Lab 04 - Docker

Besseau L onard, Gamboni Fiona, Pellissier David

Table of contents

Introduction

The main objective of this lab is to deepen the concept of load balancing, studied in the previous lab, to obtain a d namic load balancing infrastructure.

Task 0: Identify issues and install the tools

1. **[M1]** Do   ou think   e can use the current solution for a production environment? What are the main problems   hen deplo ing it in a production environment?

No. The main problem is that there is no   a   of d namicall   scaling the number of servers   ith the load balancer as the servers information is hard-coded in the settings file. We   ill al a s use the same number of servers no matter the load. In addition   ith the bug in the application, man   clients   ill lose their session and   ill have to restart a ne   session losing their progress.

2. **[M2]** Describe   hat   ou need to do to add a ne   ebapp container to the infrastructure. Give the e act steps of   hat   ou have to do   ithout modif ing the   a   the things are done. Hint: You probabl   have to modif   some configuration and script files in a Docker image.

Add the ne server to the docker file b adding a ne entr for the ne   ebserver and modif the hapro   entr to add the address of the ne server.

```
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}
```

Modif the .env file to add the ne server information

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

Add the ne server s name to the hapro .cfg file to add a ne server to the pool.

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

Rebuild the Docker image(s) and re-create the container to appl the ne config.

As e can see this is quite a length process to add a ne server and it involves a lot of do ntime for the application as e have to stop hapro to modif the config.

3. **[M3]** Based on our previous ans ers, ou have detected some issues in the current solution. No propose a better approach at a high level.

We could use a script to allo for automatic creation and removal of ne servers and modif hapro to d namicall manage the server list (detecting ne servers, removing offline servers from list). In the best scenario, hapro ould be able to use the script to adapt the number of servers online to the load.

4. **[M4]** You probabl noticed that the list of eb application nodes is hardcoded in the load balancer configuration. Ho can e manage the eb app nodes in a more d namic fashion?

The ne l created server ould automaticall announce themselves to the hapro server hen the are read to function. hapro could then use a periodical health-check to verif that the servers are still running and remove them from the list if the check fails.

5. **[M5]** In the ph sical or virtual machines of a t pical infrastructure e tend to have not onl one main process (like the eb server or the load balancer) running, but a fe additional processes on the side to perform management tasks.

   For e ample to monitor the distributed s stem as a hole it is common to collect in one centrali ed place all the logs produced b the different machines. Therefore e need a process running on each machine that ill for ard the logs to the central place. (We could also imagine a central tool that reaches out to each machine to gather the logs. That's a push vs. pull problem.) It is quite common to see a push mechanism used for this kind of task.

   Do ou think our current solution is able to run additional management processes beside the main eb server / load balancer process in a container? If not, hat is missing / required to reach the goal? If es, ho to proceed to run for e ample a log for arding process?

Yes our solution could run additional management processes besides the load balancer ith the use of daemon. This is not recommended b docker but is the simplest solution, as e ould have alread implemented the push s stem to update the hapro server list.

Another solution ould be to add another container for managing all the logs. This ould allo us to separate areas of concerns.

To for ard the logs, e could use logging protocols such as s slog, hich is alread compatible ith hapro .

6. **[M6]** In our current solution, although the load balancer configuration is changing d namicall , it doesn't d namicall follo the configuration of our distributed s stem hen eb servers are added or removed. If e take a closer look at the run.sh script, e see t o calls to sed hich ill replace t o lines in the hapro .cfg configuration

file just before e start hapro . You clearl see that the configuration file has t o lines and the script ill replace these t o lines.

What happens if e add more eb server nodes? Do ou think it is reall d namic? It's far a a from being a d namic configuration. Ca [illegible] ?

supervisor. Do not hesitate to do more research and to find more articles on that topic to illustrate the problem.

We carefull follo ed the instructions so   e did not face an difficulties during this task.

We are using a process supervisor in order to be able to run multiple processes in the same container. This is not a trivial task due to the docker philosoph   hich believes that a container should be used b a single process. To avoid this,   e use S6 as the main process   hich   ill manage all the other processes   e might   ant to run in the container. This follo s the S6 philosoph of one thing per container.

