Labo 03 - Load balancing

Authors : Adrien Barth, Jeremy Zerbib

Introduction

The goal of this lab is to be familiar with a load-balancing tool like `HAProxy`.

We did all the asked tasks in this lab, going from installing the tools to tweaking the configuration files and playing with them.

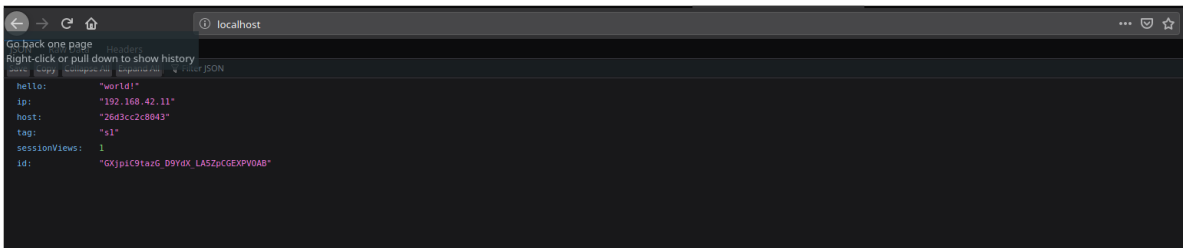
The goals in details were :

- Deploy a web application in a two-tier architecture for scalability
- Configure a load balancer
- Performance-test a load-balanced web application

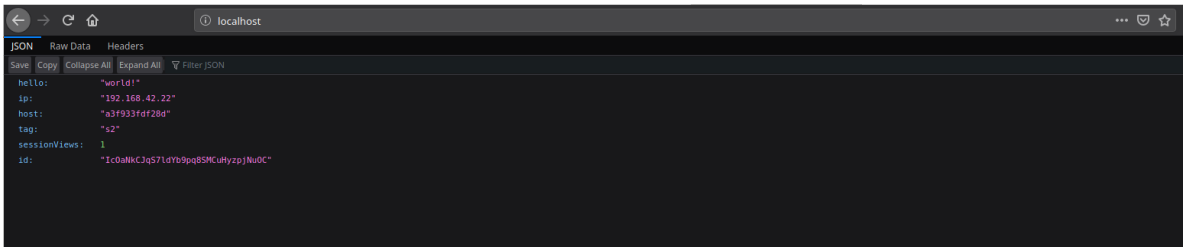
Task 1: Install the tools

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 first connect to the URL http://192.168.42.42, we can see this page appearing :

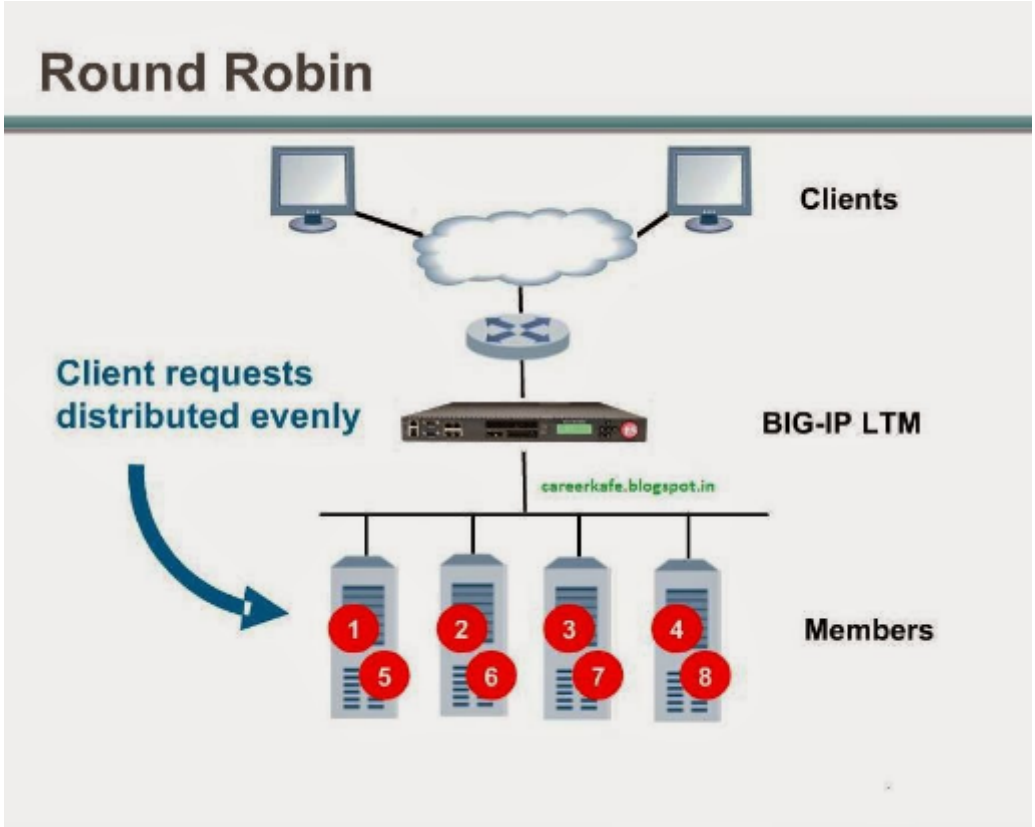


Then, we refresh the page and we see that the page loading is the following :



The load-balancing is of the type *Round-Robin*. Basically, what is happening is that the proxy is redirecting the client once on one server, once on the other. We can see that happening because of the change of the `tag`, the `host`, the `ip` and the `id` fields.

The basic functionality of a Round-Robin is explained below :

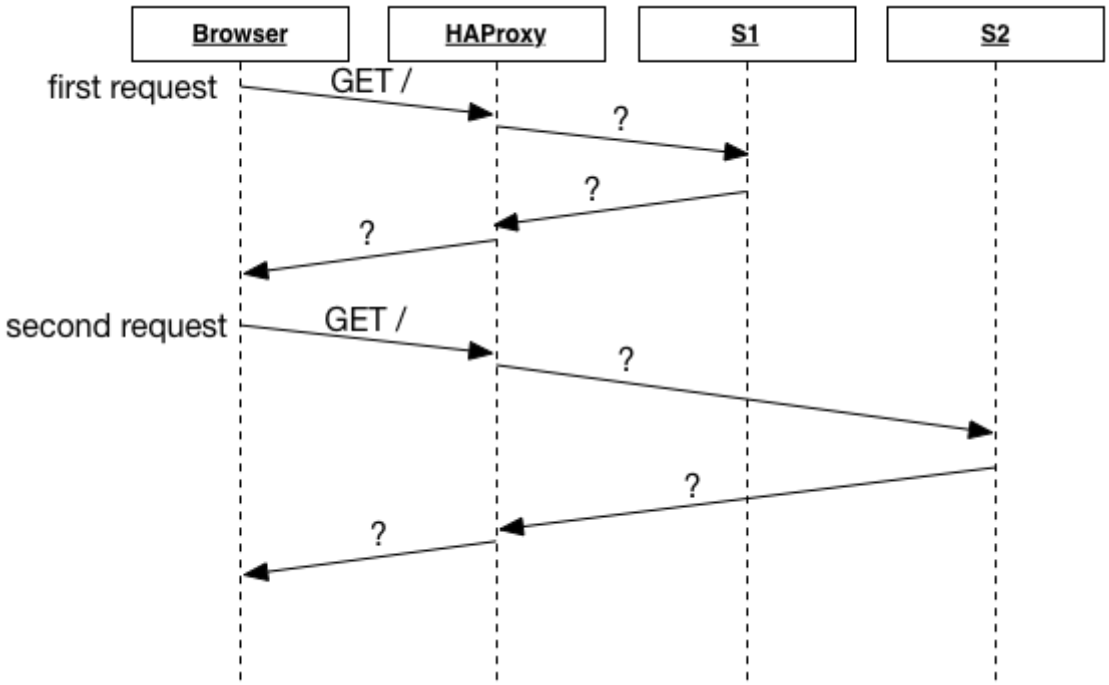


The proxy creates a queue with the servers lined up. It goes from one server to another and queues the used server back to its original spot.

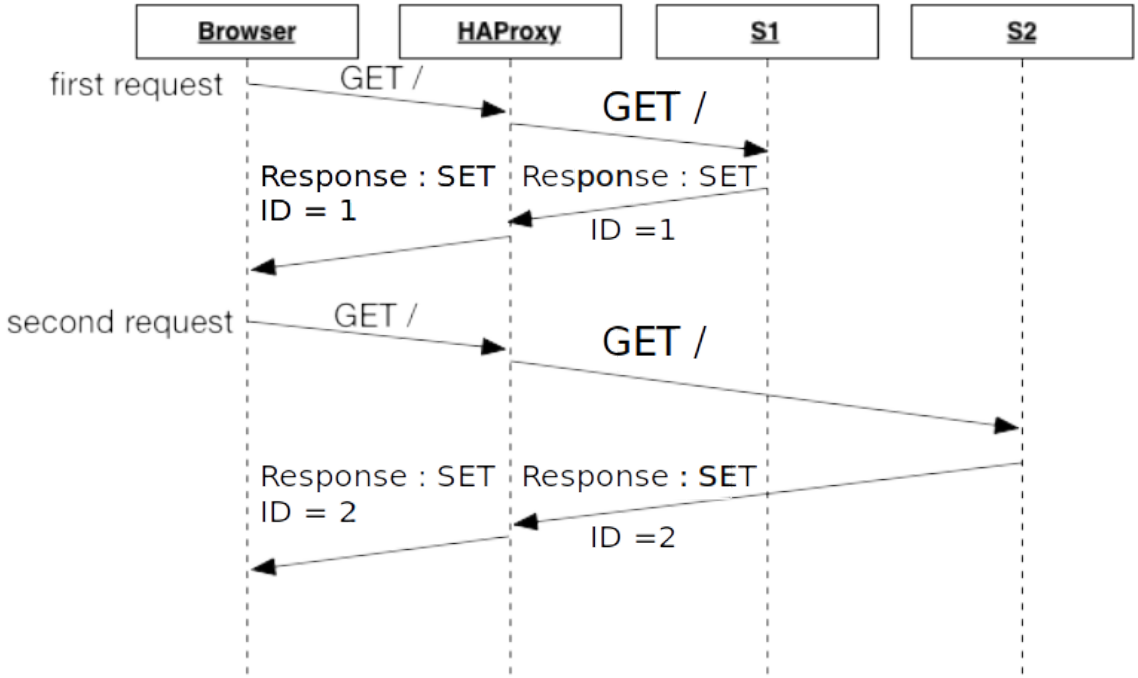
Explain what should be the correct behavior of the load balancer for session management.

