Blin Sébastien
Collin Pierre-Henri

ECOLE SUPERIEURE
D'INGENIEURS DE RENNES

École supérieure d'ingénieurs de Rennes

2ème année
Parcours Informatique

TP 2 AL

Déploiement de l'application RSS-Reader

Sous l'encadrement de :

Temple Paul

1 Introduction

Ce TP avait pour but de déployer l'application *RSS Reader* de *Netflix* a l'aide de *docker* et des briques fournies par *Netflix*.

2 Génération des images

Pour ce TP, nous avons choisi de générer 2 images *docker*. La première contenant une instance *tomcat* qui lance *eureka*. Le *Dockerfile* est basé sur une *ubuntu 14 :04*, où on y installe *java8, tomcat* et *eureka*. Nous y ajoutons les fichiers de configuration *server.xml* et *tomcat-users.xml*. Le lancement du container exécute un petit script shell :

```
cp /eureka/eureka-server/build/libs/eureka-server-1.4.7-SNAPSHOT.war\
/opt/tomcat/webapps/eureka.war
sh /opt/tomcat/bin/catalina.sh run
```

Le second *Dockerfile* génère une image contenant la partie *rss-edge* et *middletier*. On y ajoute les fichiers de configurations afin de modifier les `http://localhost` en `http://tomcat` (la raison est expliquée plus bas). Ainsi que 2 scripts, l'un démarrant la partie rss-edge, l'autre la partie middletier. Pour décider quelle partie lancer, on la donne en paramètre au lancement du container.

3 Création d'une instance applicative

Nous avons regardé 2 solutions. La première était de créer un *docker-compose*, la seconde d'utiliser l'option *–link* pour attacher 2 containers ensemble. Nous avons choisi la seconde pour plus de simplicité/flexibilité (plus facile d'isoler les containers).

3.1 Création d'une instance

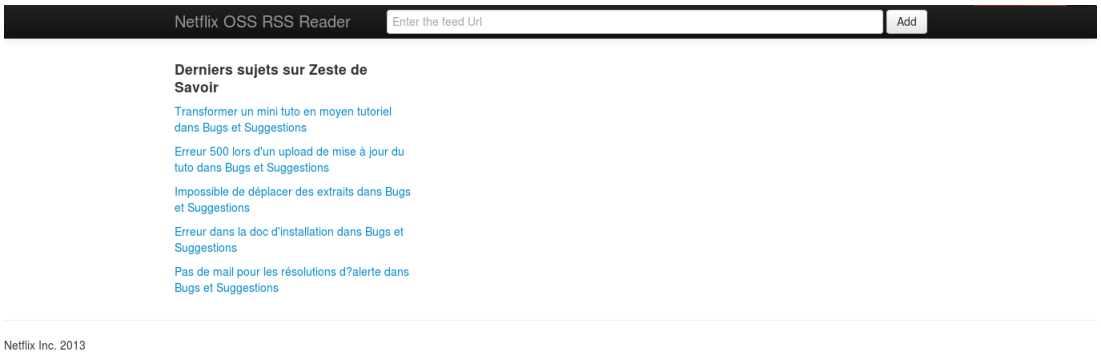
Pour lancer une instance complète, il suffit de quelques commandes :

```
sudo service nginx start
sudo service docker start
sudo docker rm edge1 edge2 middletier1 middletier2 tomcat
sudo docker run --rm -it --name tomcat -h tomcat -p 8000:80 -p 8080:8080 tomcat
sudo docker run --rm -it --name middletier1 -h middletier1 --link tomcat:tomcat \
-p 9191:9191 middletier sh middletier.sh
sudo docker run --rm -it --name edge1 -h edge1 --link tomcat:tomcat --link \
middletier1:middletier -p 9090:9090 middletier sh rss-edge.sh
```

On démarre les services *docker* et *nginx*, puis on supprime les précédentes instances (afin de vérifier si les containers ont bien été supprimé la dernière fois). On lance le container *tomcat*. Sur `http://localhost:8000`, on pourra trouver la page d'accueil de *tomcat* et *eureka* sur `http://localhost/eureka/` et `http://localhost:8080/eureka/v2/`. On lance ensuite notre *middletier* en l'attachant au container *tomcat*, sous l'host *tomcat*. Ainsi, un ping sur l'adresse `http://tomcat` depuis le container pinguera notre instance *tomcat*. On peut ensuite lancer le client *rss-edge* en l'attachant aux 2 précédents containers, sans oublier de binder le port 9090 avec un port de la machine (ici le même, on verra un bind différent plus bas).

