Lab 03 - Load balancing

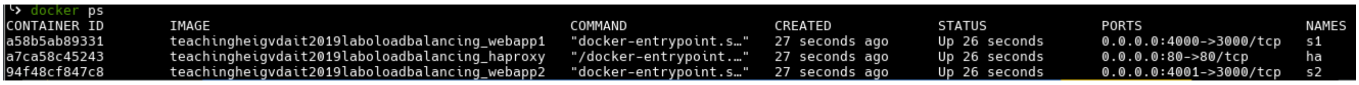
Authors : Nair Alic & Adam Zouari

Introduction

In this laboratory, we've been asked to deploy a web application in a two tier architecture. The laboratory is separated in 5 tasks, in each task we've been asked to configure and test the load balancer/proxy called HAProxy in different way. We also made several load tests using JMeter.

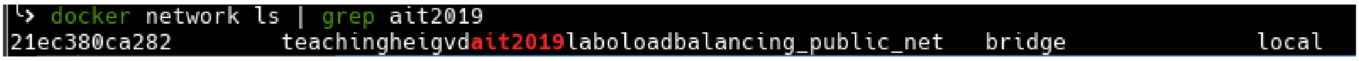
Task 1: Install the tools

We already had installed the tools in others course so we had directly launch the docker-compose and we verified that we have 3 running containers :



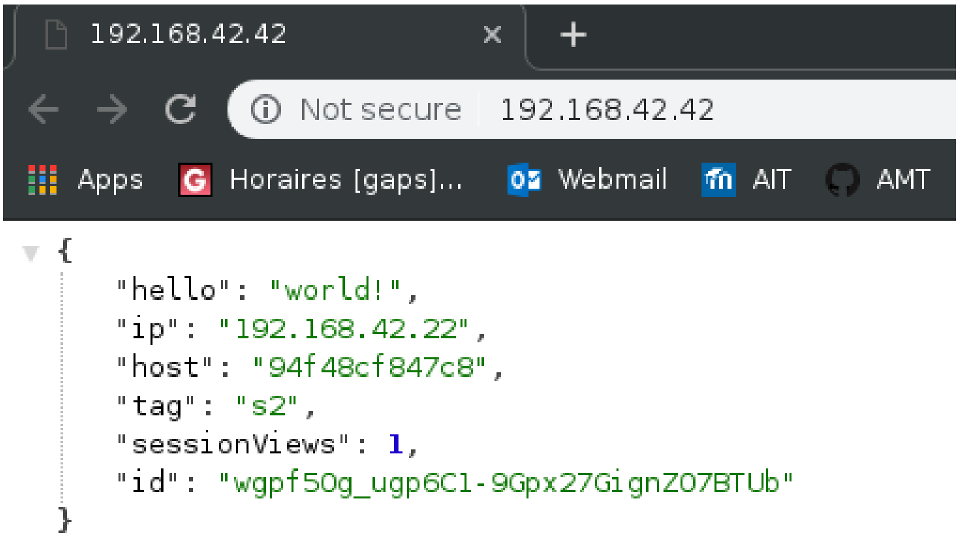
```
↳ docker ps
CONTAINER ID        IMAGE                                        COMMAND                  CREATED             STATUS              PORTS                    NAMES
a58b5ab89331        teachingheigvdait2019laboloadbalancing_webapp1   "docker-entrypoint.s…"   27 seconds ago      Up 26 seconds       0.0.0.0:4000->3000/tcp   s1
a7ca58c45243        teachingheigvdait2019laboloadbalancing_haproxy   "/docker-entrypoint.…"   27 seconds ago      Up 26 seconds       0.0.0.0:80->80/tcp       ha
94f48cf847c8        teachingheigvdait2019laboloadbalancing_webapp2   "docker-entrypoint.s…"   27 seconds ago      Up 26 seconds       0.0.0.0:4001->3000/tcp   s2
```

The containers are connected by a network bridge :



```
↳ docker network ls | grep ait2019
21ec380ca282        teachingheigvdait2019laboloadbalancing_public_net    bridge              local
```

We can now navigate to the address of the load balancer :

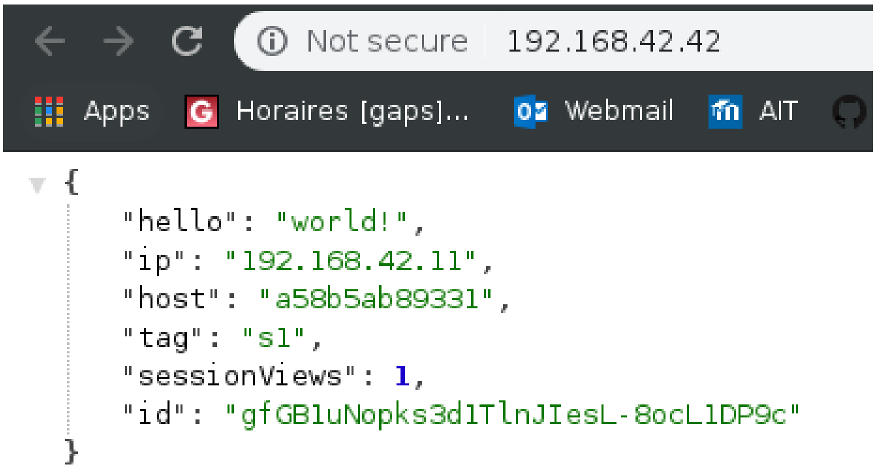


```
{
    "hello": "world!",
    "ip": "192.168.42.22",
    "host": "94f48cf847c8",
    "tag": "s2",
    "sessionViews": 1,
    "id": "wgpf50g_ugp6Cl-9Gpx27GignZ07BTUb"
}
```

Deliverables:

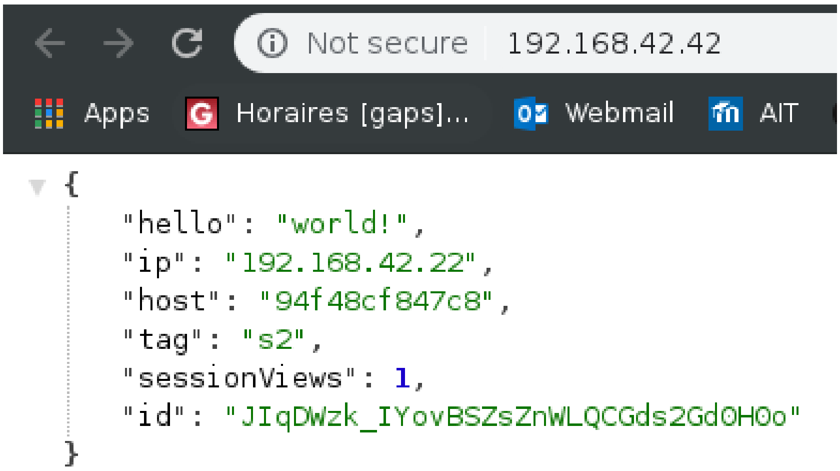
1. Explain how the load balancer behaves when you open and refresh the URL http://192.168.42.42 in your browser. Add screenshots to complement your explanations. We expect that you take a deeper a look at session management.

   When we open the URL a first time, the server `s1` respond with `gfGB....` as the session ID.



```
{
    "hello": "world!",
    "ip": "192.168.42.11",
    "host": "a58b5ab89331",
    "tag": "s1",
    "sessionViews": 1,
    "id": "gfGB1uNopks3d1TlnJIesL-8ocL1DP9c"
}
```

After a refresh, it's now the server `s2` who respond with another session ID.



