# Laboratoire 3 - Load Balancing

> Auteurs : Lucas Gianinetti, Nicolas Hungerbühler, Cassandre Wojciechowski
>
> Cours : AIT
>
> Date : 18.11.2021

## Introduction

// description globale du labo

## Tâche 1 - Installation des outils

> Les outils `Docker` et `docker-compose` sont déjà installés sur l'ordinateur utilisé pour ce laboratoire.

Nous avons donc installé `JMeter` avec la commande :

```
$ sudo apt install jmeter 'jmeter-*'
```

Nous avons ensuite lancé la première commande suivante dans le dossier racine du laboratoire, et la seconde pour vérifier que les conteneurs aient été lancés :

```
$ docker-compose up --build
$ docker ps
```

Nous constatons que trois conteneurs différents ont été démarrés : `balancing_haproxy`, `balancing_webapp1`, `balancing_webapp2`.

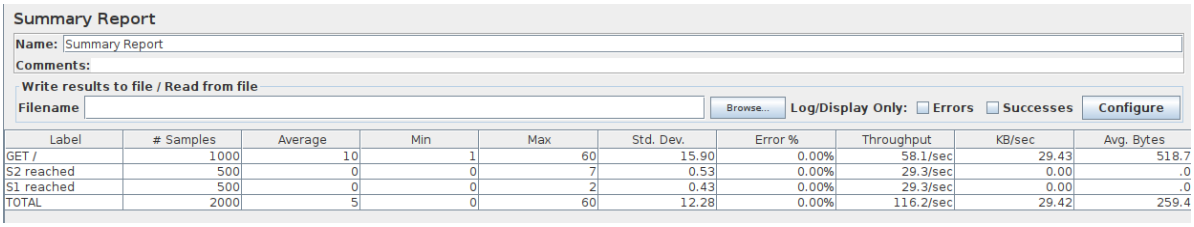
Nous vérifions que la configuration `bridge` est bien appliquée :

```
$ docker network ls
NETWORK ID     NAME                                                DRIVER
SCOPE
d2ee042a1ac2   bridge                                              bridge
local
13cfd9b0b7ae   docker_default                                      bridge
local
544dd0d58670   host                                                host
local
5f9a574fe881   none                                                null
local
5e029413fa29   teaching-heigvd-ait-2019-labo-load-balancing_public_net   bridge
local
```

En nous connectant via un browser sur l'adresse http://192.168.42.42:80, nous trouvons un document JSON :

```
{
  "hello": "world!",
  "ip": "192.168.42.11",
  "host": "8bc128915098",
  "tag": "s1",
  "sessionViews": 1,
  "id": "RYerKjPmA4IZXgnMCUvgwaiVzfoeqcCq"
}
```


Nous avons lancé le script `tester.jmx` depuis `JMeter` et nous observons le résultat suivant :

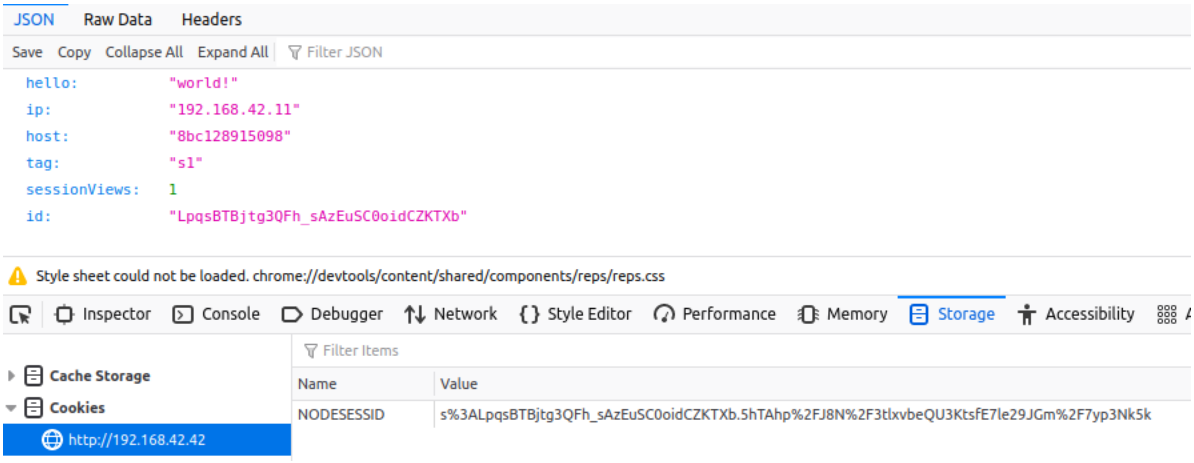



| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|
| GET / | 1000 | 10 | 1 | 60 | 15.90 | 0.00% | 58.1/sec | 29.43 | 518.7 |
| S2 reached | 500 | 0 | 0 | 7 | 0.53 | 0.00% | 29.3/sec | 0.00 | .0 |
| S1 reached | 500 | 0 | 0 | 2 | 0.43 | 0.00% | 29.3/sec | 0.00 | .0 |
| TOTAL | 2000 | 5 | 0 | 60 | 12.28 | 0.00% | 116.2/sec | 29.42 | 259.4 |

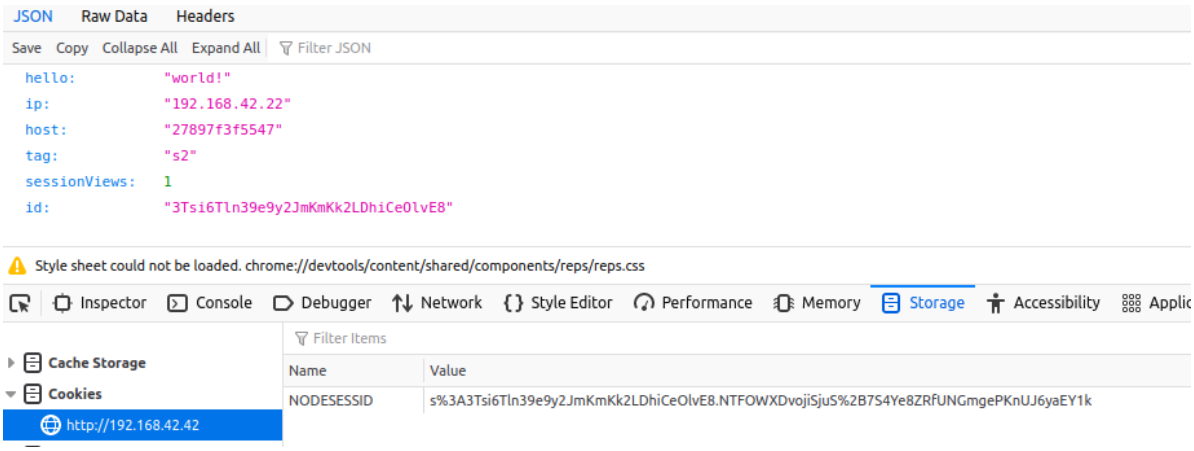
Nous constatons que les requêtes sont bien réparties équitablement entre les deux applications : 500 ont été dirigées sur `s1` et 500 sur `s2`.

**1.1 Explain how the load balancer behaves when you open and refresh the URL [http://192.168.42.42](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.**

La première requête que nous faisons nous fait apparaître le tag `s1` avec un id commençant par `Lpqs`.




La seconde fait apparaître le tag `s2` et un id commençant `3Tsi`.



Une requête sur deux aura le tag `s1` et l'autre aura donc le tag `s2`, nous pouvons donc déduire le load balancer alterne les redirections sur les deux web app.

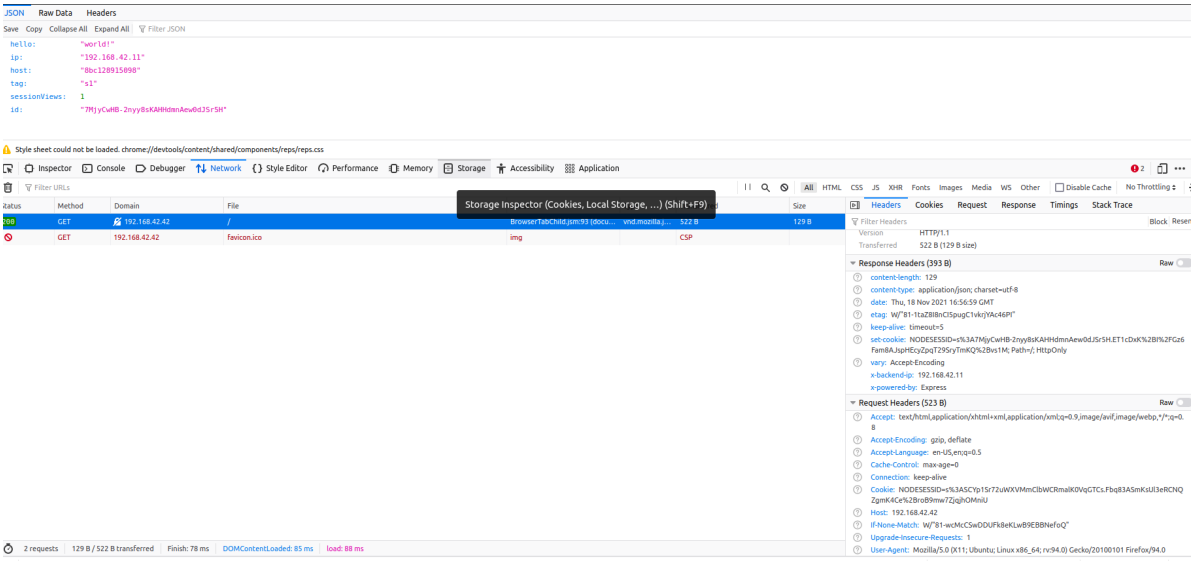
Par contre, les identifiants changent à chaque rafraichissement de page, donc nous changeons de session à chaque fois. Cela explique que le champ `sessionViews` reste toujours à 1.