The correct behavior for the session management would be to keep a given client connected to a server. It should be obvious that a client wants to keep his session opened while reloading a page. For example, if a clients aims to keep his number of connection to a server accurate, he might want to be connected every time to the same server.

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:



As the proxy runs a *Round-Robin* configuration, the communications go like this :

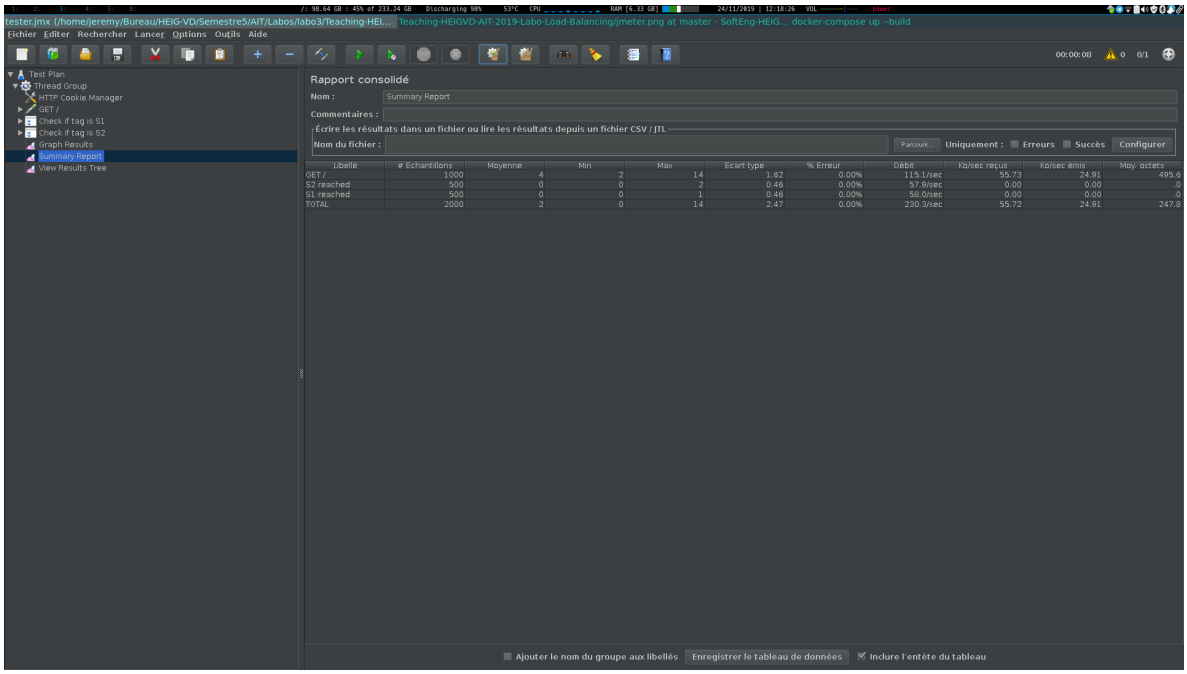


We can see that the balancing operates in a way that during the first request, the client goes to the server *S1* and sets up an ID for each request to the client. The client stores the token ID but the server does not know what to do with it, so it applies its *Round-Robin* queue algorithm.

Therefore, the client is redirected to the server *S2* and as the server does not know what to do with the ID token, it will set up a new ID.

This mechanism will go on as long as we keep refreshing the page. We can see that ID is never the same.

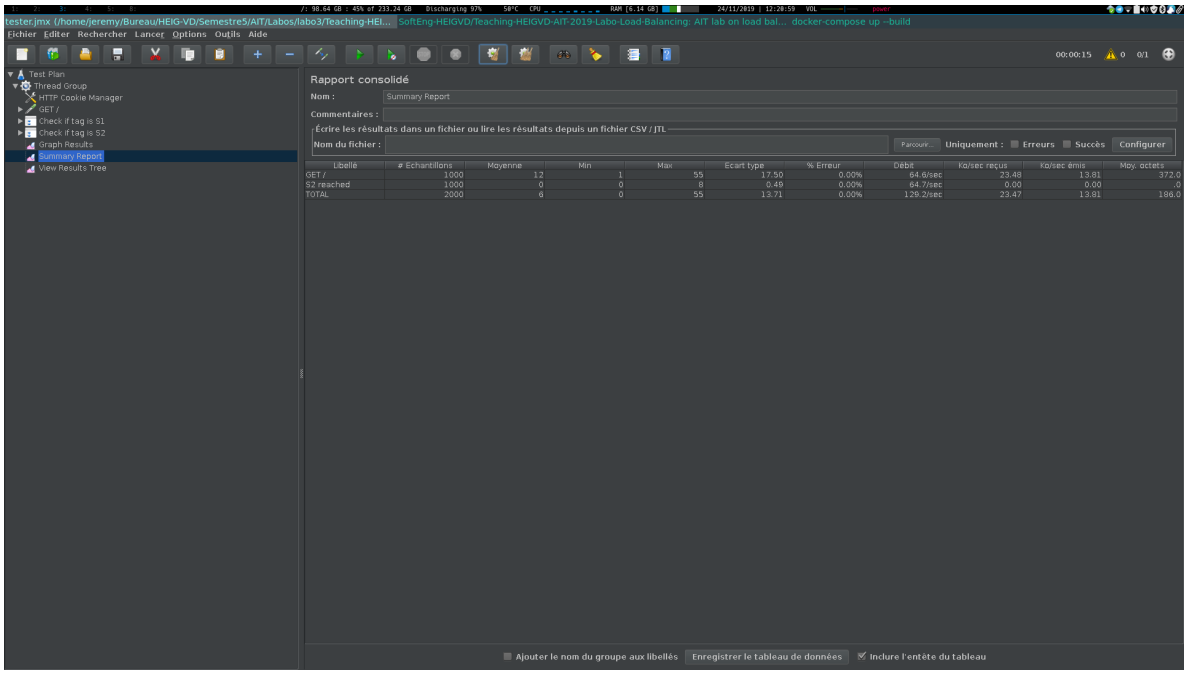
Provide a screenshot of the summary report from JMeter.



Run the following command:

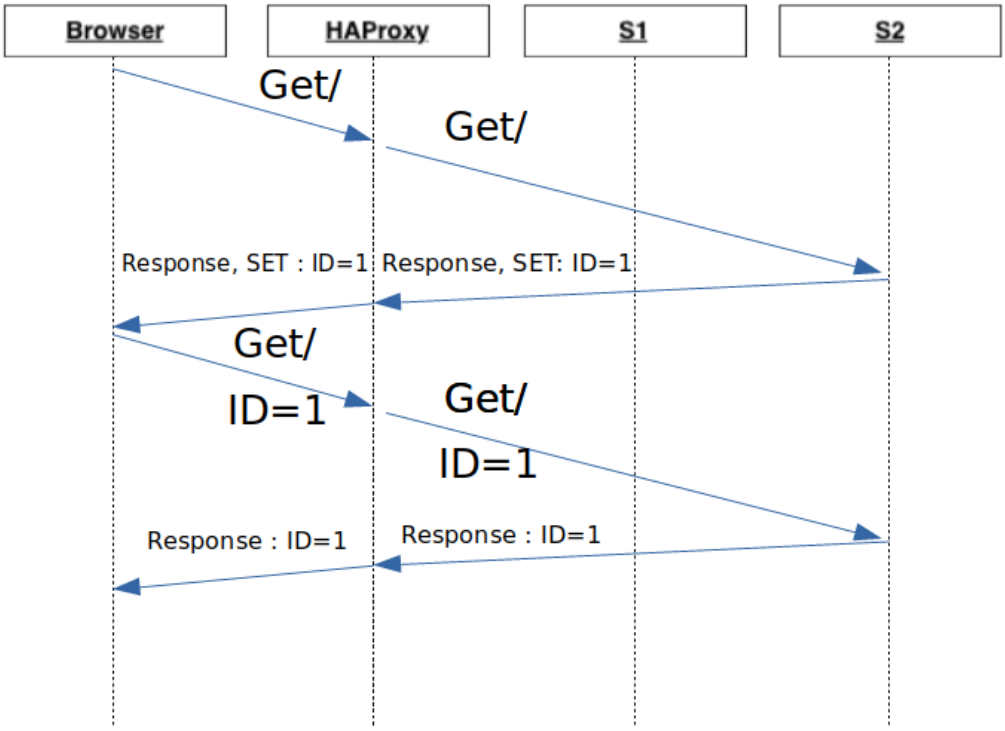
```
$ docker stop s1
```

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.



