Máster Interuniversitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones.

Trabajo final de máster.

IMPLANTACIÓN DE UN SSO

Autor: José María Guerra Torres

Director: Antoni González Ciria (ANCERT)

Profesor responsable: Víctor García Font

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Implantación de un SSO*
Nombre del autor:	*José María Guerra Torres*
Nombre del consultor/a:	*Antoni González Ciria*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	06/2018
Titulación::	*Máster Interuniversitario en Seguridad de las Tecnologías de la* *Información y de las Comunicaciones.(Mistic)*
Área del Trabajo Final:	*Trabajo Final Máster.*
Idioma del trabajo:	***Español***
Palabras clave	*SSO, alta disponibilidad, autenticacion.*

Agradecimientos

Quisiera agradecer a mi mujer Rosa y mi hija Lucía el apoyo recibido.

Al director Antoni González Ciria sus precisas indicaciones.

A mi tutor, Richard Rivera Guevara, quiero agradecerle su pronta atención a cualquier cuestión planteada.

Estoy profundamente agradecido al conjunto de la comunidad educativa de este máster, en especial a la UOC, por el acogimiento y atención recibida. ¡Me habéis hecho las cosas muy fáciles a pesar de la distancia!

Resumen.

La autorización es un concepto clave en la seguridad informática. Con este proceso demostramos quienes somos.

Es común en la vida cotidiana que desde un sistema informático donde se ha realizado una autenticación, se realice un acceso a otros entornos distintos que también precisan de autenticación.

Este hecho provoca muchos inconvenientes: fatiga de autenticación, esfuerzo memorístico, lentitud al acceder al sistema. Esto genera también problemas de importancia en la seguridad porque el usuario suele apuntar las contraseñas si éstas son muchas.

Para solucionar esto se crearon los sistemas SSO, Single Sign-On, que con un sólo acto de autenticación, se proporciona acceso a un conjunto de sistemas relacionados.

Las ventajas son evidentes: El usuario ya no se cansa, mejora su productividad y la seguridad informática se ve fortalecida.

La adopción de un sistema SSO también tiene inconvenientes: Si el servidor SSO deja de funcionar, se producirá una caída completa del sistema. Por esta razón es necesario un sistema tolerante a fallos, es decir, tiene que estar desplegado en un entorno de alta disponibilidad.

El presente trabajo final de máster tiene como propósito la realización de una infraestructura SSO dotada de Alta Disponibilidad.

Palabras clave: SSO, Autenticación, Alta Disponibilidad.

Abstract

Authorization is a key concept in computer security. With this process we demonstrate who we are.

It is common in everyday life that from a computer system where an authentication has been carried out, access to other environments that also require authentication is performed.

This fact causes many drawbacks: authentication fatigue, memory effort, slowness when accessing the system. This also generates problems of security importance because the user usually points the passwords if they are many.

To solve this, SSO systems were created, Single Sign-On, which with a single act of authentication, provides access to a set of related systems.

The advantages are obvious: The user no longer tires, improves their productivity and computer security is strengthened.

The adoption of an SSO system also has drawbacks: If the SSO server stops working, there will be a complete crash of the system. For this reason, a fault-tolerant system is necessary, that is, it must be deployed in a highly available environment.

The purpose of this final master's degree project is to carry out an SSO infrastructure equipped with High Availability.

Keywords: SSO, Authentication, High Availability.

Índice general

1. Introducción. **1**
1.1. Contexto y justificación del trabajo 1
1.2. Objetivos y requisitos. 1
1.3. Metodología a seguir. 1
1.4. Planificación. 2
1.5. Estructura de la memoria. 3
1.6. Conclusiones y trabajos futuros. 3

2. Single Sing On (SSO) **5**
2.1. Conceptos previos: Autenticación y Autorización. 5
2.2. Qué es el Single Sing On. 5
2.3. Tipos de SSO. 7
2.4. Estándares utilizados por SSO. 7
2.5. Implementaciones. 8

3. CAS. **11**
3.1. ¿Qué es CAS? 11
3.2. El protocolo CAS 11
3.2.1. Historia 11

3.3. Características de la implementación CAS 12
3.4. Funcionamiento. 13
3.5. La autenticación en *CAS Server.* 14
3.6. Clientes CAS. 14
3.7. *CASifing* una aplicación java. 15

4. Implementación de un sistema SSO. **19**
4.1. Sistema SSO escogido. 19
4.2. Descripción. 19
4.3. Seguridad. 21
4.4. Entorno de virtualización. 21
4.5. Servidores de aplicaciones. 22
4.6. Balanceo de carga y alta disponibilidad. 23

4.6.1. Balanceadores de carga. 23
4.6.2. Selección del balanceador. 25
4.6.3. HAProxy. 25
4.7. Configuración de LDAP. 28
4.7.1. OpenLDAP. 28
4.7.2. ApacheDS 29
4.7.3. Apache Directory Studio 30
4.8. Servidores CAS. 31
4.8.1. Configuración de CAS para autenticar contra LDAP. 31
4.8.2. Alta disponibilidad en CAS. 32
4.9. La aplicación prototipo. 36
4.10. Instalación y configuración de *Reverse Proxy*. 37

5. Bibliografía. 39

A. Guía para la gestión de certificados digitales. 41

B. Modificación del web.xml para CASeizar. 43

Índice de figuras

1.1. Ciclo de vida en cascada incremental 2

1.2. Planificación del TFM. 3

2.1. Esquema de Funcionamiento de un SSO 6

3.1. Flujo de autenticación de CAS 13

4.1. Infraestructura 20

4.2. Configuración maquina virtual. Primer nodo CAS 21

4.3. Configuración tomcat app1.mistic.test 22

4.4. Detalle de *ca-mistic.test* en Firefox 23

4.5. Prueba tomcat en *app1.mistic.test* 24

4.6. Balanceo de carga 24

4.7. Configuración Haproxy 26

4.8. Estadísticas de HAProxy con los dos nodos levantados 27

4.9. Estadísticas HAProxy con un nodo caído 27

4.10.Configurando OpenLdap. 28

4.11.Contenido del fichero ldif utilizado. 29

4.12.Apache Directory Studio 30

4.13.Autenticación exitosa en CAS/OpenLDAP 32

4.14.Configuración Alta Disponibilidad en CAS. 33

4.15.Configuración Pass-Through de HAProxy 36

4.16.Solicitud autenticación de la aplicación *mistic*. 37

4.17.Aplicación prototipo tras autenticar. 37

Capítulo 1

Introducción.

1.1. Contexto y justificación del trabajo

En la actualidad es común que desde un sistema informático donde se ha realizado una autenticación, se realice un acceso a otros entornos distintos que también precisan de autenticación.

Este hecho complica al usuario final al tener que recordar o mantener credenciales para cada uno de dichos entornos. Se produce la denominada *fatiga de autenticación.* Muchas veces los usuarios recurren a utilizar la misma contraseña para los entornos diferentes constituyendo una amenaza de seguridad clara.

La solución es un sistema SSO (Single Sign On), o autenticación única, donde el usuario se autentica una sola vez en un conjunto de sistemas relacionados.

1.2. Objetivos y requisitos.

El objetivo de este TFM es la realización de un sistema informático dotado de SSO de alta disponibilidad. Esta alta disponibilidad se obtendrá con, al menos, dos servidores SSO balanceados. Los datos de las credenciales se obtendrán de un servidor de directorio LDAP.

El sistema estará compuesto por una red perimetral y una red local protegida.

En la zona desmilitarizada (DMZ) se instalará un *reverse proxy*, que facilitará un acceso transparente a la aplicación ubicada en los servidores de aplicaciones. Estos servidores de aplicaciones estarán configurados en alta disponibilidad.

Las conexiones entre los diversos equipos y sistemas se realizarán utilizando protocolos seguros.

El sistema estará construido utilizando soluciones de software libre/código abierto.

1.3. Metodología a seguir.

La metodología que voy a seguir es una metodología incremental. Tras la primera iteración tenemos una versión ya operativa. Cada iteración va añadiendo nuevas funcionalidades que

cumplen con los requisitos marcados.

Esta metodología de construir el sistema informático es análogo a las metodologías ágiles de Ingeniería de Software (p.e.: metodología Scrum) y también con el ciclo de vida en cascada incremental de la Ingeniería de Software como describe la figura 1.1.

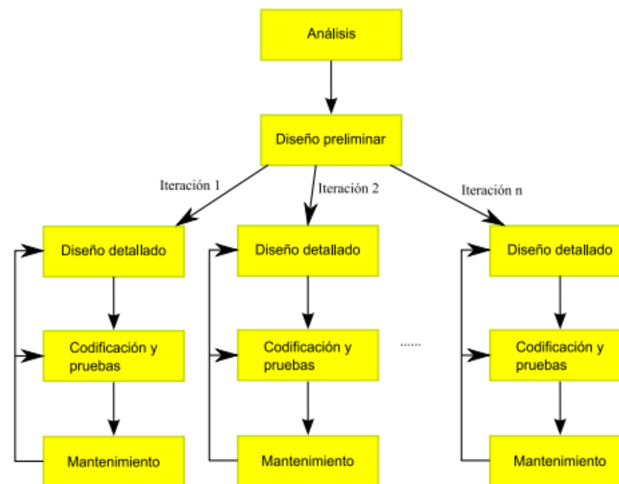

Figura 1.1: Ciclo de vida en cascada incremental

1.4. Planificación.

La planificación ha sido:

WBS	Nombre	Inicio	Fin	Duración
1	Elaboración del plan de trabajo	mar 5	mar 9	5d
2	Entrega PAC-1	mar 11	mar 11	N/D
3	**Instalación de la Infraestructura básica**	**mar 13**	**mar 15**	**3d**
3.1	Preparacion del entorno	mar 13	mar 13	1d
3.2	Preparacion maquitas virtuales necesarias en este punto.	mar 13	mar 13	1d
3.3	Instalación del servidor CAS	mar 14	mar 14	1d
3.4	Instalación del servidor de Aplicaciones.	mar 15	mar 15	1d
4	**Desarrollo y configuración de la Aplicación**	**mar 6**	**abr 9**	**33d 6h**
4.1	Desarrollo de la aplicación prototipo-	mar 16	abr 5	20d
4.2	Adaptación de a aplicación para CAS	mar 6	mar 7	2d
4.3	Pruebas	abr 9	abr 9	1d
5	Entrega PAC-2	abr 9	abr 9	N/D
6	**Configuración del Servicio de Directorio.**	**abr 10**	**abr 17**	**7d 6h**
6.1	Instalación OpenLdap en máquina virtual.	abr 10	abr 13	4d
6.2	Crear repositorio con usuarios y grupos en LDAP	abr 14	abr 16	2d
6.3	Pruebas	abr 17	abr 17	1d
7	**Mejora de la infraestructura.**	**abr 18**	**may 6**	**17d 4h**
7.1	Implementación de la alta disponibilidad en CAS	abr 18	abr 23	5d
7.2	Implementación del clúster del servidor de aplicaciones	abr 24	abr 26	3d
7.3	**Securizacion de red.**	**abr 27**	**abr 30**	**3d**
7.3.1	Configuración de los cortafuegos	abr 27	abr 30	3d
7.4	Pruebas	may 4	may 6	2d
8	Entrega PAC-3	may 8	may 8	N/D
9	Elaboración de la memoria del TFM	may 8	jun 3	26d
10	Entrega PAC-4 Memoria	jun 4	jun 4	N/D
11	Elaboración Video y Presentacion	jun 5	jun 11	6d
12	Defensa del TFM	jun 22	jun 22	N/D

Figura 1.2: Planificación del TFM.

1.5. Estructura de la memoria.

La memoria está estructurada en cuatro capítulos y un apéndice.