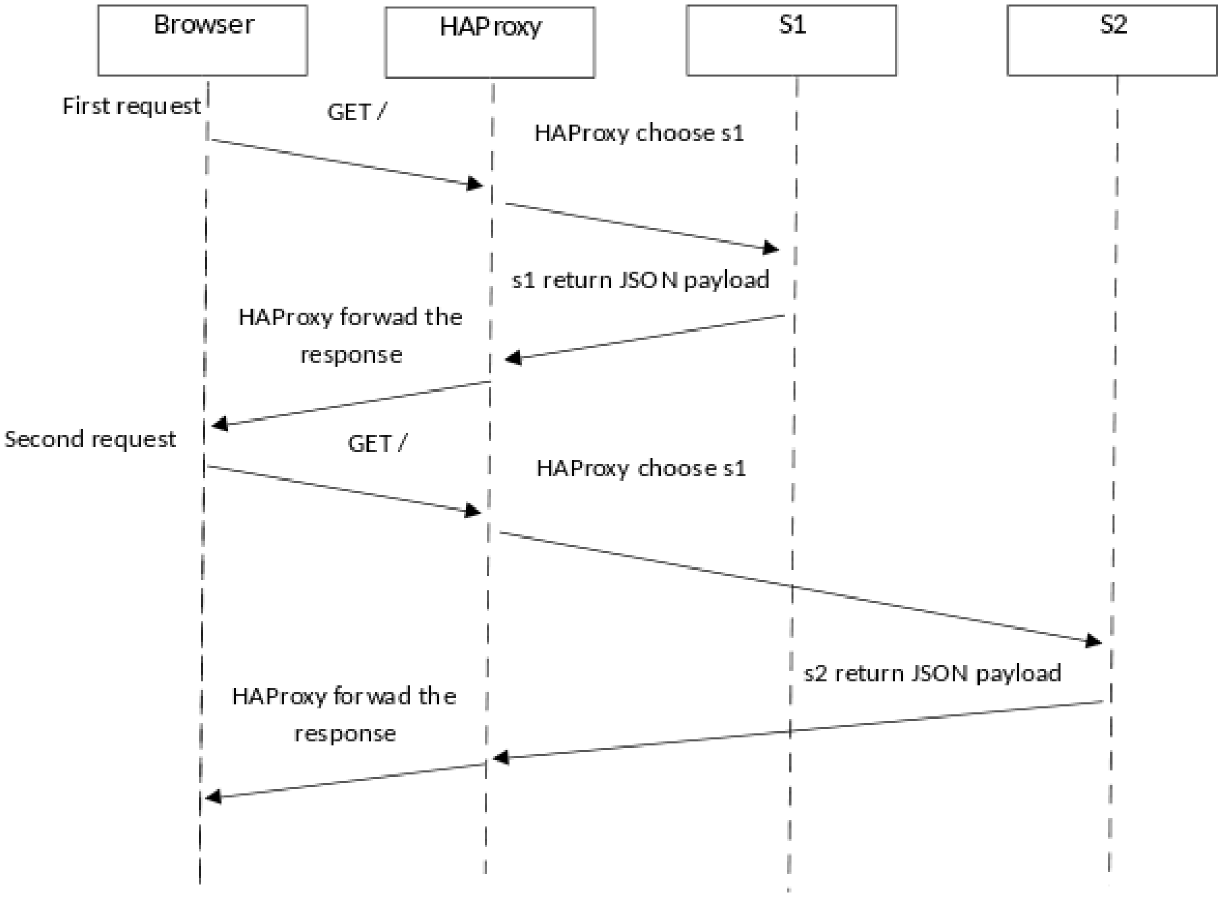
```
{
    "hello": "world!",
    "ip": "192.168.42.22",
    "host": "94f48cf847c8",
    "tag": "s2",
    "sessionViews": 1,
    "id": "JIqDWzk_IYovBSZsZnWLQCGds2Gd0H0o"
}
```

At each request the sessionID changes. The load balancer seems not to care about the session management and use a Round Robin (One request per server in a uniform rotation) schedule policy to determine to which server a request should be sent.

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

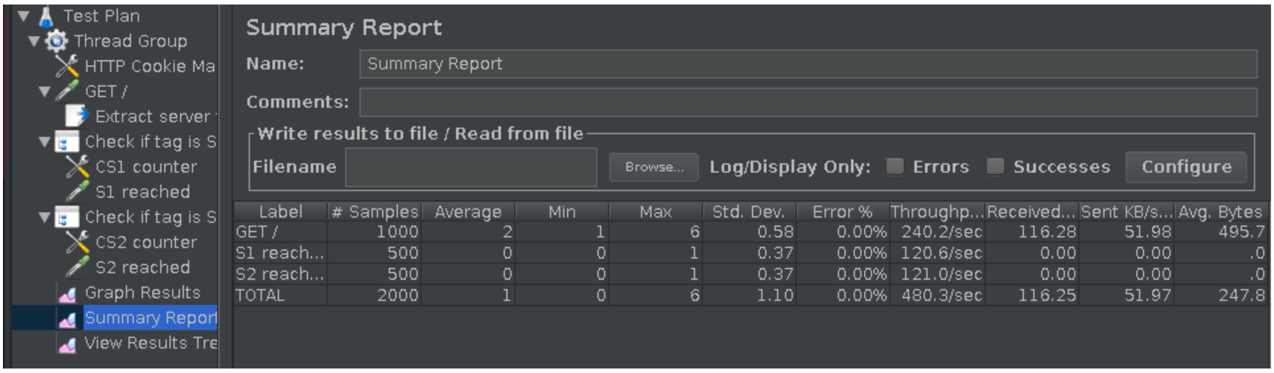
   The second request also should been handled by server `s1`, the `id` should have been the same as the first resquest and the `sessionViews` should have been incremented.

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. Here is an example:



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

   Wee that the load is uniformly distributed between the two servers.



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename

Log/Display Only: □ Errors □ Successes

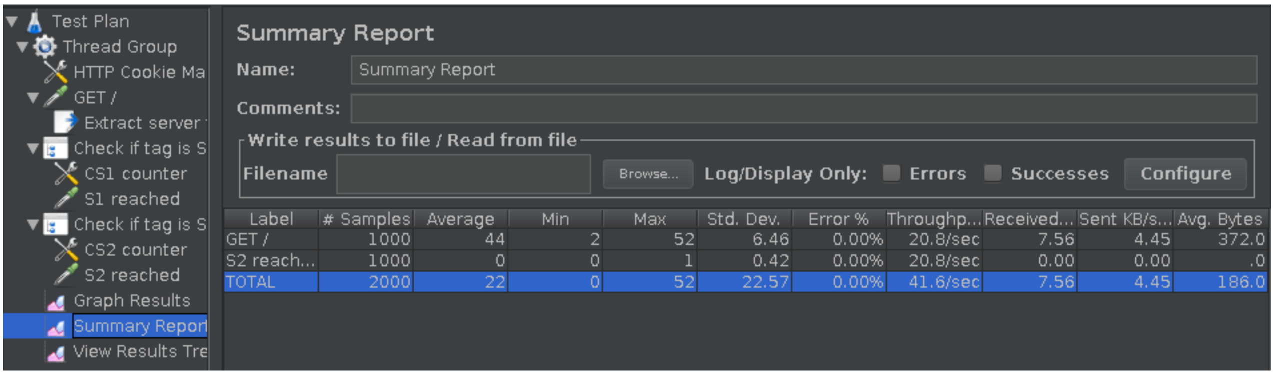
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughp...	Received...	Sent KB/s...	Avg. Bytes
GET /	1000	2	1	6	0.58	0.00%	240.2/sec	116.28	51.98	495.7
S1 reach...	500	0	0	1	0.37	0.00%	120.6/sec	0.00	0.00	.0
S2 reach...	500	0	0	1	0.37	0.00%	121.0/sec	0.00	0.00	.0
TOTAL	2000	1	0	6	1.10	0.00%	480.3/sec	116.25	51.97	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.

   Obviously after stop the server `s1`, all requests are handled by the second server `s2`.



Summary Report

Name: Summary Report

Comments:

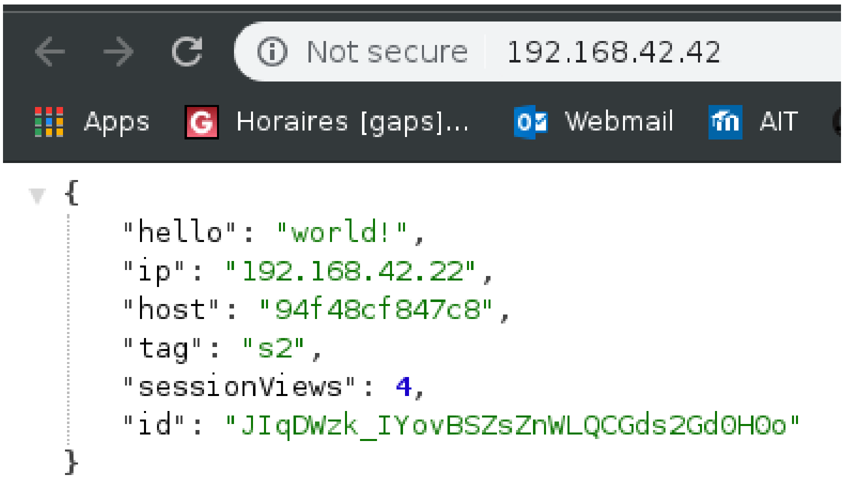
Write results to file / Read from file

Filename

Log/Display Only: □ Errors □ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughp...	Received...	Sent KB/s...	Avg. Bytes
GET /	1000	44	2	52	6.46	0.00%	20.8/sec	7.56	4.45	372.0
S2 reach...	1000	0	0	1	0.42	0.00%	20.8/sec	0.00	0.00	.0
TOTAL	2000	22	0	52	22.57	0.00%	41.6/sec	7.56	4.45	186.0

Therefore, after a refresh, the server remember that a request with thissession ID have already been made so the `sessionViews` is correctly incremented.



```
{
    "hello": "world!",
    "ip": "192.168.42.22",
    "host": "94f48cf847c8",
    "tag": "s2",
    "sessionViews": 4,
    "id": "JIqDWzk_IYovBSZsZnWLQCGds2Gd0H0o"
}
```

Task 2: Sticky sessions

It’s time to go further. At this stage, we now have a load balanced web application but the session management is totally messed up. In this task your job is to fix the configuration of HAProxy to enable sticky session management.

