Labo 4 : Docker

Auteurs : Alexandra Cerottini, Miguel Do Vale Lopes, Nicolas Ogi
Date : 12.01.2022
Repo : https://github.com/NicolasOgi/Teaching-HEIGVD-AIT-2020-Labo-Docker

Table des matières

Introduction

Ce laboratoire a pour but de configurer un load-balancer de manière dynamique en se familiarisant avec les superviseurs de processus pour Docker tout en comprenant les concepts de scaling dynamique et de gestion décentralisée de serveurs.

Tâche 0 : Identify issues and install the tools

[M1] *Do you think we can use the current solution for a production environment? What are the main problems when deploying it in a production environment?*

Non, on utilise le même nombre de serveurs qui sont configurés de manière statique. Lorsque le trafic augmente et que l'on souhaite ajouter des serveurs, il est nécessaire de mettre à jour manuellement la configuration.

[M2] *Describe what you need to do to add a new webapp container to the infrastructure. Give the exact steps of what you have to do without modifying the way the things are done. Hint: You probably have to modify some configuration and script files in a Docker image.*

Il faut ajouter la nouvelle webapp dans le docker-compose et modifier le haproxy en conséquence :

```
webapp3:
       container_name: ${WEBAPP_3_NAME}
       build:
         context: ./webapp
         dockerfile: Dockerfile
       networks:
         heig:
           ipv4_address: ${WEBAPP_3_IP}
       ports:
         - "4002:3000"
       environment:
            - TAG=${WEBAPP_3_NAME}
            - SERVER_IP=${WEBAPP_3_IP}
haproxy:
       container_name: ha
       build:
         context: ./ha
         dockerfile: Dockerfile
       ports:
```

```
      - 80:80
      - 1936:1936
      - 9999:9999
    expose:
      - 80
      - 1936
      - 9999
    networks:
      heig:
        ipv4_address: ${HA_PROXY_IP}
    environment:
        - WEBAPP_1_IP=${WEBAPP_1_IP}
        - WEBAPP_2_IP=${WEBAPP_2_IP}
        - WEBAPP_3_IP=${WEBAPP_3_IP}
```

Il faut aussi modifier le fichier .env pour y ajouter la nouvelle webapp et une adresse IP :

```
...
WEBAPP_3_NAME=s3
...
WEBAPP_3_IP=192.168.42.33
...
```

Le fichier *haproxy.cfg* doit aussi être modifié pour ajouter cette ligne :

```
...
server s3 ${WEBAPP_3_IP}:3000 check
```

[M3] *Based on your previous answers, you have detected some issues in the current solution. Now propose a better approach at a high level.*

Le grand problème avec cette solution est que l'on ajoute un nouveau conteneur manuellement, ce qui est une tâche relativement longue à faire. Il faudrait accélérer ce processus en modifiant les fichiers indiqués ci-dessus à l'aide d'un script.

[M4] *You probably noticed that the list of web application nodes is hardcoded in the load balancer configuration. How can we manage the web app nodes in a more dynamic fashion?*

On pourrait utiliser un service qui permettrait de savoir quand un nœud est ajouté ou supprimé. Ce service s'occupera ensuite de modifier le fichier du load balancer en conséquence.

[M5] *Do you think our current solution is able to run additional management processes beside the main web server / load balancer process in a container? If not, what is missing / required to reach the goal? If yes, how to proceed to run for example a log forwarding process?*

Non, un conteneur n'exécute qu'un seul processus. Pour pouvoir exécuter plusieurs processus, il est possible d'utiliser un superviseur de processus comme dans la Tâche 1.

[M6] *What happens if we add more web server nodes? Do you think it is really dynamic? It's far away from being a dynamic configuration. Can you propose a solution to solve this?*

Non ce n'est pas vraiment dynamique. En ajoutant plus de nœuds, il faudrait toujours modifier le fichier de configuration du HAProxy en y rajoutant les nouveaux **sed**. Pour rendre cela plus dynamique, il faudrait utiliser un template pour les nœuds ainsi qu'un script pour modifier le fichier de configuration du HAProxy puis redémarrer le conteneur haproxy pour que les modifications soient prises en compte.

[1.] *Take a screenshot of the stats page of HAProxy at http://192.168.42.42:1936. You should see your backend nodes.*

stats

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				1	1	-	1	1	524 263	1			0	0	0	0	0					OPEN								
Backend	0	0		0	0		0	0	52 427	0	0	0s	0	0	0	0		0	0	0	0	12s UP		0	0	0		0		

localnodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	0	-	0	0	524 263	0			0	0	0	0	0					OPEN								

nodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
s1	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	8s UP	L7OK/200 in 2ms	1	Y	-	1	1	4s	-
s2	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	12s UP	L7OK/200 in 6ms	1	Y	-	0	0	0s	-
Backend	0	0		0	0		0	0	52 427	0	0	?	0	0	0	0		0	0	0	0	12s UP		2	2	0		0	0s	