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

Le comportement qui semble logique serait qu'une session reste active tant que l'utilisateur n'a pas quitté l'application web. Cela permettrait d'incrémenter correctement le compteur de requêtes envoyées.

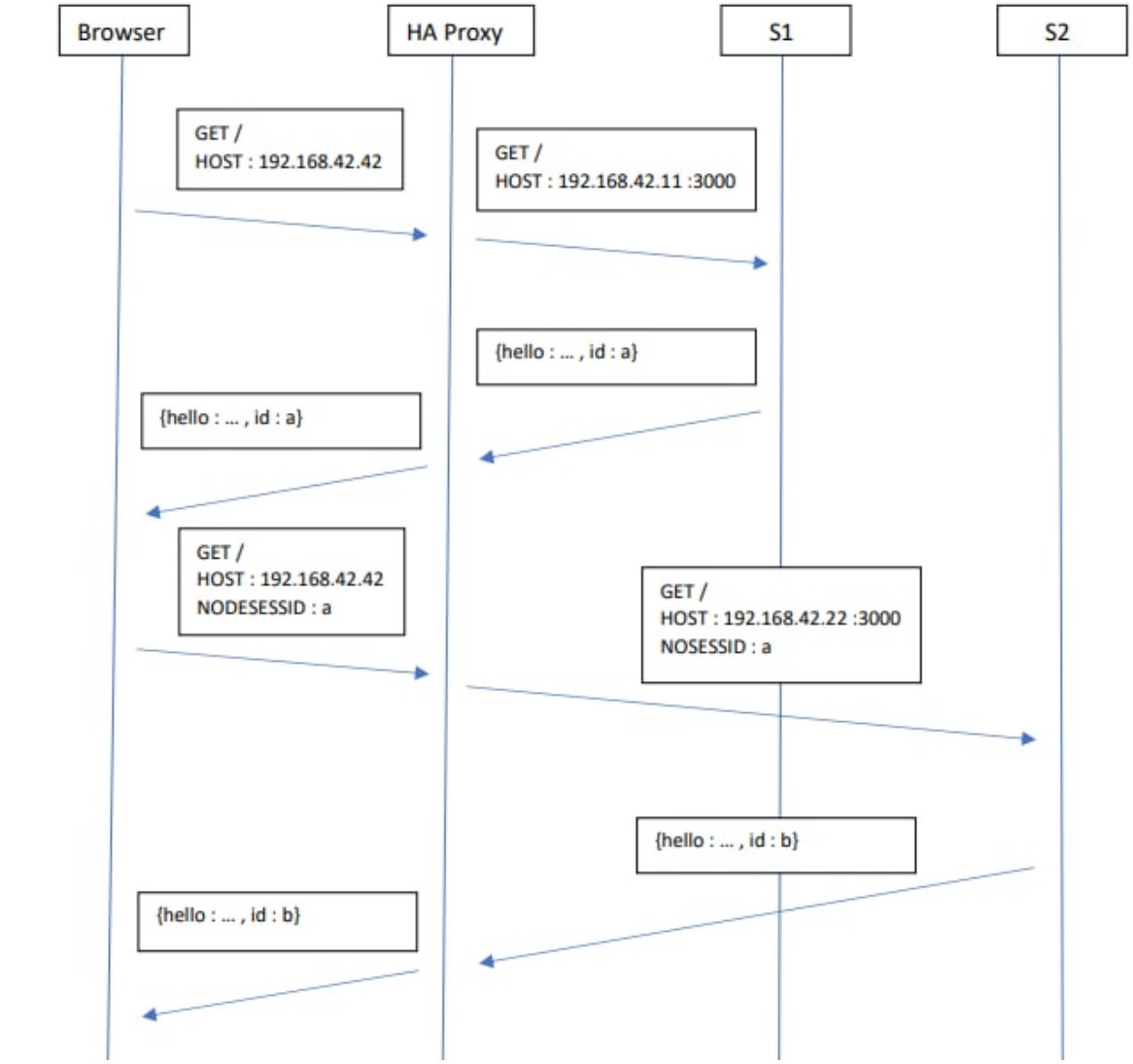
Par contre, il faut noter que l'utilisateur d'une session doit toujours être redirigé sur le même serveur.

### 1.3 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.

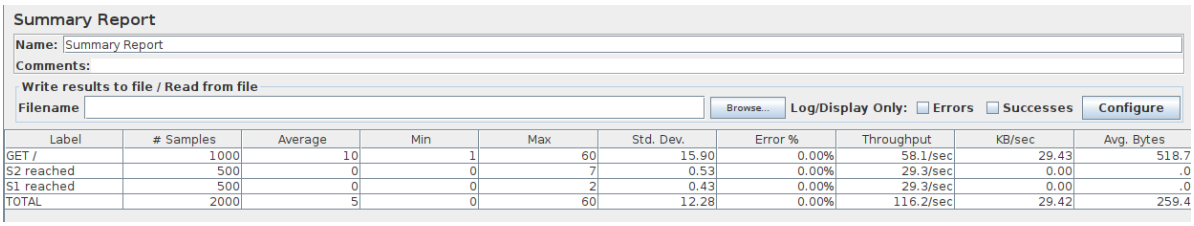



Lors de la première requête, la webapp 1 attribue un cookie à l'utilisateur. Lors de la seconde requête, ce premier cookie est envoyé à la webapp 2. Celle-ci ne le reconnait pas et en attribue donc un nouveau à l'utilisateur.

Le cookie est donc changé à chaque fois, car les webapps ne reconnaissent pas les cookies attribués par l'autre.




### 1.4 Provide a screenshot of the summary report from JMeter



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

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

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|
| GET / | 1000 | 10 | 1 | 60 | 15.90 | 0.00% | 58.1/sec | 29.43 | 518.7 |
| S2 reached | 500 | 0 | 0 | 7 | 0.53 | 0.00% | 29.3/sec | 0.00 | .0 |
| S1 reached | 500 | 0 | 0 | 2 | 0.43 | 0.00% | 29.3/sec | 0.00 | .0 |
| TOTAL | 2000 | 5 | 0 | 60 | 12.28 | 0.00% | 116.2/sec | 29.42 | 259.4 |

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

Nous arrêtons le conteneur s1 avec la commande :

```
$ docker stop s1
```

Après son arrêt, nous relançons JMeter et nous obtenons le rapport suivant :

Summary Report

Name: Summary Report

Comments:

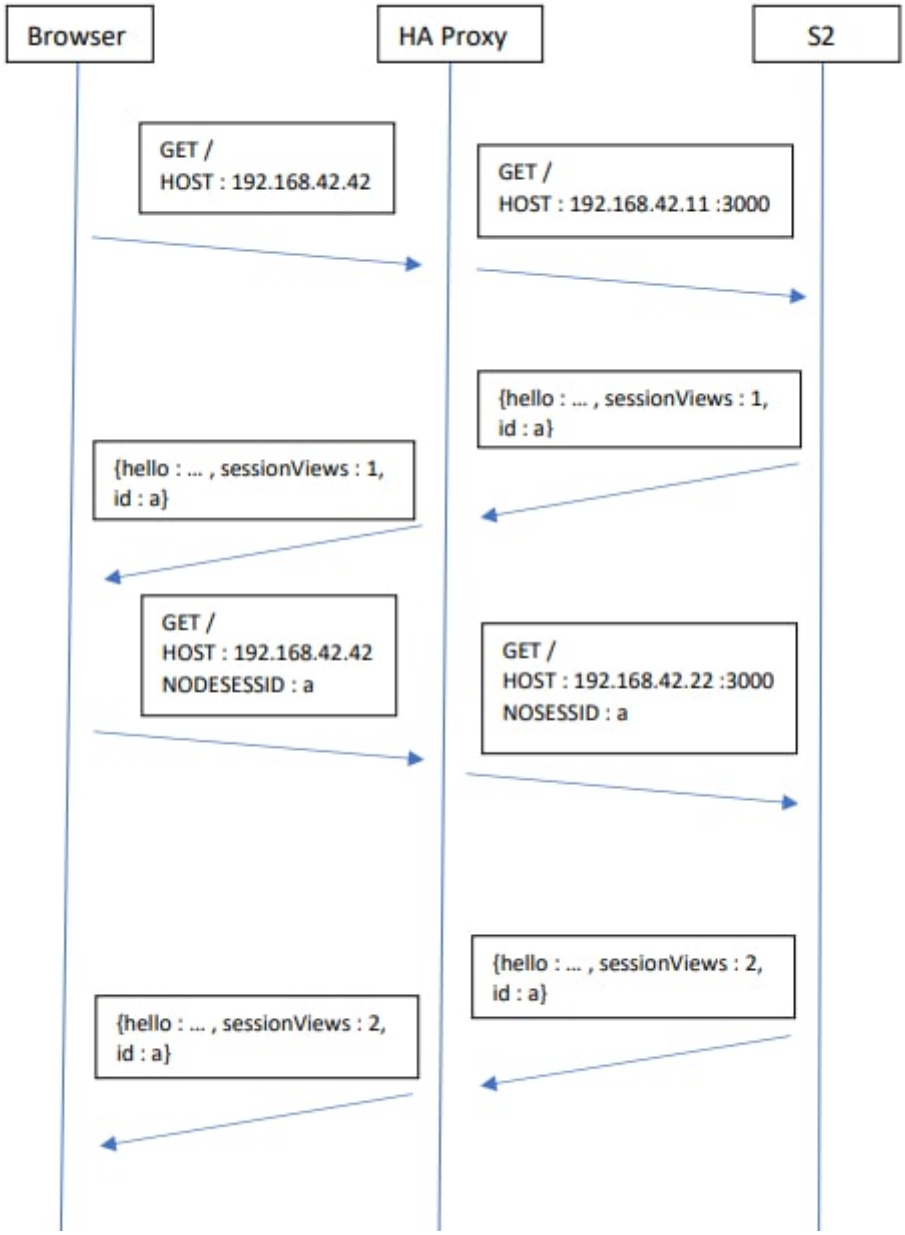
Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors Successes | Configure

