UNIVERSITÉ DE RENNES 1

ESIR ECOLE SUPERIEURE D'INGENIEURS DE RENNES

30/01/2019

TD 7

> *« J'atteste que ce travail est original, qu'il indique de façon appropriée tous les emprunts, et qu'il fait référence de façon appropriée à chaque source utilisée »*

Léo Guilpain – ESIR3 - IoT

Table des matières

Introduction

Le but de ce TP est de faire en sorte que notre application passe l'échelle. Nous allons étudier la solution « scale-out »

Partie I - Émulation de clients

Question 01

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch({args: ['--no-sandbox', '--disable-setuid-sandbox']});
  const page = await browser.newPage();
  await page.goto('http://192.168.42.17:3000/');
  await page.screenshot({path: 'example.png'});

  await browser.close();
})();

```

Figure 1 : client screenshot

Ici, nous prenons juste un screenshot de la page d'accueil de mon site. Dès que l'on se rend sur la page http://192.168.42.17:3000/ une capture d'écran est prise. Ensuite on ferme le navigateur.

Question 02

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch()
  const page = await browser.newPage()

  await page.setViewport({ width: 1920, height: 1080 })
  // go to a page setup for mouse event tracking
  await page.goto('http://192.168.42.17:3000/');
  await page.screenshot({ path: './screenshot/homepage.png' })

  //Click on album
  await page.waitForSelector("#album3");
  await page.click('#album3',{delay : 4000});
  // the screenshot should show feedback from the page that right part was clicked.
  await page.screenshot({ path: './screenshot/mouse_click1.png' })

  //Next in caroussel
  await page.waitForSelector("#next");
  await page.click("#next",{delay : 4000});
  // the screenshot should show feedback from the page that right part was clicked.
  await page.screenshot({ path: './screenshot/mouse_click2.png' })

  //Previous in caroussel
  await page.waitForSelector("#previous");
  await page.click("#previous",{delay : 4000});
  // the screenshot should show feedback from the page that right part was clicked.
  await page.screenshot({ path: './screenshot/mouse_click3.png' })

  await browser.close()
})()
```

Figure 2 : Navigation

J'ai créé un fichier naviguation.js qui permet de naviguer entre les pages de mon site. Dans un premier temps je définis la largeur ainsi que la longueur de ma page. Puis je me rends sur mon site.

Enfin avec la fonction "page.click" je spécifie l'ID du bouton sur lequel je veux cliquer. Ce code permet de cliquer sur un album puis de naviguer dans le carrousel. On peut voir « page.waitForSelector », cela permet d'attendre. Tant que la page n'est pas chargée et que

l'item correspondant à l'ID n'est pas chargé, on ne fait rien. Les screenshot m'ont permis de savoir si les commandes étaient bien effectuées.

```
myappstack_web.1.fuv7pu4aiuce@worker2    | GET /stylesheets/style.css 304 0.481 ms - -
myappstack_web.1.fuv7pu4aiuce@worker2    | GET /images/thumbnail/monkey2.jpg 200 1.521 ms - 52781
myappstack_web.1.fuv7pu4aiuce@worker2    | GET /images/rabbit3.jpeg 200 0.929 ms - 190627
myappstack_web.1.fuv7pu4aiuce@worker2    | GET /images/rabbit2.jpg 200 2.106 ms - 41859
myappstack_web.1.fuv7pu4aiuce@worker2    | GET /images/thumbnail/monkey2.jpg 200 2.007 ms - 52781
myappstack_web.1.fuv7pu4aiuce@worker2    | GET /images/bunny/3 200 50.899 ms - 2050
myappstack_web.1.fuv7pu4aiuce@worker2    | GET /stylesheets/style.css 304 0.308 ms - -
```

Figure 3 : service web logs

Question 03

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch()
  const page = await browser.newPage()

  await page.setViewport({ width: 1920, height: 1080 })
  // go to a page setup for mouse event tracking
  await page.goto('http://192.168.42.17:3000/');

  //Go to the page add Album
  await page.waitForSelector("#addAlbum");
  await page.click('#addAlbum');

  //fill the form
  await page.waitForSelector("#albumName");
  await page.type('#albumName', 'cat');

  await page.waitForSelector("#fileupload");
  const input = await page.$('#fileupload');
  await input.uploadFile('./cat3.jpg')

  //Click on submit
  await page.waitForSelector("#submit_album");
  await page.click('#submit_album');
  await page.waitForNavigation();

