MEMORIA PRÁCTICA 3

Seguridad y disponibilidad

Alejandro Santorum Varela - alejandro.santorum@estudiante.uam.es
Rafael Sánchez Sánchez - rafael.sanchezs@estudiante.uam.es

Prácticas Sistemas Informáticos II
Práctica 3 Pareja 7 Grupo 2401

30 de abril de 2020

Contents

I Introducción

Finalmente llegamos a la última práctica del curso de Sistemas informáticos II. En esta práctica tendremos como objetivo mostrar la utilización de grupos de servidores (clusters) como herramienta para aumentar la disponibilidad de las aplicaciones distribuidas.

Se emplearán las posibilidades que permite la arquitectura J2EE, estudiando los siguientes aspectos: aspectos generales de clustering, configuración de un cluster, implementación de un cluster seguro y pruebas de disponibilidad de una aplicación J2EE.

1 Ejercicio 1

Enunciado:

Ejercicio 1: Preparar 3 máquinas virtuales desde cero (a partir de la VM en moodle) 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.

Revisar y comentar la salida del mandato ssh.

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.

Respuesta a la cuestión:
En la primera máquina virtual (10.1.7.1) generamos el par de claves DSA con el comando `ssh-keygen -t dsa` obteniendo el fichero `.ssh/id_dsa.pub`. Importamos la clave pública en el resto de máquinas virtuales y probamos a acceder por SSH desde 1 a 2 y 3 (comprobando que no requiere la introducción de la clave). Ejecutamos el comando `ssh -v si2@10.1.7.2` produciendo la siguiente salida.

Salida comando ssh -v si2@10.1.7.2

```
1  OpenSSH_5.3p1 Debian-3ubuntu7, OpenSSL 0.9.8k 25 Mar 2009
2  debug1: Reading configuration data /etc/ssh/ssh_config
3  debug1: Applying options for *
4  debug1: Connecting to 10.1.7.2 [10.1.7.2] port 22.
5  debug1: Connection established.
6  debug1: identity file /home/si2/.ssh/identity type -1
7  debug1: identity file /home/si2/.ssh/id_rsa type -1
8  debug1: identity file /home/si2/.ssh/id_dsa type 2
9  debug1: Checking blacklist file /usr/share/ssh/blacklist.DSA-1024
10 debug1: Checking blacklist file /etc/ssh/blacklist.DSA-1024
11 debug1: Remote protocol version 2.0, remote software version OpenSSH_5.3p1
       Debian-3ubuntu7
12 debug1: match: OpenSSH_5.3p1 Debian-3ubuntu7 pat OpenSSH*
13 debug1: Enabling compatibility mode for protocol 2.0
14 debug1: Local version string SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7
15 debug1: SSH2_MSG_KEXINIT sent
16 debug1: SSH2_MSG_KEXINIT received
17 debug1: kex: server->client aes128-ctr hmac-md5 none
18 debug1: kex: client->server aes128-ctr hmac-md5 none
19 debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
20 debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
21 debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
22 debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
```

```
23 debug1: Host '10.1.7.2' is known and matches the RSA host key.
24 debug1: Found key in /home/si2/.ssh/known_hosts:1
25 debug1: ssh_rsa_verify: signature correct
26 debug1: SSH2_MSG_NEWKEYS sent
27 debug1: expecting SSH2_MSG_NEWKEYS
28 debug1: SSH2_MSG_NEWKEYS received
29 debug1: SSH2_MSG_SERVICE_REQUEST sent
30 debug1: SSH2_MSG_SERVICE_ACCEPT received
31 debug1: Authentications that can continue: publickey,password
32 debug1: Next authentication method: publickey
33 debug1: Trying private key: /home/si2/.ssh/identity
34 debug1: Trying private key: /home/si2/.ssh/id_rsa
35 debug1: Offering public key: /home/si2/.ssh/id_dsa
36 debug1: Server accepts key: pkalg ssh-dss blen 434
37 debug1: read PEM private key done: type DSA
38 debug1: Authentication succeeded (publickey).
39 debug1: channel 0: new [client-session]
40 debug1: Requesting no-more-sessions@openssh.com
41 debug1: Entering interactive session.
42 debug1: Sending environment.
43 debug1: Sending env LANG = C
44 Linux si2srv02 2.6.32-33-generic #72-Ubuntu SMP Fri Jul 29 21:08:37 UTC
       2011 i686 GNU/Linux
45 Ubuntu 10.04.3 LTS
46
47 Welcome to Ubuntu!
48  * Documentation:  https://help.ubuntu.com/
49 New release 'precise' available.
50 Run 'do-release-upgrade' to upgrade to it.
51
52 Last login: Thu Apr 30 09:41:21 2020 from 10.1.7.1
53
54 Loading es
55 si2@si2srv02:~$ exitdebug1: client_input_channel_req: channel 0 rtype exit
       -status reply 0
56 debug1: client_input_channel_req: channel 0 rtype eow@openssh.com reply 0
57
58 logout
59 debug1: channel 0: free: client-session, nchannels 1
60 debug1: fd 1 clearing O_NONBLOCK
61 Connection to 10.1.7.2 closed.
62 Transferred: sent 2848, received 3016 bytes, in 9.6 seconds
63 Bytes per second: sent 297.4, received 315.0
64 debug1: Exit status 0
```

Las primeras 3 líneas indican la versión de SSH utilizada, la ruta del fichero de configuración de SSH y las opciones de inicialización de SSH. Las siguientes 11 líneas (de la línea 4 a la 14) exponen la dirección IP con la que se está intentando realizar la conexión, la ruta de los ficheros de claves DSA y RSA, y el proceso por el cual se comprueba que la versión de SSH utilizada por ambas maquinas es compatible. Entre las líneas 15 y 43 se realiza el *handshake* de SSH por el cual se realizan todos los acuerdos de algoritmos de cifrado a utilizar (RSA, AES128, DH, firma digital, etc). Finalmente el resto de líneas confirma la correcta conexión.

