

DESPLIEGUE DE APLICACIONES WEB
TÉCNICO EN DESARROLLO DE APLICACIONES WEB

Gestión de despliegues

ÍNDICE

/ 1. Introducción y contextualización práctica	3
/ 2. Gestión de usuarios: dominios de seguridad para la autenticación	4
/ 3. Administración de sesiones: sesiones persistentes	5
/ 4. Administración de aplicaciones web en el servidor de aplicaciones	6
4.1. Archivos de registro y filtro de solicitudes	6
4.2. Redirección de peticiones a servidor de aplicaciones con node.js	7
4.3. Redirección de peticiones a servidor de aplicaciones con Apache	8
/ 5. Caso práctico 1: “Configuración de válvulas en Tomcat”	9
/ 6. Despliegue de aplicaciones	10
6.1. Jenkins	11
6.2. Foreverjs	12
/ 7. Caso práctico 2: “Foreverjs”	13
/ 8. Resumen y resolución del caso práctico de la unidad	14
/ 9. Bibliografía	14

OBJETIVOS

Gestionar usuarios y sesiones.

Realizar configuraciones avanzadas en los servidores de aplicaciones web.

Conocer diferentes técnicas y herramientas que nos ayudarán en el despliegue de aplicaciones.

/ 1. Introducción y contextualización práctica

En este tema, comentaremos qué aspectos son necesarios a la hora de desplegar aplicaciones webs: empezaremos con la gestión de usuarios y la configuración de sesiones persistentes y terminaremos con la creación de registros personalizados para analizar el tráfico y las peticiones que recibe nuestro servidor.

También realizaremos configuraciones en los diferentes servidores de aplicaciones para cooperar con un servidor web.

Por último, hablaremos de herramientas que nos ayudan en la gestión de despliegues y en la ejecución de nuestras aplicaciones.

Planteamiento del caso práctico inicial

A continuación, vamos a plantear un caso a través del cual podremos aproximarnos de forma práctica a la teoría de este tema.

Escucha el siguiente audio donde planteamos la contextualización práctica de este tema, encontrarás su resolución en el apartado Resumen y Resolución del caso práctico.



Fig. 1. Administración de servidores de aplicaciones web.



Audio Intro. "Servidor de prueba"

<https://bit.ly/2XeePNk>





/ 2. Gestión de usuarios: dominios de seguridad para la autenticación

A la hora de diseñar una aplicación web, ya sea por requisito de esta o porque queramos implementar **una capa más de seguridad**, se deben establecer mecanismos para autenticar usuarios contra una base de datos.

Normalmente son las páginas o las aplicaciones webs las que realizan este proceso de comprobación y autenticación contra una base de datos, ya sea MySQL, PostgreSQL, MongoDB, etc., aunque existen también otras bases de datos especializadas en la gestión de usuarios. Estas bases de datos utilizan el **protocolo LDAP**. El protocolo LDAP (*Lightweight Directory Access Protocol*) es un mecanismo que permite el **inicio de sesión basado en dominios**. Como veremos en el tema 13, este protocolo se encuentra en la capa de aplicación del modelo OSI y ofrece la posibilidad de crear un **sistema de autenticación centralizada**.

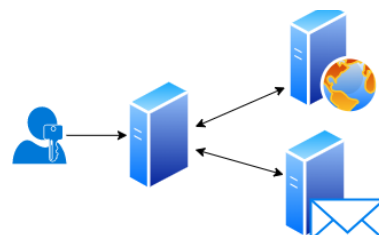


Fig. 2. Inicio de sesión SSO diferentes servicios usando LDAP - aplicaciones web.

Utilizar un sistema de autenticación **centralizado** nos facilita la **gestión de usuarios y contraseñas** en un entorno laboral. Por ejemplo, una empresa podría utilizar un servidor LDAP para que los trabajadores inicien sesión en diferentes servicios utilizando una única contraseña: el correo electrónico, el CRM, la aplicación de control horario, o la de cualquier otro servicio.

