Universitat Oberta de Catalunya

uoc.edu

Desarrollo de un sistema Single Sign On

Daniel Daher Pérez
Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones
M1.830 TFM-Ad hoc

Antonio González Ciria
06/2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Desarrollo de un sistema Single Sign On*
Nombre del autor:	*Daniel Daher Pérez*
Nombre del consultor/a:	*Antonio González Ciria*
Fecha de entrega (mm/aaaa):	06/2018
Titulación::	*Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones*
Área del Trabajo Final:	*M1.830 TFM-Ad hoc*
Idioma del trabajo:	*Español*
Palabras clave	*SSO, CAS, Alta Disponibilidad*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

Este proyecto tiene como objetivo final la realización de un sistema Single Sign On securizado. Un sistema SSO es un sistema en el cual sólo se tiene que autenticar una vez para poder acceder a una variedad de servicios.

Para realizar este objetivo se utilizará el servidor CAS como medio de autenticación al que accederemos mediante una aplicación. Para autenticarnos el servidor accederá a un servidor OpenLDAP como repositorio externo de usuarios. Esto estará implementado en un entorno de alta disponibilidad en el que existirá un clúster de servidores tras un balanceador de carga que a su vez trabajará como reverse proxy. Todas estas conexiones estarán implementadas mediante el protocolo seguro HTTPS.

Este proyecto se ha llevado a cabo en un entorno Linux y utilizando la herramienta VirtualBox. Como servidor se ha hecho uso de los servidores Apache Tomcat y Apache HTTP.

Se ha conseguido la realización del objetivo de manera segura y cumpliendo en general los objetivos propuestos. Un sistema SSO seguro con un clúster de servidores con balanceo de carga, con base de usuarios en LDAP y con conexiones seguras con HTTPS.

Abstract (in English, 250 words or less):

The final goal of this project is the realization of a secure Single Sign On system. An SSO system is a system in which you only have to authenticate once to access a variety of services.

To achieve this goal, the CAS server will be used as the authentication means that we will access through an application. To authenticate, the server will access an OpenLDAP server as an external user repository. This will be implemented in a high availability environment where there will be a server cluster after a load balancer that will work as a reverse proxy. All these connections will be implemented through the secure HTTPS protocol.

This project has been carried out in a Linux environment and using the VirtualBox tool. As a server, Apache Tomcat and Apache HTTP servers have been used.

The achievement of the objective has been achieved safely and in general meeting the proposed objectives. A secure SSO system with a server cluster with load balancing, user base in LDAP and secure connections with HTTPS.

Índice

1. Introducción

1.1 Contexto y justificación del Trabajo

El objetivo a grandes rasgos de este Trabajo de Fin de Máster es la implantación de un servidor Single Sign-On (SSO).

¿Qué es el Single Sign-On?

El SSO es un procedimiento de autenticación que habilita al usuario para acceder a varios sistemas con una sola instancia de identificación. Si lo traducimos sería algo como “autenticación única”, y viene a representar eso mismo, una forma de autenticación que gracias a sus características, permite el acceso del usuario a una variedad de sistemas o servicios tras un sólo acceso.
En el caso que ocupa este TFM vamos a montar un servidor de SSO en alta disponibilidad protegiendo un clúster de servidores de aplicaciones.

1.2 Objetivos del Trabajo

Sin entrar en mucho detalle, el desarrollo satisfactorio del TFM, se podría dividir en los siguientes objetivos:

- Creación del entorno de trabajo mediante virtualización.
- Creación de servidores de aplicaciones en nuestro entorno.
- Montaje de servidores SSO con CAS. Securización del sistema.
- Implementación de Reverse Proxies. Balanceadores de carga y encargados del login para acceder a las distintas aplicaciones.
- Implementación del sistema en Alta Disponibilidad.
- Distintas funcionalidades dependiendo del usuario que se conecte al servidor.

1.3 Enfoque y método seguido

Para empezar a trabajar habrá que preparar un entorno de trabajo. Esto se realizará mediante una virtualización de un sistema Linux, en este caso utilizaré el software VirtualBox para esta tarea.
Una vez creado el entorno se seguirá con la instalación de los servidores de aplicaciones y la estructura general del sistema.
Primero siempre se instalarán todos los softwares o sistemas que serán necesario tras el transcurso del desarrollo del TFM, los que se puedan en cada momento por supuesto. Una vez instalados seguirá la configuración completa del sistema en el que se esté trabajando para terminar creando la estructura del SSO, conexiones con otros sistemas, securización...
Para pasar de una tarea a otra se comprobará que las anteriores estén completas y, en la medida de lo posible, sin errores. A la hora de instalación de aplicaciones y acceso de los usuarios o grupos se intentará

probar el mayor número posible de combinaciones para comprobar que se respetan los accesos autorizados y los que no lo son.
En el siguiente punto veremos en detalle las tareas individuales a realizar y una pequeña descripción si es necesario.

1.4 Planificación del Trabajo

- Investigación y análisis. Fase de investigación sobre herramientas que serían necesarias para el desarrollo del TFM y sistemas similares cuya información podría ser de ayuda durante el trascurso de este trabajo.

- Montaje del entorno de trabajo. Creación del sistema virtualizado que contendrá nuestro sistema SSO.

- Montaje servidor CAS. Instalación y configuración del servidor CAS

- Montaje de un servidor de aplicaciones. Instalación y configuración del servidor de aplicaciones.

- Desarrollo de app en servidor de aplicaciones. Desarrollo de una aplicación, todavía sin securizar y con un solo perfil, desplegada en el servidor de aplicaciones.

- Securización de app mediante CAS. Se utilizará CAS para securizar la aplicación desplegada.

- Securización de comunicaciones mediante HTTPS.

- Modificaciones de la app para añadir más de un rol. Daremos el siguiente paso y permitiremos a la aplicación detectar más de un rol de usuario.

- Implementación de repositorio de usuarios externos mediante OpenLDAP. Utilizaremos OpenLDAP como repositorio externo de usuarios.

- Implementación de jerarquía de usuarios y grupos. Se utilizarán usuarios y grupos de acceso al sistema, el grupo al que pertenezca el usuario determinará el rol que posee y por lo tanto los recursos a los que tiene acceso.

- Configuración de servidores CAS con OpenLDAP. Conectaremos el sistema OpenLDAP que hemos creado con nuestro servidor CAS para autenticar y autorizar a los usuarios.

- Instalación de Reverse Proxies. Con el uso de Reverse Proxies sólo necesitaremos un dominio para acceder a nuestros servidores. También serán utilizados como balanceadores de carga.

- Implementación mecanismo de alta disponibilidad CAS

- Implementación de clúster de servidores de aplicaciones

1.5 Breve sumario de la estructura

En la redacción de la parte central de esta memoria se dividirá el contenido en cuatro partes diferenciadas:

- Implementación sistema CAS. Esta parte tratará de la creación del entorno, de la implementación del sistema CAS principal y de la aplicación con la que nos conectaremos para comprobar su funcionamiento.
- Creación y conexión al sistema OpenLDAP, desde el servidor CAS, como almacén externo de usuarios.
- Securización de conexiones HTTPS entre la aplicación, el servidor CAS y el servidor LDAP.
- Alta disponibilidad. Balanceo de carga y clúster de servidores.
- Resultados

1.6 Estado del arte

Los sistemas SSO son un tipo de sistemas que en una forma u otra están muy extendidos y vemos por todas partes. Esto se debe a sus ventajas ya que nos permite con sólo autenticarnos una vez acceder a una serie de distintos sistemas o servicios.

Un ejemplo de un sistema SSO que la mayoría usamos a diario serían los servicios de Google, ya que cuando nos loggeamos por ejemplo en el navegador Google Chrome también lo hacemos en los distintos servicios que ofrece Google como Gmail, Youtube, Google Drive...

Algunas ventajas que tiene este sistema son las siguientes:

- Reducción del riesgo de compromiso de las contraseñas ya que en cuantos menos sitios/portales se introduzcan contraseñas menos probabilidades hay de que sean robadas.
- El usuario no tiene que recordar distintas contraseñas para diferentes servicios.
- Reduce el tiempo introduciendo contraseñas del usuario para una misma identidad.

2. Estructura

2.1 Implementación sistema CAS

Introducción

En este apartado se tratará la base y estructura sobre la que se desarrollará la completitud de este TFM. Se empezará con el sistema operativo utilizado para empezar a elaborar el trabajo, las configuraciones e instalaciones de las herramientas utilizadas y finalmente entraremos en la explicación del servidor CAS creado para el inicio de este proyecto así como de la aplicación utilizada para comprobar su correcto funcionamiento.

Este apartado tiene como objetivo mostrar la primera parte de la evolución del TFM, en siguientes puntos de la memoria se podrán modificar tipos de conexiones, ejemplos, usuarios... Todo con el objetivo de enseñar en el primer momento cómo se afrontaron los problemas y como se resolvieron para en la finalidad de la memoria implementar las medidas más seguras y correctas que cumplan los objetivos de este Trabajo de Fin de Master.

