


Práctica 3: Seguridad y disponibilidad

Índice

ÍNDICE.....	1
1 OBJETIVOS.....	2
2 ENTORNO	2
3 EL PROTOCOLO SSH: ACCESO SEGURO A SERVIDORES.....	2
3.1 CARACTERÍSTICAS DE SSH	2
3.2 GENERANDO UN PAR DE CLAVES DSA	3
4 CONFIGURACIÓN EN CLUSTER	4
4.1 DEFINICIONES.....	5
4.2 ENTORNO DE TRABAJO EN EL LABORATORIO	5
4.3 PREPARACIÓN DE LA INFRAESTRUCTURA	6
4.4 CREACIÓN DEL CLUSTER	6
4.4.1 Crear la configuración de nodos SSH.....	6
4.4.2 Creación del cluster SI2Cluster.....	7
4.4.3 Modificación de la configuración de las instancias del cluster.....	8
4.5 IMPLEMENTACIÓN DEL CLUSTER.....	9
5 PREPARACIÓN DE LA PRÁCTICA PARA DESPLIEGUE EN CLUSTER	9
5.1 MODIFICACIÓN DEL CÓDIGO JAVA	9
5.2 MODIFICACIÓN EN LOS ARCHIVOS DE CONFIGURACIÓN.....	10
5.3 DESPLIEGUE DE LA APLICACIÓN	10
5.4 PRUEBA UNITARIA	10
6 CONFIGURAR UN BALANCEADOR DE CARGA	11
6.1 PROBLEMÁTICA DEL BALANCEO DE CARGA CON APLICACIONES JAVA EE	11
6.2 CONFIGURACIÓN DEL MÓDULO DE APACHE MOD_PROXY_BALANCER	11
7 JVMROUTE Y AFINIDAD DE SESIÓN	14
7.1 CAMBIO DE LA PROPIEDAD JVMROUTE.....	14
7.2 PROBAR EL EFECTO DE JVMROUTE.....	15
8 PRUEBAS DE BALANCEO DE CARGA.....	16
9 ENTREGA.....	17
10 BIBLIOGRAFÍA RECOMENDADA	17
APÉNDICE 1: ALTA DISPONIBILIDAD EN APLICACIONES JAVA EE.....	18
APÉNDICE 2: INFORMACIÓN ADICIONAL SOBRE CLUSTERS EN GLASSFISH	20
10.1 GESTIÓN DEL CLUSTER DESDE LA LÍNEA DE COMANDOS	20
10.2 DESPLEGAR UNA APLICACIÓN EN UN CLUSTER.....	20
10.2.1 Despliegue desde la consola de administración.....	20
10.2.2 Despliegue a través de la utilidad ant	21

	Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2	Práctica 3 09/04/2018
---	--	-------------------------------------

1 Objetivos

El objetivo de esta práctica es 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. En ella se estudiarán los siguientes aspectos:

- Aspectos generales de *clustering*.
- Configuración de un *cluster*.
- Implementación de un *cluster* seguro.
- Pruebas de disponibilidad de una aplicación J2EE.

2 Entorno

El entorno de realización de la práctica es el mismo que se ha empleado en las prácticas anteriores:

- Servidor de aplicaciones Glassfish V4.0
- Base de datos PostgreSQL 8.4
- El servidor web HTTP Apache como balanceador de carga con proxy inverso.
- La herramienta SSH (Secure Shell) como mecanismo de acceso seguro a los nodos del *cluster*.
- La herramienta JMeter para realización de pruebas masivas.


3 El protocolo SSH: acceso seguro a servidores

SSH son las siglas de *Secure Shell*. Aunque hemos venido utilizándolo a lo largo de las prácticas para acceder a las máquinas virtuales, merece la pena hacer la mención de algunas de las peculiaridades y beneficios que tiene frente a otros protocolos de acceso remoto como *rsh* y *telnet*.

3.1 Características de SSH

Con SSH podemos crear una sesión **autenticada y encriptada** a una máquina remota. Ya de por sí, esto es una ventaja frente al Telnet (cuyas sesiones pueden “espiarse” con mínimos conocimientos sobre redes), pero además nos proporciona otros beneficios:

- Verificación del otro extremo mediante firma digital con *claves de host*. Estas claves son únicas para cada servidor remoto, y la primera vez que se accede se pide confirmarlas. Una vez confirmadas, el cliente rechazará conectar si la clave de host cambia (el servidor ha sido suplantado).
- Autenticación del usuario remoto mediante clave privada. El servidor almacena la parte *pública* de nuestra clave, el cliente SSH tiene la clave *privada* almacenada de forma segura. Se garantiza que sólo pueda iniciar sesión remota aquel usuario que tenga la clave privada correspondiente a la clave pública remota.
- Permite encapsular otros protocolos mediante el *port tunneling*. Por ejemplo, el proceso SSH abre un puerto *local* allí donde se ejecute el cliente (opción `-L puerto_local:IPREMOTA:puerto_remoto`) que automáticamente será redirigido al puerto remoto indicado. Esto permite, entre otras cosas, acceder a través de un firewall siempre y cuando, tengamos acceso al puerto TCP 22 (predeterminado) que es donde el servidor SSH escucha peticiones.
- Incluye un mecanismo propio de transferencia de archivos (SCP) que sustituye a otros mecanismos menos seguros (FTP, RCP). El comando `scp` ya ha sido utilizado a lo largo de las prácticas.
- Posibilidad de automatizar tareas de administración, manteniendo la seguridad, pero sin solicitar contraseñas de forma interactiva. **Esto se realiza mediante el intercambio de claves públicas y privadas.** En este último punto es en el que nos centraremos en el siguiente ejercicio.

	<p>Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2</p>	<p>Práctica 3</p> <p>09/04/2018</p>
---	---	--

3.2 Generando un par de claves DSA

DSA son las siglas de *Digital Signature Algorithm*. Es un estándar federal de USA para la generación de firmas digitales. La firma digital es empleada para garantizar que el usuario no es suplantado. DSA permite el uso de varios algoritmos y tamaños de clave, y habitualmente se usan SHA-1 o SHA-2. Para una explicación formal del algoritmo detrás de DSA, consultar la bibliografía recomendada.

1. En un terminal de usuario, ejecutar:

```
ssh-keygen -t dsa
```

Se nos preguntará una clave (opcional) para cifrar la parte privada de la clave. Podemos pulsar **ENTER** para no cifrar esta clave, lo cual puede ser adecuado si la protegemos adecuadamente mediante permisos del sistema operativo, y a cambio permite usarla de forma no interactiva (sin que pida contraseña de desbloqueo).