- **Introducción.** Es este capítulo. Expone los motivos, objetivos, requisitos, metodología y la planificación de este TFM.
- **SSO.** Describe los aspectos básicos de los sistemas SSO.
- **CAS.** Describe la implementación escogida: CAS.
- **Implementación del SSO.** Constituye el núcleo de esta memoria. Es la parte más extensa al describir los detalles de la implementación.
- **Apéndice B.** Es una guía práctica para trabajar con certificados digitales.
- **Apéndice C.** Expone las modificaciones en el fichero *web.xml* para permitir que la aplicación prototipo utilice el servidor CAS.

1.6. Conclusiones y trabajos futuros.

Este trabajo es eminentemente práctico: La construcción de un sistema SSO. Para ello se ha probado varias soluciones para proveer SSO, alta disponibilidad, servicio de directorio y

reverse proxy.

Los objetivos de este trabajo se han cumplido. Se ha desarrollado una sistema SSO de alta disponibilidad basado en CAS utilizando servicios de directorio como fuente de datos.

El sistema construido daría respuesta a la mayoría de las organizaciones.

Los trabajos futuros tendrían que ir encaminados incorporar otros Authentication Handler's : JAAS, RADUUS, OAuth, x 509, etc.

Capítulo 2

Single Sing On (SSO)

2.1. Conceptos previos: Autenticación y Autorización.

La **autenticación y autorización** son dos conceptos claves en la seguridad informática.

La **autenticación** es el proceso por el que se comprueba que algo o alguien, una aplicación o una persona, es quien dice ser.

La entidad se autentica mediante “algo que sabe”, como es una contraseña; “algo que es”, un elemento biométrico, o ;“algo que tiene”, como un DNI electrónico.

Para mejorar la seguridad se puede implantar la **autenticación de múltiples factores (AMF)** que es un método de control de acceso informático en el que a un usuario se le concede acceso al sistema solo después de que presente dos o más pruebas diferentes de que es quien dice ser. Estas pruebas pueden ser diversas, como una contraseña, que posea una clave secundaria rotativa, o un certificado digital instalado en el equipo, entre otros.

La **autorización** es el proceso por el que se comprueba que un usuario o aplicaciones tienen los permisos para acceder a los recursos.

2.2. Qué es el Single Sing On.

Single sign-on (SSO) es un procedimiento de autenticación que habilita al usuario para acceder a varios sistemas con una sola instancia de identificación[1].

La figura2.1 representa[2] el esquema de funcionamiento de un SSO. Vemos que el usuario se autentica una única vez en el Dominio *Single Sign On* y obtendría acceso a dominios de identidad relacionados, manteniéndose la obligación de autenticarse para Dominios de identidad excluidos del sistema SSO.

[1]Wikipedia.

[2]Palazón, J.M., et al.(s.f.). Unidades didácticas de Identidad Digital.Barcelona. UOC

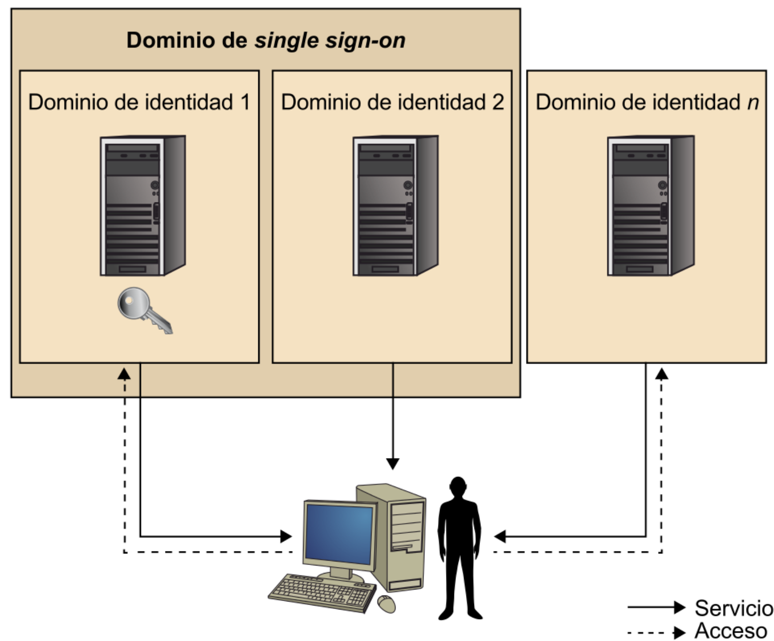

Figura 2.1: Esquema de Funcionamiento de un SSO

Ventajas de SSO.

- La principal ventaja es que desaparece la comentada *fatiga de autenticación.*
- Aumenta la velocidad de utilización. Al no necesitar recurrentes peticiones de identidad.
- No hay que hacer esfuerzo memorístico ya que sólo se precisa una contraseña.
- Implementación muy sencilla.
- Numerosas fuentes de datos para las credenciales. Podemos utilizar BBDD, Servicios de directorio, etc.

Desventajas de SSO.

Sin embargo un sistema SSO también adolece de inconvenientes.

- Mayor impacto en el caso de pérdida de credenciales. El usuario pierde el control de todos los sistemas relacionados.
- Mayor riesgo de producirse una suplantación de identidades en los accesos externos de los usuarios.
- Si el sistema donde reside este mecanismo se cae, se puede producir una denegación de acceso a todos los sistemas relacionados. Por ello el sistema SSO tiene carácter crítico y tendrá que ser un sistema de *High Availability (HA)*

2.3. Tipos de SSO.

Las soluciones existentes implementan al menos uno de los siguientes cinco paradigmas[3]:

1. **Enterprise single sign-on (E-SSO)**. Interceptan las peticiones de autenticación. Los sistemas E-SSO permiten interactuar con sistemas que pueden deshabilitar la presentación de la pantalla de login.

2. **Web single sign-on (Web-SSO).**Trabaja sólo con aplicaciones y recursos accedidos vía web. Los accesos son interceptados (por ejemplo mediante un proxy). Los usuarios no autenticados que tratan de acceder, son redirigidos a un servidor o servicio web de autenticación y regresan solo después de haber logrado un acceso exitoso o con un TOKEN de autenticación para la aplicación destino. Se utilizan cookies, parámetros por GET (más inseguro) o POST para reconocer aquellos usuarios que acceden y su estado de autenticación.

3. **Kerberos.** es un método popular de externalizar la autenticación de los usuarios. Los usuarios se registran en el servidor Kerberos y reciben un "ticket", luego las aplicaciones-cliente lo presentan para obtener acceso del recurso solicitado.

4. **Identidad federada.** Utiliza protocolos basados en estándares para habilitar que las aplicaciones puedan identificar los clientes sin necesidad de autenticación redundante.

5. **OpenID.** es un proceso de SSO distribuido y descentralizado donde la identidad se compila en una url que cualquier aplicación o servidor puede verificar.

2.4. Estándares utilizados por SSO.

Las implementaciones se apoyan en estándares.

- **SAML.** Es el acrónimo de *Security Assertion Markup Language*. Es un estándar XML para el intercambio de información de autenticación y autorización entre dominios. Es un producto de OASIS. Las entidades que participan en este protocolo son:
  - Proveedor de Identidad. Entidad que dispone de la infraestructura necesaria para la autenticación de los usuarios.
  - Proveedor de servicios. Entidad que concede a un usuario el acceso o no a sus recurso.
  - Principal. Podemos verlo como el usuario.

[3]Fuente: Wikipedia.

- **OpenID.** Este estándar proporciona un mecanismo de identificación descentralizado a través de una URL o , en la versión más reciente, XRI (eXtensible Resource Identifier). Los usuarios no necesitan poseer una cuenta el los servidores que soporten este estándar, sólo necesitan poseer el identificador proporcionado por un proveedor de identidad (IdP). Un IdP es un servidor que verifica OpenID.

- **OAuth.** Estándar que permite el flujo simple de información para sitios web o aplicaciones informáticas. Permite que un usuario de un sitio A pueda acceder a otro sitio B sin compartir toda su identidad. Es decir, OAuth proporciona un mecanismo de acceso a datos manteniendo protegidas las credenciales del usuario. Es un mecanismo utilizado por compañías como Google, Facebook, Microsoft, Twitter y Gihub.

- **XACML.** Acrónimo de *eXtensible Access Control Markup Language.* Define un lenguaje declarativo de control de acceso implementado en XML. Es utilizado fundamentalmente para sistemas de control de acceso a los recursos basado en atributos (ABAC).

- **SCIM.** Es el acrónimo de *System for Cross-domain Identity Management.* Constituye un estándar abierto para automatizar el intercambio de información de identidad de usuario entre dominios de identidad o sistemas de TI.

- **WS Federation** (*Web services Federation*). WS-Federation define mecanismos para permitir que diferentes dominios de seguridad intercambien información sobre identidades, atributos de identidad y autenticación. Estándar desarrollado por compañias como IBM, Microsoft, Novell, HP.

2.5. Implementaciones.

Son numerosas las implementaciones que proporcionan SSO. En la Wikipedia muestra esta buena enumeración sistemas SSO

De estos sistemas, considerando el propósito de este TFM, son de mayor interés los basados en licencias OpenSource/FreeSoftware.

La siguiente relación son implementaciones muy utilizadas con las características mas relevantes.

- **CAS /Central Authentication Service.** Implementación con amplio soporte. Posee licencia Apache 2.0. El desarrollo de CAS de *Apereo* es bastante conocido. Soporte para CAS, SAML1, SAML2, OAuth2, SCIM, OpenID Connect y WS-Fed protocolos. Se describirá en un capítulo aparte.

- **JBoss SSO.** Desarrollado por Red Hat. SSO federado. Free Software.

- **JOSSO.** Desarrollado por *JOSSO.* SSO Server. Free Software.

- **Keycloak.** SSO federado desarrollado por *Red Hat.* Soporta los protocolos normalizados: OpenID Connect, OAuth 2.0 and SAML 2.0 para la Web, clustering y single sign on.

- **OpenAM.** Solución de la empresa *Forge Rock*. Tiene su origen en *OpenSSO* desarrollado por *SUN*. LLeva a cabo la gestión de accesos, derechos y la plataforma del servidor de federación. Arquitectura basada en Java con soporte para los protocolos: SAML, WS-Federation, OpenID y XACML.

- **Shibboleth.** Proyecto de identidad federada, Con licencia Apache. Se apoya en el estándar SAML.

- **WSO2** Identity Server. Servidor de identidades creado por WSO2, creador del Enterprise Service Bus WSO2, con soporte para: SAML 2.0, OpenID, OpenID Connect, OAuth 2.0, SCIM, XACML, Federación pasiva.

Capítulo 3

CAS.

3.1. ¿Qué es CAS?

CAS es el acrónimo de *Central Authentication Service.* Es un tanto un protocolo como una implementación de SSO para la Web (*Web Single Sign On*).

Web Single Sign On funciona estrictamente con aplicaciones a las que se accede con un navegador web. La solicitud para acceder a un recurso web es interceptada por un componente en el servidor web o por la propia aplicación. Los usuarios no autenticados se desvían a un servicio de autenticación y se devuelve a la aplicación sólo después del éxito en la autenticación.

La implementación fue desarrollada por *Apareo* y soporta más protocolos aparte de CAS.

3.2. El protocolo CAS

Utiliza la autenticación "federada", donde toda la autenticación se realiza en el Servidor CAS, en lugar de servidores de aplicaciones individuales.

El protocolo CAS implica al menos tres componentes:

- Un navegador web de cliente,
- La aplicación web que solicita autenticación.
- El servidor CAS.

3.2.1. Historia

CAS fue concebido y desarrollado por Shawn Bayern de Tecnología y Planificación de la Universidad de Yale. Más tarde fue mantenido por Drew Mazurek en Yale. CAS 1.0 implementó el inicio de sesión único. CAS 2.0 introdujo la autenticación de proxy de múltiples niveles. Varias otras distribuciones CAS se han desarrollado con nuevas características.