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|
| GET / | 1000 | 11 | 1 | 74 | 17.01 | 0.00% | 51.5/sec | 19.89 | 395.0 |
| S2 reached | 1000 | 0 | 0 | 6 | 0.54 | 0.00% | 51.9/sec | 0.00 | .0 |
| TOTAL | 2000 | 5 | 0 | 74 | 13.28 | 0.00% | 103.1/sec | 19.88 | 197.5 |

Sur la capture ci-dessus, nous constatons que toutes les requêtes sont donc effectuées sur la webapp s2. Selon les tests manuels que nous avons pu faire, le compteur est bien incrémenté à chaque requête et le cookie utilisé reste systématiquement le même.

Le diagramme de séquence réalisé suite à cette tentative est le suivant :

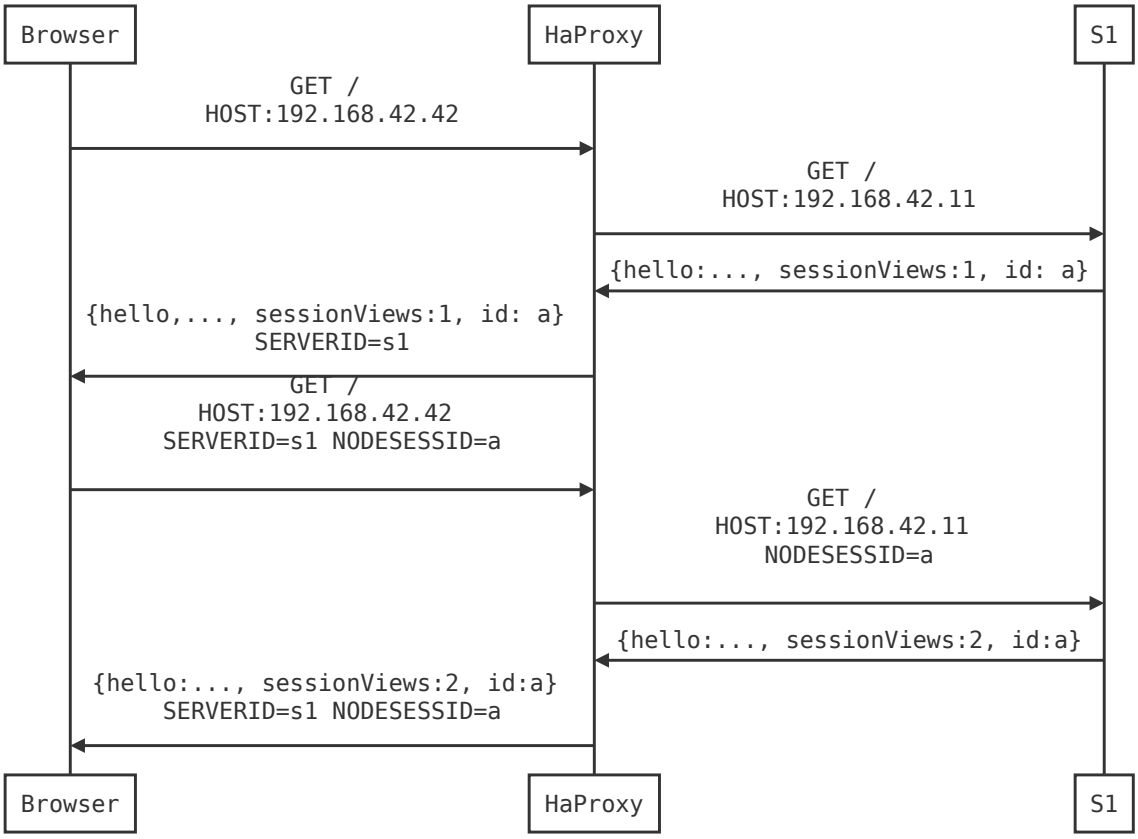



La webapp reconnait systématiquement le cookie qui est envoyé car c'est elle qui l'a attributé à la base. Il n'est jamais changé par une autre webapp et permet donc d'incrémenter le compteur `sessionViews`.

# Tâche 2 - La persistence des sessions (sticky sessions)

**2.1 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).**

Avec la methode SERVERID, le load balancer va séparer le cookie avec une partie SERVERID qui est collée au serveur correspondant. On peut ensuite récupérer le bon cookie pour récupérer la bonne session :




Dans la deuxième methode, on va insérer le SERVERID directement dans le cookie NODESESSID. Il est séparer par un ~. Comme pour la methode précédente, pour le serveur il n'y a auune différence. C'est uniquement le load balancer qui gère de mettre le SERVERID dans le NODESESSID.


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

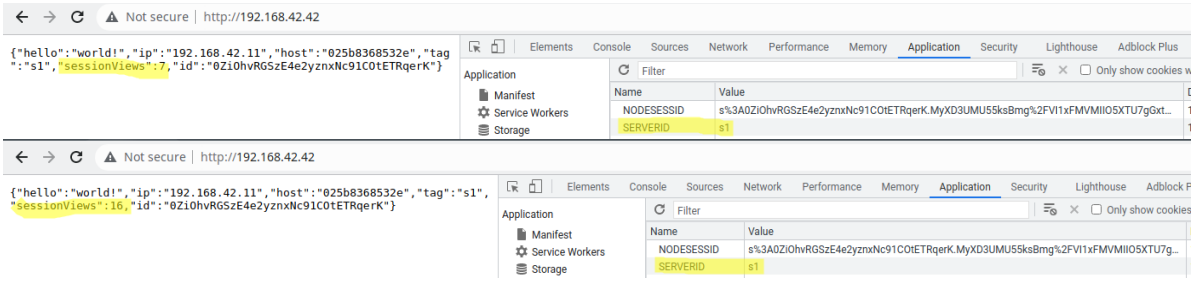
> Source : https://www.haproxy.com/fr/blog/load-balancing-affinity-persistence-sticky-sessions-what-you-need-to-know/

```
backend nodes

...

cookie SERVERID insert indirect nocache

# Define the list of nodes to be in the balancing mechanism
# http://cbonte.github.io/haproxy-dconv/2.2/configuration.html#4-server
server s1 ${WEBAPP_1_IP}:3000 check cookie s1
server s2 ${WEBAPP_2_IP}:3000 check cookie s2
```

- `cookie SERVERID insert indirect nocache`
  - Indique à HAProxy qu'il faut configurer un cookie SERVERID uniquement si l'utilisateur n'en a pas fourni un avec sa requête.
- `check cookie s1` et `check cookie s2`
  - Indique à HAProxy sur quel serveur rediriger l'utilisateur en fonction de son cookie.