  //Refresh the page in order to see the new album
  await page.waitForSelector("#refresh");
  await page.click('#refresh',{delay:5000});

  await browser.close()
})()
```

Figure 4 : add Album

Cela permet d'ajouter un album. On remplit ici le formulaire en rentrant le nom de l'album et l'image à upload.

On peut aussi ajouter une image :

```
    //Go to the page add Album
    await page.waitForSelector("#addAlbum");
    await page.click('#addAlbum');

    //fill the form
    await page.waitForSelector("#imageName");
    await page.type('#imageName', 'cat 1');
    await page.waitForSelector("#albumId");
    await page.type('#albumId', '6');

    await page.waitForSelector("#fileuploadimage");
    const input = await page.$('#fileuploadimage');
    await input.uploadFile('./cat3.jpg')

    //Click on submit
    await page.waitForSelector("#submit_image");
    await page.click('#submit_image',{delay:5000});

    //Refresh the page in order to see the new album
    await page.waitForSelector("#refresh");
    await page.click('#refresh',{delay:5000});

    await browser.close();
}
```

Figure 5 : add Image

Pour vérifier le bon fonctionnement, il suffit de regarder les logs du service Web. On obtient ceci :

```
myappstack_web.1.kn2fhky3424v@worker1    | POST /add/album 200 43.650 ms - 1735
myappstack_web.1.kn2fhky3424v@worker1    | GET /stylesheets/style.css 304 0.327 ms - -
myappstack_web.1.kn2fhky3424v@worker1    | GET /images/thumbnail/monkey2.jpg 200 1.488 ms - 52781
myappstack_web.1.kn2fhky3424v@worker1    | GET /add/ 200 29.801 ms - 1735
myappstack_web.1.kn2fhky3424v@worker1    | GET /stylesheets/style.css 304 0.330 ms - -
myappstack_web.1.kn2fhky3424v@worker1    | file received
myappstack_web.1.kn2fhky3424v@worker1    | POST /add/album 200 61.575 ms - 1735
```

Figure 6 : web log

Le client a bien correctement ajouté un album.

Question 04

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch()
  const page = await browser.newPage()

  await page.setViewport({ width: 1920, height: 1080 })
  // go to a page setup for mouse event tracking
  await page.goto('http://192.168.42.17:3000/');

  //Go to the page removeImage
  await page.waitForSelector("#removeImage");
  await page.click('#removeImage');
  await page.screenshot({ path: './screenshot/1_remove.png' })

  //fill the form
  await page.waitForSelector("#idImage");
  await page.type('#idImage', '4');
  await page.screenshot({ path: './screenshot/2_fill.png' })


  //Click on submit
  await page.waitForSelector("#submit_removeImage");
  await page.click('#submit_removeImage');
  await page.screenshot({ path: './screenshot/3_submit.png' })

  //Refresh the page
  await page.waitForSelector("#refresh");
  await page.click('#refresh',{delay:5000});
  await page.screenshot({ path: './screenshot/4_refresh.png' })

  await browser.close()
})()
```

Figure 7 : remove Image

On clique sur le bouton “Remove Image” puis on remplit le formulaire avec l’id de l’image que l’on souhaite supprimer.

Ensuite on soumet le formulaire.

```
myappstack_web.1.yr4pn6cd4rks@manager    | GET /removeImage 200 92.425 ms - 1301
myappstack_web.1.yr4pn6cd4rks@manager    | GET /stylesheets/style.css 304 0.417 ms - -
myappstack_web.1.yr4pn6cd4rks@manager    | GET /images/thumbnail/rabbit2.jpg 200 1.535 ms - 41859
myappstack_web.1.yr4pn6cd4rks@manager    | GET /images/thumbnail/cat3.jpg 200 1.831 ms - 64823
myappstack_web.1.yr4pn6cd4rks@manager    | GET /removeImage/ 200 27.716 ms - 1301
myappstack_web.1.yr4pn6cd4rks@manager    | GET /stylesheets/style.css 304 0.352 ms - -
myappstack_web.1.yr4pn6cd4rks@manager    | POST /removeImage 200 63.225 ms - 1301
myappstack_web.1.yr4pn6cd4rks@manager    | GET /stylesheets/style.css 304 0.316 ms - -
myappstack_web.1.yr4pn6cd4rks@manager    | file deleted successfully
```

Figure 8 : service web log

On peut voir dans les logs que l’image a été correctement supprimée.