En diciembre de 2004, CAS se convirtió en un proyecto de Java en el Grupo de Interés Especial de Administración (JASIG), que a partir de 2008 es responsable de su mantenimiento y desarrollo. Anteriormente llamado "Yale CAS", CAS ahora también se conoce como "Jasig CAS". En 2010, Jasig entró en conversaciones con la Fundación Sakai para fusionar las dos organizaciones. Ambas organizaciones se consolidaron como Apereo Foundation en diciembre de 2012.

En diciembre de 2006, la Fundación Andrew W. Mellon otorgó a Yale su Primer Premio Anual Mellon de Colaboración Tecnológica, por un monto de $ 50,000, para el desarrollo de CAS de Yale. [2] En el momento de ese premio CAS se usaba en "cientos de campus universitarios (entre otros beneficiarios)".

En mayo de 2014, se publicó la especificación 3.0 del protocolo CAS. [3]

3.3. Características de la implementación CAS

- Esta implementación es Open Source.
- Proporciona el protocolo Open Source CAS muy bien documentado.
- Servidor Java.
- Integración con un amplio abanico de clientes: Java, .Net, PHP, Perl, Apache, uPortal y otros.
- Proporciona numerosos módulos que permiten utilizar varios métodos de autenticación. Entre otros tenemos: LDAP, base de datos, X.509, JASS, RADIUS, SPINEGO, Apache Cassandra, Remote Adress, JWT, Rest.
- Soporte para múltiples protocolos. Además de soportar el protocolo propio,CAS, soporta los siguientes:
  - OpenID
  - OAuth
  - OpenID Connect
  - WS Federation
  - SAML1
  - SAML2
  - REST Protocol.
- Se integra con uPortal, BlueSocket, TikiWiki, Mule, Liferay, Moodle y otros Documentación de la comunidad y soporte de implementación.
- Tiene una amplia base de desarrolladores. Más de cuarenta universidades, entre otras entidades, participan en el desarrollo.

- Numerosos casos de éxito. Es de reseñar la valoración que hizo la universidad de Murcia para proveerse de un mecanismo de SSO.
- Abundante software de terceros dan soporte a CAS como mecanismo de autenticación.
- Excelentes características de escalabilidad.

3.4. Funcionamiento.

La figura[1] 3.1 describe el proceso de autenticación en CAS en una aplicación web.

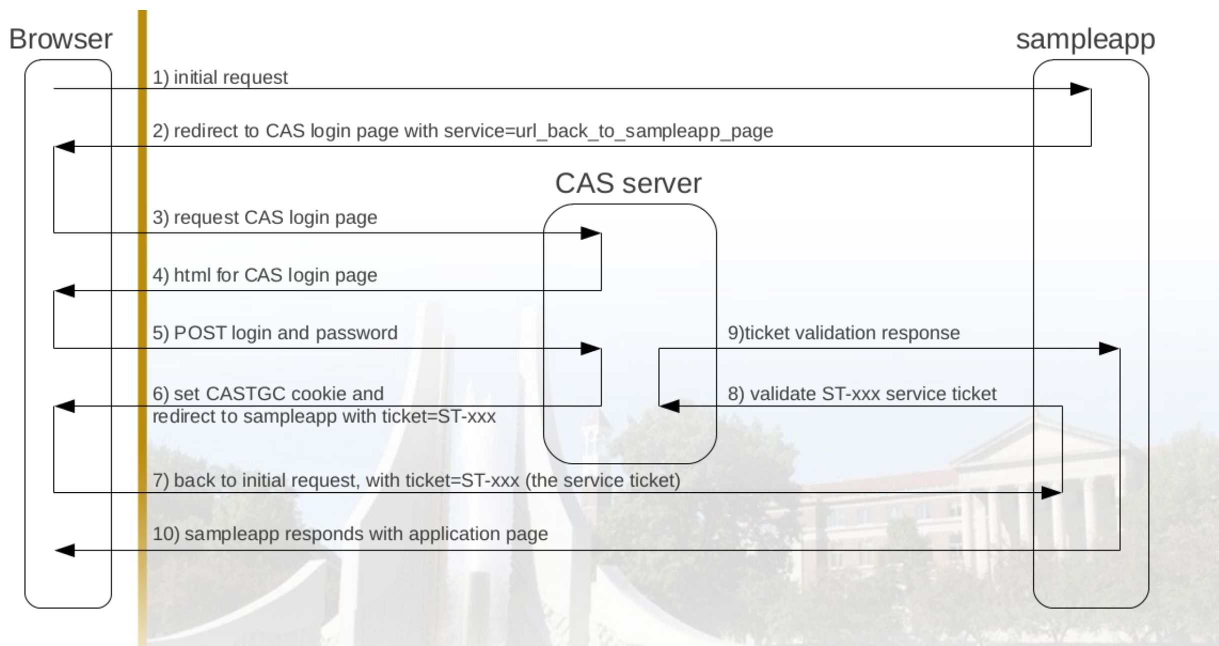

Figura 3.1: Flujo de autenticación de CAS

Básicamente se realiza lo siguiente:

1. En la aplicación existe un agente cliente CAS encargado de comprobar la existencia y validez de un ticket necesario para el acceso del usuario.
2. El usuario entra desde el navegador a la aplicación mediante la URI.
3. La aplicación comprueba si la petición contiene un ticket válido. Si existe se da acceso.
4. Si no lo tiene se redirige a la página de autenticación del servidor CAS indicando donde tiene que volver en el caso de autenticación satisfactoria.

[1]Fuente: Universidad de Purdue (EEUU).

5. Si el servidor CAS autentica al usuario generará un ticket. El cliente recibirá una cookie. Se volverá a solicitar el acceso(de forma transparente al usuario) a la aplicación pero suministrando el ticket. Tras validarlo por parte del agente cliente CAS se mostrará la página de la aplicación.

6. Si no se autentica, el servidor CAS mostrará una pantalla de error.

CAS permite la autenticación de múltiples niveles a través de la dirección proxy. Un servicio de back-end cooperativo, como una base de datos o servidor de correo, puede participar en CAS, validando la autenticidad de los usuarios a través de la información que recibe de las aplicaciones web. Por lo tanto, un cliente de correo web y un servidor de correo web pueden implementar CAS.

3.5. La autenticación en *CAS Server.*

La autenticación en CAS está controlada por el *Authentication Manager* que gestiona los diversos manejadores de autenticación, *autentication handlers*, existentes.

Cuando se le proporciona una credencial el *Authentication Manager* procede de la siguiente manera:

1. Itera sobre todos los *Autentication handlers* configurados.

2. Intenta autenticar una credencial, si un controlador *Autentication handler* la admite, para resolver un principal.

3. Verifica si la política de seguridad (por ejemplo si se tienen que cumplir todas las autenticaciones o una, etc) está satisfecha.

4. Si en base a los *Autentication Handlers* y a la política establecida no se ha autenticado , el sistema lanzará una *AuthenticationException*.

3.6. Clientes CAS.

Toda esta información está referida a la versión 5.2.x de CAS.

Es necesario instalar un cliente en la aplicación para que esta se pueda comunicar con el servidor CAS.

Apereo informa de cuatro clientes oficiales:

- .NET CAS Client
- Java CAS Client
- PHP CAS Client
- Apache CAS Client

Además existen otros clientes no oficiales y otros tantos en fase de desarrollo. Otra posibilidad es el desarrollo de nuestro propio cliente.

Al proceso de adaptar la aplicación para utilizar el cliente CAS se le denomina *CASifing*. Los siguientes Frameworks ya incorporan el soporte CAS:

- Spring Security
- Apache Shiro
- Pac4j

3.7. *CASifing* una aplicación java.

Hay que incorporar la librería *cas-client-core* al proyecto Java.
Utilizando *maven* se añade la siguiente dependencia en el archivo *pom.xml*.

```
<dependency>
    <groupId>org.jasig.cas.client</groupId>
    <artifactId>cas-client-core</artifactId>
    <version>3.4.1</version>
</dependency>
```

