Labo 3 - Load Balancing

> Auteurs: Gwendoline Dössegger, Noémie Plancherel, Gaby Roch
>
> Date: 09.12.2021

Introduction

Ce laboratoire va nous permettre de prendre en main et de mieux comprendre le fonctionnement de HAProxy. Docker sera utilisé pour avoir plusieurs serveurs sur la même machine, il y aura 1 container pour le load balancer (HAProxy) et 2 containers pour les nœuds applicatifs (nodejs). Nous pourrons comprendre comment les requêtes sont transférées aux nœuds serveurs depuis HAProxy et comment le choix de quel nœud utilisé est fait.

Pour faire de nombreux tests sur notre infrastructure, nous utiliserons JMeter. Cette application permet de lancer de multiples requêtes sur un serveur et d'analyser leur réponse. Nous pouvons également lui dire si nous voulons que chaque requête soit indépendante les unes des autres (suppression des cookies entre chaque requête) ou liées comme dans un usage normal.

Tâches

1. Installation des outils

Nous avons déjà installé `docker` et `docker-compose` avant le laboratoire. Nous devons uniquement installer `JMeter`.

Nous l'installons directement avec le paquet Debian:

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

Nous allons à présent créer les 3 containers à l'aide de la commande `docker-compose up` qui va se référer au `Dockerfile`

```
docker-compose build
# Nous avons dû désactiver un service car le port 80 était déjà occupé
sudo systemctl stop apache2.service
docker-compose up
```

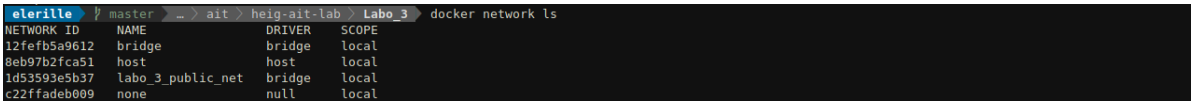
Nous pouvons à présent vérifier que les 3 containers tournent avec la commande:

```
docker ps
```

```
elerille  master  …  ait  heig-ait-lab  Labo_3  SIGINT  docker ps
CONTAINER ID   IMAGE            COMMAND                  CREATED              STATUS              PORTS
       NAMES
fb532a5ec302   labo_3_webapp1   "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:4000->3000/tcp
       s1
91087b6102a3   labo_3_webapp2   "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:4001->3000/tcp
       s2
4de2a560a5c7   labo_3_haproxy   "docker-entrypoint.s…"   About a minute ago   Up 28 seconds       0.0.0.0:80->80/tcp, 0.0.0.0:1936->1936/tcp, 0.0.0.0:9999->999
9/tcp   ha
```

Nous constatons qu'il y a un réseau virtuel qui a été créé pour le labo 3 avec la commande:

```
docker network ls
```



Une fois toute la mise en place faite, nous pouvons nous rendre sur le load balancer via l'adresse `http://192.168.42.42`.

En contrôlant, la requête de la page, nous constatons que le `NODESESSID` est bien envoyé à notre navigateur.

1.1

> 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.

Résultat lorsqu'on se rend sur `http://192.168.42.42` pour la première fois:

```
hello:        "world!"
ip:           "192.168.42.11"
host:         "fb532a5ec302"
tag:          "s1"
sessionViews: 1
id:           "1kBMaGmjCa9hmLVwWxVCENATtzGXURGv"
```

Lorsque l'on rafraîchit la page:

```
hello:        "world!"
ip:           "192.168.42.22"
host:         "91087b6102a3"
tag:          "s2"
sessionViews: 1
id:           "jrqrrcnkAygbqqMLHz7faArm9yq5jCOr"
```

On constate qu'il y a une alternance entre les deux serveurs à chaque fois qu'on rafraîchit la page. Le **round-robin** est utilisé puisque c'est une fois l'un et une fois l'autre.

Nous remarquons que le `sessionViews` est à 1, peu importe le nombre d'accès au serveur qu'on fait. Cela est dû au fait qu'aucune session n'est créée.

Nous voyons également que les cookies ne sont pas pris en compte par le load balancing à cette étape. Ceci ne permet donc pas de garder un état entre les connexions dû au fait que le protocole HTTP est un protocole sans état.

1.2

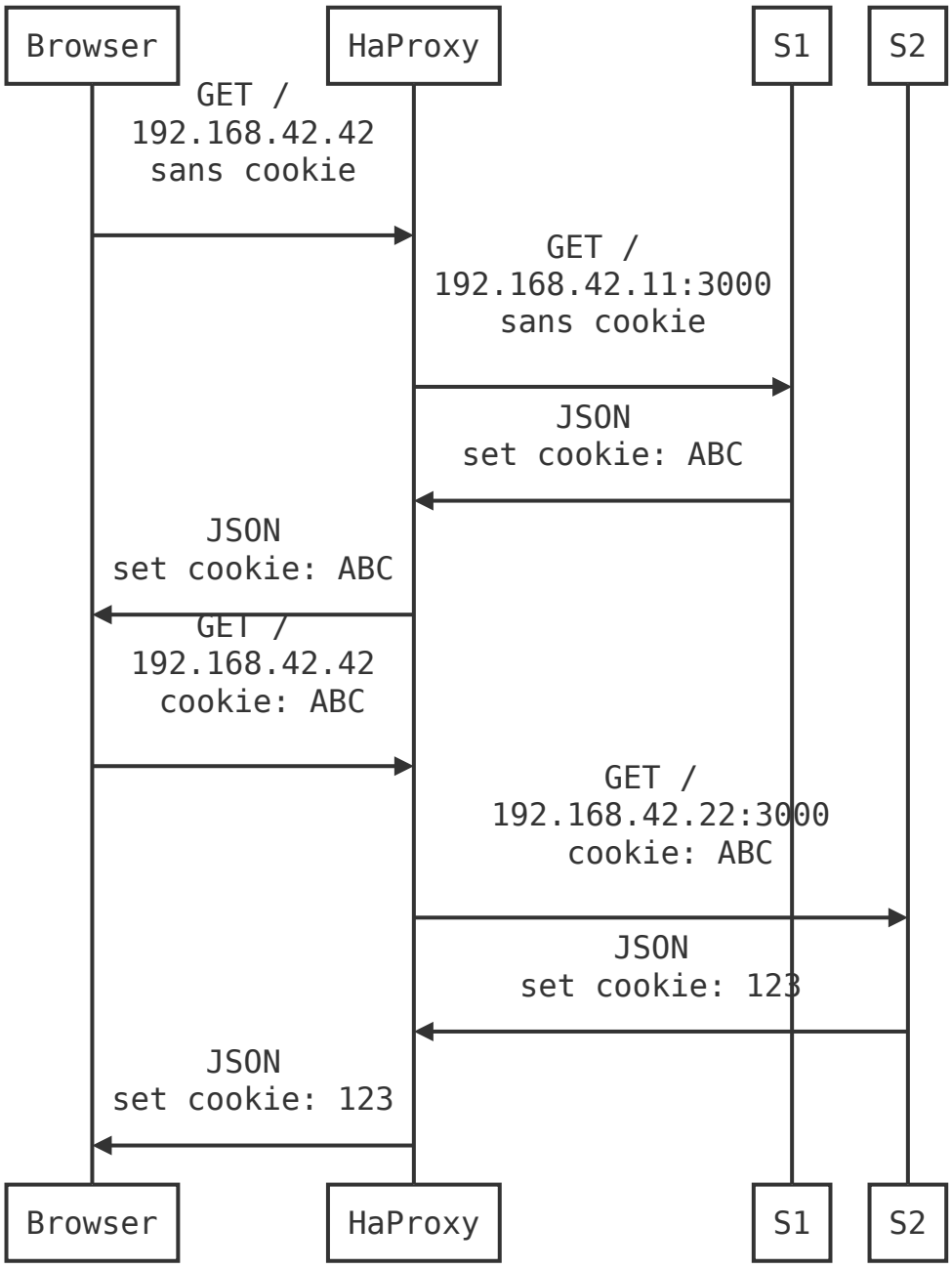
> Explain what should be the correct behavior of the load balancer for session management.

Si on prend en compte les sessions, quelque soit le nombre de requêtes que l'on fait sur le HAProxy, on communiquera toujours avec le même serveur. Ceci permettra que les informations de sessions soient cohérentes et pas réinitialisées à chaque requête.

De plus, l'attribut `sessionViews` sera incrémenté à chaque requête.

1.3

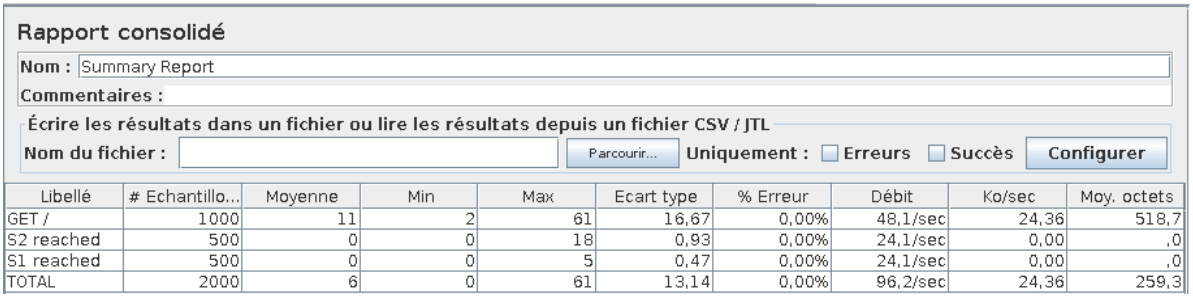
> Provide a sequence diagram to explain what is happening when one requests the URL for the first time and then refreshes the page. We want to see what is happening with the cookie. We want to see the sequence of messages exchanged (1) between the browser and HAProxy and (2) between HAProxy and the nodes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



Nous voyons que les requêtes successives sont envoyées à des serveurs différents (S1 et S2). Mais comme ils ne connaissent pas les sessions de l'un et l'autre, ils recréent des sessions à chaque fois avec un `set-cookie`.