Entorno de trabajo

Para realizar este TFM se ha decidido utilizar el sistema operativo CentOS[1] mediante una virtualización con con la herramienta VirtualBox. Con el objetivo de facilitar algunas tareas, como la visualización de las aplicaciones futuras en el navegador de la máquina virtual, se ha decidido instalar el escritorio GNOME para hacer estas tareas más llevaderas y ayudar a la correcta comprobación de los distintos objetivos del desarrollo.

Se han modificado algunas configuraciones en el VirtualBox para facilitar el acceso por ssh a la máquina así como la compartición de carpetas o el acceso a internet de nuestro SO. Esto se realiza abriendo puertos y modificando los adaptadores de red virtuales de la máquina e instalando las Guest Additions de VirtualBox, que son un conjunto de herramientas que facilitan el uso de las máquinas virtuales.

En este desarrollo se ha decidido utilizar el software Apache Tomcat como servidor de aplicaciones de prueba para empezar a trabajar. Se planea crear un portal web mediante el que nos loggearemos en nuestro servidor CAS y podremos acceder a nuestra aplicación web que dependiendo del usuario con que nos hayamos loggeados y sus permisos, tendremos permiso de acceso a unos recursos u otros, se nos mostrarán ciertas cosas o dejarán de hacerlo. Para esto utilizaremos Apache Tomcat, CAS y algunos elementos del framework Spring.

Como estos elementos necesitan Java para su utilización se ha instalado el OpenJDK 8 mediante el comando:

```
$ sudo yum install java-1.8.0-openjdk-devel
```

Desarrollo

El objetivo principal será tener dos servidores Apache Tomcat en nuestra máquina, uno para el servidor CAS y otro para nuestra aplicación.

Para comenzar vamos a descargar una instancia de Apache Tomcat[2] con el objetivo de que sirva como nuestro servidor CAS y lo extraeremos en la carpeta /opt de nuestra máquina virtual. Debido a que vamos a utilizar dos instancias de Tomcat hay que cambiar los puertos que se usarán ya que no pueden ser iguales, en nuestro caso vamos a cambiar el puerto de escucha de este primer Tomcat a 8090 (esto se realiza en el fichero /conf/server.xml) y todos los demás puertos les vamos a aumentar el valor en 1 para diferenciarlos de nuestro servidor de aplicaciones.

Descargamos la versión 4.0.0 del servidor CAS[3] y copiamos el fichero cas-server-webapp-4.0.0.war de la carpeta /modules a la carpeta /webapps del servidor apache tomcat que hemos instalado en nuestra máquina, de este modo a la hora de iniciar el Tomcat se descomprimirá el CAS server y podremos utilizarlo. Podemos probar que el acceso al servidor CAS mediante la URL http://localhost:8090/cas-server-webapp-4.0.0/login donde podremos observar el formulario de login de nuestro CAS. En un primer acceso el servidor sólo cuenta con 1 par de usuario/contraseña (casuser/Mellon), hardcodeados en el fichero de configuración /WEB-INF/deployerConfigContext.xml, así que en un primer cambio podemos modificar este fichero para añadir los usuarios deseados, en este caso se han creado dos distintos con el objetivo de realizar pruebas: (en el futuro se modificará el sistema para añadir un correcto manejo de los datos de usuario/contraseña mediante la conexión al sistema LDAP)

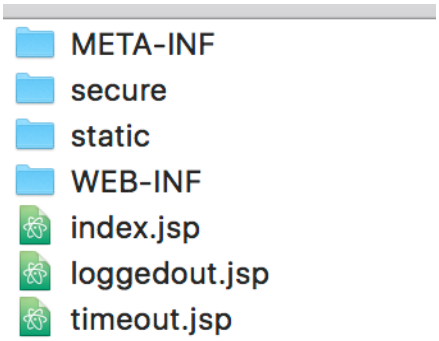
- User: ddaher1, Pass: ddaher1
- User: ddaher2, Pass: ddaher2

Estos usuarios tienen la función de mostrar la diferencia entre un usuario común y un administrador, ddaher1 y ddaher2 respectivamente. Un ejemplo de cómo podría quedar esta parte del fichero sería:

```
<bean id="primaryAuthenticationHandler"
      class="org.jasig.cas.authentication.AcceptUsersAuthenticationHandler">
    <property name="users">
        <map>
          <entry key="ddaher1" value="ddaher1"/>
          <entry key="ddaher2" value="ddaher2"/>
        </map>
    </property>
</bean>
```

Una vez ya tenemos los usuarios definidos y hemos comprobado que nos es posible loggearnos el siguiente paso es el de descargar otra instancia de apache tomcat que colocaremos en la carpeta /opt de nuestra máquina. Este nuevo servidor tendrá como objetivo ser nuestro servidor de aplicaciones donde podremos loggearnos y acceder a una app web. (URL http://localhost:8080/simplecasclient/)

Para empezar vamos a crear la carpeta de nuestra aplicación en la carpeta /webapps de nuestro servidor de aplicaciones. Esta carpeta la vamos a llamar *simplecasclient*, y tendrá la siguiente estructura general:



Donde, siguiendo la estructura general de aplicaciones de Apache Tomcat, index.jsp será la página principal de nuestra aplicación, META-INF contendrá los meta archivos que nos parezcan oportunos y WEB-INF contendrá la configuración general de la app, de su seguridad así como las librerías necesarias para la utilización de la aplicación. La carpeta *static* contendrá los posibles ficheros de estilo .css, finalmente la carpeta *secure* contendrá los archivos correspondientes a la aplicación que queremos securizar mediante CAS, los archivos visibles a los usuarios normales y los que lo son a los administradores.

El fichero applicationContext-security.xml es el fichero principal de configuración y uno muy importante, aquí tenemos que añadir las rutas de los ficheros que queremos securizar y los requisitos (roles) que son necesarios cumplir para poder acceder a ellos, las URL que conectan nuestro servidor de apps con el servidor CAS (login por ejemplo), así como un listado de usuarios que reconocemos en el sistema y los roles que poseen.

Rutas de recursos a securizar:

```
<security:http entry-point-ref="casAuthenticationEntryPoint" auto-config="true">

  <security:intercept-url pattern="/secure/extreme/**" access="ROLE_ADMIN"></security:intercept-url>
  <security:intercept-url pattern="/secure/**" access="ROLE_USER"></security:intercept-url>
  <security:intercept-url pattern="/secure/**" access="ROLE_ADMIN"></security:intercept-url>
```

Un ejemplo de uno de los enlaces del CAS que tenemos que poner, en este caso es la redirección al login del CAS:

Usuarios con sus roles respectivos:

```
<security:user-service id="userService">
  <security:user name="ddaher1" authorities="ROLE_USER"></security:user>
  <security:user name="ddaher2" authorities="ROLE_ADMIN"></security:user>
</security:user-service>
```

Ahora se van a explicar los puntos más importantes de los distintos ficheros que forman nuestra aplicación:

```
<html>
  <head>
      <meta http-equiv="content-type" content="text/html; charset=UTF-8">
      <link rel="stylesheet" href="<c:url value='/static/css/tutorial.css'/>" type="text/css" />
      <title>Home Page</title>
  </head>
<body>
<div id="content">
<h1>Home Page</h1>
<p>
Esta es una página de prueba con el objetivo de utilizar CAS para la autenticación
</p>

<p>
Si estás autenticado tu usuario es...: <%= request.getRemoteUser() %>
</p>

<%if (request.isUserInRole("ROLE_USER")) { %>
    <p>Posees el rol de usuario normal por lo que puedes acceder a las URLs de "/secure"</a>.</p>
<% } %>


<%if (request.isUserInRole("ROLE_ADMIN")) { %>
    <p>Posees el rol de administrador por lo que puedes acceder a las URLs de "/secure/extreme"</a>.</p>
<% } %>


<p>
<a href="secure/index.jsp">Página segura de ejemplo (/secure)</a></p>
<p><a href="secure/extreme/index.jsp">Página muy segura de ejemplo (/secure/extreme)</a></p>
</div>
```

1. index.jsp

Este es el fichero base de nuestra aplicación, es lo que vemos nada más entrar a la app *simplecasclient*. Lo que vemos es una página de inicio simple donde primero nos va a intentar recuperar el sistema nuestro nombre de usuario y nos lo muestra, pero si no estamos loggeados todavía nos mostrará null. Nos muestra dos enlaces, a recursos securizados y a recursos altamente securizados (/secure y /secure/extreme respectivamente), si no estamos loggeados nos redirigirá al login de CAS en cualquier enlace.

