**Kit de integración de firma con FIRe**

Manual de instalación y despliegue

Versión: 2.4

|  |  |  |  |
| --- | --- | --- | --- |
| **CONTROL DE VERSIONES** | | | |
| Título | | Manual de instalación y despliegue | |
| Autor | | Secretaría General de Administración Digital  Ministerio de Política Territorial y Función Pública | |
| Fecha versión 2.4 | | 14 de mayo de 2021 | |
|  | | | |
| Versión | Fecha | Responsable | Cambios introducidos |
| 1.0 | 11-01-2016 | DTIC | Creación del documento |
| 1.0 Rev 1 | 16-02-2016 | DTIC | Actualización de las propiedades de configuración |
| 1.0 Rev 2 | 17-02-2016 | DTIC | Adaptación al cambio de nombre a Cl@ve Firma |
| 1.0 Rev 3 | 24-02-2016 | DTIC | Actualización de las sentencias de bases de datos |
| 1.0 Rev 4 | 18-03-2016 | DTIC | Cambio de contraseña del certificado de prueba |
| 1.0 Rev 5 | 18-04-2016 | DTIC | Configuración del directorio de configuración y correcciones varias |
| 1.0 Rev 6 | 28-04-2016 | DTIC | Se renombra el conector al backend de custodia de claves |
| 1.0 Rev 7 | 25-05-2016 | DTIC | Se agrega la propiedad de configuración del ProviderName del servicio. |
| 1.0 Rev 8 | 30-05-2016 | DTIC | Actualización de los componentes distribuidos .Net y PHP. |
| 1.0 Rev 9 | 13-06-2016 | DTIC | Actualización del componente distribuido PHP. |
| 1.0 Rev 10 | 19-01-2017 | SGAD | Aclaración sobre la carga de la configuración. |
| 1.1 | 07-02-2017 | SGAD | Aclaración sobre el componente distribuido PHP. |
| 1.1 Rev 1 | 23-03-2017 | SGAD | Se actualizan las claves de los certificados de prueba y se agregan los problemas comunes. |
| 2.0 | 12-05-2017 | SGAD | Se actualiza la información a la de FIRe. |
| 2.0 Rev 1 | 10-07-2017 | SGAD | Aclaraciones en el apartado de despliegue. |
| 2.1 | 10-10-2017 | SGAD | Se documentan los requisitos para el uso de un gestor de documentos de FIRe y la configuración de la gestión compartida de sesiones entre varios servidores balanceados. |
| 2.1.1 | 28-11-2017 | SGAD | Se documenta la propiedad para la configuración del contexto público del componente central. |
| 2.2 | 16-05-2018 | SGAD | Documentación de los cambios de la nueva versión y revisión general. |
| 2.3 | 11-10-2018 | SGAD | Cifrado de contraseñas, información de migración y consideraciones adicionales. |
| 2.4 | 14-05-2021 | SGAD | Cambios en la gestión de logs, generación de estadísticas y soporte de firmas con periodo de gracia.  Actualización del proceso de despliegue para el componente de Administración. |
| 2.5 | 31-05-2021 |  | Actualización del proceso de despliegue para el componente de Administración. |

ÍNDICE

[1 Objeto del documento 7](#_Toc72148320)

[2 Introducción 8](#_Toc72148321)

[3 Arquitectura Software 9](#_Toc72148322)

[4 Componente Central 12](#_Toc72148323)

[4.1 Requisitos Software 12](#_Toc72148324)

[4.1.1 Requisitos del usuario 12](#_Toc72148325)

[4.2 Compatibilidad con dispositivos móviles 13](#_Toc72148326)

[4.3 Despliegue 14](#_Toc72148327)

[4.4 Configuración del componente 15](#_Toc72148328)

[4.4.1 Fichero config.properties 16](#_Toc72148329)

[4.4.2 Fichero platform.properties 27](#_Toc72148330)

[4.4.3 Fichero alarms\_config.properties 31](#_Toc72148331)

[4.4.4 Fichero provider\_clavefirma.properties 32](#_Toc72148332)

[4.4.5 Fichero provider\_clavefirmatest.properties 33](#_Toc72148333)

[4.4.6 Fichero provider\_fnmt.properties 35](#_Toc72148334)

[4.4.7 Fichero de apariencia de conector 37](#_Toc72148335)

[4.5 Dependencias Externas 41](#_Toc72148336)

[4.6 Base de datos 42](#_Toc72148337)

[4.6.1 Creación de la Base de datos 42](#_Toc72148338)

[4.6.2 Modelo de datos 42](#_Toc72148339)

[4.7 Clases gestoras de documentos 47](#_Toc72148340)

[4.7.1 Desarrollo de una clase gestora de documentos 48](#_Toc72148341)

[4.7.2 Alta de la clase gestora 49](#_Toc72148342)

[4.8 Clases de notificación de errores críticos 50](#_Toc72148343)

[4.8.1 Desarrollo de una clase de notificación 51](#_Toc72148344)

[4.8.2 Alta de la clase de notificación 52](#_Toc72148345)

[5 Componente Distribuido 54](#_Toc72148346)

[5.1 Java 54](#_Toc72148347)

[5.1.1 Trazas de registro del Componente Distribuido Java 56](#_Toc72148348)

[5.2 .Net 56](#_Toc72148349)

[5.3 PHP 57](#_Toc72148350)

[6 Componente de Administración 60](#_Toc72148351)

[6.1 Despliegue 60](#_Toc72148352)

[6.2 Configuración del componente 61](#_Toc72148353)

[6.2.1 Configuración de la conexión a base de datos 61](#_Toc72148354)

[6.2.2 Configuración del servidor de correo 62](#_Toc72148355)

[6.2.3 Ficheros de ejemplo 63](#_Toc72148356)

[7 Simulador de Cl@ve Firma 65](#_Toc72148357)

[7.1 Despliegue 65](#_Toc72148358)

[7.2 Configuración del componente 65](#_Toc72148359)

[7.3 Uso del servicio de simulación de Cl@ve Firma 65](#_Toc72148360)

[7.4 Usuarios de prueba 66](#_Toc72148361)

[7.4.1 Agregar nuevos usuarios de prueba 67](#_Toc72148362)

[8 Componentes adicionales 69](#_Toc72148363)

[8.1 Servicio auxiliar del conector de la FNMT 69](#_Toc72148364)

[8.2 Servicio para la consulta de logs desde el módulo de administración 69](#_Toc72148365)

[9 Configuración de los componentes de FIRe 71](#_Toc72148366)

[9.1 Uso de variables de entorno 71](#_Toc72148367)

[9.2 Cifrado de propiedades 72](#_Toc72148368)

[10 Nuevos proveedores de firma 74](#_Toc72148369)

[11 Despliegue en entornos balanceados 76](#_Toc72148370)

[ANEXO I. Migración a FIRe 2.4 79](#_Toc72148371)

[I.1. Migración desde FIRe 2.1 / 2.1.1 79](#_Toc72148372)

[I.1.1. Migración de la base de datos 79](#_Toc72148373)

[I.1.2. Migración de la configuración del componente central 80](#_Toc72148374)

[I.1.3. Migración de las aplicaciones 82](#_Toc72148375)

[I.2. Migración desde FIRe 2.2 83](#_Toc72148376)

[I.2.1. Migración de la base de datos 83](#_Toc72148377)

[I.2.2. Migración de la configuración del componente central 83](#_Toc72148378)

[I.2.3. Migración de las aplicaciones 84](#_Toc72148379)

[I.3. Migración desde FIRe 2.3 / 2.3.1 85](#_Toc72148380)

[I.3.1. Migración de la base de datos 85](#_Toc72148381)

[I.3.2. Migración de la configuración del componente central 85](#_Toc72148382)

[I.3.3. Migración de las aplicaciones 85](#_Toc72148383)

[ANEXO II. Despliegue de demostración sobre Apache Tomcat 87](#_Toc72148384)

[ANEXO III. Problemas conocidos 89](#_Toc72148385)

[III.1. Error en las firmas XAdES en despliegues sobre JBoss 89](#_Toc72148386)

[ANEXO IV. Volcado de estadísticas a base de datos 90](#_Toc72148387)

# Objeto del documento

El presente manual detalla, principalmente, la arquitectura de FIRe, sus requisitos software y los pasos a seguir para su despliegue y configuración. También se describen distintos procedimientos para la integración y configuración de FIRe con sistemas y mecanismos distintos a los integrados por defecto: nuevos servicios de firma, gestores de documentos y plataformas de validación y mejora de firmas.

# Introducción

FIRe es un sistema para la generación de firmas electrónicas con certificado de persona física. FIRe funciona exclusivamente como pasarela de firma a la que el usuario accede desde la aplicación web que solicita firmar.

FIRe permite firmar con certificados residentes en el equipo local del usuario y con certificados de proveedores de firma en la nube, como Cl@ve Firma. Gracias a ello, se hace posible firmar con unos u otros certificados sin que la aplicación deba preocuparse del origen de los mismos.

Un integrador debe conectar su aplicación con FIRe cuando desee que un usuario firme electrónicamente como parte del flujo de trabajo de su aplicación. De esta forma, un usuario podrá acceder a su aplicación, realizar el trámite oportuno y, en el momento de firmar, será redirigido automáticamente a FIRe, donde firmará los datos proporcionados por la aplicación. Una vez firmados los datos, FIRe redirigirá al usuario automáticamente de vuelta a la aplicación, que podrá mostrar el resultado al usuario y solicitar a FIRe las firmas recién realizadas.

FIRe se compone principalmente de un API cliente (Componente distribuido) y el servicio de firma (Componente Central):

* El API cliente permite hacer las llamadas al servicio de firma para la firma de uno o varios documentos (firma de lote) y posteriormente recuperar el resultado de estas operaciones. Se distribuyen tres implementaciones diferentes de este API (Java, .NET y PHP)
* El servicio de firma es el que incluye la lógica de firma y los recursos necesarios para funcionar como pasarela de firma. No existe un despliegue de este servicio al que se puedan conectar todas las aplicaciones. Cada entidad debe desplegar su propia instancia del componente central de FIRe, a la que podrán conectarse cada una de sus aplicaciones.

