18/12/2017

AIT

LAB 04 - Docker

Wojciech Myszkorowski & Jérémie Zanone
HEIG-VD

Table of contents

Introduction

In this lab we are going to to build our own docker image as a load lalancer. This is the continuation of the previous lab to introduce a dynamic decentralized managed load balancer with horizontal scalability.

Chapter 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?

No, we can not use a static load balancer in a production environment. The main problem is that we want to be able to add or remove servers without changing all the static configuration. We need something to auto-managed the servers (dynamic load balancer).

[M2]

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

We need to add a new line in the configuration file ha/config/haproxy.cfg

> server s3 <s3>:3000 check

in the ha/scripts/run.sh file too:

> sed -i 's/<s3>/$S3_PORT_3000_TCP_ADDR/g' /usr/local/etc/haproxy/haproxy.cfg

Last step we need to rebuild the containers. Modify the script **start-containers.sh** to add a webserver container.

[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 have seen that for a new node we need to modify the configuration in several files. We need something to automatise these communications between the current nodes and the load balancer. The administrator just need to monitor them without taking care of the configuration file.

[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?

In order to be more dynamic we can use some scripts to automatize the new configuration each time a node is coming up or down. Moreover we need a tool that call these scripts when it detects some new or old node.

[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 no, what is missing / required to reach the goal? If yes, how to proceed to run for example a log forwarding process?

It will not work as it needs a supervisor in order to manage multiple process. The daemon is never turned off. It uses a system init. An init system is usually part of an operating system where it manages deamons and coordinates the boot process. we can notice the existence of several init systems, like init.d or systemd. Sometimes they are also called process supervisors.

[M6]

In our current solution, although the load balancer configuration is changing dynamically, it doesn't follow dynamically 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.

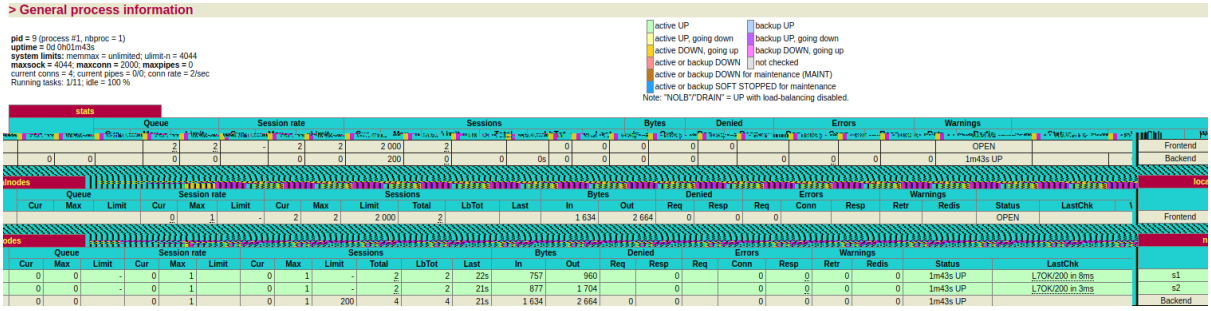
What happens if we add more web server nodes? Do you think it is really dynamic?

It's far away from being a dynamic configuration. Can you propose a solution to solve this?

The load balancer do not see them because the static configuration file is not dynamically updated when a new node is added (only configured for 2 servers). We have to implement a communication system between the server and the load balancer in order to announce what ishappenening.

Question 1

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



Question 2

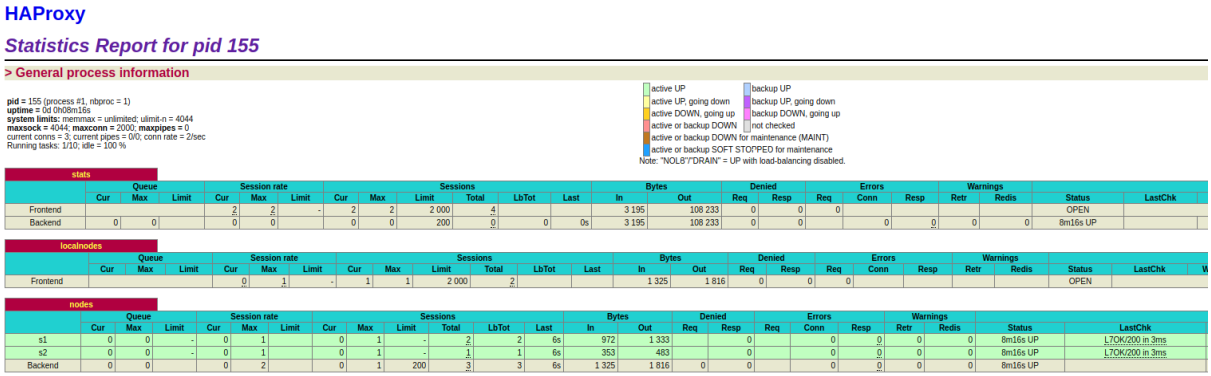
Give the URL of your repository URL in the lab report.