Por otro lado ejecutamos el comando `ssh -v si2@10.1.7.3` para realizar lo mismo entre la MV1 y la MV3, obteniendo una salida prácticamente idéntica:

Salida comando ssh -v si2@10.1.7.3

```
1 OpenSSH_5.3p1 Debian-3ubuntu7, OpenSSL 0.9.8k 25 Mar 2009
2 debug1: Reading configuration data /etc/ssh/ssh_config
3 debug1: Applying options for *
4 debug1: Connecting to 10.1.7.3 [10.1.7.3] port 22.
5 debug1: Connection established.
6 debug1: identity file /home/si2/.ssh/identity type -1
7 debug1: identity file /home/si2/.ssh/id_rsa type -1
8 debug1: identity file /home/si2/.ssh/id_dsa type 2
9 debug1: Checking blacklist file /usr/share/ssh/blacklist.DSA-1024
10 debug1: Checking blacklist file /etc/ssh/blacklist.DSA-1024
11 debug1: Remote protocol version 2.0, remote software version OpenSSH_5.3p1
       Debian-3ubuntu7
12 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 '10.1.7.3' 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
Linux si2srv03 2.6.32-33-generic #72-Ubuntu SMP Fri Jul 29 21:08:37 UTC
    2011 i686 GNU/Linux
Ubuntu 10.04.3 LTS

Welcome to Ubuntu!
 * Documentation:  https://help.ubuntu.com/
New release 'precise' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Thu Apr 30 09:39:20 2020 from 10.1.7.1

Loading es
si2@si2srv03:~$ exitdebug1: client_input_channel_req: channel 0 rtype exit
    -status reply 0
debug1: client_input_channel_req: channel 0 rtype eow@openssh.com reply 0

logout
debug1: channel 0: free: client-session, nchannels 1
debug1: fd 1 clearing O_NONBLOCK
Connection to 10.1.7.3 closed.
Transferred: sent 2848, received 3016 bytes, in 8.0 seconds
Bytes per second: sent 355.0, received 376.0
debug1: Exit status 0
```

La explicación de dicho resultado es análogo al caso de la MV1-MV2.

Terminamos este ejercicio siguiendo las recomendaciones del enunciado y guardando una copia de las tres máquinas virtuales en un dispositivo auxiliar.

2 Ejercicio 2

Enunciado:
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.

Respuesta a la cuestión:
Siguiendo los pasos del apartado 4 obtenemos una configuración válida del cluster SI2Cluster y comprobamos que las instancias del cluster y los pids de los procesos Java correspondientes están efectivamente corriendo con el comando `asadmin --user admin --passwordfile /opt/SI2/passwordfile list-instances-1`:

Verificación instancias activas

```
si2@si2srv01:~$ asadmin --user admin --passwordfile /opt/SI2/passwordfile
    list-instances -l
Name         Host       Port    Pid   Cluster     State
Instance01   10.1.7.2   24848   1953  SI2Cluster   running
Instance02   10.1.7.3   24848   1955  SI2Cluster   running
Command list-instances executed successfully.
```

Como vemos en el extracto de terminal mostrado arriba, ambas instancias están corriendo satisfactoriamente.

Para evidenciar que el cluster está configurado con una topología de 1 DAS y 2 nodos SSH, mostramos la salida del comando `ps -aef : grep java : less` para cada una de las instancias:

Topología nodo instancia 1

```
0 S si2       1953     1 15  80   0 - 136216 futex_ 10:06 ?       00:00:24
     /usr/lib/jvm/java-8-oracle/bin/java -cp /opt/glassfish4/glassfish/
    modules/glassfish.jar -XX:+UnlockDiagnosticVMOptions -XX:NewRatio=2 -XX
    :MaxPermSize=96m -Xmx128m -Xms128m -server -javaagent:/opt/glassfish4/
    glassfish/lib/monitor/flashlight-agent.jar -Djavax.net.ssl.trustStore=/
    opt/glassfish4/Node01/Instance01/config/cacerts.jks -Djdk.corba.
    allowOutputStreamSubclass=true -Dfelix.fileinstall.dir=/opt/glassfish4/
    glassfish/modules/autostart/ -Dorg.glassfish.
    additionalOSGiBundlesToStart=org.apache.felix.shell,org.apache.felix.
    gogo.runtime,org.apache.felix.gogo.shell,org.apache.felix.gogo.command,
    org.apache.felix.fileinstall -Dcom.sun.aas.installRoot=/opt/glassfish4/
    glassfish -Dfelix.fileinstall.poll=5000 -Djava.security.policy=/opt/
    glassfish4/Node01/Instance01/config/server.policy -Djava.endorsed.dirs
    =/opt/glassfish4/glassfish/modules/endorsed:/opt/glassfish4/glassfish/
    lib/endorsed -Dfelix.fileinstall.bundles.startTransient=true -Dosgi.
    shell.telnet.maxconn=1 -Dfelix.fileinstall.log.level=3 -Dcom.sun.
    enterprise.config.config_environment_factory_class=com.sun.enterprise.
    config.serverbeans.AppserverConfigEnvironmentFactory -Djavax.net.ssl.
    keyStore=/opt/glassfish4/Node01/Instance01/config/keystore.jks -Djava.
    security.auth.login.config=/opt/glassfish4/Node01/Instance01/config/
    login.conf -Dfelix.fileinstall.disableConfigSave=false -Dfelix.
    fileinstall.bundles.new.start=true -Dcom.sun.aas.instanceRoot=/opt/
    glassfish4/Node01/Instance01 -Dosgi.shell.telnet.port=26666 -Dgosh.args
    =--noshutdown -c noop=true -Dcom.sun.enterprise.security.
    httpsOutboundKeyAlias=s1as -Dosgi.shell.telnet.ip=127.0.0.1 -
    DANTLR_USE_DIRECT_CLASS_LOADING=true -Djava.awt.headless=true -Djava.
    ext.dirs=/usr/lib/jvm/java-8-oracle/lib/ext:/usr/lib/jvm/java-8-oracle/
    jre/lib/ext:/opt/glassfish4/Node01/Instance01/lib/ext -Djdbc.drivers=
    org.apache.derby.jdbc.ClientDriver -Djava.library.path=/opt/glassfish4/
    glassfish/lib:/usr/java/packages/lib/i386:/lib:/usr/lib com.sun.
    enterprise.glassfish.bootstrap.ASMain -upgrade false -read-stdin true -
    asadmin-args --host,,,si2srv01,,,--port,,,4848,,,--secure=false,,,--
    terse=false,,,--echo=false,,,--interactive=false,,,start-local-instance
    ,,,--verbose=false,,,--watchdog=false,,,--debug=false,,,--nodedir,,,/
    opt/glassfish4,,,--node,,,Node01,,,Instance01 -instancename Instance01
    -type INSTANCE -verbose false -instancedir /opt/glassfish4/Node01/
    Instance01 -asadmin-classpath /opt/glassfish4/glassfish/modules/admin-
```

```
cli.jar -debug false -asadmin-classname com.sun.enterprise.admin.cli.
AdminMain
```

