Updating service mon_grafana (id: uasmckq9ndhz9uj6hhus0afi8)
```

Pour exécuter la commande précédente, j'ai dû modifier le fichier docker-compose.yml car le service caddy avait le même port que mon application. J'ai par conséquent assigné le port 3003:3000 au caddy :

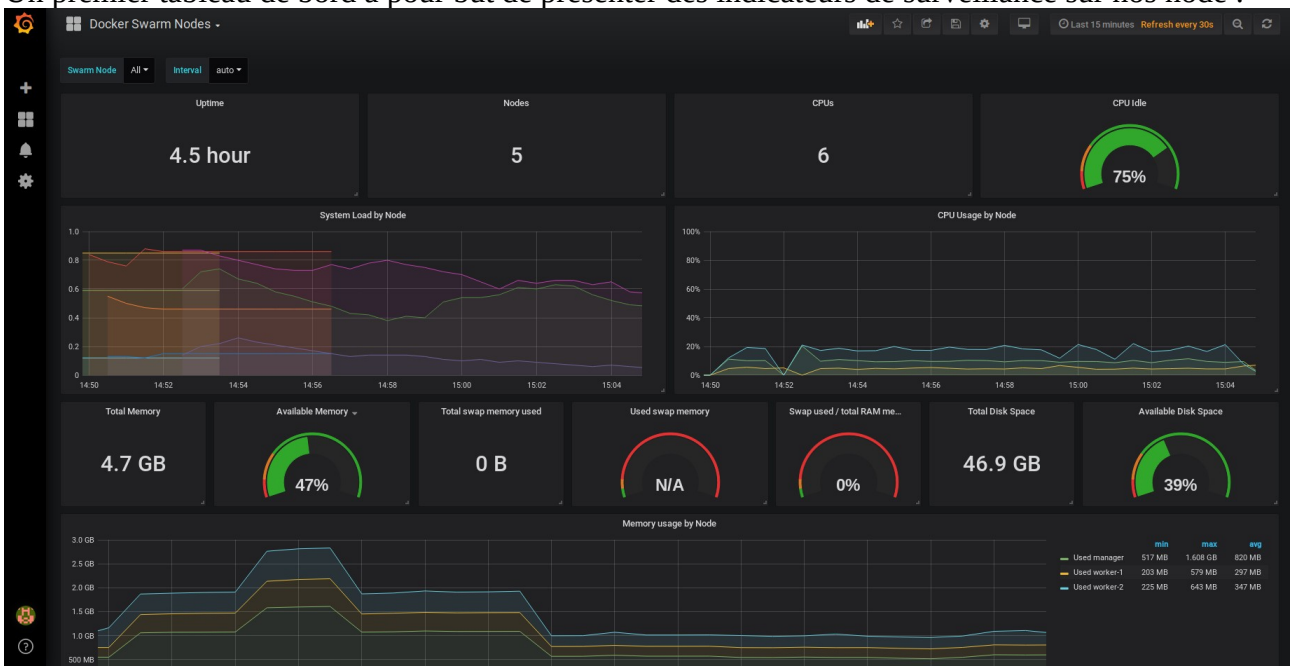
```
caddy:
  image: stefanprodan/caddy
  ports:
    - "3003:3000"
    - "9090:9090"
    - "9093:9093"
    - "9094:9094"
```

On peut désormais se connecter avec les identifiants de la commande précédente (user: admin / password : admin)

Il nous est demandé de renseigner un nouveau mot de passe pour plus de sécurité. On choisira donc *anthocloud* .

