Lab 04 - Docker

Besseau Léonard, 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 dynamic load balancing infrastructure.

Task 0: Identify issues and install the tools

1. **[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?

No. The main problem is that there is no way of dynamically scaling the number of servers with the load balancer as the servers information is hard-coded in the settings file. We will always use the same number of servers no matter the load. In addition with the bug in the application, many clients will lose their session and will have to restart a new session losing their progress.

2. **[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.

- Add the new server to the docker file by adding a new entry for the new webserver and modify the haproxy entry to add the address of the new 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}
```

- Modify the .env file to add the new server information

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

- Add the new server's name to the haproxy.cfg file to add a new server to the pool.

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

- Rebuild the Docker image(s) and re-create the container to apply the new config.

As we can see this is quite a lengthy process to add a new server and it involves a lot of downtime for the application as we have to stop haproxy to modify the config.

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

We could use a script to allow for automatic creation and removal of new servers and modify haproxy to dynamically manage the server list (detecting new servers, removing offline servers from list). In the best scenario, haproxy would be able to use the script to adapt the number of servers online to the load.

4. **[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?

The newly created server would automatically announce themselves to the haproxy server when they are ready to function. haproxy could then use a periodical health-check to verify that the servers are still running and remove them from the list if the check fails.

5. **[M5]** In the physical or virtual machines of a typical infrastructure we tend to have not only one main process (like the web server or the load balancer) running, but a few additional processes on the side to perform management tasks.

   For example to monitor the distributed system as a whole it is common to collect in one centralized place all the logs produced by the different machines. Therefore we need a process running on each machine that will forward 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 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?

Yes our solution could run additional management processes besides the load balancer with the use of daemon. This is not recommended by docker but is the simplest solution, as we would have already implemented the push system to update the haproxy server list.

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

To forward the logs, we could use logging protocols such as syslog, which is already compatible with haproxy.

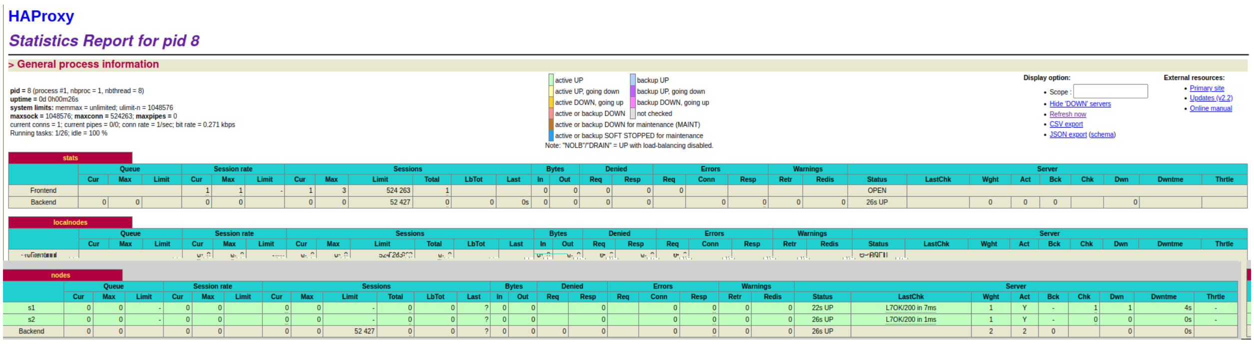
6. **[M6]** In our current solution, although the load balancer configuration is changing dynamically, it doesn't dynamically follow the configuration of our distributed system when web servers are added or removed. If we take a closer look at the run.sh script, we see two calls to sed which will replace two lines in the haproxy.cfg configuration file just before we start haproxy. You clearly see that the configuration file has two lines and the script will replace these two lines.

   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?

Nothing happens when we add more servers as haproxy cannot detect them and add them to the list. It’s only dynamic when starting the haproxy container by using the variables in the .env file. We could use a solution as described in M4.

Deliverables:

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

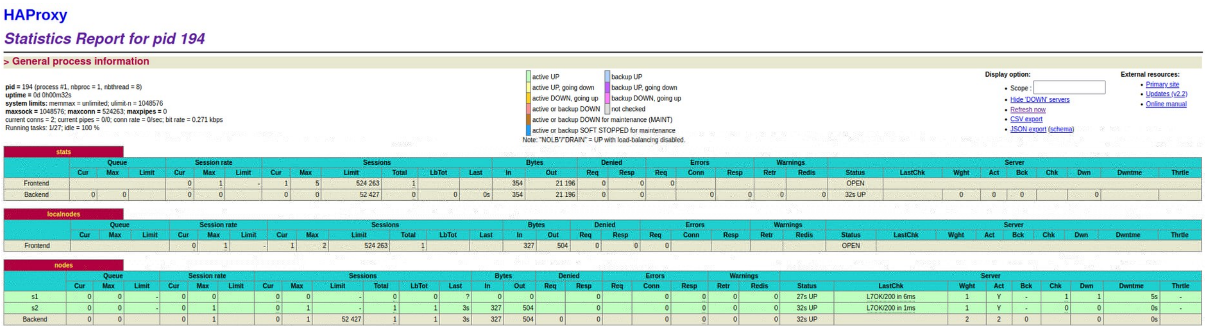


2. Give the URL of your repository URL in the lab report.

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

Task 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.



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

We carefully followed the instructions so we did not face any 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 philosophy which believes that a container should be used by a single process. To avoid this, we use S6 as the main process which will manage all the other processes we might want to run in the container. This follows the S6 philosophy of one thing per container.

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

In the Docker images files, we added the following command to copy the Serf agent run script and to make it executable:

```
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 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.