Question 05

Pour effectuer de façon aléatoire une action, il suffit de générer un nombre aléatoire puis de le traiter. On génère un nombre aléatoire entre 0 et 9. Une fois ce nombre généré, on réalise une action en fonction.

```
var random = Math.floor(Math.random() * 10);

const browser = await puppeteer.launch({args: ['--no-sandbox', '--disable-setuid-sandbox']})
const page = await browser.newPage();
await page.goto('http://192.168.42.17:3000/');


//Ajoute une image avec une probabilité d'1/10 ;
if(random == 0){
  addImage();
}

//Supprime une image avec une probabilité d'1/10 ;
else if(random ==1){
  removeImage();
}

//Se promène sur le site avec une probabilité de 8/10.
else{
  Nagation();
}
```

Figure 9 : random

Les fonctions qui sont appelées sont celles générées auparavant.

Question 06

```
FROM alekzonder/puppeteer

# Create app directory
WORKDIR app

#Copy our files in repo /app
COPY . /app/

RUN npm install

CMD [ "npm", "start" ]

```

Figure 10 : Dockerfile

On récupère l'image « alekzonder/puppeteer » puis on copie tous nos fichiers dans son dossier app. Puis on lance notre fichier « index.js » qui correspond aux actions aléatoires.

```
version: '3'
services:
  puppeteer:
    image: localhost:5000/mypuppeteerimage
    build: .

```

Figure 11 : docker-compose

Dans un premier temps, on crée le service manuellement. Pour faire cela, on « build » l'image et on la « push » sur le registre. Ensuite on crée le service. On peut donc voir que tous les services sont bien lancés "sudo docker services ls"

```
vagrant@manager:/vagrant/puppeteer$ sudo docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE                                          PORTS
jh442pyd5yeb        myappstack_redis    replicated          1/1                 redis:latest                                   *:6379->6379/tcp
zvn0yhw6glwd        myappstack_web      replicated          3/3                 localhost:5000/myappgallery:latest             *:3000->3000/tcp
rw5ghke38zdr        puppeteer_service   replicated          3/3                 localhost:5000/mypuppeteerimage:latest
qj4krwh81owy        registry            replicated          1/1                 registry:2                                     *:5000->5000/tcp
```

Figure 12 : service ls

```
myappstack_web.2.rkjyp3vrvtry@manager    | GET /images/thumbnail/monkey2.jpg 304 6.251 ms - -
myappstack_web.2.rkjyp3vrvtry@manager    | GET /images/thumbnail/rabbit2.jpg 304 14.543 ms - -
myappstack_web.2.rkjyp3vrvtry@manager    | GET /images/thumbnail/cat3.jpg 304 1.841 ms - -
myappstack_web.2.rkjyp3vrvtry@manager    | GET /stylesheets/style.css 304 15.631 ms - -
myappstack_web.2.rkjyp3vrvtry@manager    | GET /add/ 200 147.280 ms - 1808
myappstack_web.2.rkjyp3vrvtry@manager    | GET /stylesheets/style.css 304 0.717 ms - -
myappstack_web.2.rkjyp3vrvtry@manager    | GET /images/cat/6 200 316.554 ms - 1364
myappstack_web.2.rkjyp3vrvtry@manager    | GET /stylesheets/style.css 304 0.481 ms - -
myappstack_web.2.rkjyp3vrvtry@manager    | GET /images/undefined 404 32.178 ms - 1295
myappstack_web.2.rkjyp3vrvtry@manager    | GET / 304 59.235 ms - -
myappstack_web.2.rkjyp3vrvtry@manager    | GET /stylesheets/style.css 304 0.313 ms - -
myappstack_web.2.rkjyp3vrvtry@manager    | GET /images/thumbnail/rabbit2.jpg 304 1.128 ms - -
myappstack_web.2.rkjyp3vrvtry@manager    | GET /images/thumbnail/monkey2.jpg 304 1.185 ms - -
myappstack_web.2.rkjyp3vrvtry@manager    | GET /images/thumbnail/cat3.jpg 304 0.581 ms - -
myappstack_web.2.rkjyp3vrvtry@manager    | GET /removeImage/ 200 67.520 ms - 1258
myappstack_web.2.rkjyp3vrvtry@manager    | GET /stylesheets/style.css 304 0.346 ms - -
```

D'après les logs, on voit bien que le fichier index.js a bien été lancé puisqu'il y a eu un ajout d'image mais également une navigation dans les dossiers.

