UAM UNIVERSIDAD AUTONOMA DE MADRID		**Escuela Politécnica Superior** **Ingeniería Informática** **Prácticas de Sistemas Informáticos 2**			
Grupo	**2401**	**Práctica**	3	**Fecha**	04/05/2018
Alumno/a		Amor, Mourelle, Antonio			
Alumno/a		López, Ramos, Esther			

Ejercicio 1: **Preparar 3 máquinas virtuales con acceso SSH entre ellas. Esta tarea es necesaria para la correcta gestión del cluster que definiremos en el próximo apartado. Las VMs las denominaremos:**

- **si2srv01: Dirección IP 10.X.Y.1, 768MB RAM**
- **si2srv02: Dirección IP 10.X.Y.2, 512MB RAM**
- **si2srv03: Dirección IP 10.X.Y.3, 512MB RAM**

RECUERDE RANDOMIZAR LAS DIRECCIONES MAC DE CADA COPIA ANTES DE INTENTAR USAR EL NODO.

En la primera máquina (10.X.Y.1), generaremos el par de claves con DSA. A continuación importaremos la clave pública en cada uno de los otros dos nodos (10.X.Y.2 y 10.X.Y.3). Probaremos a acceder por SSH desde .1 a .2 y .3, comprobando que no requiere la introducción de la clave. Obtener una evidencia del inicio remoto de sesión mediante la salida detallada (ssh –v si2@10.X.Y.2 y ssh –v si2@10.X.Y.3). Anote dicha salida en la memoria de prácticas.

Una vez realizado este punto, detendremos las tres máquinas virtuales y obtendremos una copia de las mismas a algún medio externo (USB) para los consiguientes apartados de esta práctica. También es recomendable que preserve los directorios .ssh de cada uno de los nodos.

La salida obtenida es:

Para *ssh -v si2@10.1.4.2*:

debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug1: Connecting to 10.1.4.2 [10.1.4.2] port 22.
debug1: Connection established.
debug1: identity file /home/si2/.ssh/identity type -1
debug1: identity file /home/si2/.ssh/id_rsa type -1
debug1: identity file /home/si2/.ssh/id_dsa type 2
debug1: Checking blacklist file /usr/share/ssh/blacklist.DSA-1024
debug1: Checking blacklist file /etc/ssh/blacklist.DSA-1024
debug1: Remote protocol version 2.0, remote software version OpenSSH_5.3p1 Debian-3ubuntu7

debug1: match: OpenSSH_5.3p1 Debian-3ubuntu7 pat OpenSSH*
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-ctr hmac-md5 none
debug1: kex: client->server aes128-ctr hmac-md5 none
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
debug1: Host '10.1.4.2' is known and matches the RSA host key.
debug1: Found key in /home/si2/.ssh/known_hosts:1
debug1: ssh_rsa_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Trying private key: /home/si2/.ssh/identity
debug1: Trying private key: /home/si2/.ssh/id_rsa
debug1: Offering public key: /home/si2/.ssh/id_dsa
debug1: Server accepts key: pkalg ssh-dss blen 434
debug1: read PEM private key done: type DSA
debug1: Authentication succeeded (publickey).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: Sending environment.
debug1: Sending env LANG = C

Para *ssh -v si2@10.1.4.3*:

OpenSSH_5.3p1 Debian-3ubuntu7, OpenSSL 0.9.8k 25 Mar 2009
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug1: Connecting to 10.1.4.3 [10.1.4.3] port 22.
debug1: Connection established.
debug1: identity file /home/si2/.ssh/identity type -1
debug1: identity file /home/si2/.ssh/id_rsa type -1
debug1: identity file /home/si2/.ssh/id_dsa type 2
debug1: Checking blacklist file /usr/share/ssh/blacklist.DSA-1024
debug1: Checking blacklist file /etc/ssh/blacklist.DSA-1024
debug1: Remote protocol version 2.0, remote software version OpenSSH_5.3p1 Debian-3ubuntu7

```
debug1: match: OpenSSH_5.3p1 Debian-3ubuntu7 pat OpenSSH*
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-ctr hmac-md5 none
debug1: kex: client->server aes128-ctr hmac-md5 none
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
debug1: Host '10.1.4.3' is known and matches the RSA host key.
debug1: Found key in /home/si2/.ssh/known_hosts:2
debug1: ssh_rsa_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Trying private key: /home/si2/.ssh/identity
debug1: Trying private key: /home/si2/.ssh/id_rsa
debug1: Offering public key: /home/si2/.ssh/id_dsa
debug1: Server accepts key: pkalg ssh-dss blen 434
debug1: read PEM private key done: type DSA
debug1: Authentication succeeded (publickey).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: Sending environment.
debug1: Sending env LANG = C
```