Si estamos en esta página loggeados también se nos mostrará un pequeño mensaje diciéndonos el rol que poseemos y los recursos a los que tenemos acceso.

```
<h1>Página segura</h1>
<p>
Esta es una página segura, puedes acceder si estás loggeado con permisos de usuario
</p>

<%if (request.isUserInRole("ROLE_ADMIN")) { %>
    <p>Eres administrador, por lo que tienes permisos para ver la <a href="extreme/index.jsp">página muy segura</a>.</p>
<% } %>


<p><a href="../">Home</a></p>
<p><a href="../j_spring_security_logout">Cerrar sesión</a></p>
```

2. secure/index.jsp

A esta página sólo podemos acceder después de estar loggeados, tanto usuario como admin, y nos muestra un mensaje. En el caso de que seamos administrador también nos indica que podemos acceder a la página protegida secure/extreme/index.jsp.

Existen los botones de volver al Home y de cerrar sesión, esta última opción se realiza mediante una función de spring como podemos observar en la fotografía.

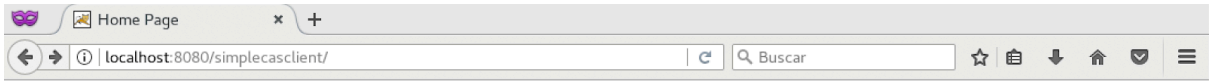
```
      <title>Página muy segura</title>
  </head>
<body>
<div id="content">
<h1>Página muy segura</h1>
Esta es una página muy segura, sólo puedes acceder si tienes permisos de administrador.

<%if (request.isUserInRole("ROLE_ADMIN")) { %>
    <p>Eres "administrador"</p>
<% } %>

<p><a href="../../">Home</a></p>
<p><a href="../../j_spring_security_logout">Cerrar sesión</a></p>
</div>
```

3. secure/extreme/index.jsp

Esta es la página que corresponde al recurso más seguro (restringido) de nuestro sistema actual ya que es necesario ser administrador para poder acceder a ella, si intentáramos entrar con un usuario con otro rol daría un mensaje de permiso denegado. Esta página nos muestra un mensaje y los botones de home y cierre de sesión.

Resultado

Estas son algunas capturas de pantalla del proceso a seguir en nuestra aplicación dentro de la máquina virtual:



Home Page

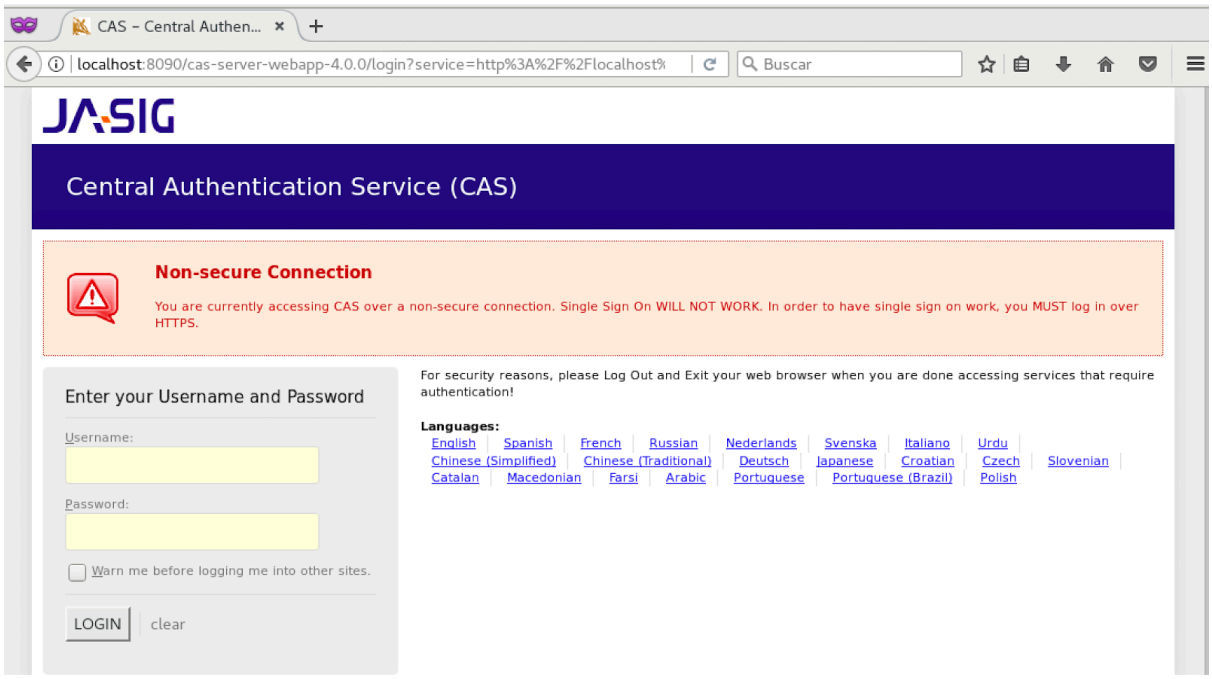
Esta es una página de prueba con el objetivo de utilizar CAS para la autenticación

Si estás autenticado tu usuario es...: null

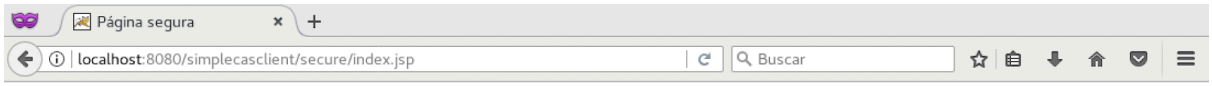
Página segura de ejemplo (/secure)

Página muy segura de ejemplo (/secure/extreme)

1. Página de inicio de *simplecasclient*. Sin autenticación



2. Login CAS al que somos redirigidos al intentar acceder a los recursos



Página segura

Esta es una página segura, puedes acceder si estás loggeado con permisos de usuario

Home

Cerrar sesión

3. Acceso al recurso /secure/index.jsp con permisos de usuario

4. Página Home con sesión de usuario iniciada

5. Página Home con sesión de admin iniciada

6. Página secure/extreme/index.jsp como administrador

7. Intento de acceso al recurso secure/extreme/index.jsp como usuario normal (ROLE_USER)

2.2 Creación y conexión del sistema OpenLDAP

Introducción

En este apartado se explicará el proceso de creación de un servidor LDAP instalando la herramienta OpenLDAP en nuestro entorno CentOS, así como la conexión desde nuestro servidor CAS a LDAP para utilizarlo como base de datos externa de usuarios.

OpenLDAP. Instalación y configuración

Para comenzar la instalación de esta herramienta procederemos a utilizar los siguientes comandos en la consola:

```
[root@localhost ~]# yum -y install openldap compat-openldap openldap-clients openldap-servers
openldap-servers-sql openldap-devel
```

Una vez instalado activaremos el servicio en esta sesión y en las siguientes, después de los reinicios, con los comandos `systemctl start slapd` y `systemctl enable slapd` respectivamente. Podemos comprobar que el servicio está en marcha de la siguiente forma:

```
[root@localhost ~]# netstat -ntup -l | grep -i 389
tcp        0      0 0.0.0.0:389             0.0.0.0:*               LISTEN      12517/slapd

tcp6       0      0 :::389                  :::*                    LISTEN      12517/slapd
```

Un primer paso en la configuración del servidor será indicar una contraseña para el usuario administrador:

```
slappasswd -h {SSHA} -s ldappassword
```

Donde “ldappassword” es la contraseña del administrador. Este comando nos devolverá nuestra clave hasheada y lista para que la añadamos a nuestro LDAP.

Creamos un fichero .ldif como el siguiente:

```
[root@localhost openLdap]# cat db.ldif
dn: olcDatabase={2}hdb,cn=config
changetype: modify
replace: olcSuffix
olcSuffix: dc=ldapserver,dc=local

dn: olcDatabase={2}hdb,cn=config
changetype: modify
replace: olcRootDN
olcRootDN: cn=admin,dc=ldapserver,dc=local

dn: olcDatabase={2}hdb,cn=config
changetype: modify
replace: olcRootPW
olcRootPW: {SSHA}ljUqMTqaoCVCagLLUsXNFKYZexc9b4p8
```

En este fichero vamos a especificar nuestro dominio (ldapserver.local), nuestro usuario administrador “admin” y su contraseña. Esta contraseña es la hasheada en el punto anterior, y en este caso coincide con “ddaher” que es la password del administrador.

Con el comando:

```
ldapmodify -Y EXTERNAL  -H ldapi:/// -f db.ldif
```

Añadimos este fichero y sus cambios al LDAP.

Después con los comandos:

```
cp /usr/share/openldap-servers/DB_CONFIG.example /var/lib/ldap/DB_CONFIG
```
y chown ldap:ldap /var/lib/ldap/*

Vamos a copiar a la carpeta /var/lib/ldap la bases de datos de ejemplo para trabajar y le cambiaremos los permisos para poder trabajar con ella.