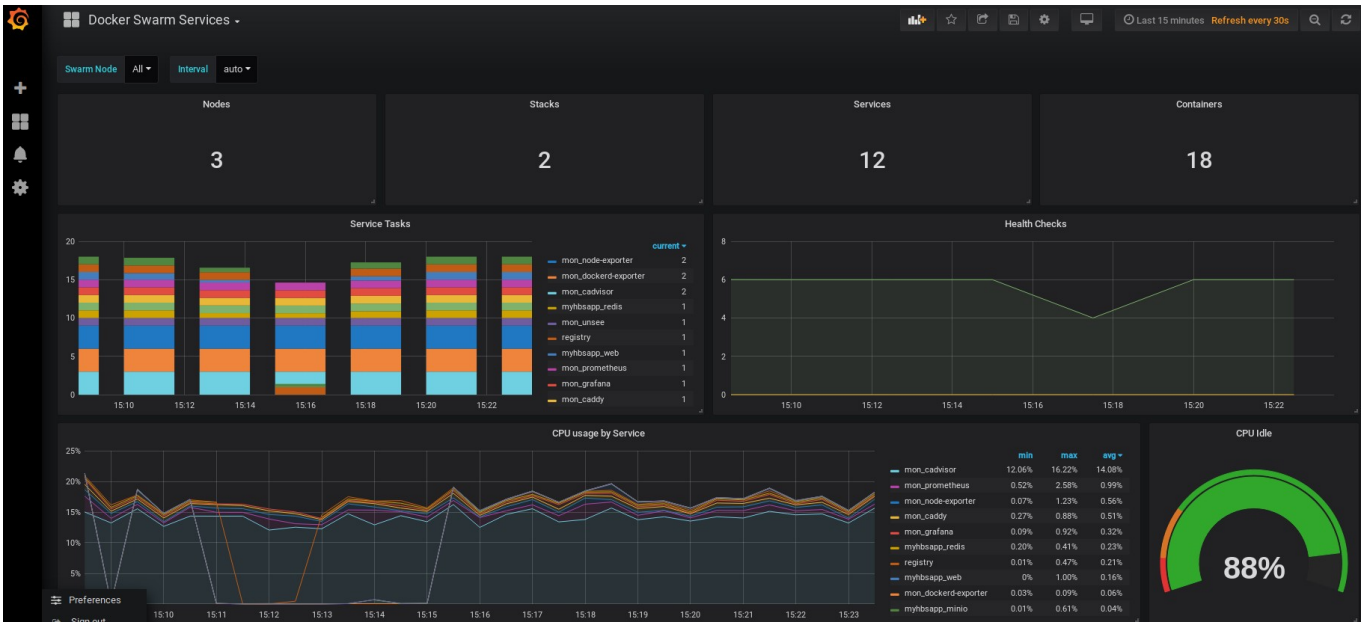
2)
Nous avons ainsi 2 tableaux de bord pour grafana.

Un premier tableau de bord a pour but de présenter des indicateurs de surveillance sur nos node :



Ce dernier peut être filtré par node. On est alors en mesure d'avoir le nombre de processeurs, l'espace disque total, des graphiques de conteneurs par swarm et par node ainsi que l'utilisation du réseau.

Nous avons ensuite un autre tableau de bord mais cette fois lié aux services :

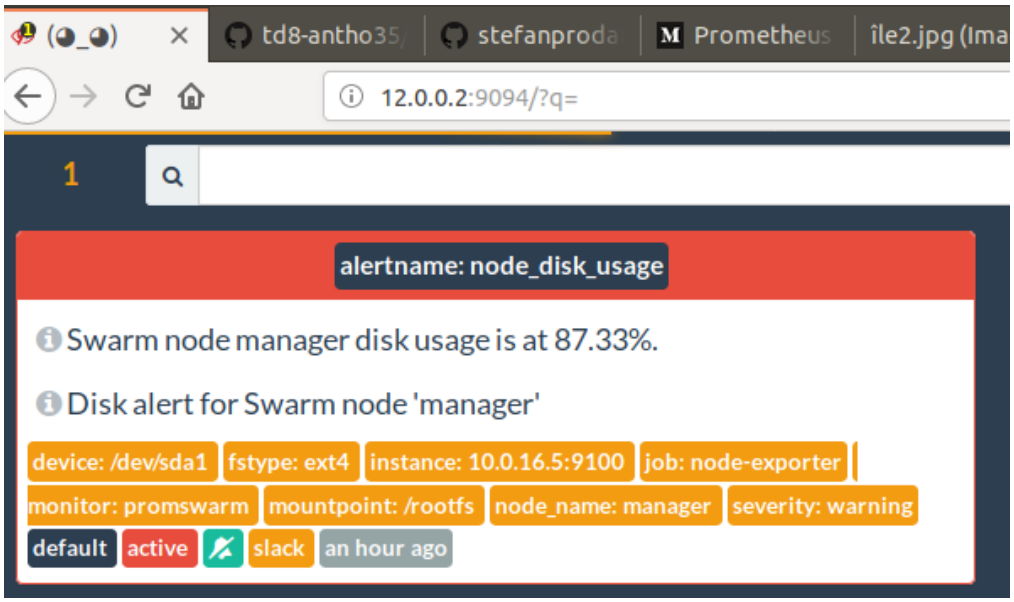


Il fournit des indicateurs permettant de surveiller l'utilisation des resources de nos stacks et services. Il répertorie le nombre de nodes, stack (ici 2, myhbsapp et mon), services et conteneurs en marche. Il détaille également les taches des services dans leur swarm mais aussi l'utilisation du réseau, de la mémoire, et du CPU par service et conteneur.