Ejercicio 2: **Realizar los pasos del apartado 4 con el fin de obtener una configuración válida del cluster SI2Cluster, con la topología indicada de 1 DAS y 2 nodos SSH de instancias. Inicie el cluster. Liste las instancias del cluster y verifique que los pids de los procesos Java (JVM) correspondientes están efectivamente corriendo en cada una de las dos máquinas virtuales. Adjunte evidencias a la memoria de la práctica.**

Creamos los dos nodos mediante el comando asadmin y verificamos que se han creado correctamente listándolos:

```
si2@si2srv01:~$ asadmin --user admin --passwordfile /opt/SI2/passwordfile list-n
odes
localhost-domain1  CONFIG  localhost
Node01  SSH  10.1.4.2
Node02  SSH  10.1.4.3
Command list-nodes executed successfully.
si2@si2srv01:~$
```

Verificamos que los nodos están activos haciéndoles un ping:

```
si2@si2srv01:~$ asadmin --user admin --passwordfile /opt/SI2/passwordfile ping-n
ode-ssh Node01
Successfully made SSH connection to node Node01 (10.1.4.2)
Command ping-node-ssh executed successfully.
si2@si2srv01:~$
```

```
si2@si2srv01:~$ asadmin --user admin --passwordfile /opt/SI2/passwordfile  ping-
node-ssh Node02
Successfully made SSH connection to node Node02 (10.1.4.3)
Command ping-node-ssh executed successfully.
si2@si2srv01:~$
```

Comprobamos también que aparecen en la consola de administración de Glassfish de la máquina 1:

Nodes

A node represents a host on which the GlassFish Server software is installed. A node must exist for every host on which GlassFish Server instances reside.

Nodes (3)

New... | Delete | Delete and Uninstall

Select	Name	Node Host	Type	Instances	Action
☐	Node01	10.1.4.2	SSH		Ping
☐	Node02	10.1.4.3	SSH		Ping
☐	localhost-domain1	localhost	CONFIG		

Creamos el cluster SI2Cluster y verificamos que se ha creado correctamente listándolo:

```
si2@si2srv01:~$ asadmin list-clusters
SI2Cluster not running
Command list-clusters executed successfully.
si2@si2srv01:~$
```

Creamos las instancias asociadas a cada nodo, iniciamos el cluster y comprobamos que los pid de las instancias coinciden con pid de procesos Java en las máquinas virtuales:

```
si2@si2srv01:~$ asadmin --user admin --passwordfile /opt/SI2/passwordfile list-i
nstances -l
Name        Host      Port   Pid   Cluster     State
Instance01  10.1.4.2  24848  1861  SI2Cluster   running
Instance02  10.1.4.3  24848  1885  SI2Cluster   running
Command list-instances executed successfully.
si2@si2srv01:~$
```

Comprobemos además el pid en cada una de las máquinas virtuales:

Instance01:

```
si2@si2srv02:~$ ps -A | grep "1861"
 1861 ?        00:00:31 java
si2@si2srv02:~$
```

Instance02:

```
si2@si2srv03:~$ ps -A | grep 1885
 1885 ?        00:00:21 java
si2@si2srv03:~$
```

En cada una de las máquinas virtuales se encuentran las respectivas instancias ejecutándose. Nótese que se han omitidos los parámetros de llamada por claridad de las capturas.

Ejercicio 3: **Pruebe a realizar un pago individualmente en cada instancia. Para ello, identifique los puertos en los que están siendo ejecutados cada una de las dos instancias (IPs 10.X.Y.2 y 10.X.Y.3 respectivamente). Puede realizar esa comprobación directamente desde la consola de administración, opción Applications, acción Launch, observando los Web Application Links generados.**
Realice un único pago en cada nodo. Verifique que el pago se ha anotado correctamente el nombre de la instancia y la dirección IP. Anote sus observaciones (puertos de cada instancia) y evidencias (captura de pantalla de la tabla de pagos).

La dirección ip de las instancias se ve a continuación. Se han habilitado tanto el puerto sin conexion segura como con conexión https.

[Instance01] http://10.1.4.2:28080/P3

[Instance01] https://10.1.4.2:28181/P3

[Instance02] http://10.1.4.3:28080/P3

[Instance02] https://10.1.4.3:28181/P3

En la siguiente captura se pueden observar los dos pagos realizados. Para el primer pago se puede observar la instancia01 con la ip 10.1.4.2 y para el segunda pago se ve la instancia02 y la ip 10.1.4.3.

