

AIT - Laboratoire 3 - Load Balancing

Authors: Olivier Koffi, Nathanaël Mizutani

Introduction

In this lab, we will use HAProxy in different cases.

At first, we will test the proxy with default configurations, that means round-robin mode without sticky-sessions

Then, we will configure the proxy to manage sticky-sessions.

Then, we will experiment the DRAIN and MAINT mode.

Finally, we will configure and compare different balancing modes.

Task 1: Install the tools

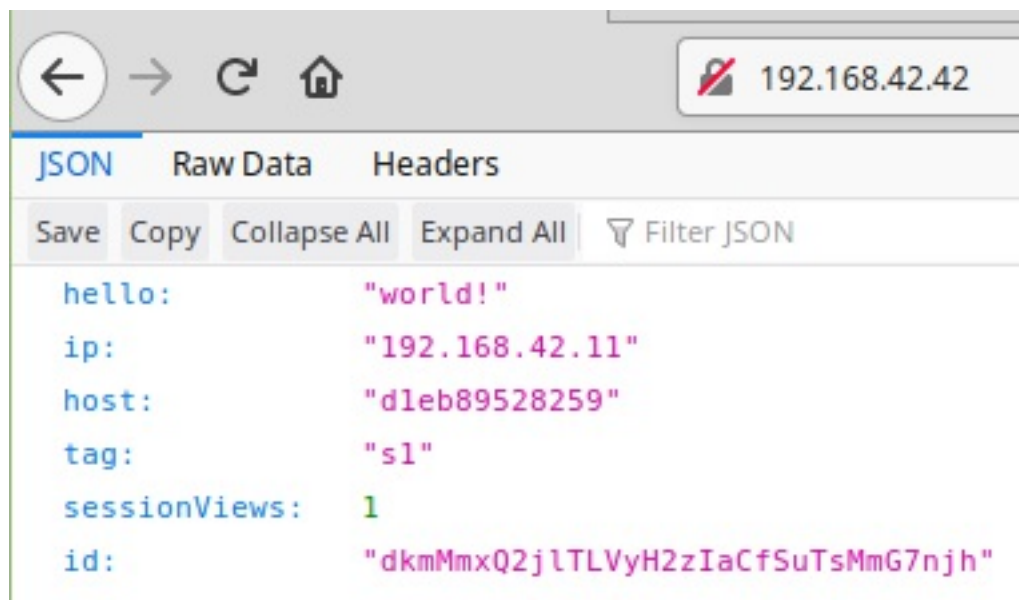
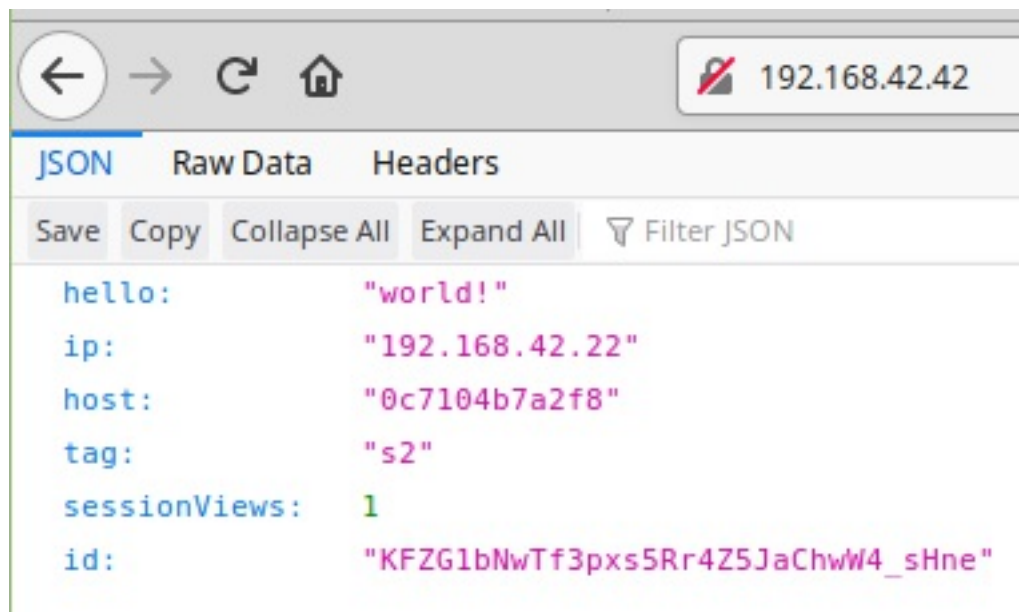
1. Explain how the load balancer behaves when you open and refresh the URL <http://192.168.42.42> in your browser. Add screenshots to complement your explanations. We expect that you take a deeper a look at session management.

Answer

When we refresh the browser on the URL <http://192.168.42.42> we can see that the request is send alternatively on each web servers. HA Proxy seems to work with a round robin configuration by default.

We also can see that the NODESESSID is refreshed for each request. This is because each server receives the token from the other and doesn't recognize it. So it sets a new cookie containing a new NODESESSID. Then, the other server do the same, etc...

In fact, we have a setup that demands sticky sessions but HA Proxy is configured as a stateless mode by default.