3)

Les données de Prometheus permettent d'avoir des graphiques sur l'utilisation du CPU, de la mémoire...

4)
On se rend à l'adresse 12.0.0.2:9094 qui correspond à Unsee : un gestionnaire d'alertes.



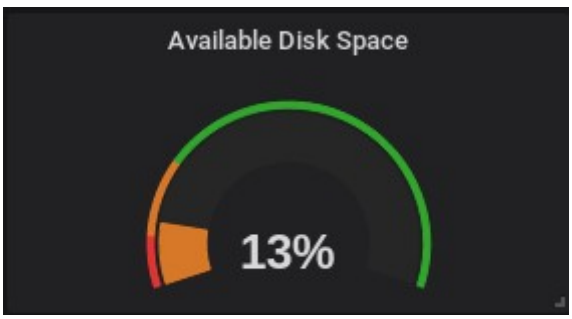
Il semblerait que mon manager commence à être un peu chargé.

5)

Des alertes peuvent être configurées dans Prometheus :
- **Swarm Node CPU Usage :** l'utilisation du processeur d'un nœud dépasse 80 % pendant 5 minutes
- **Swarm Node Memory Alert :** l'utilisation de la mémoire d'un nœud dépasse 80 % pendant 5 minutes
- **Swarm Node Disk Alert :** l'utilisation du stockage d'un nœud dépasse 85 % pendant 5 minutes
- **Swarm Node Disk Fill Rate Alert :** le stockage d'un nœud va rester en dehors de l'espace libre pendant 6 heures

Il est possible d'ajouter à cela d'autres alertes aux fichiers *swarm_node* et *swarm_task* et de relancer un « stack deploy » pour les mettre à jour.

6)
Comme remarqué précédemment, mon manager commence à être surchargé. Dans grafana on va donc aller voir plus précisément notre node (le manager) dans le tableau de bord « Docker Swarm Nodes »



En effet, il ne reste plus que 13 % d'espace libre sur le disque de mon manager.

Depuis notre manager, on cherche à voir la liste de nos images :

```
vagrant@manager:~$ sudo docker images
REPOSITORY                          TAG                        IMAGE ID            CREATED             SIZE
minio/minio                         <none>                     e3d706d85b72        3 days ago          40.6MB
127.0.0.1:5000/myhbsapp             latest                     bcc58ee15384        4 days ago          750MB
127.0.0.1:5000/puppeteer            latest                     f3780f50457e        4 days ago          1.1GB
stefanprodan/swarmprom-grafana      <none>                     cb95f9301d31        4 days ago          236MB
stefanprodan/swarmprom-node-exporter <none>                    ef0320896deb        4 days ago          22.9MB
stefanprodan/swarmprom-prometheus   <none>                     b4b1ceb3e77a        4 days ago          99.8MB
127.0.0.1:5000/myhbsapp             <none>                     aaf5b217c0ad        4 days ago          748MB
127.0.0.1:5000/myhbsapp             <none>                     9179a3703856        4 days ago          748MB
<none>                              <none>                     762141d19c48        4 days ago          748MB
127.0.0.1:5000/myhbsapp             <none>                     1eec5fe77c24        4 days ago          748MB
<none>                              <none>                     7fd2208b0106        5 days ago          1.1GB
<none>                              <none>                     48ce7c83471e        5 days ago          748MB
prom/prometheus                     <none>                     586af4300e31        6 days ago          101MB
<none>                              <none>                     add66ad0c315        6 days ago          748MB
<none>                              <none>                     65c93e47fe5c        6 days ago          748MB
node                                latest                     8c67bfd7b95b        8 days ago          899MB
127.0.0.1:5000/myhbsapp             <none>                     029f94e873bb        9 days ago          96.9MB
minio/minio                         latest                     79286ccd88c6        10 days ago         41.3MB
minio/minio                         RELEASE.2019-01-23T23-18-58Z b6b81ae546ac      10 days ago         41.3MB
registry                            <none>                     116995fd6624        2 weeks ago         25.8MB
node                                8.15.0-alpine              5c0c5c94503f        5 weeks ago         66.3MB
redis                               alpine                     b42dc832c855        6 weeks ago         40.9MB
stefanprodan/swarmprom-alertmanager <none>                     cc87506b6843        2 months ago        31.9MB
google/cadvisor                     <none>                     eb1210707573        2 months ago        69.6MB
alekzonder/puppeteer                latest                     b7407aba3236        5 months ago        1.1GB
google/cadvisor                     <none>                     4623226ef052        11 months ago       62.2MB
quay.io/prometheus/prometheus       <none>                     67141fa03496        15 months ago       80.2MB
stefanprodan/caddy                  <none>                     655880563633        15 months ago       24.7MB
redis                               3.2.6                      45c3ea2cecac        2 years ago         183MB
```

On va supprimer les images ne portant pas intérêt.