Podemos implementar esta característica en nuestras aplicaciones web utilizando conectores en **Tomcat** o con funciones que conecten con el servidor LDAP en Node.js

```
<Realm className="org.apache.catalina.realm.JNDIRealm"
  debug="99"
  connectionURL="ldap://servidor.ldap:389/"
  userPattern="{0}" />
```

Código 1. Ejemplo de conector LDAP en Apache Tomcat

<https://www.viralpatel.net/implement-ldap-authentication-in-tomcat-jboss-server-for-java-app/>

Para servidores de aplicaciones sobre Node.js, encontraremos el paquete de npm "ldap-authentication" que permite utilizar LDAP de forma sencilla.

```
let authenticated = await authenticate({
  ldapOpts: { url: 'ldap://ldap.forumsys.com' },
  userDn: 'uid=gauss,dc=example,dc=com',
  userPassword: 'password',
})
```

Código 2. Ejemplo de autenticación LDAP en Node.js con ldap-authentication -

<https://www.npmjs.com/package/ldap-authentication>



Audio 1. "Single Sign On"

<https://bit.ly/3hTdR7z>





/ 3. Administración de sesiones: sesiones persistentes

El **protocolo HTTP** es un protocolo sin estado (no orientado a conexión), ya que originalmente no se planteó para almacenar las acciones que un visitante realiza en la navegación por el servidor.

Por este motivo, es necesario un mecanismo que gestione **la interacción entre el cliente y el servidor** y permita, además, guardar información sobre la **navegación**, dicho con otras palabras: **crear sesiones**.

Para poder almacenar información de la sesión se utilizan, entre otros mecanismos, las famosas **cookies**. En las cookies se guarda información relativa a la navegación, como podría ser:

- **ID de sesión** (que permitiría seguir navegando con normalidad en caso de reconexión o un posible reinicio del servidor)
- **Datos relativos al login del usuario**

Accept	text/html,application/xhtml+xml, image/jxr, */*
Accept-language	es-ES
Usar-agent	Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Geko
Accept-encoding	gzip, deflate
Host	192.168.1.153:8080
Connection	Keep-Alive
Cookie	JSESSIONID=587715869E4A7FD3B40F9162BC93EA6A

Tabla 1. Ejemplo de cookie de sesión

En Apache Tomcat podemos configurar **sesiones persistentes**. Esta implementación permite almacenar **sesiones activas**, pero desconectadas, a disco. Esto provocaría que, al restaurar la conexión, la navegación siga en el mismo punto que se dejó. Un ejemplo es cuando las webs almacenan los productos que guardamos en los “carritos” al realizar compras por internet.

Con el fin de almacenar la información de las **sesiones inactivas** se pueden aplicar configuraciones para realizar operaciones de escritura en disco y, de esta forma, reducir el uso de la memoria.

El siguiente fragmento se corresponde a la configuración de sesiones persistentes en Apache Tomcat. Esta configuración debe realizarse sobre el fichero **server.xml**

```
<!-- PersistentManager: Uncomment the section below to test Persistent
Sessions.
<Manager className="org.apache.catalina.session.PersistentManager"
debug="0"
saveOnRestart="true"
maxActiveSessions="-1"
minIdleSwap="-1"
maxIdleSwap="-1"
maxIdleBackup="-1">
  <Store className="org.apache.catalina.session.FileStore"/>
</Manager>
```



En el siguiente enlace a la web oficial de Apache Tomcat encontraremos más información relativa a las sesiones persistentes:

<https://tomcat.apache.org/tomcat-5.5-doc/config/manager.html>

/ 4. Administración de aplicaciones web en el servidor de aplicaciones

4.1. Archivos de registro y filtro de solicitudes

Como hemos visto, Apache Tomcat almacena registros de los **eventos del servidor y de las peticiones de los clientes**. Además, nos ofrece la posibilidad de crear filtros y generar registros personalizados según las necesidades que puedan surgir. Para entenderlo con facilidad pondremos un ejemplo:

Implementamos una aplicación web con Apache Tomcat en la que, a ciertas horas, el tráfico del servidor web aumenta de manera notable. Imaginemos que nos interesaría saber qué páginas son las que más visitas reciben y desde qué países, con el objetivo de afinar más en las promociones que queremos llevar a cabo por redes sociales.

```
-Xms512M
14-Jul-2020 16:40:58.921 INFORMACIÓN [main] org.apache.catalina.startup.VersionLoggerListener.log
-Xmx1024M
14-Jul-2020 16:40:58.921 INFORMACIÓN [main] org.apache.catalina.startup.VersionLoggerListener.log
-XX:+UseParallelGC
14-Jul-2020 16:40:58.922 INFORMACIÓN [main] org.apache.catalina.startup.VersionLoggerListener.log
-Dignore.endorsed.dirs=
14-Jul-2020 16:40:58.922 INFORMACIÓN [main] org.apache.catalina.startup.VersionLoggerListener.log
-Dcatalina.base=/opt/tomcat
```

Código 4.. Ejemplo de un log de eventos de Apache Tomcat.

Estos datos podemos generarlos a partir de los registros de Tomcat utilizando **válvulas**. Estas válvulas filtran el **tráfico** de entrada al servidor rastreando la petición. Este sistema es propio de Apache Tomcat y no se utiliza en otros servidores webs.

Por defecto, Apache Tomcat ya tiene activada una válvula para generar el **Access log** y tener un acceso de registro en el formato tradicional de un servidor web. Las válvulas se pueden añadir en los ficheros **server.xml** o **context.xml**.

```
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
      prefix="localhost_access_log" suffix=".txt"
      pattern="%h %l %u %t &quot;%r&quot; %s %b" />
```

Código 5. Válvula para crear Access.log en Apache Tomcat (server.xml)

Apache Tomcat clasifica las válvulas en grupos dependiendo de su **finalidad**. Entre los diferentes **tipos** destacan los siguientes:


- Registro de Accesos
- Control de accesos
- Autenticación
- Comprobación de estado de salud del servidor



- Válvula de registro de errores
- Proxies Support


En la web oficial de Apache Tomcat encontraremos un listado completo de las válvulas que existen:

<https://tomcat.apache.org/tomcat-9.0-doc/config/valve.html>



Vídeo 1. "Configuración de válvulas en Tomcat"

<https://bit.ly/317zH0m>



4.2. Redirección de peticiones a servidor de aplicaciones con node.js

El servicio de Apache Tomcat se ejecuta en el **puerto 8080** y, como hemos visto en temas anteriores, se puede modificar para escuchar peticiones por otro puerto. Este apartado explica cómo utilizar un servidor web para actuar como **proxy** y redirigir las peticiones destinadas a un determinado FQDN al servidor correspondiente.

El funcionamiento es parecido al uso de los **Virtual Hosts** para hacer multisiting en Apache; aunque en este caso los servicios pueden estar en el mismo servidor o en otro diferente. El servidor web recibirá peticiones FQDN y las redirigirá al servidor que nosotros queramos usando Virtual Host.

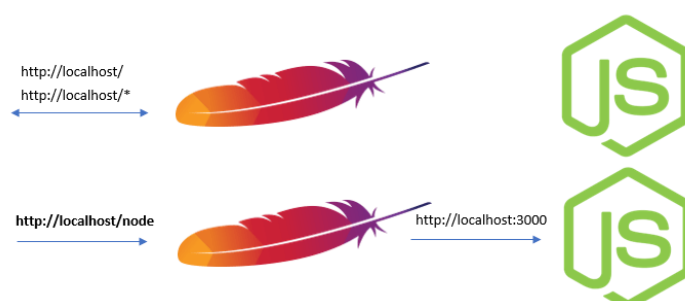


Fig. 3. Funcionamiento de Apache proxy con node.js -
<https://redstapler.co/run-node-js-apache-server/>

Utilizar un proxy para gestionar el tráfico de nuestros servidores de aplicaciones nos aportará una serie de **ventajas**, entre ellas:

- Posibilidad de acceder a varios servidores utilizando una única dirección IP
- Mejora de la seguridad
- Centralización de la autenticación

Para hacer esto posible y que Apache pueda trabajar como un proxy, es necesario activar el módulo **mod_proxy_http**.

```
sudo a2enmod proxy proxy_http  
sudo apache2ctl restart
```

Código 6. Activación del módulo proxy y reinicio del servicio Apache

```
<VirtualHost *:80>
  ServerName appsrv.local
  DefaultType text/html
  ProxyRequests off
  ProxyPreserveHost On
  ProxyPass / http://IP_NODEJS:3000/
  ProxyPassReverse / http://IP_NODEJS:3000/
</VirtualHost>
```