1.4

> Provide a screenshot of the summary report from JMeter.



Libellé	# Echantillo...	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec	Moy. octets
GET /	1000	11	2	61	16,67	0,00%	48,1/sec	24,36	518,7
S2 reached	500	0	0	18	0,93	0,00%	24,1/sec	0,00	,0
S1 reached	500	0	0	5	0,47	0,00%	24,1/sec	0,00	,0
TOTAL	2000	6	0	61	13,14	0,00%	96,2/sec	24,36	259,3

Ainsi, on voit que les requêtes sont envoyées alternativement aux serveurs S1 et S2. Chaque serveur reçoit un nombre égal de requêtes.

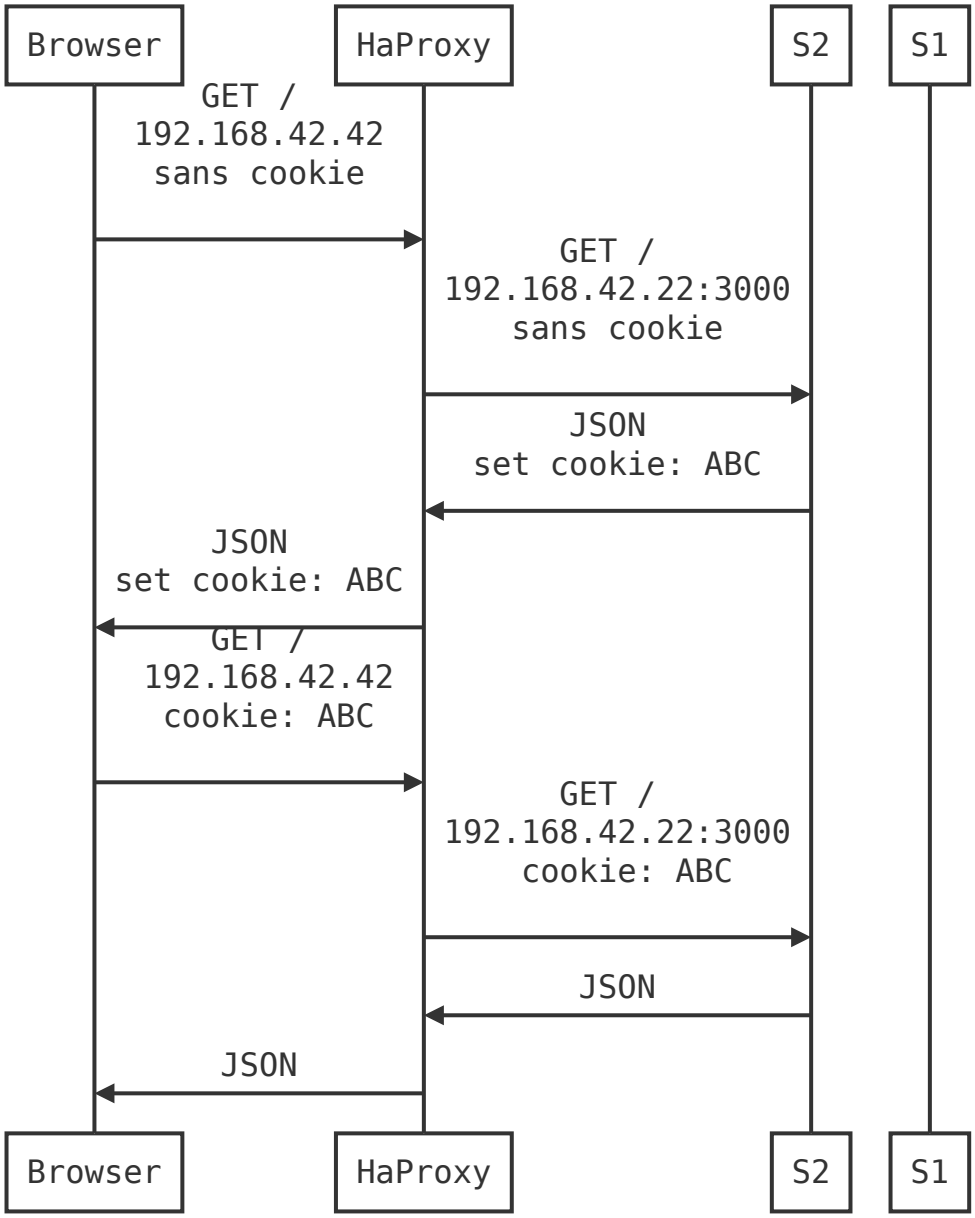
1.5

> 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.

Nous avons premièrement stoppé le serveur S1:

```
docker stop s1
```

Ainsi, HAProxy envoie des requêtes qu'au serveur actif , S2, car il détecte que S1 est stoppé. S2 ne renvoie donc pas de `set-cookie` car il connait la session reçue (cookie ABC).



Rapport consolidé

Nom : Summary Report

Commentaires :

Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : Erreurs Succès Configurer

Libellé	# Echantillo...	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec	Moy. octets
GET /	1000	12	2	54	17,24	0,00%	48,2/sec	18,58	395,0
S2 reached	1000	0	0	21	0,98	0,00%	48,2/sec	0,00	,0
TOTAL	2000	6	0	54	13,56	0,00%	96,3/sec	18,57	197,5

Nous remarquons effectivement que les tests ne sont effectués que sur S2.

2. La persistence des sessions

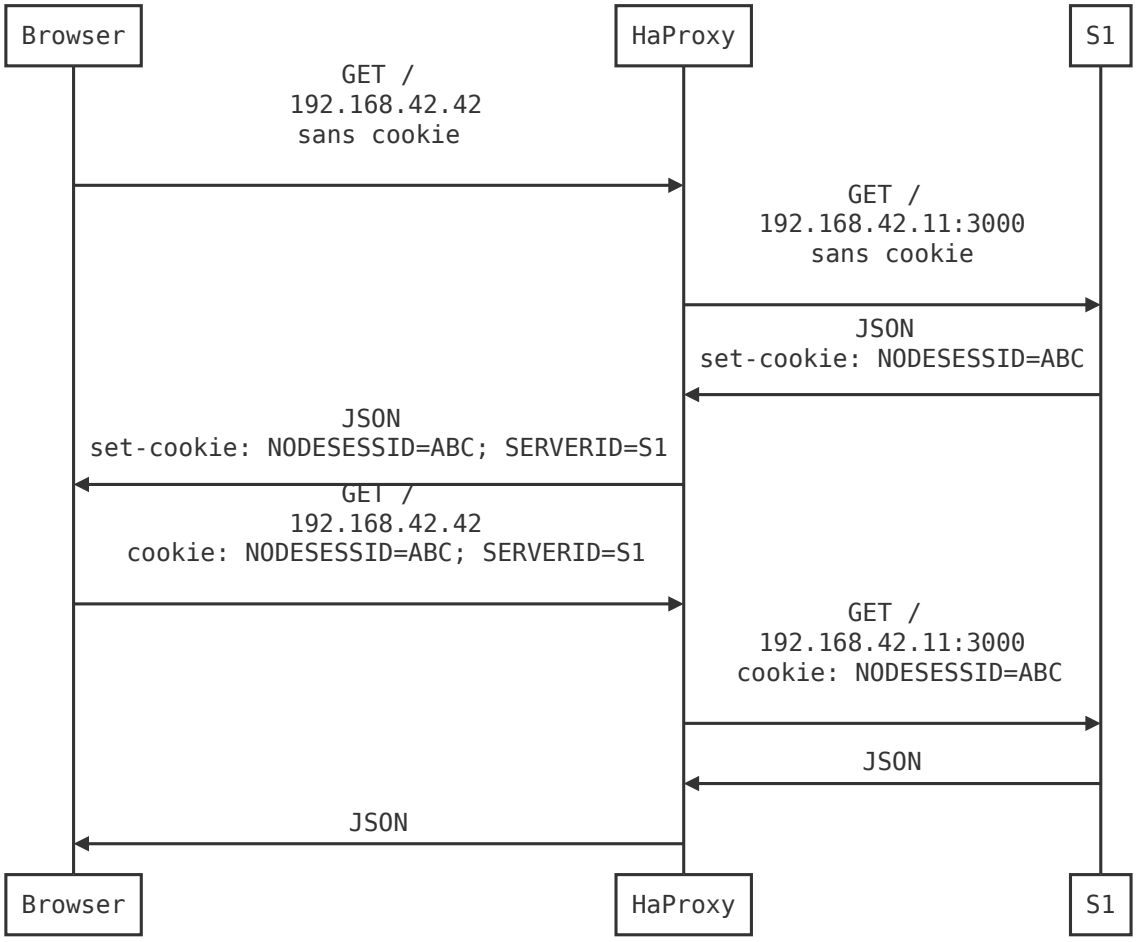
2.1

> 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).