For that, you will have to play with docker a little bit more. You might want to consult the file Docker quick reference for some useful commands and hints.

Deliverables:

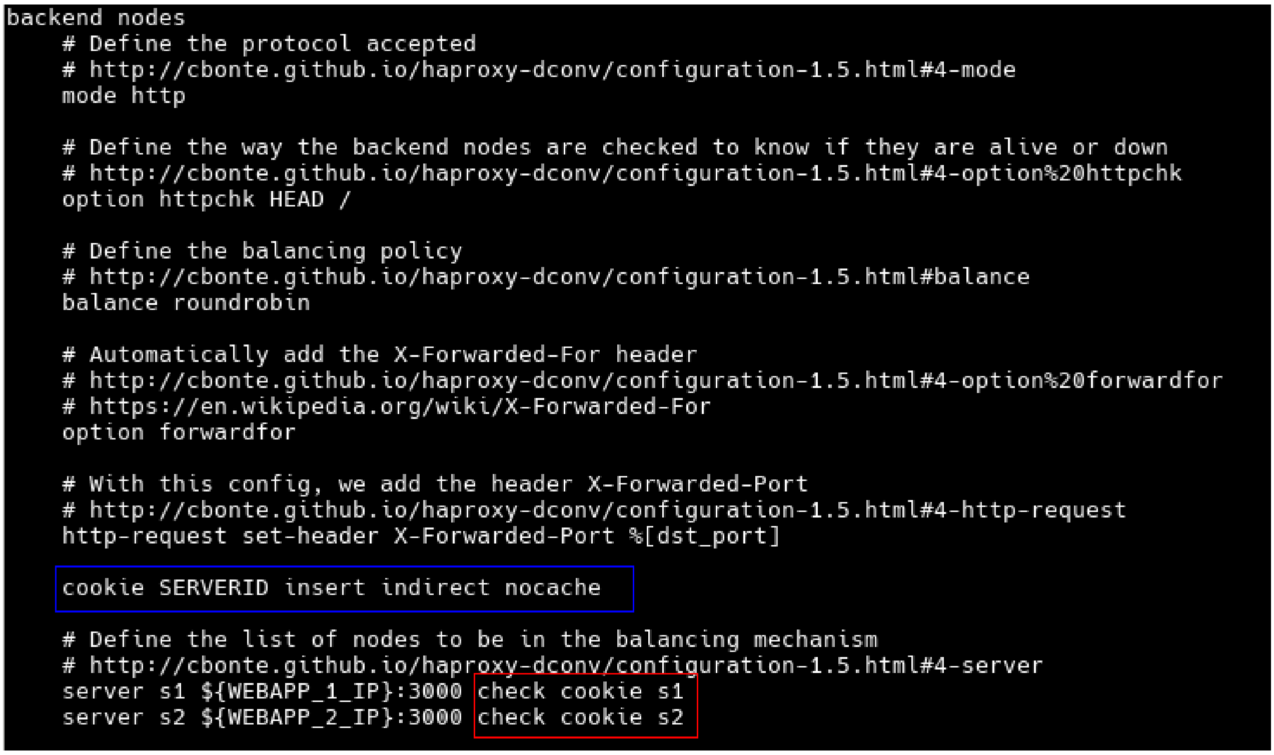
1. There is different way to implement the sticky session. One possibility is to use the SERVERID provided by HAProxy. Another way is to use the NODESESSID provided by the application. Briefly explain the difference between both approaches (provide a sequence diagram with cookies to show the difference).

   In the case of SERVERID, HAProxy will send the cookie in the first response. In the next request, the cookie will be sent by the client, and used by HAProxy to determine which server to forward the request.
   For NODESESSID, this is the application server which will send the cookie in the first response.

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

     For the next tasks we will use the SERVERID way.

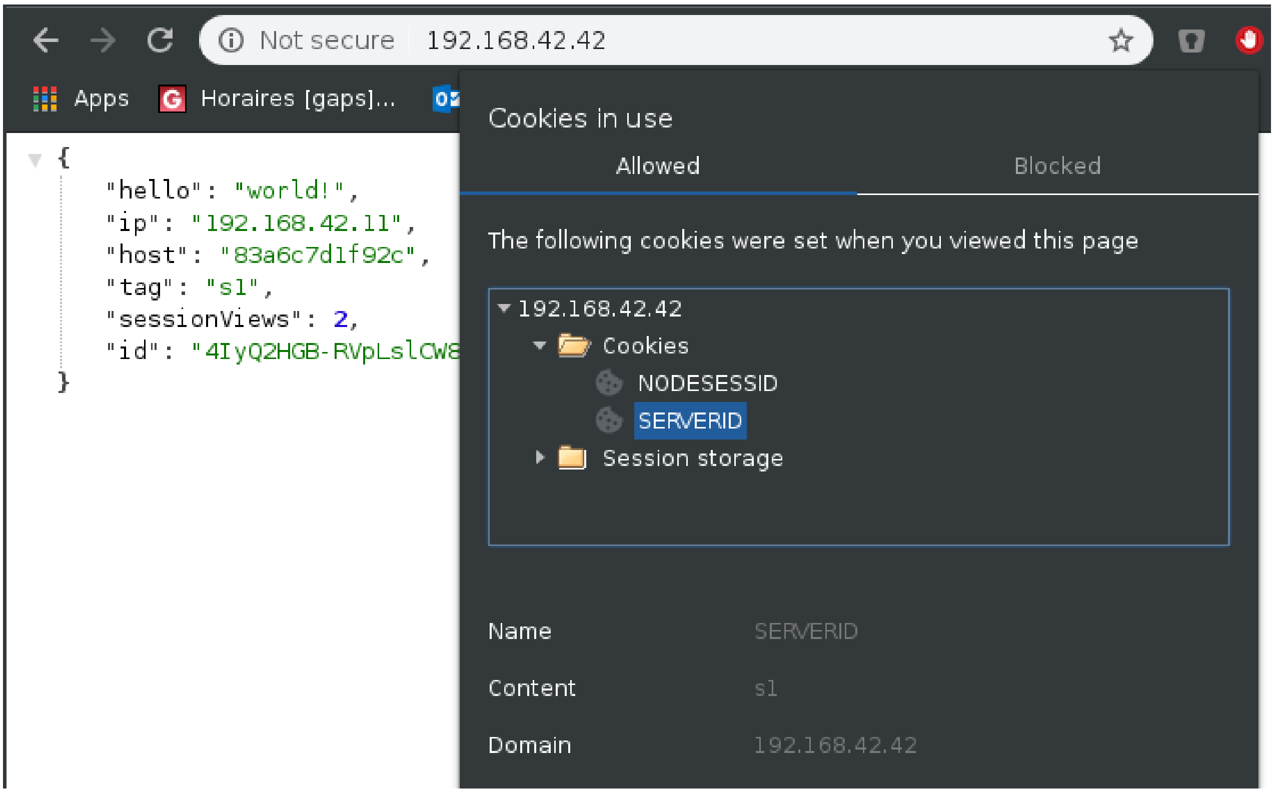
2. Provide the modified `haproxy.cfg` file with a short explanation of the modifications you did to enable sticky session management.



```
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

    # 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]

    cookie SERVERID insert indirect nocache

    # 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 blue line tells HAProxy to setup a cookie called SERVERID only if the user didn't come with suche one. In red, it provides the value of the cookie inserted by HAProxy to know which server to choose for this client.

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.