Topología nodo instancia 2

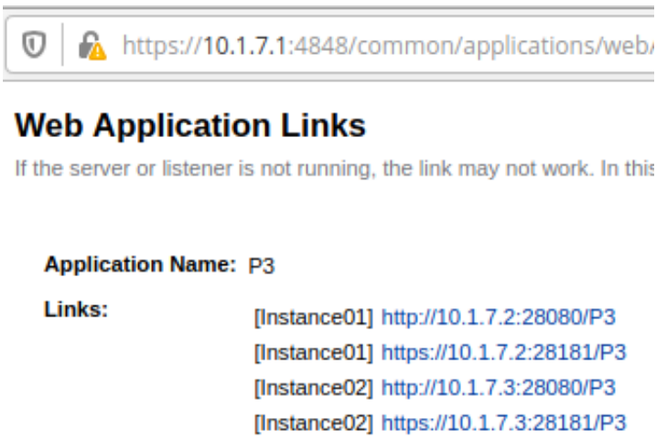
```
1 0 S si2       1955     1  7  80   0 - 134910 futex_ 10:06 ?       00:00:21
  /usr/lib/jvm/java-8-oracle/bin/java -cp /opt/glassfish4/glassfish/
  modules/glassfish.jar -XX:+UnlockDiagnosticVMOptions -XX:NewRatio=2 -XX
  :MaxPermSize=96m -Xmx128m -Xms128m -server -javaagent:/opt/glassfish4/
  glassfish/lib/monitor/flashlight-agent.jar -Djavax.net.ssl.trustStore=/
  opt/glassfish4/Node02/Instance02/config/cacerts.jks -Djdk.corba.
  allowOutputStreamSubclass=true -Dfelix.fileinstall.dir=/opt/glassfish4/
  glassfish/modules/autostart/ -Dorg.glassfish.
  additionalOSGiBundlesToStart=org.apache.felix.shell,org.apache.felix.
  gogo.runtime,org.apache.felix.gogo.shell,org.apache.felix.gogo.command,
  org.apache.felix.fileinstall -Dcom.sun.aas.installRoot=/opt/glassfish4/
  glassfish -Dfelix.fileinstall.poll=5000 -Djava.security.policy=/opt/
  glassfish4/Node02/Instance02/config/server.policy -Djava.endorsed.dirs
  =/opt/glassfish4/glassfish/modules/endorsed:/opt/glassfish4/glassfish/
  lib/endorsed -Dfelix.fileinstall.bundles.startTransient=true -Dosgi.
  shell.telnet.maxconn=1 -Dfelix.fileinstall.log.level=3 -Dcom.sun.
  enterprise.config.config_environment_factory_class=com.sun.enterprise.
  config.serverbeans.AppserverConfigEnvironmentFactory -Djavax.net.ssl.
  keyStore=/opt/glassfish4/Node02/Instance02/config/keystore.jks -Djava.
  security.auth.login.config=/opt/glassfish4/Node02/Instance02/config/
  login.conf -Dfelix.fileinstall.disableConfigSave=false -Dfelix.
  fileinstall.bundles.new.start=true -Dcom.sun.aas.instanceRoot=/opt/
  glassfish4/Node02/Instance02 -Dosgi.shell.telnet.port=26666 -Dgosh.args
  =--noshutdown -c noop=true -Dcom.sun.enterprise.security.
  httpsOutboundKeyAlias=s1as -Dosgi.shell.telnet.ip=127.0.0.1 -
  DANTLR_USE_DIRECT_CLASS_LOADING=true -Djava.awt.headless=true -Djava.
  ext.dirs=/usr/lib/jvm/java-8-oracle/lib/ext:/usr/lib/jvm/java-8-oracle/
  jre/lib/ext:/opt/glassfish4/Node02/Instance02/lib/ext -Djdbc.drivers=
  org.apache.derby.jdbc.ClientDriver -Djava.library.path=/opt/glassfish4/
  glassfish/lib:/usr/java/packages/lib/i386:/lib:/usr/lib com.sun.
  enterprise.glassfish.bootstrap.ASMain -upgrade false -read-stdin true -
  asadmin-args --host,,,si2srv01,,,--port,,,4848,,,--secure=false,,,--
  terse=false,,,--echo=false,,,--interactive=false,,,start-local-instance
  ,,,--verbose=false,,,--watchdog=false,,,--debug=false,,,--nodedir,,,/
  opt/glassfish4,,,--node,,,Node02,,,Instance02 -instancename Instance02
  -type INSTANCE -verbose false -instancedir /opt/glassfish4/Node02/
  Instance02 -asadmin-classpath /opt/glassfish4/glassfish/modules/admin-
  cli.jar -debug false -asadmin-classname com.sun.enterprise.admin.cli.
  AdminMain
```

Observando sobre todo en las primeras 5 líneas podemos ver la configuración del cluster indicadas en el apartado 4: **-XX:MaxPermSize=96m -Xmx128m -Xms128m -server**

3 Ejercicio 3

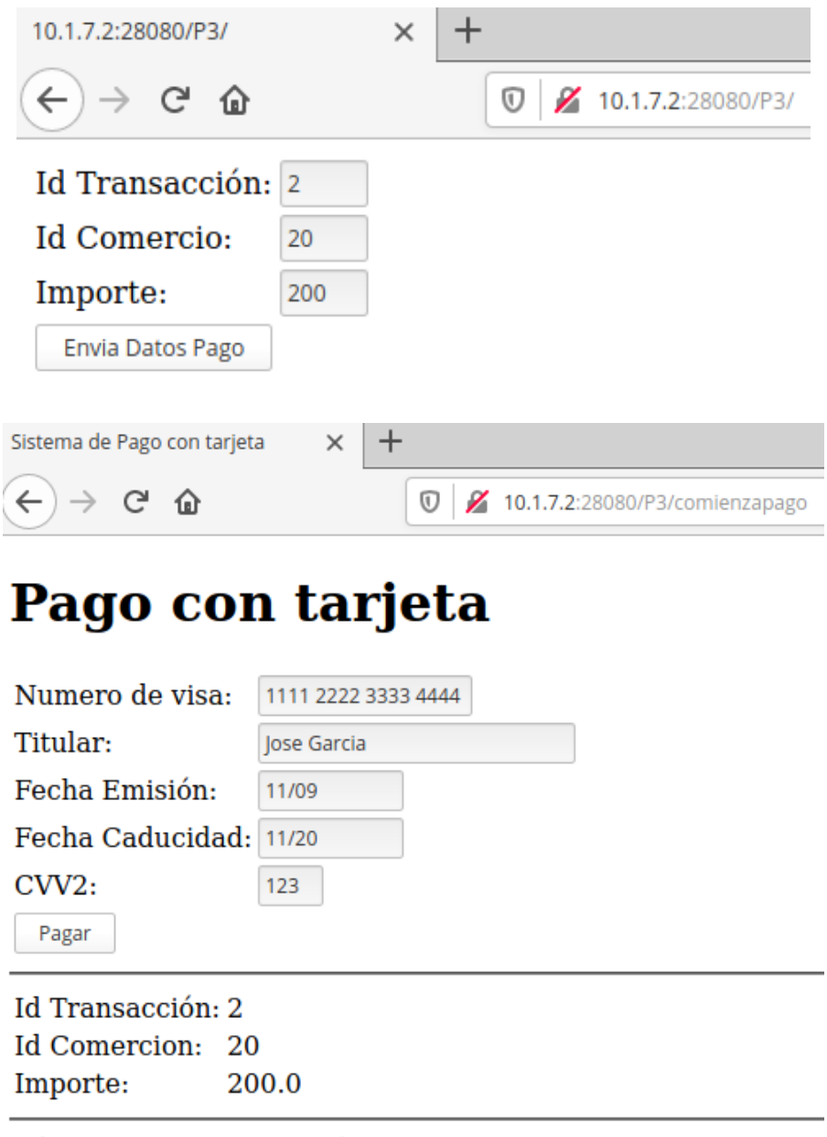
Enunciado:
Pruebe a realizar un pago individualmente en cada instancia. Para ello, identifique los puertos en los que están siendo ejecutados cada una de las dos instancias (IPs 10.X.Y.2 y 10.X.Y.3 respectivamente). Puede realizar esa comprobación directamente desde la consola de administración, opción 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).