Observaremos una salida similar a ésta:

```
Generating public/private dsa key pair.
Enter file in which to save the key (/home/si2/.ssh/id_dsa):
Created directory '/home/si2/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/si2/.ssh/id_dsa.
Your public key has been saved in /home/si2/.ssh/id_dsa.pub.
The key fingerprint is:
b0:6b:79:f5:ec:d5:a7:14:5f:d2:ab:d2:c5:47:6c:0e si2@si2srv01
The key's randomart image is:
+--[ DSA 1024]-----+
|
|
|  .      .  |
|    o    E.+|
|  . S .   +=o|
|    o . o   B=|
|    + .   + +=|
|  . .   o +...|
|           o.. |
+-----+
```

2. Una vez finalizada la ejecución anterior, observaremos que se ha creado en el directorio \$HOME/.ssh/ dos ficheros nuevos: **id_dsa** (clave privada) e **id_dsa.pub** (clave pública).
3. El fichero de la clave pública debe ser copiado a cada servidor remoto al que queramos poder iniciar sesión con nuestra clave privada. Para ello, podemos copiar dicho fichero por ejemplo mediante SCP:

```
scp .ssh/id_dsa.pub si2@10.X.Y.2:
scp .ssh/id_dsa.pub si2@10.X.Y.3:
```
4. Posteriormente, en cada uno de esos servidores, añadimos la clave pública al archivo de claves autorizadas (**.ssh/authorized_keys2**). Cada línea de ese fichero corresponde con una posible clave de un cliente al que estamos autorizando. Podemos, simplemente añadirlo con el comando cat:

```
mkdir -m 700 ~/.ssh  
cat ~/id_dsa.pub >> ~/.ssh/authorized_keys2
```

5. Por último, para verificar que la operación se ha realizado correctamente, podemos iniciar la sesión desde el primer servidor hacia aquellos donde se ha importado la clave pública. Salvo una (posible) primera validación de la clave de host remoto, la conexión ya no debería solicitar la clave

```
ssh si2@10.X.Y.2
```

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.

4 Configuración en *cluster*

Una de las características principales que deben cumplir los sistemas distribuidos es la transparencia. Como se ha presentado en las clases de teoría, consiste en la capacidad de un sistema distribuido de ocultar su estructura interna ante el usuario (usuario final o programador), presentando una apariencia única ante sus posibles variaciones.

Gracias a esta transparencia, los sistemas distribuidos deben ser capaces de trabajar adecuadamente independientemente de la carga que deban soportar (transparencia de escalado) y de las posibles incidencias que ocurran y produzcan un fallo en la operación (transparencia frente a fallos). La forma más común de hacer frente a estos requerimientos es mediante la ejecución del servicio en múltiples nodos de proceso. Estos conjuntos de servidores que prestan los mismos servicios se denominan con el término inglés *clusters* ("racimos", en castellano).

En un *cluster*, desde el punto de vista lógico, la operación entre el cliente y el servicio que gestiona el recurso, se mantiene inalterada. El cliente sigue percibiendo como si un único servidor estuviera procesando su petición, aunque, desde el punto de vista físico, existan varios nodos que pueden responder a la petición.

En esta práctica vamos a realizar una introducción a la configuración y operativa de *clusters* en un entorno J2EE. Un *cluster* se compondrá de varios servidores de aplicaciones. Cada nodo tendrá una réplica exacta del servicio o aplicación a probar, de manera que si uno de ellos sufre un fallo, el servicio se seguirá manteniendo en los demás nodos. También, dado que se dispondrá de varios elementos de proceso trabajando en paralelo, el rendimiento del conjunto será mayor que el que presenta un servidor aislado.

4.1 Definiciones

- **Domain Administration Server** (Servidor de administración del dominio): cuyas siglas son DAS. Este servidor es el responsable de la administración de los recursos del dominio. Los *clusters* que se definen sobre el dominio son un recurso más de este, con la particularidad que pueden incluir varias máquinas además del DAS.
- **Cluster** (Racimo): es un grupo de instancias del servidor que comparten una misma configuración, recursos y aplicaciones.
- **Instance**: Cada instancia alberga un servidor de aplicaciones virtual y es, normalmente, ejecutado en un servidor distinto. Un *cluster* se compone de un conjunto de instancias que se pueden distribuir en uno o varios nodos del dominio de aplicación. Tras instalar el GlassFish, por defecto, el dominio predeterminado (**domain1**) ya cuenta con una instancia **localhost-domain1**. Lo que vamos a crear durante la práctica son instancias **alternativas** a ésta, con un conjunto separado de programas en ejecución.
- **Node**: Es una configuración asociada a cada nodo donde se ejecuta una instancia. Existen de dos tipos: *nodo de configuración* (tipo “CONFIG”), que simplemente es un contenedor de la información de configuración y **nodo ssh** (tipo “SSH”) que permite, además, la gestión centralizada desde el DAS empleando dicho protocolo. De este modo se puede arrancar y parar las instancias del servidor en esa máquina, de forma remota. Para que la conexión entre el DAS y estos nodos funcione, el intercambio de claves públicas y privadas ha debido realizarse satisfactoriamente de forma previa.

4.2 Entorno de trabajo en el laboratorio

En el Apéndice 1 se muestra la arquitectura de alta disponibilidad habitual en los entornos de producción J2EE, tal como se ha presentado en las clases de teoría, y la simplificación que se propone para trabajar en el entorno de laboratorio con los recursos disponibles. Dicha arquitectura se desplegará en máquinas físicas y virtuales tal como se muestra en la siguiente figura:

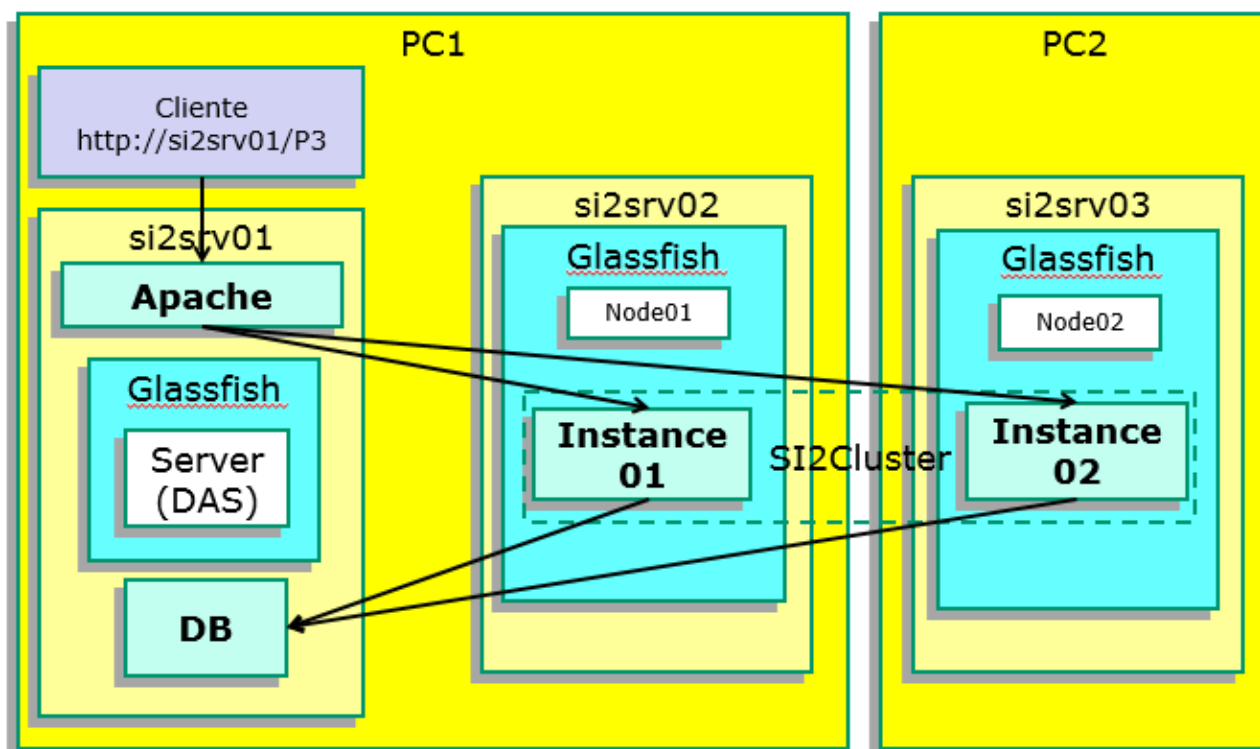



Figura 1: Diagrama de despliegue de la práctica 3, empleando tres máquinas virtuales en dos PCs

	Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2	Práctica 3 09/04/2018
---	--	-------------------------------------

4.3 Preparación de la infraestructura

Partiendo de las 3 máquinas virtuales indicadas en la figura anterior, todas ellas iniciadas, procederemos de la siguiente manera:

1. Inicie las tres máquinas virtuales tal y como se indica en la figura anterior. Asegúrese de que los parámetros de consumo de memoria se han respetado.
2. Conéctese mediante SSH al primer nodo (**si2srv01**)
3. Inicie la instancia *server* habitual de todas las prácticas, mediante el inicio del dominio domain1:
`asadmin start-domain domain1`
4. **No inicie los dominios en los nodos 2 y 3. Verifique que no esté en ejecución ningún proceso Java.**

4.4 Creación del *cluster*

Los pasos a seguir para crear y configurar el *cluster* son los siguientes:

4.4.1 Crear la configuración de nodos SSH

Para facilitar la gestión remota del *cluster*, vamos a optar por crear nodos de tipo SSH. Estos nodos **se crean desde el DAS**, y requieren que se haya realizado correctamente el **intercambio de claves** entre el servidor si2srv01 y los otros dos (si2srv02 y si2srv03).

1. Inicie sesión SSH contra la primera máquina virtual (**10.X.Y.1**). **Desde este nodo realizaremos todos los pasos siguientes, salvo que se indique lo contrario.**
2. Cree el nodo ssh *Node01*. Se requiere indicar dónde se va a ejecutar dicho nodo (IP **10.X.Y.2**), así como el nombre del usuario del sistema operativo que se usará para conectar remotamente por SSH, ya que éste (si2) es distinto del propio del GlassFish (admin). Opcionalmente también se puede indicar la ruta donde está instalado el software (remoto) del GlassFish. **Habrà que actualizar el fichero /opt/SI2/passwordfile si se ha cambiado la contraseña del usuario admin del servidor glassfish¹.**

```
asadmin --user admin --passwordfile /opt/SI2/passwordfile create-node-ssh --sshuser si2 --nodehost 10.X.Y.2 --nodedir /opt/glassfish4 Node01
```

3. Repita la operación para el nodo *Node02*

```
asadmin --user admin --passwordfile /opt/SI2/passwordfile create-node-ssh --sshuser si2 --nodehost 10.X.Y.3 --nodedir /opt/glassfish4 Node02
```

4. Pruebe a listarlos, para ello emplee el siguiente comando:

```
asadmin --user admin --passwordfile /opt/SI2/passwordfile list-nodes
```

A continuación hágales un *ping*. Para ello emplee el siguiente comando:

```
asadmin --user admin --passwordfile /opt/SI2/passwordfile ping-node-ssh Node01
asadmin --user admin --passwordfile /opt/SI2/passwordfile ping-node-ssh Node02
```

¹ Se puede evitar teclear los argumentos `--user` y `--passwordfile` mediante variables de entorno. Ver Nota más adelante.

Realizadas las operaciones, desde la consola de administración del dominio (<http://10.X.Y.1:4848>), seleccionando en el menú de navegación la opción *Nodes*, podremos ver los dos nodos creados, junto con el preexistente localhost-domain1 correspondiente al propio dominio de administración (instancia *server*).

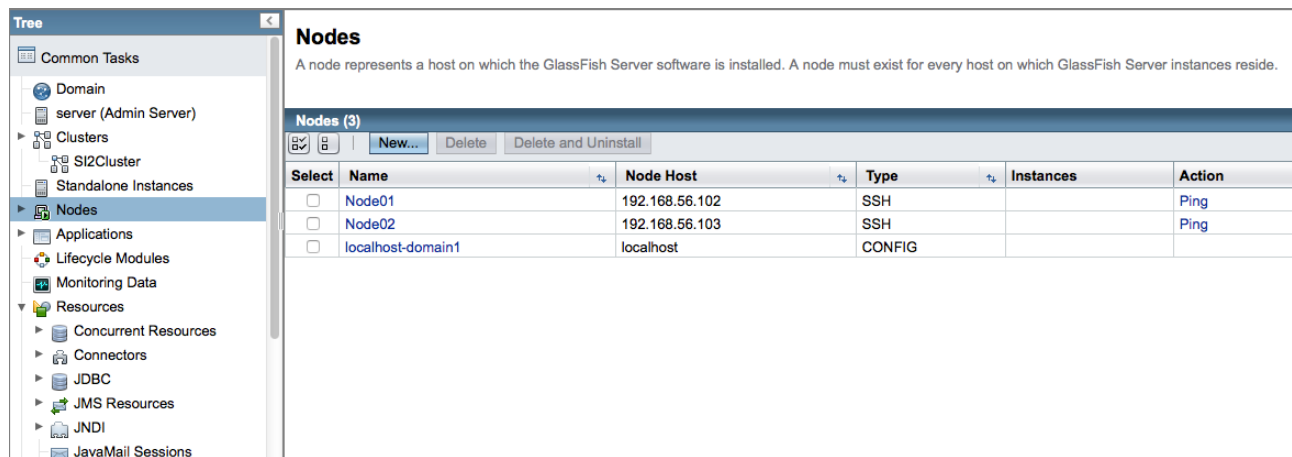


Figura 2: Nodos del cluster

4.4.2 Creación del *cluster SI2Cluster*

En lo que sigue se continuará realizando desde la línea de comandos del servidor si2srv01 (DAS).

Nota: Se puede evitar introducir el usuario y el password file estableciendo las variables de entorno:

```
export AS_ADMIN_USER=admin
export AS_ADMIN_PASSWORDFILE=/opt/SI2/passwordfile
```

Para todas las tareas que siguen debe estar iniciado el dominio.

1. Crear el cluster:

```
asadmin create-cluster SI2Cluster
```

2. Listarlo:

```
asadmin list-clusters
```

3. Puesto que el nodo del DAS va a encargarse de iniciar y detener instancias remotamente, debe asegurarse que todos los nodos se “conocen” mutuamente a través de su nombre asociado a la dirección IP. Para ello, verifique en si2srv01, si2srv02 y si2srv03 que sus respectivos ficheros `/etc/hosts` contienen algo similar a esto: (Si no fuese el caso, edite el fichero como root e incluya la información necesaria).

```
10.X.Y.1  si2srv01
10.X.Y.2  si2srv02
10.X.Y.3  si2srv03
...
```

Crear dos instancias asociadas a cada uno de los dos nodos anteriormente creados:

```
asadmin --user admin --passwordfile /opt/SI2/passwordfile create-instance --cluster SI2Cluster --node
Node01 Instance01
asadmin --user admin --passwordfile /opt/SI2/passwordfile create-instance --cluster SI2Cluster --node
Node02 Instance02
```

4. Listar las instancias recién creadas:

```
asadmin --user admin --passwordfile /opt/SI2/passwordfile list-instances -l
```

5. Iniciar el cluster:

```
asadmin --user admin --passwordfile /opt/SI2/passwordfile start-cluster SI2Cluster
```

4.4.3 Modificación de la configuración de las instancias del *cluster*

Tal y como se ha creado el *cluster*, se ha realizado una copia de la configuración *default-config* con el nuevo nombre *SI2Cluster-config* para su utilización por las instancias del *cluster*. Es necesario modificarla para adaptarla a las necesidades del entorno de las prácticas.

