

UNIVERSIDAD AUTÓNOMA DE MADRID

DEPARTAMENTO DE INFORMÁTICA

Sistemas Informáticos II

Práctica - 3

DAVID TEÓFILO GARITAGOITIA ROMERO

Registro de Cambios

Versión¹	Fecha	Autor	Descripción
1.0	28.3.2025	DT	Primera versión.

¹La asignación de versiones se realizan mediante 2 números $X.Y$. Cambios en Y indican aclaraciones, descripciones más detalladas de algún punto o traducciones. Cambios en X indican modificaciones más profundas que o bien varían el material suministrado o el contenido de la práctica.

Contents

1	Objetivos	3
2	Descripción del Proyecto	5
2.1	Propósito	5
2.2	Componentes	6
2.3	Flujo General	7
3	Instrucciones para Configuración y Despliegue	8
3.1	Configuración de Apache para el balanceo de Carga	8
3.1.1	Configuración de Apache	8
3.1.2	Configuración proxy balancer	9
3.1.3	Explicación de directivas	11
4	Balanceo de carga	14
4.1	Afinidad de sesión	17
5	Pruebas balanceador	18
6	Criterios de evaluación	23

1 Objetivos

En el ámbito de la informática, **la escalabilidad y la alta disponibilidad** son aspectos cruciales a la hora de diseñar y desplegar soluciones software tanto para garantizar un óptimo rendimiento como una grata experiencia de usuario.

Ser capaces de distribuir la carga de trabajo entre varios servidores y asegurar que la aplicación pueda mantener un alto volumen de trabajo sin afectar de manera crítica en su funcionamiento es imprescindible.

- **La escalabilidad** hace referencia a la capacidad de una aplicación para ajustarse y crecer según las demandas cambiantes de carga.

Esto puede lograrse de dos maneras:

- Escalabilidad vertical: consiste en **aumentar los recursos** de un solo servidor o máquina para **mejorar su capacidad** de procesamiento y manejar una mayor carga de trabajo. Esto se logra mediante la adición de más poder de procesamiento, como más CPU, memoria RAM, almacenamiento o incluso tarjetas gráficas (GPU) si es necesario.
 - Escalabilidad horizontal: Consiste en **agregar más servidores** o instancias de una aplicación para distribuir la carga de trabajo y aumentar la capacidad general del sistema. En lugar de mejorar un solo servidor, la infraestructura se extiende mediante la adición de múltiples servidores o nodos que trabajan de manera conjunta.
- **La alta disponibilidad** se refiere al diseño de sistemas capaces de seguir funcionando correctamente, incluso en situaciones donde uno o varios de sus componentes fallen. Este enfoque tiene como objetivo minimizar los períodos de inactividad, asegurando que el servicio esté disponible para los usuarios en todo momento, sin interrupciones significativas, lo cual es crucial en entornos donde la continuidad del servicio es esencial.

El objetivo de esta práctica es implementar una solución de alta disponibilidad y escalabilidad para una aplicación Django utilizando un clúster de servidores, un balanceador de carga y sticky sessions.

- **Cluster** Un clúster es un **conjunto de servidores que trabajan de forma conjunta** para proporcionar un servicio **más robusto y escalable**. Estos servidores están interconectados y pueden compartir recursos como bases de datos, almacenamiento y aplicaciones, **distribuyendo el trabajo** entre ellos para mejorar el rendimiento y la disponibilidad del sistema. En un clúster, si uno de los servidores falla, el sistema sigue funcionando gracias a la redundancia de los otros servidores, lo que garantiza alta disponibilidad. Los clústeres permiten **escalar horizontalmente** la infraestructura, añadiendo más servidores a medida que la demanda aumenta.
- **Balanceador de carga** herramienta o componente que **distribuye el tráfico** de las solicitudes de los usuarios entre varios servidores o instancias de una aplicación. El objetivo es **evitar** que un solo **servidor se sobrecargue** con demasiadas solicitudes, optimizando el uso de los recursos disponibles y mejorando el rendimiento general del sistema. El balanceo de carga también contribuye a la alta disponibilidad, ya que, si un servidor se cae, el balanceador puede redirigir automáticamente el tráfico a las instancias disponibles, manteniendo el servicio operativo sin interrupciones.