3.2 Vérification du bon fonctionnement

Afin de vérifier le bon fonctionnement de l'instance, il suffit de se rendre à l'adresse `http://localhost:9090/jsp/rss.jsp`. On doit maintenant pouvoir ajouter des flux RSS et obtenir une page comme présentée dans la Figure 3.2



4 Le multi-instance

4.1 Gérer plusieurs instance

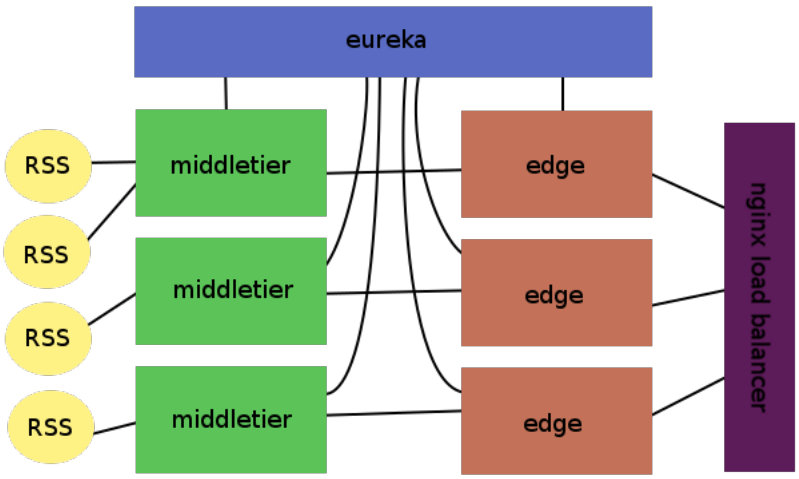
Afin de gérer plusieurs instances, il suffit de lancer plusieurs containers *edge* et *middletier*, en les linkant ensemble. Pour faire ça de manière rapide, on peut se baser sur la sortie du programme *goodMonkey.py* (en modifiant le nombre d'instances). Pour 2 instances, on aura par exemple :

```
sudo docker run --rm --name middletier1 -h middletier1 --link tomcat:tomcat -p \
9192:9191 middletier sh middletier.sh
sudo docker run --rm --name edge1 -h edge1 --link tomcat:tomcat --link \
middletier1:middletier -p 9091:9090 middletier sh rss-edge.sh
sudo docker run --rm --name middletier2 -h middletier2 --link tomcat:tomcat -p \
9193:9191 middletier sh middletier.sh&
sudo docker run --rm --name edge2 -h edge2 --link tomcat:tomcat --link \
middletier2:middletier -p 9092:9090 middletier sh rss-edge.sh&
```

Ou chaque partie *rss-edge* est reliée à son *middletier*. Il ne faut juste pas oublier de binder des ports machines différents, sinon une erreur sera lancée (ici une instance *edge* est bindée au port 9091 et l'autre au port 9092 par exemple).

4.2 Schéma de fonctionnement

Au final, voici comment l'application fonctionne actuellement (les 2 monkeys ne sont pas représentés) Figure 4.2



Fonctionnement actuel de l'application

4.3 Amélioration possibles

Ici, nous faisons fonctionner 2 *middletier* différents pour 2 instances *rss-edge*. On peut aussi imaginer faire tourner plusieurs *rss-edge* pour une seule instance de *middletier* (il suffit de changer le link du container), ce qui peut permettre d'envoyer moins de requêtes à un *rss-edge*.

4.4 Faiblesses

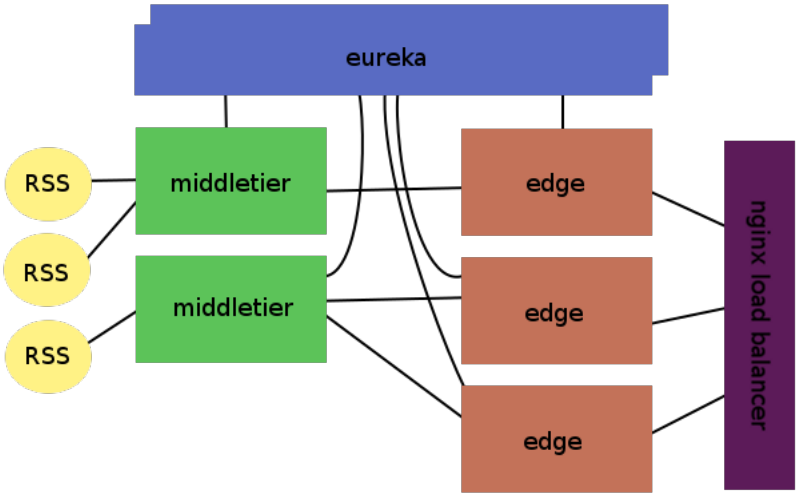
Nous avons plusieurs problèmes dans notre implémentation :

— Tout se trouve sur une seule machine, on aurait pu utiliser plusieurs machines afin d'être encore moins sensible aux pannes.