Se puede acceder a las configuraciones a través del menú de navegación de la consola de administración seleccionando la opción *Configurations*.

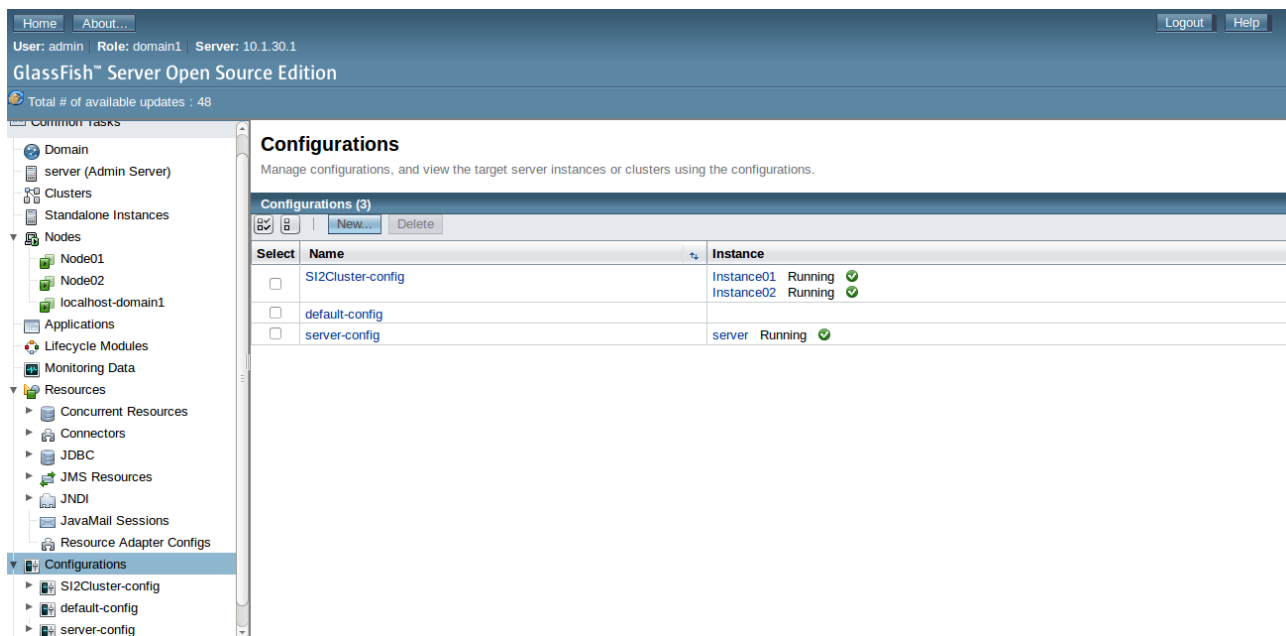



Figura 3

Seleccionar la configuración creada, *SI2Cluster-config*, para realizar los cambios necesarios en ella:

- En las opciones JVM Settings -> JVM Options:
 - Comprobar que la opción `-server` (que permite el funcionamiento multihilo) está habilitada. Esta opción es alternativa a `-client`. En caso de que no esté, se deberá reemplazar la una por la otra.
 - Añadir `-Xms128m`. Define la memoria mínima de los servidores a 128 MB.
 - Modificar `-Xmx512m` a `-Xmx128m`. Define la memoria máxima de los servidores a 128 MB.
 - Modificar `-XX:MaxPermSize=192m` a `-XX:MaxPermSize=96m`. Define el máximo tamaño del pool de objetos permanentes a 96 MB.

Siempre que se realicen cambios en la configuración será preciso rearrancar todas las instancias del *cluster*, lo cual puede realizarse simplemente mediante un reinicio del cluster (`stop-cluster`, `start-cluster`):

	Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2	Práctica 3 09/04/2018
---	--	-------------------------------------

```
asadmin stop-cluster SI2Cluster
asadmin start-cluster SI2Cluster
```

4.5 Implementación del *cluster*

En este apartado implementaremos la configuración descrita.

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) correspondientes² están efectivamente corriendo en cada una de las dos máquinas virtuales. Adjunte evidencias a la memoria de la práctica.

5 Preparación de la práctica para despliegue en *cluster*

A partir del código de la práctica 1 (P1-base) se realizarán unas pequeñas modificaciones que permitan registrar los nodos del *cluster* que ejecutan las peticiones. Esta nueva aplicación se denominará P3.

Téngase en cuenta que **haremos todo el despliegue como si todos los componentes estuvieran en el DAS** (transparencia a la ubicación) ya que va a ser el propio cluster del Glassfish el que se encargue de distribuir adecuadamente los componentes en cada uno de los nodos físicos.

Recordar:


- Renombrar la carpeta P1-base a P3
- Copiar dentro de la carpeta datagen el fichero "listado.csv" proporcionado en esta práctica. Esto es importante para que el script jmx que se va a utilizar más adelante encuentre los ficheros .csv de listas de tarjetas.
- Copiar dentro de la carpeta "sql" la carga de datos sql (insert.sql) para que todos los proyectos tengan los mismos datos en base de datos 'y para que concuerden con los valores en listado.csv.

5.1 Modificación del código Java

Las modificaciones a realizar son las siguientes:

- Modo *debug* desactivado (en caso de que estuviera habilitado por pruebas anteriores del alumno)
- Recoger como información adicional del pago el nombre de la instancia del *cluster* que ha servido la petición y la dirección IP donde se encuentra ejecutándose esa instancia. Para ello habrá que realizar las siguientes modificaciones:
 - Modificar la tabla *pago* para que contenga dos nuevas columnas: *instancia* e *ip*. La primera almacenará el nombre de la instancia y la segunda la dirección IP. Ambos pueden ser de tipo VARCHAR(50) por ejemplo.
 - Modificar el *bean Pago* para que pueda incorporar las dos nuevas propiedades anteriores.
 - Modificar los *servlets ComienzaPago* y *ProcesaPago* para que a la hora de crear el *bean Pago* se añada la información relativa a los dos parámetros anteriores.
 - El nombre de la instancia se obtiene de:
System.getProperty("com.sun.aas.instanceName")
 - La dirección IP:
java.net.InetAddress.getLocalHost().getHostAddress()

² Por ejemplo, mediante el mandato `ps -aefl | grep java`

	<p>Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2</p>	<p>Práctica 3</p> <p>09/04/2018</p>
---	---	--

- Modificar *VisaDAO* para que a la hora de insertar el pago se incluyan los dos parámetros anteriores en la tabla *pago*.