Respuesta a la cuestión:
Antes de probar a realizar cualquier pago, identificamos los puertos en los que están siendos ejecutadas cada una de las instancias (10.1.7.2 y 10.7.1.3) usando la consola de administración observando los *Web Application Links*:



Vemos que hay 4 links desplegados, dos para cada instancia (uno para el protocolo HTTP y otro para el HTTPS). Nosotros haremos los pagos en los links de HTTP de cada una de las instancias.

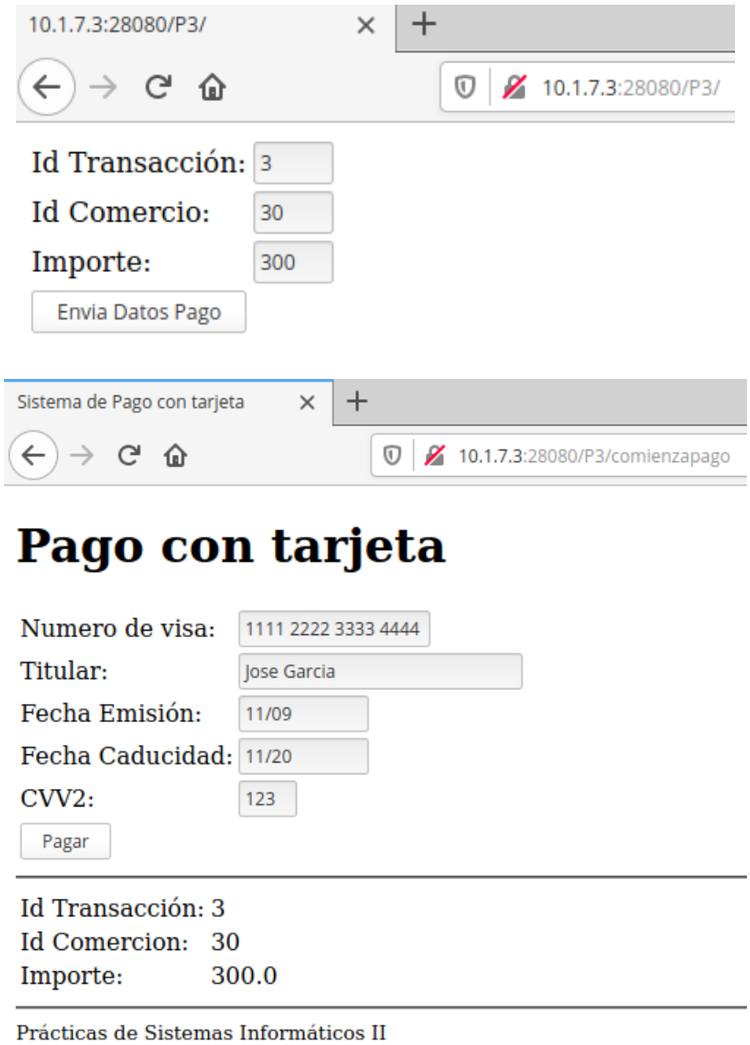
La instancia 1 se corresponde con la desplegada en la máquina virtual número 2. Será la primera en la que intentaremos realizar un pago, tal y como se muestra a continuación (comprobar link de la barra de búsqueda del navegador):



Pinchando en 'Pagar' obtenemos el siguiente mensaje de confirmación:

Todo indica que el pago ha sido realizado exitosamente en la instancia número 1 (MV2).

Realizamos el mismo proceso, pero ahora en la instancia número 2 (comprobar el cambio en el link del navegador):



Pinchando en 'Pagar' obtenemos el siguiente mensaje de confirmación:

Todo ha vuelto a ir bien en el pago de la instancia 2 (MV3).

Finalmente podemos comprobar en la base de datos que ambos pagos se han realizado satisfactoriamente. Obsérvese en la imagen de abajo los dos pagos realizados, uno en cada instancia, con la IP correspondiente.

```
visa=# select * from pago;
 idautorizacion |  idtransaccion   | codrespuesta | importe |    idcomercio    |    numerotarjeta    |          fecha             | instancia |    ip
----------------+------------------+--------------+---------+------------------+---------------------+----------------------------+-----------+----------
              1 | 2                | 000          |     200 | 20               | 1111 2222 3333 4444 | 2020-04-30 10:54:17.893206 | Instance01 | 10.1.7.2
              2 | 3                | 000          |     300 | 30               | 1111 2222 3333 4444 | 2020-04-30 10:56:10.653586 | Instance02 | 10.1.7.3
(2 rows)
```

4 Ejercicio 4

Enunciado:

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.X.Y.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:	YYYYYYYYYYYYYYYYYY
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:	ZZZZZZZZZZZZZZZZZZZZZ
Domain:	10.X.Y.1
Path:	/P3

Mostrar las pantallas y comentar: las diferencias en el contenido de las cookies respecto a jvmRoute, y cómo esta diferencia afecta a la afinidad y por qué.

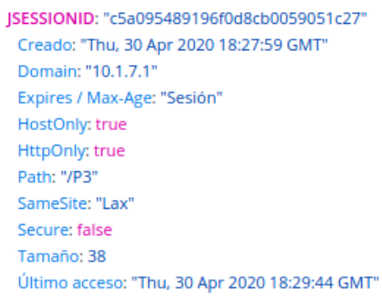
¿Se podría, en general, usar el valor `${com.sun.aas.hostName}` para la propiedad `jvmRoute`, en lugar de `${com.sun.aas.instanceName}`?

Respuesta a la cuestión:
Comenzamos eliminando las *cookies* tanto de Mozilla Firefox en la ruta `Preferencias - Privacidad - Mostrar cookies` y ahora, **sin la propiedadad jvmRoute**, accedemos a la aplicación P3 a través de la URL del balanceador (usamos 10.1.7.1/P3). Rellenamos el formulario con los datos de una tarjeta correcta, tal y como se muestra a continuación:

Realizamos pagos hasta que uno falle por falta de afinidad de sesión:

Después de unos pagos:

En este momento buscamos la *cookie* **JSESSIONID** correspondiente a la URL del balanceador, la cual se muestra a continuación:



Como se puede ver en la imagen, aún no ha sido añadida la propiedad **jvmRoute** (no aparece el nombre de la instancia en el ID).