Existen diferentes métodos de balanceo

- Round-robin: Las solicitudes entrantes se distribuyen de manera secuencial entre los servidores disponibles, sin tener en cuenta su carga actual.
 - least connections: El balanceador de carga redirige las solicitudes al servidor con el menor número de conexiones activas.
- **Sticky Sessions** Sesiones persistentes, técnica utilizada en sistemas distribuidos para garantizar que las solicitudes de un mismo usuario siempre sean enviadas al mismo servidor durante toda su sesión. Esto es importante en aplicaciones donde el estado de la sesión, como la autenticación o la información de carrito de compras, se almacena localmente en el servidor.

2 Descripción del Proyecto

Una de las características principales que deben cumplir los sistemas distribuidos es la transparencia.

La transparencia 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 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

En esta práctica vamos a partir de la aplicación desarrollada en las prácticas pasadas (sistema de votación electrónico diseñado para crear/borrar/listar votos).

Este proyecto tiene como objetivo configurar y desplegar la aplicación Django en un entorno de clúster, utilizando Apache como balanceador de carga para distribuir las solicitudes entre varias instancias de la aplicación.

El propósito es mejorar la disponibilidad, escalabilidad y rendimiento de la aplicación, al asegurar que el tráfico se distribuya de manera eficiente entre varias máquinas virtuales (VMs).

2.1 Propósito

El sistema tiene los siguientes objetivos:

- Desplegar una aplicación Django en un entorno escalable y distribuido.
- Configurar un balanceador de carga empleando Apache
- Mejorar la resiliencia, garantizar la alta disponibilidad y la redundancia.

2.2 Componentes

El proyecto utilizará las tres máquinas virtuales creadas en las prácticas anteriores.

- **Máquina virtual 1 - Base de datos PostgreSQL y Balanceador de carga:**

Propósito: Esta máquina virtual además de alojar la base de datos PostgreSQL y ejecutar una instancia de la aplicación, también se encargará de distribuir las solicitudes de los usuarios entre las instancias de Django, actuando como un balanceador de carga.

- Componentes:

- * **Sistema Operativo:** Ubuntu 24.04.
- * **PostgreSQL:** Base de datos relacional que almacena los datos de la aplicación Django.
- * **Apache HTTP Server:** Framework de desarrollo web para ejecutar la aplicación.
- * **Django:** Framework de desarrollo web para ejecutar la aplicación.
- * **Gunicorn:** Servidor WSGI para ejecutar la aplicación Django.

- **Máquina virtual 2 - Instancia Django:**

Propósito: Esta máquina ejecutará una instancia de la aplicación Django, y se encargará de servir las solicitudes utilizando Gunicorn como servidor WSGI.

- Componentes:

- * **Sistema Operativo:** Ubuntu 24.04.
- * **Django:** Framework de desarrollo web para ejecutar la aplicación.
- * **Gunicorn:** Servidor WSGI para ejecutar la aplicación Django.

- **Máquina virtual 3 - Instancia Django:**

Propósito: Equivalente a la máquina virtual 2.

2.3 Flujo General

Este flujo cubre las interacciones clave entre el cliente, el balanceador de carga, las instancias de Django y la base de datos PostgreSQL

1. **El cliente realiza una solicitud HTTP:** Un cliente (como un navegador web o una API) hace una solicitud HTTP a la aplicación Django, por ejemplo, accediendo a la página web.
2. **El balanceador de carga (Apache) recibe la solicitud:** El balanceador de carga se encarga de distribuir las solicitudes entre las dos instancias de Django, que están corriendo en las VMs 1, 2 y 3.
3. **Apache redirige la solicitud a la instancia de Django:** Apache reenvía la solicitud a una de las instancias de Gunicorn según su configuración de balanceo de carga, por ejemplo, a la VM 2.
4. **Gunicorn recibe la solicitud en la instancia Django:** En la VM 2, Gunicorn, que está corriendo la aplicación Django, recibe la solicitud HTTP y la pasa a Django a través de la interfaz WSGI.
5. **Django procesa la solicitud:** Django maneja la lógica de la aplicación y procesa la solicitud recibida.
6. **Django realiza una consulta a la base de datos PostgreSQL (VM 1):** Si la solicitud requiere acceder a la base de datos, Django usa el cliente de PostgreSQL.
7. **Procesamiento de respuesta:** PostgreSQL procesa la consulta y devuelve los datos, Django procesa los datos y genera la respuesta, Gunicorn devuelve la respuesta a Apache, Apache envía la respuesta al cliente.