A continuación, creamos un nuevo fichero en la carpeta **/etc/apache2/sites-available/** con la siguiente configuración:

Código 7. VirtualHost de Apache con configuración proxy

En el siguiente enlace encontraremos información ampliada de cómo configurar un servidor web para trabajar de manera conjunta con un servidor de aplicaciones:

<https://www.ionos.com/community/server-cloud-infrastructure/nodejs/set-up-a-nodejs-app-for-a-website-with-apache-on-ubuntu-1604/>

4.3. Redirección de peticiones a servidor de aplicaciones con Apache

Este apartado describe el procedimiento para configurar un servidor web como proxy y enviar las peticiones al

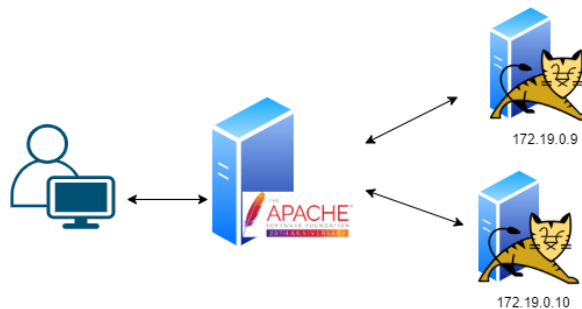


Fig. 4. Apache actuando como proxy

servidor de aplicaciones.

Para explicar el **proceso de configuración** asumiremos que:

- Tenemos un servidor de aplicaciones Apache Tomcat funcionando por el puerto 8080 con la dirección IP 172.19.0.9
- Tenemos un servidor web Apache con la configuración por defecto

IP_APACHE	appsrv.local
IP_APACHE	websrv.local

- En nuestro archivo hosts o en la zona DNS tenemos configurados los FQDN correspondientes (Codigo 8)

*Código 8. Apuntamos al servidor web*

Todas estas configuraciones las realizaremos sobre el servidor web. El primer paso es activar el módulo de Apache

```
sudo a2enmod proxy proxy_http  
sudo apache2ctl restart
```

mod_proxy_http y reiniciar el servicio.

Código 9. Activación del módulo proxy y reinicio del servicio Apache

```
<VirtualHost *:80>  
    DocumentRoot /var/www/html  
    ServerName websrv.local  
</VirtualHost>  
<VirtualHost *:80>  
    ServerName appsrv.local  
    DefaultType text/html  
    ProxyRequests off  
    ProxyPreserveHost On  
    ProxyPass / http://172.19.0.9:8080/  
    ProxyPassReverse / http://172.19.0.9:8080/  
</VirtualHost>
```

Crearemos un site nuevo con la siguiente configuración:

Código 10. Configuración Virtual Host

<https://blog.marcnuri.com/ejecutar-apache-tomcat-y-apache-httpd-en-el-puerto-80-simultaneamente/>

El primer Virtual Host hace referencia al propio servidor web; el segundo especifica los datos del servidor de aplicaciones y la IP a la que se realizará la redirección.

Tras activar el sitio y reiniciar Apache podremos acceder al servidor de aplicaciones a través del servidor web.



Vídeo 2. "Proxy HTTP Apache"

<https://bit.ly/317WtoK>



En el vídeo 2 de la unidad, podemos ver el proceso completo:

/ 5. Caso práctico 1: "Configuración de válvulas en Tomcat"

Planteamiento: Javier está encargándose de gestionar la seguridad de una aplicación web desplegada sobre un servidor Apache Tomcat.

El cliente le ha solicitado que asegure las conexiones y limite únicamente el acceso a la dirección IP de su empresa.



Además, le ha pedido que se establezca un registro de acceso al servidor.

Nudo: ¿de que formas podrá limitarlo?, ¿de que manera establecerá el registro?

```
<Valve className="org.apache.catalina.valves.RemoteHostValve" allow="IP AUTORIZADA  
1 | IP AUTORIZADA 2"/>
```

Desenlace: Javier podrá utilizar las válvulas de Apache Tomcat para limitar el acceso al servidor.