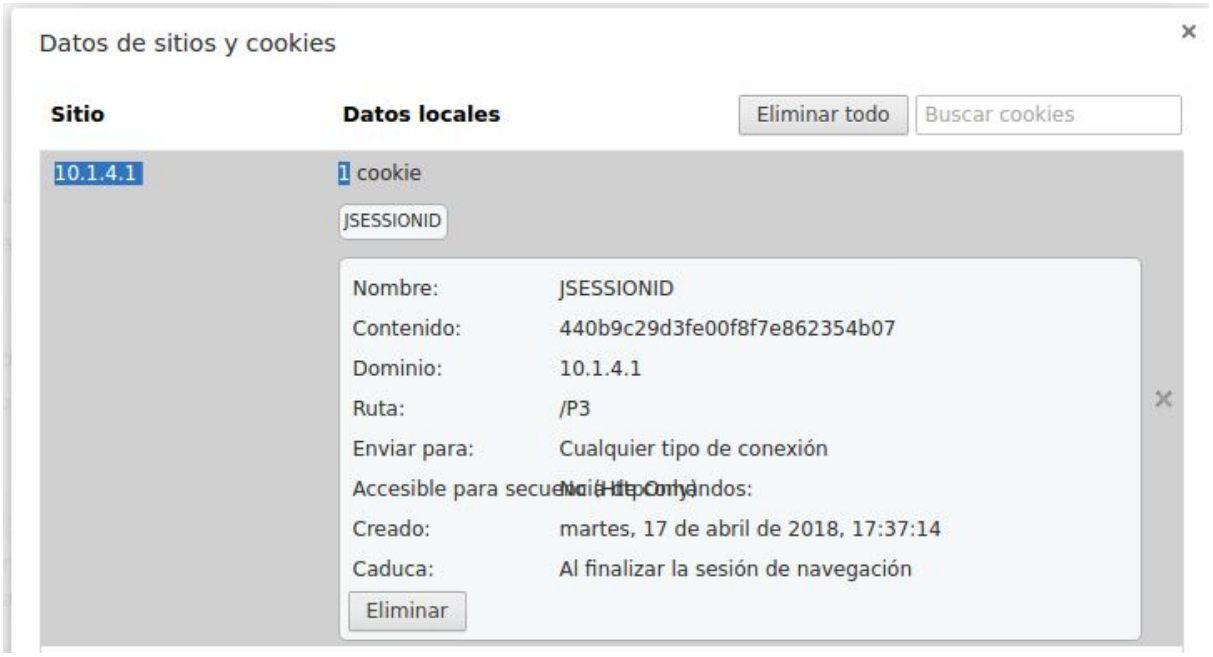
#	idautorizacion	idtransaccion	ısp	nport	comerc	numerotarjeta	fecha	instancia	ip
1	1	1	000	11	1	1111 2222 3333 4444	13/04/18 05:10	Instance01	10.1.4.2
2	2	3	000	11	1	1111 2222 3333 4444	13/04/18 05:15	Instance02	10.1.4.3

Ejercicio 4:

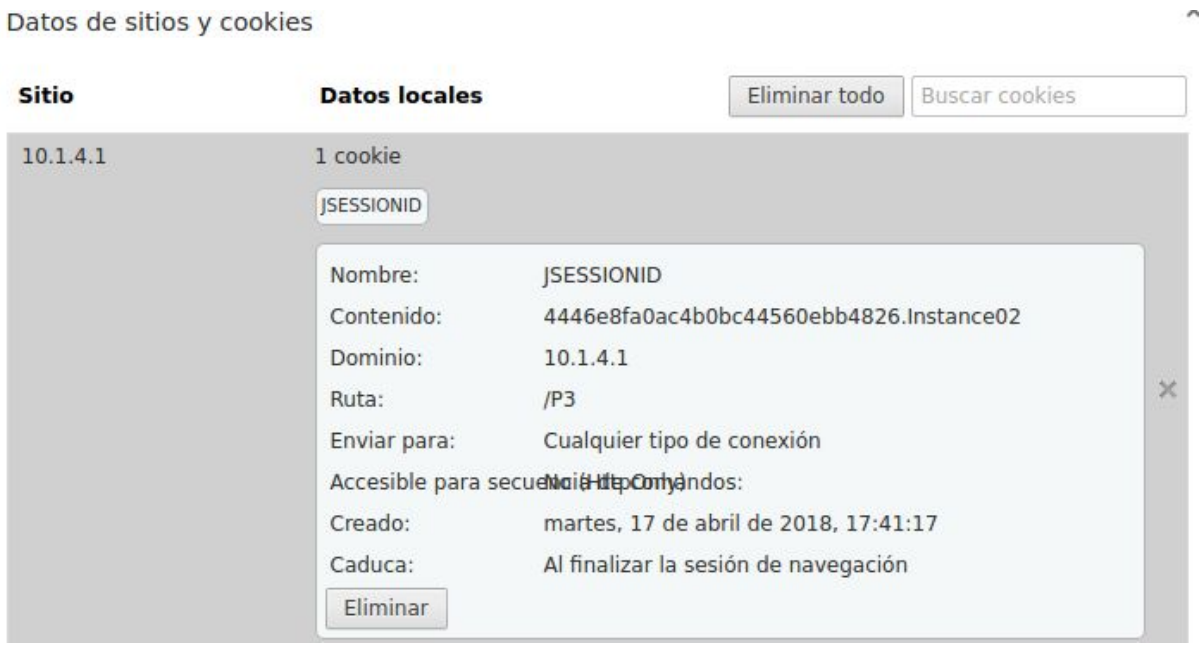
Probar la influencia de jvmRoute en la afinidad de sesión.
1- Eliminar todas las cookies del navegador
2- Sin la propiedad jvmRoute, acceder a la aplicación P3 a través de la URL del balanceador:
http://10.1.4.1/P3
3- Completar el pago con datos de tarjeta correctos.
4- Repetir los pagos hasta que uno falle debido a la falta de afinidad de sesión.
5- Mostrar la cookie "JSESSIONID" correspondiente a la URL del balanceador donde