3 Instrucciones para Configuración y Despliegue

Partiendo de las tres máquinas virtuales indicadas con anterioridad, todas ellas ejecutadas e iniciadas.

En esta sección describimos como configurar el cluster en el entorno de los laboratorios. Antes de comenzar, revertir los cambios de la P2, en settings.py, cambiando

```
SESSION_ENGINE=django.contrib.sessions.backends.db
```

```
SESSION_ENGINE=django.contrib.sessions.backends.cache.
```

Con la opción **db** las variables de sesión se almacenan en la base de datos (tabla `django_session`). Esta opción garantiza la persistencia de las variables entre los diferentes nodos del cluster, pero es más lenta, ya que cada acceso requiere una consulta a la base de datos.

Con la opción **cache** las sesiones se almacenan en el sistema de caché (por defecto la memoria del ordenador). Es mucho más rápida, pero menos persistente. Si se reinicia el servidor de caché se pierden las sesiones y la sesión no se puede compartir entre diferentes nodos.

3.1 Configuración de Apache para el balanceo de Carga

Cuando tienes una **aplicación Django** que deseas **desplegar en un cluster** de servidores, **cada nodo** del cluster **ejecutará una instancia de tu aplicación**.

Para **distribuir las peticiones** entre estos servidores de manera eficiente, utilizas un **balanceador de carga** como el que se configura en Apache con **mod_proxy_balancer**.

3.1.1 Configuración de Apache

Primero, aseguramos que Apache tiene activos todos los módulos necesarios para habilitar el balanceo de carga:

- **mod_proxy.**: Proporciona las funciones de proxy inverso.
- **mod_proxy_balancer.**: Permite la configuración del balanceo de carga.
- **mod_ssl.**: Si el entorno requiere HTTPS.

```
1      sudo a2enmod proxy proxy_balancer proxy_http  
      ↪ headers lbmethod_byrequests rewrite  
2      sudo systemctl restart apache2
```

3.1.2 Configuración proxy balancer

En este paso, se crea el archivo de configuración de Apache para gestionar el balanceo de carga.

Este archivo especifica qué servidores componen el cluster y cómo deben ser gestionadas las solicitudes. La configuración básica es la siguiente:

```
sudo nano /etc/apache2/sites-available/000-default.conf
```

```
1 ProxyRequests Off
2
3 <Proxy balancer://SI2Cluster>
4     # Miembros del cluster (instancias de la app Django)
5     BalancerMember http://ip:18000 route=Instance01
6     BalancerMember http://ip:28000 route=Instance02
7     BalancerMember http://ip:38000 route=Instance03
8
9     # Habilitar sticky session con la cookie ROUTEID
10    ProxySet stickysession=ROUTEID
11 </Proxy>
12
13 Header add Set-Cookie
14     ↪ "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/"
15     ↪ env=BALANCER_ROUTE_CHANGED
16
17 # Configurar la ubicacion para balanceo de carga
18 <Location /Pibase>
19     Require all granted
20     ProxyPass balancer://SI2Cluster
21     ProxyPassReverse balancer://SI2Cluster
22 </Location>
23
24 # Configuración del administrador del balanceador de carga
25 <Location /balancer-manager>
26     SetHandler balancer-manager
27 </Location>
28
29 # Configuración para manejar las redirecciones de Django
30     ↪ correctamente
31 RewriteEngine On
```

```
31 #Redirecciones con /P1base se manejen correctamente
32 RewriteCond %{REQUEST_URI} !~/P1base
33 RewriteCond %{REQUEST_URI} !~/balancer-manager
34 RewriteRule ^(.*)$ /P1base$1 [R,L]
35
36 # Ajustar encabezado Location para evitar afectaciones
37 Header edit Location ~/(?!balancer-manager)([~/])
    ↪ /P1base/$1
38
39
40 # Definir el nombre del servidor (ajustalo segun tu
    ↪ configuracion)
41 ServerName XX.YY.ZZ.II #La ip de tu servidor apache
```

3.1.3 Explicación de directivas

- ProxyRequests Off

Desactiva las solicitudes de proxy explícitas, es decir, desactiva la capacidad de Apache para actuar como un proxy directo (enviar solicitudes a otros servidores fuera del contexto del balanceador de carga). Al desactivarlo, Apache solo funciona como proxy inverso, lo que es lo adecuado en este caso, ya que se usará como un balanceador de carga entre los nodos del clúster.