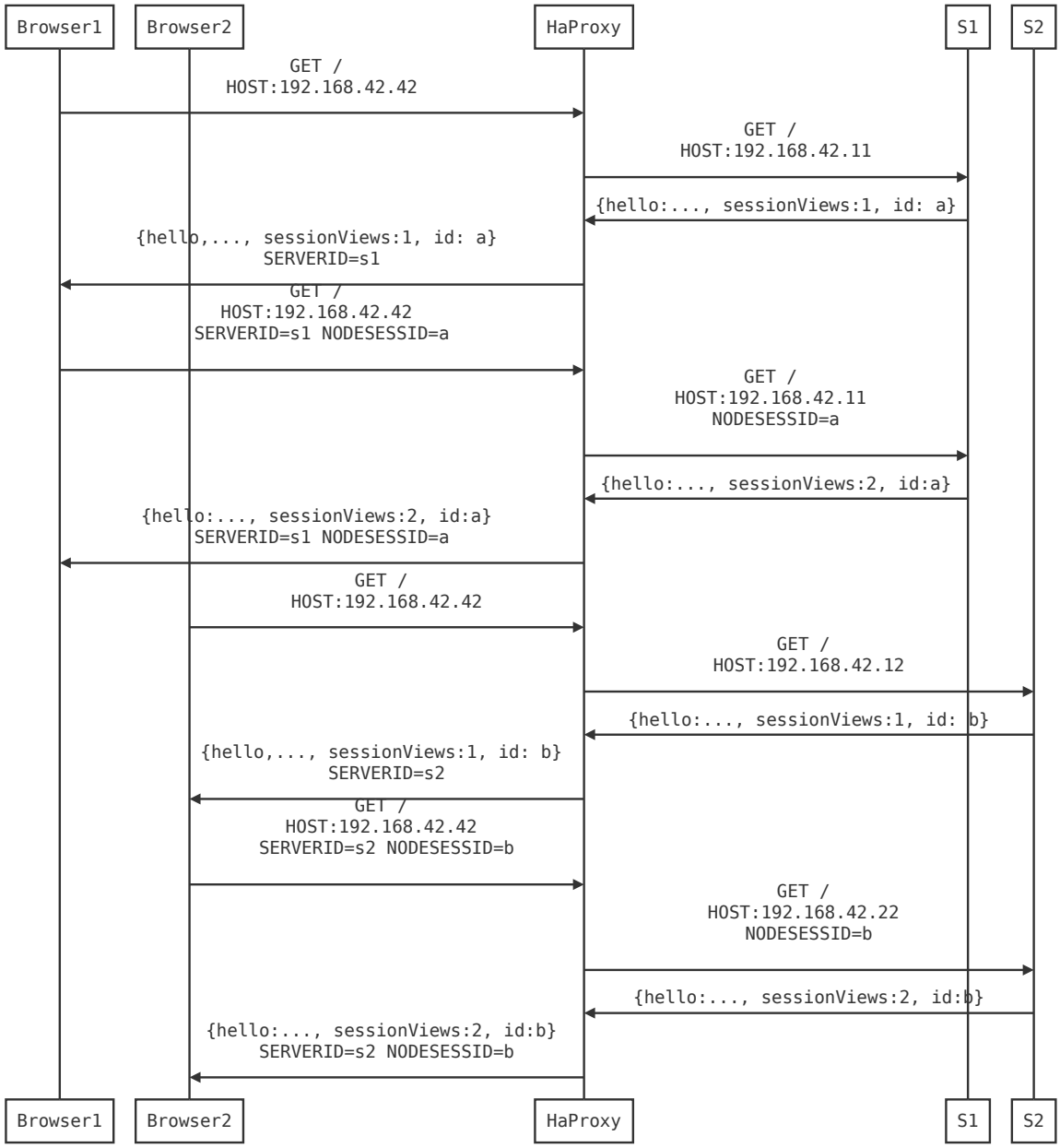
**2.3 Explain what is the behavior when you open and refresh the URL http://192.168.42.42 in your browser. Add screenshots to complement your explanations. We expect that you take a deeper a look at session management.**

L'utilisateur est toujours redirigé vers la même webapp, une même session est utilisée pour toutes les requêtes de l'utilisateur.




On peut voir ci-dessus que la session est bien identique pour les différentes requêtes d'un utilisateur. On voit aussi que le cookie SERVERID est bien présent afin d'indiquer au load balancer ou rediriger la requête (dans ce cas sur la webapp1).

**2.4 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.**




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

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|
| GET / | 1000 | 17 | 3 | 341 | 21.09 | 0.00% | 31.0/sec | 11.96 | 395.1 |
| S1 reached | 1000 | 0 | 0 | 21 | 1.06 | 0.00% | 31.7/sec | 0.00 | .0 |
| TOTAL | 2000 | 9 | 0 | 341 | 17.25 | 0.00% | 62.0/sec | 11.96 | 197.5 |

Nous observons que toutes les requêtes ont été redirigées vers la même webapp et donc que le sticky session est implémenté correctement. C'est différement du cas de la task1 ou l'on était redirigé de manière uniforme sur les deux webapps.

- **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.**

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

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|
| GET / | 2000 | 14 | 2 | 94 | 18.04 | 0.00% | 72.5/sec | 27.98 | 395.1 |
| S2 reached | 1000 | 0 | 0 | 19 | 0.88 | 0.00% | 36.8/sec | 0.00 | .0 |
| S1 reached | 1000 | 0 | 0 | 32 | 1.11 | 0.00% | 37.0/sec | 0.00 | .0 |
| TOTAL | 4000 | 7 | 0 | 94 | 14.54 | 0.00% | 145.0/sec | 27.98 | 197.5 |

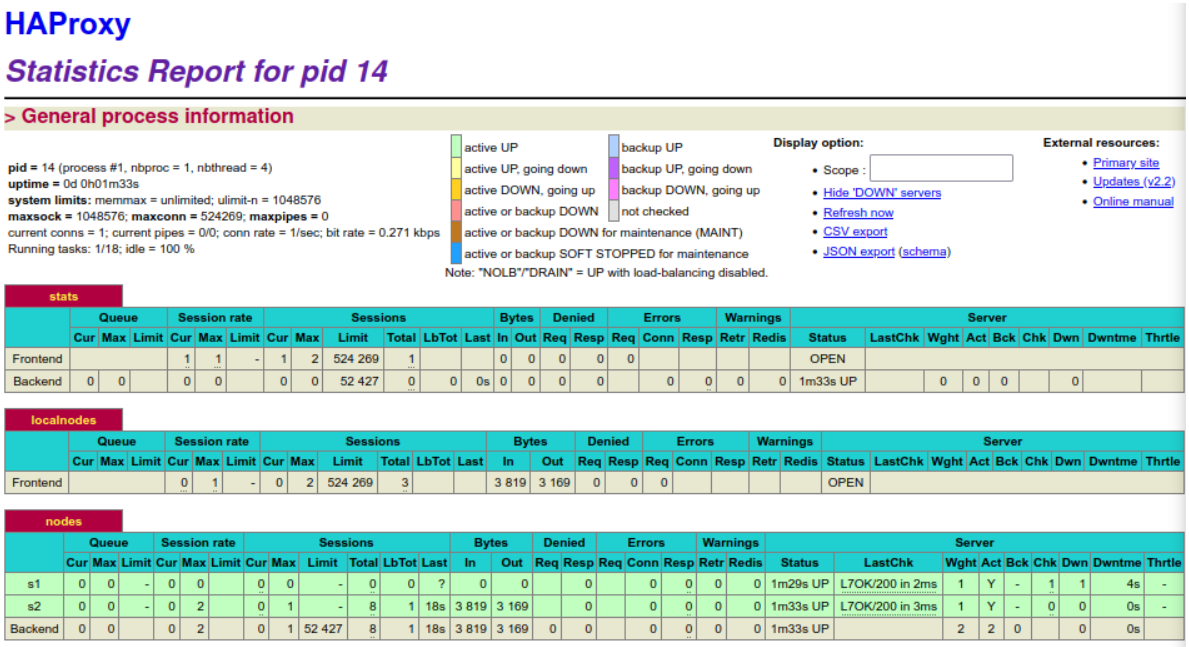
Nous avons configuré JMeter pour simuler l'envoi par deux utilisateurs de 1000 requêtes chacun. De plus les cookies ne sont pas reset à chaque nouvelle requête.

Nous observons le résultat attendu:

- Lors de sa première requête un des deux utililisateurs a été redirigé vers la webapp1. Lors des 999 requêtes suivantes il a été redirigé vers la même webapp.
- Pour l'autre utilisateur, le scénario est identique mais avec la webapp2.

# Tâche 3 - Le drainage des connexions (drain mode)

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



HAProxy

*Statistics Report for pid 14*

> General process information

**pid** = 14 (process #1, nbproc = 1, nbthread = 4)
**uptime** = 0d 0h01m33s
**system limits:** memmax = unlimited; ulimit-n = 1048576
**maxsock** = 1048576; **maxconn** = 524269; **maxpipes** = 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec; bit rate = 0.271 kbps
Running tasks: 1/18; 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
- JSON export (schema)

External resources:
- Primary site
- Updates (v2.2)
- 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 | 2 | 524 269 | 1 | | | 0 | 0 | 0 | 0 | 0 | | | | | OPEN | | | | | | | | |
| Backend | 0 | 0 | | 0 | 0 | | 0 | 0 | 52 427 | 0 | 0 | 0s | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 1m33s 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 | 2 | 524 269 | 3 | | | 3 819 | 3 169 | 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 | 1m29s UP | L7OK/200 in 2ms | 1 | Y | - | 1 | 1 | 4s | - |
| s2 | 0 | 0 | - | 0 | 2 | | 0 | 1 | - | 8 | 1 | 18s | 3 819 | 3 169 | | 0 | | 0 | 0 | 0 | 0 | 1m33s UP | L7OK/200 in 3ms | 1 | Y | - | 0 | 0 | 0s | - |
| Backend | 0 | 0 | | 0 | 2 | | 0 | 1 | 52 427 | 8 | 1 | 18s | 3 819 | 3 169 | 0 | 0 | | 0 | 0 | 0 | 0 | 1m33s UP | | 2 | 2 | 0 | | 0 | 0s | |