Después hay que modificar el descriptor de despliegue *web.xml* para filtrar las peticiones y dirigir al CAS server en el caso de no estar autenticado.

```
<filter>
  <filter-name>CAS Single Sign Out Filter</filter-name>
  <filter-class>org.jasig.cas.client.session.SingleSignOutFilter</filter-class>
<init-param>
  <param-name>casServerUrlPrefix</param-name>
  <param-value>https:cas.mistic.test:8443/cas</param-value>
</init-param>
</filter>
<listener>
  <listener-class>
    org.jasig.cas.client.session.SingleSignOutHttpSessionListener
  </listener-class>
</listener>
<filter>
  <filter-name>CAS Authentication Filter</filter-name>
    <!--<filter-class>
      org.jasig.cas.client.authentication.Saml11AuthenticationFilter
    </filter-class>-->
    <filter-class>
      org.jasig.cas.client.authentication.AuthenticationFilter
    </filter-class>
```

```
        <init-param>
          <param-name>casServerLoginUrl</param-name>
          <param-value>https://cas.mistic.test:8443/cas/login</param-value>
        </init-param>
        <init-param>
          <param-name>serverName</param-name>
          <param-value>https:app.mistic.test:8443</param-value>
        </init-param>
    </filter>
    <filter>
        <filter-name>CAS Validation Filter</filter-name>
        <!--<filter-class>
            org.jasig.cas.client.validation.Saml11TicketValidationFilter
        </filter-class>-->
        <filter-class>
            org.jasig.cas.client.validation.
                Cas30ProxyReceivingTicketValidationFilter
        </filter-class>
        <init-param>
        <param-name>casServerUrlPrefix</param-name>
        <param-value>https://cas.mistic.test:8443/cas</param-value>
        </init-param>
        <init-param>
            <param-name>serverName</param-name>
            <param-value>https://app.mistic.test:8443</param-value>
        </init-param>
        <init-param>
            <param-name>
                redirectAfterValidation
            </param-name>
            <param-value>true</param-value>
        </init-param>
        <init-param>
            <param-name>useSession</param-name>
        <param-value>true</param-value> </init-param>
        <!-- <init-param>
            <param-name>acceptAnyProxy</param-name>
            <param-value>true</param-value>
        </init-param>
        <init-param>
            <param-name>proxyReceptorUrl</param-name>
            <param-value>/sample/proxyUrl</param-value>
        </init-param>
        <init-param>
            <param-name>proxyCallbackUrl</param-name>
            <param-value>
            https://app.mistic.net:8443/sample/proxyUrl
        </param-value>
        </init-param> -->
        <init-param>
        <param-name>authn_method</param-name>
```

```
        <param-value>mfa-duo</param-value>
    </init-param>
</filter>
<filter>
    <filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>
    <filter-class>
        org.jasig.cas.client.util.HttpServletRequestWrapperFilter
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>CAS Single Sign Out Filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>CAS Validation Filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>CAS Authentication Filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Los tipos de filtros usados son:

- **CAS Authentication Filter.** Detecta si el usuario está autenticado. Si no lo está, redirige al servidor CAS y, tras la autenticación retorna a la dirección especificada en *service*.
- **CAS Validation Filter.** Valida el ticket aportado por el usuario en la solicitud.
- **CAS HttpServletRequest Wrapper Filter.** Se encarga de actualizar el objeto *request* de la clase *HttpServletRequest*. De esta forma obtenemos el usuario que se ha autenticado.

Capítulo 4

Implementación de un sistema SSO.

4.1. Sistema SSO escogido.

La solución escogida para el servicio SSO será CAS por las siguientes razones:

- Producto Open Source.
- Tiene una amplia base de desarrolladores. Más de cuarenta universidades, entre otras entidades, participan en el desarrollo.
- Numerosos casos de éxito. Es de reseñar la valoración que hizo la universidad de Murcia para proveerse de un mecanismo de SSO.
- Abundante software de terceros dan soporte a CAS como mecanismo de autenticación.
- Buenas características de escalabilidad.

4.2. Descripción.

Es necesario establecer un nombre de dominio para configurar los servidores, los certificados y el servicio de directorio.

El nombre que utilizaré es mistic.test (dc=mistic, dc=test).

La figura 4.1 representa el sistema desarrollado.

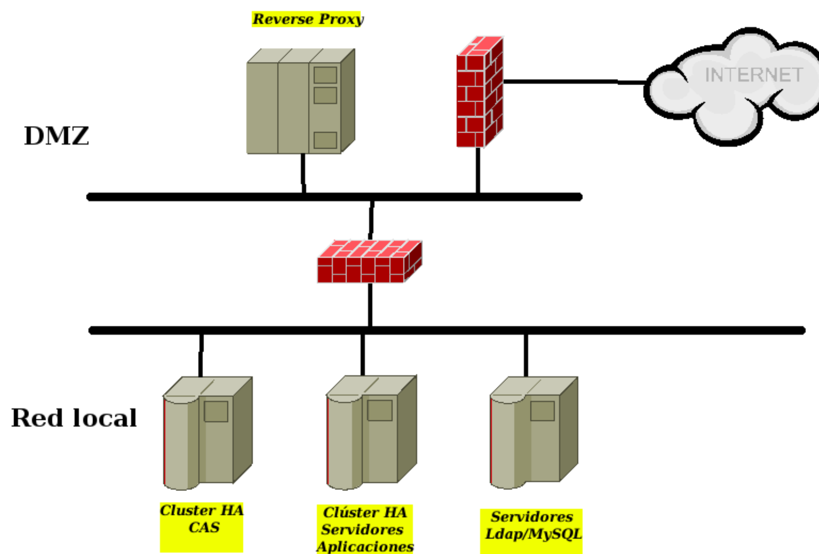

Figura 4.1: Infraestructura

Los servidores utilizados son:

Hostname	**IP**	**Función**
app1.mistic.test	192.168.1.30/24	Nodo 1 del clúster de aplicaciones
cas1.mistic.test	192.168.1.31/24	Nodo 1 del clúster CAS
ldap.mistic.test	192.168.1.32/24	Servidor LDAP y BBDD
app.mistic.test	192.168.1.33/24	Balanceador clúster aplicaciones
app2.mistic.test	192.168.1.34/24	Nodo 2 del clúster de aplicaciones
cas2.mistic.test	192.168.1.35/24	Nodo 2 del clúster CAS
cas.mistic.test	192.168.1.36/24	Balanceador clúster CAS
front.mistic.test	192.168.1.37/24	Reverse Proxy cortafuegos.

Todas las máquinas están virtualizadas y son máquinas Debian-9.3.1.

Se instala en todos los equipos el certificado de la autoridad certificadora *ca.mistic.test* tal como describo más adelante.

Los servidores de aplicaciones están realizados con *Apache Tomcat 8.5.27* .

Los balanceadores ejecutan HAProxy v1.7.5-2.

En la máquina *ldap.mistic.test* mantenemos dos servicios de directorio LDAP: OpenLDAP (puertos 389 y 636) y ApacheDS v (puertos 10389 y 10636). Esta máquina también ejecuta el servidor de BBDD *MySQL* utilizada por la aplicación prototipo inicial.

Los nodos CAS ejecutan la versión 5.2.4.

Para resolver los nombres adecuadamente introduzco los datos de los servidores en */etc/hosts*.

4.3. Seguridad.

Se hacen las siguientes consideraciones:

- A los nodos de los clústers no se puede acceder sino a través de sus front-end's.
- Los balanceadores sólo admiten conexiones al puerto 443 provenientes de la red local o la DMZ.
- Se admiten las conexiones necesarias para administrar los clústers intentando que sean transitorias.
- La máquina *ldap.mistic.test* sólo tiene que conectar con los clústers.
- Se implementan estas consideraciones mediante *Iptables* .

4.4. Entorno de virtualización.

El sistema anfitrión está basado en *Debian Linux 9.3* que ejecuta un kernel 4.9.0-6-amd64 y se ejecuta en una máquina Intel i7 con 24GB de memoria.

La herramienta utilizada para la virtualización ha sido VirtualBox v5.2.12.

Se realiza una máquina virtual que será clonada para construir los 8 *hosts* necesarios. Cada una de estos hosts estará dotado de 2 GB de memoria y 1 procesador.

Se configura el adaptador de red en modo *bridge*. Al configurarlo así, la máquina virtual tiene una dirección IP perteneciente al mismo rango de red que la máquina anfitriona (192.168.1.0/24).

La figura 4.4 muestra la ventana de configuración de la máquina virtual correspondiente al segundo nodo del clúster CAS.

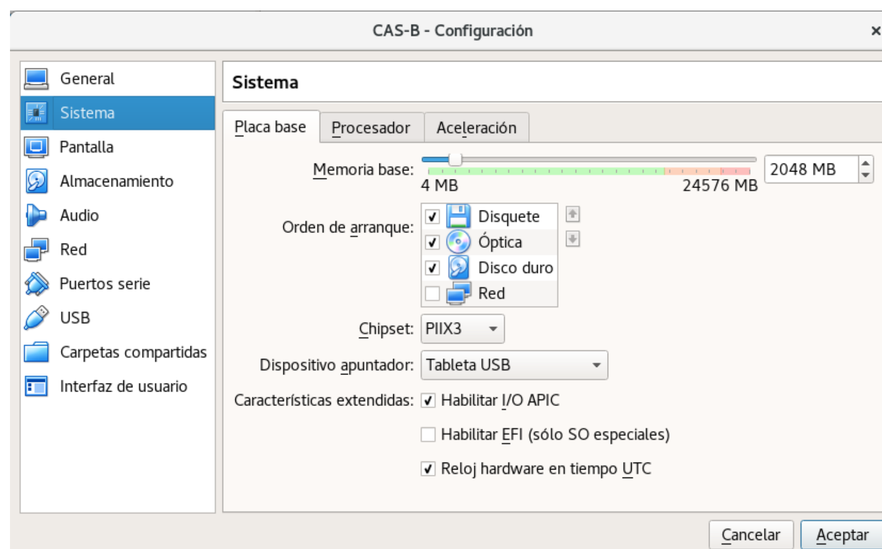

Figura 4.2: Configuración maquina virtual. Primer nodo CAS

Normalmente, las MVs se están ejecutando con un inicio "sin pantalla", esto es, en un entorno sin ventanas. Esto evita consumir más recursos de los necesarios. Al realizarse una medición de carga se comprueba que tenemos recursos suficientes.

4.5. Servidores de aplicaciones.

En esta implementación tendremos dos servidores de aplicaciones app1.mistic.net y app2.mistic.net que formarán parte de un clúster de alta disponibilidad.

Están implementados mediante un servidor *Apache Tomcat 8.5.27*. Estrictamente hablando se trata de contenedores de servlets pero son adecuados para desplegar la mayoría de aplicaciones web JEE.

El servidor tomcat lo coloco en /usr/tomcat.

Se accederá a cada uno de ellos mediante los puertos 8080 y 8443 (https).

La conexión segura se configurará utilizando el certificado de servidor emitido a su nombre. En el caso de app1.mistic.net tendrá los siguientes datos:

```
C = ES, ST = Valladolid, L = Valladolid, O = Mistic, CN = app1.mistic.test
```

La cuestión del nombre del certificado es importante dado que un cliente que comunique con el servidor con una conexión segura, sólo confiará si el certificado que presenta este último corresponde con el nombre del servidor.

Este certificado lo importo a un *keystore p12*. En dicho almacén de claves también sitúo el certificado de nuestra autoridad de certificación (*CA*): ca-mistic.test.

Seguidamente configuramos el tomcat para que utilice la conexión segura modificando el fichero */usr/tomcat/conf/server.xml*. Esto lo hago descomentando y modificando, la entrada siguiente:

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
               maxThreads="150" SSLEnabled="true">
        <SSLHostConfig>
            <Certificate certificateKeystoreFile="conf/app2.p12"
                         certificateKeystorePassword="changeit"
                         certificateKeyAlias="tomcat"
                         type="RSA" />
        </SSLHostConfig>
    </Connector>
```

Figura 4.3: Configuración tomcat app1.mistic.test

En *app1.mistic.test* ejecuto idénticos pasos utilizando su certificado.

Como se ve, he dado al certificado el alias de "tomcat". La contraseña de acceso al p12 es "changeit".

Nuestro servidor va a comunicar con el servidor CAS de forma segura. Para que esto pueda realizarse es preciso importar el certificado de la autoridad certificadora (ca-mistic.test) al almacén de claves *cacerts* ubicado en *$JAVA_HOME/lib/security*

```
# cd $JAVA_HOME/lib/security
# keytool -import -alias mistic -keystore cacerts -trustcacerts -file ca-mistic.crt
```

La contraseña por defecto de *cacerts* es “changeit”.

En los navegadores importaremos el certificado ca-mistic.test. La figura 4.5 siguiente muestra los detalles de éste:

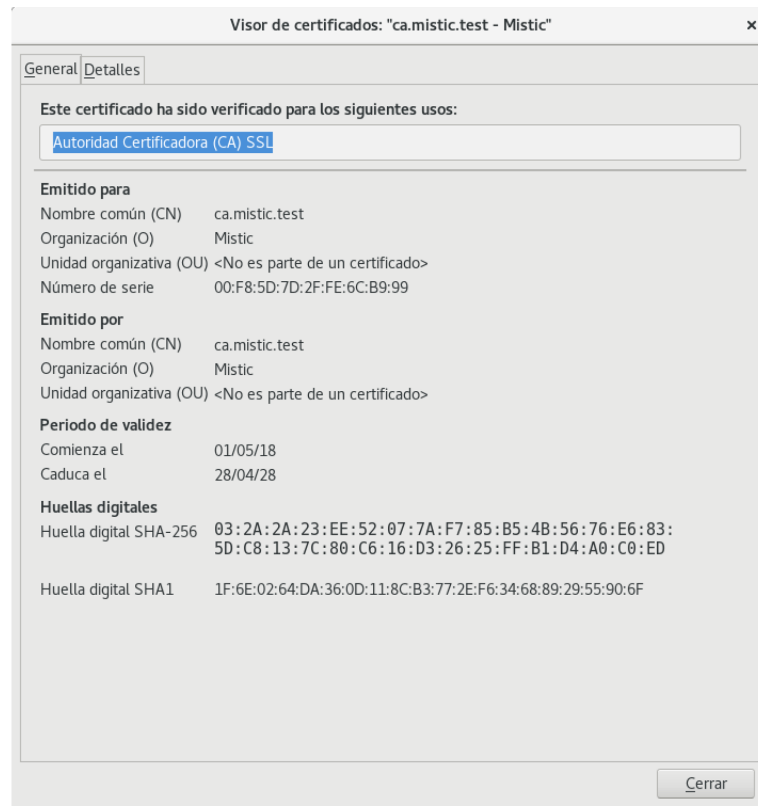

Figura 4.4: Detalle de *ca-mistic.test* en Firefox

Probando el resultado final:

4.6. Balanceo de carga y alta disponibilidad.

4.6.1. Balanceadores de carga.

Un balanceador de carga es un dispositivo hardware o software que se pone al frente de un conjunto de servidores distribuyendo los trabajos entre ellos. El balanceador proporciona una abstracción: El usuario del sistema tiene la sensación de un sólo equipo (el front-end) cuando realmente hay varios.