Basically, what is happening is that the client tries to get to *S1*. As S1 is not available, it goes to *S2* one time out of two. We can see we try to `GET` 2000 times but get a hit only half of those tries. The ID stays the same throughout the test because the server knows the first ID and therefore no need to create a new one. An user can potentially keep his session alive.

The sequence diagram shows what we is happening.



Task 2: Sticky sessions

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).

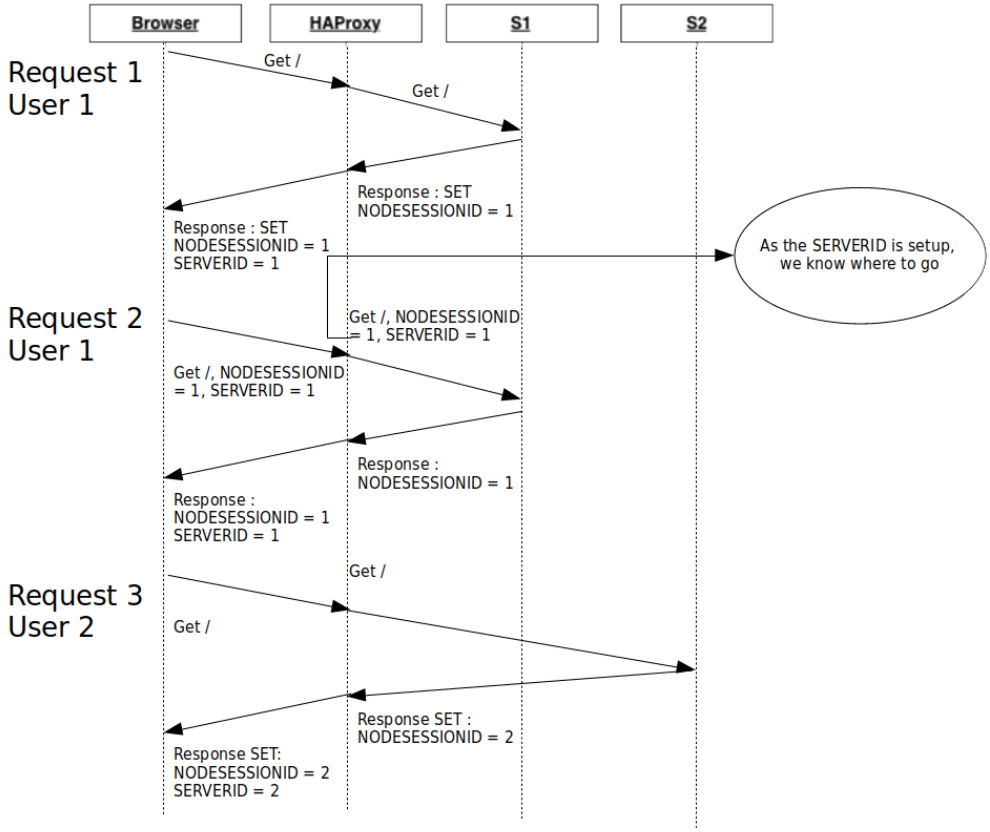
- **Choose one of the both stickiness approach for the next tasks.**

The difference between the *SERVERID* and the *NODESESSID* methods lives in the fact that the first is on a proxy side and the latter is on the client's side. Indeed, the cookie we want to create, is produced by the proxy in the *SERVERID*'s case and on client's side for the *NODESESSID*.

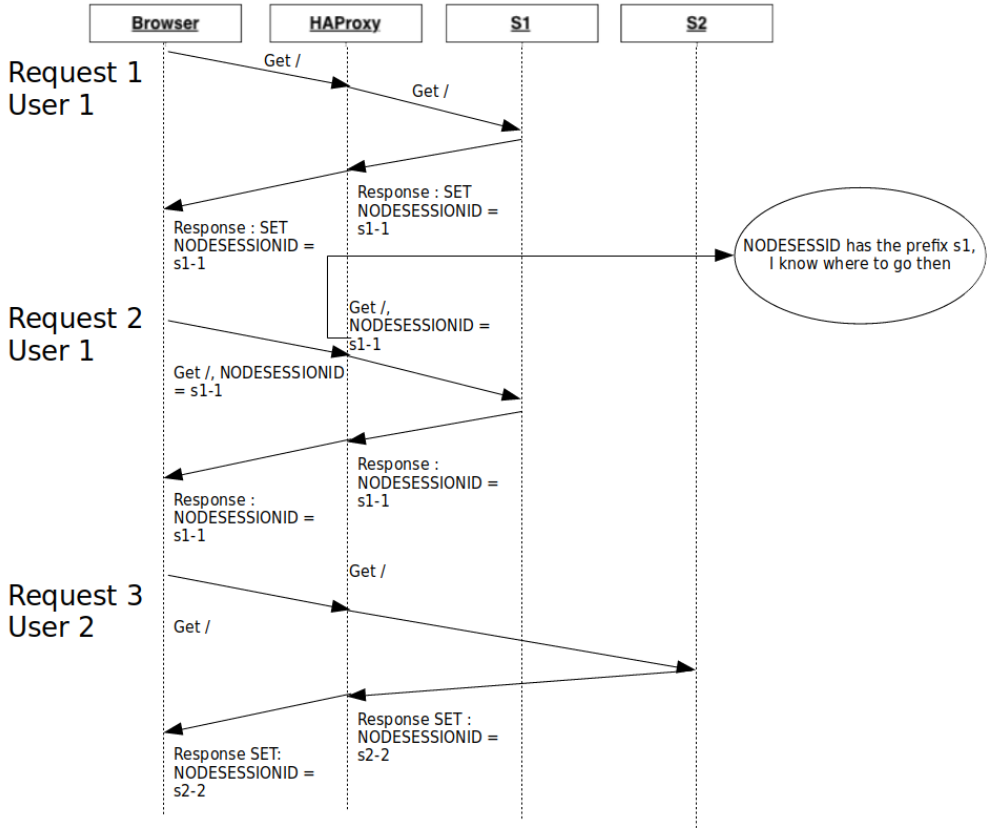
In the first case, the proxy will produce the *ID* and sticks to the packet, called SERVERID, only if the user did not start communicating with such cookie.

In the last case, the proxy will use the cookie that the client produces and sticks a prefix to it, which will be cleaned prior to transmitting the request to the server.

- SERVERID



- NODESESSID



`For the remaining of this task, we will be using the *SERVERID* stickiness method.`

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

You can see here how we got the configuration file edited in order to set up the configuration file that produces the cookie server side.

To summarize everything, here is how we do it.

We must edit the configuration file located in : `ha/config/haproxy.cfg`.

After the `backend nodes` label, right after the line `option forwardfor`, we added this line `cookie SERVERID insert indirect nocache`.

Then, we added at the end of the line `server s1 ${WEBAPP_1_IP}:3000 check` `cookie s1` and the same for `s2`.

What it does is basically adding a *cookie* named *SERVERID* if it does not exist. Then while connecting to the server, the load-balancer will check if there is a cookie available for the session.

If there is, then nothing is created and the server follows the session.

If there isn't a cookie setup, then the proxy will create a cookie and create a new session.

```
# Global configuration for HAProxy
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#3
global
    # Bind UNIX socket to get various stats
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#stats
    stats socket /var/run/haproxy.sock mode 600 level admin

    # Bind TCP port to get various stats
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#stats
    stats socket ipv4@0.0.0.0:9999 level admin
```

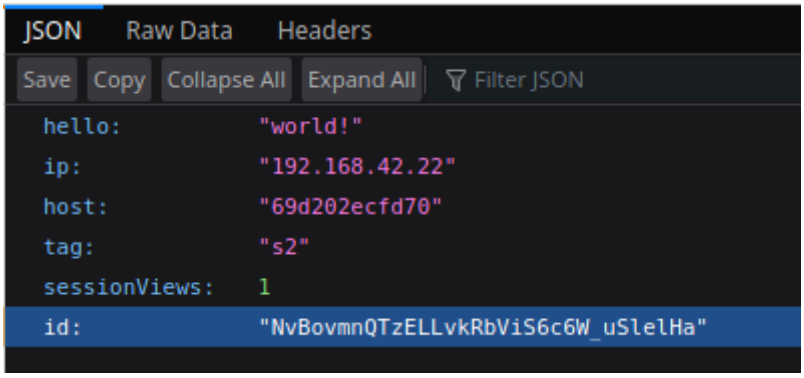
```
    # Define the timeout on the stats
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#3.1-
stats%20timeout
    stats timeout 30s

    # Configure the way the logging is done
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#log
    log 127.0.0.1 local1 notice

# Configure defaults for all the proxies configuration (applied for all the next
sections in the configuration)
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4
defaults
    # Enable logging
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-log
    log     global

    # The default mode for all the services
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-bind
    mode    http

    # Enable the logging of HTTP requests
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-
option%20httplog
    option  httplog

    # Enable the logging of null connections
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-
option%20dontlognull
    option  dontlognull

    # Configure the timeout to connect to a server
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-
timeout%20connect
    timeout connect 5000

    # Configure the timeout before cutting the connection of a client
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-
timeout%20client
    timeout client  50000

    # Same kind of configuration for the servers side
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-
timeout%20server
    timeout server  50000

# Open the metrics HAProxy page on the port 1936 on any network interface on the
host
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4
listen stats *:1936
    # Enable HAProxy to serve stats about himself and the nodes
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-
stats%20enable
    stats enable

    # Define the URI to access the stats
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-stats%20uri
    stats uri /
```