Las aplicaciones deberán estar dada de alta en el componente central de FIRe para hacer uso del mismo, frente al que se autenticarán mediante un certificado y un identificador de aplicación.

Adicionalmente, FIRe se distribuye con otro conjunto de componentes que pueden ser necesarios según el uso que se haga de FIRe o el entorno en el que se despliegue:

* Módulo de administración: Permite gestionar el alta de aplicaciones, sus certificados y responsables, visualizar estadísticas y facilitar la consulta de logs. No es obligatorio el despliegue de este componente si el sistema sólo va a permitir el uso de una aplicación y no se quiere hacer uso del resto de sus funcionalidades. El despliegue de este módulo hace necesario el uso de base de datos.
* Simulador de Cl@ve Firma: Es un servicio que emula el aspecto y funcionalidad de Cl@ve Firma. Puede desplegarse en entornos no productivos para probar la integración de las aplicaciones antes de conectar el componente central a los proveedores reales.
* Componentes adicionales: Algunos proveedores de firma en la nube pueden proporcionar servicios propios para el funcionamiento de su conector. Por ejemplo, FIRe se distribuye con un servicio de la FNMT necesario poder utilizar su proveedor. También se proporciona un servicio para facilitar la consulta de logs del componente central desde la interfaz de administración de FIRe.

# Arquitectura Software

El sistema de firma FIRe cuenta con un servicio (componente central) encargado de atender las distintas solicitudes de firma. Las aplicaciones cliente que deseen firmar harán uso del API (componente distribuido) de FIRe para solicitar las firmas a dicho servicio.

Cada una de estas aplicaciones cliente deberá autenticarse ante el componente central mediante certificado e identificador de aplicación. El componente central almacena la información de autenticación de cada una de las aplicaciones cliente y sólo admitirá las peticiones si la aplicación está correctamente autorizada. Existe un componente de administración orientado a gestionar las credenciales de las aplicaciones, aunque también permite realizar otras tareas como la consulta de logs o la consulta de estadísticas de uso.

En el caso excepcional en el que sólo se fuese a dar servicio a una única aplicación cliente, sería posible evitar el despliegue de la herramienta de administración lo que podría hacer innecesario el uso de base de datos. Para esto se incluiría la información de autenticación en uno de los ficheros de configuración del componente central. Sin el módulo de administración, sin embargo, no es posible disfrutar del resto de ventajas que este ofrece (consulta de logs del componente central y acceso a estadísticas).

La comunicación entre el componente distribuido y el componente central siempre se debe realizar sobre una conexión SSL con autenticación cliente. El certificado SSL cliente que utilice la aplicación es el que usará el componente central para autenticarla. Para permitir este modelo de autenticación, el servidor de aplicaciones no debe restringir las peticiones según el certificado cliente utilizado. Será el propio componente central el que lo haga.

Existen implementaciones del componente distribuido en lenguaje Java, .NET y PHP, de tal forma que un integrador podrá elegir el que mejor se ajuste a su aplicación.

FIRe permite a los usuarios firmar utilizando un certificado en un almacén local del usuario o mediante los certificados en la nube de diversos proveedores.

Las firmas mediante certificados locales se realizan mediante el Cliente @firma. Este permitirá al usuario utilizar los certificados del almacén interno de su navegador web y de dispositivos criptográficos externos (como el DNIe). En el caso de contar con PIN o contraseña el almacén de claves seleccionado, será el propio almacén (o el Cliente @firma cuando se delegue esta tarea) el que se encargue de pedírselo al usuario.

En el caso de las firmas con certificados en la nube, FIRe incorpora los conectores de varios proveedores, aunque el administrador del sistema es libre de incorporar otros nuevos. Los proveedores de firma en la nube que incorpora por defecto FIRe son:

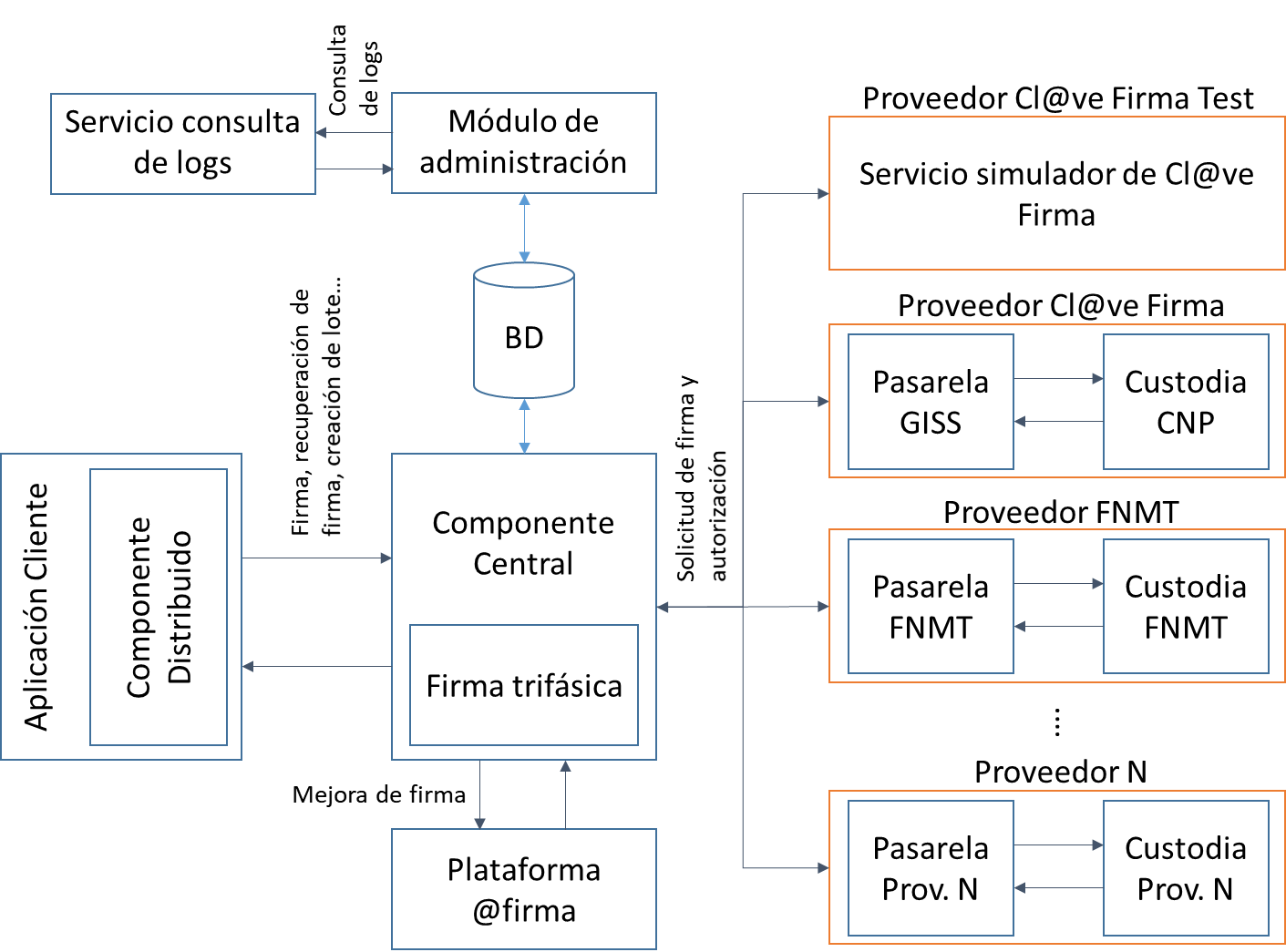
* Cl@ve Firma: Este proveedor permite a los ciudadanos firmar con su certificado personal de Cl@ve Firma. Estos certificados están custodiados por el Cuerpo Nacional de Policía (CNP) y se accede a los mismos a través de la pasarela proporcionada por la Gerencia Informática de la Seguridad Social (GISS). Para disponer de certificados de firma de Cl@ve Firma un ciudadano deberá haberse dado de alta en el servicio de Cl@ve Permanente.
* CERES Cloud ID: Este proveedor permite a los funcionarios públicos firmar con su certificado en la nube de la FNMT. En este caso, los certificados y sus claves están custodiados por la propia Fábrica Nacional de Moneda y Timbre. Para disponer de estos certificados, los funcionarios públicos deben haberse dado de alta en este servicio de la FNMT.

FIRe también incorpora un conector para hacer uso de un servicio de prueba distribuido con el propio FIRe. Este servicio emula el comportamiento e interfaz de Cl@ve Firma y sirve para facilitar las pruebas de integración de las aplicaciones con FIRe. Gracias a este servicio pueden completar el proceso de firma sin necesidad de conectarse con un proveedor real de firma en la nube.

Todas las firmas realizadas por FIRe se realizan en 3 fases, donde la primera fase (composición de los datos a firmar) y la tercera fase (composición de la firma electrónica) se realizarán en el componente central, mientras que la segunda fase (firma digital), que es la única que involucra a la clave privada del certificado, se realiza en el equipo del usuario (certificado local) o en los servidores de firma del proveedor (certificado en la nube). Todo este proceso es transparente para las aplicaciones que integran FIRe.

Si así lo indica la aplicación cliente, es posible que tras realizar una firma electrónica deba validarse esta o actualizarse a un formato longevo. Esto último es, incrustarle información adicional como, por ejemplo, un sello de tiempo o información de revocación. Estas operaciones posteriores a la firma se realizan por medio de un sistema externo de validación y mejora de firma. El único sistema para el que se integra FIRe por defecto es con la Plataforma @Firma, aunque se permite que se desarrollen y configuren conectores con otros sistemas. Ya se configure el uso de la plataforma @Firma o cualquier otra, el componente central deberá tener acceso a ella si se desea permitir que las aplicaciones cliente realicen validaciones y mejoras de firma.