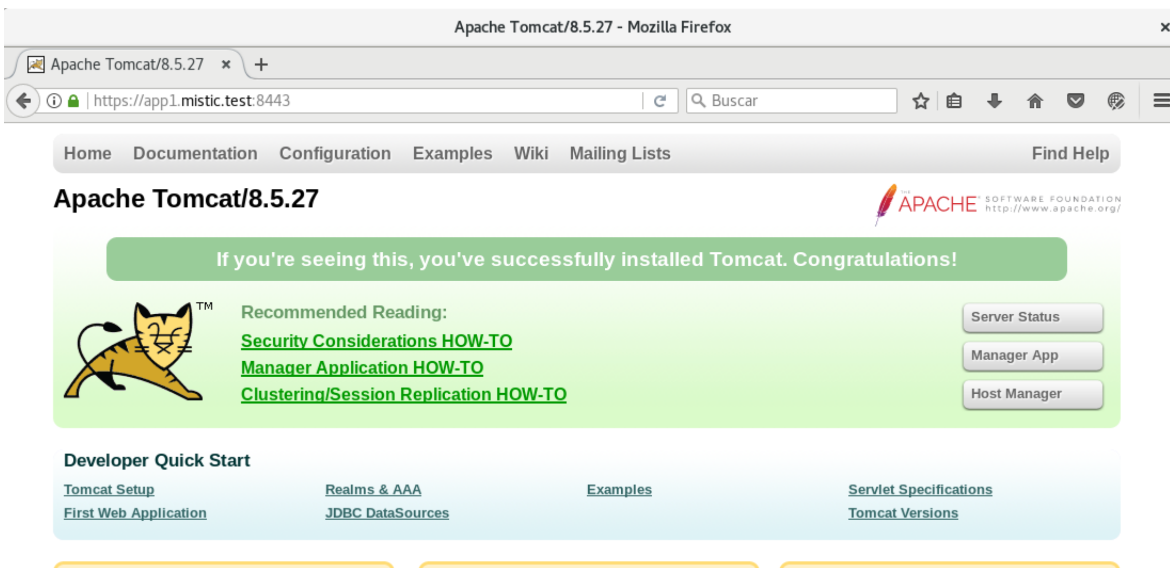

Figura 4.5: Prueba tomcat en *app1.mistic.test*

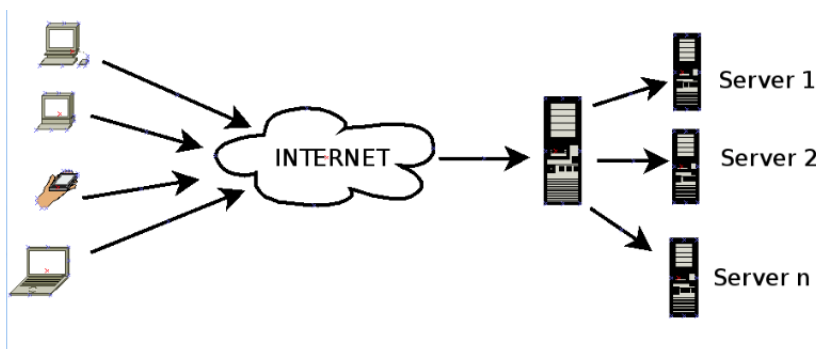

Figura 4.6: Balanceo de carga

Tipos de balanceadores:

- **Hardware.** El balanceador se presenta como un dispositivo físico especialmente concebido para esta tarea.
- **Software.** El balanceador es un programa que corre en un ordenador.
- **Basado en DNS.** Se configura el servidor para que una URI apunte a varias IP's. Mecanismo muy sencillo.

En cuanto a la política de distribución de las conexiones, nos encontramos con:

- **Round Robin**. El servidor distribuye las peticiones de forma cíclica.
- **Weighted Round Robin**. Es una variante del anterior. En este caso a cada servidor se le puede asignar un peso, Permite variar los porcentajes de asignación de peticiones. Es adecuado cuando tenemos un clúster heterogéneo formado con máquinas de distinta potencia.

- **Weighted LeastConnections**. Las peticiones se entregan al servidor que mantiene menos conexiones.
- **Ip-Hash.** El servidor asignado depende del tipo de petición. Se utiliza, como el nombre indica, para asignar las peticiones en función de la IP solicitante.

4.6.2. Selección del balanceador.

Evidentemente, el balanceador utilizado es del tipo software. Busco soluciones que sean proxy's y balanceadores.

Los programas que he analizado son:

- ***Nginx.*** Es el servidor web más utilizado como *reverse proxy*. Permite realizar el balanceo de carga. Su versión comerciar, *Nginx plus*, incorpora prestaciones interesantes para realizar el balanceo de carga.
- ***Apache2.*** Es un servidor web. Debido a su arquitectura, podemos cargar módulos para proporcionar el balanceado como puede ser *mod_jk.*
- ***HAProxy.*** No es un servidor web sino un proxy que proporciona balanceo de carga con alta disponibilidad. Es significativo que sea el alma del balanceador HA hardware ALOHA.

En los tres casos la configuración no es complicada, pero en el caso de HAProxy es realmente sencilla.

La arquitectura Apache2 es más escalable por admitir plugins. *Nginx* es la opción más utilizada como proxy inverso. Ambos casos son buenas soluciones proporcionando diversos servicios: proxy, servidor web, balanceo de carga, etc. Pero lo que necesitamos no es un sistema multipropósito sino un balanceador eficiente con capacidades avanzadas de enrutamiento. Por ello la opción elegida es HAProxy.

4.6.3. HAProxy.

Consideraciones Previas.

Hay dos formas de establecer las conexiones:

- **Terminación SSL**. La conexión SSL acaba en el balanceador, distribuyendo éste las peticiones a los nodos sin cifrar. Como el balanceador tiene que realizar el cifrado/descifrado tiene más sobrecarga. La seguridad de la conexión entre los servidores habría que buscarla por otras vías: Líneas dedicadas(posible dado la proximidad de los servidores), VPN's, etc.

- **SSL Pass-Through.** El balanceador envía las peticiones SSL directamente a los servidores. Es más seguro que el anterior. Sin embargo se pierde la capacidad de agregar o editar encabezados HTTP, ya que la petición simplemente se enruta. Los servidores de aplicaciones son incapaces de obtener los encabezados X-Forwarded-* , que pueden incluir la dirección IP, el puerto y el esquema del cliente utilizados.

La solución escogida para el clúster es la primera: Terminación SSL. De esta forma los servidores de aplicaciones estarán menos limitados.

Para esta configuración necesitamos un certificado de servidor emitido por nuestra CA : *ca-mistic.test.* Lo expedimos para app.mistic.test.(Tenemos que utilizar siempre el certificado del front-end)

Lo que necesita HAProxy es un certificado en formato pem. Para ello concateno la clave pública y privada del servidor. Además he concatenado la clave pública de *ca-mistic.test.*

Es posible utilizar dos configuraciones: capa 4 y capa 7. Esta última se utiliza cuando las aplicaciones se encuentran distribuidas en diferentes servidores. Utilizaré una configuración de capa 4 dado que la aplicación reside completa en el servidors tomcat.

Configuración de HAProxy

El servidor *app.mistic.tes*t es el encargado de balancear los servidores de aplicaciones.

Primero lo instalamos en un ordenador *debian.*

```
# apt-get install haproxy
```

Una vez que esté instalado procedemos a la configuración. Para ello edito el fichero *haproxy.cfg* situado en */etc/haproxy*

```
frontend http_front
   bind *:443 ssl crt /etc/ssl/certificados/app.pem
   stats uri /hastats
   default_backend http_back

backend http_back
   balance roundrobin
   cookie SERVERID insert indirect nocache
   server  app1 192.168.1.30:8080 check
   server  app2 192.168.1.34:8080 check
```

Figura 4.7: Configuración Haproxy

Con estas entradas configuramos un clúster con los nodos app1 y app2 que sólo atiende en el puerto 443 con protocolo SSL. La comunicación con los nodos se realiza a través de conexiones no seguras a través del puerto 8080.

Con la entrada `cookie SERVERID insert indirect no cache`, permitimos la persistencia de sesiones. La persistencia de sesiones es un requisito para que las aplicaciones puedan funcionar.

Indicamos que el algoritmo utilizado para la asignación de servidor es *roundrobin.*
Para ver las estadísticas podemos ir a la *URL https://<nombre_servidor>/hastats*
Reiniciando el balanceador:

```
# systemctl restart haproxy
```

Compruebo que funciona perfectamente. La figura 4.8 muestra las estadísticas de HAProxy. Tras apagar el servidor app2 el sistema sigue funcionando bien y la página de HAProxy informa de esta condición (figura 4.9)

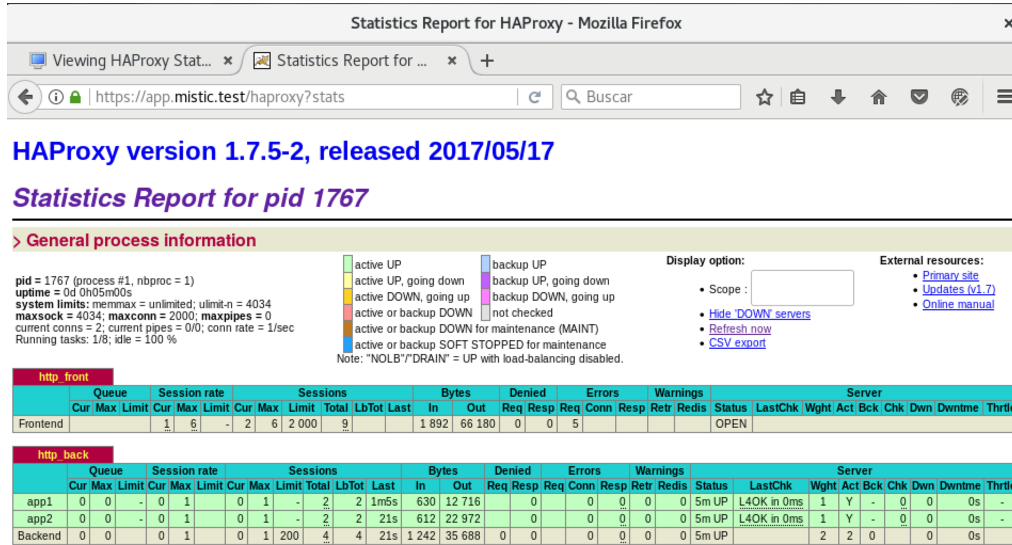

Figura 4.8: Estadísticas de HAProxy con los dos nodos levantados

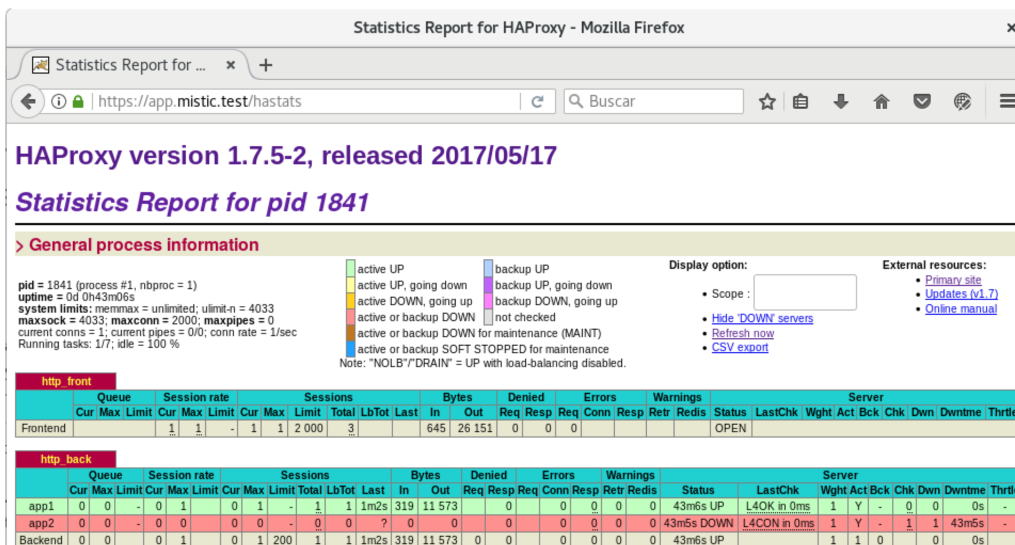

Figura 4.9: Estadísticas HAProxy con un nodo caído

4.7. Configuración de LDAP.

Configuraré un servicio de directorio que será utilizado por CAS para autenticar.