— Le fait que tout se trouve sur une seule machine fait qu'on ne peut pas forcément gérer beaucoup d'instances. Par exemple le portable de Sébastien ne gérait pas plus de 3 instances correctement.

— le multi-instance n'est disponible que pour les parties *rss-edge* et *middletier*, il faudrait faire de même pour la partie *tomcat+eureka* afin d'être moins sensible aux pannes. Si le *tomcat* tombe, on ne pourra plus démarrer d'instance (le lien ne pouvant pas se créer).

Ce qui donne le schéma de la Figure 4.4



Fonctionnement amélioré de l'application.

5 Load-balancing entre les instances

Le *load-balancing* entre les différentes instances se font à l'aide de *nginx*, qui écoute par défaut sur le port 80 de la machine. Il suffit d'ajouter ces quelques lignes dans le fichier de configuration (sur Fedora : */etc/nginx/nginx.conf*) :

```
http {
  upstream netflix-rss {
                least_conn;
                server localhost:9092;
                server localhost:9091;
        }

        server {
                listen 8000;
                location / {
                  proxy_pass http://netflix-rss;
                }
        }
}
```

Le *least_ conn* servant à envoyer les connexions au serveur ayant reçu le moins de requêtes. Ici, nous ne gérons que 2 instances sur les ports 9091 et 9092. Si nous décidons de gérer plus d'instances, il faudrait donc rajouter des lignes dans *server*.

6 Des monkeys

Afin de tenter de réagir aux pannes, nous les simulons avec des *Monkeys*. Nous avons donc créé 2 *Monkeys*. Le premier se charge de détruire un des containers de manière aléatoire, l'autre se charge de rétablir les containers manquants.

6.1 BADMONKEY

6.1.1 Implémentation

Le premier *Monkey* est implémenté à l'aide d'une simple ligne de bash :

```
docker rm $(docker stop $(docker ps | \
awk '/middletier*|edge*/ {print $13}' | \
shuf -n 1)) # Bad Monkey, kill a random docker
```

Cette ligne se charge de prendre un docker au hasard parmi les *middletier* et les *rss-edge*, l'arrête et le supprime.

6.1.2 Amélioration

On pourrait améliorer ce *Monkey* en le lançant de manière aléatoire, ainsi qu'améliorer le petit morceau de *awk* (et éviter le $13).

6.2 GOODMONKEY

6.2.1 Implémentation

Le premier *Monkey* est implémenté à l'aide d'un petit script python générant les commandes a exécuter :

```
#!/bin/python
import commands

MAX_INSTANCE = 2

listeActive = commands.getstatusoutput("docker ps")[1]
for i in range(1,MAX_INSTANCE+1):
        if 'middletier' + str(i) not in listeActive:
                port = 9191 + i
                print("sudo docker rm middletier{0}".format(i))
                print("sudo docker run --rm --name middletier{0} -h middletier{0} --link
tomcat:tomcat -p {1}:9191 middletier sh middletier.sh&".format(i, port))
        if 'edge' + str(i) not in listeActive:
                port = 9090 + i
                print("sudo docker rm edge{0}".format(i))
                print("sudo docker run --rm --name edge{0} -h edge{0} --link tomcat:tomcat
--link middletier{0}:middletier -p {1}:9090 middletier sh rss-edge.sh&"
.format(i, port))
```

Le programme se lançant avec :

```
python goodMonkey.py | bash -x
```

6.2.2 Amélioration

L'exécution des commandes n'est pas top et devraient être modifiée, les containers se lancent souvent mal et tous en même temps. De plus on devrait mettre en place un *cron* afin de ne pas à avoir à le lancer soi-même.

7 Conclusion

Au final, nous avons une application fonctionnelle, même si le déploiement des containers donne quelques bogues du côté du *middletier* (l'attache ne se fait pas toujours et on doit relancer le container pour que ça fonctionne). De plus, nous avons utilisé divers outils que nous n'avions pas forcément utilisé avant ce TP comme *docker, nginx, tomcat* par exemple.