```
{
    "hello": "world!",
    "ip": "192.168.42.11",
    "host": "83a6c7d1f92c",
    "tag": "s1",
    "sessionViews": 2,
    "id": "4IyQ2HGB-RVpLslCW8
}
```

The following cookies were set when you viewed this page

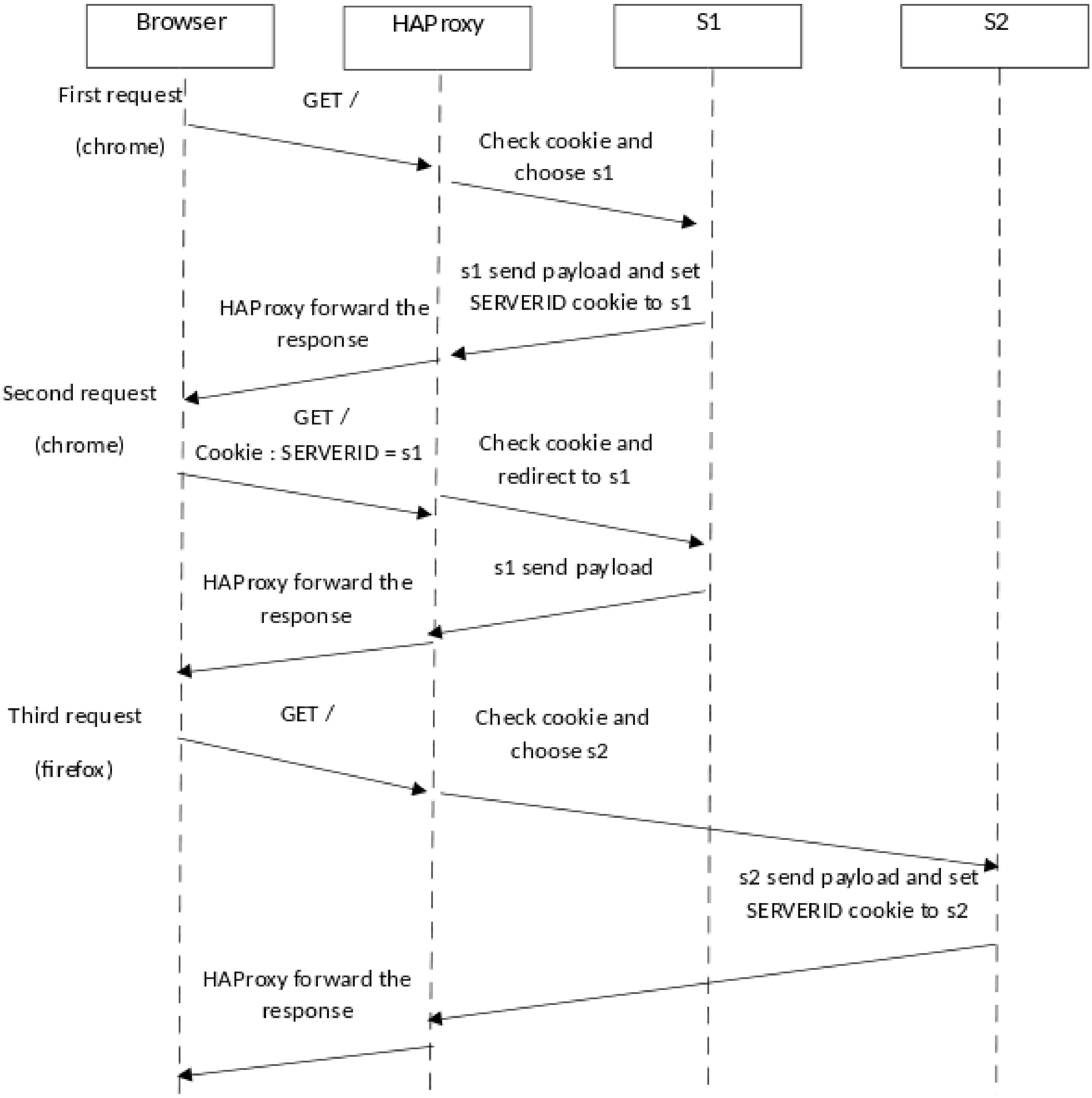
Name SERVERID

Content s1

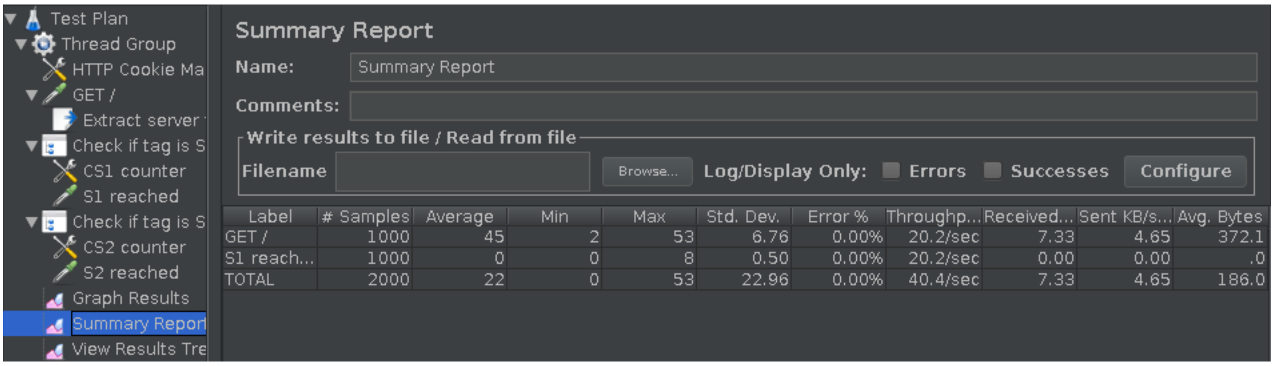
Domain 192.168.42.42

Now the session management seems to be correct. After a refresh we see that the value of `sessionViews` is 2 and the SERVERID cookie is set 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.


Browser
HAProxy
S1
S2
First request
(chrome)
GET /
Check cookie and choose s1
s1 send payload and set SERVERID cookie to s1
HAProxy forward the response
Second request
(chrome)
GET /
Cookie : SERVERID = s1
Check cookie and redirect to s1
s1 send payload
HAProxy forward the response
Third request
(firefox)
GET /
Check cookie and choose s2
s2 send payload and set SERVERID cookie to s2
HAProxy forward the response

5. Provide a screenshot of JMeter's summary report. Is there a difference with this run and the run of Task 1?



Summary Report

Name: Summary Report

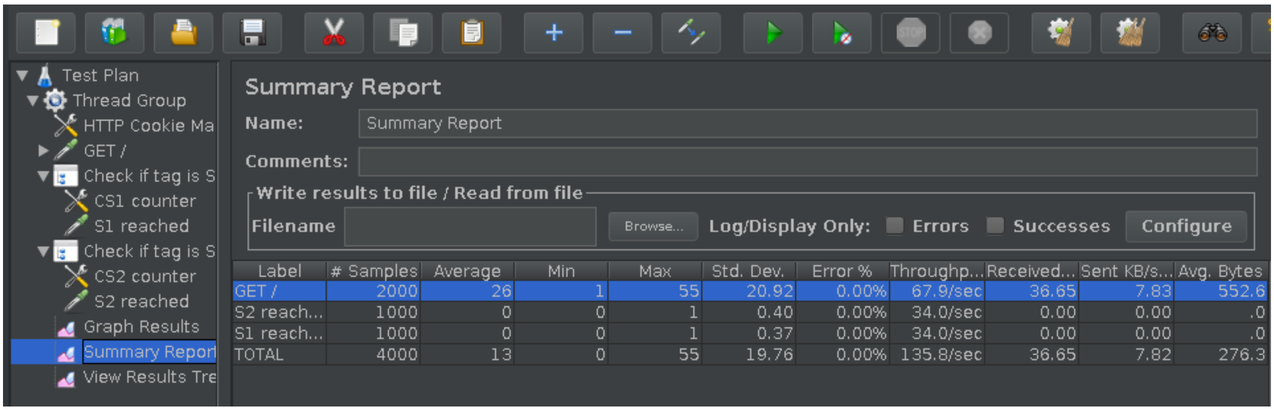
Comments:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughp...	Received...	Sent KB/s...	Avg. Bytes
GET /	1000	45	2	53	6.76	0.00%	20.2/sec	7.33	4.65	372.1
S1 reach...	1000	0	0	8	0.50	0.00%	20.2/sec	0.00	0.00	.0
TOTAL	2000	22	0	53	22.96	0.00%	40.4/sec	7.33	4.65	186.0

The behaviour is the same that when we stopped one of the server. All request reach one of the server because they all use the same SERVERID.

- 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.

6. Provide a screenshot of JMeter's summary report. Give a short explanation of what the load balancer is doing.



Summary Report

Name: Summary Report

Comments:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughp...	Received...	Sent KB/s...	Avg. Bytes
GET /	2000	26	1	55	20.92	0.00%	67.9/sec	36.65	7.83	552.6
S2 reach...	1000	0	0	1	0.40	0.00%	34.0/sec	0.00	0.00	.0
S1 reach...	1000	0	0	1	0.37	0.00%	34.0/sec	0.00	0.00	.0
TOTAL	4000	13	0	55	19.76	0.00%	135.8/sec	36.65	7.82	276.3

Now the load balancer use the Round Robin algorithm to distribute uniformly the SERVERID between the request. That involves that the the request will be uniformly distribued but now with a notion of session. If a request with a SERVERID comes, it will be handled by the same server that the first time.