> SERVERID

Nous voyons sur le diagramme ci-dessous, que le client possède deux cookies différents:

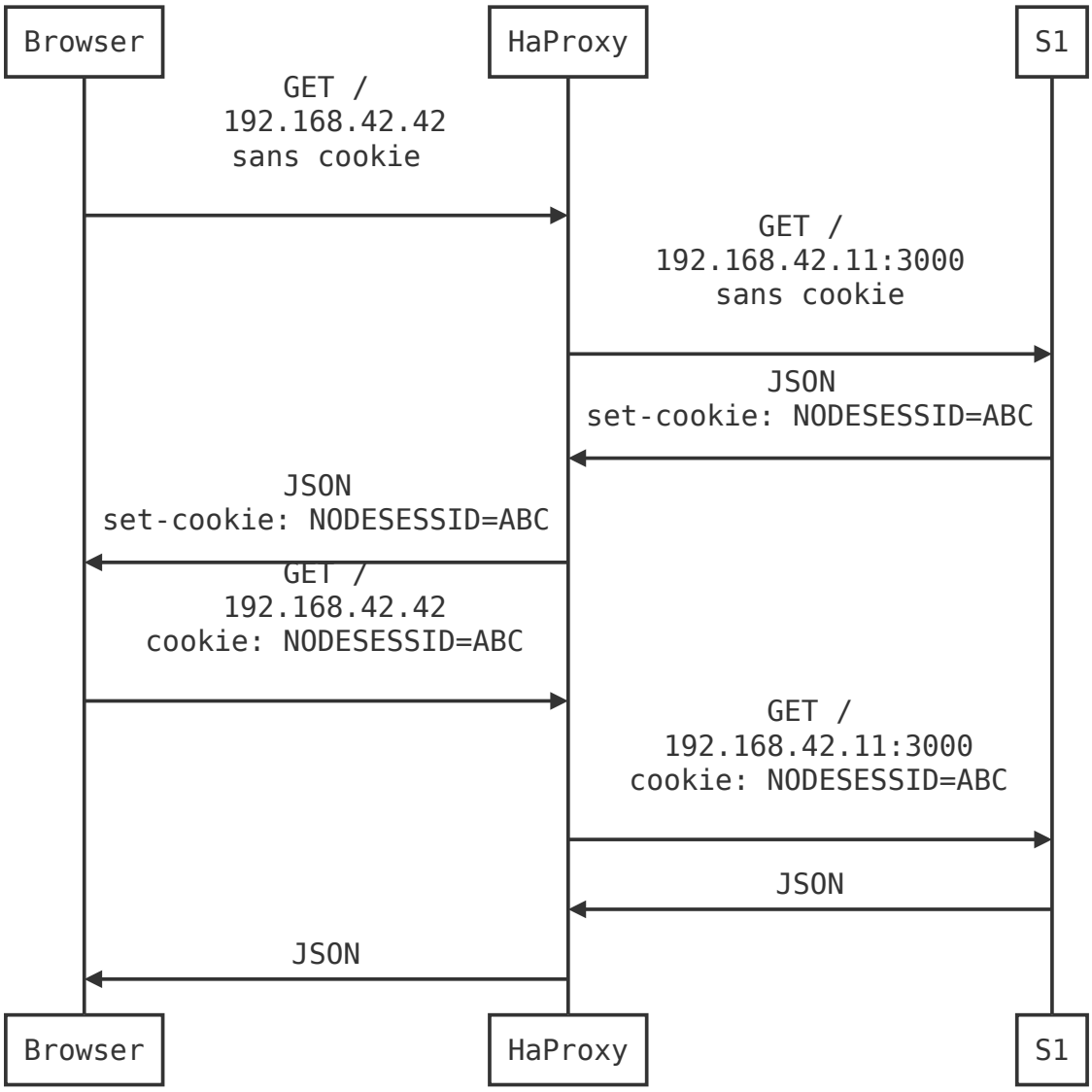
- `NODESSESSID` qui est lié à l'application
- `SERVERID` qui identifie le noeud et qui est défini par le HAProxy. Il n'est pas lié à l'application et donc il fonctionne quelque soit l'application.



> NODESESSID

Lors de l'utilisation de `NODESESSID`, nous ne configurons plus qu'un cookie qui est directement lié à notre application. Il pourrait y avoir un souci si l'application déciderait de modifier le cookie et que HAProxy n'arrive pas à detecter la modification. HAProxy ne saurait plus vers quel serveur il doit envoyer la requête, donc il pourrait avoir un problème avec la session.

Lors de l'utilisation de cet unique cookie, HAProxy sait vers quel serveur rediriger les requêtes car il garde une table de mapping en interne.



2.2

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

```
# 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 cookie S1 check
server s2 ${WEBAPP_2_IP}:3000 cookie S2 check

cookie SERVERID insert
```

Nous avons modifié le fichier de configuration de HAProxy afin de configurer le sticky session. Pour cela, nous avons ajouté un paramètre, `cookie S1` et `cookie S2`, sur chaque noeud de la liste afin de lui ajouter un cookie.

Puis nous avons ajouté une ligne qui précise que les cookies utilisés sont de type `SERVERID` et le paramètre `insert` permet d'ajouter un cookie s'il n'existe pas.

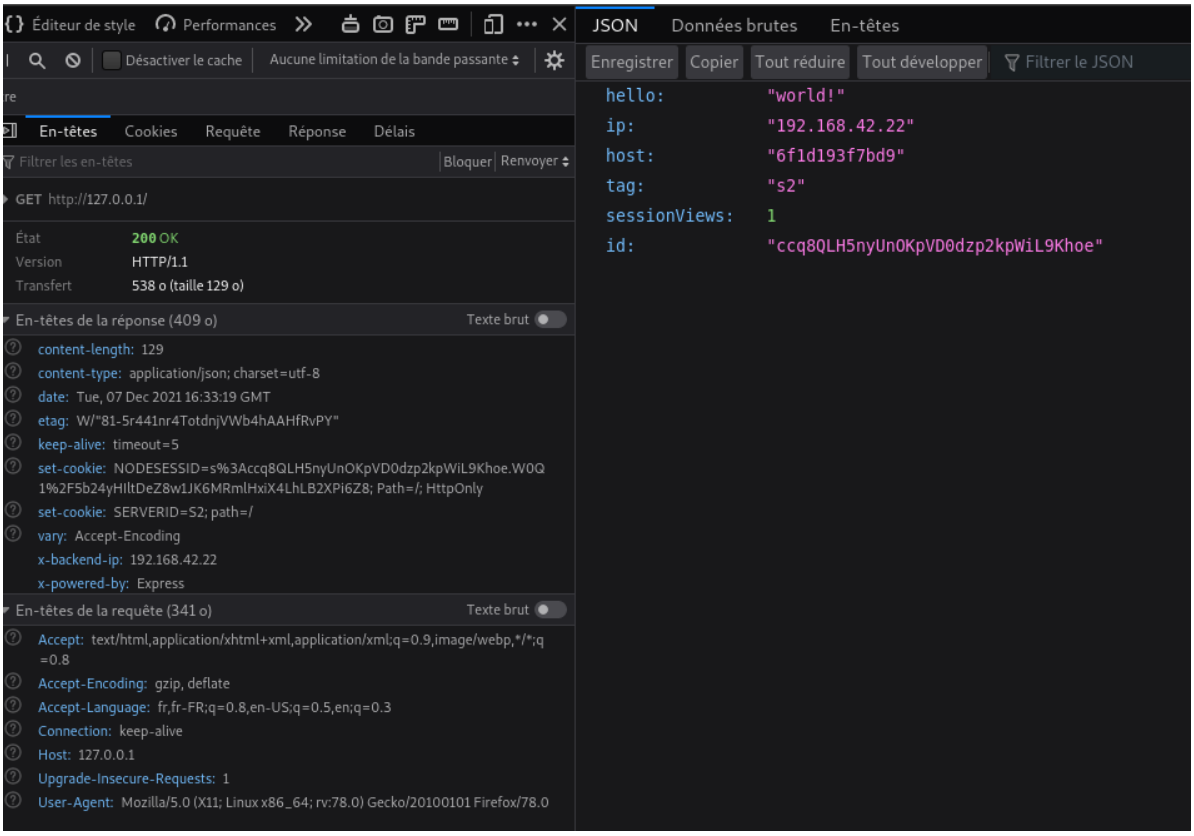
Ainsi, pour la suite des manipulations, nous utiliserons l'approche `SERVERID`.

```
elerille  master + 1  …  ait  heig-ait-lab  Labo_3  git diff
diff --git a/ha/config/haproxy.cfg b/ha/config/haproxy.cfg
index c3b6d47..24872d2 100755
--- a/ha/config/haproxy.cfg
+++ b/ha/config/haproxy.cfg
@@ -107,8 +107,10 @@ backend nodes

     # 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
-    server s2 ${WEBAPP_2_IP}:3000 check
+    server s1 ${WEBAPP_1_IP}:3000 cookie S1 check
+    server s2 ${WEBAPP_2_IP}:3000 cookie S2 check
+
+    cookie SERVERID insert

 # Other links you will need later for this lab
 #
```