Comme on peut le voir le service est fonctionnel, on l'ajoute dans le stack pour pouvoir tout déployer ensemble.

docker-stack.yml

```yaml
version: "3"
services:
  web:
    image: localhost:5000/myappgallery
    deploy:
      replicas: 3
      restart_policy:
        condition: on-failure
    ports:
      - 3000:3000
    volumes:
      - './public/images:/usr/src/myappgallery/public/images'
    networks:
      - test_net

  redis:
    image: redis
    ports:
      - "6379:6379"
    volumes:
      - "/vagrant/myappgallery/data:/data"
    deploy:
      placement:
        constraints: [node.role == manager]
    command: redis-server --appendonly yes
    networks:
      - test_net

  puppeteer:
    image: localhost:5000/mypuppeteerimage
    deploy:
      replicas: 3
      restart_policy:
        condition: on-failure
    networks:
      - test_net
networks:
  test_net:
    external: true
```

Figure 13 : docker-stack

On déploie le stack :

```
vagrant@manager:/vagrant/myappgallery$ sudo docker stack deploy -c docker-stack.yml myappstack
Creating service myappstack_puppeteer
Creating service myappstack_web
Creating service myappstack_redis
```

Figure 14 : stack deploy

```
vagrant@manager:/vagrant/myappgallery$ sudo docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE                                          PORTS
81rpx3u1zzym        myappstack_puppeteer   replicated          3/3                 localhost:5000/mypuppeteerimage:latest
wplujvc6pcxd        myappstack_redis       replicated          1/1                 redis:latest                                   *:6379->6379/tcp
m4ccqnzv6zgj        myappstack_web         replicated          3/3                 localhost:5000/myappgallery:latest             *:3000->3000/tcp
qj4krwh81owy        registry               replicated          1/1                 registry:2                                     *:5000->5000/tcp
```

Figure 15 : service ls

Les trois services sont bien correctement lancés.

Si on veut faire des screenshot de l'application alors il faut faire un volume permettant de monter le répertoire screenshot et de récupérer les captures.

Figure 16 : visualizer

J'ai également rajouté un « visualizer » pour pouvoir afficher les services sur mes machines.

Je me suis servi de ce site : https://docs.docker.com/v17.12/get-started/part5/

Partie II - Monitoring avec Prometheus

Question 01

Pour créer ce fichier, on modifie le fichier “install-docker.yml” en ajoutant ceci :

```
- name: Ansible create file daemon.json
  copy:
    dest: "/etc/docker/daemon.json"
    content: |
      {
      "metrics-addr" : "0.0.0.0:9323",
      "experimental" : true
      }
```

Figure 17 : création du fichier daemon.json

Je me suis servi de ce site : http://www.mydailytutorials.com/ansible-create-files/

```
vagrant@manager:/etc/docker$ ls
daemon.json  key.json
vagrant@manager:/etc/docker$ cat daemon.json
{
"metrics-addr" : "0.0.0.0:9323",
"experimental" : true
}
```

Figure 18 : cat daemon.json

Comme on peut le voir, le fichier à bien été créé sur les machines.

Question 02

Après avoir fait ces modifications, il faut faire « vagrant up –provision » puis il faut relancer docker en effectuant la commande : « etc/init.d/docker restart »

```
vagrant@manager:~$ curl http://192.168.42.17:9323/metrics
# HELP builder_builds_failed_total Number of failed image builds
# TYPE builder_builds_failed_total counter
builder_builds_failed_total{reason="build_canceled"} 0
builder_builds_failed_total{reason="build_target_not_reachable_error"} 0
builder_builds_failed_total{reason="command_not_supported_error"} 0
builder_builds_failed_total{reason="dockerfile_empty_error"} 0
builder_builds_failed_total{reason="dockerfile_syntax_error"} 0
builder_builds_failed_total{reason="error_processing_commands_error"} 0
builder_builds_failed_total{reason="missing_onbuild_arguments_error"} 0
builder_builds_failed_total{reason="unknown_instruction_error"} 0
# HELP builder_builds_triggered_total Number of triggered image builds
# TYPE builder_builds_triggered_total counter
builder_builds_triggered_total 0
```

Figure 19 : curl metrics

Le curl fonctionne correctement.