Task 3: Drain mode

HAProxy provides a mode where we can set a node to DRAIN state. In this case, HAProxy will let *current* sessions continue to make requests to the node in DRAIN mode and will redirect all other traffic to the other nodes.

In our case, it means that if we put `s1` in DRAIN mode, all new traffic will reach the `s2` node and all current traffic directed to `s1` will continue to communicate with `s1`.

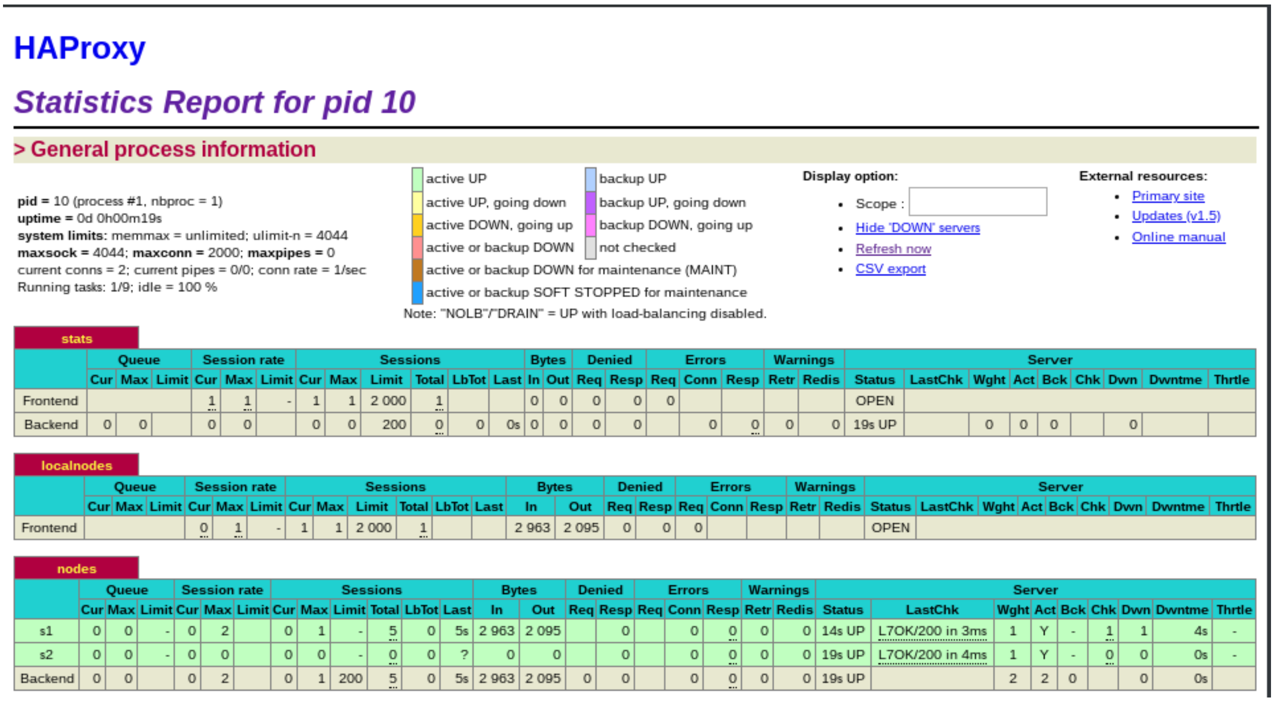
Another mode is MAINT mode which is more intrusive than DRAIN. In this mode, all current and new traffic is redirected to the other active nodes even if there are active sessions.

In this task, we will experiment with these two modes. We will base our next steps on the work done on Task

2. We expect you have a working Sticky Session configuration with two web app nodes up and running called `s1` and `s2` .

Deliverables:

1. Take a screenshot of the Step 5 and tell us which node is answering.



HAProxy

Statistics Report for pid 10

> General process information

pid = 10 (process #1, nbproc = 1)
uptime = 0d 0h00m19s
system limits: memmax = unlimited; ulimit-n = 4044
maxsock = 4044; **maxconn =** 2000; **maxpipes =** 0
current conns = 2; current pipes = 0/0; conn rate = 1/sec
Running tasks: 1/9; 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

External resources:

- Primary site
- Updates (v1.5)
- 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	2 000	1			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	19s 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	-	1	1	2 000	1			2 963	2 095	0	0	0					OPEN								

nodes

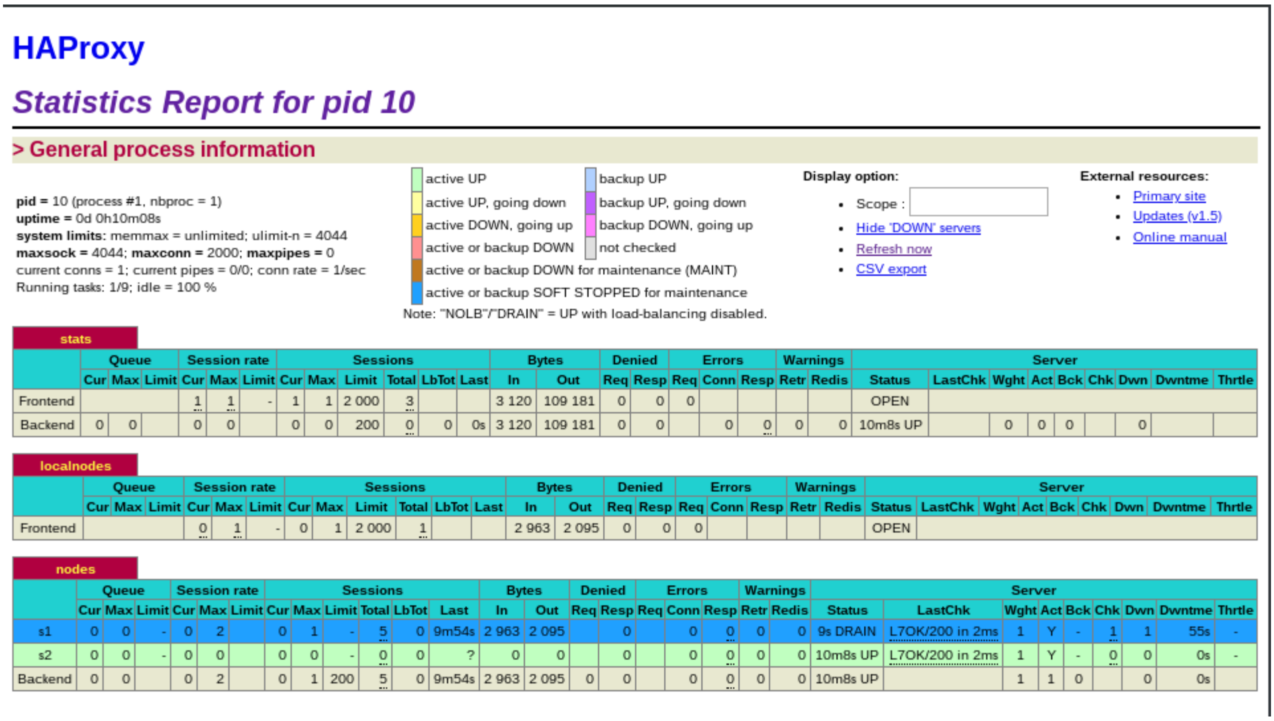
	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
s1	0	0	-	0	2		0	1	-	5	0	5s	2 963	2 095		0		0	0	0	0	14s UP	L7OK/200 in 3ms	1	Y	-	1	1	4s	-
s2	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	19s UP	L7OK/200 in 4ms	1	Y	-	0	0	0s	-
Backend	0	0		0	2		0	1	200	5	0	5s	2 963	2 095	0	0		0	0	0	0	19s UP		2	2	0		0	0s	

Here we see that is the node s1 that answers.

2. Based on your previous answer, set the node in DRAIN mode. Take a screenshot of the HAProxy state page.

To switch in drain mode our s1 node :