- Header add Set-Cookie:

Esta directiva añade una cookie llamada ROUTEID a las respuestas enviadas al cliente. ROUTEID es la cookie que se utilizará para gestionar la afinidad de sesión (sticky session).

Esta cookie se establecerá con el valor de la variable de entorno:

BALANCER_WORKER_ROUTE

Que contiene la información del servidor de destino al que se dirigió la solicitud (en este caso, las instancias del clúster).

- BalancerMember:

Define las instancias del clúster. Aquí, hay dos instancias de servidores de aplicaciones Django (Instance01 y Instance02), cada una con su respectiva dirección IP y puerto (`http://XX.YY.ZZ.II:P1` y `http://XX.YY.ZZ.II:P2`).

`route=Instance0X`: Define un identificador de ruta para este servidor en particular. El balanceador de carga usará este identificador para asignar las solicitudes a esta instancia de forma persistente.

- ProxySet `stickysession=ROUTEID`:

Configura el balanceador para utilizar la cookie `ROUTEID` para la afinidad de sesión (sticky session). Si un usuario ya tiene una sesión activa, la cookie `ROUTEID` asegura que las solicitudes del mismo usuario se dirijan siempre a la misma instancia del clúster.

- `<Location /P1base>` Este bloque define las reglas para las URL que deben ser balanceadas por el proxy.

`ProxyPass balancer://SI2Cluster`: Aquí, Apache redirige las solicitudes que lleguen al clúster `SI2Cluster` para balancear la carga entre las instancias.

`ProxyPassReverse balancer://SI2Cluster`: Esto asegura que las respuestas de los servidores del clúster sean adecuadamente redirigidas de vuelta al cliente.

- `<Location /balancer-manager>`

Proporciona acceso a una interfaz de administración del balanceador de carga, donde puedes monitorear el estado del balanceo de carga y la salud de las instancias del clúster.

- RewriteEngine

Maneja las redirecciones en el servidor Apache para asegurarse de que todas las solicitudes y respuestas de la aplicación Django estén correctamente adaptadas al balanceador, la directiva `RewriteCond` establece una condición que debe cumplirse para que la regla de reescritura siguiente se aplique. En este caso, la condición especifica que la regla solo se aplicará a las solicitudes cuya URI (la

parte de la URL después del dominio) no comience con /P1base o /balancer-manager.

Para confirmar que el fichero de configuración esta correctamente modificado podeis realizar varias comprobaciones:

- `sudo apachectl configtest` (Comprobación de la sintaxis del fichero de configuración)
- `journalctl -xeu apache2.service` (Comprobación de los logs del servicio de apache)
- `sudo tail -n 50 /var/log/apache2/error.log` (Comprobación de los logs de error de apache)

Una vez aplicadas las configuraciones y desplegado el balanceador de apache, podrá acceder a la consola del balanceador accediendo a la url:

localhost:18080/balancer-manager.

Debería aparecer algo similar a lo mostrado a continuación.

Server Version: Apache/2.4.58 (Ubuntu)
 Server Built: 2024-10-02T12:40:51
 Balancer changes will NOT be persisted on restart.
 Balancers are inherited from main server.
 ProxyPass settings are inherited from main server.

LoadBalancer Status for [balancer://si2cluster](#) [p2c258133_si2cluster]

MaxMembers	StickySession	DisableFailover	Timeout	FailoverAttempts	Method	Path	Active
3 [3 Used]	ROUTEID	Off	0	2	byrequests	/P1base	Yes

Worker URL	Route	RouteRedir	Factor	Set	Status	Elected	Busy	Load	To	From
http://10.0.2.2:18000	Instance01		1.00	0	Init Ok	0	0	0	0	0
http://10.0.2.2:28000	Instance02		1.00	0	Init Ok	0	0	0	0	0
http://10.0.2.2:38000	Instance03		1.00	0	Init Ok	0	0	0	0	0

Apache/2.4.58 (Ubuntu) Server at 127.0.0.1 Port 18080

Figure 1: Balanceador de carga de apache

Ejercicio 1: incluya en la memoria de la practica pruebas de que se ha configurado y ejecutado el balanceador de carga de apache. Se debe incluir capturas de la ejecucion `sudo systemctl status apache2` así como `sudo apachectl configtest` junto capturas de la interfaz de administracion.