La configuración de la restricción se aplicará sobre el fichero content.xml.

```
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"  
    prefix="localhost_access_log" suffix=".txt"  
    pattern="%h %l %u %t &quot;%r&quot; %s %b" />
```

Código 11. Ejemplo de un log de eventos de Apache Tomcat.

El registro de acceso se puede configurar en el archivo server.xml estableciendo un Access.log

Código 12. Ejemplo de un log de eventos de Apache Tomcat.

/ 6. Despliegue de aplicaciones

El empaquetado de aplicaciones en ficheros **.war** ha facilitado el despliegue de aplicaciones de una manera asombrosa,

Desplegar	
Desplegar directorio o archivo WAR localizado en servidor	
Trayectoria de Contexto (opcional):	<input type="text"/>
Version (for parallel deployment):	<input type="text"/>
URL de archivo de Configuración XML:	<input type="text"/>
URL de WAR o Directorio:	<input type="text"/>
<input type="button" value="Desplegar"/>	
Archivo WAR a desplegar	
Seleccione archivo WAR a cargar	<input type="button" value="Seleccionar archivo"/> Ningún archivo seleccionado
<input type="button" value="Desplegar"/>	

hasta el punto de poder cargar una aplicación en nuestro servidor con tan solo unos pocos clics.

Apache Tomcat incluye una herramienta para desplegar nuevas aplicaciones en el servidor utilizando la interfaz web. Desde aquí podremos subir una aplicación en formato **.war** y tenerla operativa de manera casi instantánea.

Fig. 5. Gestor de aplicaciones Web de Tomcat

En el siguiente enlace podemos descargar una aplicación empaquetada para hacer pruebas de subida mediante el gestor de Tomcat:

```
jar cvf app.war WEB-INF
```

<https://www.middlewareinventory.com/blog/sample-web-application-war-file-download/>



En temas anteriores, vimos cómo se podían utilizar los entornos de desarrollo para **empaquetar aplicaciones**. No obstante, desde la consola de **Ubuntu** también podemos generar un archivo **.war** utilizando el comando **jar**:

```
app1
|--- app.war
|--- index.jsp
|--- WEB-INF
|   |--- classes
|   |--- lib
|   |--- web.xml
```

Código 13. Empaquetado en extensión **.war** utilizando un terminal. <https://www.javatpoint.com/war-file>

Para empaquetar una aplicación deberemos tener un archivo **index.jsp** y un descriptor de despliegue (**web.xml**) almacenados bajo la estructura que ya conocemos.

Código 14.. Estructura aplicación JAVA

Archivo WAR a desplegar	
Seleccione archivo WAR a cargar	<input type="button" value="Seleccionar archivo"/> app.war <input type="button" value="Desplegar"/>

Una vez tengamos nuestros archivos preparados y el código compilado, podemos comprimirlo utilizando el comando **jar cvzf**.

Por último, iniciaremos sesión en el gestor de Tomcat y subiremos el archivo **.war**. Terminaremos pulsando el botón **desplegar**.

Fig. 6. Despliegue de aplicaciones desde la GUI

6.1. Jenkins

En el desarrollo y despliegue de aplicaciones web, cada vez tiene más importancia el término **DevOps**. Este término viene de **Development Operations** y consiste en la integración de los grupos de desarrollo y de los operadores de servicio, que se encargan de adaptar los programas a los sistemas utilizados y optimizar el rendimiento.

Para facilitar esta integración entre equipos de trabajo se utilizan herramientas para mejorar la **coordinación y la comunicación**. Entre ellas, encontramos Jenkins.

WAR file

The Web application ARchive (WAR) file version of Jenkins can be installed on any operating system or platform that supports Java.

To download and run the WAR file version of Jenkins:

1. Download the [latest stable Jenkins WAR file](#) to an appropriate directory on your machine.
2. Open up a terminal/command prompt window to the download directory.
3. Run the command `java -jar jenkins.war`.
4. Browse to `http://localhost:8080` and wait until the **Unlock Jenkins** page appears.
5. Continue on with the [Post-installation setup wizard](#) below.