Para trabajar en el sistema serán necesarios distintos SCHEMAS que los añadiremos con los comandos:

```
ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/openldap/schema/cosine.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/openldap/schema/nis.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/openldap/schema/inetorgperson.ldif
```

Estructura del LDAP

Ahora vamos a crear los ficheros base.ldif y organizations.ldif donde vamos a crear la estructura de nuestro servidor LDAP:

```
[root@localhost openLdap]# cat base.ldif
dn: dc=ldapserver,dc=local
dc: ldapserver
objectClass: top
objectClass: domain

dn: cn=admin ,dc=ldapserver,dc=local
objectClass: organizationalRole
cn: admin
description: LDAP Admin
```

```
[root@localhost openLdap]# cat organizations/organizations.ldif
dn: ou=People,dc=ldapserver,dc=local
objectClass: organizationalUnit
ou: People

dn: ou=Groups,dc=ldapserver,dc=local
objectClass: organizationalUnit
ou: Groups
```

En esta estructura de prueba creamos el objeto “top” que será nuestro dominio ldapserver.local, creamos la entrada correspondiente al usuario administrador del ldap y creamos dos organizational units, estrucuturas para organizar usuarios y grupos. En este caso se crearon las organizationalUnit People (usuarios) y Groups (grupos).

Con el comando:

```
ldapadd -x -W -D "cn=admin,dc=ldapserver,dc=local" -f base.ldif
ldapadd -x -W -D "cn=admin,dc=ldapserver,dc=local" -f organizations.ldif
```

Añadimos estas entradas al LDAP. Nos fijamos que en esta entrada nos estamos autenticando como administradores del servicio, así que nos pedirá la contraseña de administrador (ddaher) que habíamos modificado antes.

Con esto ya está creada la estructura base del LDAP, ahora sólo queda añadir los usuarios y los atributos que se deseen. Un ejemplo de usuario sería:

```
[root@localhost openLdap]# cat people/dani.ldif
dn: uid=dani,ou=People,dc=ldapserver,dc=local
cn: dani
givenName: dani
sn: daher
uid: dani
uidNumber: 5000
gidNumber: 10000
homeDirectory: /home/dani
mail: dani@ldapserver.local
objectClass: top
objectClass: posixAccount
objectClass: shadowAccount
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
loginShell: /bin/bash
userPassword: {SHA}cZJNx4ADfgZUw/yrGh6mXaSswZ4=
```

Al agregar este fichero .ldif a nuestro LDAP estaríamos añadiendo un usuario llamado “dani” a nuestra base de datos. Este usuario posee una serie de atributos, puede haber más o menos, como el título que tiene, el número de habitación, el correo electrónico... Este usuario en concreto tiene el identificador de usuario (uid) “dani” y pertenece al organizationalUnit de People. Su nombre de dominio completo (dn) sería: uid=dani,ou=People,dc=ldapserver,dc=local, especificando que es el usuario dani dentro de la organización llamada People que pertenece al componente de dominio ldapserver que a su vez pertenece a local. La contraseña ha sido hasheada como anteriormente creamos la del administrador.

Para buscar en nuestro sistema un usuario en particular podríamos usar, por ejemplo, el comando:

```
ldapsearch -x uid=dani -b ou=People,dc=ldapserver,dc=local
```

Con el objetivo de obtener los grupos a los que pertenecen los usuarios y facilitar el manejo y actualización de membresías se van a instalar dos Overlays en LDAP:

memberOf:

```
[root@localhost openLdap]# cat overlays/memberof_config.ldif
dn: cn=module,cn=config
cn: module
objectClass: olcModuleList
olcModuleLoad: memberof
olcModulePath: /usr/lib64/openldap

dn: olcOverlay={0}memberof,olcDatabase={2}hdb,cn=config
objectClass: olcConfig
objectClass: olcMemberOf
objectClass: olcOverlayConfig
objectClass: top
olcOverlay: memberof
olcMemberOfDangling: ignore
olcMemberOfRefInt: TRUE
olcMemberOfGroupOC: groupOfNames
olcMemberOfMemberAD: member
olcMemberOfMemberOfAD: memberOf
```

refint:

```
[root@localhost overlays]# cat refint2.ldif
dn: olcOverlay={1}refint,olcDatabase={2}hdb,cn=config
objectClass: olcConfig
objectClass: olcOverlayConfig
objectClass: olcRefintConfig
objectClass: top
olcOverlay: {1}refint
olcRefintAttribute: memberof member manager owner
```

Estos módulos se encargan de especificar los miembros de un grupo y de actualizar los grupos y sus dependencias en caso de que se modifiquen sus miembros o se actualicen sus componentes, respectivamente.

Un ejemplo de grupo utilizando el overlay memberOf donde especificamos los miembros de dicho grupo con el atributo member es el siguiente:

```
[root@localhost openLdap]# cat groups/grupousuario.ldif
dn: cn=grupousuario,ou=Groups,dc=ldapserver,dc=local
objectClass: groupofnames
cn: grupousuario
description: Usuarios comunes
member: uid=dani,ou=People,dc=ldapserver,dc=local
```

Finalmente añadimos el servicio ldap al firewall del sistema para que podamos conectarnos al servicio:

```
firewall-cmd --permanent --add-service=ldap
firewall-cmd –reload
```

Con esto ya hemos creado nuestro sistema LDAP y su estructura, ahora vamos a proceder a la conexión con CAS.

Conexión con CAS

Para poder utilizar LDAP con CAS y Apache Tomcat lo primero es añadir la dependencia de LDAP y CAS al fichero pom.xml:

```
<dependency>
    <groupId>org.jasig.cas</groupId>
    <artifactId>cas-server-support-ldap</artifactId>
    <version>${cas.version}</version>
</dependency>
```

El punto central y más importante de la conexión a LDAP es modificar la configuración de CAS en el *authenticationManager* para que utilice el AuthenticationHandler que vamos a crear para LDAP en vez del que viene por defecto.

Toda la configuración de LDAP la almacenamos en el fichero ldapConfig.xml. En este fichero se almacena toda la configuración técnica de la conexión, pero podríamos decir que la parte más importante es la siguiente:

```
<bean id="primaryAuthenticationHandler" class="org.jasig.cas.authentication.LdapAuthenticationHandler"
  p:principalIdAttribute="uid" c:authenticator-ref="authenticator"  init-method="initialize">
  <property name="principalAttributeMap">
    <map>
                <entry key="memberOf" value="memberOf" />
                <entry key="cn" value="cn" />
                <entry key="sn" value="Sn" />
            </map>
  </property>
</bean>

<bean id="authenticator" class="org.ldaptive.auth.Authenticator"
  c:resolver-ref="dnResolver" c:handler-ref="authHandler" />

<bean id="dnResolver" class="org.ldaptive.auth.PooledSearchDnResolver"
  p:baseDn="${ldap.baseDn}" p:subtreeSearch="true" p:allowMultipleDns="false"
  p:connectionFactory-ref="searchPooledLdapConnectionFactory"
  p:userFilter="${ldap.authn.searchFilter}" />

<bean id="searchPooledLdapConnectionFactory" class="org.ldaptive.pool.PooledConnectionFactory"
  p:connectionPool-ref="searchConnectionPool" />

<bean id="searchConnectionPool" parent="abstractConnectionPool"
  p:connectionFactory-ref="searchConnectionFactory" />

<bean id="searchConnectionFactory" class="org.ldaptive.DefaultConnectionFactory"
  p:connectionConfig-ref="searchConnectionConfig" />

<bean id="searchConnectionConfig" parent="abstractConnectionConfig"
  p:connectionInitializer-ref="bindConnectionInitializer" />

<bean id="bindConnectionInitializer" class="org.ldaptive.BindConnectionInitializer"
  p:bindDn="${ldap.managerDn}">
  <property name="bindCredential">
    <bean class="org.ldaptive.Credential" c:password="${ldap.managerPassword}" />
  </property>
</bean>
```