5.2 Modificación en los archivos de configuración

Las modificaciones a realizar son las siguientes:

- Cambiar el fichero “postgresql.xml” por el suministrado.
- Modificar build.properties

Archivo build.properties:

Original	Nuevo
<pre> nombre=P1-base ... as.host=localhost ... as.target=server ... </pre>	<pre> nombre=P3 ... as.host=10.X.Y.1 ... as.target=SI2Cluster ... </pre>

- Modificar postgresql.properties:

Archivo postgresql.properties:

Original	Nuevo
<pre> ... db.host=localhost ... db.client.host=localhost </pre>	<pre> ... db.host=10.X.Y.1 ... db.client.host=10.X.Y.1 </pre>

5.3 Despliegue de la aplicación

Desplegar la aplicación tal como se hizo en la práctica 1:

ant todo

Eliminar aplicaciones y recursos desplegados anteriormente en cualquiera de los 3 servidores Glassfish a utilizar.

5.4 Prueba unitaria

A continuación haremos un **acceso individual** a cada una de las dos instancias. Todavía no contamos con un *balanceador* que garantice la transparencia para el usuario (lo veremos más adelante) pero comprobaremos que todo lo creado hasta el momento es válido.

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).

6 Configurar un balanceador de carga

El objetivo de este apartado es dotar de verdadera transparencia a la ubicación a nuestra aplicación. Para ello, haremos que la dirección IP del DAS responda a todas las peticiones externas, pero balanceando la carga entre cada uno de los nodos del *cluster*.

Como se presentó en el diagrama de despliegue el balanceador de carga residente en Apache se va a instalar en el servidor `si2srv01`, compartiendo máquina virtual con el servidor de bases de datos y con el DAS. El servidor Apache ya se encuentra instalado en la máquina virtual, y se activa automáticamente al arrancarla.

6.1 Problemática del balanceo de carga con aplicaciones JAVA EE

El principal problema de realizar un balanceo de carga es que, después de todo, la aplicación está ejecutándose en varios nodos separados por lo que *en principio* dichas instancias no comparten información (entorno de aplicación separado, copias distintas de memoria, de sesiones y EJBs).

Para resolver esta cuestión, GlassFish proporciona un mecanismo de *replicación de memoria* que puede ser configurado, pero que al final puede resultar costoso computacionalmente (cada modificación en un dato de memoria debe ser reenviado a todos los nodos del cluster, Glassfish usa el algoritmo GMS con reenvíos UDP *multicast* cada pocos segundos).

Sin embargo existe una solución mucho más simple que en muchos casos resuelve el problema cuando son aplicaciones con muchos clientes potenciales: **afinidad de la sesión**. Ésta garantiza que, una vez creado el objeto *Session* (que tiene una correspondencia con una *Cookie* de sesión), el balanceador reenviará futuras peticiones de este mismo cliente (con la misma cookie de sesión) al mismo servidor Glassfish. En situaciones en que hay muchos clientes, se garantiza un reparto equitativo sin mayor impacto para el rendimiento.

6.2 Configuración del módulo de Apache `mod_proxy_balancer`

Los pasos para configurarlo son los siguientes.

1. Crear el archivo de configuración del módulo `mod_proxy_balancer`

`/etc/apache2/mods-available/proxy_balancer.conf`

Este archivo se debe crear como usuario `root`

La configuración mínima del balanceador, teniendo en cuenta que el *cluster* sobre el que desplegar tiene la topología descrita anteriormente, sería similar a la siguiente:

A	<code>ProxyRequests Off</code>
B	<pre><Proxy balancer://SI2Cluster> BalancerMember http://10.x.y.2:XXXXX route=Instance01 BalancerMember http://10.x.y.3:XXXXX route=Instance02 </Proxy></pre>
C	<pre><Location /P3> Order allow,deny Allow from all ProxyPass balancer://SI2Cluster/P3 stickysession=JSESSIONID jsessionid scolonpathdelim=On ProxyPassReverse balancer://SI2Cluster/P3 </Location></pre>
D	<pre><Location /balancer-manager> SetHandler balancer-manager </Location></pre>

Las directivas de configuración empleadas tienen el siguiente significado:

A. Configura el *proxy* para inhibir su funcionamiento como proxy directo, y permitir únicamente la funcionalidad de *proxy* inverso.

B. Define los servidores que pertenecen al *cluster* `SI2Cluster` para realizar el balanceo de carga entre ellos. **El alumno deberá configurar las direcciones IP correctas de sus servidores.**

En esta directiva, la etiqueta *route* indica el **nombre de la instancia** que responde en la correspondiente URL. Permite mantener la afinidad de la sesión con el mismo servidor en las peticiones del cliente.

C. Identificación de las URL a balancear. En este caso, las que respondan al patrón `/P3`, contexto de la aplicación que se desplegará.

Las directivas `Order` y `Allow` permiten definir los clientes con derecho de uso de estas reglas. En el ejemplo está definido un acceso general.

La directiva `ProxyPass` identifica la conversión a realizar en la URL. En este caso, toda URL con path `/P3` será redirigida a `balancer://SI2Cluster/P3`, pasando, por tanto, a ser gestionada por el grupo de servidores definidos en (B).

Dentro de esta directiva, el parámetro

```
stickysession = JSESSIONID|jsessionid
```

proporciona el nombre del identificador de sesión del cliente. En caso de utilizarse nombres distintos del identificador en *cookie* y en *URL rewrite*, se ponen ambos separados por "|".

El parámetro:

```
scolonpathdelim=on
```

es necesario para que Apache Tomcat pueda incluir la información de ruta dentro del path que contiene la URL, en caso de usarse URL *rewrite*. La información de sesión y de ruta estará separada de la URL original por el carácter ";".

Una petición que lleve como información asociada:

```
Set-Cookie: JSESSIONID=63aa7dd138864cde0bcafd6193ce.Instance02; Path=/P3
```

contiene información de ruta, y para mantener la afinidad de la sesión se enviará directamente a la instancia *Instance02*. Sin embargo, si no existe `JSESSIONID`, el balanceador considera que no hay sesión establecida y utilizará el algoritmo de balanceo para determinar el destino de la petición dentro del *cluster*.

La directiva `ProxyPassReverse` permite el funcionamiento correcto de la redirección de peticiones desde el servidor balanceado, de modo que contengan siempre la información de path externa y no la manejada dentro del *cluster*.

D. Activa el path `/balancer-manager` (accesible desde `http://10.X.Y.1/balancer-manager`) para acceder a la información de estadísticas de balanceo del servidor Apache. La información que presenta esta página se puede ver en la siguiente figura.

Load Balancer Manager for 10.1.99.1

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://10.1.99.2:28080	Instance01		1	0	Ok	2	1.0K	1.2K
http://10.1.99.3:28080	Instance02		1	0	Err	1	0	0

Apache/2.2.14 (Ubuntu) Server at 10.1.99.1 Port 80

Figura 4: Aspecto que presenta la página de status del balanceador

En el caso de la figura, el estado de la instancia *Instance02* es "Err", que indica que se encuentra inactivo. El balanceador enviará todas las peticiones a *Instance01*, hasta que detecte que *Instance02* vuelve a estar activo. El balanceador comprueba por defecto cada 60 seg. si un servidor inactivo ha vuelto a la actividad. Ese tiempo, como muchos otros parámetros de configuración del balanceador, puede ser cambiado por configuración en las directivas *BalancerMember* y *ProxyPass*.