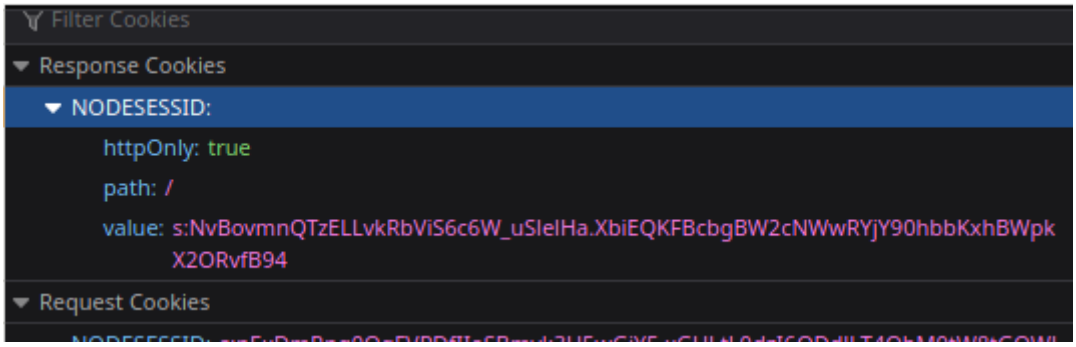
```
    # Avoid leaking more info than necessary with hiding the version of HAProxy
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-
stats%20hide-version
    stats hide-version

# Define the frontend configuration. In fact, that's the part that configure how
HAProxy will handle
# the requests from the outside world:
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4
frontend localnodes
    # Bind the port 80 to listen incoming outside connections (from the outside
world)
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-bind
    bind *:80

    # Define which protocol is enabled on the binded ports.
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-mode
    mode http

    # Use the backend configuration references by the backend name section in
this configuration
    # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-
default_backend
    default_backend nodes

# Define the backend configuration. In fact, that's the part that configure what
is not directly
# accessible from the outside of the network.
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4
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

    cookie SERVERID insert indirect nocache

    # 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

# Other links you will need later for this lab
#
# About cookies: http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-cookie
```

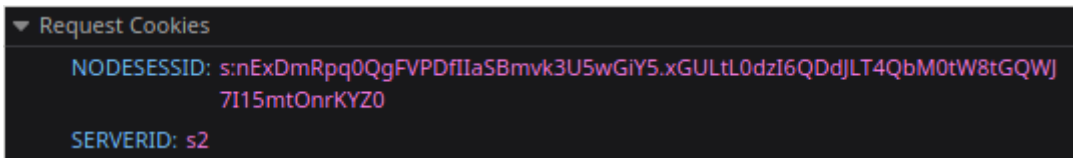
Explain what is the behavior when you open and refresh the URL http://192.168.42.42 in your browser. Add screenshots to complement your explanations. We expect that you take a deeper a look at session management.

After editing the configuration file, we can see the expected behavior through those screenshots :





We can see here that the first time we sent the request, `sessionview` has a value of 1, that we receive only the `NODESESSID` cookie.



We can also see that we send the new cookie, `SERVERID` with the request. The value is set to *S2* as we started on the server *S2* at the beginning of our request.

After reloading the page, we get this :

```
JSON   Raw Data   Headers
Save   Copy   Collapse All   Expand All   Filter JSON
hello:          "world!"
ip:             "192.168.42.22"
host:           "69d202ecfd70"
tag:            "s2"
sessionViews:   2
id:             "NvBovmnQTzELLvkRbViS6c6W_uSlelHa"
```

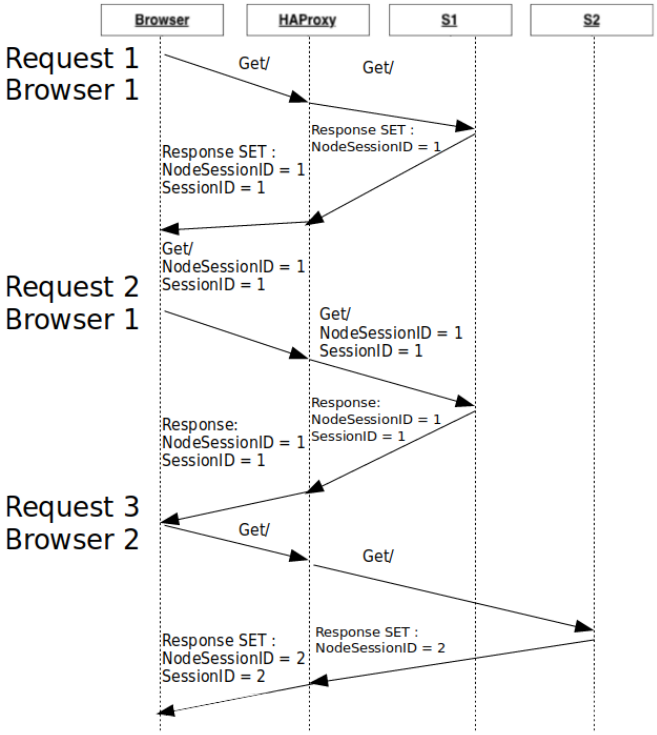

The cookie received is the following :


```
Filter Cookies
Request Cookies
    NODESESSID: s:NvBovmnQTzELLvkRbViS6c6W_uSlelHa.XbiEQKFBcbgBW2cNWwRYjY90hbbKxhBWp
                kX2ORvfB94
    SERVERID: s2
```


We see that the session ID is saved thanks to the `SERVERID`.

The session can stick if I do not change the user doing the request.

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.

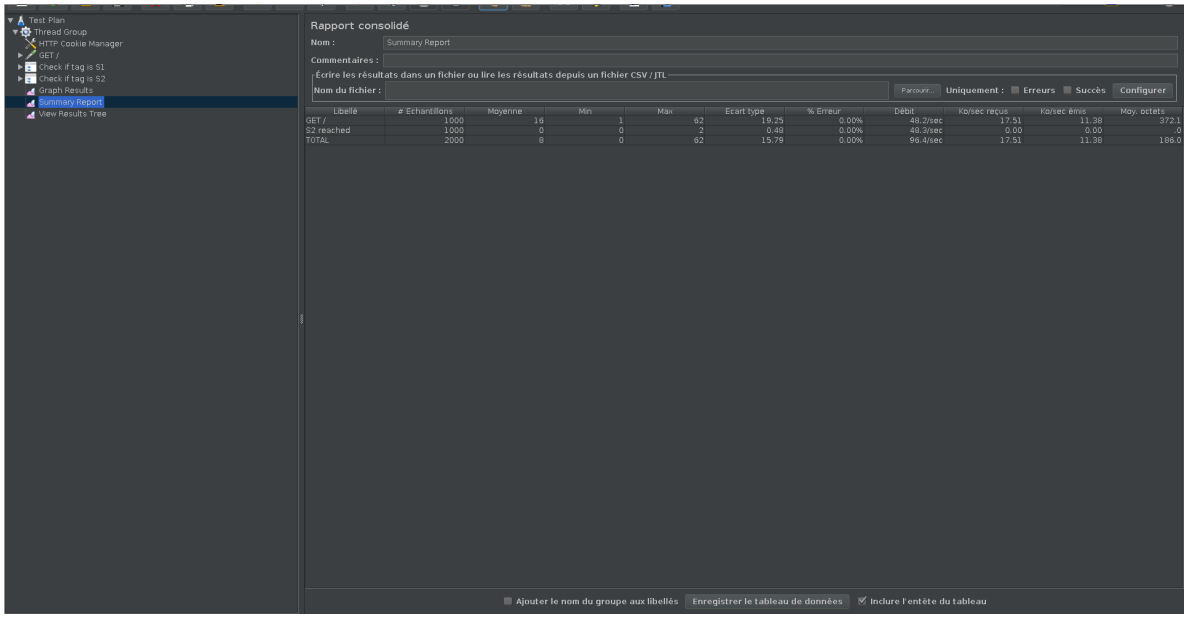


For this part we can see the exchange between all the parts of the circuit in the sequence diagram above.

When a user uses a second browser, what happens is the following :

- First the user connect to *S1* on the first browser.
- The proxy goes to the first server in the *Round-Robin* queue. (*S1* in our case)
- The server responds with the *NodeSessionID* appropriate for the session.
- The proxy creates a new cookie, *SERVERID*, and sticks it to the session.
- While the user connects from the same browser, the session will stick.
- If the user opens a new browser, or a private navigation window on the same browser, the session will change.