Hay varias soluciones Open Source que implementan el protocolo LDAP. Las mas conocidas son OpenLDAP y ApacheDS. He instalado las dos con la configuración por defecto:

- OpenLDAP corriendo contra los puertos 385 y 636(acceso seguro).
- ApacheDS utiliza 10385 y 10636 (acceso seguro).

4.7.1. OpenLDAP.

Para instalar el servidor del directorio OpenLDAP ejecuto en la máquina *ldap.mistic.test:*

```
# apt-get install slapd
```

Es recomendable instalar *ldap-utils* para facilitar la comunicación con el servidor Ldap.

Una vez instalado procedo a configurarlo utilizando el nombre de dominio *mistic.test (dc=mistic, dc=test).* Para ello utilizo el comando:

```
# dpkg-reconfigure slapd
```

Mostrándose la imagen de la figura 4.10.

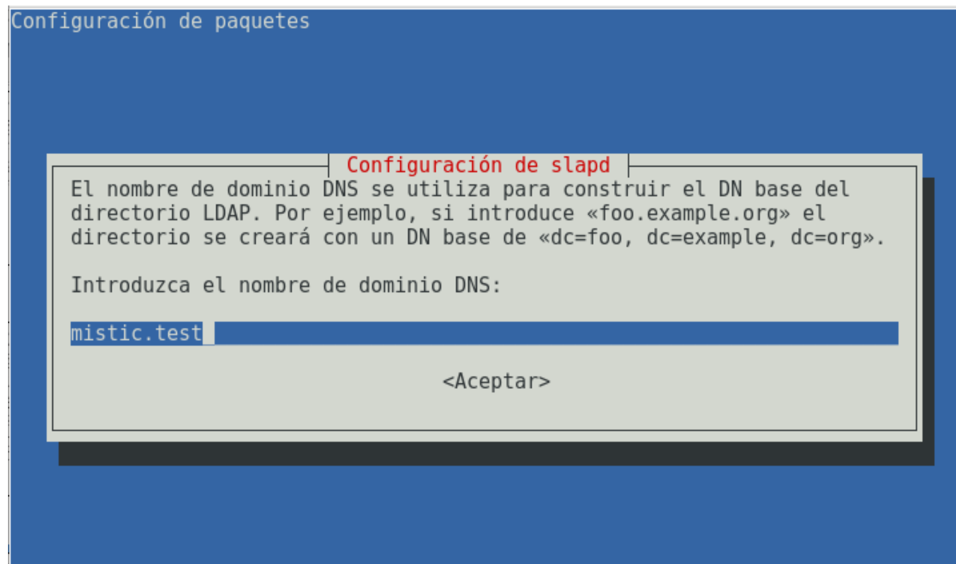

Figura 4.10: Configurando OpenLdap.

Arranco el servicio:

```
# service slapd start
```

Genero dos unidades organizativas (OU): *usuarios* y *grupos* e introduzco cinco usuarios dentro de la primera OU. Todo lo realizo a través de ficheros ldif. La figura 4.11 muestra la definición de las OU's y la incorporación de un usuario.

```
dn: ou=usuarios,dc=mistic,dc=test
objectClass: organizationalUnit
ou: usuarios

dn: ou=grupos,dc=mistic,dc=test
objectClass: organizationalUnit
ou: grupos

dn: uid=jefe,ou=usuarios,dc=mistic,dc=test
objectClass: person
objectClass: inetOrgPerson
cn: jefe
businessCategory: responsable
departmentNumber: ventas
displayName: Jefe Jefazo
givenName: Jefe
sn: jefazo
mail: jefe@mistic.test
uid: jefe
userPassword:: bWlzdGlj
```

Figura 4.11: Contenido del fichero ldif utilizado.

Para incorporar estos datos al directorio ejecuto el comando:

```
#ldapadd -x -W -D "cn=admin,dc=mistic,dc=test" -f carga.ldif
```

ldapadd es un comando proporcionado por el paquete ldap-utils. Otros comandos son:

- *ldapmodify* -Modifica una entrada en el directorio
- *ldapadd* - Añade una nueva entrada.
- *ldapdelete* - Borra una entrada.
- *ldapmodrdn* - Renombra una entrada.
- *ldappasswd* - Cambia la contraseña.

4.7.2. ApacheDS

El paquete *deb* es inválido para la versión del SO existente en *ldap.mistic.test*
Procedo a la instalación manual ejecutando:

```
# ./apacheds-2.0.0-M24-64bit.bin
```

A través del cliente Apache Directory Studio (mediante exportación/importación de ficheros ldif) replico los datos del dominio *mistic.test* del servidor OpenLDAP al servidor ApacheDS.

-

4.7.3. Apache Directory Studio

La herramienta que facilita mucho la labor de administrar un directorio LDAP es *Apache Directory Studio*. Gestiona varios implementaciones: OpenLDAP, Active Directory de Microsoft o ApacheDS.

En el caso de que gestionemos ApacheDS podemos configurar el servidor desde este cliente. Así ha sido como he configurado el acceso seguro en puerto 10636. He indicado la ruta donde se encuentra el keystore (Importante que el formato sea jks) en la máquina *ldap.mistic.test.*

El aspecto de ApacheDS es el que apreciamos en la figura 4.12. Se ve tres conexiones: Mistic (OpenLDAP) y dos conexiones a ApacheDS.

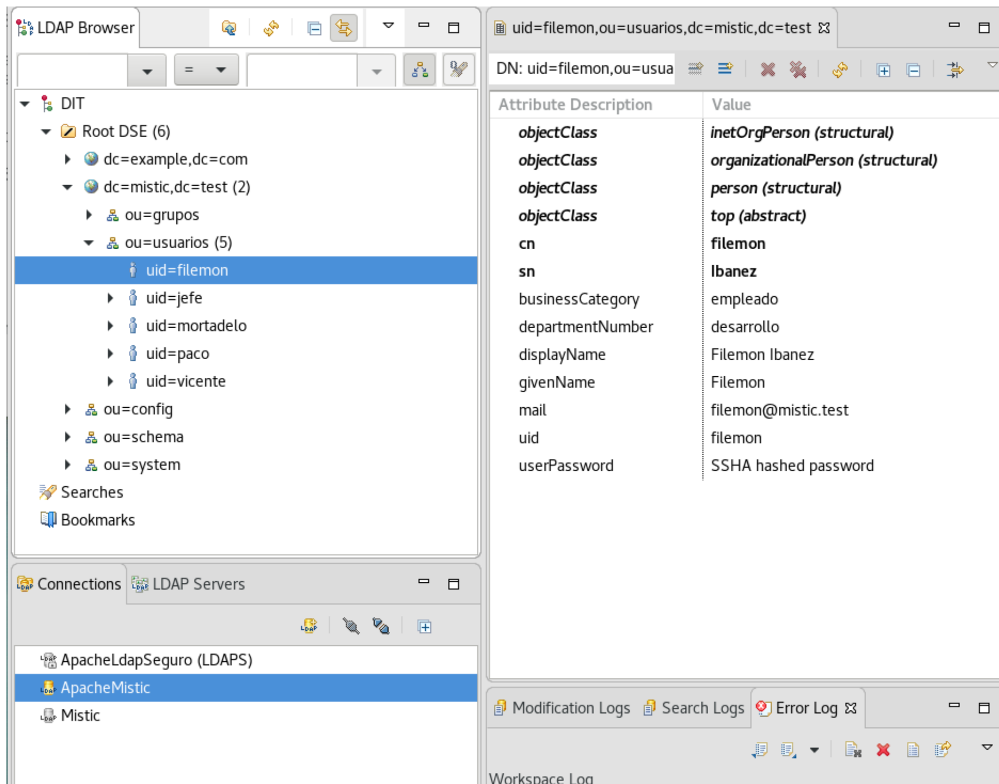

Figura 4.12: Apache Directory Studio

4.8. Servidores CAS.

Disponemos de dos servidores CAS para proporcionar alta disponibilidad: *cas1.mistic.test* y *cas2.mistic.test*

CAS es una aplicación web que desplegaré en servidores *Apache Tomcat 8.5.27*. Los configuro de la misma forma que los servidores de aplicación.

En el inicio de este trabajo configuré la versión 4.2.1. La configuración de esta versión se realiza a través de ficheros xml y es a veces poco clara. Finalmente he utilizado la versión 5.2.4 y definitivamente ¡ES MUCHO MEJOR!.

La configuración de las versiones 5.x.x se realiza a través de ficheros *properties*. Estos ficheros, se ubican en la ruta /etc/cas, sin embargo si las propiedades añadidas no son muchas, es adecuado introducirlas en el fichero *application.properties* que está en *WEB-INF/classes*.

Para la generación del ejecutable *cas.war* descargo *cas-overlay-template-master.zip*. Tras la descomprimir, se tiene un proyecto maven. En el fichero *pom.xm*l se añadirán todas las dependencias que precisemos. Después ejecutaremos:

```
#mvn clean

#mvn compile

#mvn install
```

Encontraremos el fichero *cas.war* en el directorio *target*. Para su despliegue se sitúa en el directorio *webapps* de *tomcat*

4.8.1. Configuración de CAS para autenticar contra LDAP.

Utilizo el servicio *ApacheDS* proporcionado por *ldap.mistic.test*

Para que el servidor CAS soporte LDAP introducimos en *pom.xml* la siguiente dependencia:

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

En la pagina de Apareo se encuentra una ingente documentación sobre CAS

En el fichero *application.properties* añado las siguientes entradas:

```
cas.authn.ldap[0].type=DIRECT
cas.authn.ldap[0].ldapUrl=ldap://ldap.mistic.test:10389
cas.authn.ldap[0].useSsl=false
cas.authn.ldap[0].useStartTls=false
cas.authn.ldap[0].connectTimeout=5000
cas.authn.ldap[0].subtreeSearch=true
cas.authn.ldap[0].baseDn=ou=usuarios,dc=mistic,dc=test
cas.authn.ldap[0].userFilter=uid={user}
cas.authn.ldap[0].bindDn=uid=jefe,ou=usuarios,dc=mistic, dc=test
cas.authn.ldap[0].bindCredential=mistic
cas.authn.ldap[0].enhanceWithEntryResolver=true
cas.authn.ldap[0].dnFormat=uid=%s,ou=usuarios,dc=mistic,dc=test
cas.authn.ldap[0].principalAttributeId=uid
cas.authn.ldap[0].principalAttributePassword=password
```

Las credenciales utilizadas para realizar la consulta son las del usuario jefe (dn: uid=jefe,ou=usuarios,d dc=mistic) de esta forma funciona tanto en LDAP como en ApacheDS (cambiando los puertos de conexión evidentemente).

Lo pruebo y autentica correctamente a los miembros de la ou=usuarios de mistic.test. (figura 4.13)

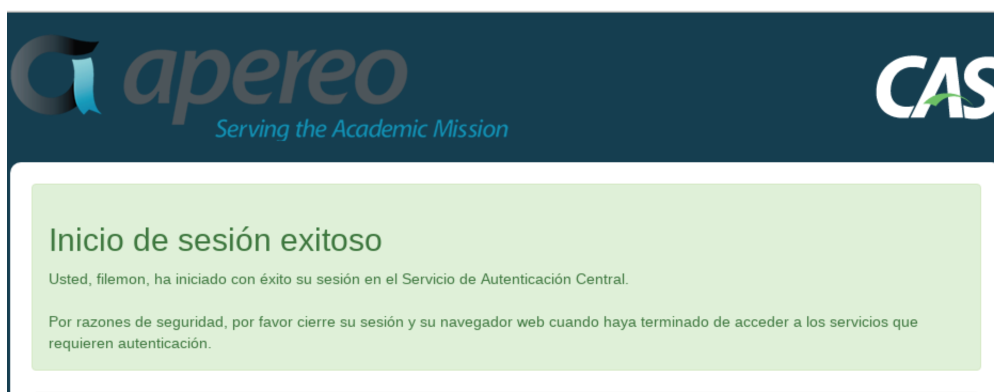

Figura 4.13: Autenticación exitosa en CAS/OpenLDAP

La comunicación por SSL es inmediata indicando el almacén de las claves del servidor, su contraseña, el certificado ca-mistic.test , el puerto de acceso 10636 y indicando SSL=true.