https://github.com/Grem25/Teaching-HEIGVD-AIT-2016-Labo-Docker

Chapter 1 - Add a process supervisor to run several processes

Question 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.



HAProxy

Statistics Report for pid 155

> General process information

pid = 155 (process #1, nbproc = 1)
uptime = 0d 0h08m16s
system limits: memmax = unlimited; ulimit-n = 4044
maxsock = 4044; **maxconn** = 2000; **maxpipes** = 0
current conns = 3; current pipes = 0/0; conn rate = 2/sec
Running tasks: 1/10; idle = 100 %

active UP
active UP, going down
active DOWN, going up
active or backup DOWN
active or backup DOWN for maintenance (MAINT)
active or backup SOFT STOPPED for maintenance
backup UP
backup UP, going down
backup DOWN, going up
not checked
Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

stats

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings			
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk
Frontend				2	2	-	2	2	2 000	4			3 195	108 233	0	0	0					OPEN	
Backend	0	0		0	0		0	0	200	0	0	0s	3 195	108 233	0	0		0	0	0	0	8m16s UP	

localnodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings			
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	W
Frontend				0	1	-	1	1	2 000	2			1 325	1 816	0	0	0					OPEN	

nodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings			
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk
s1	0	0	-	0	1		0	1	-	2	2	6s	972	1 333		0		0	0	0	0	8m16s UP	L7OK/200 in 3ms
s2	0	0	-	0	1		0	1	-	1	1	6s	353	483		0		0	0	0	0	8m16s UP	L7OK/200 in 3ms
Backend	0	0		0	2		0	1	200	3	3	6s	1 325	1 816	0	0		0	0	0	0	8m16s UP	

Question 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.

It is not a new docker container that are initializing. It is simply a new process that is called a supervisor. We can see it as a toolbox for low-level process and service administration, providing different sets of independent tools that can be used inside our container.

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

Question 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.

Example:

```
|-- root folder
  |-- logs
    |-- task 1
    |-- task 3
    |-- ...
```

```
vagrant@ubuntu-14:/vagrant$ cd logs/task\ 2/
vagrant@ubuntu-14:/vagrant/logs/task 2$ ls
ha  s1  s2
```

We have used the redirection to take the output from the logs of each container and we set them into files at the location asked.
“docker logs ha > “logs/task 2/ha” this is an example of the command that can provide us with a text file containing the logs.

Question 2

Give the answer to the question about the existing problem with the current solution. “describe which problem exists with the current solution based on the previous explanations and remarks. Propose a solution to solve the issue.”

The current problem is the fact that if one is started before the other it will not be able to resolve the connection with the other because he does not know it yet. For example if we start the nodes first (s1,s2) they will try to connect to the serf agent that is not running yet therefore it is unsolvable at this point.

The solution is to separately launch the containers without linking them directly in the docker run command. When Serf is coupled to the environment variables then they are used to dynamically configure the cluster when a node arrives or leaves the ship.

Question 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.

Each container with a running Serf agent talks to others using a decentralized peer-to-peer protocol (see next paragraph) to exchange information. They form a cluster of nodes. The main information they exchange is the existence of nodes in the cluster and what their IP addresses are. When a node appears or disappears the Serf agents tells the other about the event. A Serf agent can trigger the execution of local scripts when it receives an event.

Serf uses a gossip protocol to broadcast messages to the cluster. The gossip protocol is based on "SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol", with a few minor adaptations, mostly to increase propagation speed and convergence rate.

Task 3 - React to membership changes

Question 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.

See logs/task 3/ha_alone, logs/task3/ha_s1, logs/task3/ha_s2, logs/task3/s1, logs/task3/s2

Question 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.