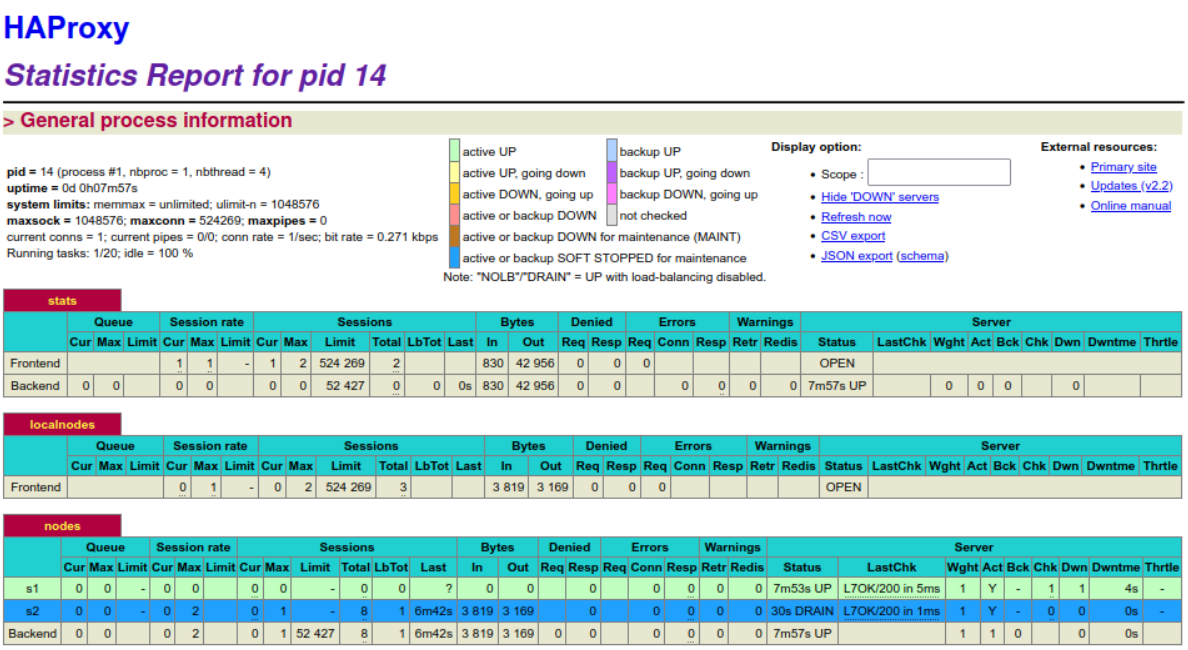
Sur la capture ci-dessus, nous constatons que la node qui a été atteinte est la node `s2`.

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

Après avoir entré les commandes suivantes, nous avons repris une capture d'écran de la page de statistiques HAProxy :

```
$ socat - tcp:192.168.42.42:9999
prompt
> set timeout cli 1d
> set server nodes/s2 state drain
```

Nous obtenons une page de statistiques actualisée :



HAProxy

Statistics Report for pid 14

> General process information

pid = 14 (process #1, nbproc = 1, nbthread = 4)
uptime = 0d 0h07m57s
system limits: memmax = unlimited; ulimit-n = 1048576
maxsock = 1048576; maxconn = 524269; maxpipes = 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec; bit rate = 0.271 kbps
Running tasks: 1/20; 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
JSON export (schema)

External resources:
Primary site
Updates (v2.2)
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 | 2 | 524 269 | 2 | | | 830 | 42 956 | 0 | 0 | 0 | | | | | OPEN | | | | | | | | |
| Backend | 0 | 0 | | 0 | 0 | | 0 | 0 | 52 427 | 0 | 0 | 0s | 830 | 42 956 | 0 | 0 | | 0 | 0 | 0 | 0 | 7m57s 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 | 2 | 524 269 | 3 | | | 3 819 | 3 169 | 0 | 0 | 0 | | | | | OPEN | | | | | | | | |

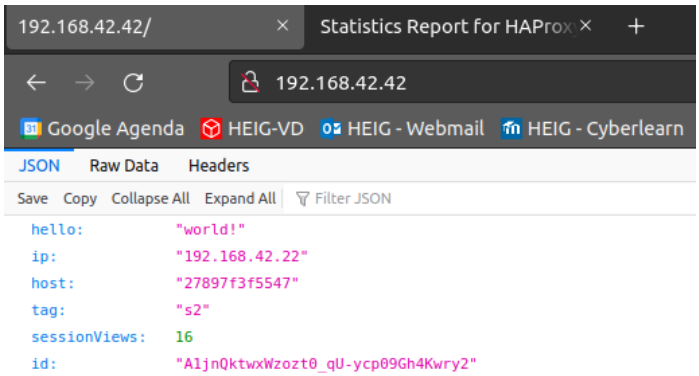
nodes

| | 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 | 7m53s UP | L7OK/200 in 5ms | 1 | Y | - | 1 | 1 | 4s | - |
| s2 | 0 | 0 | - | 0 | 2 | | 0 | 1 | - | 8 | 1 | 6m42s | 3 819 | 3 169 | | 0 | | 0 | 0 | 0 | 0 | 30s DRAIN | L7OK/200 in 1ms | 1 | Y | - | 0 | 0 | 0s | - |
| Backend | 0 | 0 | | 0 | 2 | | 0 | 1 | 52 427 | 8 | 1 | 6m42s | 3 819 | 3 169 | 0 | 0 | | 0 | 0 | 0 | 0 | 7m57s UP | | 1 | 1 | 0 | | 0 | 0s | |

Nous constatons que la couleur de la ligne concernant la node `s2` a changé. La colonne `STATUS` indique également que le mode `DRAIN` est activé depuis 30 secondes.

### 3.3 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?

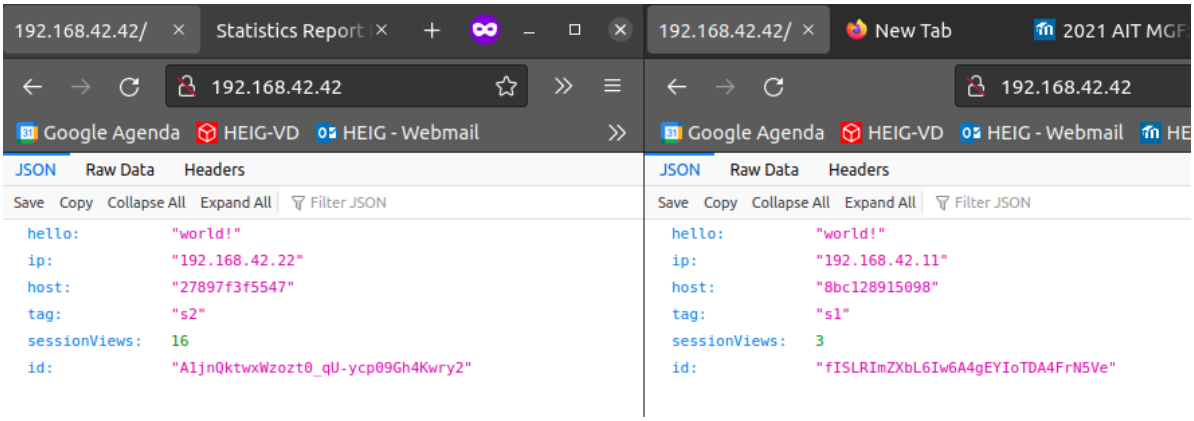
En rafraichissant la fenêtre du navigateur ouverte et connectée sur la node `s2`, nous constatons que le compteur continue d'être incrémenté :




Le mode `DRAIN` permet de retirer le serveur du load balancing, mais les connexions qui sont déjà établies restent en place.

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

Lorsque nous ouvrons un nouveau navigateur pour nous rendre sur l'adresse mentionnée, nous tombons sur la node `s1` :




Sur la capture d'écran ci-dessus, nous voyons la fenêtre de gauche connectée sur la node `s2` (la première fenêtre qui a été ouverte avant le mode `DRAIN`) et la fenêtre de droite qui est le nouveau navigateur connecté sur la node `s1`. Lorsque nous rafraichissons cette fenêtre, nous restons toujours sur la même node.

**3.5 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?**

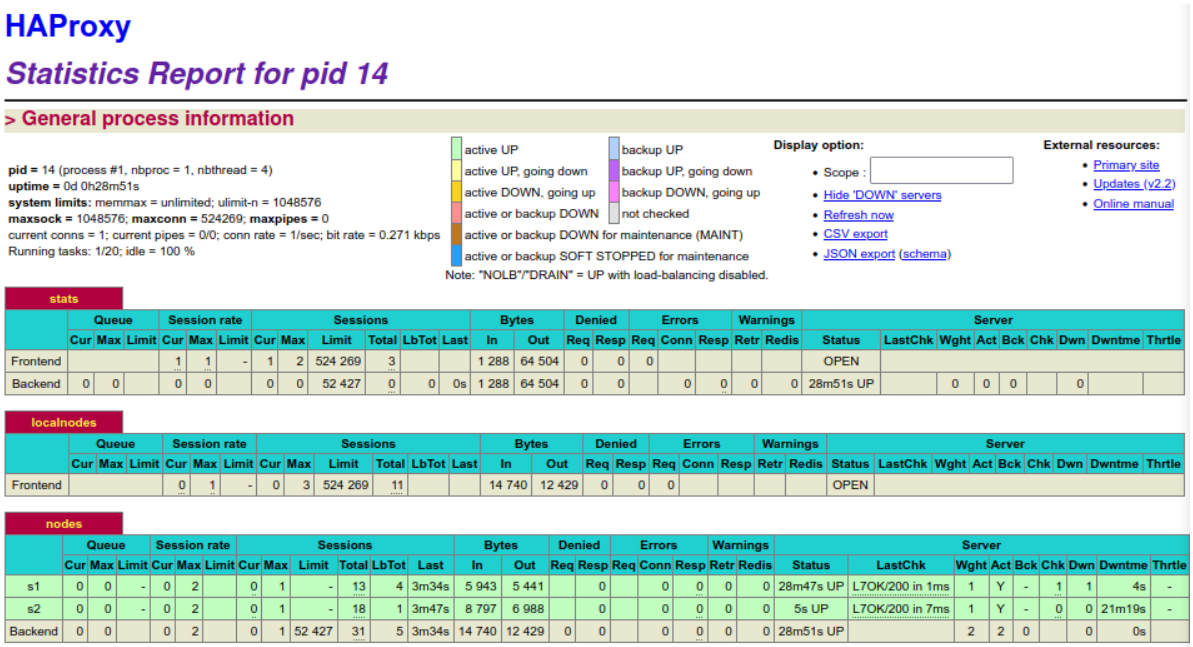
Non, nous n'atteignons jamais la node `s2` qui est la node en mode `DRAIN` malgré avoir rafraichit la page et supprimé les cookies plusieurs fois. Nous restons sur la node `s1`. C'est exactement ce qui est supposé se passer, car c'est le rôle du mode `DRAIN` de ne pas accepter de nouvelles connexions mais de garder actives celles qui sont déjà établies.