```
$ socat - tcp:192.168.42.42:9999
prompt

> set server nodes/s1 state drain
```



HAProxy

Statistics Report for pid 10

> General process information

pid = 10 (process #1, nbproc = 1)
uptime = 0d 0h10m08s
system limits: memmax = unlimited; ulimit-n = 4044
maxsock = 4044; **maxconn =** 2000; **maxpipes =** 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec
Running tasks: 1/9; 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

External resources:

- Primary site
- Updates (v1.5)
- 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	2 000	3			3 120	109 181	0	0	0					OPEN								
Backend	0	0		0	0		0	0	200	0	0	0s	3 120	109 181	0	0		0	0	0	0	10m8s 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	1	2 000	1			2 963	2 095	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	2		0	1	-	5	0	9m54s	2 963	2 095		0		0	0	0	0	9s DRAIN	L7OK/200 in 2ms	1	Y	-	1	1	55s	-
s2	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	10m8s UP	L7OK/200 in 2ms	1	Y	-	0	0	0s	-
Backend	0	0		0	2		0	1	200	5	0	9m54s	2 963	2 095	0	0		0	0	0	0	10m8s UP		1	1	0		0	0s	

Now we see that the status of s1 is DRAIN.

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?

   After refreshing our browser, s1 still responding. That is because HAProxy will 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?

   There is no change because the browser still have the cookie of the of the previous connection. s1 still responding.

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?

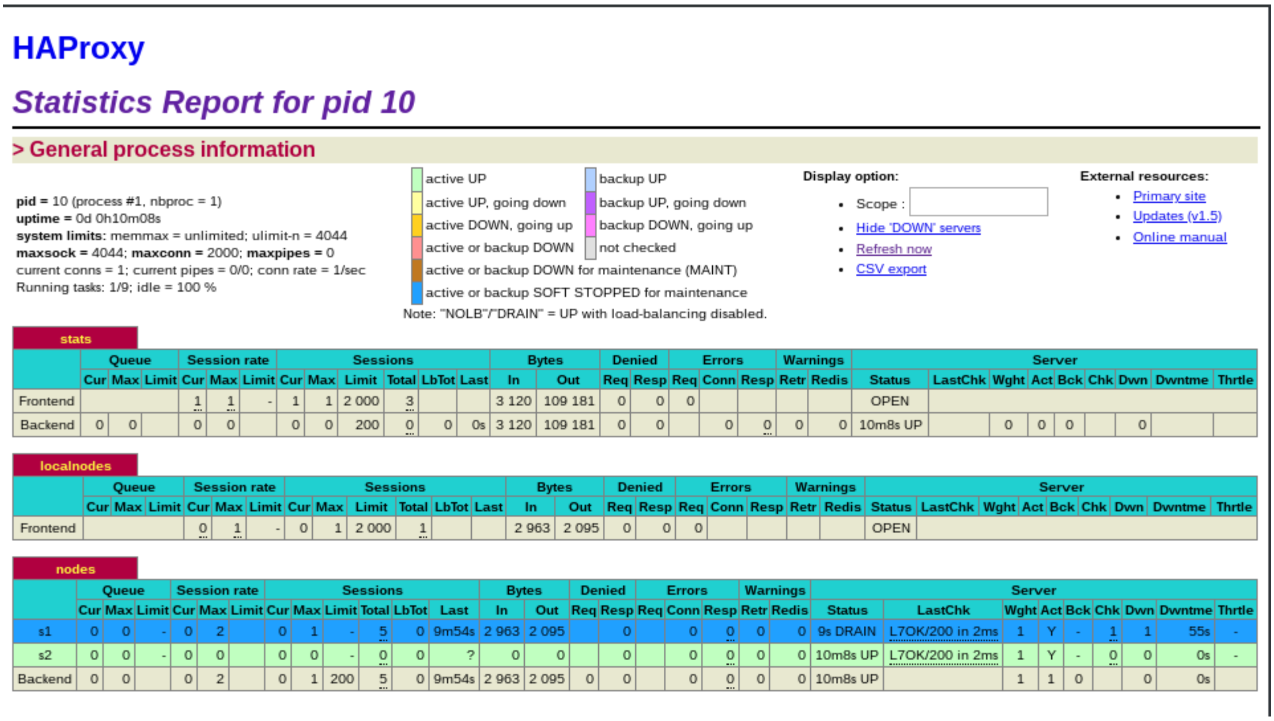
   Now, when we clear the cookie we create a new session so all traffic will redirect to the other node s2. We don't reach the node in DRAIN mode.

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.

   To get back in READY mode :

   ```
   set server nodes/s1 state ready
   ```

   Now, HAProxy use RoundRobin to redirect the requests between the two nodes.



HAProxy

Statistics Report for pid 10

> General process information

pid = 10 (process #1, nbproc = 1)
uptime = 0d 0h10m08s
system limits: memmax = unlimited; ulimit-n = 4044
maxsock = 4044; **maxconn =** 2000; **maxpipes =** 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec
Running tasks: 1/9; 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

External resources:

- Primary site
- Updates (v1.5)
- 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	2 000	3			3 120	109 181	0	0	0					OPEN								
Backend	0	0		0	0		0	0	200	0	0	0s	3 120	109 181	0	0		0	0	0	0	10m8s UP		0	0	0		0		

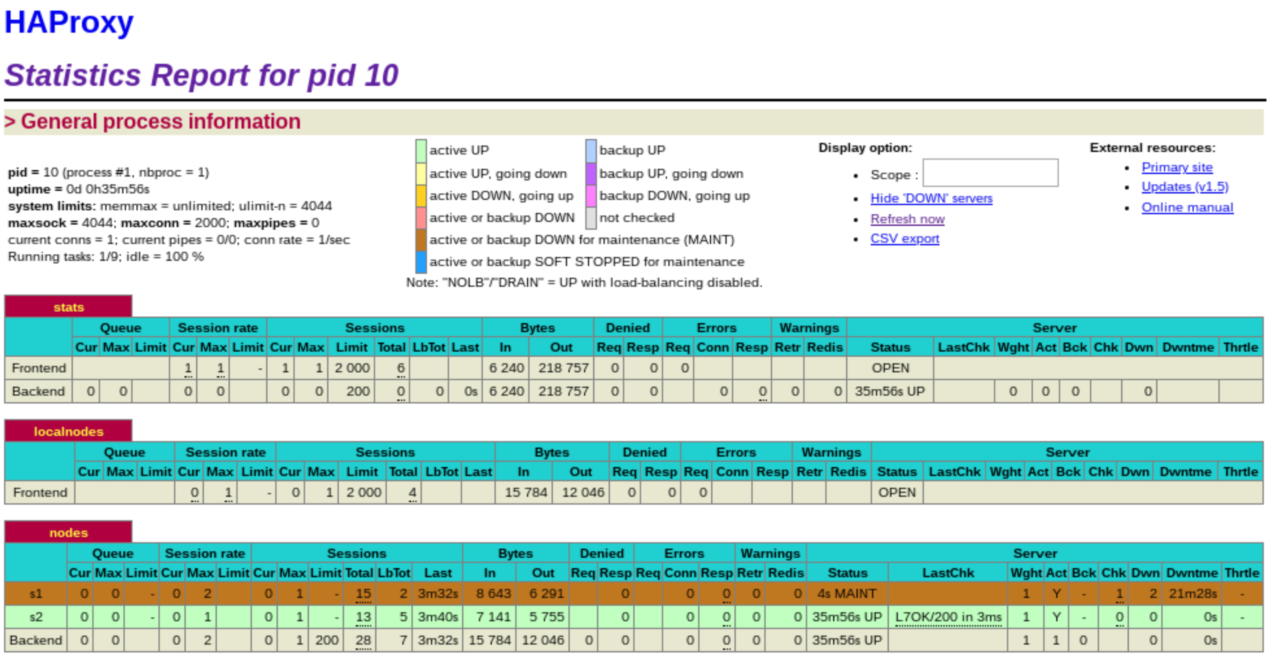
localnodes	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	1	-	0	1	2 000	1			2 963	2 095	0	0	0					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	2		0	1	-	5	0	9m54s	2 963	2 095		0		0	0	0	0	9s DRAIN	L7OK/200 in 2ms	1	Y	-	1	1	55s	-
s2	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	10m8s UP	L7OK/200 in 2ms	1	Y	-	0	0	0s	-
Backend	0	0		0	2		0	1	200	5	0	9m54s	2 963	2 095	0	0		0	0	0	0	10m8s UP		1	1	0		0	0s	

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.
To switch in MAINT mode :

```
set server nodes/s1 state ready
```

Now, in MAINT mode, HAProxy will redirect ALL requests to the node s2. Even the current sessions wil be redirect to it and we will lose the session (counter restarted).



HAProxy

Statistics Report for pid 10

> General process information

pid = 10 (process #1, nbproc = 1)
uptime = 0d 0h35m56s
system limits: memmax = unlimited; ulimit-n = 4044
maxsock = 4044; **maxconn =** 2000; **maxpipes =** 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec
Running tasks: 1/9; 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

External resources:

- Primary site
- Updates (v1.5)
- 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	2 000	6			6 240	218 757	0	0	0					OPEN								
Backend	0	0		0	0		0	0	200	0	0	0s	6 240	218 757	0	0		0	0	0	0	35m56s 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	1	2 000	4			15 784	12 046	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	2		0	1	-	15	2	3m32s	8 643	6 291		0		0	0	0	0	4s MAINT		1	Y	-	1	2	21m28s	-
s2	0	0	-	0	1		0	1	-	13	5	3m40s	7 141	5 755		0		0	0	0	0	35m56s UP	L7OK/200 in 3ms	1	Y	-	0	0	0s	-
Backend	0	0		0	2		0	1	200	28	7	3m32s	15 784	12 046	0	0		0	0	0	0	35m56s UP		1	1	0		0	0s	

Task 4: Round robin in degraded mode.

In this part, we will try to simulate a degraded mode based on the round-robin previously configured.

To help experimenting the balancing when an application started to behave strangely, the web application has a REST resource to configure a delay in the response. You can set an arbitrary delay in milliseconds. Once the delay is configured, the response will take the amount of time configured.

To set the timeout, you have to do a `POST` request with the following content (be sure the `Content-Type` header is set to `application/json`. The configuration is applicable on each node. Therefore, you can do one `POST` request on `http://192.168.42.42/delay` and taking a look at the response cookies will tell you which node has been configured.

```
{
  "delay": 1000
}
```

The previous example will set a delay of 1 second.

Or retrieve the IP of the container you want to configure and then do the `curl` command to configure the delay.