[2.] *Give the URL of your repository URL in the lab report*

https://github.com/NicolasOgi/Teaching-HEIGVD-AIT-2020-Labo-Docker

Tâche 1 : Add a process supervisor to run several processes

[1.] *Take a screenshot of the stats page of HAProxy at http://192.168.42.42:1936. You should see your backend nodes. It should be really similar to the screenshot of the previous task.*

stats

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				1	1	-	1	3	524 263	1			0	0	0	0	0					OPEN								
Backend	0	0		0	0		0	0	52 427	0	0	0s	0	0	0	0		0	0	0	0	20s UP		0	0	0		0		

localnodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	1	-	0	3	524 263	1			482	500	0	0	0					OPEN								

nodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
s1	0	0	-	0	1		0	1	-	1	1	8s	482	500		0		0	0	0	0	20s UP	L7OK/200 in 4ms	1	Y	-	0	0	0s	-
s2	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	20s UP	L7OK/200 in 8ms	1	Y	-	0	0	0s	-
Backend	0	0		0	1		0	1	52 427	1	1	8s	482	500	0	0		0	0	0	0	20s UP		2	2	0		0	0s	

[2.] *Describe your difficulties for this task and your understanding of what is happening during this task. Explain in your own words why we are installing a process supervisor. Do not hesitate to do more research and to find more articles on that topic to illustrate the problem*

Nous n'avons pas rencontré de difficultés particulières lors de cette tâche. Nous avons simplement mis en place un superviseur de processus afin de pallier au comportement de Docker qui définit qu'un seul processus est exécuté par conteneur.
Ainsi, avec S6, il est possible de choisir quel processus redémarrer sans que le conteneur ne s'arrête. Cela fonctionne car S6 est le processus qui s'exécute au premier-plan et met les autres en arrière-plan ce qui empêche le conteneur de s'arrêter si un des processus devait crasher.

Tâche 2 : Add a tool to manage membership in the web server cluster

[1.] *Provide the docker log output for each of the containers: ha, s1 and s2. You need to create a folder logs in your repository to store the files separately from the lab report. For each lab task create a folder and name it using the task number. No need to create a folder when there are no logs.*

Les logs sont disponibles dans le dossier logs/task2.

[2.] *Give the answer to the question about the existing problem with the current solution.*

Le problème avec la solution actuelle est que les nœuds s1 et s2 s'enregistrent au cluster uniquement en passant par ha. Cela veut dire que si le conteneur HAProxy n'est pas démarré, s1 et s2 ne pourront pas rejoindre le cluster. Cela va à l'encontre du principe de Serf qui est normalement censé donner la possibilité aux machines de rejoindre le cluster en passant par n'importe quelle autre machine au lieu d'une seule.

[3.] *Give an explanation on how Serf is working. Read the official website to get more details about the GOSSIP protocol used in Serf. Try to find other solutions that can be used to solve similar situations where we need some auto-discovery mechanism.*

Un agent Serf va rejoindre le cluster à travers le load-balancer pour autant que celui-ci soit accessible. Lorsqu'un nœud rejoint ou quitte le cluster, Serf utilise le protocole GOSSIP pour informer les autres nœuds de l'événement.
Il existe d'autres solutions comme etcd ou encore Zookeeper qui permettent de découvrir automatiquement les nœuds et d'évaluer leur état dans un cluster.

Tâche 3 : React to membership changes

[1.] *Provide the docker log output for each of the containers: ha, s1 and s2. Put your logs in the logs directory you created in the previous task.*

Les logs sont disponibles dans le dossier logs/task3.

[2.] *Provide the logs from the ha container gathered directly from the /var/log/serf.log file present in the container. Put the logs in the logs directory in your repo.*

Les logs sont disponibles dans le dossier logs/task3. Le fichier concerné se nomme **ha_serf.log**.

Tâche 4 : Use a template engine to easily generate configuration files

[1.] *You probably noticed when we added xz-utils, we had to rebuild the whole image which took some time. What can we do to mitigate that? Take a look at the Docker documentation on image layers. Tell us about the pros and cons to merge as much as possible of the command.*

Avec la première version, chaque commande RUN va créer un nouveau layer, alors qu'avec la seconde, il n'y a qu'un seul layer créé. Cela veut dire que lorsque nous avons ajouté xz-utils, il a fallu reconstruire le layer existant complètement au lieu de juste en créer un nouveau par-dessus avec une nouvelle commande RUN.

La première version est plus lourde que la deuxième mais est plus avantageuse lorsque plusieurs conteneurs partagent la même image. Un nouveau conteneur va ainsi profiter des layers créés pour se lancer plus rapidement qu'avec la deuxième version.

There are also some articles about techniques to reduce the image size. Try to find them. They are talking about squashing or flattening images.

Articles:

- https://mresetar.github.io/2020-03-20-squashing-docker-images-for-smaller-size/
- https://www.closeit.co/closeit-blog/flattening-docker-images

[2.] *Propose a different approach to architecture our images to be able to reuse as much as possible what we have done. Your proposition should also try to avoid as much as possible repetitions between your images.*

Il faudrait commencer le Dockerfile en se basant sur une image qui regroupe la plupart des instructions en commun. Les images haproxy et webapp construiront ensuite leur image en se basant sur l'image en commun à l'aide du mot-clé **FROM**.