se vea:
Name: JSESSIONID
Content: YYYYYYYYYYYYYYYYYYY
Domain: 10.X.Y.1
Path: /P3
6- Añadir la propiedad "jvmRoute" al cluster y rearrancar el cluster.
7- Eliminar todas las cookies del nevegador.
8- Acceso a la aplicación P3 a través de la URL del balanceador: http://10.X.Y.1/P3
9- Completar el pago con datos de tarjeta correctos. Se pueden repetir los pagos y no fallarán.
10- Mostrar la cookie "JSESSIONID" correspondiente a la URL del balanceador donde se vea:
Name: JSESSIONID
Content: ZZZZZZZZZZZZZZZZZZZZ
Domain: 10.X.Y.1
Path: /P3
Mostrar las pantallas y comentar: las diferencias en el contenido de las cookie respecto a jvmRoute, y cómo esta diferencia afecta a la afinidad y por qué.

Esta es la cookie que obtenemos sin añadir jvmRoute:



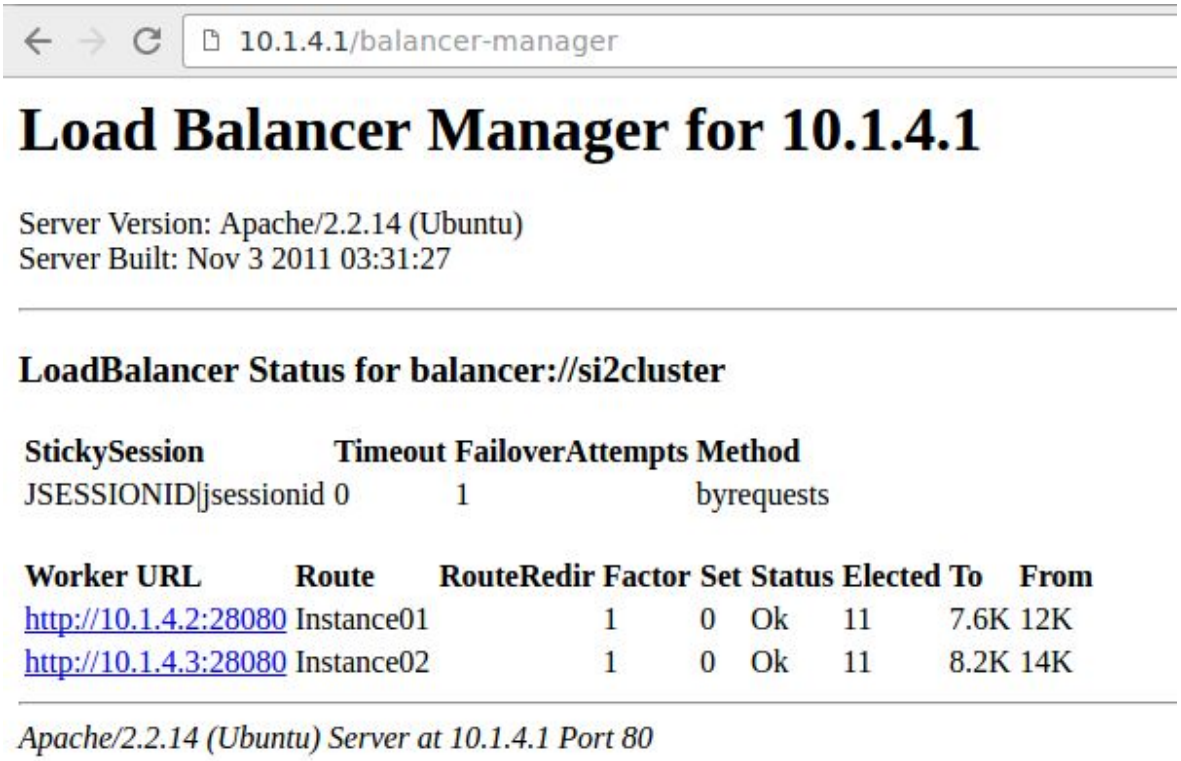
Tras añadir jvmRoute obtenemos la siguiente cookie, donde se comprueba que ahora en el campo "Contenido", aparece el nombre de la instancia, en este caso Instance02.