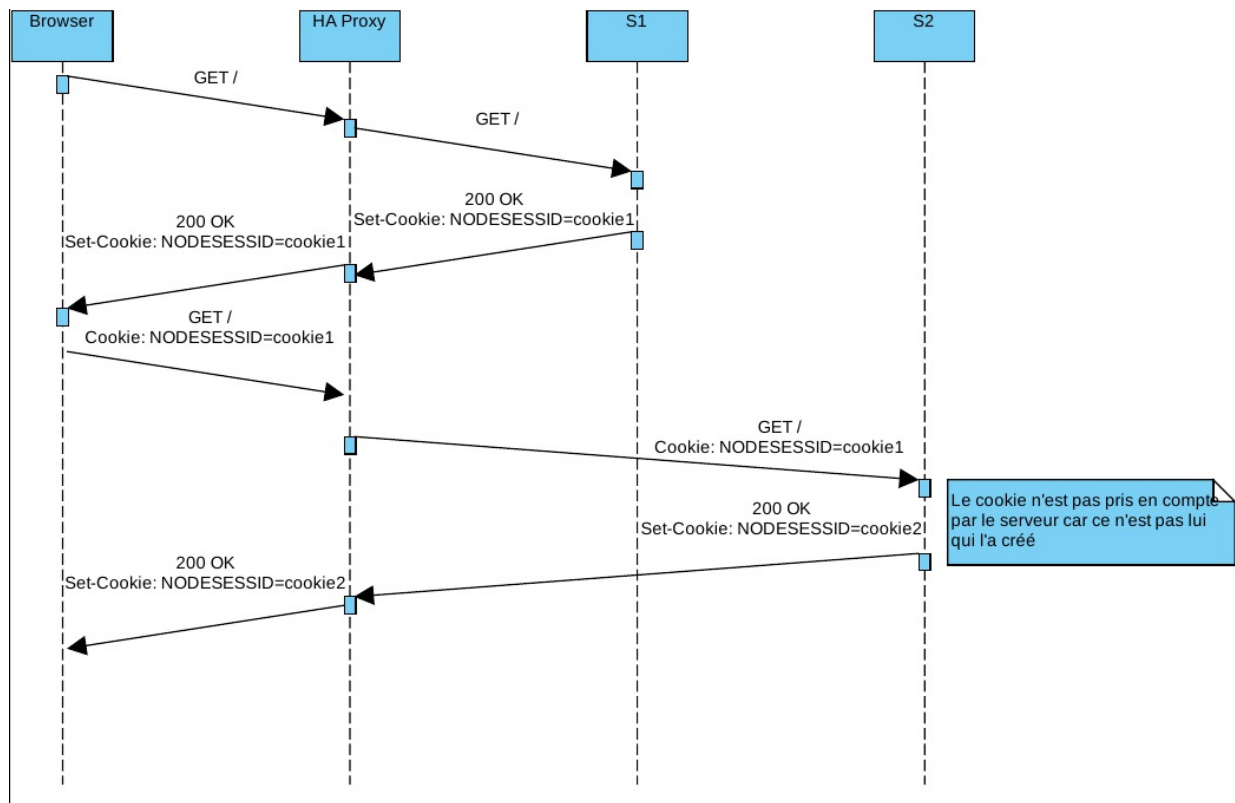
2. Explain what should be the correct behavior of the load balancer for session management.

Answer

HA Proxy should be configured to be statefull. That means it should forwards existing sessions on the correct server. This is what we call sticky sessions.

3. Provide a sequence diagram to explain what is happening when one requests the URL for the first time and then refreshes the page. We want to see what is happening with the cookie. We want to see the sequence of messages exchanged (1) between the browser and HAProxy and (2) between HAProxy and the nodes S1 and S2.

Answer



4. Provide a screenshot of the summary report from JMeter.

Answer

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	1000	1	1	8	0.64	0.00%	231.4/sec	112.00	49.15	495.6
S2 reached	500	0	0	1	0.36	0.00%	118.7/sec	0.00	0.00	0
S1 reached	500	0	0	1	0.37	0.00%	119.0/sec	0.00	0.00	0
TOTAL	2000	1	0	8	1.02	0.00%	462.7/sec	111.98	49.14	247.8

5. Run the following command:

```
$ docker stop s1
```

Clear the results in JMeter and re-run the test plan. Explain what is happening when only one node remains active. Provide another sequence diagram using the same model as the previous one.

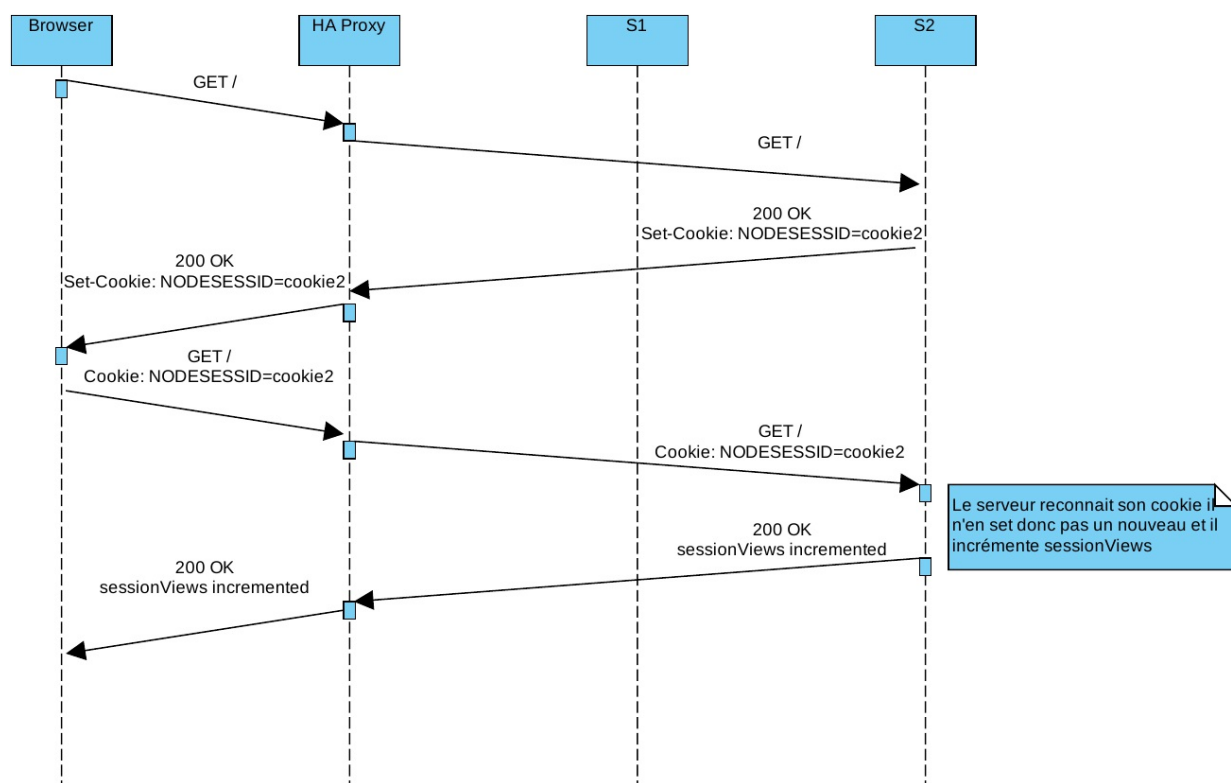
Answer

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	1000	4	1	20	2.13	0.00%	103.5/sec	37.60	21.72	372.0
S2 reached	1000	0	0	9	0.75	0.00%	105.4/sec	0.00	0.00	.0
TOTAL	2000	2	0	20	2.45	0.00%	207.0/sec	37.59	21.72	186.0

Now HA Proxy forwards all traffic on S2 because it's the only server up.