[3.] *Provide the /tmp/haproxy.cfg file generated in the ha container after each step. Place the output into the logs folder like you already did for the Docker logs in the previous tasks. Three files are expected. In addition, provide a log file containing the output of the docker ps console and another file (per container) with docker inspect <container>. Four files are expected.*

Les fichiers de configuration haproxy ainsi que les logs se trouvent dans le dossier logs/task4.

[4.] *Based on the three output files you have collected, what can you say about the way we generate it? What is the problem if any?*

Le problème est qu'à chaque nouvel événement, le fichier de configuration est généré à nouveau en écrasant le contenu précédent.

Tâche 5 : Generate a new load balancer configuration when membership changes

[1.] *Provide the file /usr/local/etc/haproxy/haproxy.cfg generated in the ha container after each step. Three files are expected. In addition, provide a log file containing the output of the docker ps console and another file (per container) with docker inspect <container>. Four files are expected.*

Les fichiers se trouvent dans le dossier logs/task5/1.

[2.] *Provide the list of files from the /nodes folder inside the ha container. One file expected with the command output.*

Les fichiers se trouvent dans le dossier logs/task5/2.

[3.] *Provide the configuration file after you stopped one container and the list of nodes present in the /nodes folder. One file expected with the command output. Two files are expected. In addition, provide a log file containing the output of the docker ps console. One file expected.*

Les fichiers se trouvent dans le dossier logs/task5/3.

Tâche 6 : Make the load balancer automatically reload the new configuration

[1.] *Take screenshots of the HAProxy stat page showing more than 2 web applications running. Additional screenshots are welcome to see a sequence of experimentations like shutting down a node and starting more nodes. Also provide the output of docker ps in a log file. At least one file is expected. You can provide one output per step of your experimentation according to your screenshots.*

4 webapps running

nodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
935dd39a9fde	0	0	-	0	1		0	1	-	3	3	13s	1 936	1 508		0		0	0	0	0	1m35s UP	L7OK/200 in 4ms	1	Y	-	0	0	0s	-
a02b1f2450a4	0	0	-	0	1		0	1	-	3	3	6s	1 940	1 502		0		0	0	0	0	1m35s UP	L7OK/200 in 2ms	1	Y	-	0	0	0s	-
da878cedc9de	0	0	-	0	1		0	1	-	3	3	4s	1 934	1 492		0		0	0	0	0	1m35s UP	L7OK/200 in 4ms	1	Y	-	0	0	0s	-
f2d6c6f51b1f	0	0	-	0	1		0	1	-	3	3	3s	1 936	1 494		0		0	0	0	0	1m35s UP	L7OK/200 in 6ms	1	Y	-	0	0	0s	-
Backend	0	0		0	2		0	1	52 427	12	12	3s	7 746	5 996	0	0		0	0	0	0	1m35s UP		4	4	0		0	0s	

Le fichier contenant le résultat de la commande docker ps se trouve dans le dossier logs/task6.

a02b1f2450a4 (s2) shut down

nodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
935dd39a9fde	0	0	-	0	1		0	1	-	1	1	6s	650	502		0		0	0	0	0	16s UP	L7OK/200 in 5ms	1	Y	-	0	0	0s	-
da878cedc9de	0	0	-	0	1		0	1	-	1	1	4s	646	500		0		0	0	0	0	16s UP	L7OK/200 in 3ms	1	Y	-	0	0	0s	-
f2d6c6f51b1f	0	0	-	0	1		0	1	-	1	1	3s	648	502		0		0	0	0	0	16s UP	L7OK/200 in 2ms	1	Y	-	0	0	0s	-
Backend	0	0		0	1		0	1	52 427	3	3	3s	1 944	1 504	0	0		0	0	0	0	16s UP		3	3	0		0	0s	

Le fichier contenant le résultat de la commande docker ps se trouve dans le dossier logs/task6.

[2.] *Give your own feelings about the final solution. Propose improvements or ways to do things differently. If any, provide references to your readings for the improvements.*

La solution finale obtenue à la fin de ce laboratoire résout bien les problèmes évoqués au début mais n'est tout de même pas entièrement dynamique. Avec un reverse proxy proposé par Traefik, on obtient une solution équivalente voire meilleure et plus simplement mise en place.

Difficultés

Nous n’avons pas rencontré de grosses difficultés, mise à part quelques erreurs de copie de commandes de notre part qui nous ont fait perdre un peu de temps à déboguer lorsque le résultat attendu n’était pas atteint.

Conclusion

Pour conclure, nous avons su mettre en place une infrastructure semi-dynamique contenant un reverse-proxy servant de load-balancer ainsi que des conteneurs de backend sans grandes difficultés, nous avons également pu nous familiariser avec le principe de protocole de GOSSIP et consolider notre base concernant les conteneurs Docker.