Ahora es el momento de añadir la **propiedad jvmRoute** al cluster y rearrancar el mismo. Después eliminamos de nuevo todas las *cookies* y accedemos a la aplicacion P3 a traves de la URL del balanceador, completando el pago con datos de tarjeta correctos. Se pueden repetir los pagos y no fallarán.

A continuación mostramos la *cookie* **JSESSIONID** modificada con respecto a la versión anterior (ahora sí aparece el nombre de la instancia en el ID de la *cookie*):

Es fácil ver que la diferencia principal entre ambas *cookies* es el nombre de la instancia que procesa la petición. Parece una diferencia mínima, pero gracias a esto se puede **mantener la afinidad de la sesion en el balanceador de carga** ya que este último tiene la capacidad de saber qué instancia procesa cada petición y así balancear la carga correctamente.

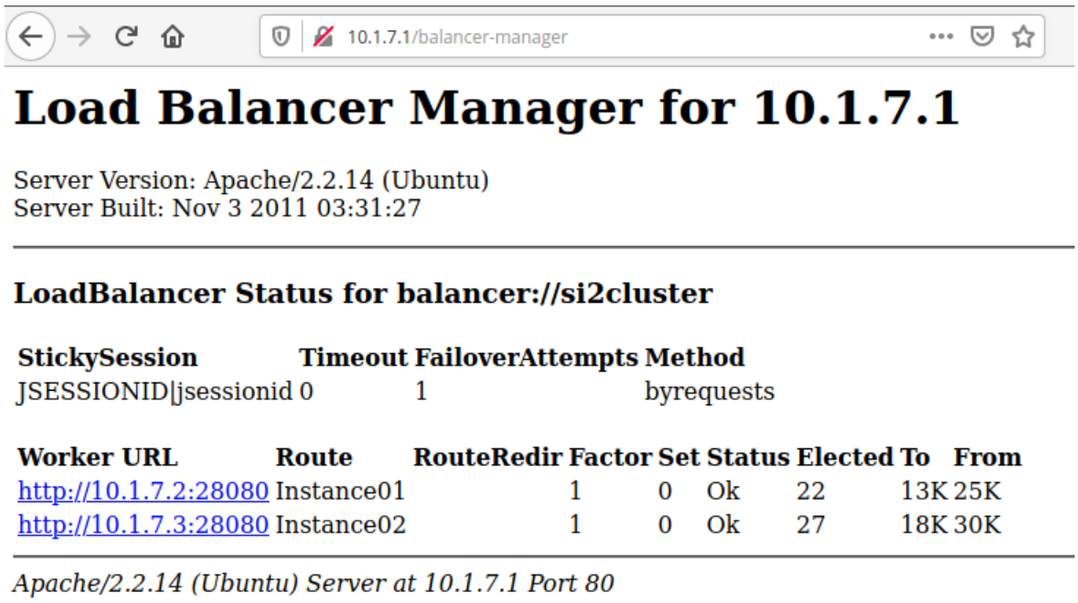
Por último, se cuestiona la posibilidad de usar el valor `${com.sun.aas.hostName}` para la propiedad jvmRoute, en lugar de `${com.sun.aas.instanceName}`. Creemos que la respuesta es **no**, ya que el nombre del *host* es común para ambas instancias y, por lo tanto, **no** se podrá **mantener la afinidad de la sesio**n en el balanceador de carga, al **no** existir informacion de la **instancia que ha procesad**o cada petición.

5 Ejercicio 5

Enunciado:
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.
Comentad la información mostrada en la página del Load Balancer Manager

Respuesta a la cuestión:
Debido a las circunstancias actuales, no disponemos de varios ordenadores con los que realizar varios pagos simultáneos. Para intentar solucionar esto, realizaremos varios pagos desde el mismo ordenador pero desde navegadores diferentes. Para ello añadiremos la propiedad **jvmRoute** tanto en Mozilla Firefox como en Google Chrome y eliminaremos las *cookies* de ambos navegadores antes de realizar cualquier pago.

Una vez hecho esto, realizamos varios pagos "simultáneos" (aquí simultáneo quiere decir que pinchamos el botón 'Pagar' en un navegador y, con la mayor rapidez posible, realizamos el pago desde el otro navegador). Realizamos varios pagos con este procedimiento, 49 para ser exactos, y los resultados obtenidos en el **Load Balancer Manager** son los siguientes:



Como se puede observar, el balanceo de carga no es perfecto ya que se desearía idealmente que una instancia procesase 25 y la otra 24, pero el resultado es bastante decente. **Una instancia ha procesado 22 peticiones y la otra 27 y cada usuario mantiene la sesión iniciada sobre el mismo nodo**. Esto puede deberse a la política de reparto interna del balanceador (por ejemplo, si se reparten de forma aleatoria con un 50% de probabilidad cada una, el resultado obtenido es bastante común) o incluso que la carga del sistema no ha sido lo suficientemente grande como para exigir al balanceador un gran esfuerzo en el reparto. En cualquier caso, queda probado que las peticiones son repartidas entre ambos nodos del sistema.

6 Ejercicio 6

Enunciado:
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.

Respuesta a la cuestión:
Utilizando el comando mencionado en el enunciado (que es el mismo que hemos usado en el ejercicio 2) podemos determinar el PID de la instancia 1, que como bien hemos visto en el ejercicio 5 es la que menos peticiones ha procesado hasta el momento.

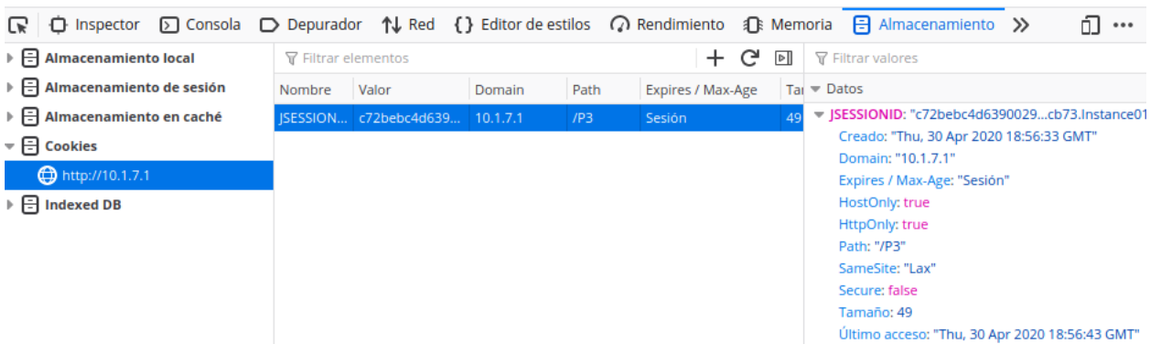
```
ADING=true -Djava.awt.headless=true -Djava.ext.dirs=/usr/lib/jvm/java-8-oracle/l
ib/ext:/usr/lib/jvm/java-8-oracle/jre/lib/ext:/opt/glassfish4/Node01/Instance01/
lib/ext -Djdbc.drivers=org.apache.derby.jdbc.ClientDriver -Djava.library.path=/o
pt/glassfish4/glassfish/lib:/usr/java/packages/lib/i386:/lib:/usr/lib com.sun.en
terprise.glassfish.bootstrap.ASMain -upgrade false -read-stdin true -asadmin-arg
s --host,,,si2srv01,,,--port,,,4848,,,--secure=false,,,--terse=false,,,--echo=fa
lse,,,--interactive=false,,,start-local-instance,,,--verbose=false,,,--watchdog=
false,,,--debug=false,,,--nodedir,,,/opt/glassfish4,,,--node,,,Node01,,,Instance
01 -instancename Instance01 -type INSTANCE -verbose false -instancedir /opt/glas
si2@si2srv02:~$ kill -9 2891
si2@si2srv02:~$ ps -aef | grep java | less
si2       3163  3143  0 11:59 tty1     00:00:00 grep java
si2@si2srv02:~$ _
```