J'ai donc supprimé l'image node (899MB), les images de 127.0.0.1:5000/myhbsapp qui sont dupliqués. Ceci à l'aide de la commande « sudo docker rmi IMAGE_ID »
Après modification, voici l'état du disque :

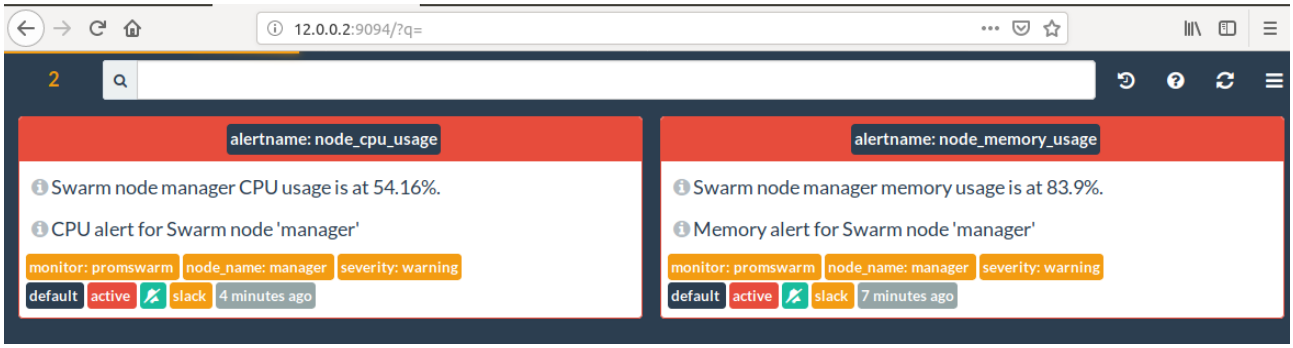
De l'espace a donc été fait sur le manager.

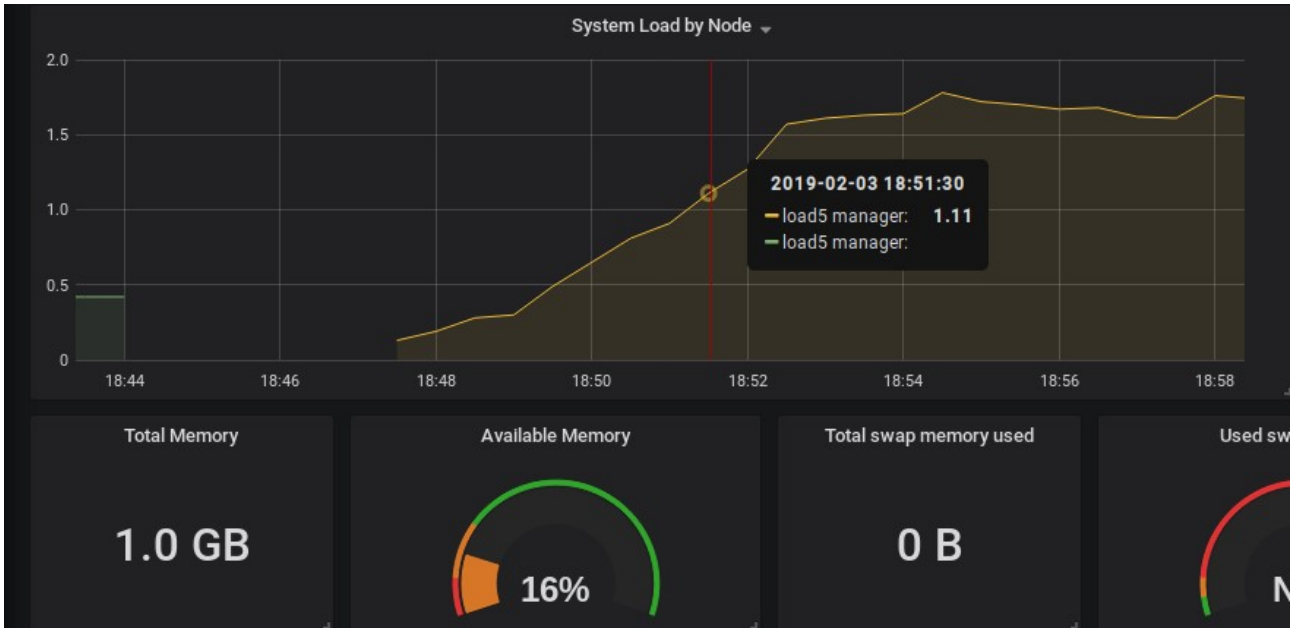
A présent, nous allons voir si notre application supporte l'interaction de plusieurs utilisateurs. J'ai dans un premier temps modifié mon fichier puppeteer.js, de façon à ce qu'il émule un client navigant sur un premier album et ouvrant une photo, il revient à l'accueil pour se rendre sur un second album. Il supprime cet album et le réajoute par la suite de façon à ce que son interaction n'interfère par avec les autres utilisateurs.

On va créer 5 utilisateurs qui se connectent à notre application de façon récurrente.
On crée donc 5 répliques pour notre service « myhbsapp_puppeteer » :

```
vagrant@manager:/vagrant/myhbsapp/swarmprom$ sudo docker service scale myhbsapp_puppeteer=5
myhbsapp_puppeteer scaled to 5
overall progress: 2 out of 5 tasks
1/5: starting  [=============================================>     ]
2/5: ready     [=====================================>            ]
3/5: running   [==================================================>]
4/5: ready     [=====================================>            ]
5/5: running   [==================================================>]
```

Des alertes apparaissent :





L'utilisation de la mémoire a augmenté : il n'y a plus que 16 % d'espace mémoire.
J'ai donc cherché à trouver quelles étaient les sources de cette surcharge.