4.8.2. Alta disponibilidad en CAS.

La configuración recomendada por *Apereo* para aportar alta disponibilidad viene dada por la figura 4.14

Vemos que los servidores CAS se encuentran balanceados utilizando los servidores *authn-1, authn-1*,...,authn-n como fuentes de credenciales.

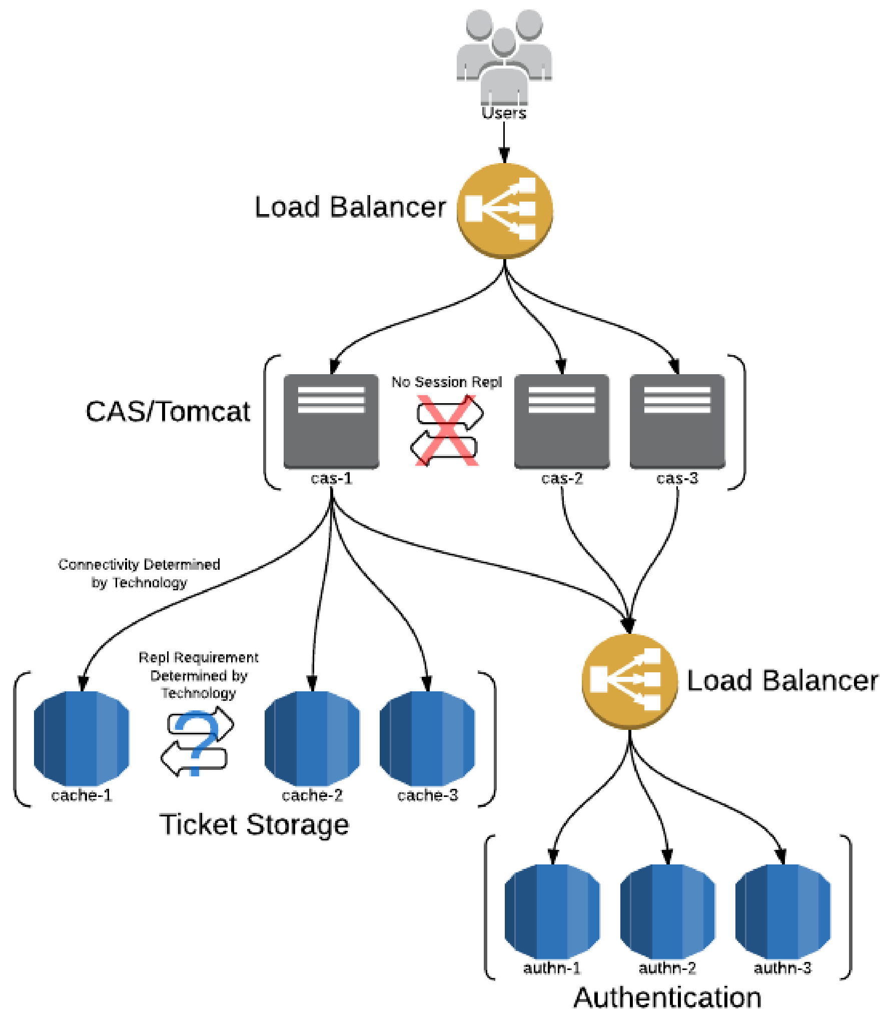

Figura 4.14: Configuración Alta Disponibilidad en CAS.

Es necesario que todos los servidores CAS compartan los tickets generados para que cualquier servidor reconozca como propios los tickets emitidos por otro. El mecanismo para ello es la utilización de *Tickets Storage*.

Básicamente existen dos formas de implementar la alta disponibilidad en CAS:

- Un solo nodo en un entorno virtualizado. Si falla la máquina virtual es transferida a otro host físico. Es la forma más simple y más utilizada en entornos empresariales.
- Varios nodos. Máquinas diferentes conectadas a un balanceador.

En la arquitectura de varios nodos podemos considerar que existen dos modos de funcionamiento:

- ***Activo/Pasivo.*** Solo uno de los nodos recibe una petición.Se utiliza un balanceo ordinario donde solo un nodo CAS proporciona un ticket. Si se da un fallo lo único que obliga es a volver a iniciar sesión.

- ***Activo/Activo.*** Todos los nodos reciben la petición pero sólo uno responde. Es necesario utilizar un almacén de tickets para que los nodos puedan compartirlos.

Tickets Storage.

Hay dos componentes configurables:
TicketRegistry : proporciona almacenamiento de tickets duradero.
ExpirationPolicy : proporciona un marco de políticas de caducidad de tickets.
Existe varios tipos de Almacenamientos de Tickets:

- Predeterminado: Utiliza un mapa interno en memoria para almacenamiento y recuperación de tickets. Este componente no conserva el estado del ticket en los reinicios y no es una solución adecuada para los entornos de clúster CAS que se implementan en el modo activo / activo.
- Basados en caché. Proporcionan una solución de alto rendimiento para el almacenamiento de tickets en implementaciones de alta disponibilidad. Los siguientes componentes están soportados:
    - Default
    - Hazelcast
    - Ehcache
    - Ignite
    - Memcached
    - Infinispan
- Basados en BBDD. Los tickets son almacenados en bases de datos SQL y se accede a ellos utilizando la capa JPA, Java Persistence API, de java.
- Basados en BBDD NoSQL. Los siguientes sistemas de BBDD NoSQL están soportados:
    - Infinispan
    - Couchbase
    - Redis
    - MongoDb
    - DynamoDb
- Replicación segura de caché. Varios registros de tickets basados en caché admiten la replicación segura a través de la red. Por seguridad éstos viajan cifrados.

La solución escogida para almacenar los tickets es *Hazelcast* que es una implementación basada en la librería de grid Hazelcast.

Para configurar Hazelcast volvemos a compilar añadiendo en el *pom.xml* la dependencia:

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

La configuración es sencilla añadiendo las siguientes entradas al fichero *application.properties*:

```
cas.ticket.registry.hazelcast.cluster.evictionPolicy=LRU
cas.ticket.registry.hazelcast.cluster.maxNoHeartbeatSeconds=300
cas.ticket.registry.hazelcast.cluster.multicastEnabled=false
cas.ticket.registry.hazelcast.cluster.tcpipEnabled=true
cas.ticket.registry.hazelcast.cluster.members=localhost
cas.ticket.registry.hazelcast.cluster.loggingType=slf4j
cas.ticket.registry.hazelcast.cluster.instanceName=cas1
cas.ticket.registry.hazelcast.cluster.port=5701
cas.ticket.registry.hazelcast.cluster.portAutoIncrement=true
cas.ticket.registry.hazelcast.cluster.maxHeapSizePercentage=85
cas.ticket.registry.hazelcast.cluster.backupCount=1
cas.ticket.registry.hazelcast.cluster.asyncBackupCount=0
cas.ticket.registry.hazelcast.cluster.members=cas1.mistic.test,cas2.mistic.test
```

Para evitar el error de "servicio no autorizado" tenemos que asegurarnos que está presente, en el fichero properties, la entrada:

`cas.serviceRegistry.initFromJson=true`

Además tiene que existir el fichero *WEB-INF/classes/services/HTTPSandIMAPS-10000001.json* conteniendo

```
{   "@class" : "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "^(https|imaps)://.*",
  "name" : "HTTPS and IMAPS",
  "id" : 10000001,
  "description" : "This service definition authorizes all application
      urls that support HTTPS and IMAPS protocols.",
  "evaluationOrder" : 10000 }
```

Que autoriza a las aplicaciones a acceder a CAS mediante https o imap.

Balanceado de carga

HAProxy es una buena solución pero lo tenemos que configurarlo como *Pass-Through* dado que el servidor CAS necesita ver que la conexión es segura. El cliente verá el certificado del servidor CAS que no corresponderá con el nombre del *Front-End*, avisando de sitio inseguro. La solución es configurar el Tomcat donde esté el servidor CAS con el certificado del *Front-End.*

La configuración del balanceador que recibe conexiones en el puerto 443 y reenvía al puerto 8443 de los nodos es la señalada en la en la figura4.15.

```
frontend front_cas
   bind *:443
   stats uri /hastats
   #option tcplog
   mode tcp
  default_backend back_cas

backend back_cas
   mode tcp
   option ssl-hello-chk
   cookie SERVERID insert indirect nocache
   server  cas1 192.168.1.31:8443 check
   server  cas1 192.168.1.34:8443 check
```

Figura 4.15: Configuración Pass-Through de HAProxy

Pruebo el acceso a cas desde el balanceador con la url: https://cas.mistic.test/cas y funciona correctamente.

4.9. La aplicación prototipo.

La aplicación prototipo tiene como fin comprobar que se puede autenticar desde CAS. Cualquier aplicación se configuraría de la misma forma.

Para “caseificar” la aplicación tenemos que incorporar a ésta la librería del cliente CAS. Como se trata de un proyecto *maven* lo realizamos añadiendo a la aplicación la dependencia:

```
<dependency>
    <groupId>org.jasig.cas.client</groupId>
    <artifactId>cas-client-core</artifactId>
    <version>3.4.1</version>
</dependency>
```

Lo siguiente será introducir las modificaciones en el fichero *web.xml*. La modificación que presento permite comprobar que los dos servidores CAS están compartiendo tickets. Para la solicitar la autenticación se utiliza *cas2.mistic.test* y para la validación *cas1.mistic.test*. En este caso se están utilizando los certificados de servidor en lugar del certificado del balanceador. Tras la prueba sustituyo la llamadas a los servidores cas1.mistic.test y cas2.mistic.test por llamadas al balanceador cas.mistic.test contra el puerto 443.

El contenido de *web.xml* se puede ver en el anexo B de este documento.

Tras introducir la url: https://app.mistic.test/mistic

Aparece la página de autenticación (figura 4.9)de CAS. El resultado tras pasar este proceso se ve en la figura 4.9 .

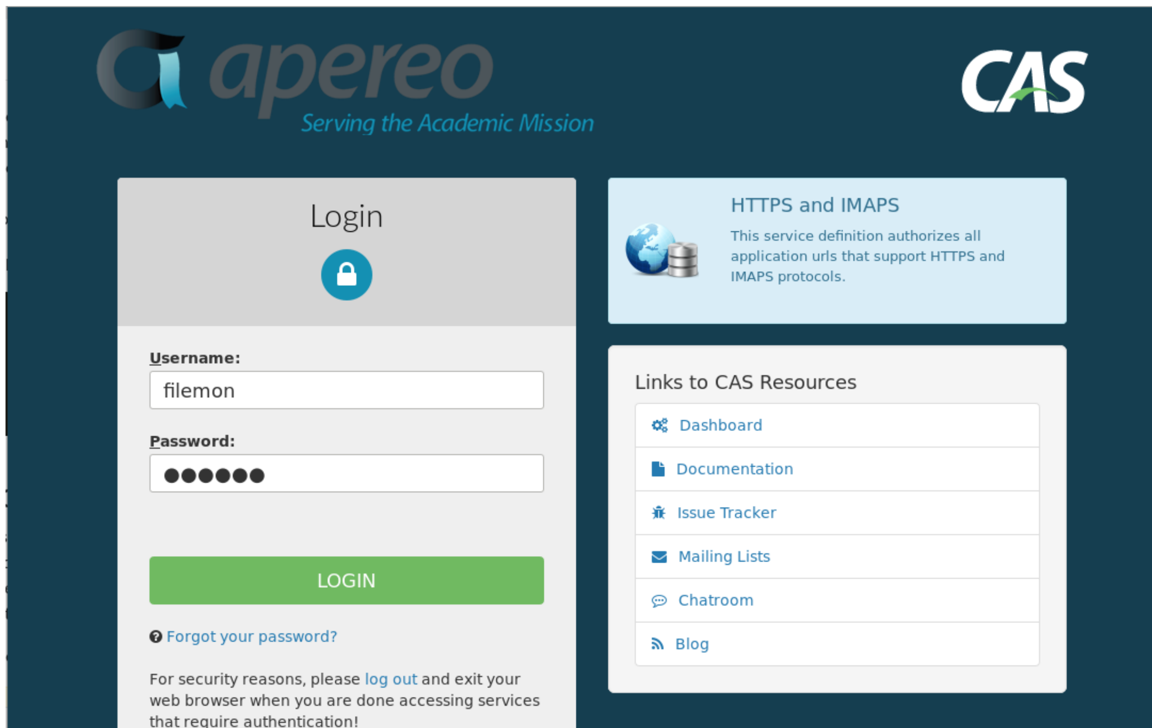

Figura 4.16: Solicitud autenticación de la aplicación *mistic*.