S2 now recognizes the cookie NODESESSID it set so it doesn't need to set a new cookie.

The session is established, so the sessionViews is incremented at each new request but it's not because of HA Proxy is configured to manage sticky sessions, it's only because there is only one server up so all the traffic is always forward there.



Task 2: Sticky sessions

1. There is different way to implement the sticky session. One possibility is to use the SERVERID provided by HAProxy. Another way is to use the NODESESSID provided by the application. Briefly explain the difference between both approaches (provide a sequence diagram with cookies to show the difference).

Answer

With SERVERID, HA Proxy is configured to add a cookie (named SERVERID) in the server response to identify the server. When the client send another request, the proxy read the SERVERID cookie and forwards it to the correct server. HA Proxy inserts SERVERID cookie in the

server's response only if it doesn't exist in the client's request, if it already exists it just forwards requests and responses.

With NODESESSID, it's almost the same operation. The application on the server set a NODESESSID and then the proxy concatenate the server id in the cookie. When the client send another request, the proxy read the cookie and extract its part the redirect on the correct server.

Choose one of the both stickiness approach for the next tasks.

We'll use the SERVERID mechanism for the following manipulations.

2. Provide the modified `haproxy.cfg` file with a short explanation of the modifications you did to enable sticky session management.

Answer

```
backend nodes
# Define the protocol accepted
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-mode
mode http

# Define the way the backend nodes are checked to know if they are alive or down
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-option%20httpchk
option httpchk HEAD /

# Define the balancing policy
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#balance
balance roundrobin

# Modifications
cookie SERVERID insert indirect

# Automatically add the X-Forwarded-For header
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-option%20forwardfor
# https://en.wikipedia.org/wiki/X-Forwarded-For
option forwardfor

# With this config, we add the header X-Forwarded-Port
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-http-request
http-request set-header X-Forwarded-Port %[dst_port]

# Define the list of nodes to be in the balancing mechanism
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-server
server s1 ${WEBAPP_1_IP}:3000 check cookie s1
server s2 ${WEBAPP_2_IP}:3000 check cookie s2
```

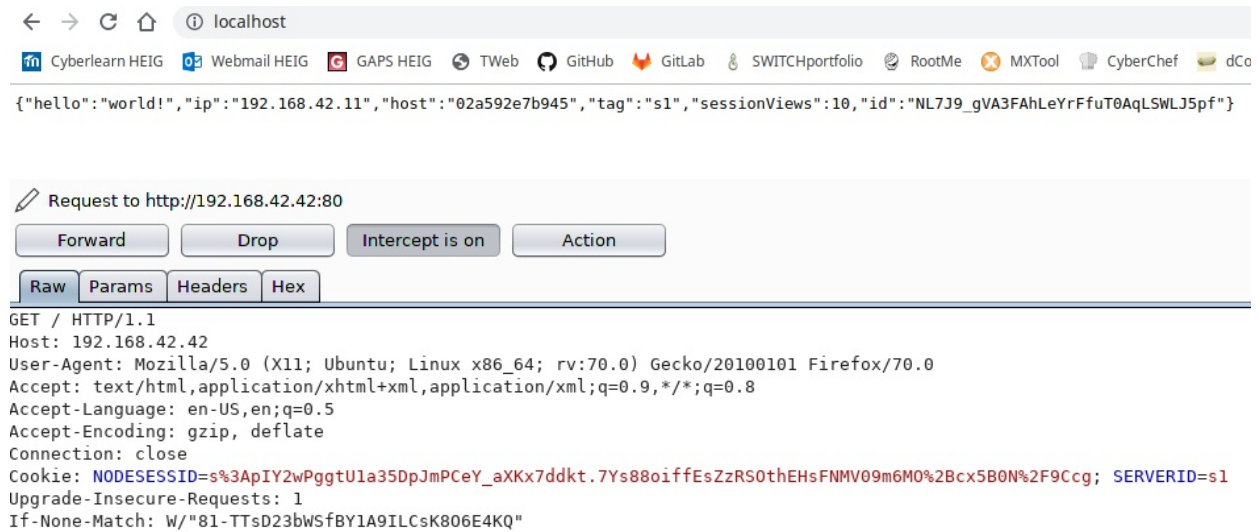
The line starting with `cookie` indicates to HA Proxy that it must insert a `Set-Cookie` named `SERVERID` in the response's header when a client requests for the first time (or rather if the client has not a `SERVERID` cookie already set).

We also added `cookie s1` for server `s1` and `cookie s2` for server `s2` to set the content of the `SERVERID` cookie that will be store in the client's browser.

3. Explain what is the behavior when you open and refresh the URL <http://192.168.42.42> in your browser. Add screenshots to complement your explanations. We expect that you take a deeper a

look at session management.