Ejercicio 2: Con unicamente la instancia de la MV1 en ejecución, acceda por el balanceador a la aplicación, describa los valores que se muestran por la consola del balanceador y el redireccionamiento adjuntando una captura del balanceador.

4 Balanceo de carga

Para este ejercicio vamos a realizar unas modificaciones en la aplicación. Debemos añadir un nuevo campo en los votos de manera que también quede registrado la instancia desde la que se guardo. Para ello vamos a modificar el modelo de votos añadiendo el campo instancia.

```
instancia = models.CharField(max_length=24, default=1)
```

Luego, configuraremos las funciones encargadas de registrar un nuevo voto para que tengan en cuenta este nuevo valor. Para ello vamos a utilizar la propiedad de `hostname`, importamos `socket` y asignamos el valor.

```
voto_data['instancia'] = socket.gethostname()
```

El script utilizado para la creación de máquinas está configurado de manera predeterminada para asignar a todas las máquinas el nombre de la última máquina creada.

Para asegurarnos de que podamos identificar cada máquina, modificar el archivo `/etc/hostname` en cada máquina y asignar un nombre único a cada una, podeis utilizar el comando `vi` o `nano` para ello, recuerda usarlos con permisos de administrador.

El archivo `/etc/hostname` contiene el nombre de la máquina que el sistema operativo utilizará para identificarla dentro de la red. Para cambiar el nombre de host, se debe editar este archivo y reemplazar el valor actual por el nombre deseado. Es importante elegir un nombre que permita distinguir claramente cada máquina. Después de modificar el archivo, reiniciar la máquina para que el nuevo `hostname` entre en vigor

y se aplique correctamente. De esta forma, cada máquina tendrá un hostname único y podrá ser identificada sin problemas dentro de la red.

El siguiente paso será modificar el html de get votos para que muestre este nuevo atributo. Para ello añade el voto.instancia al bucle del template.

Con todo ello, cuando registremos un voto desde cada instancia se guardará y mostrará desde que instancia se guardo el voto:

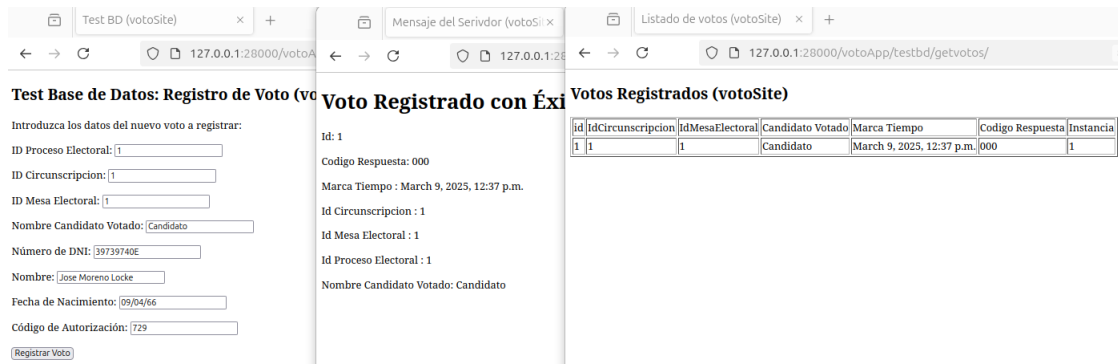


Figure 2: Atributo Instancia

Podeis comprobar que se guarda correctamente accediendo a la base de datos.