Question 03

Pour commencer, on crée un fichier “prometheus.yml”. Ce fichier nous est donné sur le site : https://docs.docker.com/config/thirdparty/prometheus/

```yaml
# 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).

  external_labels:
      monitor: 'codelab-monitor'

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first.rules"
  # - "second.rules"

scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'
    static_configs:
      - targets: ['192.168.42.17:9090']

  - job_name: 'docker'
    static_configs:
      - targets: ['192.168.42.17:9323']
```

Figure 20 : prometheus.yml

Ensuite, dans notre fichier “docker-install” il suffit de rajouter les lignes suivantes pour copier notre fichier “prometheus.yml” sur nos machines.

```yaml
- name: Copy prometheus in repository /tmp
  copy:
    src: prometheus.yml
    dest: /tmp/prometheus.yml
```

Figure 21 : add prometheus

Enfin, on vérifie que le fichier a bien été copié sur la machine :

```
vagrant@manager:/$ cd tmp
vagrant@manager:/tmp$ cat prometheus.yml
# 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).

  external_labels:
      monitor: 'codelab-monitor'

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first.rules"
  # - "second.rules"

scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'
    static_configs:
      - targets: ['192.168.42.17:9090']

  - job_name: 'docker'
    static_configs:
      - targets: ['192.168.42.17:9323']
```

Figure 22 : cat prometheus

Ensuite on crée le service. Ce dernier est bien lancé comme on peut le voir :

```
vagrant@manager:/vagrant$ sudo docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE                                PORTS
05op530bd83v        my-prometheus       replicated          1/1                 prom/prometheus:latest               *:9090->9090/tcp
q4p1jj6so90z        myappstack_puppeteer replicated         2/3                 localhost:5000/mypuppeteerimage:latest
iy8birvsseza        myappstack_redis    replicated          1/1                 redis:latest                         *:6379->6379/tcp
zk2e8znvjugu        myappstack_visualizer replicated        1/1                 dockersamples/visualizer:stable      *:8080->8080/tcp
p6yaczbw8exh        myappstack_web      replicated          3/3                 localhost:5000/myappgallery:latest   *:3000->3000/tcp
x3lo3loh6x55        registry            replicated          1/1                 registry:2                           *:5000->5000/tcp
```

Figure 23 : service ls

Ensuite, on vérifie en se rendant sur le lien : http://192.168.42.17:9090/targets

On obtient ceci :

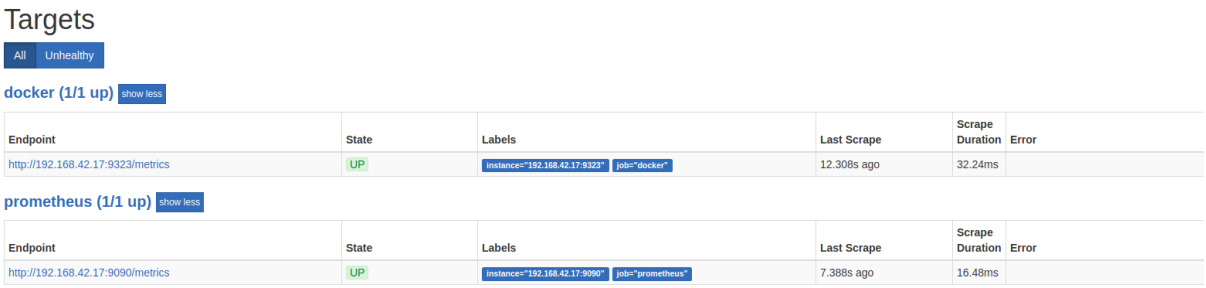

Figure 24 : target

Nous pouvons voir que nous avons bien accès aux métriques du site. On regarde ensuite dans un graphique les données de visite.

Avant de faire un ping, notre graphique ressemble à cela :

Figure 25 : graph avant ping

Ensuite, on crée un service qui ping le site « docker.com ».

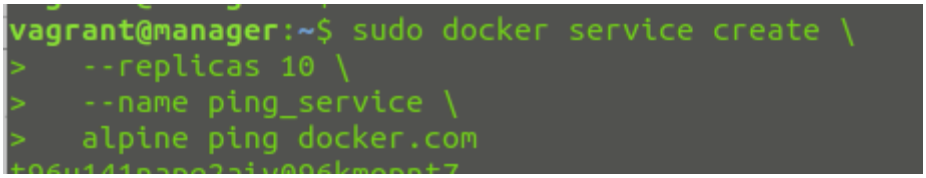

Figure 26 : ping service

Puis on regarde le graphique et on peut voir que le compteur a augmenté.

Figure 27 : graph après ping

Les métriques permettent de fournir un rapport sur différentes actions de l'utilisateur (par ex : trafic par page, source du trafic, durée moyenne de visite). Le plus utilisé est le taux de rebond. Ce taux permet de connaître l'intérêt des utilisateurs lorsqu'ils visitent notre site.

J'ai utilisé ce site : https://www.rouillier.ca/blog/six-metriques-google-analytics/

Conclusion

Après avoir émuler des clients qui exécutent des actions sur notre applications, nous avons mis en place la supervision grâce à Prometheus.