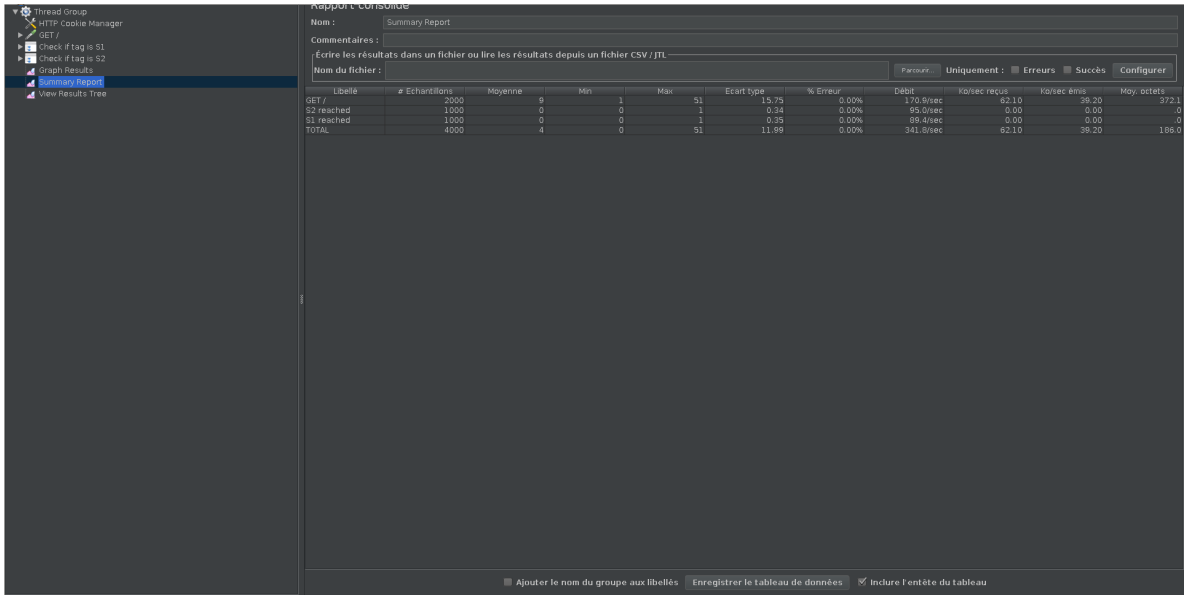
Provide a screenshot of JMeter's summary report. Is there a difference with this run and the run of Task 1?



There is a difference between this situation and the one in *Task 1*. It lies in the fact that only on server is reached in this situation because of the cookie we set. The proxy makes sure that the session sticks throughout the thread's life. That's the reason why we never reach the *S1* server.

- **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.**

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

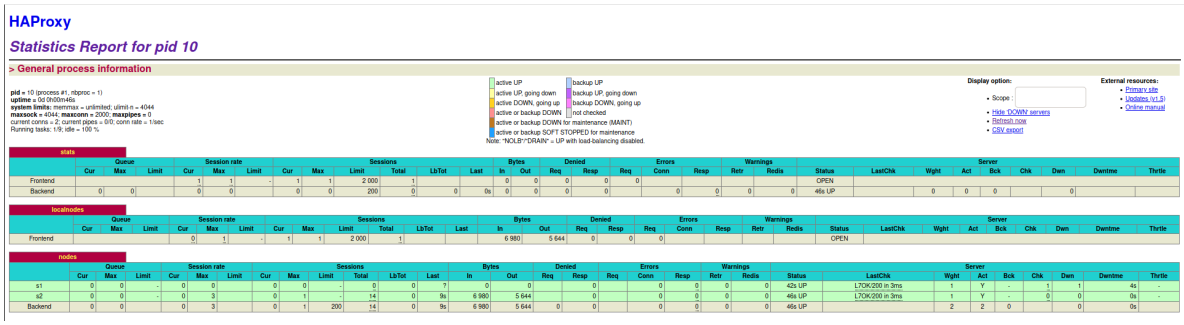


The load balancer is doing its job, by the looks of it.

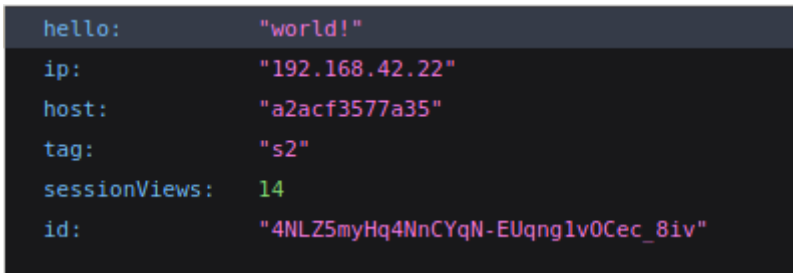
We can that the first request is reaching the *S2* server first then the second thread is redirected to *S1*. The *Round-Robin* algorithm is doing what it is supposed to do and the load is equally balanced between the two servers.

Task 3 : Drain mode

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

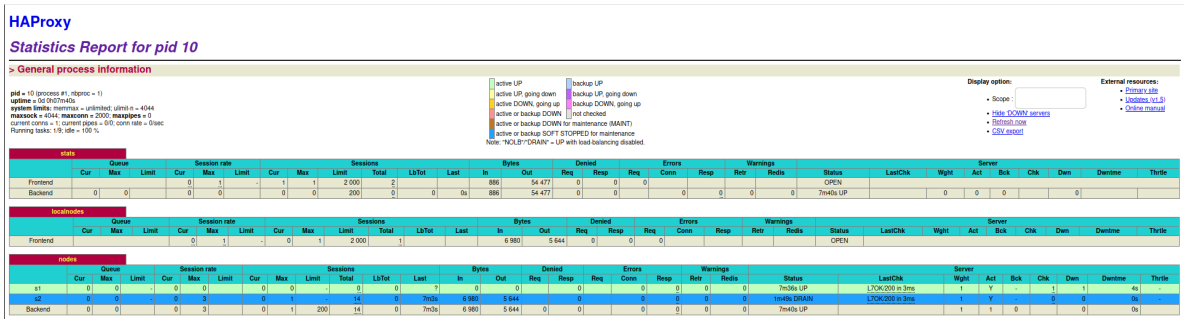


Given the info on the browser, the node *S2* is reached every time and we can see that, given the cookie we set up prior to this task, that we stay in this node.



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

```
~/Téléchargements/apache-jmeter-5.2/bin  socat - tcp:192.168.42.42:9999
prompt

>

> help
Unknown command. Please enter one of the following commands only :
  clear counters : clear max statistics counters (add 'all' for all counters)
  clear table    : remove an entry from a table
  help           : this message
  prompt         : toggle interactive mode with prompt
  quit           : disconnect
  show info      : report information about the running process
  show pools     : report information about the memory pools usage
  show stat      : report counters for each proxy and server
  show errors    : report last request and response errors for each proxy
  show sess [id] : report the list of current sessions or dump this session
  show table [id]: report table usage stats or dump this table's contents
  get weight     : report a server's current weight
  set weight     : change a server's weight
  set server     : change a server's state or weight
  set table [id] : update or create a table entry's data
  set timeout    : change a timeout setting
  set maxconn    : change a maxconn setting
  set rate-limit : change a rate limiting value
  disable        : put a server or frontend in maintenance mode
  enable         : re-enable a server or frontend which is in maintenance mode
  shutdown       : kill a session or a frontend (eg:to release listening ports)
  show acl [id]  : report available acls or dump an acl's contents
  get acl        : reports the patterns matching a sample for an ACL
  add acl        : add acl entry
  del acl        : delete acl entry
  clear acl <id> : clear the content of this acl
  show map [id]  : report available maps or dump a map's contents
  get map        : reports the keys and values matching a sample for a map
  set map        : modify map entry
  add map        : add map entry
  del map        : delete map entry
  clear map <id> : clear the content of this map
  set ssl <stmt> : set statement for ssl

> set timeout cli 1d

> set server nodes/s2 state drain

>
```



We can see that we are in `drain` mode on the *S2* node.

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?

```
JSON  Raw Data  Headers
Save  Copy  Collapse All  Expand All  Filter JSON
hello:          "world!"
ip:             "192.168.42.22"
host:           "a2acf3577a35"
tag:            "s2"
sessionViews:   22
id:             "4NLZ5myHq4NnCYqN-EUqng1vOCec_8iv"
```


After a few refresh, we can see that we stayed on the same node. The fact that we put the `Drain` mode on means that we are blocking any connection on another server but any user can establish a new session on this server. It blocks any load balancing but still allow any health-checks or a new persistent connection. (cf. slides from the course on `High performance systems` slide 40).

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

We are going to the other server, *S1*.