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

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

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

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.

A Serf agent will contact the load balancer to join the cluster. If the cluster does not already exist, it will be created at this point. At this stage, the ha proxy container is essential otherwise, the startup for the serf agent will fail as it could not join the cluster.
To inform the members of the cluster of the new composition of the cluster (members arriving or leaving), Serf will use the GOSSIP protocol to broadcast the information to the cluster. The GOSSIP protocol is based on the SWIM protocol and was modified to increase propagation and converge rate.

An alternative could be the solution used by Traefik. It uses the docker API to detect running containers and the metadata of these containers to identify their services.
In our case, the ha container could detect every container with 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, we added the following command to copy 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 your logs in the logs directory you created in the previous task.

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

- Ha started: haonly (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 directly from the /var/log/serf.log file present in the container. Put the logs in the logs directory in your 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, we added the following command to copy the haproxy configuration template in */config* :

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

1. You probably noticed when we added xz-utils, we have 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. In other words, compare:

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

Each RUN instruction line will create a new read-only layer which will increase the image size but will allow rebuilding the image quicker as it can be cached and not be rebuilt. However once the image is ready, the best practice is to minimize the number of layers.

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

Squashing is a technique that squash multiple docker layers into one to create an image with fewer and smaller layers. More information can be found [here].

Flattening consists of creating an image from a running container with all the layers to only have the final layer in the new 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 what we have done. Your proposition should also try to avoid as much as possible repetitions between your images.

Both our images need NodeJS and have common packages. We could create an image containing the shared layer between all our images and then add the necessary layer to the base image. On the other hand, any change in the base image would force rebuilding all the depending images.

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.

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

- haproxy1.cfg
- haproxy2.cfg
- haproxy3.cfg

- 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.

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 you have collected, what can you say about the way we generate it? What is the problem if any?

The content of the file is overwritten every time a new 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/haproxy/haproxy.cfg generated in the ha container after each step. Three files are expected.

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) with docker inspect <container>. Four files are expected.

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 expected with the command output.

The file is *lsoutput*.

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.

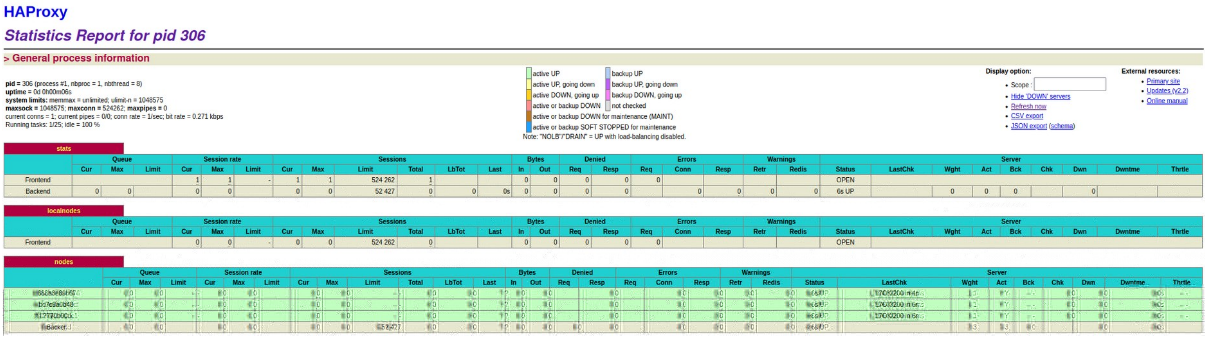
The ls output is *lsoutput1* and the ha iconfig is *1stopped.cfg*

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

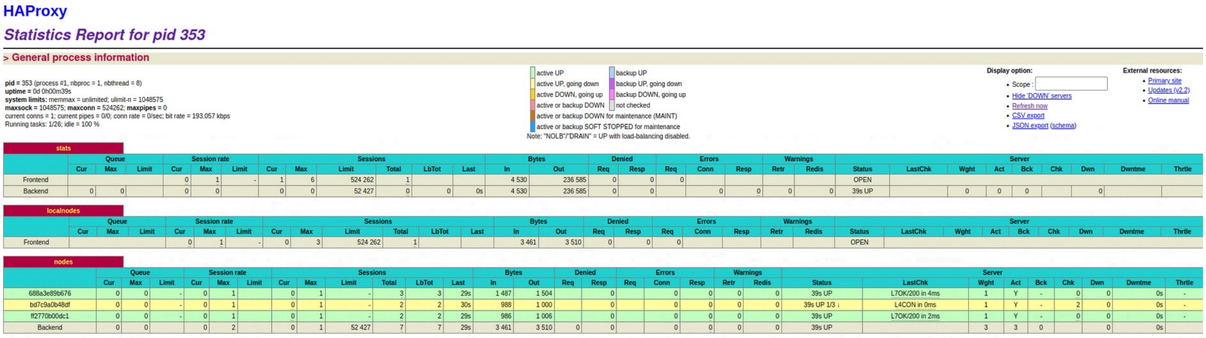
The file is *ps1*.

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

1. Take a 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.



The haproxy page with 3 instances.



One instance stopped.

- 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.

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

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.

The solution implemented is good and might be a good first base to use for a production deployment. One aspect that could be improved is the creation of new nodes which still need to specify the ip for each node.

Difficulties

We didn’t encounter difficulties during this lab because the instructions were straightforward and well documented.

Conclusion

This lab taught us a way to set up a dynamic infrastructure. It allowed us to discover useful tools such as Serf and techniques to reduce the size of Docker images on the way.