2.3

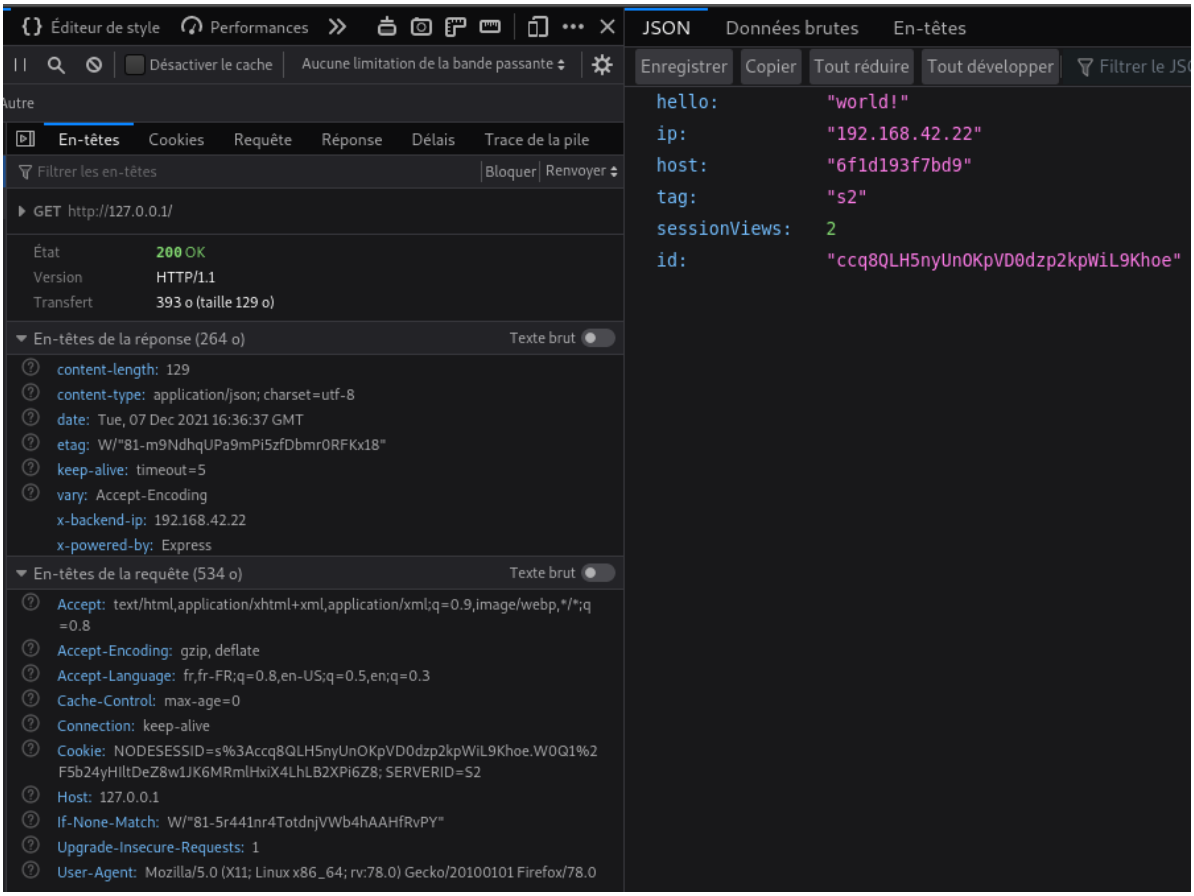
> Explain what is the behavior when you open and refresh the URL http://192.168.42.42 in your browser. Add screenshots to complement your explanations. We expect that you take a deeper a look at session management.

Nous constatons que lors de la première requête, nous sommes redirigés sur le S2 et nous pouvons voir que dans la réponse il y a les deux cookies, `NODESSESSID` et `SERVERID`. Pour le second cookie, nous voyons que sa valeur est `S2`, ce qui est correct.



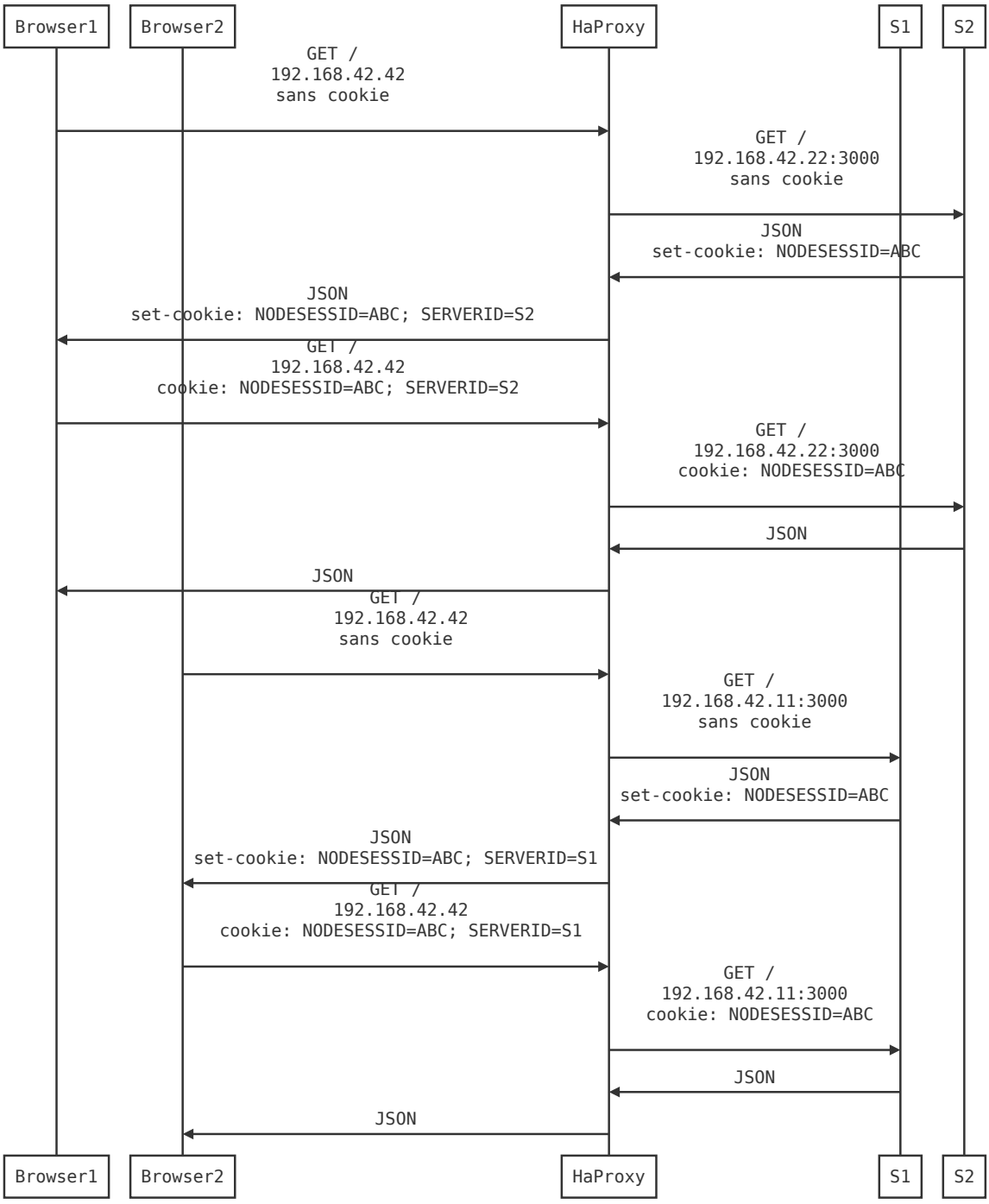
Lorsque l'on rafraîchit la page, nous sommes également redirigés vers le serveur S2. Nous constatons que le `sessionViews` est incrémenté de 1. Le comportement est celui attendu car grâce aux cookies, nous gardons la même session.

De plus, nous voyons que dans la requête, le cookie `NODESSESSID` reçu lors de la première requête, est envoyé vers le serveur afin d'indiquer l'id de la session.



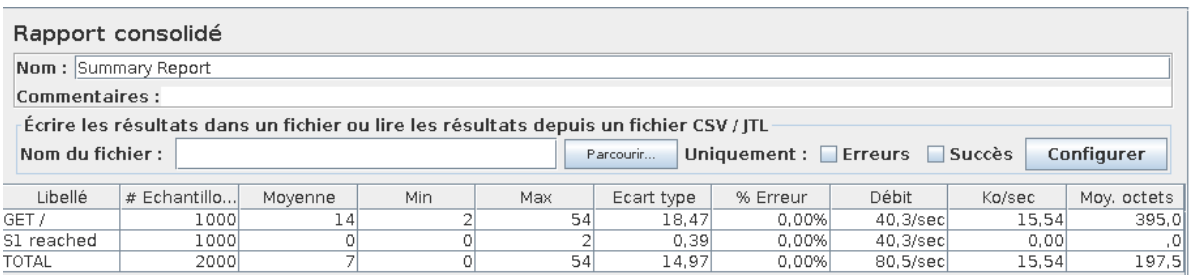
2.4

> 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?



Rapport consolidé

Nom : Summary Report

Commentaires :

Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

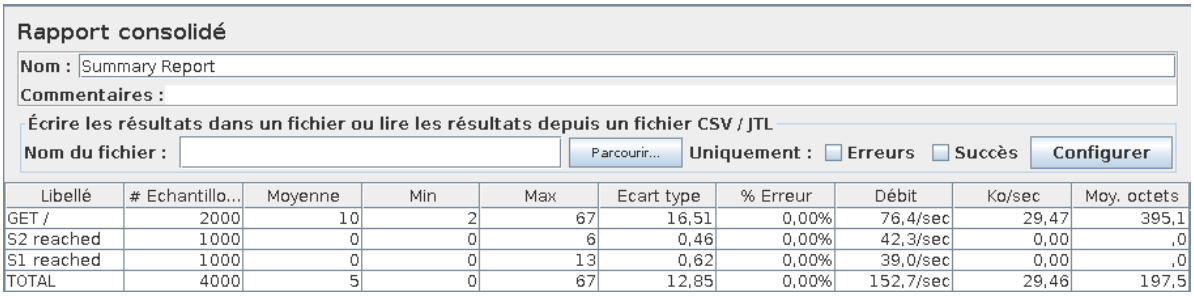
Nom du fichier : Parcourir... Uniquement : Erreurs Succès Configurer

Libellé	# Echantillo...	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec	Moy. octets
GET /	1000	14	2	54	18,47	0,00%	40,3/sec	15,54	395,0
S1 reached	1000	0	0	2	0,39	0,00%	40,3/sec	0,00	,0
TOTAL	2000	7	0	54	14,97	0,00%	80,5/sec	15,54	197,5

Nous remarquons que toutes les requêtes atteignent le serveur S1 alors que dans la task 1 les requêtes étaient partagés entre les serveurs S1 et S2 dû à la configuration du round-robin. Nous pouvons expliquer cette différence avec le fait que pour cette tâche, le serveur S1 est configuré avec des sticky sessions et donc les requêtes du navigateur seront toujours redirigées vers S1.