En la imagen anterior se ha matado al proceso que ejecutaba la instancia número 1. En la misma imagen se puede comprobar el éxito del comando **kill** cuando se usa el comando **ps -aef : grep java** y el resultado es nulo. Adicionalmente, podemos comprobar que ahora la instancia 1 está muerta con el comando **asadmin list-instances -l** y fijándonos en la columna *state* vemos que el estado de la instancia 1 es **not running**.

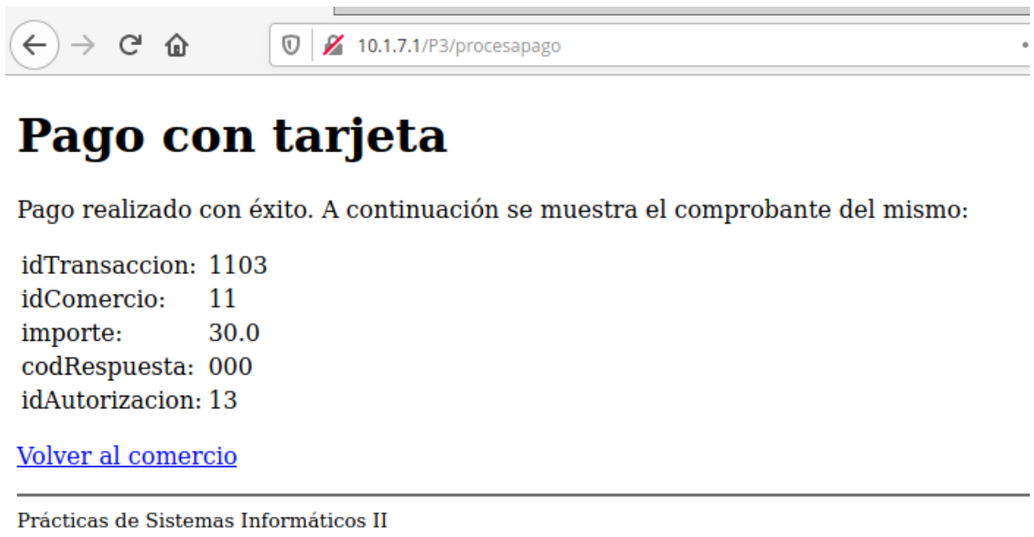
```
si2@si2srv01:~$ asadmin list-instances -l
Name        Host      Port   Pid   Cluster     State
Instance01  10.1.7.2  24848  --    SI2Cluster   not running
Instance02  10.1.7.3  24848  2904  SI2Cluster   running
Command list-instances executed successfully.
```

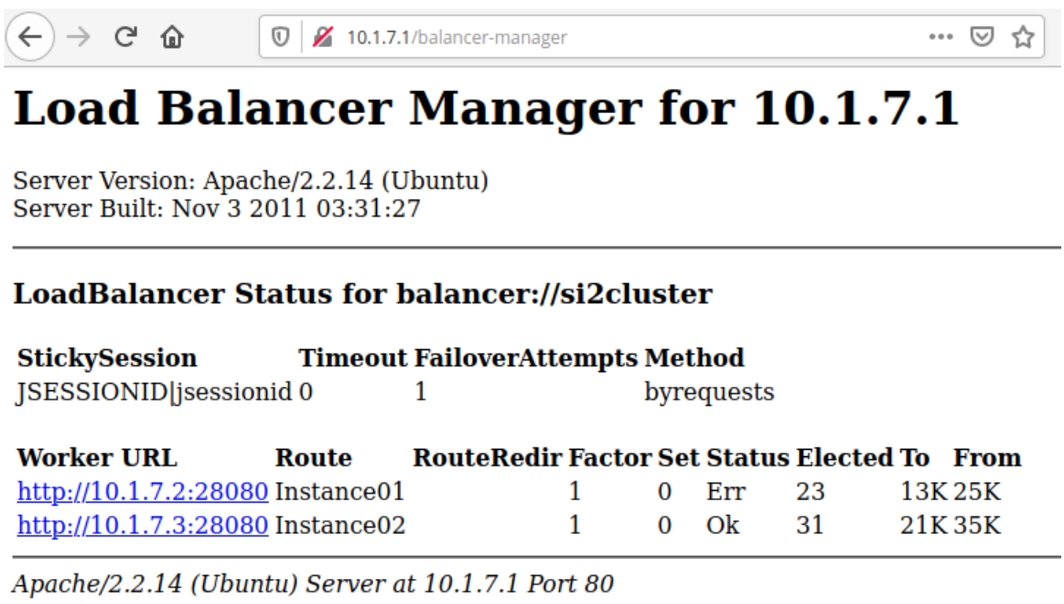
Ahora abrimos el navegador que estaba enlazado al nodo de la instancia 1 (compruébese la *cookie* mostrada en la imagen) y realizamos un pago:

Id Transacción:
Id Comercio:
Importe:
Envia Datos Pago



Comprobamos que el pago se realiza exitosamente siendo rederigido al nodo 2:





En especial, en esta última imagen podemos ver que el número de peticiones procesadas por la instancia 2 ha aumentado, a pesar de usar el cliente con la *cookie* de la instancia 1.

7 Ejercicio 7

Enunciado:
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**.

Respuesta a la cuestión:
En el ejercicio 6 teníamos un cliente con la *cookie* de la instancia 1, la cual fue eliminada y ese cliente pasó a estar conectado a la instancia 2. En este ejercicio veremos que le pasa a ese cliente si rearrancamos la instancia 1 de nuevo.