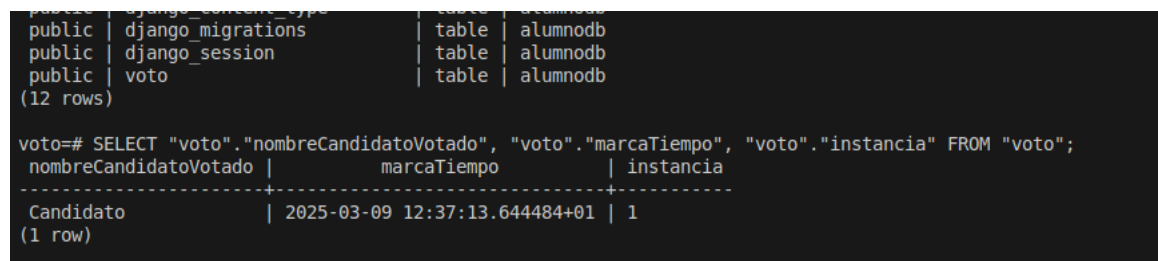


Figure 3: Atributo Instancia

Ejercicio 3: Pruebe a registrar un voto individualmente en cada instancia. Para ello, desde la consola de balanceo, fuerce el uso de una única instancia, compruebe que los votos creados con una instancia, son accesibles desde el resto de instancias.

Los pasos para este ejercicio serían los siguientes:

- desactiva la instancia uno desde el balanceador.

- Insertas el voto en otra instancia.
- Compruebas que el voto se ve correctamente en la instancia 1

← → ↻

localhost:18080/balancer-manager/

☆

Load Balancer Manager for localhost

Server Version: Apache/2.4.58 (Ubuntu)
Server Built: 2024-10-02T12:40:51
Balancer changes will NOT be persisted on restart.
Balancers are inherited from main server.
ProxyPass settings are inherited from main server.

LoadBalancer Status for [balancer:/si2cluster](#) [p2c258133_si2cluster]

MaxMembers	StickySession	DisableFallover	Timeout	FailoverAttempts	Method	Path	Active
3 [3 Used]	ROUTEID	Off	0	2	byrequests/P1base		Yes

Worker URL	Route	RouteRedir	Factor	Set	Status	Elected	Busy	Load	To	From
http://10.0.2.2:18000	Instance01		1.00	0	Init Dis	1	0	100	559	0
http://10.0.2.2:28000	Instance02		1.00	0	Init Ok	1	0	100	612	2.1K
http://10.0.2.2:38000	Instance03		1.00	0	Init Dis	2	0	100	1.1K	4.3K

Edit worker settings for [http://10.0.2.2:38000](#)

Load factor:	<input type="text" value="1.00"/>																		
LB Set:	<input type="text" value="0"/>																		
Route:	<input type="text" value="Instance03"/>																		
Route Redirect:	<input type="text"/>																		
Status:	<table><thead><tr><th>Ignore Errors</th><th>Draining Mode</th><th>Disabled</th><th>Hot Standby</th><th>Hot Spare</th><th>Stopped</th></tr></thead><tbody><tr><td>On <input type="radio"/></td><td>On <input type="radio"/></td><td>On <input checked="" type="radio"/></td><td>On <input type="radio"/></td><td>On <input type="radio"/></td><td>On <input type="radio"/></td></tr><tr><td>Off <input checked="" type="radio"/></td><td>Off <input checked="" type="radio"/></td><td>Off <input type="radio"/></td><td>Off <input checked="" type="radio"/></td><td>Off <input checked="" type="radio"/></td><td>Off <input checked="" type="radio"/></td></tr></tbody></table>	Ignore Errors	Draining Mode	Disabled	Hot Standby	Hot Spare	Stopped	On <input type="radio"/>	On <input type="radio"/>	On <input checked="" type="radio"/>	On <input type="radio"/>	On <input type="radio"/>	On <input type="radio"/>	Off <input checked="" type="radio"/>	Off <input checked="" type="radio"/>	Off <input type="radio"/>	Off <input checked="" type="radio"/>	Off <input checked="" type="radio"/>	Off <input checked="" type="radio"/>
Ignore Errors	Draining Mode	Disabled	Hot Standby	Hot Spare	Stopped														
On <input type="radio"/>	On <input type="radio"/>	On <input checked="" type="radio"/>	On <input type="radio"/>	On <input type="radio"/>	On <input type="radio"/>														
Off <input checked="" type="radio"/>	Off <input checked="" type="radio"/>	Off <input type="radio"/>	Off <input checked="" type="radio"/>	Off <input checked="" type="radio"/>	Off <input checked="" type="radio"/>														

Submit

Apache/2.4.58 (Ubuntu) Server at localhost Port 18080

Figure 4: Desactivar instancias 1 y 3

← → ↻

localhost:18080/P1base/votoApp/voto/

☆

Voto Registrado con Éxito (votoSite)