```
$ docker inspect <containerName>

$ curl -H "Content-Type: application/json" -X POST -d '{"delay": 1000}' http://<containerIp>:3000/delay
```

To reset the delay configuration, just do a `POST` with 0 as the delay value.

Prepare your JMeter script with cookies erased (this will simulate new clients for each requests) and 10 threads this will simulate 10 concurrent users.

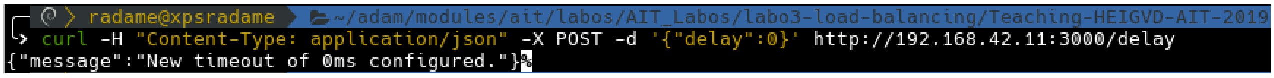
Remark: In general, take a screenshot of the summary report in JMeter to explain what is happening.

Deliverables:

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

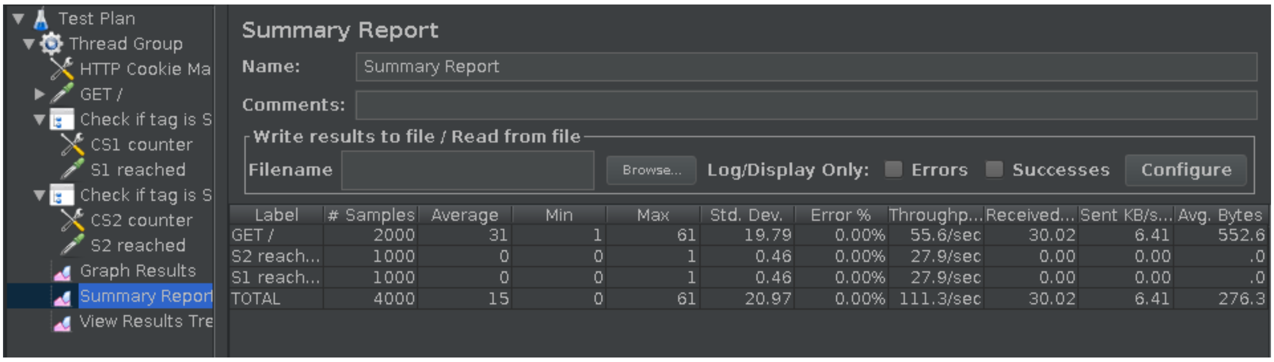
1. Be sure the delay is of 0 milliseconds is set on `s1`. Do a run to have base data to compare with the next experiments.

   To be sure that the delay are set on 0 on s1, we do a POST with 0 as the delay value.

   ```
   ╭─ radame@xpsradame  ~/adam/modules/ait/labos/AIT_Labos/labo3-load-balancing/Teaching-HEIGVD-AIT-2019
   ╰─> curl -H "Content-Type: application/json" -X POST -d '{"delay":0}' http://192.168.42.11:3000/delay
   {"message":"New timeout of 0ms configured."}%
   ```

   The following results will be a base to compare with the next experiments.



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

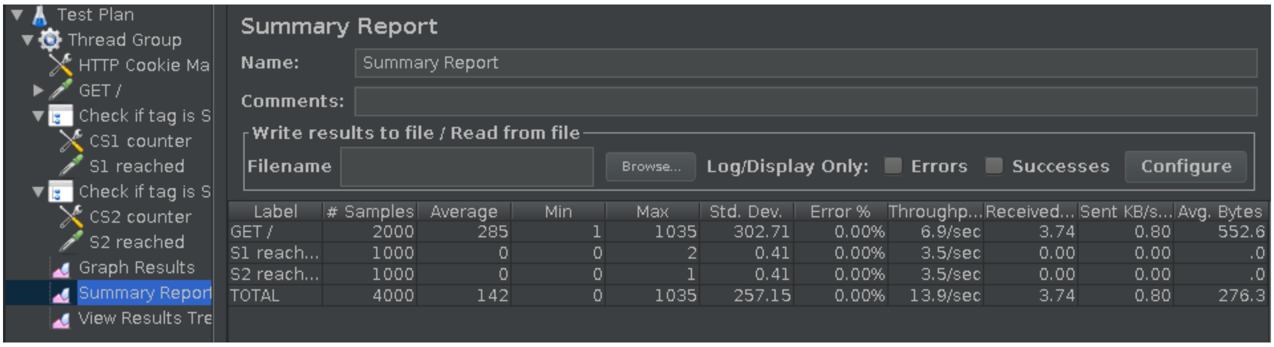
Filename

Log/Display Only: □ Errors □ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughp...	Received...	Sent KB/s...	Avg. Bytes
GET /	2000	31	1	61	19.79	0.00%	55.6/sec	30.02	6.41	552.6
S2 reach...	1000	0	0	1	0.46	0.00%	27.9/sec	0.00	0.00	.0
S1 reach...	1000	0	0	1	0.46	0.00%	27.9/sec	0.00	0.00	.0
TOTAL	4000	15	0	61	20.97	0.00%	111.3/sec	30.02	6.41	276.3

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

   Wee that the throughput is 10 times less than the base result. Therefore HAProxy still manage correctly the requests.



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

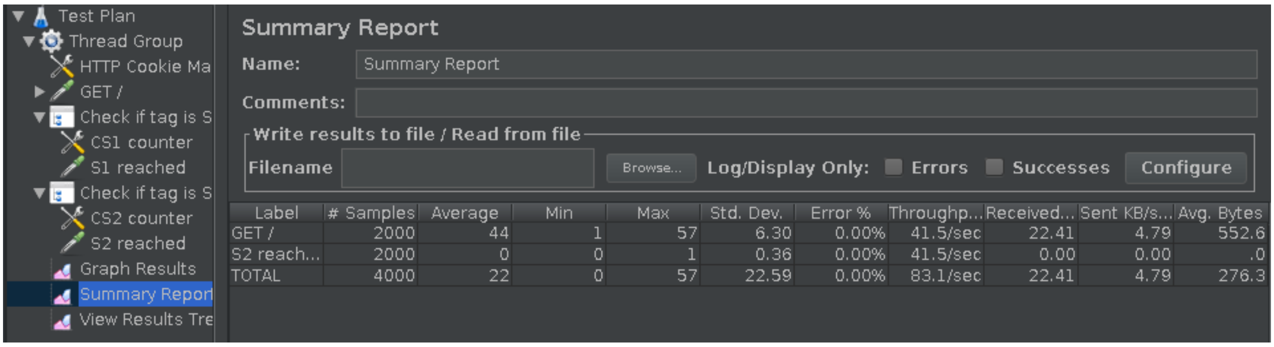
Filename

Log/Display Only: □ Errors □ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughp...	Received...	Sent KB/s...	Avg. Bytes
GET /	2000	285	1	1035	302.71	0.00%	6.9/sec	3.74	0.80	552.6
S1 reach...	1000	0	0	2	0.41	0.00%	3.5/sec	0.00	0.00	.0
S2 reach...	1000	0	0	1	0.41	0.00%	3.5/sec	0.00	0.00	.0
TOTAL	4000	142	0	1035	257.15	0.00%	13.9/sec	3.74	0.80	276.3

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

   Now, with ad delay of 2500 ms on s1, it is totally unreachable. All requests are routed to s2.



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename

Log/Display Only: □ Errors □ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughp...	Received...	Sent KB/s...	Avg. Bytes
GET /	2000	44	1	57	6.30	0.00%	41.5/sec	22.41	4.79	552.6
S2 reach...	2000	0	0	1	0.36	0.00%	41.5/sec	0.00	0.00	.0
TOTAL	4000	22	0	57	22.59	0.00%	83.1/sec	22.41	4.79	276.3

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

5. Update the HAProxy configuration to add a weight to your nodes. For that, add `weight [1-256]` where the value of weight is between the two values (inclusive). Set `s1` to 2 and `s2` to 1. Redo a run with 250ms delay.

6. Now, what happened when the cookies are cleared between each requests and the delay is set to 250ms ? We expect just one or two sentence to summarize your observations of the behavior with/without cookies.

Task 5: Balancing strategies

In this part of the lab, you will be less guided and you will have more opportunity to play and discover HAProxy. The main goal of this part is to play with various strategies and compare them together.

We propose that you take the time to discover the different strategies in HAProxy documentation and then pick two of them (can be round-robin but will be better to chose two others). Once you have chosen your strategies, you have to play with them (change configuration, use Jmeter script, do some experiments).

Deliverables:

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

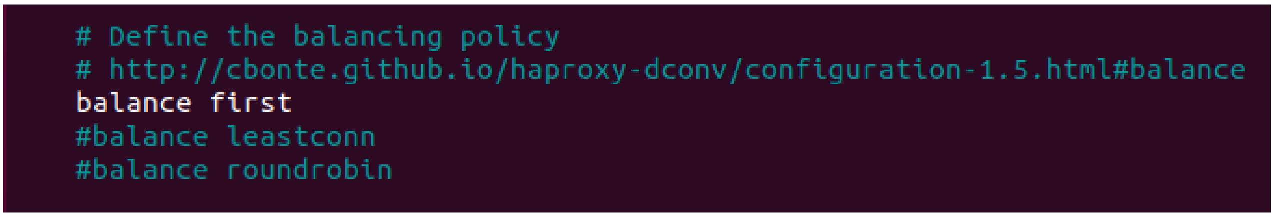
   In the HAProxy documentation there are many different type of load balancing strategies. We've decided to choose these two :

   first : This strategy is simple. The first server with available connections will receive the connection. They are chosen from the lowest to the highest ID. When the server reaches the "maxconn" value, the next server is used. But we have to be sure to set the "maxconn" setting if not, it doesn't make sense to use this strategy.

   leastconn : In this case, the server with the lowest number of connections will receive the connection. If multiple servers may be chosen, round robin is performed to ensure to use them all at lease one time. This strategy is not very well recommended for short HTTP sessions. But in case of long sessions such as LDAP, SQL, etc. it's a quite good strategy.

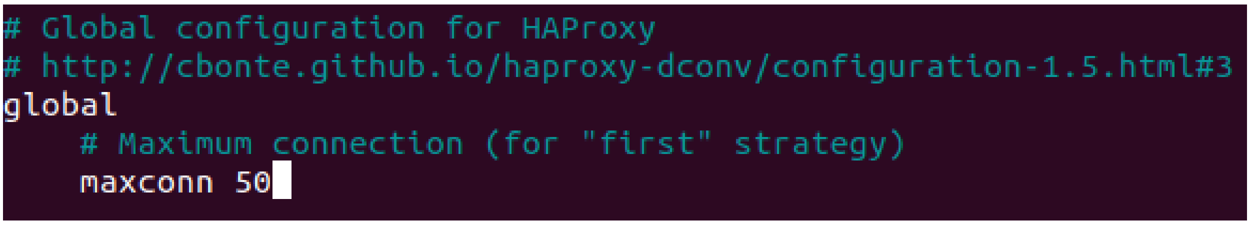
2. Provide evidences that you have played with the two strategies (configuration done, screenshots, …)

   For both strategies, we have to change the configuration file "haproxy.cfg" in order to use it. In the section "backend nodes", you will find the balancing policy :



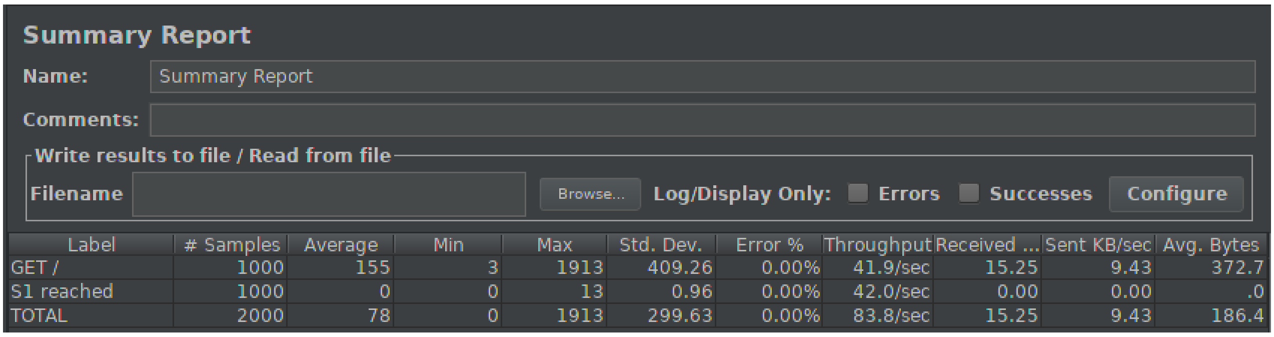
```
    # Define the balancing policy
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#balance
    balance first
    #balance leastconn
    #balance roundrobin
```

   As said, for the "first" strategy, we have to put a maxconn setting in order to have some results. For this you have to put in the section "global" the setting like this (We have tested with different maxconn values):