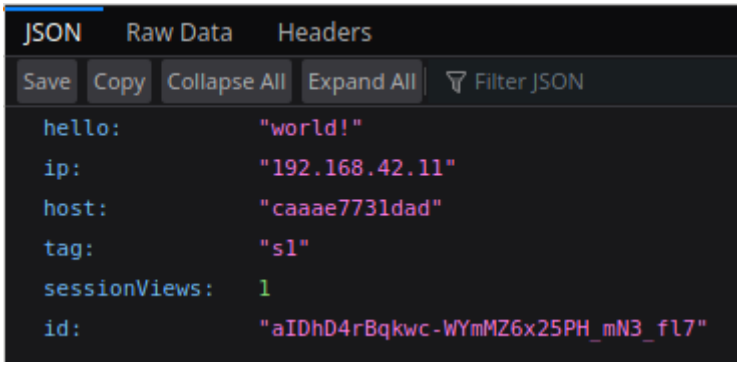

```
JSON  Raw Data  Headers
Save  Copy  Collapse All  Expand All  Filter JSON
hello:          "world!"
ip:             "192.168.42.11"
host:           "caaae7731dad"
tag:            "s1"
sessionViews:   10
id:             "NSPkVGXcngyUsYbr-9s6D0GJ3Qr6Lcz5"
```


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?


```
JSON  Raw Data  Headers
Save  Copy  Collapse All  Expand All  Filter JSON
hello:          "world!"
ip:             "192.168.42.11"
host:           "caaae7731dad"
tag:            "s1"
sessionViews:   1
id:             "Tvt80P6zwSHdAEc1c-bZfwB8ASNmuAWi"
```

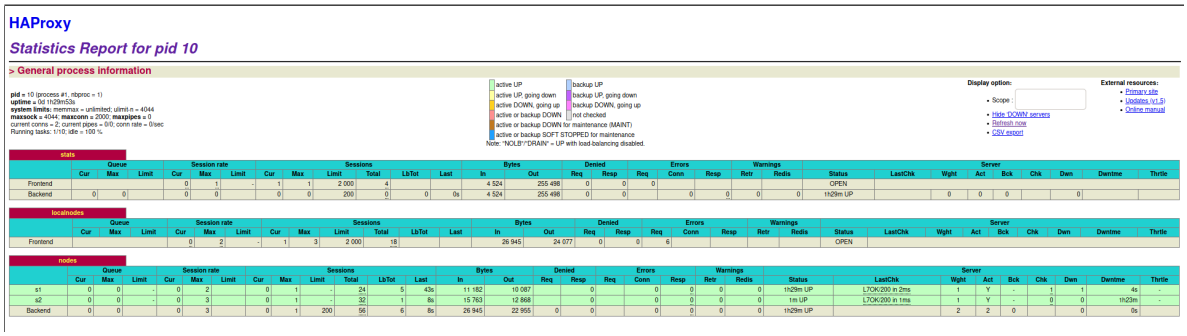


```
JSON  Raw Data  Headers
Save  Copy  Collapse All  Expand All  Filter JSON
hello:          "world!"
ip:             "192.168.42.11"
host:           "caaae7731dad"
tag:            "s1"
sessionViews:   1
id:             "403XA8PSbsst_SBE58M6WnPVSaPbSo3n"
```



After multiple tries, we can see that we are always reaching the node *S1*, the on that is not on `DRAIN` mode, and we cannot reach the drained node. We can see that the session ids are different every time. The server has no way of knowing that the user is the same so that's why it assigns a new ID every time.

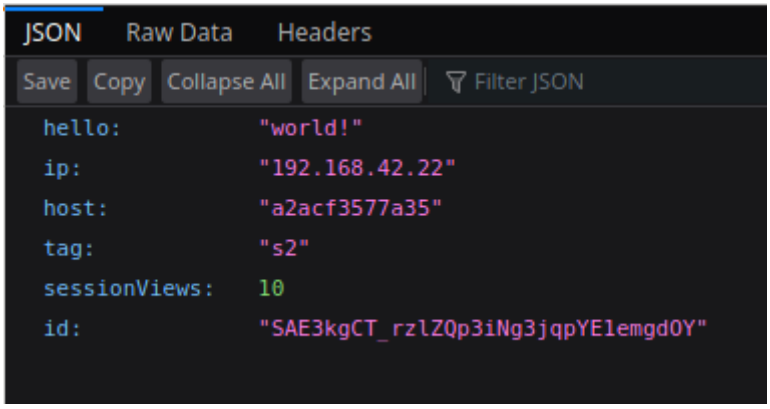
Reset the node in READY mode. Repeat the three previous steps and explain what is happening. Provide a screenshot of HAProxy's stats page.



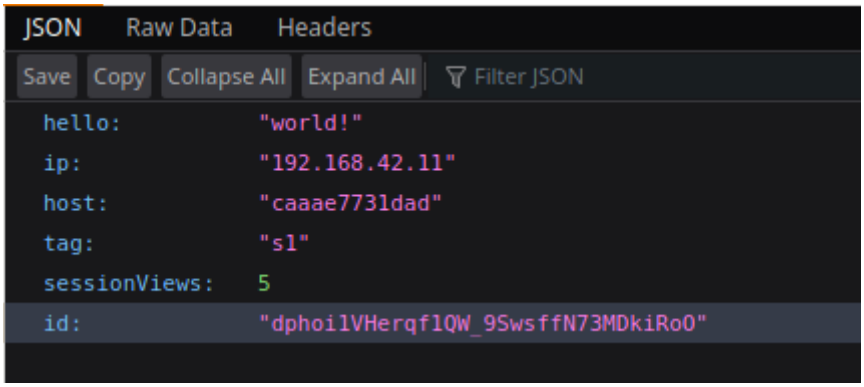
The node is reset to `READY` which means normal mode according to the slides on `High performance systems` slide 40 from the course material.

Therefore, the behavior is the one we expected.

First, we connect to a node, *S2* in our case, and if refresh the page a few times, we stay on it, thanks to the cookie set up in task 2.

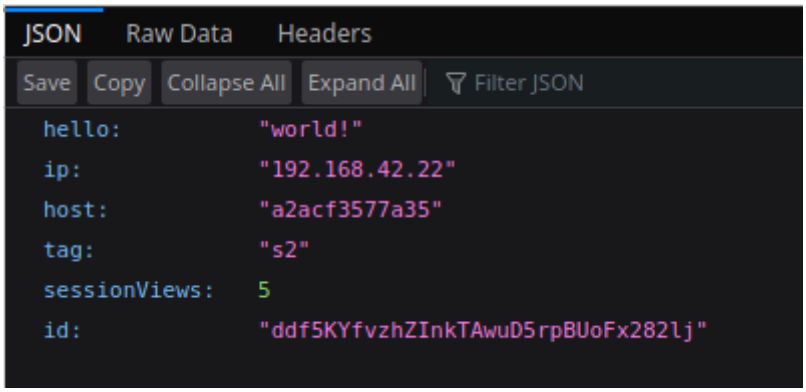


Then, we open another browser and go to the *S1* node.



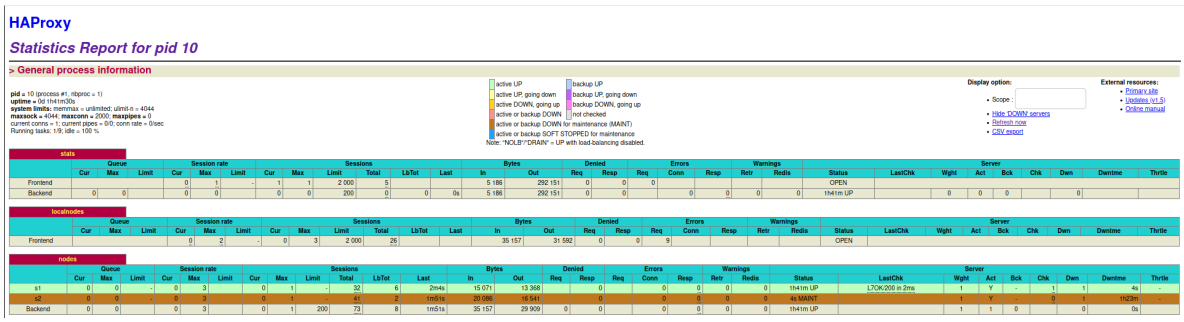
We can see that we are redirected to the other node, as a normal *Round-Robin* algorithm would do. But we might need something more to convince you.

So we open a third browser and connect to server.

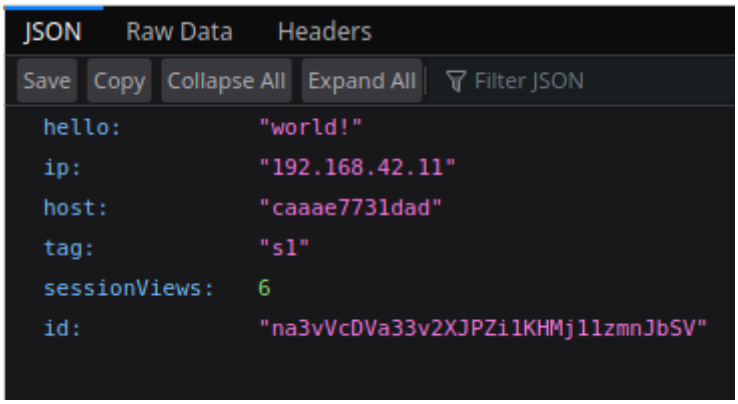


We see that we are redirected to *S2*.

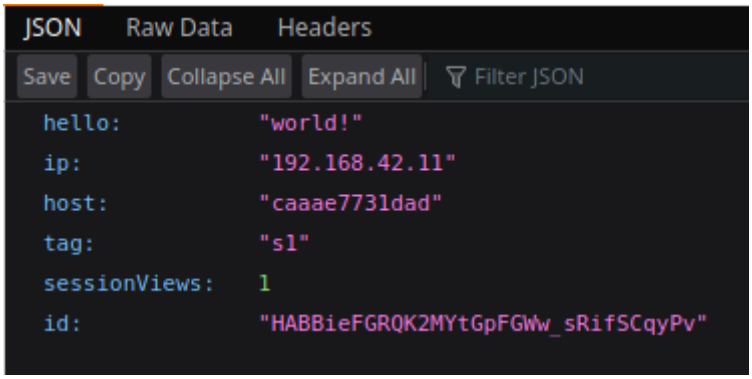
Finally, set the node in MAINT mode. Redo the three same steps and explain what is happening. Provide a screenshot of HAProxy's stats page.