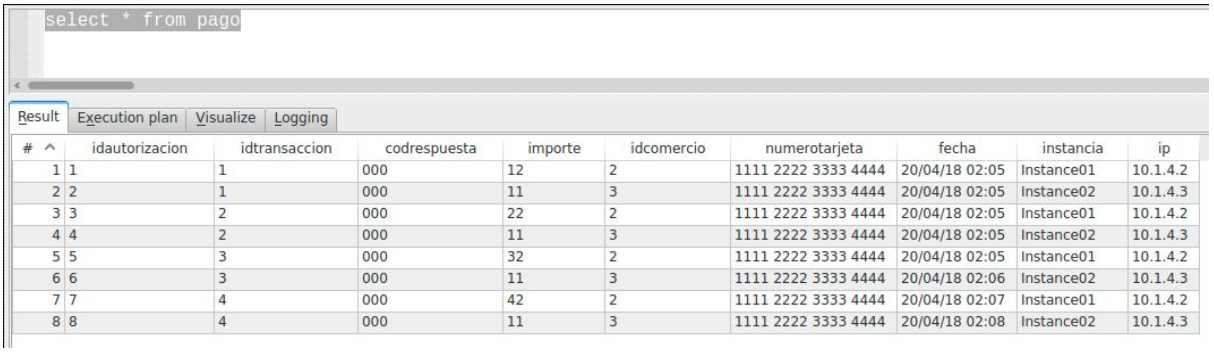
Ejercicio 5:

Probar el balanceo de carga y la afinidad de sesión, realizando un pago directamente contra la dirección del cluster http://10.X.Y.1/P3 desde distintos ordenadores. Comprobar que las peticiones se reparten entre ambos nodos del cluster, y que se mantiene la sesión iniciada por cada usuario sobre el mismo nodo.

Se han realizado 8 pagos en total, 4 desde PC1 y 4 desde PC2. Desde el PC1 se ha escogido id de comercio 2 y desde el PC2 id de comercio 3. A continuación mostramos la ventana de Balancer Manager, donde observamos que en cada instancia hay 11 peticiones:



Desde TOra observamos que el id de comercio 2 ha sido asignado a la instancia 01 y el id de comercio 3 a la instancia 02. Creemos que esto se debe a que el navegador de cada PC tiene guardada una cookie correspondiente a la instancia que primero le asigna el balanceador de carga y a partir de la primera, todas las peticiones serán redirigidas a la misma instancia.



#	idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta	fecha	instancia	ip
1	1	1	000	12	2	1111 2222 3333 4444	20/04/18 02:05	Instance01	10.1.4.2
2	2	1	000	11	3	1111 2222 3333 4444	20/04/18 02:05	Instance02	10.1.4.3
3	3	2	000	22	2	1111 2222 3333 4444	20/04/18 02:05	Instance01	10.1.4.2
4	4	2	000	11	3	1111 2222 3333 4444	20/04/18 02:05	Instance02	10.1.4.3
5	5	3	000	32	2	1111 2222 3333 4444	20/04/18 02:05	Instance01	10.1.4.2
6	6	3	000	11	3	1111 2222 3333 4444	20/04/18 02:06	Instance02	10.1.4.3
7	7	4	000	42	2	1111 2222 3333 4444	20/04/18 02:07	Instance01	10.1.4.2
8	8	4	000	11	3	1111 2222 3333 4444	20/04/18 02:08	Instance02	10.1.4.3

Ejercicio 6:

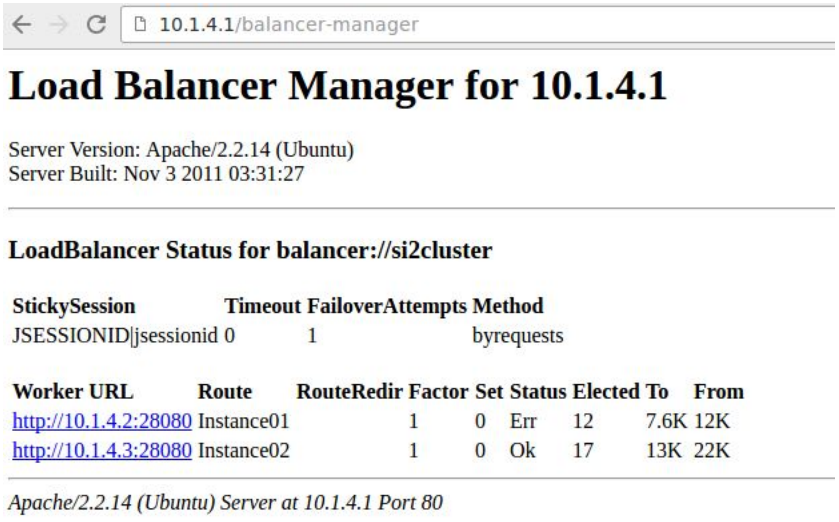
Comprobación del proceso de fail-over. Parar la instancia del cluster que haya tenido menos elecciones hasta el momento. Para ello, identificaremos el pid (identificador del proceso java) de la instancia usando las herramientas descritas en esta práctica o el mandato 'ps –aef | grep java'. Realizaremos un kill -9 pid en el nodo correspondiente. Vuelva a realizar peticiones y compruebe (accediendo a la página /balancer-manager y revisando el contenido de la base de datos) que el anterior nodo ha sido marcado como "erróneo" y que todas las peticiones se dirijan al nuevo servidor. Adjunte la secuencia de comandos y evidencias obtenidas en la memoria de la práctica.

Desde VM2 hemos realizado la siguiente secuencia de comandos:

```
si2@si2srv02:~$ ps -A | grep java
 1319 ?        00:00:44 java
si2@si2srv02:~$ kill -9 1319
```

Tras esto, desde el PC1, al que el balanceador de carga había asociado la instancia 01, realizamos un pago y obtenemos una salida correcta. Consultamos las cookies en el navegador de PC1 y obtenemos que efectivamente ahora tiene asignada la instancia 02:

Desde el balancer manager podemos ver que la Instancia 01 está inactiva, además de que la instancia 02 ahora tiene más peticiones que la 01:

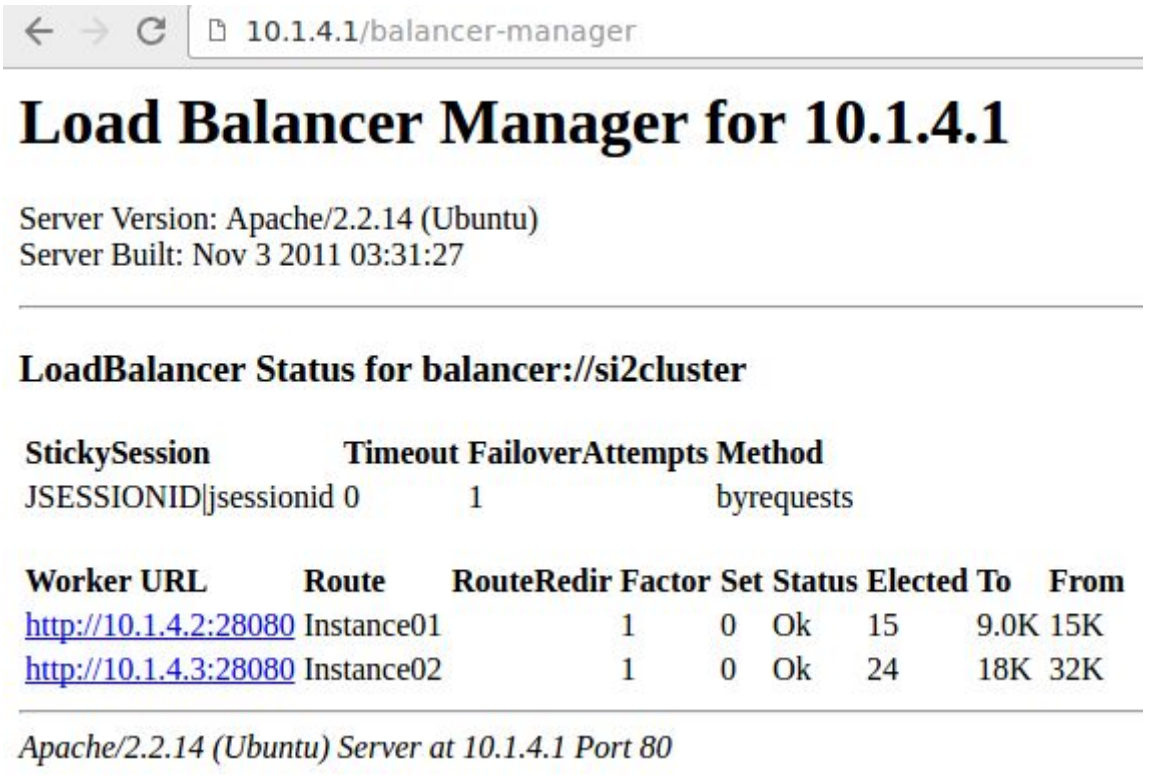


Ejercicio 7:

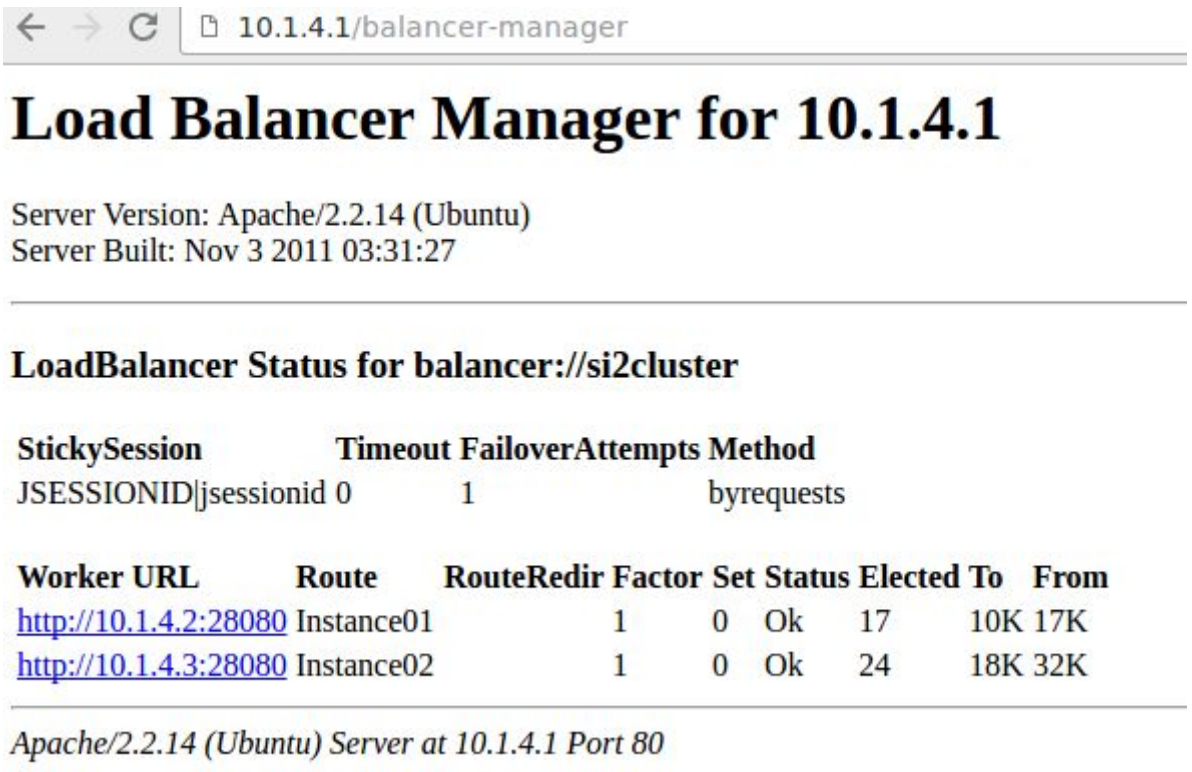
Comprobación del proceso de fail-back. Inicie manualmente la instancia detenida en el comando anterior. Verificar la activación de la instancia en el gestor del balanceador. Incluir todas las evidencias en la memoria de prácticas y comentar qué sucede con los nuevos pagos. Consulte los apéndices para información detallada de comandos de gestión individual de las instancias.

Tras volver a iniciar la instancia y realizar pagos desde el PC1, comprobamos que las

peticiones siguen yendo a la instancia 02, ya que la cookie en el navegador del PC1 sigue manteniendo la instancia del ejercicio anterior, la 02.