```
# Global configuration for HAProxy
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#3
global
    # Maximum connection (for "first" strategy)
    maxconn 50
```

   Here are the result for the "first" strategy :



Summary Report

Name: Summary Report

Comments:

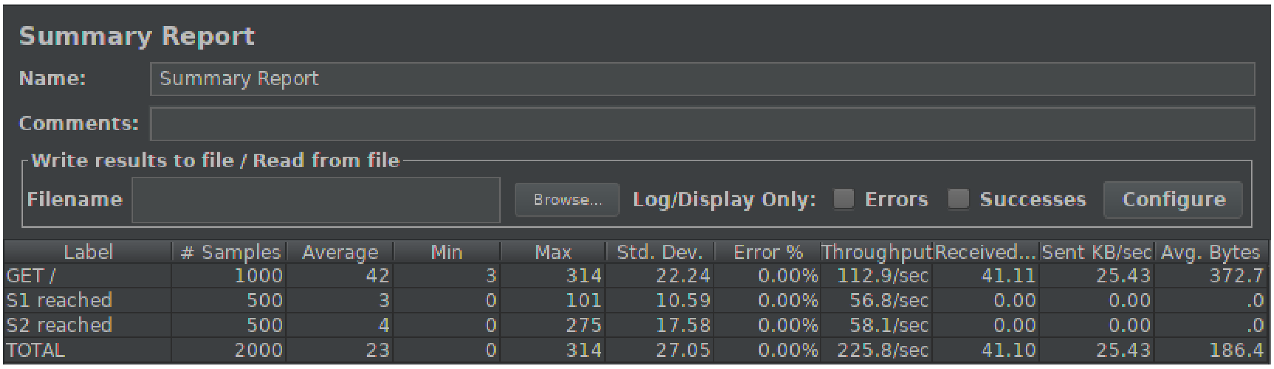
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
GET /	1000	155	3	1913	409.26	0.00%	41.9/sec	15.25	9.43	372.7
S1 reached	1000	0	0	13	0.96	0.00%	42.0/sec	0.00	0.00	.0
TOTAL	2000	78	0	1913	299.63	0.00%	83.8/sec	15.25	9.43	186.4

   We see that the connection goes always to server 1 (10 users and 100 loops here). We have

tried several configurations on Jmeter and in the “maxconn” setting (here maxconn is 1). The S1 always respond, it never balance to S2. This solution is not adapted for our use.

For leastconn strategy, we have made few Jmeter load test and it seems to be a good load balancer. We have made two load test using cookies and not. The tests are made with 10 user and 100 loops.

Here are the results using the cookies :



Summary Report

Name: Summary Report

Comments:

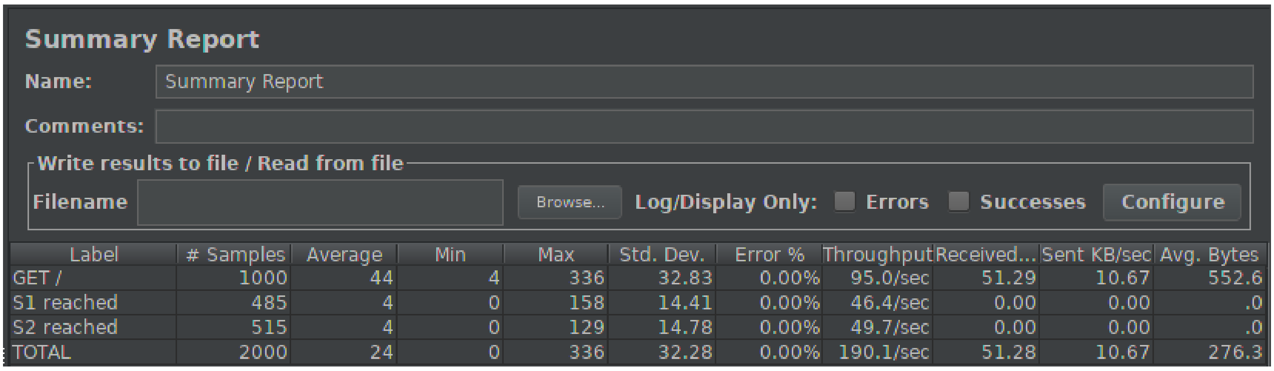
Write results to file / Read from file

Filename

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received...	Sent KB/sec	Avg. Bytes
GET /	1000	42	3	314	22.24	0.00%	112.9/sec	41.11	25.43	372.7
S1 reached	500	3	0	101	10.59	0.00%	56.8/sec	0.00	0.00	.0
S2 reached	500	4	0	275	17.58	0.00%	58.1/sec	0.00	0.00	.0
TOTAL	2000	23	0	314	27.05	0.00%	225.8/sec	41.10	25.43	186.4

We can see that the repartition is very good.

Here are the results when not using cookies :



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received...	Sent KB/sec	Avg. Bytes
GET /	1000	44	4	336	32.83	0.00%	95.0/sec	51.29	10.67	552.6
S1 reached	485	4	0	158	14.41	0.00%	46.4/sec	0.00	0.00	.0
S2 reached	515	4	0	129	14.78	0.00%	49.7/sec	0.00	0.00	.0
TOTAL	2000	24	0	336	32.28	0.00%	190.1/sec	51.28	10.67	276.3

Results are not bad at all.

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

   In comparison, we see that these two strategies are quite different. They have advantage and disadvantage. Before choosing one, we have to be sure what we want to do. The “first” strategy doesn’t suit well for this lab, as we want to give charges on both server not only on one. So the best one between these two is the “leastconn” one, as we can see the results, the charges are well balanced between the two servers.

Conclusion

This laboratory allowed us to the test in different way the tools JMeter and HAProxy, these are very useful and have many different options in order to test load balancing in a multi-tiered web application. We will keep in mind that load balancing can have strange behaviour without careful setup.