En el siguiente diagrama se ilustra la arquitectura básica del sistema:



Los componentes software que se incluyen en el kit de integración y su correspondencia con los bloques del diagrama son los siguientes:

* Componente central. Es el módulo principal de FIRe y en el que reside toda la funcionalidad de firma:
  + fire-signature.war
* Componente distribuido. Son las librerías utilizadas para que las aplicaciones puedan hacer uso de las funcionalidades de FIRe:
  + Java: fire-client-2.4.jar
  + .Net: fire\_client.dll
  + PHP: fire\_client.php
* Módulo de Administración. Es el componente desde el que se gestionan las aplicaciones que tienen acceso a FIRe y otros aspectos para su gestión y aprovechamiento:
  + fire-admin-web.war
* Servicio simulador de Cl@ve Firma. Es un servicio que emula el comportamiento de Cl@ve Firma, pero sin conectarse con ninguna pasarela remota, con lo que permite realizar pruebas sin dar de alta nuestra aplicación en ningún servicio. Sólo se debería desplegar en entornos no productivos:
  + clavefirma-test-services.war
* Aplicación cliente. Es la aplicación que solicita firmar por medio del componente distribuido. En el kit de integración se distribuye la aplicación de ejemplo:
  + fire-test-jsp.war
* Componentes adicionales. Son servicios, no dependientes de FIRe, necesarios para el funcionamiento de algún proveedor de firma concreto o alguna funcionalidad opcional:
  + Servicio de consulta de logs. Servicio para la consulta y descarga de logs desde la herramienta de administración. Este servicio se puede desplegar en cada uno de los nodos en el que se encuentre el componente central de FIRe, para después consultar todos ellos desde el módulo de administración.
    - log-consumer-service.war
  + Servicio auxiliar FNMT. Servicio requerido por el conector de CERES Cloud ID para su correcto funcionamiento (no representado en el diagrama). Se debería desplegar junto al componente central.
    - fnmt-fire-service.war

# Componente Central

El componente central es el módulo principal de FIRe. Es el que aglomera toda la lógica de firma, la conexión con los distintos proveedores de firma en la nube y con la plataforma de mejora y validación de firma, entre otras cosas.

Cada organismo que desee integrar FIRe deberá desplegar el componente central en sus entornos. Después, en cada una de las aplicaciones que deseen realizar firmas de usuario con FIRe, se deberá utilizar el componente distribuido para conectar con este componente central.

A continuación, se detallan los diferentes aspectos que deben tenerse en cuenta para desplegar y configurar el componente central.

## Requisitos Software

* Java 1.8 o superior (Compatible con Oracle Java y OpenJDK).
* Servidor de aplicaciones. Puede optarse por una de las siguientes opciones:
  + Servidor de aplicaciones JEE versión 6 o superior (JBoss, WebSphere, GlassFish, etc.)
  + Servidor de *servlets* de Java versión 2.4 o superior (Apache Tomcat, etc.)
* Servidor web.
  + Permitiría no exponer directamente el servidor de aplicaciones a la web y facilitar la configuración de algunos de los requisitos descritos en el apartado Despliegue.
* Sistema de Gestión de Base de Datos compatible JDBC (MySQL, Oracle, etc.).
  + Esto es necesario sólo si deseamos desplegar el módulo de administración para dar servicio a más de una aplicación desde el componente central.
  + También será necesario si vamos a desplegar el sistema en más de un nodo en un entorno de alta disponibilidad y queremos que las sesiones de los nodos se compartan a través de base de datos.
  + Los controladores JDBC del sistema de gestión de base de datos elegido, deberán estar instalados en el *classpath* del servidor de aplicaciones. Por ejemplo, para Tomcat bastará con copiar el controlador (.jar) a la carpeta “/lib”. Para otros servidores de aplicaciones, podría ser necesario alguna configuración adicional..

En caso de contar con un servidor de *servlets*, se desplegará la distribución de los servicios compuesta de archivos WAR.

En caso de contar con un servidor completo de aplicaciones JEE, se podrá desplegar la distribución compuesta de archivos WAR o el fichero EAR del proyecto.

Para saber cómo realizar el despliegue de aplicaciones en su servidor, consulte la documentación de su software.

### Requisitos del usuario

El entorno de ejecución del usuario de FIRe también debe cumplir con ciertos requisitos cuando se permita u obligue al uso de certificados locales. Estos requisitos varían según la herramienta de firma que se configure. Los requisitos recomendados son:

* Navegador web:
  + Mozilla Firefox
  + Google Chrome
  + Microsoft Edge
  + Apple Safari
  + Microsoft Internet Explorer
* Cliente de firma:
  + AutoFirma 1.6.5
  + Cliente de firma Android 1.7.0
  + Cliente de firma iOS 1.7.0

En la página de firma con certificado local, se muestran al usuario los requisitos que necesita basándose en el entorno detectado.

Si desea exponer estos requisitos a sus usuarios antes de iniciar el trámite web, puede permitir mostrarles la página web localizada en “public/static/afirma\_compatibility.html” dentro del componente central de FIRe:



## Compatibilidad con dispositivos móviles

FIRe es compatible con dispositivos móviles a nivel general, pero pueden existir requisitos de compatibilidad concretos para los distintos proveedores de certificados.

Entre los proveedores incluidos de serie en FIRe, se identifican los siguientes requisitos de compatibilidad:

* **Firma con certificado local**:
  + Requiere tener instalado el Cliente de firma Android 1.7 o el Cliente de firma iOS 1.7.
  + Sólo está disponible para las operaciones de firma simple. No está disponible la operación de firma de múltiples documentos (firma de lote). En caso de iniciar una firma de lote desde un dispositivo móvil, FIRe mostrará inactivo el proveedor de firma local. En caso de forzar al uso de este proveedor, FIRe redirigirá automáticamente a la página de error indicada por la aplicación.

Las aplicaciones móviles son especialmente restrictivas con respecto a la confianza en los certificados SSL de las páginas y servicios a los que se accede. Si desea que su despliegue de FIRe sea compatible con dispositivos móviles, utilice para cifrar su conexión un certificado SSL reconocido internacionalmente. Una alternativa para los despliegues orientados a las aplicaciones utilizadas por un conjunto de usuarios gestionados es instalar el certificado SSL en el almacén de confianza de cada uno de los dispositivos de los usuarios.

## Despliegue

El componente central de FIRe expone una serie de servicios que sólo deben estar disponibles en el entorno en el que se encuentren las aplicaciones cliente que quieran usar sus funcionalidades de firma. Sin embargo, es necesario que otros servicios y recursos del componente central estén disponibles de cara a los usuarios finales y, por tanto, que se despliegue en un entorno público o DMZ.

Es necesario configurar el servidor de aplicaciones para que los accesos a los servicios y recursos del componente central sólo puedan realizarse sobre SSL con certificado cliente. Con esto se consigue que sólo puedan hacer uso de estos servicios las aplicaciones que se hayan dado de alta en el sistema y se autentiquen mediante el certificado cliente con el que se registrasen.

Sin embargo, existen páginas web, servicios y recursos del componente central que deben ser accesibles para el usuario (ruta “public” del módulo “fire-signature”) sin necesidad de autenticación, por lo que se deberá configurar una conexión SSL sin autenticación cliente para el acceso a ellos. Si se requiriese certificado cliente para acceder a estos recursos, el navegador web o Java se lo pediría directamente al usuario, dando lugar a un diálogo de selección que puede confundirle. Esto significa que se debe configurar el que, por defecto, sería el contexto “fire-signature/public” para que no solicite certificados cliente. Disponer esta configuración podría requerir del uso de un servidor web.

En el contexto “fire-signature/public” también se despliegan algunos servicios que requiere el Cliente @firma para su funcionamiento. Si en nuestro despliegue de FIRe habilitamos el proveedor “local” para permitir que las aplicaciones realicen firmas con certificado local, es importante tener en cuenta que el Cliente @firma comprueba por defecto la validez de los certificados SSL. Para evitar incidencias al respecto, debemos utilizar en nuestro despliegue un certificado SSL válido, correspondiente al dominio en el que se realiza el despliegue y reconocido por defecto por Java. En caso contrario, es posible que AutoFirma no pueda conectarse con los servicios del componente central de FIRe. Como alternativa, sobre todo para los entornos de desarrollo, el usuario podría deshabilitar la opción “Aceptar sólo conexiones con sitios seguros” en la pestaña General de las preferencias de AutoFirma. De esta forma, AutoFirma no tendrá problemas para acceder a los servicios del componente central. Para la prueba con aplicaciones móviles, sí se deberán utilizar certificados SSL reconocidos y válidos para el dominio en el que se utilicen.

Para habilitar el uso de SSL cliente salvo en el contexto “/public”, por ejemplo, si tuviésemos un servidor de aplicaciones Apache Tomcat y un servidor web Apache HTTPD con *virtualhosts* configurados, podríamos utilizar la siguiente configuración en el servidor Apache:

…

SSLEngine on

SSLVerifyClient none

# Habilitamos la autenticacion SSL cliente para acceso al componente central

<Location "/fire-signature">

SSLRenegBufferSize 20982000

LimitRequestBody 20982000

SecRequestBodyAccess Off

SSLVerifyClient optional\_no\_ca

SSLVerifyDepth 2

</Location>

# Deshabilitamos la autenticacion SSL cliente para el subcontexto publico

<Location "/fire-signature/public">

SSLVerifyClient none

</Location>

# Redirigir los certificados SSL cliente a Tomcat

SSLOptions +ExportCertData

…

## Configuración del componente

El componente central de FIRe utiliza múltiples ficheros de propiedades para configurar su comportamiento, cada uno de los cuales está orientado a establecer las propiedades principales del módulo, la configuración de un proveedor en particular, la configuración de un gestor de documentos o el acceso a un sistema remoto de validación y actualización de firmas.

Todos los ficheros de configuración del componente central siguen los criterios definidos en el apartado 9 Configuración de los componentes de FIRe.