2. Activar la inclusión de esta configuración en el arranque del servidor apache:

```
cd /etc/apache2/mods-enabled
sudo ln -sf ../mods-available/proxy_balancer.conf
```

Tened en cuenta que el comando sudo solicitará el password del propio usuario del sistema, si2, que es la misma con la que se ha iniciado sesión.

3. En el mismo directorio, activar los módulos de Apache necesarios para cubrir la funcionalidad de balanceador de carga:

```
sudo ln -sf ../mods-available/proxy.load
sudo ln -sf ../mods-available/proxy_http.load
sudo ln -sf ../mods-available/proxy_balancer.load
```

4. Rearrancar el servidor apache:

```
sudo service apache2 restart
```

7 jvmRoute y afinidad de sesión

Esta propiedad del cluster “jvmRoute” hace que cada una de las instancias del *cluster* modifique la generación del identificador de sesión para incluir en él información de la instancia que ha procesado la petición. En este caso, el identificador que se introducirá será el nombre asignado a la instancia en su creación. Gracias a esto se puede mantener la **afinidad de la sesión** en el balanceador de carga.

7.1 Cambio de la propiedad jvmRoute

Esta propiedad puede ser modificada dentro de la configuración “SI2Cluster-config” del cluster “SI2Cluster”. Tal y como vimos en el punto 4.4.3 se puede acceder a dicha configuración a través de la consola de administración seleccionando la opción *Configurations -> SI2Cluster-config*

En este caso la propiedad jvmRoute forma parte de las propiedades de sistema y son accesibles en la sección “System Properties”:

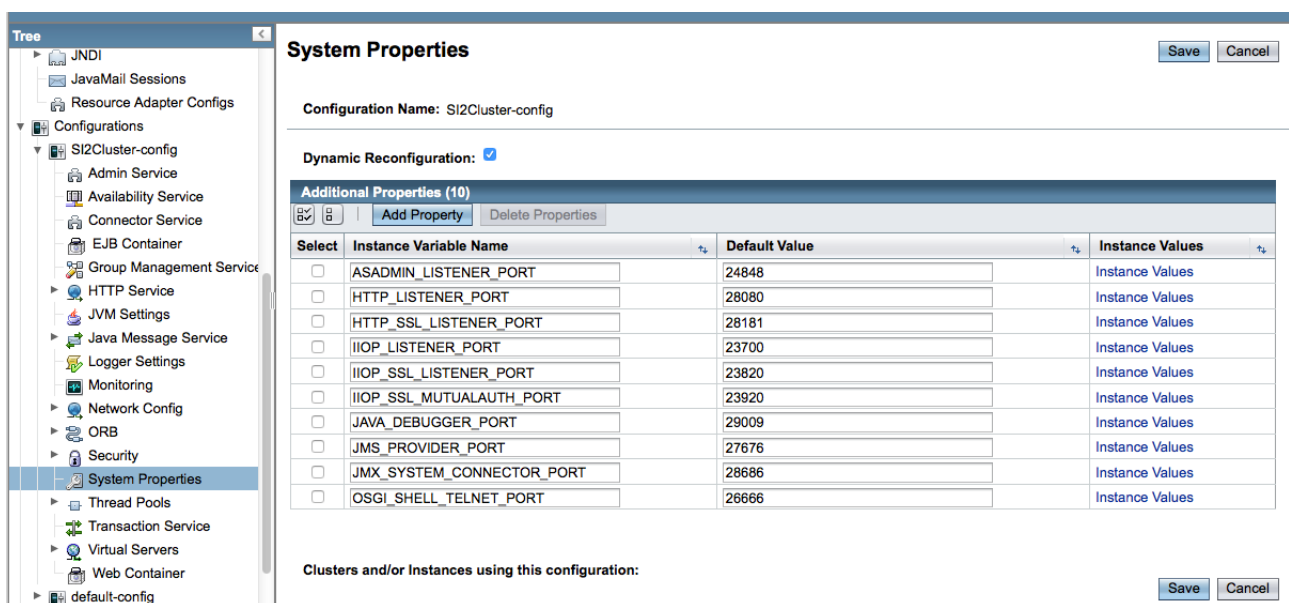


Figura 5: Pantalla de la consola de administración dentro de la sección “System Properties” de la configuración “SI2Cluster-config”

Dentro de las propiedades de sistema para SI2Cluster-config se puede añadir la propiedad

jvmRoute

indicando como valor por defecto para ella

`${com.sun.aas.instanceName}`

y guardando los cambios mediante el botón “Save”.

Una vez modificada cualquier propiedad del cluster es necesario reiniciar reiniciarlo para que el cambio surta efecto:

```
asadmin stop-cluster SI2Cluster
```

```
asadmin start-cluster SI2Cluster
```

7.2 Probar el efecto de jvmRoute

Para poder comprobar el efecto de jvmRoute en la afinidad de sesión es imprescindible poder acceder a la ventana de gestión de cookies del navegador que se vaya a utilizar.

En el caso de Firefox:

“Preferencias” -> “Privacidad” -> “Mostrar cookies...”

En el caso de Chrome:

Selecciona "Configuración" -> "Mostrar opciones avanzadas...". En la sección "Privacidad", haz clic en el botón "Configuración de contenido...". En la sección "Cookies", haz clic en "Todas las cookies y los datos de sitios"

Desde esta ventana es posible visualizar y eliminar las cookies del navegador.

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: YYYYYYYYYYYYYYYYYYYY
Domain: 10.X.Y.1
Path: /P3

6- Añadir la propiedad “jvmRoute” al cluster y rearrancar el cluster.

7- Eliminar todas las cookies del navegador.

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: ZZZZZZZZZZZZZZZZZZZZZZ
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.

8 Pruebas de balanceo de carga

Tras el despliegue de la aplicación P3 en el *cluster*, se realizarán pruebas para verificar el funcionamiento correcto del balanceo de carga.

Para realizar los ejercicios que se presentan a continuación la propiedad `jvmRoute` debe estar activa.

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.

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.

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.**


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.
- 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.

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.

	Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2	Práctica 3 09/04/2018
---	--	-------------------------------------

9 Entrega

La entrega de la práctica debe incluir:

- Código completo de la aplicación P3, preparado para el despliegue sobre el *cluster* SI2Cluster, sin incluir los directorios **build** y **dist**
- Ficheros de configuración de Glassfish, domain.xml, de los tres servidores:
DAS: /opt/glassfish4/glassfish/domains/domain1/config/domain.xml
Instance01: /opt/glassfish4/Node01/Instance01/config/domain.xml
Instance02: /opt/glassfish4/Node02/Instance02/config/domain.xml
- Fichero de configuración del balanceador
/etc/apache2/mods-available/proxy_balancer.conf
- Memoria de la ejecución de la práctica, en la que se detallen respuesta y evidencias a todas las cuestiones planteadas.