We can see in the above screenshot that the *S2* node is set up in `MAINT` mode and below that when we connect to the server, it cannot reach the *S2* node.



Even when we try on another browser.



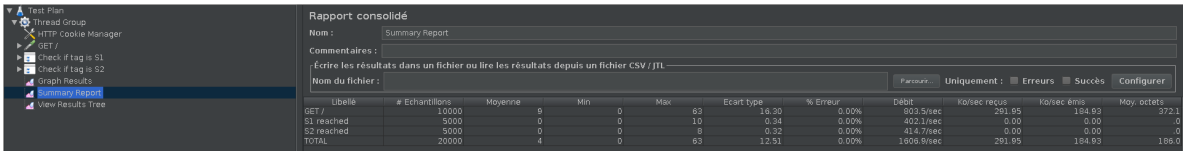
We can see a different ID but accessing the same server.

Task 4 : Round robin in degraded mode.

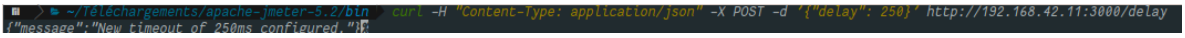
We based our task on a previous version of the questions so we set the requests to execute 10 threads.
We did this because if we didn't we couldn't see the load-balancing operation with one thread

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

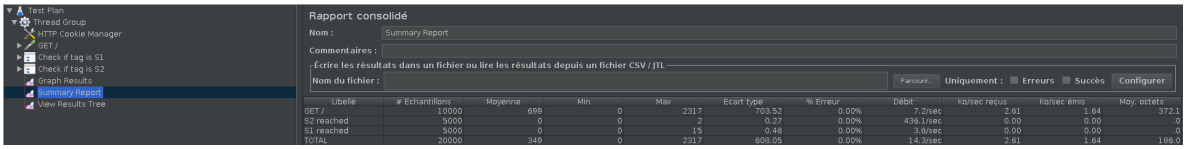
Be sure the delay is of 0 milliseconds is set on `s1`. Do a run to have base data to compare with the next experiments.



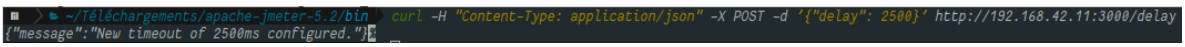
Set a delay of 250 milliseconds on `s1`. Relaunch a run with the JMeter script and explain what it is happening?



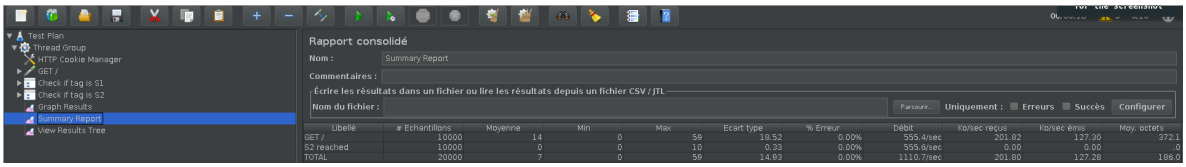
After increasing the delay between each answer on *S1*, we can see that it takes a significant amount of time to finish the requests, as on *S2* the time seems to be almost the same.



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



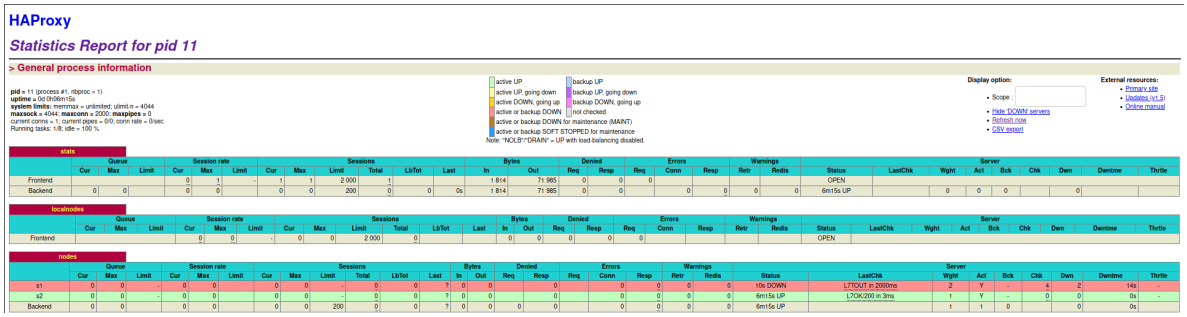
The *S1* node is not even attained by the users because of the timeout. Therefore, only *S2* is shown in the result.



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

`JMeter` does not prompt any error.

What happens is that the load balancer, `HAProxy`, analyses the delay and thinks that the *S1* is down. Therefore, JMeter does not think that the node has a problem but it receives the information that the node is down. Therefore, it is a normal behavior.



HAProxy

Statistics Report for pid 11

> General process information

pid = 11 (process #1, nbproc = 1)
uptime = 0d 0h06m15s
system limits: memmax = unlimited; ulimit-n = 4044
maxsock = 4044; **maxconn** = 2000; **maxpipes** = 0
current conns = 1; current pipes = 0/0; conn rate = 0/sec
Running tasks: 1/8; 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.

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				0	1	-	1	1	2 000	1			1 814	71 985	0	0	0					OPEN								
Backend	0	0		0	0		0	0	200	0	0	0s	1 814	71 985	0	0		0	0	0	0	6m15s UP		0	0	0		0		

localnodes	Queue			Session rate			Sessions						Bytes		Denied		Errors		Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	0	-	0	0	2 000	0				0	0	0	0					OPEN								

nodes	Queue			Session rate			Sessions						Bytes		Denied		Errors		Warnings			Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
s1	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	10s DOWN	L7TOUT in 2000ms	2	Y	-	4	2	14s	-
s2	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	6m15s UP	L7OK/200 in 3ms	1	Y	-	0	0	0s	-
Backend	0	0		0	0		0	0	200	0	0	?	0	0	0	0		0	0	0	0	6m15s UP		1	1	0		0	0s	

As we can see, the node is found to be down

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.

```
# Define the backend configuration. In fact, that's the part that configure what is not directly
# accessible from the outside of the network.
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4
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

    cookie SERVERID insert indirect nocache

    # 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 weight 2 check cookie s1
    server s2 ${WEBAPP_2_IP}:3000 weight 1 check cookie s2
```

```
■ ❯ ☁ ~/B/H/Semestre5/AIT/Labos/labo3/Teaching-HEIGVD-AIT-2019-Labo-Load-Balancing ❯ on ☁ master !3 ?4 ❯ curl -H "Content-Type: application/json" -X POST -d '{"delay": 250}' http://192.168.42.11:3000/delay
{"message":"New timeout of 250ms configured."}%
■ ❯ ☁ ~/B/H/Semestre5/AIT/Labos/labo3/Teaching-HEIGVD-AIT-2019-Labo-Load-Balancing ❯ on ☁ master !3 ?4 ❯ curl -H "Content-Type: application/json" -X POST -d '{"delay": 250}' http://192.168.42.22:3000/delay
{"message":"New timeout of 250ms configured."}%
```

We set a delay of 250 ms for the two nodes.

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	10000	2432	257	4354	628.30	0.00%	4.0/sec	1.46	0.91	377.2
S1 reached	2611	0	0	10	0.44	0.00%	1.1/sec	0.00	0.00	.0
S2 reached	7389	0	0	24	0.67	0.00%	2.9/sec	0.00	0.00	.0
TOTAL	20000	1216	0	4354	1294.92	0.00%	7.9/sec	1.46	0.91	188.6

The balance is not respected at all. As the weight condition is greater in the *S1* node, it is normal for a load balancer to put more requests directed towards the one that can hold more. As we set a delay on the two nodes, the execution is even longer than before.

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.

Libelle	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	10000	1783	253	5636	1005.94	0.00%	1.3/sec	0.72	0.15	552.7
S2 reached	4342	0	0	15	0.41	0.00%	34.6/min	0.00	0.00	.0
S1 reached	5658	0	0	18	0.54	0.00%	45.1/min	0.00	0.00	.0
TOTAL	20000	891	0	5636	1140.69	0.00%	2.7/sec	0.72	0.15	276.3

We can see that the *S1* node is more reached than *S2*. The reason for that is that the weight of the node is heavier in *S1* and cookies are reseting between each requests.

We can see that the balance is almost evenly shared with the cookie being reseted between every request.

Task 5 : Balancing strategies

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

The two strategies we chose are :

- `static-rr`
- `leastconn`

We chose those two strategies because :

- `static-rr` is almost the same as the one we saw during this lab and we wanted to know what the difference might be. It lies in the fact that you cannot perform any changes on the fly of the configuration.
- `leastconn` seems interesting in our case because it should behave the same way that `round-robin` did. The algorithm is also dynamic so we should be able to change weights on the fly. Furthermore, it is not recommended to use it in short sessions like `HTTP`. It is more suitable for `LDAP` connections, for example. The fact that the load might not be evenly distributed in this case is a possibility which we wanted to experiment.

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

- `static-rr` : We configured the load balancing in `static-rr`.

```
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

    # 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

    cookie SERVERID insert indirect nocache

    # 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 weight 2 check cookie s1
    server s2 ${WEBAPP_2_IP}:3000 weight 1 check cookie s2

# Other links you will need later for this lab
#
# About cookies: http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-cookie
#
```

  Then we launched the docker containers in order to test if the configuration was up.