El listado de ficheros de configuración utilizado por el componente central es el siguiente:

* config.properties
  + Configura las conexiones con la base de datos, ficheros temporales, el sistema de estadísticas, el sistema de logs y los proveedores que se pueden utilizar para firmar.
* platform.properties
  + Configura el acceso a la plataforma de validación y mejora de firmas.
  + Este fichero sólo se utilizará cuando las aplicaciones soliciten la validación de las firmas generadas o su actualización a formatos longevos. No es necesario incorporarlo si nuestras aplicaciones no necesitan esto.
* alarms\_config.properties
  + Configura la comunicación con el servicio de gestión de alarmas del componente.
  + Este fichero sólo se utilizará cuando en el componente central se configure que las alarmas críticas deben notificarse al administrador. Esto se configura a través de la propiedad “alarms.notifier” del fichero “config.properties”. No es necesario incorporarlo si no se configura esta propiedad.
* *provider\_PROVEEDOR*.properties
  + Por cada proveedor de firma en la nube configurado en el fichero “config.properties”, se debe crear un fichero “*provider\_PROVEEDOR*.properties” en donde “*PROVEEDOR*” es el nombre asignado al proveedor en cuestión.
  + Cada uno de estos ficheros será utilizado para configurar su conector correspondiente.
* docmanager.*GESTORDOCUMENTOS*.properties
  + Por cada gestor de documentos configurado y distinto al por defecto (que podría no haber ninguno) se cargará un fichero con nombre “docmanager.*GESTORDOCUMENTOS*.properties”, en donde “*GESTORDOCUMENTOS*” es el nombre asignado al gestor.
  + Cada uno de estos ficheros será utilizado para configurar su conector correspondiente.

### Fichero config.properties

El fichero config.properties contiene la configuración con la base de datos, ficheros temporales, la firma de lotes, los logs, el sistema de estadísticas, la clase para el descifrado de propiedades y el conector de *backend* que se debe utilizar. El listado completo de claves que se puede configurar son:

* bbdd.driver (Opcional)
  + Clase del driver JDBC que se desea utilizar.
  + Si esta propiedad no se encuentra en el fichero, se descartará el uso de base de datos, lo cual sólo permitiría el acceso a una única aplicación cliente. Revise la propiedad bbdd.conn, para más detalles de cómo evitar el uso de base de datos.
* bbdd.conn (Opcional)
  + Cadena de conexión a la base de datos del sistema.
  + Si esta propiedad no se encuentra en el fichero, se descartará el uso de base de datos, lo cual sólo permitiría el acceso a una única aplicación cliente. Si no se configura esta propiedad, **será obligatorio** establecer las siguientes dos propiedades en este fichero:
    - default.appId
      * Identificador de la única aplicación que puede solicitar firmas. Este identificador se comprobará cada vez que se reciba una petición. Puede ser cualquier cadena de texto.
    - default.certificate
      * Certificado codificado en base 64 con el que se autentica la aplicación.
* cipher.class (Opcional)
  + Nombre de la clase encargada de descifrar propiedades de este o el resto de ficheros de configuración del componente central.
  + La clase configurada debe implementar la interfaz “es.gob.fire.server.decipher.PropertyDecipher” localizada en el módulo fire-signature-decipher.
  + Los valores o fragmentos cifrados del fichero de configuración deberán expresarse de la forma: {@ciphered: DATO\_CIFRADO\_BASE64 }
  + Consulte el apartado 9.2 Cifrado de propiedades para más información.
* validator.class (Opcional)
  + Clase conectora para el uso de funciones de validación de firma y actualización a formatos longevos.
  + Para utilizar la Plataforma @firma como medio para la actualización y validación de las firmas generadas se utilizará el valor: es.gob.fire.upgrade.afirma.AfirmaValidator
  + No es necesario configurar esta propiedad si no se desea proporcionar la función de validación y actualización de firmas. En caso de que una aplicación solicite la validación o actualización de una firma y esta propiedad no se encuentre configurada se devolverá un error.
  + En caso de establecer esta propiedad, sería necesario configurar el acceso al sistema de validación y actualización por medio del fichero platform.properties.
* batch.maxDocuments
  + Número máximo de documentos que se pueden agregar a un lote de firma. Si se intentan agregar más documentos, la operación de agregar documento devolverá un error.
  + Si se establece el valor 0, se considerará que no hay límite de documentos en un lote.
  + Por defecto, 10.
* temp.dir
  + Ruta del directorio para el almacenamiento temporal de documentos.
  + Por defecto, directorio de temporales del sistema.
  + Se recomienda que se configure un directorio propio para los temporales de FIRe ya que sobre este directorio se aplicará la política de borrado de ficheros caducados.
* temp.fire.timeout
  + Número de segundos que pueden transcurrir antes de considerar caducado un fichero temporal de FIRe. Pasado ese tiempo, la sesión se considerará caducada y el fichero podría borrarse.
  + Por defecto, 10 minutos.
* sessions.dao (Opcional)
  + Gestor para la compartición de sesiones entre nodos balanceados.
  + **IMPORTANTE:** Esto solo debe usarse cuando se despliegue el componente central en varios nodos balanceados y no se compartan los objetos en memoria entre ellos.
  + Valores disponibles:
    - es.gob.fire.server.services.internal.sessions.FileSystemSessionsDAO
      * Compartición de sesiones a través de disco.
      * Al usar este valor, el valor por defecto para la propiedad “sessions.documents.dao” pasará a es “es.gob.fire.server.services.internal.sessions.FileSystemTempDocumentsDAO”.
    - es.gob.fire.server.services.internal.sessions.DBSessionsDAO
      * Compartición de sesiones a través de base de datos.
      * Requiere que se haya creado en el esquema de bases de datos las tablas concretas para el uso de esta funcionalidad.
      * Al usar este valor, el valor por defecto para la propiedad “sessions.documents.dao” pasará a es “es.gob.fire.server.services.internal.sessions.DBTempDocumentsDAO”.
      * Para el uso de este valor, es necesario que se cree en base de datos las tablas correspondientes para la compartición de sesiones y documentos.
* sessions.documents.dao (Opcional)
  + Gestor para la compartición de documentos temporales entre nodos balanceados.
  + Esto solo debe usarse cuando se despliegue el componente central en varios nodos balanceados, para que todos tengan acceso a los temporales para la gestión de las operaciones entre varios nodos.
  + El valor por defecto de esta propiedad depende del valor establecido en la propiedad “sessions.dao”.
  + Valores disponibles:
    - es.gob.fire.server.services.internal.sessions.FileSystemTempDocumentsDAO
      * Guardado de datos en disco a través del directorio temporal configurado en la propiedad "temp.dir"
    - es.gob.fire.server.services.internal.sessions.DBTempDocumentsDAO
      * Guardado de temporales en base de datos. Para la conexión con la base de datos se utiliza la propiedad "bbdd.conn".
      * Requiere que se haya creado en el esquema de bases de datos las tablas concretas para el uso de esta funcionalidad.
      * La configuración de este valor o el uso de este DAO por la configuración de la propiedad sessions.dao requerirá que se cree en base de datos las tablas correspondientes para la compartición de sesiones y documentos.
* http.cert.attr
  + Nombre del atributo en el que buscar los certificados SSL cliente cuando no se encuentren como atributos de la operación.
  + Esto puede ser necesario cuando se conecta un Apache y el servidor de aplicación con un proxy-pass en lugar de mediante AJP.
  + Comúnmente no será necesario configurar este parámetro.
  + Por defecto, “X-Client-Cert”.
* providers
  + Listado de proveedores habilitados para su uso por parte de las aplicaciones.
  + Los valores se indican consecutivos, separados por comas (',').
  + Al usuario se le mostraran los proveedores configurados y el orden que se indique en esta propiedad, salvo que la aplicación cliente defina su propia selección de proveedores. En ese caso, se mostrarán sólo los proveedores solicitados y en el orden indicado por la aplicación.
  + Si sólo se define un proveedor, se utilizará directamente sin necesidad de que lo seleccione el usuario.
  + Si el nombre de algún proveedor se antecede del carácter arroba ('@'), se considerará que es imprescindible que aparezca y se mostrará al usuario incluso si no estaba entre la selección de proveedores de la aplicación. Esta opción debe manejarse con cuidado, pues puede conllevar que en una aplicación se firme con certificados distintos a los esperados.
  + El nombre de proveedor "local", permite el uso de certificados locales.
  + Todos los proveedores distintos de "local" deben declarar en este fichero su clase conectora mediante una propiedad llamada "provider.*PROVEEDOR*".
  + Por cada proveedor distinto de “local” se debería agregar un fichero “*PROVEEDOR*\_config.properties” con la configuración necesaria para que el conector acceda a él.
  + Por ejemplo, si se establece el valor “clavefirmatest,local”, se debería configurar la propiedad “provider.clavefirmatest” con el nombre de la clase conectora de este proveedor y crearse el fichero “clavefirmatest\_config.properties” con las propiedades de configuración que requiera este conector. A los usuarios, por defecto, les aparecería como opciones para firmar el proveedor de firma en la nube “clavefirmatest” (nombre por defecto del simulador de Cl@ve Firma) y la opción de firma con certificado local.
* provider.*PROVEEDOR*
  + Propiedad que deberá establecerse por cada proveedor definido en la propiedad “providers”.
  + Deberá asignarse a esta propiedad el nombre completo de la clase conectora del proveedor.
  + FIRe integra, por defecto, las siguientes clases conectoras de proveedores en la nube:
    - es.gob.fire.server.connector.clavefirma.ClaveFirmaConnector
      * Clase conectora para el uso de los certificados de Cl@ve Firma.
    - es.gob.fire.server.connector.test.TestConnector
      * Clase conectora el uso de un servicio de pruebas que emula el funcionamiento y la apariencia de Cl@ve Firma.
      * El servicio de pruebas al que se conecta se encuentra en el archivo “clavefirma-test-services.war”, distribuido junto con FIRe.
    - es.fnmt.fire.signature.connector.TrustedXConnector
      * Clase conectora del proveedor de CERES Cloud ID.
      * Para que este conector sea funcional es necesario el despliegue del servicio “fnmt-fire-service.war”.
  + Consulte el apartado 10 Nuevos proveedores de firma para saber cómo agregar nuevos proveedores a FIRe.
* provider.*PROVEEDOR*.secure (Opcional)
  + Propiedad que opcionalmente puede establecerse por cada proveedor definido en la propiedad “providers”.
  + Esta propiedad permite definir si el proveedor con nombre “*PROVEEDOR*” se considera seguro (true) o no (false).
  + Si el proveedor se considera seguro, FIRe se ahorrará validar las firmas generadas con sus certificados, ya que se sabe que todos ellos son válidos y de confianza. Sí se validarán las multifirmas cuando se solicite, ya que otros certificados de la firma podrían no ser válidos.
  + Por defecto, los proveedores de Cl@ve Firma y la CERES Cloud ID deberían considerarse seguros, ya que ninguno de los dos permitiría firmar con el certificado si este no estuviese vigente y activo, por lo que no se necesita validación.
  + El proveedor “local” nunca se considerará seguro.
* provider.*PROVEEDOR*.info.file (Opcional)
  + Propiedad que opcionalmente puede establecerse por cada proveedor definido en la propiedad “providers” para indicar que se desea que el proveedor se muestre con una apariencia distinta a la establecida por defecto por el conector que utiliza.
  + En esta propiedad se puede establecer el nombre de un fichero Properties con la configuración que se desee que se aplique al proveedor con nombre “*PROVEEDOR*”.
  + Consulte el apartado 4.4.7 Fichero de apariencia de conector para más detalles.
* local.verification.key (Opcional)
  + Cadena de texto a partir de la cual generar una clave HMAC con la que comprobar los cifrados realizados por la aplicación de firma local (Cliente @firma).
  + Cuando se solicita firmar con un certificado local, se preparan los datos a firmar y se envía al equipo del usuario los datos mínimos que debe cifrar con la clave privada del certificado de firma. Después, la aplicación devuelve a FIRe los datos que ha cifrado y los datos ya cifrados. El valor de esta cadena se utiliza para generar una clave HMAC que nos permita identificar que los datos enviados a cifrar y los datos recibidos de vuelva no se han modificado. En caso de no serlo, es posible que alguien estuviese realizando un ataque *man-in-the-middle* (MiTM), capturando los datos a firmar y firmando otros en su lugar, y se lanzaría un error.
  + En caso de no establecerse valor en esta propiedad, no se realizará la verificación descrita.
* docmanager.default
  + Gestor de documentos por defecto. Este es el gestor que se aplicará cuando la aplicación no indique expresamente cual desea utilizar.
  + Este valor sólo debería cambiarse cuando el comportamiento de todas las aplicaciones que hagan uso del componente central obtengan los datos y almacenen las firmas mediante gestores de documentos distintos al por defecto y se desee simplificarles la integración para que omitan la configuración del gestor por parámetro.
  + Por defecto, “es.gob.fire.server.services.document.DefaultFIReDocumentManager”. Este gestor establece que los datos deberán ser proporcionados por la aplicación y las firmas generadas se devolverán a la misma.
  + Consulte el apartado 4.6 Clases gestoras de documentos para saber más de las clases gestoras de documentos.
* docmanager.*NOMBREGESTOR* (Opcional)
  + Define un nuevo gestor de documentos cuyo nombre será el indicado por el texto que sustituya a la cadena *NOMBREGESTOR*.
  + Se puede definir esta propiedad tantas veces como se desee, utilizando cada vez un nombre de gestor distinto.
  + El valor de esta propiedad deberá ser el nombre cualificado de la clase gestora de documentos.
  + Consulte el apartado 4.6 Clases gestoras de documentos para saber más de las clases gestoras de documentos.
* docmanager.*NOMBREGESTOR*.requestor.valid (Opcional)
  + Define cuales de las aplicaciones dadas de alta en el sistema pueden utilizar el gestor de documentos *NOMBREGESTOR*.
  + El valor a establecer en esta propiedad será el listado de los identificadores de aplicación, separados por comas (‘,’), que pueden usar el gestor.
  + Si no se indica esta propiedad, se le permitirá el uso del gestor a todas las aplicaciones que no aparezcan en el listado de aplicaciones no permitidas (docmanager.*NOMBREGESTOR*.requestor.invalid) o, directamente, a todas si no se indica tampoco esa propiedad.
* docmanager.*NOMBREGESTOR*.requestor.invalid (Opcional)
  + Define cuales de las aplicaciones dadas de alta no pueden utilizar el gestor de documentos *NOMBREGESTOR*.
  + El valor a establecer en esta propiedad será el listado de los identificadores de aplicación, separados por comas (‘,’), que no pueden usar el gestor.
  + Si no se indica esta propiedad, se permitirá el uso del gestor a todas las aplicaciones o, si se indicó un listado de aplicaciones permitidas (docmanager.*NOMBREGESTOR*.requestor.valid), a las indicadas en dicho listado.
* alarms.notifier (Opcional)
  + Establece el nombre de la clase encargada de notificar errores críticos en el sistema. Algunos errores, por ejemplo, son que no se pueda acceder a un recurso o que haya fallado la conexión con un proveedor de firma.
  + Si no se establece esta propiedad, estos errores no se informarán de un modo particular. Únicamente quedarán registrados en la traza de registro de la aplicación.
  + Con FIRe se proporciona una implementación de notificador que envía los avisos a GrayLog. Para configurar este notificador se debe utilizar el valor:
    - es.gob.fire.alarms.GrayLogAlarmNotifier
  + Si se establece esta propiedad, será necesario proporcionar el fichero de configuración "alarms\_config.properties" con las propiedades para el funcionamiento del clase.
  + Consulte el apartado 4.8 Clases de notificación de errores críticos para saber más de las clases gestoras de documentos.
* pages.title
  + Título que aparecerá en las páginas web del componente central.
  + Se permite el uso de entidades HTML para insertar caracteres que puedan producir problemas de codificación ("&aacute;", "&ntilde;", "&amp;"...)
  + Por defecto, “FIRma Electrónica – FIRe”.
* pages.logo (Opcional)
  + URL externa del logotipo que mostrar en las páginas web del componente central.
  + Por defecto, no se proporciona un valor y se muestra el logotipo de Gobierno de España.
* pages.public.url
  + URL base desde la que acceder al contexto público del componente central (páginas y servicios que deben estar accesibles para los usuarios finales).
  + Esta propiedad sólo se debería configurar cuando en el servidor web se establezcan URL diferenciadas para el acceso al contexto público y el general del componente central. A la URL que se configure siempre se le agregará el sufijo “/public”. Esto puede ser necesario cuando no se pueda o no se desee exponer los servicios privados de FIRe de cara a Internet.
  + Si se realizan dos despliegues separados del componente central (uno para atender las llamadas de los usuarios y otro para atender las llamadas del servidor) será necesario configurar la propiedad "sessions.dao" para permitir compartir las sesiones entre ellos.
  + Por defecto, no se proporciona un valor y se accederá a las páginas utilizando el contexto de despliegue del WAR seguido de “/public”.
* logs.dir (Opcional)
  + Ruta absoluta del directorio en el que almacenar el fichero de *log* del componente central. El directorio debe existir y se deben poseer permisos de lectura sobre el mismo.
  + El fichero se llamará “fire-signature.log” y, conforme rote (según la configuración de la propiedad “logs.rollingPolicy”), se le irá agregando el sufijo de la fecha/hora a la que pertenece.
  + Si no se configura o no se puede crear/editar el fichero, no se generarán *logs* en un fichero externo.
  + Se configure o no, los logs se seguirán imprimiendo en el fichero de log del servidor de aplicaciones.
* logs.rollingPolicy (Opcional)
  + Política de rotado del fichero de *log*. Admite los valores:
    - DAY: Rota el archivo a la media noche todos los días.
    - HOUR: Rota el archivo al inicio de cada hora.
    - MINUTE: Rota el archivo al inicio de cada minuto.
  + Según la política seleccionada, se agregará como sufijo la fecha/hora al nombre de los ficheros de *logs* conforme van rotando.
  + Esta propiedad no tiene efecto si no se ha establecido una ruta válida en la propiedad “logs.dir”.
  + Por defecto, DAY.
* logs.level.fire (Opcional)
  + Nivel mínimo de las trazas de *log* generadas por FIRe que se desean mostrar.
  + Los niveles de *log* admitidos son, de menor a mayor:
    - FINE: Muestra las trazas de error, de advertencia, de operación y de depuración.
    - INFO: Muestra las trazas de error, de advertencia y de operación.
    - WARNING: Muestra las trazas de error y de advertencia.
    - SEVERE: Sólo muestra las trazas de error.
    - OFF: Sin trazas.
  + Si no se establece, se utilizará el valor INFO.
  + Por defecto, INFO.
* logs.level.afirma (Opcional)
  + Nivel mínimo de las trazas de *log* generadas por las bibliotecas de firma utilizadas por FIRe que se desean mostrar.
  + Los niveles de *log* admitidos son, de menor a mayor:
    - FINE: Muestra las trazas de error, de advertencia, de operación y de depuración.
    - INFO: Muestra las trazas de error, de advertencia y de operación.
    - WARNING: Muestra las trazas de error y de advertencia.
    - SEVERE: Sólo muestra las trazas de error.
    - OFF: Sin trazas.
  + Si no se establece, se utilizará el valor WARNING.
  + Por defecto, WARNING.
* logs.level (Opcional)
  + Nivel mínimo de las trazas de *log* generados por el resto de bibliotecas integradas en el componente central de FIRe que se desean mostrar.
  + Este valor permite ajustar el nivel de trazas de cualquier biblioteca que no utilice los *logs* “es.gob.afirma” y “es.gob.fire”, como clases importadas (FireDocumentManager, PropertyDecipher…), conectores con servicios remotos distintos al de Cl@ve Firma y bibliotecas externas utilizadas.
  + Los niveles de *log* admitidos son, de menor a mayor:
    - FINEST
    - FINER
    - FINE
    - CONFIG
    - INFO
    - WARNING
    - SEVERE
    - OFF
  + Por defecto, OFF.
* statistics.policy
  + Establece la política del guardado de datos para la generación de estadísticas referentes a las firmas y las transacciones realizadas.
  + Los valores disponibles son:
    - 0: No se registrarán datos para la generación de estadísticas. Este es el valor por defecto.
    - 1: Se guardarán datos en disco para que el usuario administrador volcarlos en base de datos. El volcado en base de datos puede realizarse mediante la herramienta FireStatisticsDbLoader.jar. Consulte el apartado ANEXO IV Volcado de estadísticas a base de datos para más información.
    - 2: -> Se registran las estadísticas en ficheros log y se realiza automáticamente un volcado diario a base de datos.
  + Los ficheros con las estadísticas de uso se almacenan en el directorio configurado en la propiedad “statistics.dir”.
  + Para la explotación de los datos estadísticos es necesario configurar FIRe con una base de datos y hacer uso del módulo de administración de firma.
  + Los datos guardados engloban la configuración de las firmas generadas, si eran firma simples o parte de una operación de firma masiva, la aplicación que las solicita, el tamaño de los datos procesados, el identificador del proveedor de firma empleado (Cl@ve Firma, si se usó certificado local, etc.), el tipo de navegador web del usuario y si la operación finalizó correctamente o si se produjo algún error.
  + En ningún caso se almacenan datos del usuario, información de sus certificados o el contenido de los documentos firmados.
* statistics.dir (Opcional)
  + Ruta del directorio en el que se almacenarán los datos estadísticos.
  + Si no se configura, no se generarán los datos estadísticos.
* statistics.dumptime (Opcional)
  + Configura la hora a la que se realizará a diario el volcado de los datos estadísticos en base de datos.
  + Esta propiedad sólo tiene efecto cuando se configura el valor "2" en la propiedad "statistics.policy".
  + La hora se deberá indicar mediante una cadena de texto con formato “HH:MM:SS”, donde:
    - HH: Hora del día en formato 24 horas.
    - MM: Minutos.
    - SS: Segundos.
  + Si se configura la política de volcado automático en base de datos, pero no se establece este valor, el volcado se realizará a las 00:00 horas de cada día.