Answer

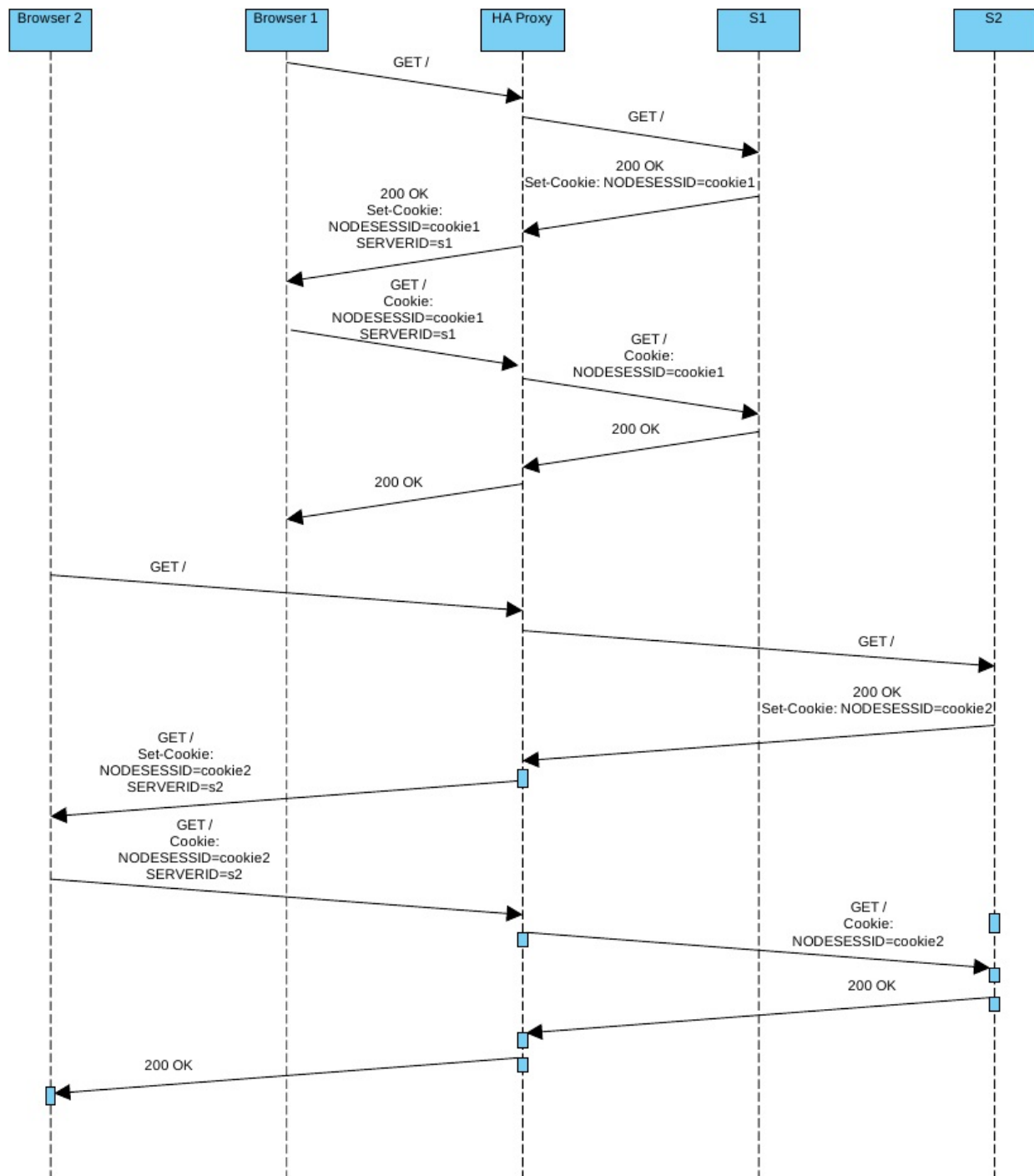


Now, when the page is refreshed several times, the NODESESSID still the same and `sessionViews` is incremented. That shows that HA Proxy is now configured to manage sticky sessions.

In the above image, we can see the SERVERID cookie set to s1. That's because HA Proxy set it up in the first response and now as long as the client's browser keeps the SERVERID cookie, HA proxy will receive it in each requests and forwards messages to s1.

4. Provide a sequence diagram to explain what is happening when one requests the URL for the first time and then refreshes the page. We want to see what is happening with the cookie. We want to see the sequence of messages exchanged (1) between the browser and HAProxy and (2) between HAProxy and the nodes S1 and S2. We also want to see what is happening when a second browser is used.

Answer



5+6. Provide a screenshot of JMeter's summary report. Is there a difference with this run and the run of Task 1? Give a short explanation of what the load balancer is doing.

- Clear the results in JMeter.
- Now, update the JMeter script. Go in the HTTP Cookie Manager and ~~uncheck~~ verify that the box Clear cookies each iteration? is unchecked.
- Go in Thread Group and update the Number of threads . Set the value to 2.


```

> oki@oki-HP-ProBook-450-G5:~$ socat - tcp:192.168.42.42:9999
prompt

> set timeout cli 1d

> set server nodes/s2 state drain

```

192.168.42.42:1936

<

We can see that the s2 node line changed to blue (active or backup SOFT STOPPED for maintenance)

3. Refresh your browser and explain what is happening. Tell us if you stay on the same node or not. If yes, why? If no, why?

Answer

192.168.42.42

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

```

hello: "world!"
ip: "192.168.42.22"
host: "344815313f4f"
tag: "s2"
sessionViews: 31
id: "\NfwVIbR4kGhqxV1DAb3Bk6t9ltcIgKq"

```

When we refresh the browser, we stay on the same node.

It's normal because as explain in the course, DRAIN mode let current sessions continue to make requests to the node in DRAIN mode and will redirect all other traffic to the other nodes.

4. Open another browser and open `http://192.168.42.42` . What is happening?

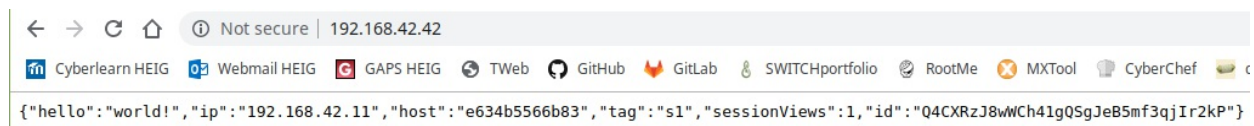
Answer



As expected, all other traffic is redirect to the other node s1.

5. Clear the cookies on the new browser and repeat these two steps multiple times. What is happening? Are you reaching the node in DRAIN mode?