Id: 1
Codigo Respuesta: 000
Marca Tiempo : March 1, 2025, 9:02 p.m.
Id Circunscripcion : 0
Id Mesa Electoral : 0
Id Proceso Electoral : 0
Nombre Candidato Votado: Candidato

Figure 5: Voto en instancia 2

4.1 Afinidad de sesión

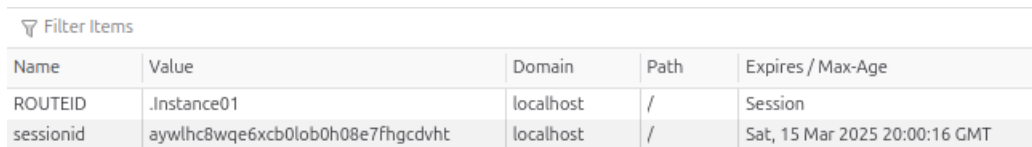
El objetivo de este ejercicio es probar cómo el balanceador de carga Apache maneja la afinidad de sesión utilizando la directiva ProxySet stickysession=ROUTEID.

Contexto: El balanceador de carga Apache está configurado para distribuir las solicitudes entre dos instancias de una aplicación Django. La propiedad stickysession se configura con la cookie ROUTEID, que asegura que todas las solicitudes del mismo cliente se direccionen a la misma instancia del servidor, manteniendo la sesión activa incluso si el balanceador de carga distribuye las solicitudes a diferentes servidores.

Se evaluará cómo esta configuración garantiza que todas las solicitudes de un mismo cliente se dirijan a la misma instancia del servidor, manteniendo la sesión activa a través de la cookie ROUTEID.

Antes de comenzar las pruebas, asegúrate de eliminar todas las cookies almacenadas en tu navegador para garantizar que las pruebas se realicen de manera limpia y no haya interferencias con sesiones anteriores.

Se puede comprobar el valor de la cookie accediendo a las herramientas de desarrollador, desde ahí podremos ver el valor de la cookie.



Filter Items				
Name	Value	Domain	Path	Expires / Max-Age
ROUTEID	.Instance01	localhost	/	Session
sessionid	aywlhc8wqe6xcb0lob0h08e7fhgcvht	localhost	/	Sat, 15 Mar 2025 20:00:16 GMT

Figure 6: Cookie afinidad de sesión

5 Pruebas balanceador

Ejercicio 4: Suprimiendo la afinidad de sesión, (eliminando las configuraciones de sticky session) accede a la aplicación a través de la URL del balanceador. Complete el voto con datos de censo correcto, repita los votos hasta que uno falle por la afinidad de sesión.

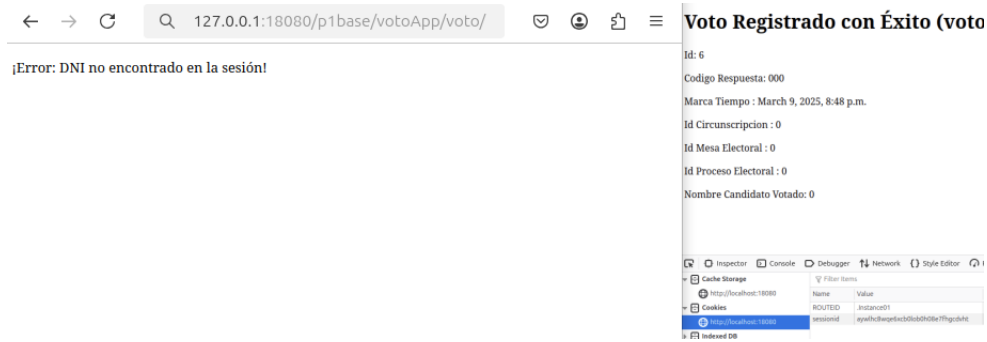


Figure 7: Error sesion

A continuación, verifica que el balanceador de carga esté utilizando ProxySet stickysession=ROUTEID para mantener la afinidad de sesión. La cookie ROUTEID se generará automáticamente, y se usará para asegurar que las solicitudes de este cliente se dirijan siempre a la misma instancia de servidor, de acuerdo con la configuración del balanceador.

Explique en qué y cómo afecta el uso de ProxySet stickySession con respecto a errores.