En este fichero podemos ver la bean principal que es nuestro primaryAuthenticationHandler donde nos conectamos a LDAP y vamos a utilizar

un método de autenticación mediante búsqueda de *uid*. Tenemos un mapa de atributos que son los atributos que queremos obtener de nuestro sistema ldap además de nuestro usuario/contraseña, y todo en el mismo momento de la conexión de autenticación. Para realizar esta recolección hay que modificar el fichero de configuración principal *deployerConfigContext.xml* para permitir esta acción, esto se hace añadiendo las siguientes beans:

```
<bean id="attributeRepository"
      class="org.jasig.cas.persondir.LdapPersonAttributeDao"
      p:connectionFactory-ref="searchPooledLdapConnectionFactory"
      p:baseDN="${ldap.baseDn}"
      p:searchControls-ref="searchControls"
      p:searchFilter="uid={0}"
       init-method="initialize">
      <property name="requireAllQueryAttributes" value="true" />

    <property name="resultAttributeMapping">
        <map>
            <!--
               | Key is LDAP attribute name, value is principal attribute name.
               -->
            <entry key="memberOf" value="memberOf" />
            <entry key="cn" value="cn" />
            <entry key="sn" value="Sn" />
        </map>
    </property>
</bean>
<bean id="searchControls"
        class="javax.naming.directory.SearchControls"
        p:searchScope="2"
        p:countLimit="10" />

<bean id="serviceRegistryDao" class="org.jasig.cas.services.InMemoryServiceRegistryDaoImpl"
        p:registeredServices-ref="registeredServicesList" />

<util:list id="registeredServicesList">

    <bean class="org.jasig.cas.services.RegexRegisteredService"
          p:id="0" p:name="HTTP and IMAP" p:description="Allows HTTP(S) and IMAP(S) protocols"
          p:serviceId="^(https?|imaps?)://.*" p:evaluationOrder="10000001" >
          <property name="allowedAttributes">
            <list>
              <value>memberOf</value>
              <value>cn</value>
              <value>Sn</value>
            </list>
        </property>
      </bean>
```

Donde especificamos una vez más los atributos deseados. El punto más importante es la propiedad *allowedAttributes* donde tenemos que permitir en el filtro de CAS que se envíen estos atributos, de manera contraria no podremos realizar el envío.

En el resto de beans de nuestro fichero ldap realizamos conexión, para todo este procedimiento se utilizan constantes del tipo *ldap.baseDN*, estas constantes que tienen los valores para conectarnos a ldap y utilizarlos en nuestro código están almacenados en el fichero *ldap.properties*, algunas de estos valores son los siguientes:

```
#=========================================
# General properties
#=========================================
# ldap.url=ldaps://localhost:636
ldap.url=ldap://localhost
# Start TLS for SSL connections
ldap.useStartTLS=false

# Directory root DN
ldap.rootDn=dc=ldapserver,dc=local

# Base DN of users to be authenticated
ldap.baseDn=ou=People,dc=ldapserver,dc=local

# LDAP connection timeout in milliseconds
ldap.connectTimeout=3000

# Manager credential DN
ldap.managerDn=cn=admin,dc=ldapserver,dc=local

# Manager credential password
ldap.managerPassword=ddaher
```

Aquí especificamos la url de nuestro ldap, nuestro dominio, nuestro usuario administrador con el que ejecutaremos las búsquedas... Existen más constantes como la que especifica la ruta de búsqueda de usuarios que señaliza hacia nuestra organizationalUnit People.

Con estas modificaciones ya nos podríamos autenticar desde nuestra webapp utilizando el repositorio LDAP mediante http, en el siguiente punto se explicarán las modificaciones para realizar las conexiones por HTTPS.

Recuperación de atributos en el cliente

Para poder recuperar los atributos de nuestro usuario, los grupos a los que es miembro por ejemplo, y según a qué grupos pertenezca poder realizar una asignación de permisos u otra hace falta realizar la conexión mediante HTTPS. En este apartado se explicó la retribución de atributos pero no será totalmente funcional hasta que se implemente en el siguiente punto la conexión utilizando SSL. En el desarrollo de este TFM se han conseguido enviar los atributos deseados desde el servidor CAS a la aplicación cliente pero no se ha podido implementar la recogida de los atributos desde el cliente, esto ha sido por causas de falta de tiempo y desconocimiento específico. Tengo conocimiento de que hay que implementar una clase UserDetail como servicio para modificar la forma en la que se realiza la asignación de permisos (actualmente hardcodeada en fichero) pero no he sido capaz de implementarlo por diversos problemas y no he dispuesto de tiempo para buscar otras alternativas.

Incluso si la recogida de atributos ha sido infructuosa puedo asegurar que aunque no la haya podido realizar por problemas la parte más difícil era conectar

correctamente el ldap y realizar el envío de atributos. En nuestro fichero de log, catalina.out, de nuestra aplicación se puede observar que se reciben los atributos correctamente:

```
<cas:serviceResponse xmlns:cas='http://www.yale.edu/tp/cas'>
  <cas:authenticationSuccess>
    <cas:user>dani</cas:user>
    <cas:attributes>


<cas:memberOf>cn=grupousuario,ou=Groups,dc=ldapserver,dc=local        </cas:memberOf>

<cas:cn>dani        </cas:cn>

<cas:Sn>daher        </cas:Sn>


</cas:attributes>


  </cas:authenticationSuccess>
</cas:serviceResponse>
```

Para poder comprobar que se envían bien los atributos y observar esta salida se ha modificado el fichero casServiceValidationSuccess.jsp de WEB-INF/view/jsp/protocol/2.0 añadiendo la siguiente configuración:

```
<cas:attributes>
  <c:forEach var="auth" items="${assertion.chainedAuthentications}">
    <c:forEach var="attr" items="${auth.principal.attributes}" >
      <cas:${fn:escapeXml(attr.key)}>${fn:escapeXml(attr.value)}        </cas:${fn:escapeXml(attr.key)}>
    </c:forEach>
  </c:forEach>
</cas:attributes>
```

2.3 Securización mediante HTTPS

Para realizar la securización mediante HTTPS en nuestro sistema (cliente -> servidor CAS -> LDAP -> servidor CAS -> cliente) tenemos que usar certificados y keys, y para realizarlo tenemos que utilizar los mismos certificados en todos los componentes de nuestra conexión para que puedan confiar los unos en los otros y realizar así las conexiones utilizando SSL.

Certificados

Con el objetivo de securizar nuestro sistema vamos a proceder a crear diferentes certificados, claves y keyStores. En nuestro sistema nos lo hemos planteado como primero crear los certificados y claves para securizar el LDAP y luego utilizaremos las claves creadas ahí para securizar el servidor y el cliente. Para realizarlo utilizaremos las herramientas *openssl* y y *keytool*:

/*creamos el certificado raíz, usando la configuración existente en openssl.cnf*/

openssl req -new -x509 -days 3650 -extensions v3_ca -keyout cakey.pem -out cacert.pem -config openssl.cnf

openssl req -new -nodes -out ldap-req.pem -keyout ldap-key.pem -config openssl.cnf

openssl ca -config openssl.cnf -out ldap-cert.pem -infiles ldap-req.pem

/*creamos keystore para almacenar el certificado que usará tomcat para cifrar conexiones*/
/*Generamos el almacén de claves para tomcat con keytool*/

keytool -genkey -alias tomcat -keyalg RSA -keystore tomcat.jks

/*Generamos el CSR para que se pueda generar posteriormente el certificado*/

keytool -certreq -keyalg RSA -alias tomcat -file tomcat-req.csr -keystore tomcat.jks

/*Creamos el certificado a partir del CSR*/

openssl ca -config openssl.cnf -out tomcat-cert.pem -infiles tomcat-req.csr

/*Importamos la CA al keystore para luego poder importar nuestro certificado*/

keytool -import -alias root_ca -keystore tomcat.jks -trustcacerts -file cacert.pem

/*importamos el certificado creado*/

keytool -import -alias tomcat -keystore tomcat.jks -file tomcat-cert.pem

/*creamos un truststore con el certificado de la CA*/

keytool -import -file cacert.pem -alias firstCA -keystore mytruststore.ks

Con estos comandos ya hemos creado todo lo que necesitaremos para realizar la securización.

OpenLDAP

Una vez ya creados los certificados tenemos que configurar LDAP para usarlos. Esto lo vamos a realizar con el siguiente fichero .ldif:

```
[root@localhost openLdap]# cat addSSL.ldif
dn: cn=config
changetype: modify
replace: olcTLSCACertificateFile
olcTLSCACertificateFile: /etc/ldap/sas12/cacert.pem

dn: cn=config
changetype: modify
replace: olcTLSCertificateFile
olcTLSCertificateFile: /etc/ldap/sas12/ldap-cert.pem

dn: cn=config
changetype: modify
replace: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /etc/ldap/sas12/ldap-key.pem
```

En este fichero especificamos la ruta de los certificados que creamos antes (los correspondientes en cada caso). Con el comando siguiente añadimos el fichero a nuestro sistema:

ldapmodify -Y EXTERNAL -H ldapi:/// -f addSSL.ldif

Ahora hay que configurar LDAP para que escuche por SSL, esto se hace modificando el fichero /etc/sysconfig/slapd y añadiendo *ldaps:///* en las URL que escucha:

```
SLAPD_URLS="ldapi:/// ldap:/// ldaps:///"
```

Ahora reiniciamos el servicio con *systemctl restart slapd* y ya tenemos a LDAP escuchando en SSL por el puerto 636.

Lo último sería modificar el firewall acorde con nuestro ldap:

firewall-cmd --permanent --add-service=ldaps
firewall-cmd –reload

Y ya tenemos la conexión a OpenLDAP securizada.