Probamos a borrar las cookies del PC1 y consultamos de nuevo el balancer manager:



Observamos como el balanceador de carga asigna las nuevas peticiones a la instancia 01.

Ejercicio 8:

Fallo en el transcurso de una sesión.

- **Desde un navegador, comenzar una petición de pago introduciendo los valores del mismo en la pantalla inicial y realizando la llamada al servlet ComienzaPago.**

- **Al presentarse la pantalla de "Pago con tarjeta", leer la instancia del servidor que ha procesado la petición y detenerla. Se puede encontrar la instancia que ha procesado la petición revisando la cookie de sesión (tiene la instancia como sufijo), el balancer-manager o el server.log de cada instancia.**

  Hemos elegido el navegador del PC1, que está asociado con la instancia 1 como podemos ver en las cookies.

  A continuación detenemos la instancia en la VM2, tras haber completado los campos de Idcomercio, Idtransaccion y el importe:

```
si2@si2srv02:~$ ps -A | grep java
 1918 ?        00:00:32 java
si2@si2srv02:~$ kill -9 1918
```

- **Completar los datos de la tarjeta de modo que el pago fuera válido, y enviar la petición.**

  Tras rellenar los campos y enviar el formulario obtenemos una salida incorrecta:

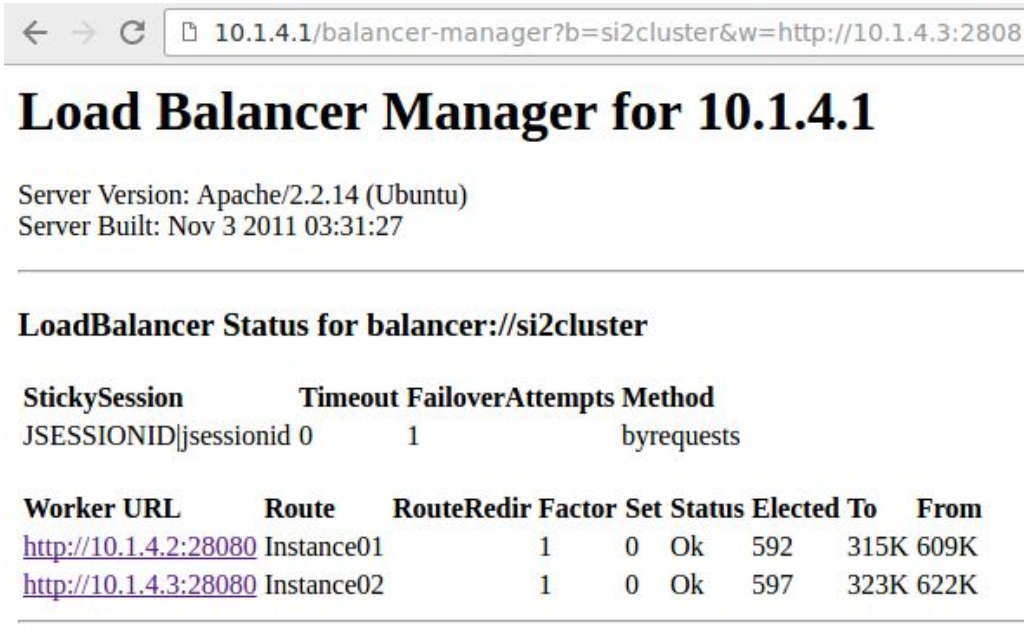
- **Observar la instancia del cluster que procesa el pago, y razonar las causas por las que se rechaza la petición.**

  El pago ha resultado erróneo puesto que la sesión se guardó en la instancia 01 y con ella el id de comercio, id de transacción e importe, pero al cerrar esta instancia, se redirige la petición de procesar pago a la instancia 02 que no tiene esos datos necesarios para guardar el pago en la base de datos y muestra un mensaje erróneo.

Ejercicio 9:

Modificar el script de pruebas JMeter desarrollado durante la P2. (P2.jmx) Habilitar un ciclo de 1000 pruebas en un solo hilo contra la IP del cluster y nueva URL de la aplicación: http://10.X.Y.1/P3
Eliminar posibles pagos previos al ciclo de pruebas. Verificar el porcentaje de pagos realizados por cada instancia, así como (posibles) pagos correctos e incorrectos. ¿Qué algoritmo de reparto parece haber seguido el balanceador? Comente todas sus conclusiones en la memoria de prácticas.

Lanzando el P3.jmx con unas visitas previas de 92 para Intance01 y 97 para instance02 vemos como el ressultado final es de 592 y 597 lo que supone un reparto equitativo.



El algoritmo utilizado es Request Counting Algorithm, el asignado por defecto por Apache.