2.7

> 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.



Libellé	# Echantillo...	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec	Moy. octets
GET /	2000	10	2	67	16,51	0,00%	76,4/sec	29,47	395,1
S2 reached	1000	0	0	6	0,46	0,00%	42,3/sec	0,00	,0
S1 reached	1000	0	0	13	0,62	0,00%	39,0/sec	0,00	,0
TOTAL	4000	5	0	67	12,85	0,00%	152,7/sec	29,46	197,5

Nous voyons que le premier thread sera assigné à un serveur grâce à un cookie et il sera toujours redirigé dessus. Le second thread est assigné au deuxième serveur et il restera également dessus.

3. Le drainage des connexions

3.1

> Take a screenshot of step 5 and tell us which node is answering.

Nous avons rafraîchit la page plusieurs fois, 18 fois, et nous constatons que le noeud qui répond est le serveur S2.

JSON Données brutes En-têtes

Enregistrer Copier Tout réduire Tout développer Filtrer le JSON


```
hello:          "world!"
ip:             "192.168.42.22"
host:           "698dbc3f27bb"
tag:            "s2"
sessionViews:   19
id:             "BZCqGJjYOjAd8xIMR3_qi1lunLrWvTaC"
```


HAProxy

Statistics Report for pid 12

> General process information

pid = 12 (process #1, nbproc = 1, nbthread = 8)
uptime = 0d 0h05m50s
system limits: memmax = unlimited; ulimit-n = 1048576
maxsock = 1048576; **maxconn =** 524263; **maxpipes =** 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec; bit rate = 0.271 kbps
Running tasks: 1/26; 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	1	524 263	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	5m50s 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	524 263	2			8 783	6 988	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	5m46s UP	L7OK/200 in 4ms	1	Y	-	1	1	4s	-
s2	0	0	-	0	2		0	1	-	18	1	16s	8 783	6 988		0		0	0	0	0	5m50s UP	L7OK/200 in 7ms	1	Y	-	0	0	0s	-
Backend	0	0		0	2		0	1	52 427	18	1	16s	8 783	6 988	0	0		0	0	0	0	5m50s UP		2	2	0		0	0s	

Nous constatons que les deux serveurs sont UP et que nous avons une session sur S2.

3.2

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

Afin de configurer le mode `DRAIN` sur S2, on exécute cette commande avec `socat` :

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

Nous voyons qu'en bleu foncé sur la capture ci-dessous, le serveur S2 est bien en mode `DRAIN`.

HAProxy

Statistics Report for pid 11

> General process information

pid = 11 (process #1, nbproc = 1, nbthread = 8)
uptime = 0d 0h10m18s
system limits: memmax = unlimited; ulimit-n = 1048576
maxsock = 1048576; **maxconn =** 524263; **maxpipes =** 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec; bit rate = 0.271 kbps
Running tasks: 1/28; 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	1	524 263	2			1 368	63 774	0	0	0					OPEN								
Backend	0	0		0	0		0	0	52 427	0	0	0s	1 368	63 774	0	0		0	0	0	0	10m18s 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	524 263	1			1 494	1 264	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	10m14s UP	L7OK/200 in 4ms	1	Y	-	1	1	4s	-
s2	0	0	-	0	1		0	1	-	3	0	6s	1 494	1 264		0		0	0	0	0	14s DRAIN	L7OK/200 in 6ms	1	Y	-	0	0	0s	-
Backend	0	0		0	1		0	1	52 427	3	0	6s	1 494	1 264	0	0		0	0	0	0	10m18s UP		1	1	0		0	0s	

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 not, why?

Nous sommes toujours sur le même noeud car le cookie nous indique que nous devons aller sur ce noeud (`s2`) et donc HAProxy nous laisse communiquer avec le `s2`.

Ce comportement est expliqué par le mode `DRAIN` car étant donné qu'on a configuré le serveur S2 dans ce mode et qu'on avait déjà une session liée à ce serveur, on est toujours redirigé sur ce dernier. Par contre, le nouveau trafic sera redirigé sur S1.

JSON Données brutes En-têtes

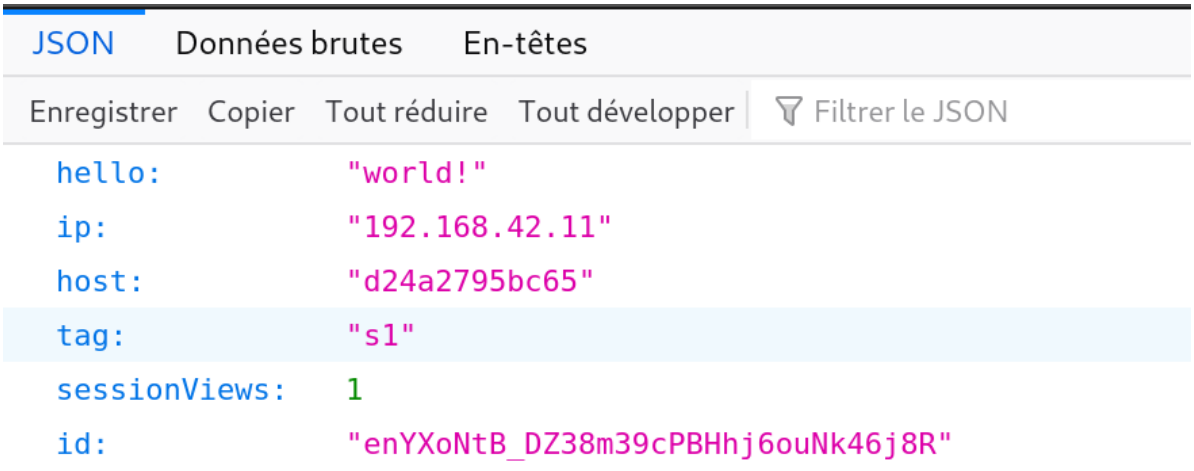
Enregistrer Copier Tout réduire Tout développer Filtrer le JSON


```
hello:          "world!"
ip:             "192.168.42.22"
host:           "698dbc3f27bb"
tag:            "s2"
sessionViews:   21
id:             "BZCqGJjYOjAd8xIMR3_qi1lunLrWvTaC"
```


3.4

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

Nous sommes sur le S1 car c'est une nouvelle connexion et qu'on avait encore aucune session.



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?

On se retrouve de nouveau sur le S1 ce qui est normal car nous n'avons plus de cookie `SERVEID` lié au S2. De ce fait, nous ne pouvons plus atteindre S2, qui est le noeud configuré en mode `DRAIN`.

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 exécuté la commande suivante:

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

Nous voyons sur la capture ci-dessous que les deux serveurs sont UP et que le serveur S2 n'est plus configuré en mode `DRAIN`.

HAProxy

Statistics Report for pid 11

> General process information

pid = 11 (process #1, nbproc = 1, nbthread = 8)
uptime = 0d 0h26m03s
system limits: memmax = unlimited; ulimit-n = 1048576
maxsock = 1048576; **maxconn =** 524263; **maxpipes =** 0
current conns = 1; current pipes = 0/0; conn rate = 0/sec; bit rate = 0.271 kbps
Running tasks: 1/27; 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				0	1	-	1	3	524 263	4			2 622	128 754	0	0	0					OPEN								
Backend	0	0		0	0		0	0	52 427	0	0	0s	2 622	128 754	0	0		0	0	0	0	26m3s 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	524 263	14			16 712	14 056	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	4		0	1	-	24	4	3m3s	11 268	9 625		0		0	0	0	0	25m59s UP	L7OK/200 in 4ms	1	Y	-	1	1	4s	-
s2	0	0	-	0	3		0	1	-	11	0	37s	5 444	4 431		0		0	0	0	0	3s UP	L7OK/200 in 5ms	1	Y	-	0	0	15m56s	-
Backend	0	0		0	4		0	1	52 427	35	4	37s	16 712	14 056	0	0		0	0	0	0	26m3s UP		2	2	0		0	0s	

Nous avons tout d'abord utilisé Firefox pour faire les premières étapes; lors de la première connexion, on est redirigé sur le serveur S2. Une fois la page rafraîchie, on reste sur S2. Ceci est dû aux cookies.

Dans un second temps, nous avons utilisé un autre navigateur et nous avons été redirigés sur S1. Nous avions une chance sur deux de tomber sur le serveur S1 dû au round-robin.

Nous avons ensuite nettoyer les cookies du second navigateur et rafraîchit la page, nous avons atteint le serveur S2 car le navigateur n'avait aucune session active. En répétant la manipulation plusieurs fois, nous sommes tombés en alternance sur S1 puis sur S2 étant donné que HAProxy est configuré en mode round-robin.