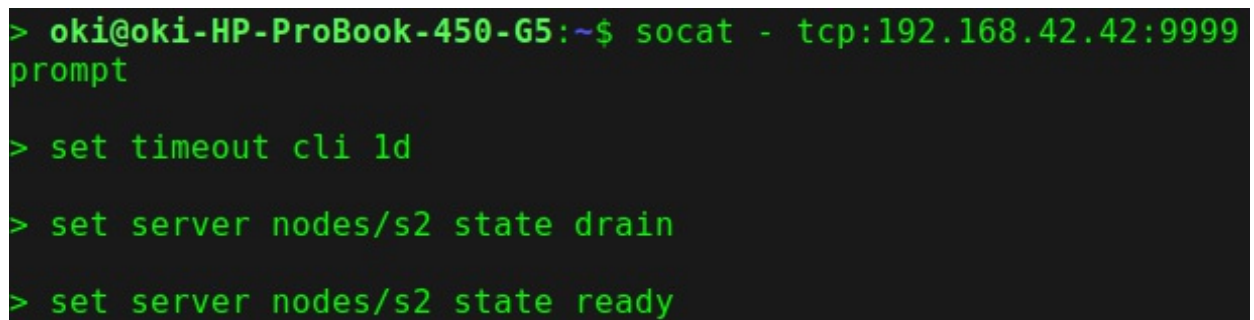
Answer



The DRAIN mode node (s2) is never reach because only the current sessions when I set to node in DRAIN mode are redirect on this node. Again, all other traffic will be redirect on s1.

6. Reset the node in READY mode. Repeat the three previous steps and explain what is happening. Provide a screenshot of HAProxy's stats page.

Answer



</

Now, we can see that s2 is reached one in two with the new browser when cleaning cache between each request. This is the normal round robin mode.

7. Finally, set the node in MAINT mode. Redo the three same steps and explain what is happening. Provide a screenshot of HAProxy's stats page.

Answer

```
> oki@oki-HP-ProBook-450-G5:~$ socat - tcp:192.168.42.42:9999
prompt

> set timeout cli 1d

> set server nodes/s2 state drain

> set server nodes/s2 state ready

> set server nodes/s2 state maint
```

192.168.42.42

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

hello:	"world!"
ip:	"192.168.42.11"
host:	"e634b5566b83"
tag:	"s1"
sessionViews:	2
id:	"bdGVtW3qn62d46IKPldJA6lwsUX0cxKM"

← → ↻ 🏠 ⓘ Not secure | 192.168.42.42

Cyberlearn HEIG Webmail HEIG GAPS HEIG TWeb GitHub GitLab SWITCHportfolio RootMe MXTool CyberChef

```
{
  "hello": "world!",
  "ip": "192.168.42.11",
  "host": "e634b5566b83",
  "tag": "s1",
  "sessionViews": 1,
  "id": "CHLoXgZ4ntkPAL0iaz-NhGKsQQ3s4NRr"
}
```

192.168.42.1936

HAProxy

Statistics Report for pid 12

General process information

pid = 12 (process #1, rdtproc = 1)

uptime = 50 (9h34m00s)

system limits: memmax = unlimited, ulimit = 4044

maxsock = 4044, maxconn = 2000, maxqueue = 0

current conn = 1, current pages = 0/0, conn rate = 1/sec

Running tasks: 1/0, 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

Note: "NOLEBTORANK" = UP with load-balancing disabled

backup UP

backup UP, going down

backup DOWN, going up

just checked

not checked

Display option:

Scope:

Hide DOWN servers

Refresh page

CSV export

External resources:

Primary site

Upstream

Downstream

Online manual

Stats

	Cur	Max	Limit	Queue	Max	Limit	Session rate	Cur	Max	Limit	Sessions	Total	LaTot	Last	In	Out	Bytes	Denied	Req	Resp	Req	Conn	Resp	Req	Warnings	Ret	ReDis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Downtime	Thrtle
Frontend	0	0	0	1	2	1	0	1	1	1	2 000	6	0	0h	2 195	91 061	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Backend	0	0	0	0	0	0	0	0	0	0	200	0	0	0h	2 195	91 061	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Stats (all backends)

	Cur	Max	Limit	Queue	Max	Limit	Session rate	Cur	Max	Limit	Sessions	Total	LaTot	Last	In	Out	Bytes	Denied	Req	Resp	Req	Conn	Resp	Req	Warnings	Ret	ReDis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Downtime	Thrtle
Frontend	0	0	0	0	5	1	0	0	3	1	2 000	54	0	0h	23 055	21 307	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Backend	0	0	0	0	0	0	0	0	0	0	0	0	0	0h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Stats (nodes)

	Cur	Max	Limit	Queue	Max	Limit	Session rate	Cur	Max	Limit	Sessions	Total	LaTot	Last	In	Out	Bytes	Denied	Req	Resp	Req	Conn	Resp	Req	Warnings	Ret	ReDis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Downtime	Thrtle
s1	0	0	0	0	1	1	0	0	1	1	11	8	0	0h	5 573	5 334	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
s2	0	0	0	0	5	1	0	0	1	1	35	3	12m07s	17 482	14 028	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Backend	0	0	0	0	5	1	0	0	1	200	46	11	1m07s	23 055	19 362	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

This time, we can see that both browser (firefox with current session and chrome without session) have switch to s1 and the s2 node line in the HAProxy's state page change to brown (active or backup DOWN for maintenance (MAINT)).

As expected, in MAINT mode, the node can't be reach be any requests even these with sessions already established. All traffic is redirect on the only node UP, s1.

Task 4: Round Robin in degraded mode

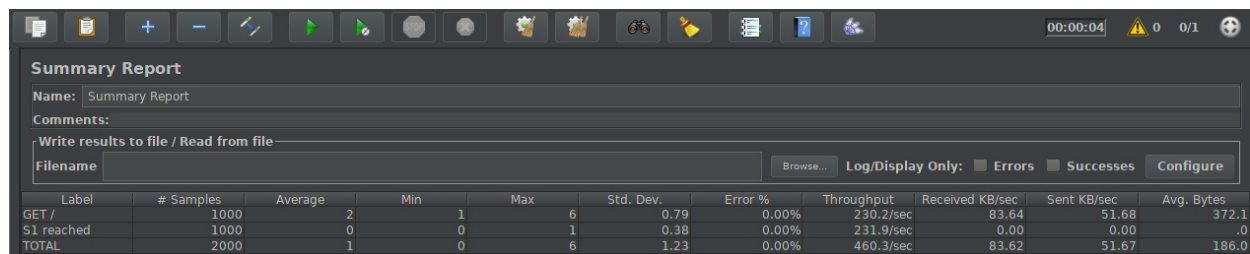
Remark: In general, take a screenshot of the summary report in JMeter to explain what is happening.