Jenkins es un **servidor de automatización desarrollado en Java**. El propósito de esta herramienta es facilitar los



procesos de integración de nuevas partes de código y en despliegues de archivos .war.

```
Java -jar jenkins.war --httpPort=9090
```

En su web oficial encontraremos las instrucciones necesarias para instalar Jenkins. En este caso, seguiremos las



S	W	Nombre	Último Éxito	Último Fallo	Última Duración
		Tomcat	N/D	20 Seg - #2	45 Ms

instrucciones para utilizar el archivo WAR y ejecutarlo directamente sobre Java.

Fig. 7. Instalación de Jenkins desde un archivo WAR - <https://www.jenkins.io/doc/book/installing/>

Jenkins escuchará por el puerto 8080, pero podemos configurar este puerto en la ejecución del programa:

Código 15. Iniciar Jenkins en otro puerto

Fig. 8. Despliegue con Jenkins

Encontraremos más información sobre el despliegue sobre Tomcat en el siguiente enlace:

<https://www.jdev.it/deploying-your-war-file-from-jenkins-to-tomcat/>

6.2. Foreverjs

Una de las **desventajas** de utilizar **node.js** es tener que lanzar manualmente los programas. Esto supone que, si el

```
usuario@srvapp01:~$ sudo npm install forever -g
```

servidor se queda bloqueado o se reinicia, alguien tendrá que estar al tanto de la situación y relanzar el script.

Para solucionar esta debilidad aparece **Forever.js**. Una utilidad del paquete NPM que permite que un **script se ejecute continuamente**.

```
usuario@srvapp01:~$ forever app.js
warn: --minUptime not set. Defaulting to: 1000ms
warn: --spinSleepTime not set. Your script will exit if it does not stay up for at least 1000ms

usuario@srvapp01:~$ forever -c node app.js
warn: --minUptime not set. Defaulting to: 1000ms
warn: --spinSleepTime not set. Your script will exit if it does not stay up for at least 1000ms
Example app listening on port 3000!
```

Esta aplicación se puede utilizar no solo con Node.js, si no con cualquier otro comando especificando el argumento -c.



Código 16. Instalación forever - <https://github.com/foreversd/forever>

```

usuario@srvapp01:~$ forever -c node app.js &
[1] 1109
usuario@srvapp01:~$ warn: --minUptime not set. Defaulting to: 1000ms
warn: --spinSleepTime not set. Your script will exit if it does not stay up for at least 1000ms
Example app listening on port 3000!

usuario@srvapp01:~$ forever list
info: Forever processes running
data: uid command script forever pid id logfile uptime
data: [0] ABZV node app.js 1109 1120 /home/usuario/.forever/ABZV.log 0:0:0:5.991
usuario@srvapp01:~$ forever stop app.js
error: Forever detected script was killed by signal: SIGKILL
info: Forever stopped process:
uid command script forever pid id logfile uptime
[0] ABZV node app.js 1109 1120 /home/usuario/.forever/ABZV.log 0:0:0:11.501
[1]+ Done forever -c node app.js

```

Una vez instalado, podemos lanzar aplicaciones de Node utilizando el comando **forever app.js** o **forever -c node app.js**.

Código 17. Instalación forever - <https://github.com/foreversd/forever>

Para listar y parar los procesos en ejecución podemos utilizar los siguientes comandos:

Código 18. Listado de procesos y parada de un programa ejecutado con forever

En los siguientes enlaces podemos ampliar información y conocer todas las opciones que ofrece el paquete Forever:

<https://www.npmjs.com/package/forever>

<https://github.com/foreversd/forever>

/ 7. Caso práctico 2: “Foreverjs”

Planteamiento: Fernando acaba de recibir una llamada urgente de un cliente que no puede acceder a la aplicación web que le han desarrollado.

Tras revisar el servidor detecta que node.js no está en ejecución.

Lanza el proceso y en pocos minutos tiene resuelta la incidencia, pero no el problema.

```
usuario@srvapp01:~$ sudo npm install forever -g
```

Ahora necesita una solución que levante automáticamente el servicio en caso de reinicio o bloqueo del servidor.

Nudo: ¿qué solución puede implementar para evitar próximas incidencias?