HAProxy

Statistics Report for pid 11

> General process information

pid = 11 (process #1, nbproc = 1, nbthread = 8)
uptime = 0d 0h34m45s
system limits: memmax = unlimited; ulimit-n = 1048576
maxsock = 1048576; **maxconn =** 524263; **maxpipes =** 0
current conns = 2; current pipes = 0/0; conn rate = 1/sec; bit rate = 67.977 kbps
Running tasks: 1/28; 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	3	524 263	5			3 534	172 142	0	0	0					OPEN								
Backend	0	0		0	0		0	0	52 427	0	0	0s	3 534	172 142	0	0		0	0	0	0	34m45s 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	2	-	1	3	524 263	23			32 697	27 694	0	0	3					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	4		0	1	-	29	7	49s	13 979	11 903		0		0	0	0	0	34m41s UP	L7OK/200 in 1ms	1	Y	-	1	1	4s	-
s2	0	0	-	0	4		0	1	-	36	4	35s	18 718	15 080		0		0	0	0	0	8m45s UP	L7OK/200 in 4ms	1	Y	-	0	0	15m56s	-
Backend	0	0		0	4		0	1	52 427	65	11	35s	32 697	26 983	0	0		0	0	0	0	34m45s UP		2	2	0		0	0s	

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 avons exécuté la commande suivante:

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

Nous voyons sur la capture ci-dessous que les deux serveurs sont UP et que le serveur S2 est configuré en mode `MAINT`.

HAProxy

Statistics Report for pid 11

> General process information

pid = 11 (process #1, nbproc = 1, nbthread = 8)
uptime = 0d 0h36m50s
system limits: memmax = unlimited; ulimit-n = 1048576
maxsock = 1048576; **maxconn =** 524263; **maxpipes =** 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec; bit rate = 0.000 kbps
Running tasks: 1/28; 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	3	524 263	6			3 992	193 844	0	0	0					OPEN								
Backend	0	0		0	0		0	0	52 427	0	0	0s	3 992	193 844	0	0		0	0	0	0	36m50s 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	2	-	0	3	524 263	24			34 197	28 838	0	0	3					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	4		0	1	-	29	7	2m54s	13 979	11 903		0		0	0	0	0	36m46s UP	L7OK/200 in 4ms	1	Y	-	1	1	4s	-
s2	0	0	-	0	4		0	1	-	39	4	22s	20 218	16 224		0		0	0	0	0	7s MAINT		1	Y	-	0	1	16m3s	-
Backend	0	0		0	4		0	1	52 427	68	11	22s	34 197	28 127	0	0		0	0	0	0	36m50s UP		1	1	0		0	0s	

Nous avons tout d'abord utilisé Firefox pour faire les premières étapes; la session déjà existante qui pointait sur S2, une fois rafraîchie, a été redirigé vers S1 qui est le seul serveur actif.

Dans un second temps, nous avons utilisé un autre navigateur et nous sommes directement dirigés sur S1.

Peu importe le nombre de rafraîchissements et du nombre de cookies, toutes les requêtes sont redirigées vers S1 dû au fait que le serveur S2 est configuré en mode `MAINT` et que tout le nouveau et actuel trafic est redirigé vers les autres noeuds actifs.

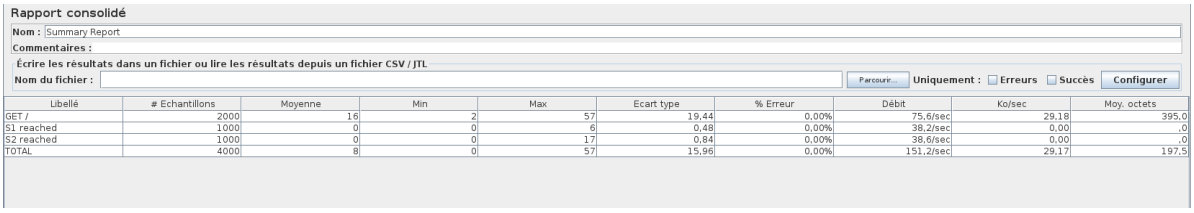
4. Le mode dégradé avec Round Robin

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.

On envoie une requête POST avec un delay de 0 milliseconde pour s'assurer qu'on n'a rien de configuré sur S1. On le fait également pour S2.

```
curl -H "Content-Type: application/json" -X POST -d '{"delay": 0}'
http://192.168.42.11:3000/delay
curl -H "Content-Type: application/json" -X POST -d '{"delay": 0}'
http://192.168.42.22:3000/delay
```


Rapport consolidé

Nom : Summary Report
Commentaires :
Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL
Nom du fichier : Parcourir... Uniquement : Erreurs Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec	Moy. octets
GET /	2000	16	2	57	19,44	0,00%	75,6/sec	29,18	395,0
S1 reached	1000	0	0	6	0,48	0,00%	38,2/sec	0,00	,0
S2 reached	1000	0	0	17	0,84	0,00%	38,6/sec	0,00	,0
TOTAL	4000	8	0	57	15,96	0,00%	151,2/sec	29,17	197,5

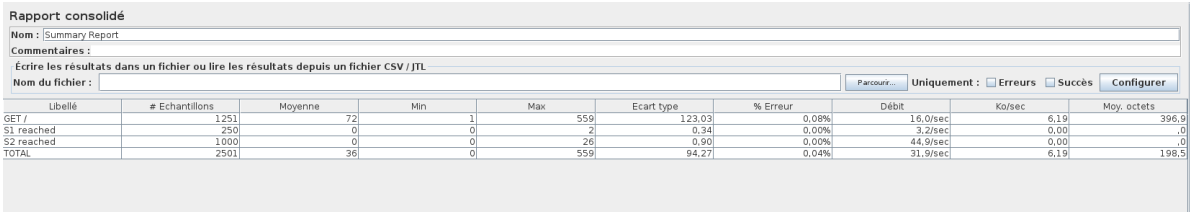
Nous voyons dans le rapport JMeter que HAProxy fonctionne normalement et envoie les requêtes en alternance sur les serveurs S1 et S2.

4.2

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

Nous ajoutons un délai de 250 millisecondes sur le serveur S1 en utilisant la commande `curl`:

```
curl -H "Content-Type: application/json" -X POST -d '{"delay": 250}'
http://192.168.42.11:3000/delay
# Résultat
{"message":"New timeout of 250ms configured."}
```


Rapport consolidé

Nom : Summary Report

Commentaires :

Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : Erreurs Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec	Moy. octets
GET /	1251	72	1	559	123,03	0,08%	16,0/sec	6,19	396,9
S1 reached	250	0	0	2	0,34	0,00%	3,2/sec	0,00	,0
S2 reached	1000	0	0	26	0,90	0,00%	44,9/sec	0,00	,0
TOTAL	2501	36	0	559	94,27	0,04%	31,9/sec	6,19	198,5

Nous n'avons pas attendu que S1 effectue ses 1000 requêtes car cela nous aurait pris trop de temps d'attendre la fin du rapport.

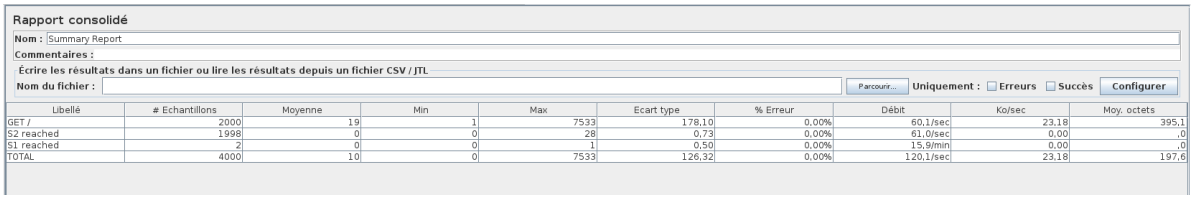
Nous constatons effectivement que S1 effectue des requêtes toutes les 250 millisecondes. Nous nous retrouvons avec une moyenne de `3.2 - 3.3` requêtes par secondes alors que le nombre de requêtes par secondes de `S2` est beaucoup plus élevé. Aussi, nous remarquons également qu'il n'y pas eu d'erreur lors du test malgré la lenteur de notre service. Les requêtes ont été transmises entre le S1 et S2.

4.3

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

Nous ajoutons un délai de 2500 millisecondes sur le serveur S1 en utilisant la commande `curl`:

```
curl -H "Content-Type: application/json" -X POST -d '{"delay": 2500}'
http://192.168.42.11:3000/delay
# Résultat
{"message":"New timeout of 2500ms configured."}
```


Rapport consolidé

Nom : Summary Report

Commentaires :

Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : Erreurs Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec	Moy. octets
GET /	2000	19	1	7533	178,10	0,00%	60,1/sec	23,18	395,1
S2 reached	1998	0	0	28	0,73	0,00%	61,0/sec	0,00	,0
S1 reached	2	0	0	1	0,50	0,00%	15,9/min	0,00	,0
TOTAL	4000	10	0	7533	126,32	0,00%	120,1/sec	23,18	197,6

Nous voyons que S1 a uniquement effectué 2 requêtes car il avait un temps de réponse relativement lent. Le load balancer redirige vers les différents serveurs en fonction de leur état. Ici, le S1 étant plus lent, HAProxy a redirigé toutes les autres requêtes vers S2.

4.4

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

Nous constatons que pour l'étape 2, avec un délai de 250 millisecondes, il y a 0.08% d'erreur pour les requêtes du serveur S1 ce qui correspond à une requête échouée lorsqu'on a cliqué sur le bouton stop et que la réponse n'était pas encore arrivée. Sans cette erreur de manipulation, il devrait y avoir 0% d'erreur malgré une grande lenteur du service.