See logs/task 3/serf.log

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

Question 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:

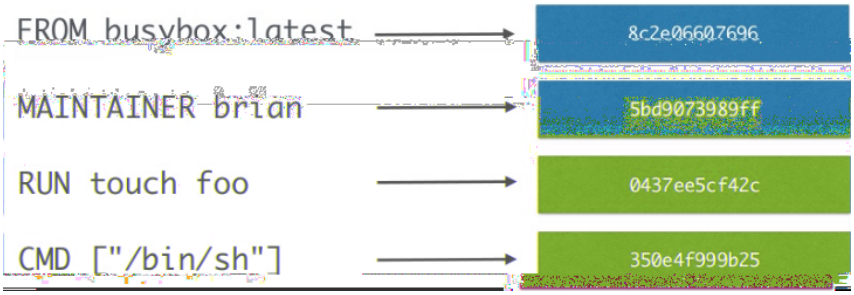
```
RUN command 1
RUN command 2
RUN command 3
```

vs.

```
RUN command 1 && command 2 && command 3
```

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

Each Dockerfile instruction generates a new layer:



Once the build is complete, Docker creates a new image loading the differences from each layer into a single new layer and references all the parent's layers.
In other words: when squashing, Docker will take all the filesystem layers produced by a build and collapse them into a single new layer.

This can simplify the process of creating minimal container images, but may result in slightly higher overhead when images are moved around (because squashed layers can no longer be shared between images). Docker still caches individual layers to make subsequent builds fast.

Question 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.

Start with FROM scratch and package only the bins/libs we need for our app. Changes to copied files will also invalidate image cache. Place the instructions least likely to change at the top of your Dockerfile, make changes/additions at the bottom. Place instructions you use across all of your images (MAINTAINER) at the top so they can be re-used across all images. Place .dockerignore in the root of build context with list of file/directory patterns to be excluded from build context.

Question 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.*

See logs/task 4/ha_proxy_cfg, logs/task 4/ha_with_s1, logs/task 4/ha_with_s2

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.*

See logs/task 4/docker_ps, logs/task 4/docker_inspect_h1, logs/task 4/docker_inspect_h2

Question 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?

It is not a very optimal way to monitor the nodes. Because we use “<” instead of “<<”, the previous log will be overwritten after each new incoming node.

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

Question 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.

The files of configuration can be found in this directory:

1. logs/task 5/haproxy_ha.cfg
2. logs/task 5/haproxy_s1.cfg
3. logs/task 57haproxy_s2.cfg

The files of inspection can be found in this directory:

1. logs/task 5/docker_inspect_ha
2. logs/task 5/docker_inspect_s1
3. logs/task 5/docker_inspect_s2
4. logs/task 5/docker_ps

Question 2

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

The file can be found in that directory:
logs/task 5/ha_nodes

Question 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.

The files can be found in this directory:

1. logs/task 5/ha_s1_killed_nodes
2. logs/task 5/docker_ps_s1_killed

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

Question 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.

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.

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

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
Frontend				2	2	-	2	2	2 000	2			0	0	0	0	0					OPEN					
Backend	0	0		0	0		0	0	200	0	0	0s	0	0	0	0		0	0	0	0	1m18s UP		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
Frontend				0	3	-	0	3	2 000	3			910	1 753	0	0	2					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
47571f48d8d6	0	0	-	0	1		0	1	-	1	1	1m15s	437	479		0		0	0	0	0	1m18s UP	L7OK/200 in 4ms	1	Y
e01eb2990268	0	0	-	0	1		0	1	-	1	1	1m14s	473	850		0		0	0	0	0	1m18s UP	L7OK/200 in 3ms	1	Y
Backend	0	0		0	1		0	1	200	2	2	1m14s	910	1 329	0	0		0	0	0	0	1m18s UP		2	2

File: /logs/task 6/docker_ps

Question 2

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

This solution works well and follows the specification of a dynamic load balancer. However, it is dependent on a complex docker image built with the attachement of several scripts and use of external framework for its implementation.

here is an article about an integrated Docker load balancer:

https://docs.docker.com/docker-cloud/apps/load-balance-hello-world/

Difficulties

Sometimes it was hard to find the information we needed moreover some questions were hard to understand.

Conclusion

During this lab we had the opportunity to continue to play and discover more about the world of HAProxy but this time paired with Docker and others powerful tool like Serf and Handlebars.

We found the lab particularly long and quite annoying since the manipulations were copy/paste code and then answer more or less complex questions that pushed us into large documentations in order to gather informations . However, it was useful for us to understand how a dynamic load balancer works.