CAS Server y cliente

- Server:

Ahora que tenemos LDAP por SSL tenemos que modificar en Tomcat la forma en que teníamos de conectarnos al servidor LDAP.

Lo primero es modificar nuestro fichero de configuración de variables LDAP, y modificaremos la línea de *ldap.url* con el valor:

ldap.url=ldaps://localhost:636

Ya que no nos vamos a conectar por http normal sino por https tenemos que utilizar *ldap**s*** en vez de *ldap* al igual que debemos especificar el puerto de escucha.

- Server y cliente:

Después de haber modificado la url de ldap en el server el resto de los cambios en configuración son similares tanto en el servidor CAS como en el cliente ya que ambos están utilizando Apache Tomcat para su despliegue.

Empezamos modificando el fichero *server.xml* de Tomcat para modificar la forma en la que enviamos y aceptamos conexiones, esto se realiza sustituyendo el *Connector* HTTP que teníamos de forma predeterminada por el conector HTTPS siguiente (server CAS y cliente deben utilizar puertos distintos, 9443 y 9444 en este caso):

```
<Connector port="9443" protocol="org.apache.coyote.http11.Http11NioProtocol"
           maxThreads="150" SSLEnabled="true">
    <SSLHostConfig>
        <Certificate certificateKeystoreFile="conf/tomcat.jks" alias="tomcat"
                     type="RSA" />
    </SSLHostConfig>
</Connector>
```

Este conector implementa las conexiones por SSL en nuestro servidor y además le estamos indicando que el certificado que vamos a utilizar para este conector es el tomcat.jks que lo hemos creado anteriormente, como lo explicamos en el punto de Certificados.

Lo último que hay que modificar en ambas partes es el script catalina.sh que es el script del Tomcat que se ejecuta para iniciar. En este script tenemos que añadir a la variable JAVA_OPTS una serie de valores para indicar el trustcore de nuestros certificados:

JAVA_OPTS="$JAVA_OPTS -Djavax.net.ssl.trustStore=$CATALINA_HOME /conf/mytruststore.ks -Djavax.net.ssl.trustStorePassword=changeit"

En la carpeta /conf tenemos nuestro mytruststore.ks, y hay que señalar que le hemos añadido la contraseña con la que se aseguró el almacén de claves al crear las claves (*changeit* en nuestro caso).

- Cliente:

De manera individual en el cliente lo único que hay que cambiar es los enlaces en el fichero de configuración donde indicamos la dirección de CAS para conectarnos por los nuevos enlaces con https y puerto 9443, ejemplo:

Antes: http://localhost/cas/login
Con SSL: https://localhost:9443/cas/login

Con esta configuración ya tenemos implementada la securización mediante HTTPS entre las conexiones de nuestro sistema.

2.4 Alta disponibilidad

Alta disponibilidad (High availability) es un protocolo de diseño del sistema y su implementación asociada que asegura un cierto grado absoluto de continuidad operacional durante un período de medición dado. Disponibilidad se refiere a la habilidad de la comunidad de usuarios para acceder al sistema, someter nuevos trabajos, actualizar o alterar trabajos existentes o recoger los resultados de trabajos previos.

En nuestros sistema para implementar la alta disponibilidad vamos a implementar un balanceador de carga, que actuará como reverse proxy, que será el encargado de dar entrada a un clúster de servidores. En el clúster de servidores que vamos a crear vamos a tener dos instancias de nuestro servidor CAS, de forma que si un servidor sufre un gran número de peticiones o llegara a colapsar tendríamos un servidor secundario al que podrían ir las peticiones para poder seguir prestando el servicio de manera ininterrumpida.

Balanceador de carga

Como balanceador de carga hemos utilizado Apache HTTP Server con el conector/plugin mod_jk. El servidor Apache es un servidor web y el conector mod_jk es lo que nos permitirá convertir nuestro servidor Apache en un balanceador de carga para nuestro sistema.

La instalación del servidor la realizamos mediante consola en nuestro sistema CentOS con el comando:

```
yum install httpd-devel apr apr-devel apr-util apr-util-devel gcc
gcc-c++ make autoconf libtool
```

Este comando nos instalará el servidor Apache con las herramientas que necesitaremos para su correcta utilización con mod_jk.

Lo primero es descargarnos el conector mod_jk desde la página de Apache. Una vez descargado entramos a la carpeta *native* del conector descargado y vamos a ejecutar los siguientes comando para compilar el conector y añadirlo a nuestro sistema:

```
./configure --with-apxs=/usr/sbin/apxs
make
libtool --finish /usr/lib64/httpd/modules
make install
```

El siguiente paso es crear el fichero de configuración (si no está creado ya) principal de nuestro worker (servidor): /etc/httpd/conf/workers.properties

```
[root@localhost sas12]# cat /etc/httpd/conf/workers.properties
worker.list=loadbalancer,status
worker.server1.port=8009
worker.server1.host=localhost
worker.server1.type=ajp13

worker.server2.port=9009
worker.server2.host=localhost
worker.server2.type=ajp13

worker.server1.lbfactor=1
worker.server2.lbfactor=1

worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=server1,server2
worker.loadbalancer.sticky_session=true

worker.status.type=status
```

- worker.list: vamos a crear dos worker distintos, uno que será nuestro load balancer y un worker que nos permitirá ver el estado de nuestros servidores así podremos realizar tareas de monitorización.
- worker.loadbalancer.type: especificamos que este worker va a ser un load balancer.
- worker.loadbalancer.balance_workers: es una lista de los servidores a los que vamos a aplicar el balanceo de carga, como tenemos dos servidores CAS en nuestro caso creamos los server1 y server2.
- worker.loadbalancer.sticky_session: especificamos que queremos que se utilice las sticky sessions con el objetivo de compartir las sesiones entre los distintos servidores.
- worker.status.type: especificamos que este worker tratará el módulo status.
- worker.serverX.port: puerto por el que se conectará el servidor Apache a los servidores CAS (tienen que ser distintos).
- worker.serverX.host: la IP del servidor en la red. localhost en nuestro caso.
- worker.serverX.type: es el tipo de conexión, en este caso utilizaremos la conexión ajp13 con los conectores que implementaremos más adelante.
- worker.serverX.lbfactor: es un factor que decide la preferencia de los servidores, a cual dar más preferencia al redireccionar peticiones en función de un número de peticiones máximas tras el que cambiaría por ejemplo. En este caso como no tenemos ninguno que queremos que sea especial ponemos el mismo factor y se dividirán las peticiones equitativamente.

Ahora para poder establecer las conexiones y las redirecciones y que sea mediante SSL necesitamos instalar el mod_ssl en apache con:

```
sudo yum install mod_ssl openssl
```

Y ahora vamos a modificar su fichero de configuración /etc/httpd/conf.d/ssl.conf para que se encargue de redireccionar nuestras peticiones como load balancer y cargue nuestros módulos:

```
##
## SSL Virtual Host Context
##

<VirtualHost _default_:443>

# General setup for the virtual host, inherited from global configuration
#DocumentRoot "/var/www/html"
ServerName localhost

JkMount /status status
JkMount /* loadbalancer
```

Vamos a crear un host virtual en el puerto 443 para aceptar las nuestras peticiones.

La primera variable es ServerName y tendremos que poner el host de nuestro servidor Apache, localhost en nuestro caso.

Vamos a realizar el redireccionamiento de dos formas con el comando JkMount de nuestro módulo:

JkMount /status status: realiza el forward de las peticiones a ServerName/status al worker que es llamado *status*. Este worker está definido en workers.properties.

JkMount /* loadbalancer: realiza el forward de las peticiones a ServerName/* (seguido de cualquier cosa) al worker loadbalancer. Este worker está definido en worker.properties.

Tenemos que activar el protocolo SSL si no lo está ya con *SSLEngine on*.

Al igual que en puntos anteriores para realizar las conexiones mediante HTTPS es necesario añadir los certificados. Vamos a añadir los certificados que creamos en el punto de LDAP de la siguiente forma:

```
#   Server Certificate:
# Point SSLCertificateFile at a PEM encoded certificate.  If
# the certificate is encrypted, then you will be prompted for a
# pass phrase.  Note that a kill -HUP will prompt again.  A new
# certificate can be generated using the genkey(1) command.
SSLCertificateFile /etc/ldap/sas12/ldap-cert.pem

#   Server Private Key:
#   If the key is not combined with the certificate, use this
#   directive to point at the key file.  Keep in mind that if
#   you've both a RSA and a DSA private key you can configure
#   both in parallel (to also allow the use of DSA ciphers, etc.)
SSLCertificateKeyFile /etc/ldap/sas12/ldap-key.pem
```

Esta es toda la configuración necesaria por parte del load balancer de Apache HTTP Server.

Clúster de servidores

Para nuestro clúster de servidores vamos a copiar nuestro servidor Tomcat con CAS.