#### Fichero de ejemplo

Un ejemplo de fichero de configuración sería:

bbdd.driver=com.mysql.jdbc.Driver

bbdd.conn=jdbc:mysql://172.24.31.110:3306/fire\_db?user=us&password={@ciphered:UhA/s67=}

cipher.class=es.gob.fire.mipaquete.ClaseDescifrado

validator.class=es.gob.fire.upgrade.afirma.AfirmaValidator

batch.maxDocuments=30

temp.dir=C:/pruebas/temp\_clavefirma

temp.fire.timeout=600

http.cert.attr=X-Client-Cert

providers=clavefirma,local

provider.clavefirma=es.gob.fire.server.connector.clavefirma.ClaveFirmaConnector

provider.clavefirma.secure=true

local.verification.key=HolaMundo!

docmanager.default=es.gob.fire.server.services.document.DefaultFIReDocumentManager

pages.title=FIRe

pages.logo=https://miorganismo/resources/img/logo.jpg

logs.dir=C:/pruebas/logs

logs.rollingPolicy=DAY

logs.level.fire=FINE

logs.level.afirma=WARNING

logs.level=WARNING

statistics.policy=2

statistics.dir=C:/pruebas/Estadisticas

statistics.dumptime=16:50:00

En este fichero se indica que la conexión a la base de datos se realizará mediante un controlador JDBC de MySQL con la cadena de conexión “jdbc:mysql://172.24.31.110:3306/fire\_db?user=us&password={@ciphered:UhA/s67=}” (consulte con la documentación de su controlador JDBC para determinar el formato apropiado para su cadena de conexión), en donde “{@ciphered:UhA/s67=}” será sustituido por la cadena resultante de descifrar el texto Base 64 “UhA/s67=” mediante la clase “es.gob.fire.mipaquete.ClaseDescifrado”. Para la validación y actualización de firmas se utilizará el conector con la Plataforma @firma (“es.gob.fire.upgrade.afirma.AfirmaValidator”). Los datos específicos para la conexión con @firma se deberán indicar en el fichero platform.properties. Se ha definido que el número máximo de documentos que se puede adjuntar a un lote será 30, cuál es el directorio para el guardado de temporales y que el tiempo de caducidad de estos es de 600 segundos. El certificado cliente SSL puede llegar a través de la cabecera HTTP “X-Client-Cert”. Se han definido dos proveedores: “clavefirma”, que tiene la clase conectora “es.gob.fire.server.connector.clavefirma.ClaveFirmaConnector” (el conector de Cl@ve Firma) y se considerar un proveedor seguro; y “local”, que permite utilizar los certificados disponibles en el equipo del usuario. El proveedor local usará la cadena “HolaMundo!” para generar la clave HMAC con la que se realizarán las validaciones de las peticiones enviadas a firma con el Cliente @firma. Se usará el gestor de documentos por defecto, el título de la página de FIRe que se mostrará al usuario será “FIRe” y en la cabecera aparecerá el logo que se encuentra en la URL “https://miorganismo/resources/img/logo.jpg”. Los logs se generarán en el directorio “C:/pruebas/logs” e irán rotando a diario. Se mostrarán todas las trazas de FIRe (incluidas las de debug) pero las trazas del núcleo de firma y el resto de bibliotecas sólo se mostrarán si muestran advertencias o errores. Los datos datos estadísticos se irán almacenando en ficheros y volcándose a diario en base de datos (statistics.policy=2). Los ficheros se almacenarán en el directorio “C:/pruebas/Estadisticas” y se volcarán todos los días a las 16:50.

### Fichero platform.properties

Este fichero configura la conexión con el servicio de validación y actualización de firmas a formatos longevos. En FIRe se incluye la lógica para realizar estas operaciones mediante la Plataforma @firma y, en este apartado, se listan las propiedades necesarias para poder conectar con ella y realizar las diversas operaciones.

Si deseamos hacer uso de una instancia de la Plataforma @firma para la actualización y validación de firmas, el administrador de la misma deberá dar de alta su aplicación y habilitarle las credenciales de acceso y permisos correspondientes. Consulte con el administrador de la Plataforma @firma a la que desea acceder para más detalles.

Un desarrollador podría implementar mecanismos de validación y actualización que no dependiesen de la Plataforma @firma y configurar el componente central para utilizarlos. En caso de hacerlo, las propiedades que deberían aparecer en ese fichero serían las necesarias por este otro mecanismo de validación y actualización de firmas.

Si no se desease validar las firmas generadas ni extenderlas a formatos longevos desde FIRe, no sería necesario configurar este fichero ni tener acceso a la Plataforma @firma.

Para conectar con la Plataforma @firma es necesario definir en este fichero las siguientes claves:

* afirma.appId
  + Identificador de la aplicación frente a la Plataforma @firma.
* webservices.timeout
  + Tiempo máximo de espera tras el envío de una petición expresado en milisegundos.
* webservices.endpoint
  + URL del servicio general de la Plataforma @firma, debe tener la barra “/” al final.
  + La URL del servicio la proporciona el administrador de la instancia de la Plataforma @firma.
* webservices.service.signupgrade
  + Nombre del servicio Web de mejora de firmas.
  + El nombre del servicio lo proporciona el administrador de la instancia de la Plataforma @firma. Comúnmente, DSSAfirmaVerify.
* webservices.service.recoversignature
  + Nombre del servicio Web de recuperación de firmas actualizadas después de un periodo de gracia.
  + El nombre del servicio lo proporciona el administrador de la instancia de la Plataforma @firma. Comúnmente, DSSAsyncRequestStatus.
* com.trustedstore.path
  + Ruta del almacén con los certificados de confianza para las conexiones SSL.
  + Si no se indica, se usará el almacén por defecto de la JRE o el indicado en la ejecución del servidor de aplicaciones.
* com.trustedstore.password
  + Contraseña del almacén con los certificados de confianza para las conexiones SSL.
* com.trustedstore.type
  + Tipo del almacén con los certificados de confianza para las conexiones SSL. Puede tener los valores:
    - JKS: Almacén de claves de Java.
    - PKCS12: Almacén de claves PKCS#12.
  + El valor por defecto viene predefinido por la JRE empleada.
* webservices.authorization.method
  + Tipo de autenticación contra la Plataforma @firma. Puede tener los valores:
    - none
      * Acceso al servicio sin autenticación. Es el valor por defecto.
    - BinarySecurityToken
      * Autenticación mediante certificado. Este es el valor que deberá usarse habitualmente.
    - UsernameToken
      * Autenticación con usuario y contraseña.
* webservices.authorization.ks.path
  + Almacén de claves con los certificados y claves privadas para la autenticación SSL cliente contra la Plataforma @firma.
  + El administrador de la instancia de la Plataforma @firma en cuestión deberá habilitar el acceso por medio de la clave pública del certificado que se desee utilizar.
* webservices.authorization.ks.type
  + Tipo del almacén de claves con los certificados y claves privadas para la autenticación SSL cliente contra la Plataforma @firma. Puede tener los valores:
    - JKS: Almacén de claves de Java.
    - PKCS12: Almacén de claves PKCS#12.
  + El valor por defecto viene predefinido por la JRE empleada.
* webservices.authorization.ks.password
  + Contraseña del almacén de claves con los certificados y claves privadas para la autenticación SSL cliente contra la Plataforma @firma.
* webservices.authorization.ks.cert.alias
  + Alias del certificado dentro del almacén indicado que se debe usar para la autenticación SSL cliente contra la Plataforma @firma.
  + Se recomienda que se evite el uso de alias con caracteres no ASCII.
* webservices.authorization.ks.cert.password
  + Contraseña del certificado dentro del almacén indicado que se debe usar para la autenticación SSL cliente contra la Plataforma @firma.
* webservices.authorization.user.name
  + Nombre de usuario que se debe usar para la autenticación contra la Plataforma @firma cuando se utiliza el método de autenticación “UsernameToken”.
* webservices.authorization.user.password
  + Contraseña que se debe usar para la autenticación contra la Plataforma @firma cuando se utiliza el método de autenticación “UsernameToken”.
* webservices.authentication.ts.path
  + Ruta del almacén con el certificado utilizado para la firma de las respuestas de la Plataforma @firma.
  + Si no se indica el almacén, la contraseña o el alias del certificado, no se validará la firma de la respuesta.
* webservices.authentication.ts.password
  + Contraseña del almacén con el certificado de firma de las respuestas del servicio.
* webservices.authentication.ts.type
  + Tipo del almacén de certificados con el certificado de firma. Puede tener los valores:
    - JKS: Almacén de claves de Java.
    - PKCS12: Almacén de claves PKCS#12.
  + El valor por defecto viene predefinido por la JRE empleada.
* webservices.authentication.cert.alias
  + Alias con el que se identifica dentro del almacén al certificado de firma de las respuestas de la Plataforma @firma.

Un ejemplo de fichero de propiedades para acceso a la Plataforma @firma para validación y mejora de las firmas podría ser el siguiente, a expensas de rellenar los datos proporcionados por el administrador de su proveedor de servicios de @firma.