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

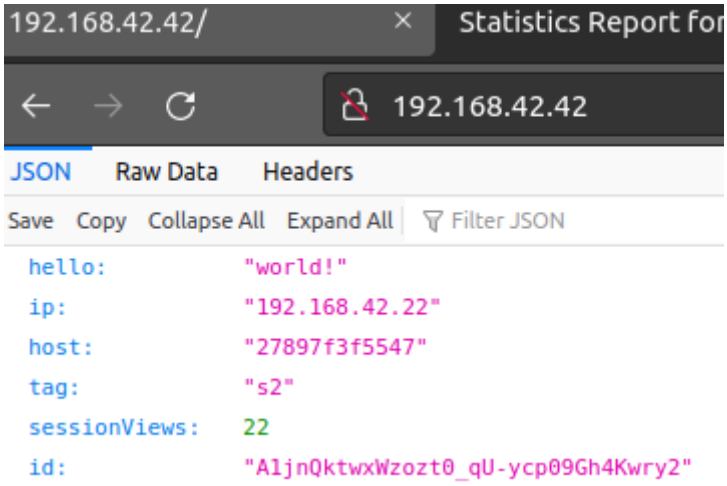
Nous avons entré la commande suivante afin de remettre la node `s2` en état `READY` :

```
> set server nodes/s2 state ready
```

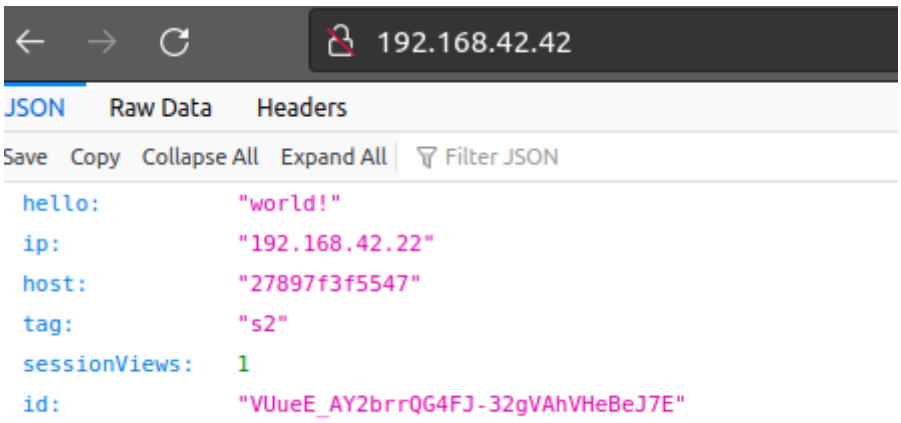
Nous rafraichissons la page de statistiques de HAProxy pour que la modification soit prise en compte et nous obtenons la capture d'écran suivante :




Nous constatons que l'état a bien été modifié car la colonne `STATUS` indique de nouveau un état `UP`. En rafraichissant la page de navigateur déjà connectée sur la node `s2`, nous voyons qu'elle reste sur la même node :




En ouvrant une nouvelle fenêtre de navigation, nous allons sur l'adresse `http://192.168.42.42/` et nous constatons que nous nous retrouvons sur la node `s2` mais que le compteur est de nouveau à 1. Nous en déduisons donc qu'une nouvelle session a été ouverte pour ce nouveau navigateur :




Le fonctionnement normal est donc retrouvé, il est de nouveau possible de se connecter sur la node `s2` avec une nouvelle session.

**3.7 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.**

Nous modifions le mode avec la commande :

```
> set server nodes/s2 state maint
```

La page de statistiques de HAProxy a maintenant un affichage différent :

**HAProxy**

***Statistics Report for pid 14***

**> General process information**

**pid =** 14 (process #1, nbproc = 1, nbthread = 4)
**uptime =** 0d 0h46m12s
**system limits:** memmax = unlimited; ulimit-n = 1048576
**maxsock =** 1048576; **maxconn =** 524269; **maxpipes =** 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec; bit rate = 0.271 kbps
Running tasks: 1/20; 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
- JSON export (schema)

**External resources:**
- Primary site
- Updates (v2.2)
- 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 | 2 | 524 269 | 4 | | | 1 746 | 86 190 | 0 | 0 | 0 | | | | | OPEN | | | | | | | | |
| Backend | 0 | 0 | | 0 | 0 | | 0 | 0 | 52 427 | 0 | 0 | 0s | 1 746 | 86 190 | 0 | 0 | | 0 | 0 | 0 | 0 | 46m12s UP | | 0 | 0 | 0 | | 0 | | |

**localnodes**

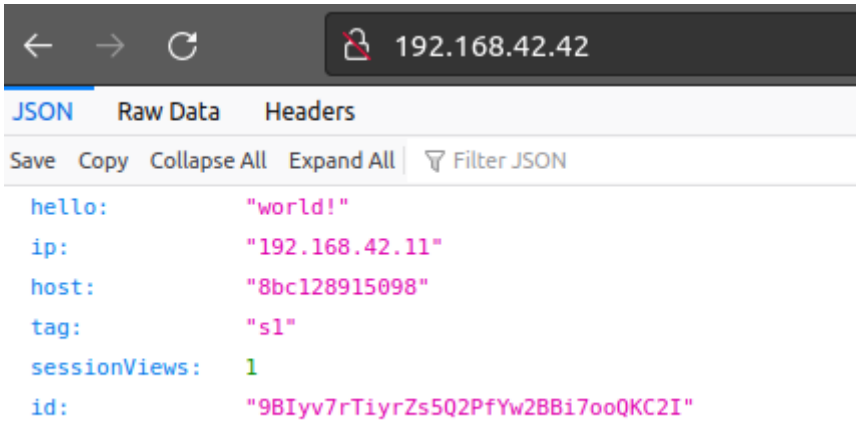
| | Queue | | | Session rate | | | Sessions | | | | | | Bytes | | Denied | | Errors | | | Warnings | | Server | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Cur | Max | Limit | Cur | Max | Limit | Cur | 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 | 3 | 524 269 | 23 | | | 28 749 | 23 734 | 0 | 0 | 0 | | | | | OPEN | | | | | | | | |

**nodes**

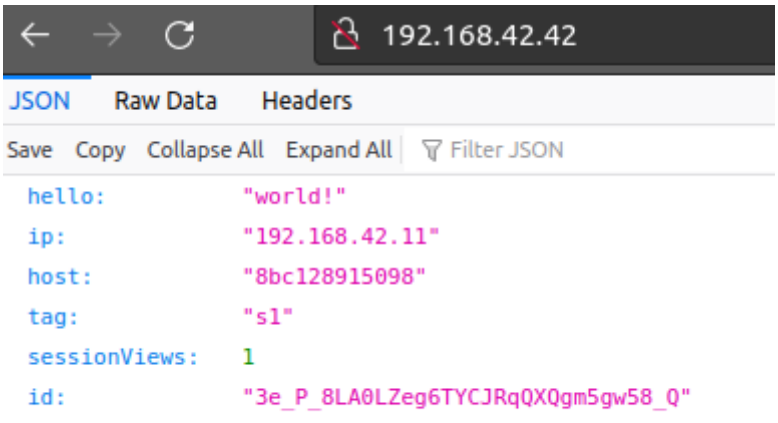
| | Queue | | | Session rate | | | Sessions | | | | | | Bytes | | Denied | | Errors | | | Warnings | | Server | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Cur | Max | Limit | Cur | Max | Limit | Cur | 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 | - | 27 | 5 | 3m36s | 12 678 | 10 898 | | 0 | | 0 | 0 | 0 | 0 | 46m8s UP | L7OK/200 in 4ms | 1 | Y | - | 1 | 1 | 4s | - |
| s2 | 0 | 0 | - | 0 | 2 | | 0 | 1 | - | 33 | 2 | 3m30s | 16 071 | 12 836 | | 0 | | 0 | 0 | 0 | 0 | 3s MAINT | | 1 | Y | - | 0 | 1 | 21m22s | - |
| Backend | 0 | 0 | | 0 | 2 | | 0 | 1 | 52 427 | 60 | 7 | 3m30s | 28 749 | 23 734 | 0 | 0 | | 0 | 0 | 0 | 0 | 46m12s UP | | 1 | 1 | 0 | | 0 | 0s | |

Nous constatons que la couleur de la ligne de la node `s2` a changé, et la colonne `STATUS` indique que cette node est en mode `MAINT`.