As the behavior is pretty similar we had to test that if we modified on the fly the configuration and see if the changes were effective.
First we checked the configuration :

HAProxy

Statistics Report for pid 11

> General process information

pid = 11 (process #1, nbproc = 1)
uptime = 0d 0h00m38s
system limits: memmax = unlimited; ulimit-n = 4044
maxsock = 4044; **maxconn** = 2000; **maxpipes** = 0
current conns = 1; current pipes = 0/0; conn rate = 0/sec
Running tasks: 1/8; 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.

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				0	1	-	1	1	2 000	1			926	53 908	0	0	0					OPEN								
Backend	0	0		0	0		0	0	200	0	0	0s	926	53 908	0	0		0	0	0	0	38s UP		0	0	0		0		

localnodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	0	-	0	0	2 000	0			0	0	0	0	0					OPEN								

nodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
s1	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	33s UP	L7OK/200 in 4ms	2	Y	-	1	1	5s	-
s2	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	38s UP	L7OK/200 in 3ms	1	Y	-	0	0	0s	-
Backend	0	0		0	0		0	0	200	0	0	?	0	0	0	0		0	0	0	0	38s UP		3	2	0		0	0s	

We can see that the configuration is up and running in `static-rr`.
Then we log into the HAProxy's Docker and modify the configuration.

```
~/B/H/Semestre5/AIT/L/labo3/Teaching-HEIGVD-AIT-2019-Labo-Load-Balancing/ha on master !2 ?6 docker exec -it ha /bin/bash
root@7bd1c1cffd2c:/# ls
bin   config  docker-entrypoint.sh  home  lib64  mnt  proc  run   scripts  sys  usr
boot  dev     etc                   lib   media  opt  root  sbin  srv      tmp  var
root@7bd1c1cffd2c:/# cd config/
haproxy.cfg   rsyslogd.cfg
root@7bd1c1cffd2c:/# cd config/
root@7bd1c1cffd2c:/config# nano haproxy.cfg
bash: nano: command not found
root@7bd1c1cffd2c:/config# vi haproxy.cfg
root@7bd1c1cffd2c:/config# cat haproxy.cfg
# Define the backend configuration. In fact, that's the part that configure what is not direct
# accessible from the outside of the network.
# http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4
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

    # 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

    cookie SERVERID insert indirect nocache

    # 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 weight 2 check cookie s1

# Other links you will need later for this lab
#
# About cookies: http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-cookie
#
root@7bd1c1cffd2c:/config#
```

We removed a node from the load-balancing configuration and we waited a few minutes before reloading the configuration page.
After reloading it, we checked if the node *S2* was removed or not.

HAProxy

Statistics Report for pid 11

> General process information

pid = 11 (process #1, nbproc = 1)
uptime = 0d 0h04m06s
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: 1/8; 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.

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	2			1 284	71 899	0	0	0					OPEN								
Backend	0	0		0	0		0	0	200	0	0	0s	1 284	71 899	0	0		0	0	0	0	4m6s UP		0	0	0		0		

localnodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	0	-	0	0	2 000	0			0	0	0	0	0					OPEN								

nodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
s1	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	4m1s UP	L7OK/200 in 2ms	2	Y	-	1	1	5s	-
s2	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	4m6s UP	L7OK/200 in 2ms	1	Y	-	0	0	0s	-
Backend	0	0		0	0		0	0	200	0	0	?	0	0	0	0		0	0	0	0	4m6s UP		3	2	0		0	0s	

We can see that *S2* is still present in the config.

- `leastconn` : We configured the load balancing in `leastconn` mode.

```
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

    # 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

    cookie SERVERID insert indirect nocache

    # 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
    server s2 ${WEBAPP_2_IP}:3000 check

# Other links you will need later for this lab
#
# About cookies: http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-cookie
#
```

We removed the cookie for the test.

We'll set them back later on.

We can see that the configuration is up and running :

HAProxy

Statistics Report for pid 10

> General process information

pid = 10 (process #1, nbproc = 1)
uptime = 0d 0h00m06s
system limits: memmax = unlimited; ulimit-n = 4044
maxsock = 4044; **maxconn** = 2000; **maxpipes** = 0
current conns = 1; current pipes = 0/0; conn rate = 0/sec
Running tasks: 1/8; 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.

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				0	1	-	1	1	2 000	1			384	17 936	0	0	0					OPEN								
Backend	0	0		0	0		0	0	200	0	0	0s	384	17 936	0	0		0	0	0	0	6s UP		0	0	0		0		

localnodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	0	-	0	0	2 000	0			0	0	0	0	0					OPEN								

nodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
s1	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	2s UP	L7OK/200 in 2ms	1	Y	-	1	1	4s	-
s2	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	6s UP	L7OK/200 in 4ms	1	Y	-	0	0	0s	-
Backend	0	0		0	0		0	0	200	0	0	?	0	0	0	0		0	0	0	0	6s UP		2	2	0		0	0s	

Then we connect for the first time.

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON


```
hello:          "world!"
ip:             "192.168.42.22"
host:           "791543f74bd3"
tag:            "s2"
sessionViews:   1
id:             "0wf5WXzdVjKnAMJTnlpU3XaHi9gGIY2F"
```


Then the second.

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

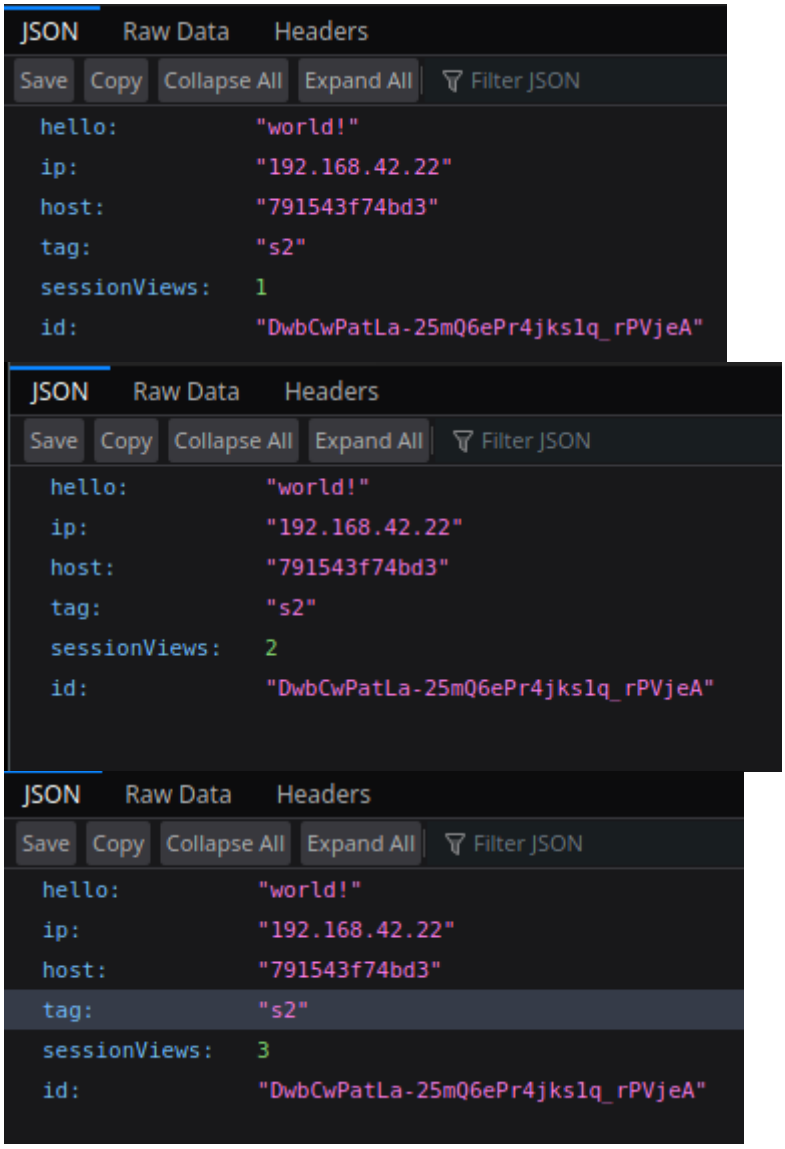

```
hello:          "world!"
ip:             "192.168.42.11"
host:           "0741d904b299"
tag:            "s1"
sessionViews:   1
id:             "VCNxzkSNWjAXmpCO2GRlOHYO30CSE1-Q"
```


We can see that the behavior is, expected the same as the `roundrobin` mode because we have two nodes.

Let's see what happens if we enable the cookies !

```
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

    # 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

    cookie SERVERID insert indirect nocache

    # 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

# Other links you will need later for this lab
#
# About cookies: http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#4-cookie
#
```

We then proceed to connect to the main page and refresh two times.



We can see that the cookie is always taken into account.
It overlaps the configuration as in `round robin` mode.

Compare the both strategies and conclude which is the best for this lab (not necessary the best at all).
For this lab, we think that the best mode would be `leatconn` as it is essentially the same behavior as `round-robin`.
`static-rr` is too constraint of a mode to make change on the fly.

Conclusion

To conclude this lab, we now know how to use and configure the `HAProxy` tool. We know which load-balancing strategy is the best to use and in which case to use it.