# Identificador de la aplicacion frente a la Plataforma @firma

afirma.appId=gob.sgad.fire

# Timeout de conexion (-1 sin timeout)

webservices.timeout=50000

# URL del servicio Afirma. Debe tener la barra ("/") final

webservices.endpoint = https://afirma.es/services/

# Servicio DSS de validación y actualización de firmas

webservices.service.signupgrade=DSSAfirmaVerify

# Servicio DSS de recuperación de asíncrona de firmas

webservices.service.recoversignature=DSSAsyncRequestStatus

# Metodo de autenticacion

webservices.authorization.method=BinarySecurityToken

# Almacen de confianza para conexiones seguras

com.trustedstore.path = /opt/users/fire/cert/afirma\_ts.jks

com.trustedstore.password = pruebas

com.trustedstore.type = JKS

# Propiedades para el método BinarySecurityToken

webservices.authorization.ks.path = /opt/users/fire/cert/afirma\_ks.p12

webservices.authorization.ks.type = PKCS12

webservices.authorization.ks.password = pruebas

webservices.authorization.ks.cert.alias = certAfirma

webservices.authorization.ks.cert.password = pruebas

# Almacen con el certificado de firma de las respuestas

webservices.authentication.ts.path = /opt/users/fire/cert/ws\_response\_ts.jks

webservices.authentication.ts.password = pruebas

webservices.authentication.ts.type = JKS

webservices.authentication.ts.cert.alias = cert\_firma\_soap

### Fichero alarms\_config.properties

Este fichero configura la clase de envío de notificaciones de FIRe, encargada de enviar los errores críticos en el sistema. El envío de estas notificaciones es opcional y está deshabilitada por defecto, por lo que este fichero de configuración no será necesario a menos que el administrador active expresamente las notificaciones por medio de la propiedad “alarms.notifier” del fichero “config.properties”.

En este fichero se deben establecer las propiedades necesarias para configurar el envío de notificaciones, así como cualquier valor estático que se desee dependiente de la aplicación o el entorno de ejecución. Las propiedades concretas dependen de la clase de envío de notificaciones configurada. Si el administrador configurase una clase de envío de notificaciones propia, las propiedades de configuración establecidas en este fichero son las que se enviarán al método init(Properties) de dicha clase.

#### Envío de notificaciones a GrayLog

FIRe incluye una clase para el envío de las notificaciones a GrayLog. En caso de configurar dicha clase (consulte la propiedad “alarms.notifier” en el apartado 4.3.1 Fichero config.properties), las propiedades que deben configurarse en el fichero alarms\_config.properties fichero son:

* enabled
  + Permite habilitar o deshabilitar el envío de notificaciones. Esto permitiría evitar que se envíen las notificaciones incluso cuando la clase de envío a GrayLog está configurada en config.properties.
  + Con el valor “true” se envían las notificaciones a GrayLog. Con el valor “false”, no se envían.
  + Por defecto, “false”.
* destination.host
  + Establece el dominio o IP del servidor de GrayLog.
* destination.port
  + Establece el puerto en el que escucha las peticiones el servidor de GrayLog.

Adicionalmente, se enviarán a GrayLog en cada notificación el valor de cualquier propiedad configurada en este fichero y cuyo nombre empiece por el prefijo “field.”. Esto permitirá recibir en el *stream* de GrayLog información estática sobre el despliegue que envía la notificación. Algunos ejemplos de valores que se podrían proporcionar son:

* field.app
  + Nombre de la aplicación (FIRe), por si el *stream* de GrayLog diese servicio a varias.
* field.entorno
  + Identificador del entorno de despliegue (PRO, PRE, DES, etc). Esta información se enviará junto a cada notificación para que la instancia de GrayLog, si es compartida por varios entornos, pueda diferenciar desde qué entorno se envía cada una de ellas.
* field.host
  + Nombre o IP del equipo desde el que se emite la notificación. Esta información puede resultar de interés si disponemos de un despliegue en alta disponibilidad y quisiéramos averiguar que nodo es el que notifica el problema. En este caso, el valor debería ser distinto por cada nodo en el que se encuentre FIRe desplegado.

Un ejemplo de fichero de propiedades para el envío de notificaciones a GrayLog sería el siguiente:

# Acceso habilitado

enabled=true

# IP de acceso al servicio de GrayLog

destination.host=192.157.32.15

# Puerto de acceso al servicio de GrayLog

destination.port=9000

# Nombre de la aplicacion

field.app=fire

# Entorno en el que se encuentra desplegado el servicio (PRE, PRO, DES, etc)

field.entorno=DES

### Fichero provider\_clavefirma.properties

Es el fichero por defecto para la configuración del conector de Cl@ve Firma. Este fichero sólo será necesario cuando se configure la clase conectora “es.gob.test.server.connector.clavefirma.ClaveFirmaConnector”.

En caso de que el administrador cambiase el nombre de proveedor que configura esta clase conectora, será necesario cambiar el nombre de este fichero a como corresponda. Por ejemplo, si quisiéramos llamar a este proveedor “giss”, a este fichero lo llamaríamos provider\_giss.properties, en el fichero config.properties estableceríamos la propiedad “provider.giss=es.gob.test.server.connector.clavefirma.ClaveFirmaConnector” para configurar el conector y en el mismo fichero habilitaríamos el uso del conector agregando el nombre “giss” al listado de proveedores indicados en la propiedad “providers”.

***Los valores a configurar en este fichero deberían ser proporcionadas por la GISS, ya que son los que se usarán para acceder al servicio remoto de Cl@ve Firma.***

Las propiedades a configurar en este fichero son:

* providerName
  + Configura el nombre de proveedor con el que se identifica el servicio contra la GISS.
  + Este parámetro es obligatorio, debe ir en MAYÚSCULAS y se forma de la concatenación del NIF del organismo y del código DIR3 unidos por un guion bajo (NIF\_DIR3), tal como se ha especificado en al ACL (formulario) de alta en el sistema de la GISS.
* allowRequestNewCert
  + Configura si el proveedor debe permitir a los usuarios el generar certificados cuando no tengan ninguno expedido (true) o que no se permita generarlos (false).
  + Por defecto, true.
* URL\_GATEWAY
  + URL hacia la pasarela de firma centralizada.
* AUTH\_STORE
  + Ruta hacia el almacén PKCS#12 con el certificado cliente autorizado para acceder al servidor remoto de firma centralizada.
* AUTH\_STORE\_PASS
  + Contraseña del almacén PKCS#12 con el certificado cliente autorizado para acceder al servidor remoto de firma centralizada indicado anteriormente.

Un ejemplo de este fichero podría ser:

providerName=S1234567E\_E12345678

allowRequestNewCert=true

URL\_GATEWAY=https://clavefirma.giss.es/ClaveFirmaService

AUTH\_STORE=//users/apache/clavefirma/auth.p12

AUTH\_STORE\_PASS=12345678

### Fichero provider\_clavefirmatest.properties

Es el fichero por defecto para la configuración del conector del simulador de pruebas de Cl@ve Firma. Este fichero sólo será necesario cuando se configure la clase conectora “es.gob.test.server.connector.test.TestConnector”.

En caso de que el administrador cambiase el nombre de proveedor que configura esta clase conectora, será necesario cambiar el nombre de este fichero a como corresponda. Por ejemplo, si quisiéramos llamar a este proveedor “test”, a este fichero lo llamaríamos provider\_test.properties, en el fichero config.properties estableceríamos la propiedad “provider.test=es.gob.test.server.connector.test.TestConnector” para configurar el conector y en el mismo fichero habilitaríamos el uso del conector agregando el nombre “test” al listado de proveedores indicados en la propiedad “providers”.

Las propiedades a configurar en este fichero son:

* endpoint
  + Permite configurar la ruta base de los servicios de prueba.
  + Si no se establece esta propiedad o se establece vacía, se interpretará que la URL base de los servicios de prueba es:

https://127.0.0.1:8443/clavefirma-test-services/

* allowRequestNewCert
  + Configura si el proveedor debe permitir a los usuarios el generar certificados cuando no tengan ninguno expedido (true) o que no se permita generarlos (false).
  + Esta opción permite emular a la propiedad homónima del conector de Cl@ve Firma.
  + Por defecto, true.
* ssl.keystore (Opcional)
  + Ruta absoluta del almacén de claves de autenticación en caso de que el servicio de pruebas se encuentre en un servidor con SSL con autenticación cliente.
  + En caso de no indicarse, la conexión con el servicio de pruebas no debe requerir autenticación SSL cliente.
* ssl.keystorePass (Opcional)
  + Contraseña del almacén de claves de autenticación en caso de que el servicio de pruebas se encuentre en un servidor con SSL con autenticación cliente.
  + En caso de no indicarse, la conexión con el servicio de pruebas no debe requerir autenticación SSL cliente.
* ssl.keystoreType (Opcional)
  + Tipo de almacén de claves de autenticación en caso de que el servicio de pruebas se encuentre en un servidor con SSL con autenticación cliente. Los valores posibles son “PKCS12” y “JKS”.
  + En caso de no indicarse, se usará el tipo de almacén por defecto de la JRE.
* ssl.truststore
  + Ruta absoluta del almacén de certificados de confianza en caso de querer configurar un almacén particular en lugar del por defecto de Java.
  + Si no se desean realizar autenticaciones sobre el certificado SSL del servidor, se puede configurar el valor “all”.
* ssl.truststorePass (Opcional)
  + Contraseña del almacén de certificado de confianza en caso de querer configurar un almacén particular en lugar del por defecto de Java.
  + No es necesario configurarla si en la propiedad “ssl.truststore” se ha configurado el valor “all”.
* ssl.truststoreType (Opcional)
  + Tipo de almacén de certificados de confianza en caso de querer configurar un almacén particular en lugar del por defecto de Java. Los valores posibles son “PKCS12” y “JKS”.
  + En caso de no indicarse, se usará el tipo de almacén por defecto de la JRE.
  + No es necesario configurarla si en la propiedad “ssl.truststore” se ha configurado el valor “all”.