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

In the Docker images files,   e added the follo ing command to cop the Serf agent run script and to make it e ecutable:

```
COPY services/serf /etc/services.d/serf
RUN chmod +x /etc/services.d/serf/run
```

1. Provide the docker log output for each of the containers: ha, s1 and s2. You need to create a folder logs in our repositor to store the files separatel from the lab report. For each lab task create a folder and name it using the task number. No need to create a folder   hen there are no logs.

The logs can be found in the logs/task2 folder.

2. Give the ans er to the question about the e isting problem   ith the current solution.

With the current solution, the nodes all have to be registered through the hapro   container   hich creates a single point of failure. If the hapro   container is not available, no ne   server can join the cluster. This is not   hat Serf is made for, as Serf is designed to allo   joining a cluster from multiple machines and not one.

3. Give an e planation on ho   Serf is   orking. Read the official   ebsite to get more details about the GOSSIP protocol used in Serf. Tr to find other solutions that can be used to solve similar situations   here   e need some auto-discover mechanism.

A Serf agent   ill contact the load balancer to join the cluster. If the cluster does not alread e ist, it   ill be created at this point. At this stage, the ha pro   container is essential other ise, the startup for the serf agent   ill fail as it could not join the cluster.
To inform the members of the cluster of the ne   composition of the cluster (members arriving or leaving), Serf   ill use the GOSSIP protocol to broadcast the information to the cluster. The GOSSIP protocol is based on the SWIM protocol and   as modified to increase propagation and converge rate.

An alternative could be the solution used b Traefik. It uses the docker API to detect running containers and the metadata of these containers to identif their services.

In our case, the ha container could detect ever container ith a custom label like WEBAPP and use it as nodes in his load-balancing.

Task 3: React to membership changes

In the Docker images files, e added the follo ing command to cop the scripts responsible to log members that join or leave the cluster :

```
RUN mkdir -p /serf-handlers
COPY scripts/member-join.sh /serf-handlers/member-join.sh
COPY scripts/member-leave.sh /serf-handlers/member-leave.sh
RUN chmod +x /serf-handlers/*
```

1. Provide the docker log output for each of the containers: ha, s1 and s2. Put our logs in the logs director ou created in the previous task.

The logs can be found in the *logs/task3* folder. The logs corresponding to each moment are:

- Ha started: haonl (ha log)
- S1 started: ha+s1 (ha log) + s1 (s1 log)
- S2 started: ha+s1+s2(ha log) + s2 (s2 log) + s1Withs2 (s1 log)

2. Provide the logs from the ha container gathered directl from the /var/log/serf.log file present in the container. Put the logs in the logs director in our repo.

The logs can be found in the *logs/task3* folder under the name *serf.log*.

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

In the Docker images files, e added the follo ing command to cop the hapro configuration template in */config* :

```
RUN mkdir -p /config
COPY config/haproxy.cfg.hb /config/haproxy.cfg.hb
```

1. You probabl noticed hen e added -utils, e have to rebuild the hole image hich took some time. What can e do to mitigate that? Take a look at the Docker documentation on [image la ers](). Tell us about the pros and cons to merge as much as possible of the command. In other ords, compare:

   RUN command 1
   RUN command 2
   RUN command 3
   vs.
   RUN command 1 && command 2 && command 3

Each RUN instruction line ill create a ne read-onl la er hich ill increase the image si e but ill allo rebuilding the image quicker as it can be cached and not be rebuilt. Ho ever once the image is read , the best practice is to minimi e the number of la ers.

> There are also some articles about techniques to reduce the image si e. Tr to find them. The are talking about squashing or flattening images.

Squashing is a technique that squash multiple docker la ers into one to create an image ith fe er and smaller la ers. More information can be found here.

Flattening consists of creating an image from a running container ith all the la ers to onl have the final la er in the ne image. More information can be found here.

2. Propose a different approach to architecture our images to be able to reuse as much as possible hat e have done. Your proposition should also tr to avoid as much as possible repetitions bet een our images.

Both our images need NodeJS and have common packages. We could create an image containing the shared la er bet een all our images and then add the necessar la er to the base image. On the other hand, an change in the base image ould force rebuilding all the depending images.

3. Provide the /tmp/hapro .cfg file generated in the ha container after each step. Place the output into the logs folder like ou alread did for the Docker logs in the previous tasks. Three files are e pected.

The outputs for the template are in the *logs/task4* folder. In order there is :

- hapro 1.cfg
- hapro 2.cfg
- hapro 3.cfg

> In addition, provide a log file containing the output of the docker ps console and another file (per container) ith docker inspect <container>. Four files are e pected.