Voici donc les conteneurs les plus volumineux.

```
CONTAINER ID        NAME                                CPU %               MEM USAGE / LIMIT   MEM %               NET I/O             BLOCK I/O           PIDS
39047f1f6fb4        mon_prometheus.1.pyue109n431wwlj6kjrernn3f   0.05%               208.1MiB / 992MiB   20.98%              7.54MB / 488kB      688MB / 14.6MB      11

CONTAINER ID        NAME                                                                CPU %               MEM USAGE / LIMIT   MEM %               NET I/O             BLOCK I/O           PIDS
59c5623c6969        mon_cadvisor.uvgk5xbnt3oqovqalpqc0daj8.vvevswtif6wjpy8tv5uyi6od8   4.56%               60.67MiB / 128MiB   47.40%              45.5kB / 3.17MB     395MB / 0B          18

CONTAINER ID        NAME                                CPU %               MEM USAGE / LIMIT   MEM %               NET I/O             BLOCK I/O           PIDS
f1b2031cbcb8        mon_grafana.1.syk3w678kfq272cd5vvdtk1oa   0.06%               22.81MiB / 128MiB   17.82%              681kB / 830kB       498MB / 2.33MB      14
```

Il semblerait que le conteneur lié à cadvisor soit le plus conséquent. Une idée serait de s'intéresser à trouver comment réduire la quantité de données enregistrées par cadvisor.

Conclusion

Au cours de ce TP, nous avons pu installer un ensemble d'outils permettant de recueillir des données et mesures sur notre application. Ces données ont été affichés à l'aide de grafana. Un gestionnaire d'alertes à également été mis en place. Grâce à cette infrastructure, nous avons pu tester les limites de notre application afin d'en identifier les goulots d'étranglement.