Desenlace: Fernando puede instalar el paquete *forever* que se encargará de ejecutar el programa de manera continua,

```

usuario@srvapp01:~$ nano /etc/rc.local
#!/bin/bash
#forever -c node /home/usuario/app.js &
usuario@srvapp01:~$ sudo chmod +x /etc/rc.local

```

incluso en caso de error o desconexión. La instalación la realizará con el comando *npm install*:

Código 19. Instalación forever - <https://github.com/foreversd/forever>



A continuación, para conseguir que el script se ejecute aunque el servidor se reinicie, añadirá las líneas correspondientes al fichero **rc.local** en Linux. Este fichero permite ejecutar scripts una vez el sistema arranque, de esta manera no tendrá que ejecutarlo de forma manual.

Código 20. Configuración script Rc.local para ejecutar aplicaciones en el arranque del sistema.

A partir de este momento, si el servidor volviese a reiniciar, el script se ejecutará y Fernando podrá estar tranquilo.

/ 8. Resumen y resolución del caso práctico de la unidad

En esta unidad, hemos aprendido los sistemas básicos para **gestionar usuarios y sesiones** en un servidor de aplicaciones web.

También hemos experimentado con los **registros** y de las **válvulas** de Tomcat.

Por otro lado, también hemos visto los procedimientos para conseguir que un servidor web y un servidor de aplicaciones cooperen. Esto es posible haciendo que el servidor web actúe como **proxy**.

Finalmente, hemos hablado de herramientas que facilitan los despliegues como el servidor de automatización **Jenkins**, o **foreverjs** que permite que nuestro programa esté siempre disponible.

Resolución del caso práctico de la unidad

En el caso práctico inicial se plantea la siguiente situación: hemos desplegado una aplicación sobre un servidor que ocasionalmente se reinicia. Al ser un servidor destinado a pruebas, no se utiliza para dar servicios a clientes, por lo que si se reinicia no supone un problema; no obstante, en cada reinicio perdemos la información relativa a la navegación.

La solución sería configurar sesiones persistentes en Apache Tomcat. De esta manera se almacenaría información de la sesión en el servidor y en caso de bloqueos o reinicios, podremos seguir donde lo dejamos.

/ 9. Bibliografía

- B. (2020, 30 marzo). *The Difference Between Active Directory and LDAP*. Recuperado de <https://www.varonis.com/blog/the-difference-between-active-directory-and-ldap/>
- F. (s. f.). *foreversd/forever*. Recuperado de <https://github.com/foreversd/forever>
- Manejo de sesiones | Marco de Desarrollo de la Junta de Andalucía. (s. f.). Recuperado de <http://www.juntadeandalucia.es/servicios/madeja/sites/default/files/historico/1.3.0/contenido-libro-pautas-65.html>
- McClanahan, C. R. (2020, 30 junio). *Apache Tomcat 9 Configuration Reference (9.0.37) - The Valve Component*. Recuperado de <https://tomcat.apache.org/tomcat-9.0-doc/config/valve.html>
- npm: *forever*. (2020, 22 mayo). Recuperado de <https://www.npmjs.com/package/forever>
- npm: *ldap-authentication*. (2020, 13 junio). Recuperado de <https://www.npmjs.com/package/ldap-authentication>
- Regueira, M. (2019, 8 agosto). *¿Qué es LDAP?* Recuperado de <https://www.elgrupoinformatico.com/tutoriales/que-ldap-t74212.html>
- Use LDAP and Active Directory to authenticate Node.js users*. (s. f.). Recuperado de <https://developer.ibm.com/technologies/node-js/tutorials/se-use-ldap-authentication-authorization-nodejs-cloud-application/>
- W. (2013, 30 diciembre). *Deploying your war file from Jenkins to Tomcat* – JDev. Recuperado de <https://www.jdev.it/deploying-your-war-file-from-jenkins-to-tomcat/>

Webgrafía

- <https://www.ietf.org/rfc/rfc2616.txt>
- https://www.incibe.es/extfrontinteco/img/File/intecocert/Formacion/EstudiosInformes/gestion_sesiones_web_seguridad.pdf
- <https://blog.marcnuri.com/ejecutar-apache-tomcat-y-apache-httpd-en-el-puerto-80-simultaneamente/>
- <https://www.oxxus.net/tutorials/tomcat/tomcat-valve>