The files are in the *logs/task4* folder. The files are *ps* for the ps file and *ha*, *s1* and *s2* for the container details.

4. Based on the three output files ou have collected, hat can ou sa about the a e generate it? What is the problem if an ?

The content of the file is over ritten ever time a ne container joins the cluster.

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

All the files are located in the *logs/task5* folder.

1. Provide the file /usr/local/etc/hapro   /hapro   .cfg generated in the ha container after each step. Three files are e pected.

The files are *noNode*, *1node* and *2node*.

In addition, provide a log file containing the output of the docker ps console and another file (per container)   ith docker inspect <container>. Four files are e pected.

The files are called ps for the ps file and ha, s1 and s2 for the container details.

2. Provide the list of files from the /nodes folder inside the ha container. One file e pected   ith the command output.

The file is *lsoutput*.

3. Provide the configuration file after   ou stopped one container and the list of nodes present in the /nodes folder. One file e pected   ith the command output. T   o files are e pected.

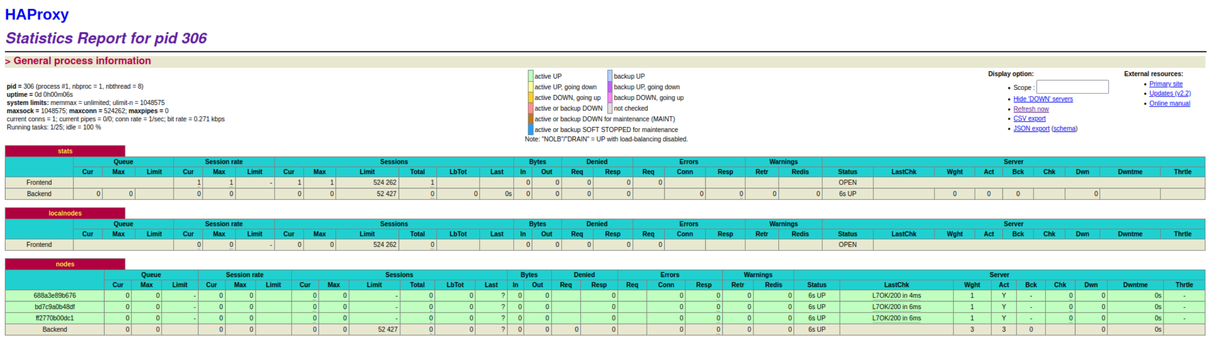
The ls output is *lsoutput1* and the ha iconfig is *1stopped.cfg*

In addition, provide a log file containing the output of the docker ps console. One file e pected.

The file is *ps1*.

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

1. Take a screenshots of the HAPro   stat page sho   ing more than 2   eb applications running. Additional screenshots are   elcome to see a sequence of e perimentations like shutting do   n a node and starting more nodes.



HAProxy

Statistics Report for pid 306

> General process information

pid = 306 (process #1, nbproc = 1, nbthread = 8)
uptime = 0d 0h00m06s
system limits: memmax = unlimited; ulimit-n = 1048575
maxsock = 1048575; **maxconn =** 524262; **maxpipes =** 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec; bit rate = 0.271 kbps
Running tasks: 1/25; idle = 100 %

🟩 active UP	🟦 backup UP
🟨 active UP, going down	🟪 backup UP, going down
🟧 active DOWN, going up	🟪 backup DOWN, going up
🟥 active or backup DOWN	⬜ not checked
🟫 active or backup DOWN for maintenance (MAINT)	
🟦 active or backup SOFT STOPPED for maintenance	

Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:
- Scope :
- Hide 'DOWN' servers
- Refresh now
- CSV export
- JSON export (schema)

External resources:
- Primary site
- Updates (v2.2)
- Online manual

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	524 262	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	6s UP		0	0	0		0		