A continuación mostramos la salida obtenida después de rearrancar la instancia 1, así como el balanceador de carga después del arranque pero antes de realizar cualquier pago:

```
si2@si2srv01:~$ asadmin start-instance Instance01
Waiting for Instance01 to start .....
Successfully started the instance: Instance01
instance Location: /opt/glassfish4/Node01/Instance01
Log File: /opt/glassfish4/Node01/Instance01/logs/server.log
Admin Port: 24848
Command start-local-instance executed successfully.
The instance, Instance01, was started on host 10.1.7.2
Command start-instance executed successfully.
si2@si2srv01:~$ asadmin list-instances -l
Name        Host      Port   Pid   Cluster     State
Instance01  10.1.7.2  24848  3258  SI2Cluster   running
Instance02  10.1.7.3  24848  2904  SI2Cluster   running
Command list-instances executed successfully.
```

Load Balancer Manager for 10.1.7.1

Server Version: Apache/2.2.14 (Ubuntu)
Server Built: Nov 3 2011 03:31:27

LoadBalancer Status for balancer://si2cluster

StickySession	Timeout	FailoverAttempts	Method
JSESSIONID|jsessionid	0	1	byrequests

Worker URL	Route	RouteRedir	Factor	Set	Status	Elected	To	From
http://10.1.7.2:28080	Instance01		1	0	Ok	23	13K	25K
http://10.1.7.3:28080	Instance02		1	0	Ok	31	21K	35K

Apache/2.2.14 (Ubuntu) Server at 10.1.7.1 Port 80

En la primera imagen se puede ver el comando utilizado para el rearranque, así como la lista actualizada del estado de las instancias, donde la instancia 1 ya está corriendo satisfactoriamente. En la segunda imagen, podemos ver que se han elegido 23 peticiones para la instancia 1 y 31 peticiones para la instancia 2.

A continuación realizamos un pago con el cliente que antiguamente (ejercicio 5) tenía la *cookie* de la instancia 1, pero que después del ejercicio 6 pasó a estar enlazado a la instancia 2. Después de realizar dicho pago, comprobamos el estado final del balanceador de carga, el cual se muestra en la siguiente imagen:

Load Balancer Manager for 10.1.7.1

Server Version: Apache/2.2.14 (Ubuntu)
Server Built: Nov 3 2011 03:31:27

LoadBalancer Status for balancer://si2cluster

StickySession	Timeout	FailoverAttempts	Method
JSESSIONID|jsessionid	0	1	byrequests

Worker URL	Route	RouteRedir	Factor	Set	Status	Elected	To	From
http://10.1.7.2:28080	Instance01		1	0	Ok	23	13K	25K
http://10.1.7.3:28080	Instance02		1	0	Ok	34	23K	37K

Apache/2.2.14 (Ubuntu) Server at 10.1.7.1 Port 80

A pesar de que el cliente tenga la *cookie* de la instancia 1 y que esta está corriendo, las 3 peticiones que componen un pago han sido procesadas por la instancia 2. Esto es debido a que tras el ejercicio anterior, en el que la instancia 1 estaba tirada, este cliente ha mantenido la sesión iniciada sobre el mismo nodo (instancia 2). Esta caracterización ya se conocía desde

el ejercicio 5, donde se nos especificaba que "la sesión iniciada por cada usuario se mantiene sobre el mismo nodo", por lo que no hay sorpresas en la ejecución de este ejercicio.

8 Ejercicio 8

Enunciado: **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.
• Completar los datos de la tarjeta de modo que el pago fuera válido, y enviar la petición.
• Observar la instancia del cluster que procesa el pago, y razonar las causas por las que se rechaza la petición.

Respuesta a la cuestión:
Comenzamos un pago introduciendo los valores del mismo en la pantalla inicial y realizamos la llamada a ComienzaPago. Cuando estemos en la pantalla mostrada en la imagen siguiente, leemos la *cookie* de la instancia que ha procesado la petición:

Cuando sabemos la instancia que está atendiendo al pago inicializado, la terminamos con el comando `asadmin stop-instance Instance01`, tal y como se muestra a continuación:

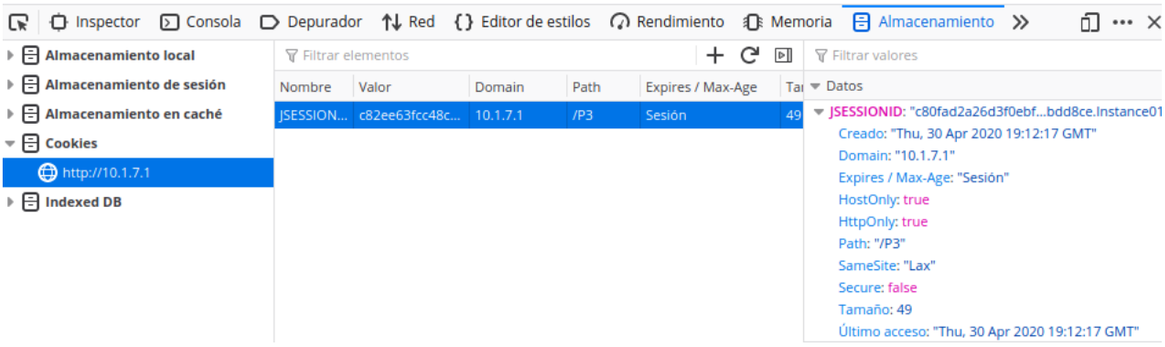
```
si2@si2srv01:~$ asadmin stop-instance Instance01
The instance, Instance01, is stopped.
Command stop-instance executed successfully.
si2@si2srv01:~$ asadmin list-instances -l
Name        Host      Port   Pid   Cluster     State
Instance01  10.1.7.2  24848  --    SI2Cluster   not running
Instance02  10.1.7.3  24848  2904  SI2Cluster   running
Command list-instances executed successfully.
```

Hecho esto, continuamos con el pago, obteniendo un aviso de pago incorrecto. Esto es debido a que, al matar la instancia 1, el resto del pago se ha procesado en la instancia 2 (gracias al balanceador de carga), pero esta instancia no tenía los datos de la primera pantalla, dando como resultado un error de "Pago incorrecto".

Pago con tarjeta

Pago incorrecto

Prácticas de Sistemas Informáticos II



En la siguiente imagen del balanceador de carga podemos ver que se han realizado nuevas peticiones en la instancia 1 (el inicio del pago) como en la instancia 2 (al fin y al cabo lanzar un error de pago incorrecto es algo que se tiene que procesar).

Load Balancer Manager for 10.1.7.1

Server Version: Apache/2.2.14 (Ubuntu)
Server Built: Nov 3 2011 03:31:27

LoadBalancer Status for balancer://si2cluster

StickySession	Timeout	FailoverAttempts	Method
JSESSIONID|jsessionid	0	1	byrequests

Worker URL	Route	RouteRedir	Factor	Set	Status	Elected	To	From
http://10.1.7.2:28080	Instance01		1	0	Err	26	15K	28K
http://10.1.7.3:28080	Instance02		1	0	Ok	35	24K	38K

Apache/2.2.14 (Ubuntu) Server at 10.1.7.1 Port 80

9 Ejercicio 9

Enunciado:
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.