Nomenclatura del fichero a entregar: SI2P3_<grupo>_<pareja>.zip (ejemplo: SI2P3_2311_1.zip)

De acuerdo al calendario de prácticas publicado, esta práctica se entregará la semana del **1 al 4 de mayo de 2018, justo antes de la correspondiente sesión de prácticas en la que se realizará el examen de prácticas de la asignatura.**

Dado que los días 1 de mayo (martes) y 2 de mayo (miércoles) son festivos, los alumnos que tengan docencia esos días deberán entregar la práctica el día **3 de mayo (por confirmar)**. Justo antes de la hora en la que dé comienzo su examen de prácticas de la asignatura.

10 Bibliografía recomendada

Digital Signature Algorithm

URL: http://en.wikipedia.org/wiki/Digital_Signature_Algorithm

SUN Microsystems, Clustering in *Sun GlassFish 4.0*

URL: <https://glassfish.java.net/docs/4.0/ha-administration-guide.pdf>

Apache HTTP server Version 2.2 Module mod_proxy.

URL: http://httpd.apache.org/docs/2.2/mod/mod_proxy.html

Apéndice 1: Alta disponibilidad en aplicaciones JAVA EE

La alta disponibilidad es un problema global de una instalación informática, y, por tanto, afecta a todos los elementos de la cadena de procesamiento. Tal como se ha presentado en clase de teoría, esta cadena de procesamiento en un sistema distribuido basado en la arquitectura J2EE tiene la siguiente estructura:

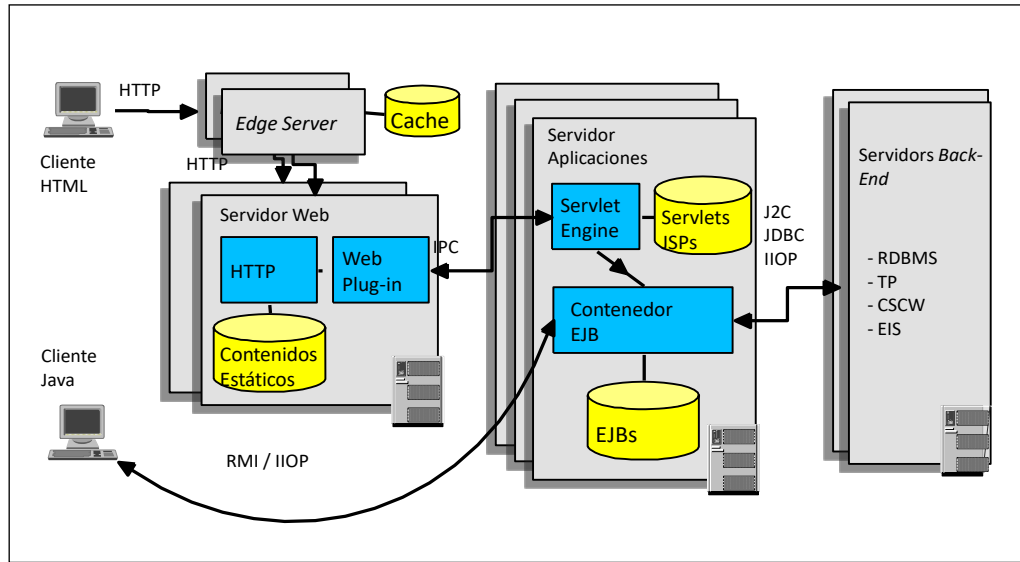


Figura 6: Cadena de procesamiento en arquitectura distribuida J2EE

Simplificando las estructuras internas de los elementos, considerando un sistema con redundancias simple (dos nodos de proceso equivalentes por cada elemento de la cadena de proceso), y particularizando al caso que venimos presentando en las prácticas, la arquitectura de alta disponibilidad objetivo se puede resumir en la figura 7. Los elementos que se presentan en ella son:

- **Balanceador de carga:** Dispositivo que recibe todas las peticiones de los clientes y las reparte entre los primeros nodos de proceso. La disponibilidad de estos sistemas suele ser tipo Activo-Pasivo, con detección de caídas en el nodo principal mediante un protocolo de alta disponibilidad (VRRP, por ejemplo).
- **Servidor HTTP:** Primer elemento de proceso de las peticiones de los clientes. Sirve los contenidos estáticos y detecta las peticiones que requieren la ejecución de un proceso para resolución, enviándoselas al siguiente elemento de la cadena de procesamiento. Todos los nodos de proceso están activos.
- **Servlet Engine:** Entorno de ejecución de los *servlets* que implementan la funcionalidad requerida de la aplicación. En la configuración que presentamos, todos los nodos de proceso están activos.
- **Gestor de bases de datos:** Almacén de información. La disponibilidad habitual de estos elementos es tipo Activo-Pasivo.

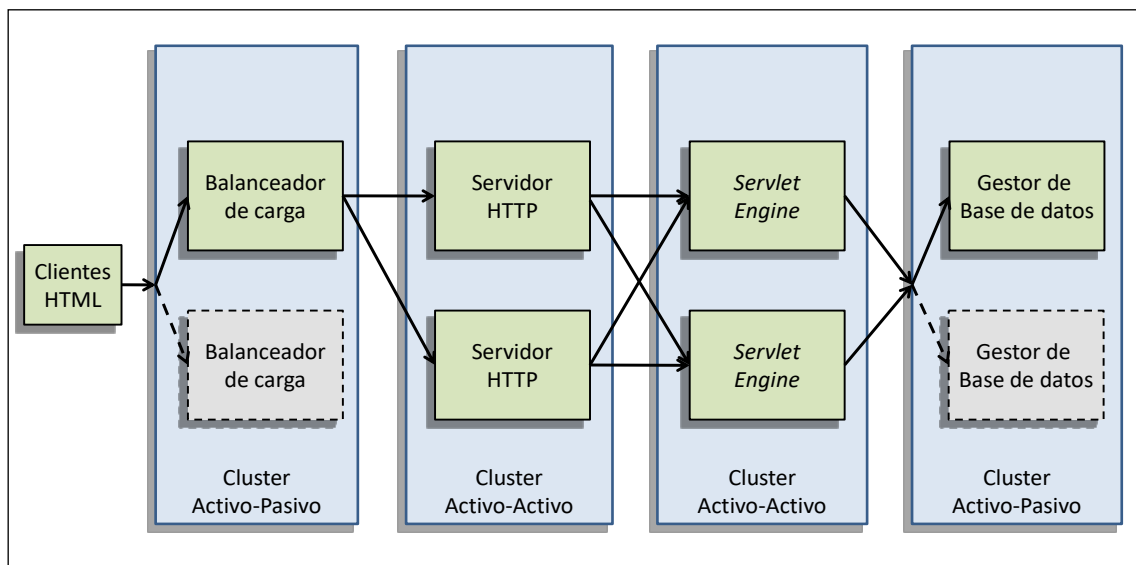


Figura 7: Arquitectura de alta disponibilidad objetivo

En nuestro entorno de laboratorio, dadas las limitaciones de ordenadores disponibles, trabajaremos con una estructura simplificada, tal como se muestra en la siguiente figura:

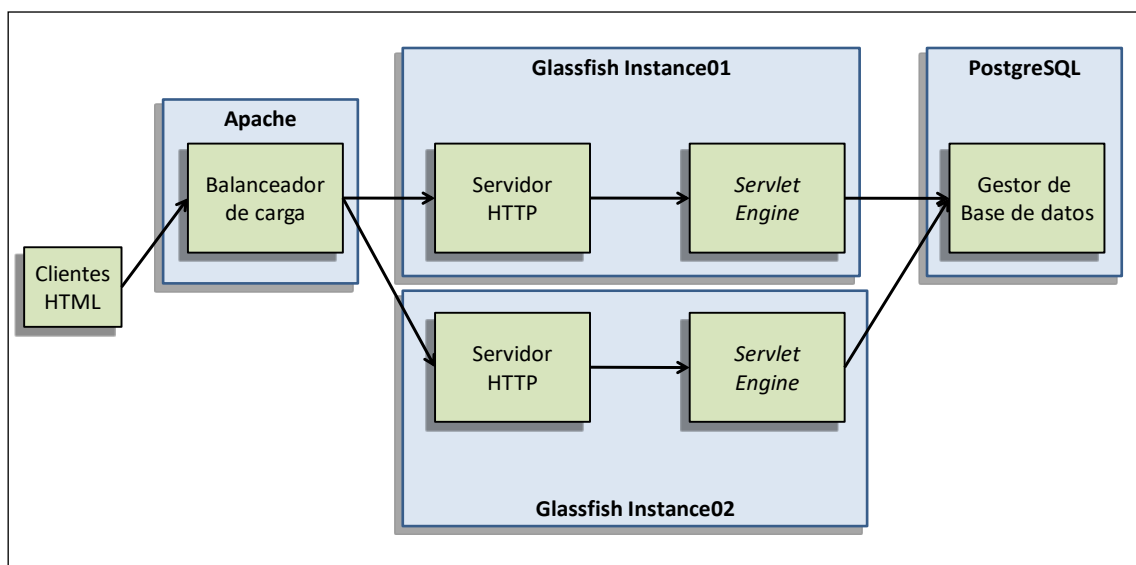



Figura 8: Estructura simplificada de la arquitectura

La primera de las simplificaciones que realizamos en el entorno se debe a que estamos trabajando con un único servidor para procesar los contenidos estáticos y dinámicos. Por tanto, los bloques de Servidor HTTP y Servlet Engine se implementan con el servidor de aplicaciones Glassfish.

Por su parte, la capa del gestor de base de datos se implementa con el gestor PostgreSQL, en un proceso independiente.

La parte del balanceador de carga se ha implementado mediante el módulo *mod_proxy_balancer* del servidor HTTP Apache. Este servidor, por tanto, no se empleará para entregar servicios estáticos, sino únicamente como distribuidor de trabajo a los nodos de proceso de Glassfish.

En la práctica únicamente nos centraremos en la alta disponibilidad Activo-Activo de los servidores HTTP y Servlet Engine. De los servicios en configuración Activo-Pasivo únicamente se implementará el nodo activo.

	Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2	Práctica 3 09/04/2018
---	--	-------------------------------------

Apéndice 2: Información adicional sobre *clusters* en Glassfish

10.1 Gestión del cluster desde la línea de comandos

Todos estos comandos se ejecutan contra el DAS. A él se dirigirán por defecto todos los comandos que se ejecuten en la línea de comandos del nodo que lo contiene.

Arrancar el cluster

```
asadmin start-cluster SI2Cluster
```

Al arrancar un *cluster*, se arrancan también todas las instancias que tuviera asociadas.

Parar el cluster

```
asadmin stop-cluster SI2Cluster
```

Listar instancias

Para listar todas las instancias:

```
asadmin list-instances -l
```

Arrancar una instancia

Se puede arrancar una instancia en particular:

```
bin/asadmin start-instance <nombre_de_instancia>
```

Parar una instancia

Se puede detener una instancia en particular:

```
asadmin stop-instance <nombre_de_instancia>
```

10.2 Desplegar una aplicación en un cluster

10.2.1 Despliegue desde la consola de administración

El proceso de despliegue de una aplicación sobre un *cluster* es similar al presentado en prácticas anteriores.

Desde la consola de administración el proceso sería:

1. Seleccionar *Applications > Web Applications > Deploy*
2. Subir el archivo .war en el área de texto "*Packaged file to be uploaded to the server*".
3. Seleccionar sobre que servidores se quiere desplegar la aplicación. Por defecto, se despliega en el servidor autónomo ("server"). Se pueden añadir más servidores o quitar los que hubiere. Para desplegarlo en un cluster habrá que incluir éste dentro de la lista de servidores (véase la figuras más abajo).

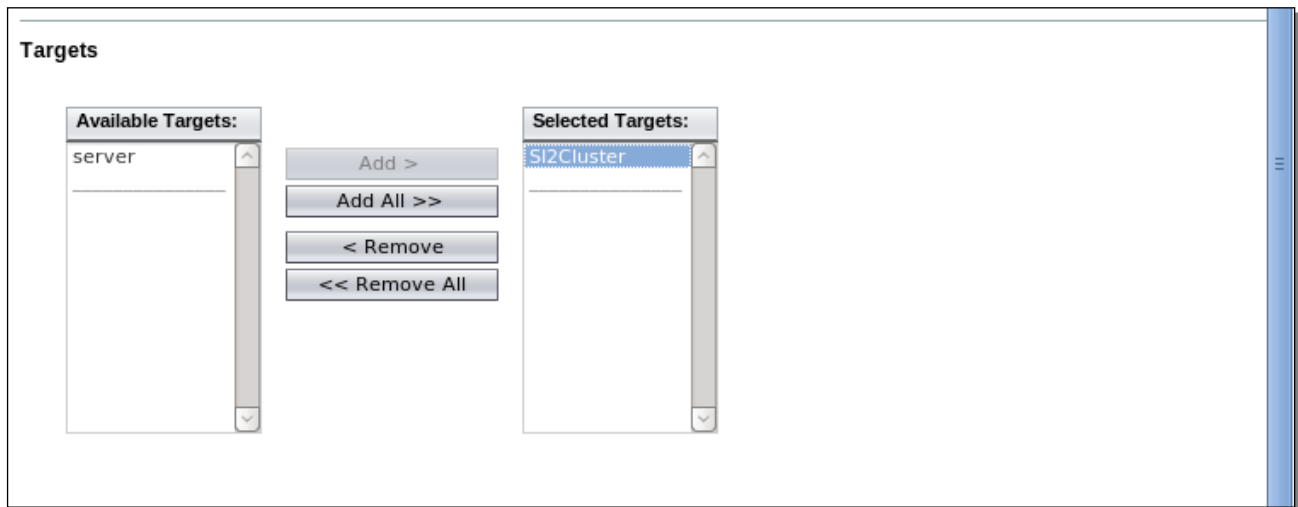


Figura 9: Adición manual del target correspondiente al cluster para un despliegue manual

Tras este despliegue sería necesario identificar los recursos que la aplicación va a necesitar, como por ejemplo, las conexiones a bases de datos a través de los *data sources*.

10.2.2 Despliegue a través de la utilidad ant

A través de la utilidad *ant* que se ha empleado en el resto de las prácticas es posible desplegar una aplicación directamente en un *cluster*, así como crear y asignar todos los recursos necesarios para su funcionamiento.

Los archivos de configuración de *ant* que se han entregado en las prácticas anteriores ya están preparados para esa función. El procedimiento es, en resumen, añadir la opción `-target NombreDelCluster` a cada comando de despliegue `asadmin`.