Lorsque nous rafraichissons la page déjà connectée sur la node `s2`, nous atteignons la node `s1` avec un compteur avec une valeur de 1, donc une nouvelle session :




Puis, nous ouvrons une nouvelle page et nous atteignons également la node `s1`, car la node `s2` est en état de maintenance (`MAINT`) et elle n'accepte plus aucune connexion (celles déjà établies sont arrêtées) :




# Tâche 4 - Le mode dégradé avec Round Robin

**4.1 Make sure a delay of 0 milliseconds is set on `s1`. Do a run to have a baseline to compare with in the next experiments.**

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|
| GET / | 2000 | 10 | 1 | 55 | 17.04 | 0.00% | 124.3/sec | 47.95 | 395.1 |
| S1 reached | 1000 | 0 | 0 | 1 | 0.29 | 0.00% | 62.8/sec | 0.00 | .0 |
| S2 reached | 1000 | 0 | 0 | 20 | 0.76 | 0.00% | 64.2/sec | 0.00 | .0 |
| TOTAL | 4000 | 5 | 0 | 55 | 13.20 | 0.00% | 248.5/sec | 47.94 | 197.5 |

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

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|
| GET / | 2000 | 155 | 1 | 555 | 156.60 | 0.00% | 6.6/sec | 2.53 | 395.1 |
| S1 reached | 1000 | 0 | 0 | 2 | 0.31 | 0.00% | 3.3/sec | 0.00 | .0 |
| S2 reached | 1000 | 0 | 0 | 5 | 0.35 | 0.00% | 69.9/sec | 0.00 | .0 |
| TOTAL | 4000 | 77 | 0 | 555 | 135.13 | 0.00% | 13.1/sec | 2.53 | 197.5 |

Nous pouvons observer que le rendement de s1 a grandement diminué et est passé à 3.3 requêtes par secondes.

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

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|
| GET / | 2000 | 8 | 1 | 53 | 15.76 | 0.00% | 155.3/sec | 59.93 | 395.1 |
| S2 reached | 2000 | 0 | 0 | 11 | 0.43 | 0.00% | 155.4/sec | 0.00 | .0 |
| TOTAL | 4000 | 4 | 0 | 53 | 11.94 | 0.00% | 310.6/sec | 59.92 | 197.5 |

Nous observons que s1 ne reçoit plus aucune requête et que toutes les requêtes sont redirigées vers s2. (Explications dans le point suivant)

**4.4 In the two previous steps, are there any errors? Why?**

On ne voit pas d'erreur directement dans JMeter, mais si on regarde dans les outputs de docker-compose on peut voir un message d'erreur nous indiquant qu'il considère s1 comme étant down. Cela s'explique par le fait que la durée de vérification (2001ms) est plus courte que la durée du délai que nous avons configuré. Par ce fait toutes les requêtes sont redirigées sur s2.

Voici l'erreur retournée : `[WARNING] 340/142844 (10) : Server nodes/s1 is DOWN, reason: Layer7 timeout, check duration: 2001ms. 1 active and 0 backup servers left. 1 sessions active, 0 requeued, 0 remaining in queue.`

**4.5 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 a 250ms delay.**

```
backend nodes
[...]
server s1 ${WEBAPP_1_IP}:3000 check cookie s1 weight 2
server s2 ${WEBAPP_2_IP}:3000 check cookie s2 weight 1
```

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|
| GET / | 2000 | 154 | 1 | 558 | 156.35 | 0.00% | 6.6/sec | 2.55 | 395.1 |
| S1 reached | 1000 | 0 | 0 | 8 | 0.52 | 0.00% | 3.3/sec | 0.00 | .0 |
| S2 reached | 1000 | 0 | 0 | 5 | 0.32 | 0.00% | 78.7/sec | 0.00 | .0 |
| TOTAL | 4000 | 77 | 0 | 558 | 134.78 | 0.00% | 13.2/sec | 2.55 | 197.5 |

Comme il n'y a que deux threads et qu'il y a une schedulin policy Round robin, un utilisateur sera envoyé sur chaque serveur et dans ce cas là nous n'observons pas l'impact du poids.

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

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|
| GET / | 2000 | 332 | 1 | 795 | 262.54 | 0.00% | 5.2/sec | 2.92 | 575.7 |
| S1 reached | 1334 | 0 | 0 | 11 | 0.45 | 0.00% | 3.5/sec | 0.00 | .0 |
| S2 reached | 666 | 0 | 0 | 12 | 0.70 | 0.00% | 1.7/sec | 0.00 | .0 |
| TOTAL | 4000 | 166 | 0 | 795 | 249.09 | 0.00% | 10.4/sec | 2.92 | 287.8 |

Comme les cookies sont effacés entre chaque requête, chaque requête est considérée comme venant d'un nouvel utilisateur. Dans ce cas là, nous pouvons observer l'impact du poids des serveurs. Nous observons donc que dans 2/3 des cas les requêtes sont redirigées vers s1 et dans le tier restant vers s2.

# Tâche 5 - Les stratégies de load balancing

> Source utilisée : http://cbonte.github.io/haproxy-dconv/2.2/configuration.html
>
> La documentation fournie dans la donnée était dépréciée, d'où notre choix d'utiliser une autre source.

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

```
   leastconn   The server with the lowest number of connections receives the
               connection. Round-robin is performed within groups of servers
               of the same load to ensure that all servers will be used. This
               algorithm is dynamic, which means that server weights may be
               adjusted on the fly for slow starts for instance.

   random      A random number will be used as the key for the consistent
               hashing function. This means that the servers' weights are
               respected, dynamic weight changes immediately take effect, as
               well as new server additions. This algorithm is also known as
               the Power of Two Random Choices and is described here :
               http://www.eecs.harvard.edu/~michaelm/postscripts/handbook2001.pdf
```

### 5.2 Provide evidence that you have played with the two strategies (configuration done, screenshots)

#### 5.2.1 `leastconn`

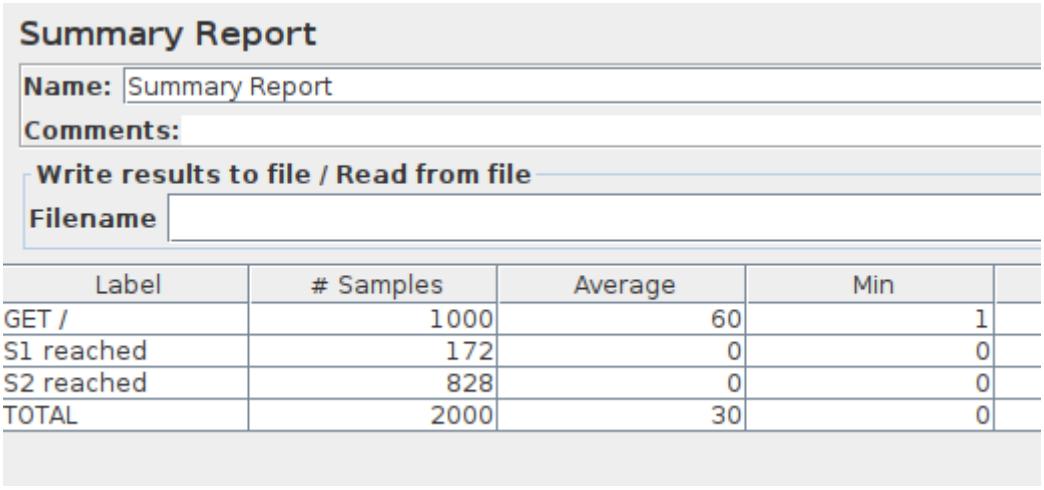
Nous avons modifié la configuration comme suit :

```
backend nodes
[...]
    balance leastconn
[...]
    server s1 ${WEBAPP_1_IP}:3000 cookie S1 check
    server s2 ${WEBAPP_2_IP}:3000 cookie S2 check
```

Nous avons essayé en ajoutant un delay de 250 ms sur la node `s1` grâce à la commande :

```
$ curl -H "Content-Type: application/json" -X POST -d '{"delay": 250}'
http://192.168.42.11:3000/delay
```

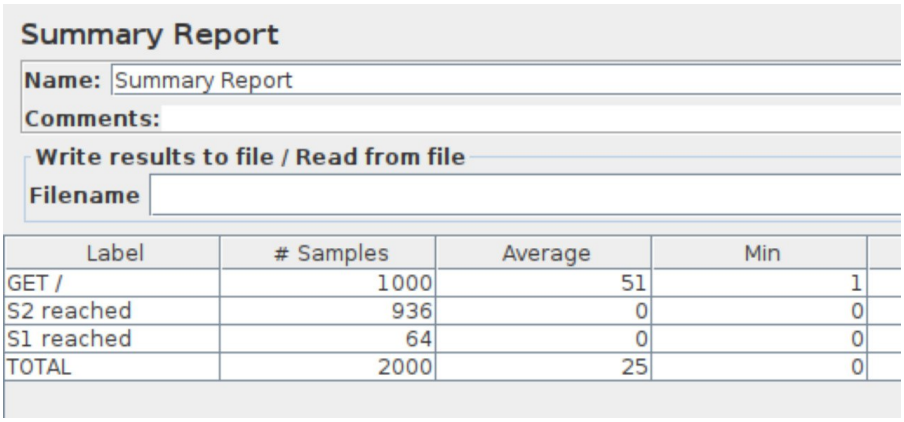
Nous avons lancé le script `tester.jmx` sur JMeter avec deux threads et 500 requêtes par thread, pour obtenir le résultat suivant :




| Label | # Samples | Average | Min |
|---|---|---|---|
| GET / | 1000 | 60 | 1 |
| S1 reached | 172 | 0 | 0 |
| S2 reached | 828 | 0 | 0 |
| TOTAL | 2000 | 30 | 0 |

Nous constatons que la node `s1` a reçu moins de requêtes (172), car elle mettait plus de temps à répondre. Le load balancer répartit alors les requêtes sur la node `s2` qui est disponible.