Remark: Make sure you have the cookies are kept between two requests.

1. Be sure the delay is of 0 milliseconds is set on `s1`. Do a run to have base data to compare with the next experiments.

Answer

```
oki@oki-HP-ProBook-450-G5:~/Documents/HEIG/S5/1_AIT/Labos/Labo3$ docker inspect s1 | grep -nie ipaddress
199:       "SecondaryIPAddresses": null,
205:       "IPAddress": "",
222:       "IPAddress": "192.168.42.11",
oki@oki-HP-ProBook-450-G5:~/Documents/HEIG/S5/1_AIT/Labos/Labo3$ curl -H "Content-Type: application/json" -X POST -d '{"delay": "0"}' http://192.168.42.11:3000/delay
{"message": "New timeout of 0ms configured."}
oki@oki-HP-ProBook-450-G5:~/Documents/HEIG/S5/1_AIT/Labos/Labo3$
```

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

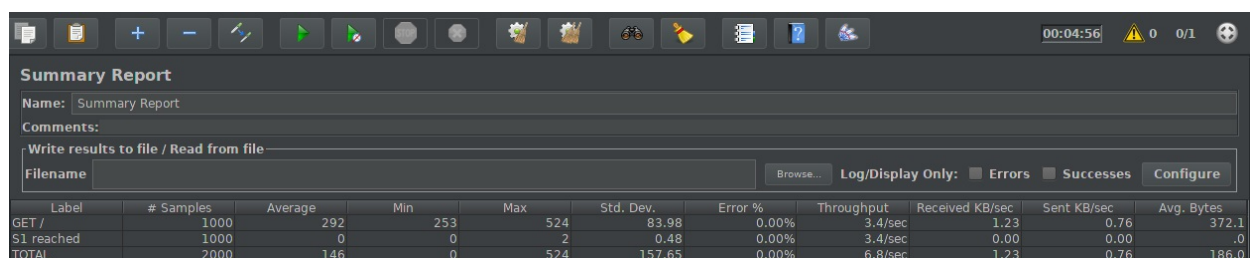
Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	1000	2	1	6	0.79	0.00%	230.2/sec	83.64	51.68	372.1
S1 reached	1000	0	0	1	0.38	0.00%	231.9/sec	0.00	0.00	.0
TOTAL	2000	1	0	6	1.23	0.00%	460.3/sec	83.62	51.67	186.0

2. Set a delay of 250 milliseconds on `s1`. Relaunch a run with the JMeter script and explain what it is happening?

Answer

```
oki@oki-HP-ProBook-450-G5:~/Documents/HEIG/S5/1_AIT/Labos/Labo3$ curl -H "Content-Type: application/json" -X POST -d '{"delay": 250}' http://192.168.42.11:3000/delay
{"message": "New timeout of 250ms configured."}oki@oki-HP-ProBook-450-G5:~/Documents/HEIG/S5/1_AIT/Labos/Labo3$
```



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes

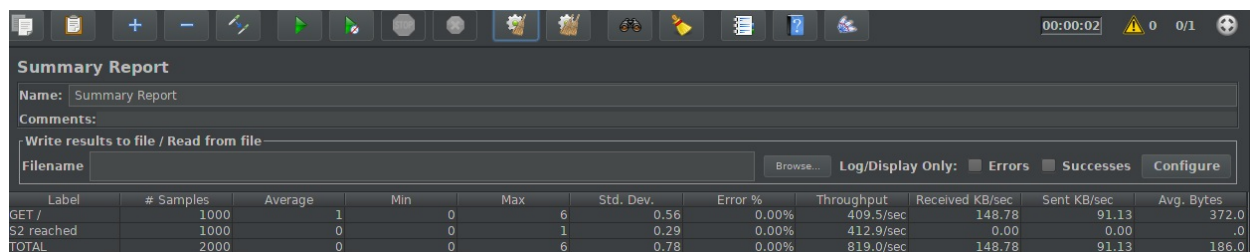
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	1000	292	253	524	83.98	0.00%	3.4/sec	1.23	0.76	372.1
S1 reached	1000	0	0	2	0.48	0.00%	3.4/sec	0.00	0.00	.0
TOTAL	2000	146	0	524	157.65	0.00%	6.8/sec	1.23	0.76	186.0

Now we can see the average response time for a request on s1 is 292 [ms]. It's because we configured s1 to put a delay of 250 [ms] before send a response.

3. Set a delay of 2500 milliseconds on `s1`. Same than previous step.

Answer

```
oki@oki-HP-ProBook-450-G5:~/Documents/HEIG/S5/1_AIT/Labos/Labo3$ curl -H "Content-Type: application/json" -X POST -d '{"delay": 2500}' http://192.168.42.11:3000/delay
{"message": "New timeout of 2500ms configured."}oki@oki-HP-ProBook-450-G5:~/Documents/HEIG/S5/1_AIT/Labos/Labo3$
```



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	1000	1	0	6	0.56	0.00%	409.5/sec	148.78	91.13	372.0
S2 reached	1000	0	0	1	0.29	0.00%	412.9/sec	0.00	0.00	.0
TOTAL	2000	0	0	6	0.78	0.00%	819.0/sec	148.78	91.13	186.0

Now we can see that s1 is never reached. All traffic goes on s2.