Respuesta a la cuestión:
Modificamos el fichero P2.jmx de la P2 y habilitamos un ciclo de 1000 pruebas en un solo hilo contra la IP del cluster y URL de la aplicación (http://10.1.7.1/P3).

Antes de ejecutar las pruebas, eliminamos todos los pagos realizados hasta el momento:

Load Balancer Manager for 10.1.7.1

Server Version: Apache/2.2.14 (Ubuntu)
Server Built: Nov 3 2011 03:31:27

LoadBalancer Status for balancer://si2cluster

StickySession	Timeout	FailoverAttempts	Method
JSESSIONID|jsessionid	0	1	byrequests

Worker URL	Route	RouteRedir	Factor	Set	Status	Elected	To	From
http://10.1.7.2:28080	Instance01		1	0	Ok	0	0	0
http://10.1.7.3:28080	Instance02		1	0	Ok	0	0	0

Apache/2.2.14 (Ubuntu) Server at 10.1.7.1 Port 80

Se puede ver que el número de peticiones recibidas por cada instancia es nulo en este momento.

Ejecutamos las pruebas con ***jMeter*** obteniendo los siguientes resultados.

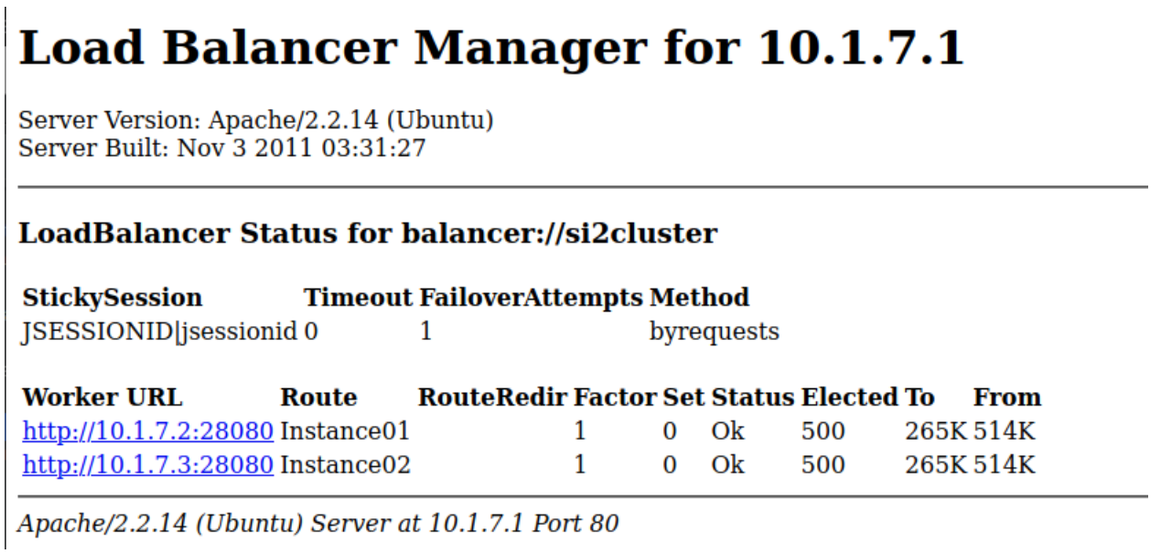
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Etiqueta	# Muestras	Media	Mediana	90% Line	95% Line	99% Line	Mín	Máx	% Error	Rendimiento	Kb/sec	Sent KB/sec
2	P3	1000	17	11	26	32	55	5	1949	0,000%	53,83580	73,89	0,00
3	Total	1000	17	11	26	32	55	5	1949	0,000%	53,83580	73,89	0,00

Comprobación en la **BD**:

```
si2@si2srv01:~$ psql visa -U alumnodb
psql (8.4.10)
Type "help" for help.

visa=# select count(*) from pago;
 count
-------
  1000
(1 row)
```

Información en el **Load Balancer Manager**:



Load Balancer Manager for 10.1.7.1

Server Version: Apache/2.2.14 (Ubuntu)
Server Built: Nov 3 2011 03:31:27

LoadBalancer Status for balancer://si2cluster

StickySession	Timeout	FailoverAttempts	Method
JSESSIONID|jsessionid	0	1	byrequests

Worker URL	Route	RouteRedir	Factor	Set	Status	Elected	To	From
http://10.1.7.2:28080	Instance01		1	0	Ok	500	265K	514K
http://10.1.7.3:28080	Instance02		1	0	Ok	500	265K	514K

Apache/2.2.14 (Ubuntu) Server at 10.1.7.1 Port 80

Utilizando la información aportada por el reporte de **jMeter** tras la prueba podemos ver que cada petición ha necesitado un **tiempo medio de 17 ms** de ejecución, con un rendimiento de 53'8. Destacable el porcentaje de **0% de errores**.

Por otro lado, el **Load Balancer Manager** nos indica que la carga se ha repartido equitativamente entre ambas instancias, con 500 peticiones para la instancia 1 y 500 para la 2. Esto nos da mucha información sobre el posible algoritmo de balanceo utilizado.

Una respuesta aparentemente lógica es que el algoritmo utilizado ha sido un proceso aleatorio con una probabilidad de elección del 50% para la instancia 1 y otro 50% para la instancia 2, pero esta respuesta es altamente errónea, debido a que la probabilidad de que las peticiones se hubiesen repartido equitativamente es prácticamente nula.

El algoritmo de balanceo utilizado es probablemente ***Round Robin*** (las peticiones son distribuidas entre las instancias de forma cíclica, independientemente de la carga de cada instancia) o **LRU** (*Last Recently Used*), que en el caso de dos instancias son equivalentes. Cualquiera de estos algoritmos hubiese realizado una carga exacta del 50% para cada instancia, tal y como ha resultado en la prueba.

X Conclusiones

Como habíamos dicho, el objetivo de esta práctica era mostrar la utilización de grupos de servidores (clusters) como herramienta para aumentar la disponibilidad de las aplicaciones distribuidas, utilizando las posibilidades que permite la arquitectura J2EE y estudiando los aspectos generales de clustering, configuración de un cluster, implementación de un cluster seguro y pruebas de disponibilidad de una aplicación J2EE.

Terminado el curso práctico de SI2, miramos adelante con la posibilidad de aplicar todo lo aprendido, ya sea en la docencia o profesionalmente.

XI Bibliografía

- SUN Microsystems, Clustering in Sun GlassFish 4.0
- Digital Signature Algorithm
- Apache HTTP server Version 2.2 Module mod_{proxy}
- Documentación P3 Moodle UAM