Le load balancer permet de détecter l'état des serveurs. Etant donné que S1 étant trop lent, HAProxy l'a retiré du pool de serveur et a redirigé le trafic vers S2 pour garder un tafic cohérent et sans erreur.

4.5

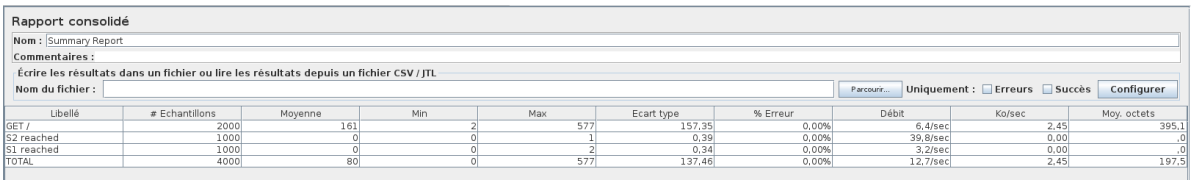
> Update the HAProxy configuration to add a weight to your nodes. For that, add `weight [1-256]` where the value of weight is between the two values (inclusive). Set `s1` to 2 and `s2` to 1. Redo a run with a 250ms delay.

Nous avons directement modifier la configuration HAProxy et ajouter un paramètre `weigt` pour les deux serveurs. Le serveur S1 a une poids de `2` et le serveur S2 un poids de `1`.

```
# 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 cookie S1 check weight 2
server s2 ${WEBAPP_2_IP}:3000 cookie S2 check weight 1
```

Nous ajoutons également un délai de 250 millisecondes sur le serveur S1.

```
curl -H "Content-Type: application/json" -X POST -d '{"delay": 250}'
http://192.168.42.11:3000/delay
# Résultat
{"message":"New timeout of 250ms configured."}
```



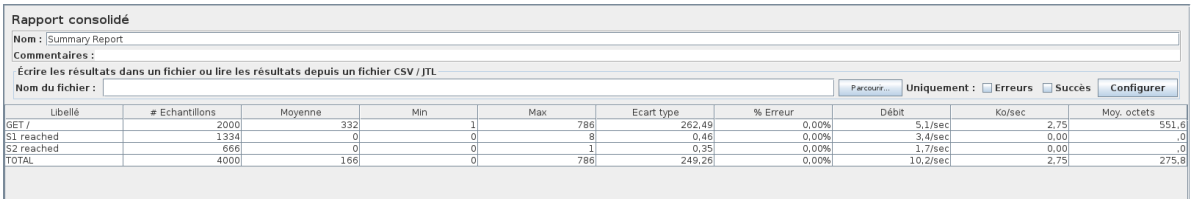
Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec	Moy. octets
GET /	2000	161	2	577	157,35	0,00%	6,4/sec	2,45	395,1
S2 reached	1000	0	0	1	0,39	0,00%	39,8/sec	0,00	,0
S1 reached	1000	0	0	2	0,34	0,00%	3,2/sec	0,00	,0
TOTAL	4000	80	0	577	137,46	0,00%	12,7/sec	2,45	197,5

Ainsi, nous voyons sur le rapport JMeter que HAProxy fonctionne normalement; les requêtes sont redirigées de manière égale entre les deux serveurs. Ce résultat est dû au sticky session.

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.

> Rapport sans les cookies



Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec	Moy. octets
GET /	2000	332	1	786	262,49	0,00%	5,1/sec	2,75	551,6
S1 reached	1334	0	0	8	0,46	0,00%	3,4/sec	0,00	,0
S2 reached	666	0	0	1	0,35	0,00%	1,7/sec	0,00	,0
TOTAL	4000	166	0	786	249,26	0,00%	10,2/sec	2,75	275,8

Nous remarquons donc qu'en nettoyant les cookies entre chaque requêtes, 2/3 des requêtes vont sur S1 et le 1/3 restant sur S2, on peut expliquer cela avec le fait que S1 a un poids de 2 et S2 un poids de 1, et donc S1 reçoit logiquement 2x plus de requêtes que S2.

> Avec les cookies

Alors qu'avec les cookies, chaque serveur reçoit 1000 requêtes car lorsqu'un client est connecté à un noeud, il y restera connecté jusqu'à la fin du test (malgré la lenteur du service) dû au sticky session.

5. Les stratégies de load balancing

5.1

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

> **Leastconn**

La stratégie `leastconn` permet au serveur qui a le moins de connexions, de recevoir la nouvelle connexion. Elle utilise round-robin dans le cas où la charge est égale sur tous les serveurs pour s'assurer qu'ils seront tous utilisés. L'algorithme est dynamique dans le sens où les poids des serveurs seront ajustés en fonction du nombre de connexions actives.

Nous avons décidé de choisir cette stratégie car elle peut être intéressante lors d'ajout de délai sur un serveur.

> **First**

La stratégie `first` permet à un serveur qui a des slots de connexions de libre, de recevoir la nouvelle connexion. Le serveur est choisi en fonction de son id de manière croissante. Dès qu'un serveur atteint son nombre maximum de connexion (`maxconn`), c'est le serveur suivant qui sera utilisé. Le but de cet algorithme est d'optimiser le nombre de serveurs utilisés.

Nous avons décidé d'utiliser cette stratégie car elle permet d'éteindre les serveurs avec des grands id lorsqu'il y a peu de charge sur l'application.

5.2

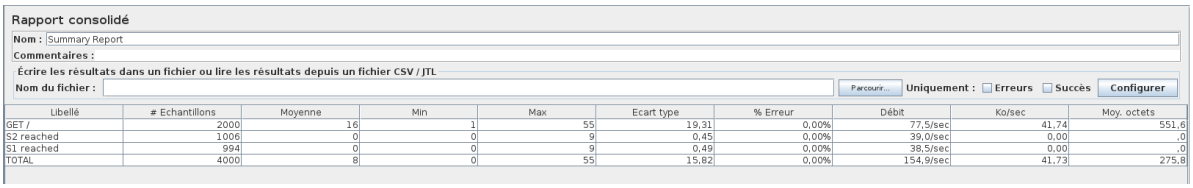
> Provide evidence that you have played with the two strategies (configuration done, screenshots, ...)

> **Leastconn**

Configuration dans HAProxy afin d'utiliser la stratégie `leastconn`:

```
# Define the balancing policy
# http://cbonte.github.io/haproxy-dconv/2.2/configuration.html#balance
balance leastconn
```

> Nettoyage des cookies, sans delay


Rapport consolidé

Nom : Summary Report

Commentaires :

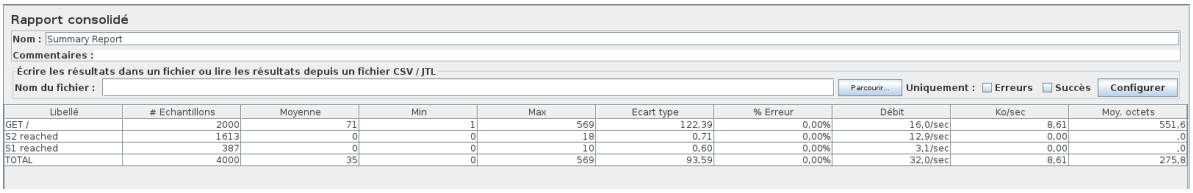
Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : Erreurs Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec	Moy. octets
GET /	2000	16	1	55	19,31	0,00%	77,5/sec	41,74	551,6
S2 reached	1006	0	0	9	0,45	0,00%	39,0/sec	0,00	,0
S1 reached	994	0	0	9	0,49	0,00%	38,5/sec	0,00	,0
TOTAL	4000	8	0	55	15,82	0,00%	154,9/sec	41,73	275,8

S2 a un peu plus travaillé que S1, car à un moment donné S1 devait être un peu plus chargé que S2, mais de manière générale la charge est répartie de manière égale (round-robin lors d'une charge égale).

> Nettoyage des cookies, delay de 250 millisecondes sur S1


Rapport consolidé

Nom : Summary Report

Commentaires :

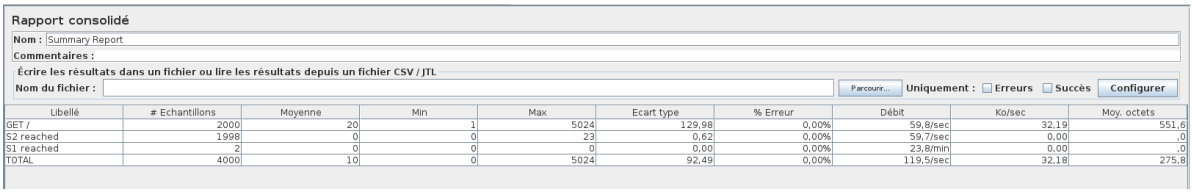
Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : Erreurs Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec	Moy. octets
GET /	2000	71	1	569	122,39	0,00%	16,0/sec	8,61	551,6
S2 reached	1613	0	0	18	0,71	0,00%	12,9/sec	0,00	,0
S1 reached	387	0	0	10	0,60	0,00%	3,1/sec	0,00	,0
TOTAL	4000	35	0	569	93,59	0,00%	32,0/sec	8,61	275,8

Nous pouvons constater que S1 étant plus lent à répondre, il a donc traité moins de requêtes que le serveur S2.