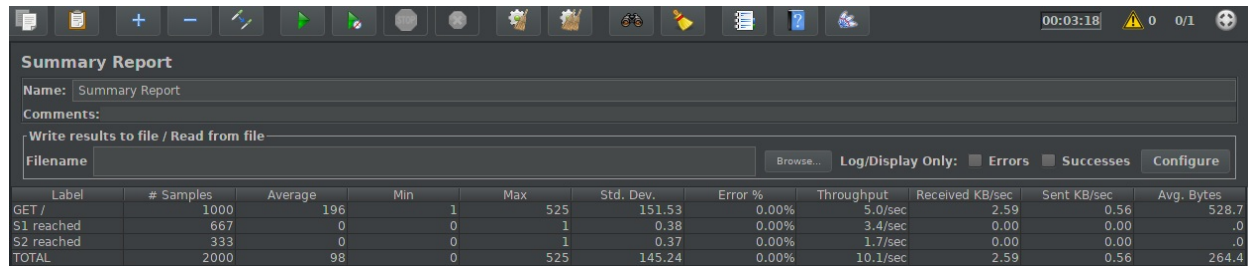
4. In the two previous steps, are there any error? Why?

Statistics Report for pid 10

- We can see that there is no difference in terms of response's time because HA Proxy is configured to manage sticky sessions so all traffic is send to s1 because there is a session

between the client and s1.

Without cookie :



The screenshot shows a 'Summary Report' window with a toolbar at the top. Below the toolbar, there are fields for 'Name' (Summary Report) and 'Comments'. A section for 'Write results to file / Read from file' includes a 'Filename' field and a 'Browse...' button. To the right of this section are checkboxes for 'Log/Display Only', 'Errors', and 'Successes', along with a 'Configure' button. The main part of the window is a table with the following data:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	1000	196	1	525	151.53	0.00%	5.0/sec	2.59	0.56	528.7
S1 reached	667	0	0	1	0.38	0.00%	3.4/sec	0.00	0.00	.0
S2 reached	333	0	0	1	0.37	0.00%	1.7/sec	0.00	0.00	.0
TOTAL	2000	98	0	525	145.24	0.00%	10.1/sec	2.59	0.56	264.4

- Now HA Proxy behaves like a "normal" load balancer because there is a new session at each request.

S1 weighs 2 and S2 weighs 1, that means on 3 requests two will be send to S1 and one to S2.

So the average response's time is a bit better because S2 has "0" [ms] delay.

But it would be a better configuration to set a heavier weight to S2 because it has a better time response than S1. It would improve the average response's time.

Task 5: Balancing strategies

1. Briefly explain the strategies you have chosen and why you have chosen them.

Answer

- static-rr:
Each server is used in turns, according to their weights.
This algorithm is as similar to roundrobin except that it is static, which means that changing a server's weight on the fly will have no effect. It also uses slightly less CPU to run (around -1%). I chose this one to compare performances with an algorithm that has better performances with long sessions.
- leastconn:
This system configures HAProxy to choose the server with the lowest number of active connections. If several servers have the same load, then Round-robin between them is applicated.
This algorithm is recommended where very long sessions are expected.
I chose this one to compare the performances with an algorithm that has better performances with short sessions.

2. Provide evidences that you have played with the two strategies (configuration done, screenshots, ...)

Answer

static-rr:

Configuration

```
backend nodes
# Define the protocol accepted
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-mode
mode http

# Define the way the backend nodes are checked to know if they are alive or down
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-option%20httpchk
option httpchk HEAD /

# Define the balancing policy
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#balance
balance static-rr

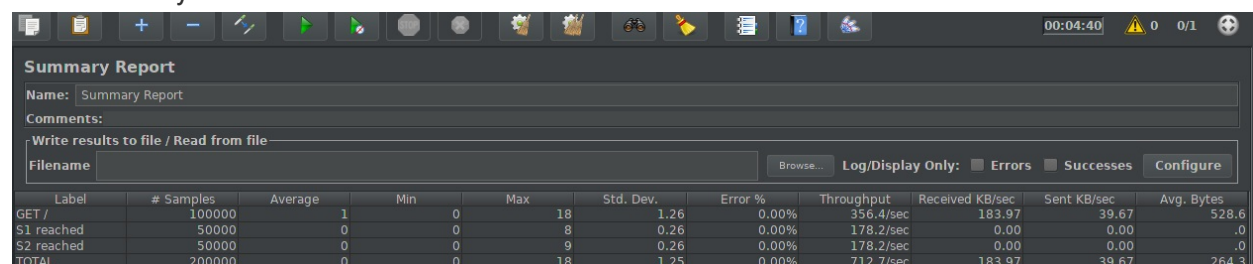
# Modifications
cookie SERVERID insert indirect

# Automatically add the X-Forwarded-For header
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-option%20forwardfor
# https://en.wikipedia.org/wiki/X-Forwarded-For
option forwardfor

# With this config, we add the header X-Forwarded-Port
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-http-request
http-request set-header X-Forwarded-Port %[dst_port]

# Define the list of nodes to be in the balancing mechanism
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-server
server s1 ${WEBAPP_1_IP}:3000 cookie s1 weight 1 check
server s2 ${WEBAPP_2_IP}:3000 cookie s2 weight 1 check
```

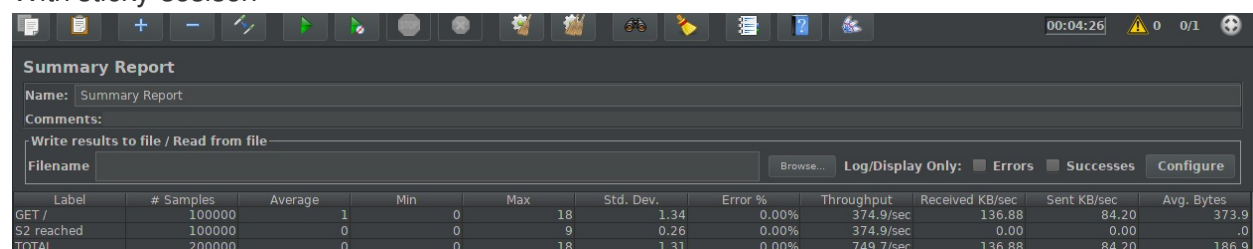
Without sticky-session



The screenshot shows the HAProxy statistics page with a 'Summary Report' section. The report includes fields for Name, Comments, and a table of statistics. The table has columns for Label, # Samples, Average, Min, Max, Std. Dev., Error %, Throughput, Received KB/sec, Sent KB/sec, and Avg. Bytes. The data shows a total of 200,000 samples with an average of 1 and a maximum of 18. The throughput is 712.7/sec, and the received and sent data rates are 183.97 KB/sec and 39.67 KB/sec respectively.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	100000	1	0	18	1.26	0.00%	356.4/sec	183.97	39.67	528.6
S1 reached	50000	0	0	8	0.26	0.00%	178.2/sec	0.00	0.00	.0
S2 reached	50000	0	0	9	0.26	0.00%	178.2/sec	0.00	0.00	.0
TOTAL	200000	0	0	18	1.25	0.00%	712.7/sec	183.97	39.67	264.3