Antes de proseguir con la creación del clúster es necesario cambiar los puertos en el fichero de configuración del servidor Tomcat que hemos copiado, Tomcat2 de aquí en adelante.

Ahora vamos a empezar con las modificaciones que tendremos que realizar en los dos servidores para crear el clúster.

- Fichero server.xml:

Además de haber cambiado los puertos de *Server* como se nombró antes hay que realizar un par de cambios.

Anteriormente nos conectábamos por HTTPS a nuestro servidor CAS mediante el puerto 9443 pero esto ahora ya no es necesario porque no nos vamos a conectar directamente a nuestro servidor sino vamos a conectarnos al reverse proxy y éste nos redireccionará al CAS.

En el fichero workers.properties no utilizamos el puerto 9443 sino que utilizamos los puertos 8009 y 9009 y el tipo de conexión ajp13 y de eso nos vamos a encargar. Esto se debe a que Apache HTTP no se conectará a los miembros de su propio dominio mediante HTTPS sino por el protocolo AJP y por eso ya no nos sirve el *Connector* HTTPS que teníamos sino que tenemos que crear un nuevo conector tipo ajp:

```
<!-- Define an AJP 1.3 Connector on port 8009 -->
<Connector port="8009" protocol="AJP/1.3" redirectPort="7443" />
```

Este es el conector que utilizamos en Tomcat1, en Tomcat2 es igual pero con el puerto 9009 como puerto principal y otro puerto de redirección.

También nos fijamos que en el fichero de workers utilizamos los nombres server1 y server2 así que hay que especificarlos aquí también para que se puedan detectar:

```
<Engine name="Catalina" defaultHost="localhost" jvmRoute="server1">
```

Añadiremos el atributo *jvmRoute* al *Engine* de los dos servidores Tomcat y pondremos los nombres server1 y server2 a nuestros servidores respectivos.

Como último punto a modificar de este fichero es el especificar que los Tomcat pertenecen a un clúster de servidores, para poder replicar la información de sesión, y esto se realiza añadiendo la siguiente línea dentro de la etiqueta *Engine*:

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
```

- Fichero web.xml:

La última modificación es del fichero web.xml dentro de nuestro CAS en el servidor Tomcat. Con motivo similar a la última modificación tenemos que añadir la línea <*distributable* /> en el fichero.

- Cliente:

En el cliente hay que cambiar los enlaces por los que nos conectábamos a CAS ya que ahora no utilizamos el puerto 9443 sino que nos conectamos al reverse proxy. Para el nuevo enlace bastará con introducir https://localhost/cas/login por ejemplo.

Tickets

El punto final de la implementación de la alta disponibilidad, pero no por eso menos importante más bien lo contrario, es el correcto manejo de los tickets.

Ya tenemos implementado nuestro balanceador de carga con nuestro clúster de servidores y todo por HTTPS, pero si intentamos conectarnos desde el cliente en el momento de autenticarnos nos dirá que no reconoce el ticket enviado por el servidor CAS y esto se debe a que la forma en la que CAS realiza el manejo de tickets no está pensado para un entorno en alta disponibilidad.

Para solventar este problema vamos a implementar un replicado de tickets por caché utilizando la herramienta de software libre Ehcache.

Para realizar esto se ha modificado el fichero ticketRegistry.xml que venía por defecto en los servidores CAS y se les ha cambiado su contenido por el necesario para la configuración de Ehcaché. Parte de la configuración se podría ver así:

```
<bean id="cacheManager"
      class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean"
      p:configLocation="classpath:ehcache-replicated.xml"
      p:shared="false"
      p:cacheManagerName="ticketRegistryCacheManager" />

<bean id="ticketRMISynchronousCacheReplicator"
      class="net.sf.ehcache.distribution.RMISynchronousCacheReplicator">
      <constructor-arg index="0" value="true"/>
      <constructor-arg index="1" value="true"/>
      <constructor-arg index="2" value="true"/>
      <constructor-arg index="3" value="true"/>
      <constructor-arg index="4" value="true"/>
    </bean>
```

En la captura anterior vemos que nombramos el fichero ehcache-replicated.xml y este fichero será donde especifiquemos la configuración de cada uno de los servidores Tomcat y el fichero más importante. El fichero del Tomcat1 sería el siguiente:

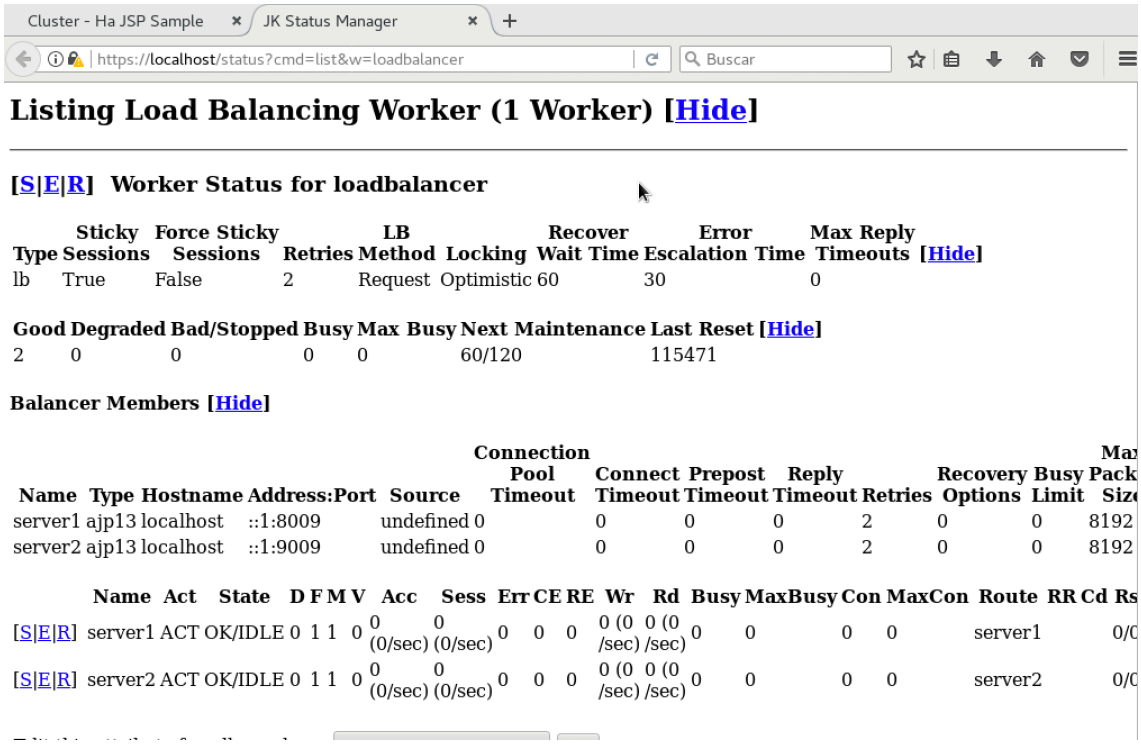
```
<ehcache updateCheck="false" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://ehcache.sf.net/ehcache.xsd">

  <diskStore path="java.io.tmpdir/cas" />

<!--  <cacheManagerPeerProviderFactory class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"-->
<!--    properties="peerDiscovery=automatic, multicastGroupAddress=230.0.0.1, multicastGroupPort=4446, timeToLive=32" />-->
<!--->

  <cacheManagerPeerProviderFactory class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
    properties="hostName=localhost, peerDiscovery=automatic, multicastGroupAddress=239.1.1.1, multicastGroupPort=4567, timeToLive=32" />

  <cacheManagerPeerListenerFactory class="net.sf.ehcache.distribution.RMICacheManagerPeerListenerFactory"
    properties="hostName=localhost, port=40001"
    propertySeparator="," />

  <defaultCache
    maxElementsInMemory="10000"
    eternal="false"
    timeToIdleSeconds="300"
    timeToLiveSeconds="300"
    overflowToDisk="true"
    maxElementsOnDisk="10000000"
    diskPersistent="false"
    diskExpiryThreadIntervalSeconds="120"
    memoryStoreEvictionPolicy="LRU" />