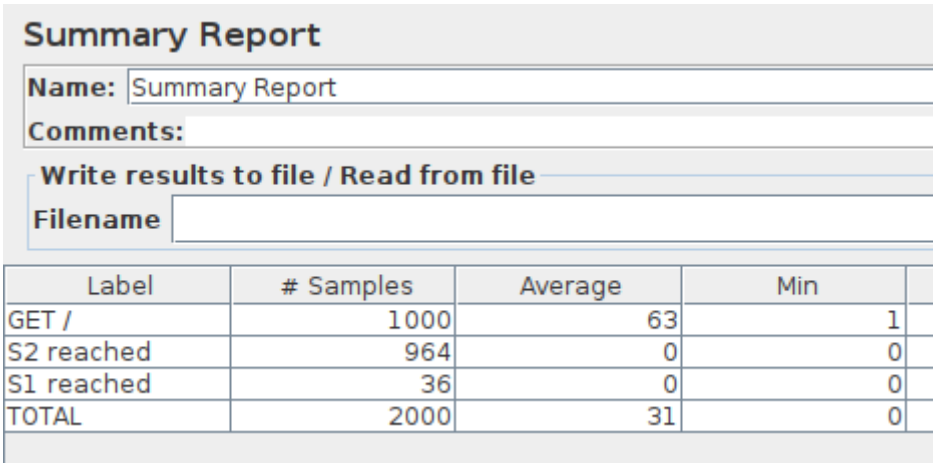
Avec un delay de 500 ms, nous avons obtenu le résultat suivant :




| Label | # Samples | Average | Min |
|---|---|---|---|
| GET / | 1000 | 51 | 1 |
| S2 reached | 936 | 0 | 0 |
| S1 reached | 64 | 0 | 0 |
| TOTAL | 2000 | 25 | 0 |

Seules 64 requêtes ont été dirigées vers la node `s1`.

Nous avons ensuite essayé d'ajouter un delay plus grand, c'est-à-dire 1000 ms :




| Label | # Samples | Average | Min |
|---|---|---|---|
| GET / | 1000 | 63 | 1 |
| S2 reached | 964 | 0 | 0 |
| S1 reached | 36 | 0 | 0 |
| TOTAL | 2000 | 31 | 0 |

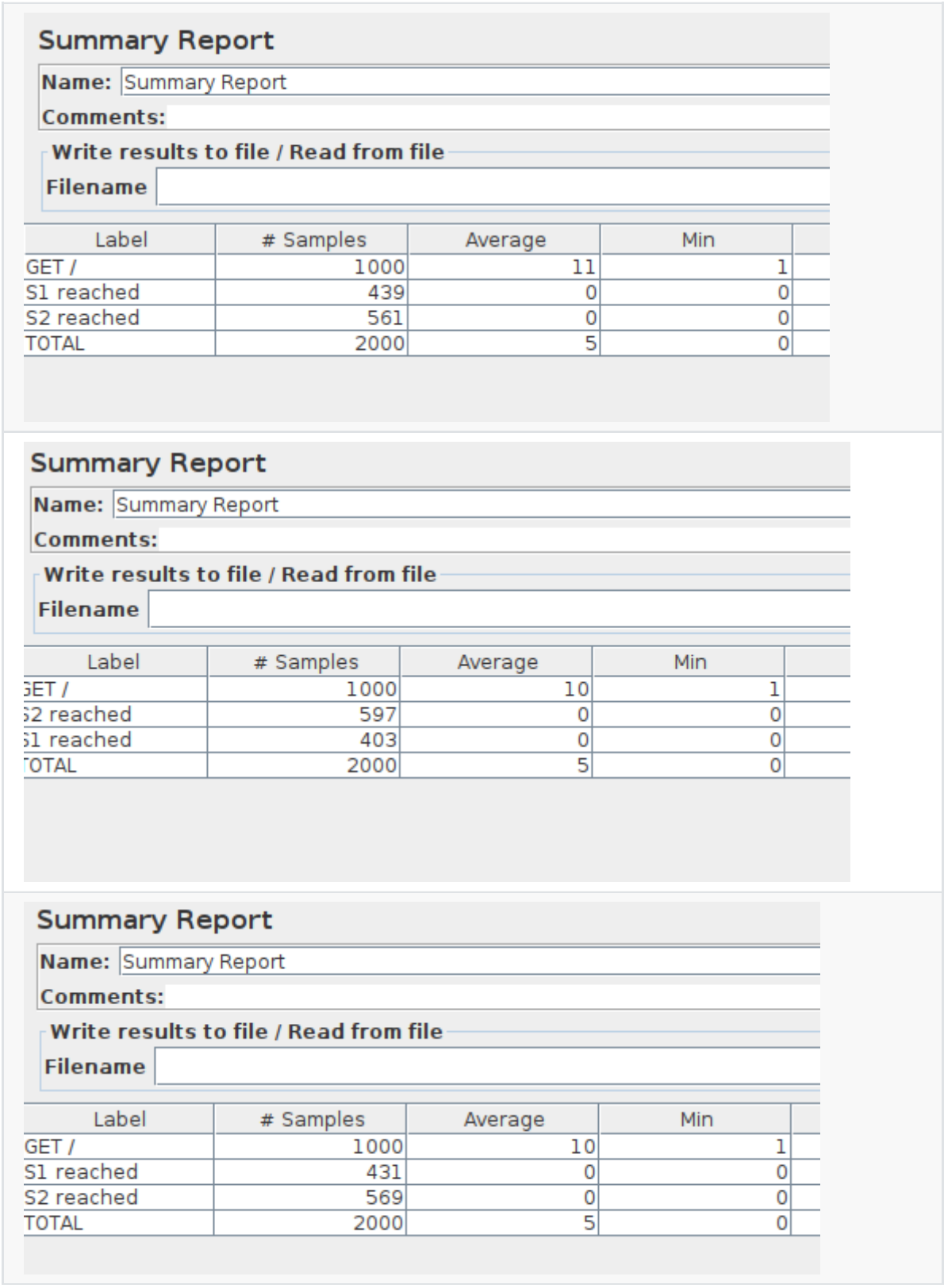
Lorsque nous augmentons le delay en le doublant, nous constatons que la node `s1` reçoit quasiment deux fois moins de requêtes, car elle est deux fois plus lente à répondre. Avec un delay de 1000 ms, elle reçoit 36 requêtes, soit deux fois moins qu'avec un delay de 500 ms (64 reçues).

#### 5.2.2 `random`

Nous avons modifié la configuration comme suit :

```
backend nodes
[...]
    balance random
[...]
    server s1 ${WEBAPP_1_IP}:3000 check
    server s2 ${WEBAPP_2_IP}:3000 check
```

Nous avons fait plusieurs essais avec JMeter pour tester si les nodes sont réellement atteintes de manière aléatoire, et nous concluons que c'est le cas, car chacun de nos essais donne un résultat différent :



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename

| Label | # Samples | Average | Min |
|---|---|---|---|
| GET / | 1000 | 11 | 1 |
| S1 reached | 439 | 0 | 0 |
| S2 reached | 561 | 0 | 0 |
| TOTAL | 2000 | 5 | 0 |

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename

| Label | # Samples | Average | Min |
|---|---|---|---|
| GET / | 1000 | 10 | 1 |
| S2 reached | 597 | 0 | 0 |
| S1 reached | 403 | 0 | 0 |
| TOTAL | 2000 | 5 | 0 |

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename

| Label | # Samples | Average | Min |
|---|---|---|---|
| GET / | 1000 | 10 | 1 |
| S1 reached | 431 | 0 | 0 |
| S2 reached | 569 | 0 | 0 |
| TOTAL | 2000 | 5 | 0 |

**5.3 Compare the two strategies and conclude which is the best for this lab (not necessary the best at all).**

Le scheduling leastconn est plus adapté à des situations ou la durée de la session est longue (LDAP, SQL, TSE,...) mais moins à des cas comme http.
Random quand à lui est plus adapté dans les cas ou le  nombre de serveurs est grand et dans un cas ou des serveurs sont fréquement ajoutés ou retirés.
Dans notre cas leastconn serait plus adapté, il pourrait permettre de répartir la charge sur les serveurs plus uniformément dans le cas ou certaines requêtes arrivent avec du délai.

# Conclusion

Nous avons appris comment configurer un load-balancer et comment celui-ci réagit à différentes situations (un serveur est down, certaines requêtes arrivent avec du délai, ...).

Nous avons aussi appris à utiliser le load-balancer avec différentes scheduling policy.

Ce fut long mais ce fut fun.

Bonnes fêtes de fin d'année