With sticky-session



The screenshot shows the HAProxy statistics page with a 'Summary Report' section. The report includes fields for Name, Comments, and a table of statistics. The table has columns for Label, # Samples, Average, Min, Max, Std. Dev., Error %, Throughput, Received KB/sec, Sent KB/sec, and Avg. Bytes. The data shows a total of 200,000 samples with an average of 1 and a maximum of 18. The throughput is 749.7/sec, and the received and sent data rates are 136.88 KB/sec and 84.20 KB/sec respectively.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	100000	1	0	18	1.34	0.00%	374.9/sec	136.88	84.20	373.9
S2 reached	100000	0	0	9	0.26	0.00%	374.9/sec	0.00	0.00	.0
TOTAL	200000	0	0	18	1.31	0.00%	749.7/sec	136.88	84.20	186.9

leastconn:

Configuration

```
backend nodes
# Define the protocol accepted
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-mode
mode http

# Define the way the backend nodes are checked to know if they are alive or down
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-option%20httpchk
option httpchk HEAD /

# Define the balancing policy
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#balance
balance leastconn

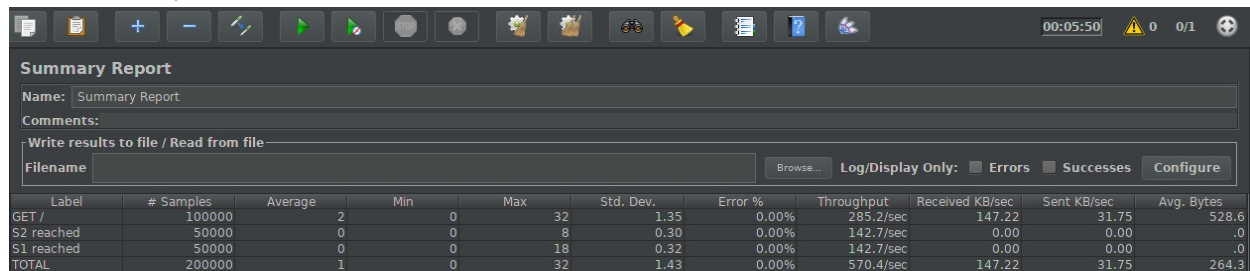
# Modifications
cookie SERVERID insert indirect

# Automatically add the X-Forwarded-For header
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-option%20forwardfor
# https://en.wikipedia.org/wiki/X-Forwarded-For
option forwardfor

# With this config, we add the header X-Forwarded-Port
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-http-request
http-request set-header X-Forwarded-Port %[dst_port]

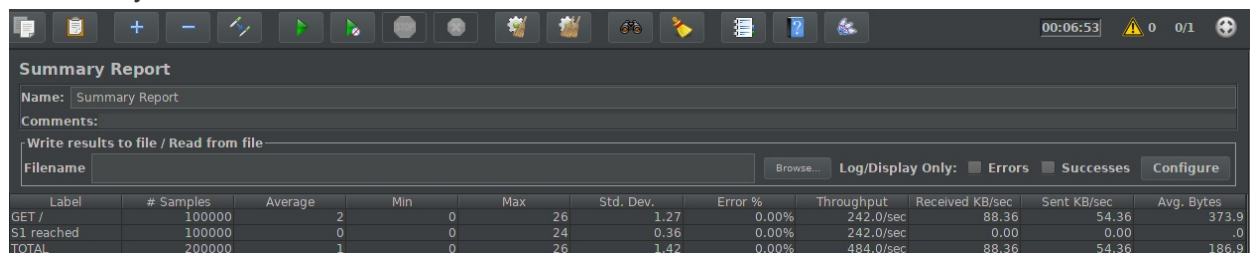
# Define the list of nodes to be in the balancing mechanism
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-server
server s1 ${WEBAPP_1_IP}:3000 cookie s1 weight 1 check
server s2 ${WEBAPP_2_IP}:3000 cookie s2 weight 1 check
```

Without sticky-session



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	100000	2	0	32	1.35	0.00%	285.2/sec	147.22	31.75	528.6
S2 reached	50000	0	0	8	0.30	0.00%	142.7/sec	0.00	0.00	.0
S1 reached	50000	0	0	18	0.32	0.00%	142.7/sec	0.00	0.00	.0
TOTAL	200000	1	0	32	1.43	0.00%	570.4/sec	147.22	31.75	264.3

With sticky-session



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	100000	2	0	26	1.27	0.00%	242.0/sec	88.36	54.36	373.9
S1 reached	100000	0	0	24	0.36	0.00%	242.0/sec	0.00	0.00	.0
TOTAL	200000	1	0	26	1.42	0.00%	484.0/sec	88.36	54.36	186.9

3. Compare the both strategies and conclude which is the best for this lab (not necessary the best at all).

Answer

For tests, we put the total number of requests at 100'000 to better see performances.

As we expect, the static-rr mode has better response's time average. This is because is the round-robin algorithm is really easy and HAProxy doesn't need to do any complex operation before

forward the request and the response. And because we have only short session time for our application, rr-static is a better solution than leastconn.

Conclusion

This lab allowed us to get familiar with the HAProxy environment and its features. We also found out some other balancing modes (e.g. first, source, uri) that can be used for specific infrastructure types or requirements.

HAProxy configuration file is pretty much easy to administrate and the community edition is free. This tool could be used as reverse proxy for a small or medium infrastructure.