</ehcache>
```

En este fichero especificamos que queremos realizar Ehcache en nuestro sistema utilizando una búsqueda automática de los miembros activos. Esta búsqueda se crea mediante la dirección multicas suministrada y en el host *localhost*. En los dos servidores Tomcat hay que especificar un puerto distinto, 40001 en Tomcat1 y 40002 en Tomcat2.

Para utilizar Ehcache se añadieron las dependencias y librerías correspondientes.

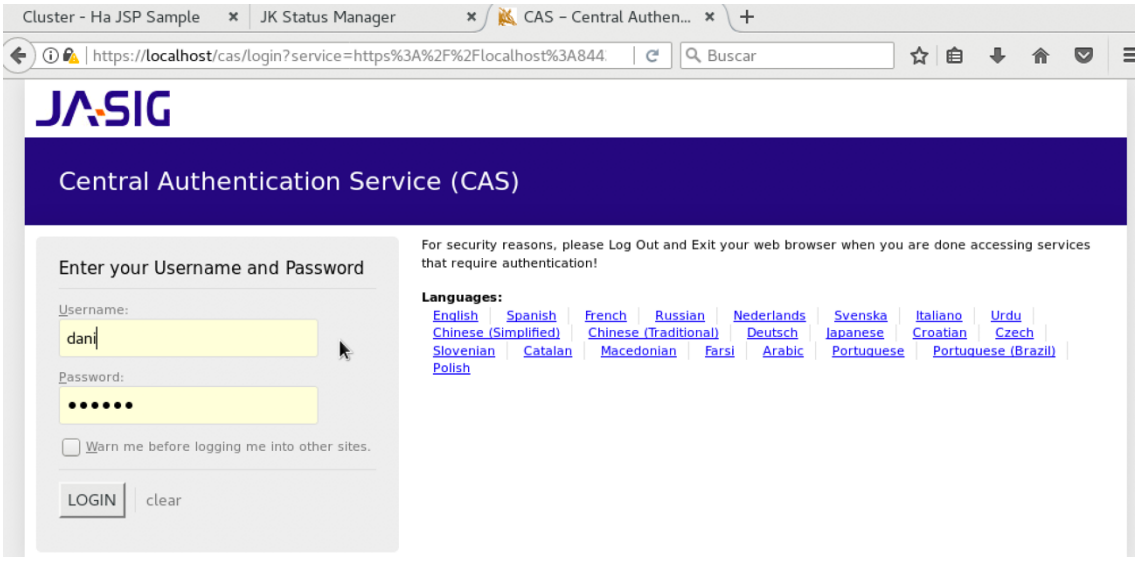
2.5 Resultados

localhost/status de los miembros del clúster de servidores, los vemos activos y sin problemas:

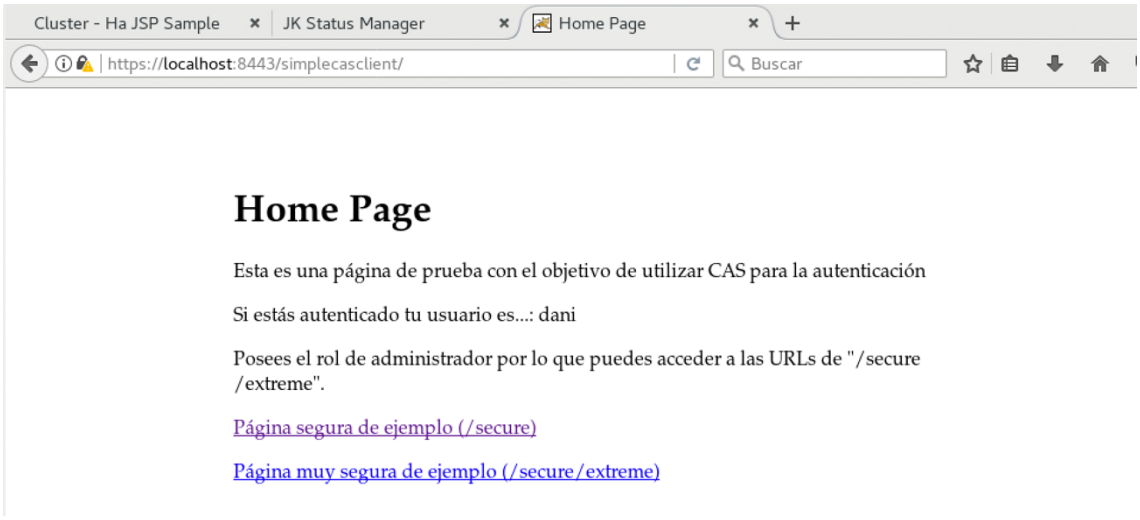


Cliente sin autenticarnos por https:

Accedemos a un recurso protegido y nos conectamos a CAS para autenticarnos. Usando HTTPS y sin puerto, ya que nos estamos conectando al reverse proxy:



Cliente una vez loggeado:



Tras habernos autenticado y actuado en la página si volvemos a la página de status vemos como se ha balanceado la carga entre los distintos servidores (columna con atributo V):

	Name	Act	State	D	F	M	V	Acc	Sess	Err	CE	RE	Wr	Rd	Busy	MaxBusy	Con	MaxCon	Route	RR	Cd	Rs
[S|E|R]	server1	ACT	OK	0	1	1	3	6 (0/sec)	1 (0/sec)	0	0	0	3.9K (0 /sec)	20K (0 /sec)	0	2	3	3	server1		0/0	11
[S|E|R]	server2	ACT	OK	0	1	1	1	1 (0/sec)	1 (0/sec)	0	0	0	446 (0 /sec)	461 (0 /sec)	0	1	1	1	server2		0/0	11

Incluso si apagamos un servidor Tomcat (1 por ejemplo) y nos intentamos autenticar en una página de navegación de incógnito el server2 acepta la conexión y lleva su tarea a cabo sin problema. Podemos comprobar que aunque haya un servidor apagado el server2 obtuvo la carga y trabajó:

	Name	Act	State	D	F	M	V	Acc	Sess	Err	CE	RE	Wr	Rd	Busy	MaxBusy	Con	MaxCon	Route	RR	Cd	R
[S|E|R|T]	server1	ACT	ERR	0	1	1	2	13 (0/sec)	3 (0/sec)	1	0	0	7.8K (0 /sec)	39K (0 /sec)	0	2	4	5	server1		54/1	
[S|E|R]	server2	ACT	OK	0	1	1	4	9 (0/sec)	4 (0/sec)	0	0	0	5.2K (0 /sec)	21K (0 /sec)	0	2	4	4	server2		0/0	

3. Conclusiones

A título personal este TFM me ha parecido muy interesante ya que los sistemas SSO son un producto necesario en la actualidad y que podemos encontrar en numerosas situaciones ya que ofrecen diversas ventajas a los usuarios como acceder a un número de servicios autenticándose una sola vez.

De este proyecto me llevo gran cantidad de conocimientos, sobretodo en el área de certificados y balanceo de carga, áreas de las cuales no poseía conocimientos previos.

Sin duda creo que los puntos más tediosos del TFM han sido las conexiones entre CAS y LDAP y el balanceo de carga con el clúster de servidores. Todo esto debido a la falta de documentación o, más bien, a lo difícil que es llevar a cabo lo documentado por la ingente cantidad de distintas versiones y configuraciones posibles que existen. Es muy importante tener un buen recuento de las librerías que se instalan, de las que son necesarias y de sus respectivas dependencias ya que es muy fácil caer en errores de este tipo. En gran parte es por la necesidad de utilizar y unificar tantos sistemas distintos para funcionar en un conjunto como son LDAP, Apache Tomcat, Apache HTTP, CAS...

El objetivo general del TFM se ha cumplido pero si tengo que decidir la parte que no se ha cumplido sería la forma en la que se gestionan los permisos en el cliente. Como ya se nombró en el punto correspondiente de la memoria, los grupos aunque son recibidos en el cliente, desde el CAS->LDAP, no son gestionados correctamente ya que falta la implementación de una clase que los acepte y dependiendo de los grupos a los que pertenece el usuario autenticado ofrezca unos permisos u otros. Actualmente los grupos se encuentran hardcodeados en el fichero correspondiente en la aplicación. Por problemas de tiempo y diversos problemas en la implementación no se pudo llevar a cabo pero esto no quita que el objetivo general y más complejo del TFM haya sido conseguido satisfactoriamente.

El sistema implementado en este TFM es un sistema “índice” vamos a decir, ya que los servidores y las aplicaciones son de prueba. De cara a líneas de trabajo futuro se pueden crear aplicaciones reales con bases de datos de usuarios reales. En este caso el clúster de servidores sólo incluye dos instancias de servidores CAS, en un futuro podemos crear más redirecciones en nuestro balanceador de carga con distintos servidores web, aplicaciones... todo lo que se desee.

A modo de conclusión podemos decir que se ha conseguido el objetivo principal del TFM, se ha realizado satisfactoriamente un sistema SSO donde mediante una aplicación podemos acceder a un clúster de servidores al que accedemos mediante un reverse proxy, y todo esto mediante HTTPS.

4. Bibliografía

[1] Everything ISO - https://www.centos.org/download/

[2] Versión 8.5.29 - https://tomcat.apache.org/download-80.cgi

[3] Versión 4.0.0 https://www.apereo.org/projects/cas/download-cas

6. Anexos

Listado de apartados que son demasiado extensos para incluir dentro de la memoria y tienen un carácter autocontenido (por ejemplo, manuales de usuario, manuales de instalación, etc.)

Dependiente del tipo de trabajo, es posible que no haya que añadir ningún anexo.