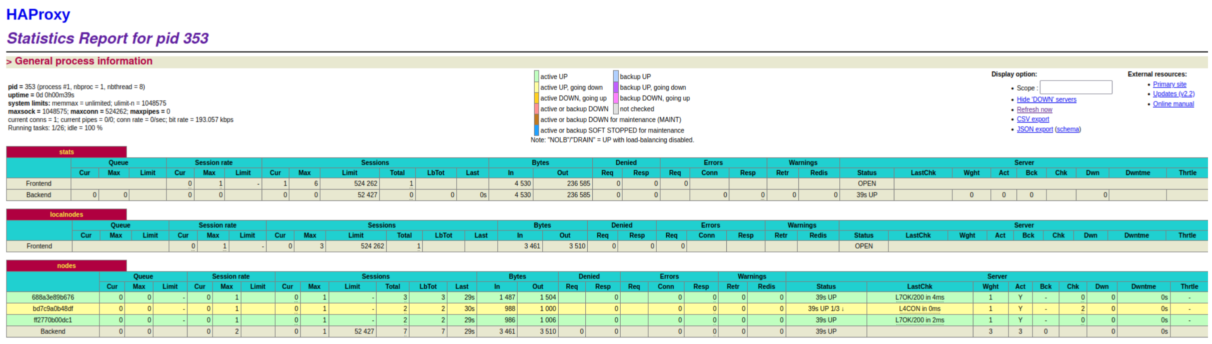
localnodes

	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	524 262	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
688a3e89b676	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	6s UP	L7OK/200 in 4ms	1	Y	-	0	0	0s	-
bd7c9a0b48df	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	6s UP	L7OK/200 in 6ms	1	Y	-	0	0	0s	-
ff2770b00dc1	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	6s 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	6s UP		3	3	0		0	0s	

The hapro   page   ith 3 instances.



HAProxy

Statistics Report for pid 353

> General process information

pid = 353 (process #1, nbproc = 1, nbthread = 8)
uptime = 0d 0h00m39s
system limits: memmax = unlimited; ulimit-n = 1048575
maxsock = 1048575; **maxconn =** 524262; **maxpipes =** 0
current conns = 1; current pipes = 0/0; conn rate = 0/sec; bit rate = 193.057 kbps
Running tasks: 1/26; idle = 100 %

🟩 active UP	🟦 backup UP
🟨 active UP, going down	🟪 backup UP, going down
🟧 active DOWN, going up	🟪 backup DOWN, going up
🟥 active or backup DOWN	⬜ not checked
🟫 active or backup DOWN for maintenance (MAINT)	
🟦 active or backup SOFT STOPPED for maintenance	

Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:
- Scope :
- Hide 'DOWN' servers
- Refresh now
- CSV export
- JSON export (schema)

External resources:
- Primary site
- Updates (v2.2)
- Online manual

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				0	1	-	1	6	524 262	1			4 530	236 585	0	0	0					OPEN								
Backend	0	0		0	0		0	0	52 427	0	0	0s	4 530	236 585	0	0		0	0	0	0	39s 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	524 262	1			3 461	3 510	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
688a3e89b676	0	0	-	0	1		0	1	-	3	3	29s	1 487	1 504		0		0	0	0	0	39s UP	L7OK/200 in 4ms	1	Y	-	0	0	0s	-
bd7c9a0b48df	0	0	-	0	1		0	1	-	2	2	30s	988	1 000		0		0	0	0	0	39s UP 1/3 ↓	L4CON in 0ms	1	Y	-	2	0	0s	-
ff2770b00dc1	0	0	-	0	1		0	1	-	2	2	29s	986	1 006		0		0	0	0	0	39s UP	L7OK/200 in 2ms	1	Y	-	0	0	0s	-
Backend	0	0		0	2		0	1	52 427	7	7	29s	3 461	3 510	0	0		0	0	0	0	39s UP		3	3	0		0	0s	

One instance stopped.

> Also provide the output of docker ps in a log file. At least one file is e pected. You can provide one output per step of   our e perimentation according to   our screenshots.

The log for the *ps*   ith 3 nodes is in *logs/task6/ps1*.
The log for the ps   ith 2 nodes up is in *logs/task6/ps2.*

2. Give   our o   n feelings about the final solution. Propose improvements or   a s to do things differentl . If an , provide references to   our readings for the improvements.

The solution implemented is good and might be a good first base to use for a production deplo ment. One aspect that could be improved is the creation of ne   nodes   hich still need to specif   the ip for each node.

Difficulties

We didn t encounter difficulties during this lab because the instructions   ere straightfor   ard and   ell documented.

Conclusion

This lab taught us a   a   to set up a d namic infrastructure. It allo   ed us to discover useful tools such as Serf and techniques to reduce the si e of Docker images on the   a .