Ejercicio 5: Verificar que el balanceador de carga maneja el failover correctamente, redirigiendo el tráfico a una nueva instancia del servidor si una de las instancias cae.

- Identificar la instancia con menos elecciones: Utilizando el gestor de balanceo de carga (accediendo a la página `/balancer-manager`), observa el número de "elecciones" o "requests" para cada instancia. Esta métrica indicará cuántas solicitudes ha recibido cada servidor.
- Parar la instancia del clúster con menos elecciones; para ello bastará con parar el demonio de `gunicorn`.
- Realiza peticiones nuevamente accediendo a la URL del balanceador. Accede a la página `/balancer-manager` para verificar que la instancia detenida ha sido marcada como "errónea" y que el balanceador está redirigiendo todas las peticiones a la instancia restante.
- Demuestra borrando cookies y accediendo varias veces que el balanceador dirige a la instancia que no está caída.

Ejercicio 6: Verificar que el balanceador de carga reactiva correctamente una instancia previamente detenida (failback).

- Inicie la instancia detenida. Para ello simplemente vuelva a lanzar el servicio de gunicorn y compruebe en los logs que se inicia correctamente, como medida adicional, prueba a acceder directamente a la instancia desde el navegador sin pasar por el balanceador y compruebe que cargue la aplicación.
- Accede nuevamente a la página /balancer-manager del balanceador de carga y verifica que la instancia previamente caída se haya reactivado y esté ahora en funcionamiento.
- Vuelve a realizar una votación completando los datos del censo como en ejercicios anteriores.
- Observar cómo el balanceador distribuye las solicitudes entre las instancias activas. Si el failback está funcionando correctamente, la carga debe distribuirse entre las instancias disponibles, incluyendo la recién reactivada.
- Incluye las evidencias del proceso de reactivación de la instancia, así como cualquier cambio visible en la consola del balanceador de carga.

Ejercicio 7: Verificar el comportamiento del balanceador y el proceso de failover en el transcurso de una sesión activa.

- Abre un navegador y comienza una votación completando el formulario.
- Justo cuando la pantalla pida completar la información sobre el censo, observa la instancia que ha procesado la solicitud y detén esa instancia.
- Completar los datos del censo de modo que el voto fuera válido, y enviar la petición.
- Observar la instancia del cluster que procesa el voto, y razonar las causas por las que se rechaza la petición de registro de voto.

Ejercicio 8: Pruebas de carga con JMeter

- Abre el archivo de pruebas P2.jmx y modifica el ciclo de pruebas para realizar 1000 pruebas en un solo hilo contra la IP del clúster y la nueva URL de la aplicación (Asegúrate de eliminar cualquier voto previo antes de iniciar el ciclo de pruebas para evitar interferencias).
- Limpia los datos de apache de forma que ambas instancias empiecen con 0 datos.
- Después de ejecutar las pruebas, verifica cómo se distribuyen los votos entre las instancias del clúster.
- deduce qué algoritmo de reparto parece estar utilizando el balanceador de carga. Algunas preguntas clave a responder son:
 - ¿El balanceador distribuye las solicitudes de manera equitativa?
 - ¿Hay alguna instancia que reciba significativamente más o menos solicitudes que otras?
 - ¿El balanceador parece seguir un algoritmo de round-robin o un algoritmo de afinidad de sesión basado en la cookie ROUTEID?
- Incluye las evidencias de los resultados obtenidos en JMeter y cualquier comentario sobre el comportamiento observado durante las pruebas.

Material a entregar: Se debe subir a Moodle un fichero en formato zip que contenga la memoria (en formato pdf) respondiendo a todos los ejercicios y cuestiones planteadas en esta guía junto con los ficheros de configuración y el código utilizada para implementar los diferentes proyectos. Se considerará que el código funciona si lo hace en los ordenadores del laboratorio. El código entregado en Moodle será el evaluado, bajo ninguna condición se evaluará el código existente en los diversos repositorios.

6 Criterios de evaluación

Para aprobar es necesario haber implementado el balanceador de carga satisfaciendo los ejercicios 3, 5 y 6. En caso de no aprobar esos ejercicios, la nota máxima de la práctica será de un 4.9.

Ejercicio	Puntuacion
1	0.5
2	0.5
3	1.5
4	1.0
5	1.5
6	1.5
7	1.5
8	1.0
Memoria	1.0