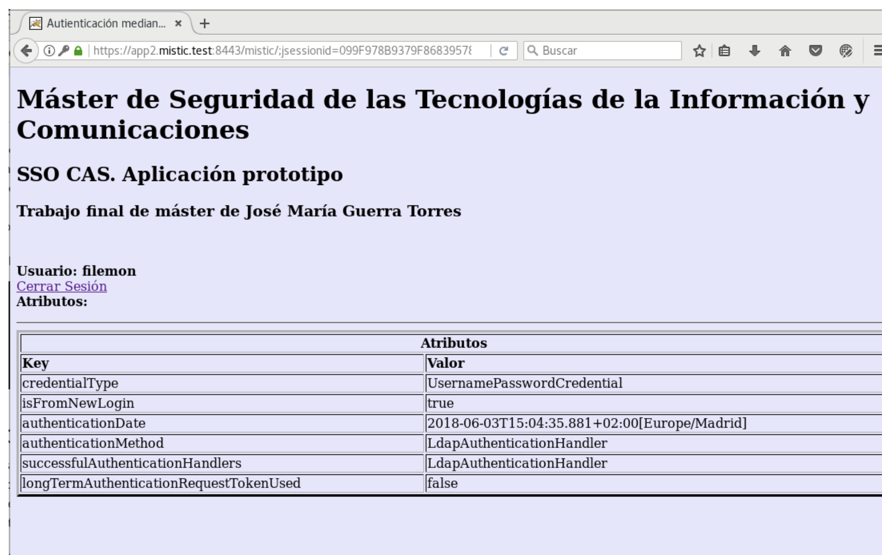

Figura 4.17: Aplicación prototipo tras autenticar.

4.10. Instalación y configuración de *Reverse Proxy*.

Un proxy inverso es un tipo de proxy que ejerce de intermediario entre un cliente y los servidores. El cliente verá al *reverse proxy* como el servidor real.

Un proxy inverso en nuestro trabajo actúa para interponerse entre internet y clúster de aplicaciones y lo situaremos en la DMZ.

Configuraremos el cortafuegos de tal forma que solo permita el acceso al servidor de aplicaciones.

La implementación elegida es *nginx* que es presentado como un reverse proxy eficiente.

Lo instalo en el servidor *front.mistic.test y* añado el fichero de configuración *app.conf* en el directorio */etc/nginx/conf.d* con el contenido:

```
server {
  listen 443 ssl default_server;
  listen [::]:443
  ssl default_server;
  server_name aplicaciones.mistic.test;
  ssl on;
  ssl_certificate /etc/nginx/ssl/aplicaciones.pem;
  ssl_certificate_key /etc/nginx/ssl/aplicaciones.key;
  ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
  ssl_ciphers "HIGH:!aNULL:!MD5 or HIGH:!aNULL:!MD5:!3DES";
ssl_prefer_server_ciphers on;
  location /{
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_pass https://app.mistic.test/;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_ssl_certificate /etc/nginx/ssl/app.pem;
    proxy_ssl_certificate_key /etc/nginx/ssl/app.key;
  }
}
```

Esto hace que se cree el servidor virtual *aplicaciones.mistic.test* que apunta al clúster de aplicaciones. Es necesario haber generado el conjunto de certificados para este servidor.

El nombre de este servidor virtual habrá que resolverlo. Para ello introducimos la entrada *aplicaciones.mistic.test* en cada fichero */etc/hosts* apuntando a la misma dirección IP que *front.mistic.test.*

Capítulo 5

Bibliografía.

SSO. Single Sign-On.

Palazón et al. (2017) *Single sign-on y federación de identidades.*Barcelona:UOC.

del Campo, Ruth.(2011)*Acceso remoto a sistemas corporativos: gestión de identidades, single sign-on y teletrabajo.* Madrid:Astic.

https://www.openssl.org/docs/[Consulta: marzo-mayo 2018]

https://en.wikipedia.org/wiki/Single_sign-on[Consulta: marzo-mayo 2018]

https://esn.wikipedia.org/wiki/Single_sign-on[Consulta: marzo-mayo 2018]

https://www.chakray.com/que-es-el-single-sign-on-sso-definicion-caracteristicas-y-ventajas/[Con marzo-mayo 2018]

https://www.openssl.org/docs/[Consulta: marzo-mayo 2018]

https://apereo.github.io/cas/4.2.x/index.html[Consulta: marzo-mayo 2018]

https://apereo.github.io/cas/5.2.x/index.html[Consulta: marzo-mayo 2018]

https://www.adictosaltrabajo.com/tutoriales/implementando-ssocas/[Consulta: marzo-mayo 2018]

https://github.com/apereo/cas[Consulta: mayo 2018]

LDAP.

https://www.openldap.org/[Consulta: abril-mayo del 2018]

http://directory.apache.org/apacheds/[Consulta: abril-mayo del 2018]

http://directory.apache.org/studio/[Consulta: abril-mayo del 2018]

Proxy's y High Availability.

https://nginx.org/en/docs/ [Consulta: Abril-Mayo del 2018]

http://www.haproxy.org/[Consulta: Abril-Mayo 2018]

https://httpd.apache.org/docs/trunk/es/howto/reverse_proxy.html[Consulta: Abril-Mayo del 2018]

Certificados digitales.

https://www.openssl.org/docs/[Consulta: marzo-mayo 2018]

Apéndice A

Guía para la gestión de certificados digitales.

Creacíon de una CA (Autoridad Certificadora):

La llamamos ca-mistic

```
# openssl req -nodes -x509 -newkey rsa:2048 -days 3650 -keyout
ca-mistic.key -out ca-mistic.crt
```

Creación de certificados de servidor.

Importante: Tiene que expedirse a nombre del servidor.
Generamos la clave privada:

```
# openssl genrsa -out <nombre_servidor>.key 2048
```

Solicitamos firma.

```
# openssl req -new -key <nombre_servidor>.key -out <nombre_servidor>.csr
```

Firma por parte de la autoridad certificadora creada.

```
# openssl x509 -CA ca-mistic.crt -CAkey ca-mistic.key -req
-in <nombre_servidor>.csr -days 3650 -CAcreateserial -sha256
-out <nombre_servidor>.crt
```

Creación de almacén en formato PKCS12 (*.p12)

En este caso utilizo los alias ”tomcat” y ”root” para el certificado del servidor y el certificado de la CA respectivamente.

```
# openssl pkcs12 -export -in <nombre_certificado>.crt

-inkey <nombre_certificado>.key -out <nombre_certificado>.p12

-name tomcat -CAfile <nombre_ca>.crt -caname root -chain
```

Convertir almacenes de claves: de p12 a jks

```
# keytool -v -importkeystore -srckeystore <nombre_almacen>.p12
-srcstoretype PKCS12 -destkeystore <nombre_almacen>.jks
-deststoretype JKS
```

Almacenar certificados en fichero pem

La orden siguiente almacena la clave pública y privada del servidor app.mistic.test y la clave pública de la autoridad de certificación ca-mistic.test en un fichero de formato pem mostrándolo en pantalla (tee).

```
# cat app.crt app.key ca-mistic.crt| tee app.pem
```

Mostrar los datos de un certificado

```
# openssl req -in <nombre>.csr -noout -text
```

Instalación del certificado CA en Debian.

La clave pública del certificado de nuestra CA esta contenida en *ca-mistic.crt*

Llevamos la clave pública del certificado generado *ca-mistic.crt* al directorio */usr/share/ca-certificates/ca-mistic*

```
# mkdir /usr/share/ca-certificates/ca-mistic
# cp ca-mistic.crt  /usr/share/ca-certificates/ca-mistic/
```

Editamos el fichero */etc/ca-certificates.conf* añadiendo la entrada *ca-mistic/ca-mistic.crt*

Finalmente los incorporamos al sistema:

```
# dpkg-reconfigure ca-certificates
```

Apéndice B

Modificación del web.xml para CASeizar.

Estas son las modificaciones en el fichero web.xml de la aplicación prototipo para habilitar la utilización de CAS. Además, verifica que los dos nodos están compartiendo tickets utilizando el servidor *cas2.mistic.test* para la autenticación y el servidor *cas1.mistic.test* para la validación.

```
<!--CONFIGURACION  PARA CASSIFICAR LA APLICACIÓN PROTOTIPO-->

 <filter>
        <filter-name>CAS Single Sign Out Filter</filter-name>
        <filter-class>org.jasig.cas.client.session.SingleSignOutFilter</filter-
class>
        <init-param>
            <param-name>casServerUrlPrefix</param-name>
            <param-value>https://cas1.mistic.test:8443/cas</param-value>
        </init-param>
    </filter>

    <listener>
        <listener-
class>org.jasig.cas.client.session.SingleSignOutHttpSessionListener</listener-
class>
    </listener>

    <filter>
        <filter-name>CAS Authentication Filter</filter-name>

        <filter-class>org.jasig.cas.client.authentication.AuthenticationFilter</
filter-class>
        <init-param>
            <param-name>casServerLoginUrl</param-name>
            <param-value>https://cas2.mistic.test:8443/cas/login</param-value>
        </init-param>
        <init-param>
            <param-name>serverName</param-name>
            <param-value>https://app2.mistic.test:8443</param-value>
        </init-param>
    </filter>

    <filter>
        <filter-name>CAS Validation Filter</filter-name>

        <filter-
class>org.jasig.cas.client.validation.Cas30ProxyReceivingTicketValidationFilter</
filter-class>
        <init-param>
            <param-name>casServerUrlPrefix</param-name>
            <param-value>https://cas1.mistic.test:8443/cas</param-value>
        </init-param>
        <init-param>
            <param-name>serverName</param-name>
            <param-value>https://app2.mistic.test:8443</param-value>
        </init-param>
        <init-param>
            <param-name>redirectAfterValidation</param-name>
            <param-value>true</param-value>
        </init-param>
        <init-param>
            <param-name>useSession</param-name>
            <param-value>true</param-value>
        </init-param>

        <init-param>
            <param-name>authn_method</param-name>
            <param-value>mfa-duo</param-value>
        </init-param>
    </filter>

    <filter>
        <filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>
        <filter-class>org.jasig.cas.client.util.HttpServletRequestWrapperFilter</
filter-class>
    </filter>
```

```xml
    <filter-mapping>
        <filter-name>CAS Single Sign Out Filter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter-mapping>
        <filter-name>CAS Validation Filter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter-mapping>
        <filter-name>CAS Authentication Filter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter-mapping>
        <filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <welcome-file-list>
        <welcome-file>
            index.jsp
        </welcome-file>
    </welcome-file-list>
</web-app>
```