> Nettoyage des cookies, delay de 2500 millisecondes sur S1


Rapport consolidé

Nom : Summary Report

Commentaires :

Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : Erreurs Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec	Moy. octets
GET /	2000	20	1	5024	129,98	0,00%	59,8/sec	32,19	551,6
S2 reached	1998	0	0	23	0,62	0,00%	59,7/sec	0,00	,0
S1 reached	2	0	0	0	0,00	0,00%	23,8/min	0,00	,0
TOTAL	4000	10	0	5024	92,49	0,00%	119,5/sec	32,18	275,8

Nous pouvons remarquer que S1 étant plus lent à répondre, il a donc traité beaucoup moins de requêtes que le serveur S2. Puisque S1 demande plus de temps pour répondre et que le serveur S2 est disponible plus rapidement, le load balancer transmet les requêtes vers S2 et plus vers S1.

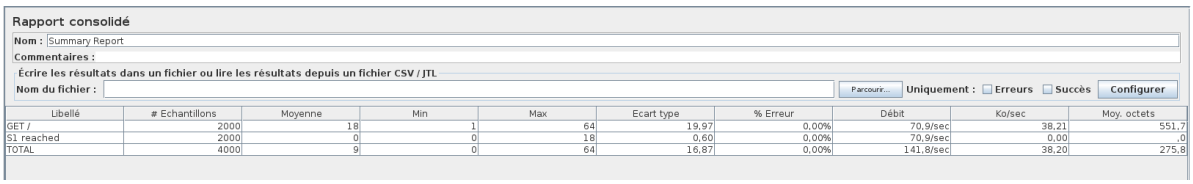
First

Configuration dans HAProxy afin d'ajouter la stratégie `first` :

```
# Define the balancing policy
# http://cbonte.github.io/haproxy-dconv/2.2/configuration.html#balance
balance first
# 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 cookie S1 check maxconn 2
server s2 ${WEBAPP_2_IP}:3000 cookie S2 check maxconn 2
```

Nous avons dû ajouter un paramètre `maxconn` sur les deux serveurs afin qu'ils puissent prendre en charge deux connexions en parallèle au maximum.

> Nettoyage des cookies, sans delay, avec 2 threads


Rapport consolidé

Nom : Summary Report

Commentaires :

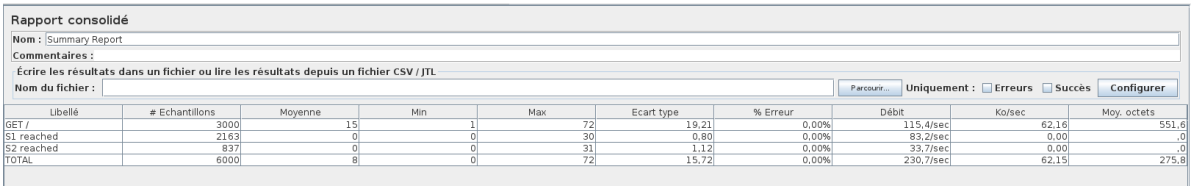
Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : Erreurs Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec	Moy. octets
GET /	2000	18	1	64	19,97	0,00%	70,9/sec	38,21	551,7
S1 reached	2000	0	0	18	0,60	0,00%	70,9/sec	0,00	,0
TOTAL	4000	9	0	64	16,87	0,00%	141,8/sec	38,20	275,8

Nous avons configuré JMeter pour utiliser 2 threads et vu que HAProxy est configuré pour avoir 2 connexions par serveur, nous tombons que sur S1 car à aucun moment nous avons suffisament de connexions ouvertes pour que HAProxy nous envoie sur S2.

> Nettoyage des cookies, sans delay, avec 3 threads


Rapport consolidé

Nom : Summary Report

Commentaires :

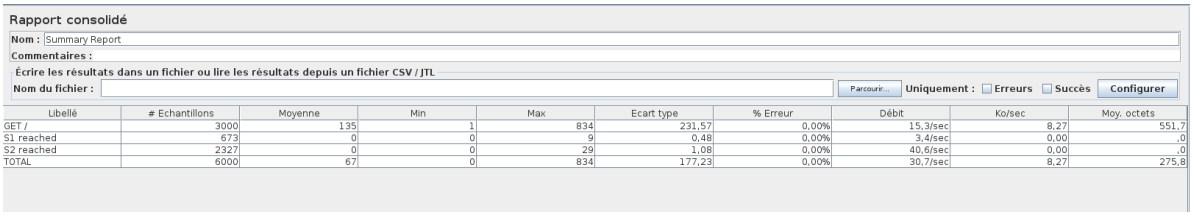
Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : Erreurs Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec	Moy. octets
GET /	3000	15	1	72	19,21	0,00%	115,4/sec	62,16	551,6
S1 reached	2163	0	0	30	0,80	0,00%	83,2/sec	0,00	,0
S2 reached	837	0	0	31	1,12	0,00%	33,7/sec	0,00	,0
TOTAL	6000	8	0	72	15,72	0,00%	230,7/sec	62,15	275,8

En ayant configuré JMeter avec 3 threads, nous voyons que HAProxy redirige à peu près 1/3 des connexions sur S2. Nous constatons que c'est en dessous de 1/3 car il y a certainement des moments de vide lors des requêtes que fait JMeter et donc HAProxy repasse sur deux serveurs.

> Nettoyage des cookies, avec un delay de 250 millisecondes sur S1, avec 3 threads


Rapport consolidé

Nom : Summary Report

Commentaires :

Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : Erreurs Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec	Moy. octets
GET /	3000	135	1	834	231,57	0,00%	15,3/sec	8,27	551,7
S1 reached	673	0	0	9	0,48	0,00%	3,4/sec	0,00	,0
S2 reached	2327	0	0	29	1,08	0,00%	40,6/sec	0,00	,0
TOTAL	6000	67	0	834	177,23	0,00%	30,7/sec	8,27	275,8

Nous remarquons que 1/5 des requêtes sont envoyées sur S1, alors que nous avons que 3 threads et que S1 traite deux requêtes en parallèle. Malgré que S1 soit lent, nous voyons que notre service est encore passablement rapide pour une grande majorité des utilisateurs.

5.3

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

> **First**

Cette stratégie permet d'aligner la charge d'un serveur en fonction de sa capacité calculée en amont. Elle offre donc la possibilité d'éteindre les serveurs non-utilisés et de minimiser les coûts en employant uniquement la quantité minimale de serveurs selon la charge courante. Attention, l'utilisation de cette stratégie nécessite que les serveurs puissent traiter plusieurs requêtes en même temps.

> **Leastconn**

Cette deuxième stratégie est plus adaptée pour des sessions à longue durée. Cela permet de ne pas renvoyer une requête à ce serveur, notament lorsqu'il est occupé à traiter un requête d'un client qui prend plusieurs secondes/minutes. Cependant pour que cela fonctionne, il faut que la connexion TCP reste ouverte, sinon le HAProxy n'a pas connaissance de ce traitement.

> **Conclusion**

C'est relativement compliqué d'indiquer quelle est la meilleure stratégie pour ce laboratoire car il ne correspond pas vraiment à un cas réel. Cependant si on prend comme cas d'analyse:

Un site web, dont les fichiers statiques (HTML/CSS/JS) sont servis par un autre serveur, que le service dynamique permet d'afficher des informations simples et rapides à calculer (une liste de prix, par exemple) mais dont certaines requêtes peuvent prendre plus de temps que d'autres (une liste de prix avec de multiple filtre, par exemple). Dans ce cas-là, nous conseillons d'utiliser le `leastconn`.

La stratégie `leastconn` est la meilleure selon nous. Elle permet une meilleure répartition de la charge car elle prend en compte le fait que certaines requêtes sont plus longues à traiter. Elle ne permet pas de mettre en veille certains serveurs, ainsi pour un petit nombre de nœuds c'est relativement compliqué, de ne pas pouvoir mettre des serveurs en veille, car il y a peu de marge de manœuvre sur le nombre de serveurs up/down.

Conclusion

Ce laboratoire a permis de tester différentes approches de load balancer:

- **round-robin** : les serveurs reçoivent de nouvelles connexions chacun leur tour
- **leastconn** : le serveur ayant le moins de connexion reçoit la prochaine nouvelle connexion
- **first** : le premier serveur reçoit autant de connexions qu'il est capable de gérer (selon la configuration) puis c'est le tour du serveur suivant

Nous avons également expérimenté 2 methodes pour lier les sessions des clients avec un unique serveur:

- **SERVERID** : un cookie est ajouté par HAProxy pour lui permettre d'identifier à quel nœud il doit transferer la requête
- **Cookie de session de l'application** : HAProxy réutilise un cookie de l'application et utilise un mappage interne pour identifier à quel serveur il doit transmettre la requête. Ce cookie doit être unique et avoir de très faible risque de collisions entre les différents nœuds.

Lors du fonctionnement de HAProxy les nœuds peuvent être dans divers états, voici les 3 qui ont été testés durant ce laboratoire:

- **READY** : Mode de fonctionnement standard, le serveur est apte à recevoir des requêtes
- **DRAIN** : Le serveur peut continuer à recevoir des requêtes de session en cours, mais il n'obtient plus aucune nouvelle session
- **MAINT** : Le serveur est en mode maintenance, plus aucune requête ne lui est transmisse. Les sessions qui lui étaient liées vont donc se briser et de nouvelles sessions seront créées sur un autre serveur (ou plusieurs)

L'outils JMeter a été utilisé pour faire des tests. Seul un script fourni a été utilisé, peu de connaissances ont été aquises sur cette application.

Annexes

https://d2c.io/post/haproxy-load-balancer-part-2-backend-section-algorithms

http://cbonte.github.io/haproxy-dconv/configuration-1.6.html#balance