Un ejemplo de este fichero podría ser:

endpoint=https://clavefirmagiss:8443/clavefirma-test-services

allowRequestNewCert=true

ssl.keystore=C:/Users/ClaveFirma/certificados/client\_ssl.jks

ssl.keystorePass=12341234

ssl.keystoreType=JKS

ssl.truststore=all

#ssl.truststorePass=

#ssl.truststoreType=

### Fichero provider\_fnmt.properties

Este es el fichero de configuración del conector con el servicio de firma en la nube de la FNMT (CERES Cloud ID) y sólo será necesario cuando se configure el uso de este proveedor a través de la clase conectora “es.fnmt.fire.signature.connector.TrustedXConnector”.

En caso de que el administrador cambiase el nombre de proveedor que configura esta clase conectora, será necesario cambiar el nombre de este fichero a como corresponda. Por ejemplo, si quisiéramos llamar a este proveedor “cloudid”, a este fichero lo llamaríamos provider\_cloudid.properties, en el fichero config.properties estableceríamos la propiedad “provider.cloudid=es.fnmt.fire.signature.connector.TrustedXConnector” para configurar el conector y en el mismo fichero habilitaríamos el uso del conector agregando el nombre “cloudid” al listado de proveedores indicados en la propiedad “providers”.

Para el funcionamiento del conector de CERES Cloud ID es necesario además configurar unos ficheros adicionales de configuración y desplegar el servicio “fnmt-fire-service.war”. Consulte el apartado para conocer el resto de ficheros de configuración de los que hay que disponer y el apartado 8.1 Servicio auxiliar del conector de la FNMT para más información sobre el servicio auxiliar del conector de la FNMT.

El conector de la FNMT está inicialmente preparado para la firma con certificados de empleado público y los valores de las opciones de configuración incluidas por defecto están orientados a tal fin.

***Las propiedades de este fichero deberían ser proporcionadas por la FNMT en base al acuerdo alcanzado con ella.*** En este apartado se muestran los valores comunes para la configuración del uso de certificados de empleado público.

Las propiedades a configurar en este fichero son:

* dtsCacher
  + Indica la clase Java que se encargará del almacén temporal de firmas en el sistema.
  + Por defecto, debe utilizarse el valor “es.fnmt.fire.signature.connector.DataToSignCacherFileSystem”.
* trustedXUrl
  + URL del servicio TrustedX eIDAS.
* trustedXUser
  + Nombre de usuario del servicio TrustedX eIDAS.
* trustedXPassword
  + Contraseña del usuario del servicio TrustedX eIDAS.
* trustedXClientId
  + Identificador de la aplicación cliente de TrustedX eIDAS.
* trustedXApiKey
  + Código del API de TrustedX eIDAS para la aplicación cliente.
* authServiceId
  + Identificador del servicio de autenticación.
  + Por defecto, debe utilizarse el valor “as-employee-fire”.
* defaultDomain
  + Dominio por defecto del servicio.
  + Por defecto, debe utilizarse el valor “employee”.
* caTxNoQSCD
  + Lista de CAs que NO funcionan como QSCD. Los nombres deben aparecer separados por comas.
  + Estos nombres se utilizan como filtro, por lo que no es necesario el nombre completo, basta con una parte del mismo.
  + Si no se indica, se tomarán todas las CAs como QSCD.
* definedLabels
  + Lista de etiquetas de TrustetX que definen las identidades cloudid (separadas por comas).
  + Por defecto, debe utilizarse el valor “fnmt,cloudid,grupo2,x509:keyUsage:contentCommitment”.
* definedLabels\_QSCD
  + Lista de etiquetas de TrustetX que definen las identidades QSCD (separadas por comas).
  + Por defecto, debe utilizarse el valor “fnmt,cloudidqscd,grupo3,x509:keyUsage:contentCommitment”.
* codeService
  + URL del servicio de gestión de redirecciones *Oauth*. Este es un servicio auxiliar que acompaña a FIRe y que deberá desplegarse única y exclusivamente cuando se configure este conector.
  + Por ejemplo, “https://demo.com/fnmt-fire-service/OauthHelper”.
* scopeAttrsManageAndUserList
  + *Scopes* de gestión y listado de atributos y usuarios (separados por un espacio).
  + Por defecto, debe utilizarse el valor “urn:fnmt:nif urn:safelayer:eidas:account:user:list”.
* scopeProfile
  + *Scope* de perfil.
  + Por defecto, debe utilizarse el valor “urn:safelayer:eidas:sign:identity:profile”.
* scopeACG
  + *Scope* para pedir la autorización ACG a TX para identidades cloudid.
  + Por defecto, debe utilizarse el valor “urn:safelayer:eidas:sign:identity:profile”.
* scopeACG\_QSCD
  + *Scope* para pedir la autorización ACG a TX para identidades QSCD.
  + Por defecto, debe utilizarse el valor “urn:safelayer:eidas:sign:identity:use:server:qscd”.
* scopeProfileAndServer
  + *Scopes* de perfil y servidor para la firma con identidades cloudid (separados por un espacio).
  + Por defecto, debe utilizarse el valor “urn:safelayer:eidas:sign:identity:profile urn:safelayer:eidas:sign:identity:use:server”.
* scopeProfileAndServer\_QSCD
  + *Scopes* de perfil y servidor para la firma con identidades QSCD (separados por un espacio).
  + Por defecto, debe utilizarse el valor “urn:safelayer:eidas:sign:identity:profile urn:safelayer:eidas:sign:identity:use:server:qscd”.

Un ejemplo de fichero de configuración puede ser el siguiente:

|  |
| --- |
| # CA's que en TrustedX no son QSCD, separadas por comas  # No es necesario el nombre completo, basta con una parte del mismo  caTxNoQSCD=AC Administrac  #AC Administración Pública  definedLabels=fnmt,cloudid,grupo2,x509:keyUsage:contentCommitment  #AC Sector Público  definedLabels\_QSCD=fnmt,cloudidqscd,grupo3,x509:keyUsage:contentCommitment  codeService=http://192.168.56.80:7011/fnmt-fire-service/OauthHelper  # Scopes  scopeAttrsManageAndUserList=urn:fnmt:nif urn:safelayer:eidas:account:user:list  scopeProfile=urn:safelayer:eidas:sign:identity:profile  # scope de ACG  #AC Administración Pública  scopeACG=urn:safelayer:eidas:sign:identity:profile  #AC Sector Público  scopeACG\_QSCD=urn:safelayer:eidas:sign:identity:use:server:qscd  #AC Administración Pública  scopeProfileAndServer=urn:safelayer:eidas:sign:identity:profile urn:safelayer:eidas:sign:identity:use:server  #AC Sector Público  scopeProfileAndServer\_QSCD=urn:safelayer:eidas:sign:identity:profile urn:safelayer:eidas:sign:identity:use:server:qscd |

#### Configuración adicional del conector de la FNMT

El conector de la FNMT hace uso de un par de ficheros de configuración adicionales. Estos ficheros están orientados a configurar algunas de las implementaciones internas que se utilizan para la compartición de datos entre el conector de CERES Cloud ID y el servicio auxiliar de la FNMT. Estos ficheros de configuración son asyncnotifier.properties y dtscacher.properties. Ambos se deberán situar junto al resto de ficheros de configuración de FIRe y tendrán una única propiedad definida:

* tmpdir
  + Ruta absoluta del directorio temporal en el que almacenar los ficheros de compartición de datos entre el servicio auxiliar de la FNMT y el conector de CERES Cloud ID.

### Fichero de apariencia de conector

Es posible personalizar la apariencia y los textos que muestra un proveedor de FIRe por defecto. Esto es común cuando se reaprovecha uno de los conectores que incorpora FIRe de base para acceder a un sistema de firma en la nube distinto del original o cuando se desea acceder a varios despliegues de un mismo sistema de firma en la nube utilizando un mismo conector. Por ejemplo, si quisiéramos configurar y tener habilitado simultáneamente el conector de CERES Cloud ID dos veces, una para el acceso al entorno de desarrollo y otra al despliegue en producción. Para que estos dos conectores no se presenten exactamente igual al usuario, podríamos definir un fichero de apariencia para uno de ellos.

Un fichero de apariencia de conector puede tener cualquier nombre y deberá situarse junto al resto de ficheros de configuración del componente central. Es posible indicar a FIRe que utilice un fichero de estilo distinto al original usando en el fichero config.properties la propiedad:

* provider.*PROVEEDOR*.info.file
  + La partícula PROVEEDOR será el nombre del proveedor para el que se desea cambiar la apariencia.
  + El valor de esta propiedad será el nombre del fichero que se debe cargar. Este fichero se buscará en el directorio de ficheros de configuración o, si no está definido, en el *classpath* de la aplicación.

Las propiedades que se deben establecer en el fichero de configuración de estilo son las siguientes:

* logo
  + Logo del proveedor.
  + Debe indicarse la ruta a un recurso interno con la imagen PNG o una URL a la propia imagen. La URL puede ser de tipo "http:", "https:" o "data:". Con este último tipo se podría empotrar la imagen en Base 64 dentro de la propiedad.
  + El tamaño recomendado de imagen es 319 x 109px.
* title
  + Título del proveedor. Se mostrará también como texto alternativo de la imagen.
* header
  + Cabecera a mostrar en la opción seleccionable del proveedor.
* description
  + Descripción a mostrar en la opción seleccionable del proveedor.
* noregistered
  + Texto a mostrar cuando se detecte que el usuario no está registrado en el proveedor.
* needjavascript
  + Indica si el proveedor necesita JavaScript para funcionar (true o false).
  + Si se necesita y se detecta que el navegador no lo tiene habilitado, se bloqueará el acceso a este proveedor.

Los textos pueden contener caracteres y elementos codificados en HTML, ya que se mostrarán directamente como parte de los proveedores seleccionables. En la siguiente imagen se pueden ver los elementos principales que se muestran por cada proveedor:



La imagen anterior se corresponde con el siguiente fichero de configuración:

|  |
| --- |
| # Logo del proveedor. Debe indicarse la ruta interna a una imagen PNG  # con el logo o una URL a la propia imagen. La URL puede ser de tipo  # "http:", "https:" o "data:". El tamano recomendado de imagen es 309 x 109px.  logo=es/gob/fire/server/connector/test/logo-clavefirma-test.png  # Titulo del proveedor  title=Simulador de Cl@ve Firma  # Cabecera a mostrar en la opcion seleccionable del proveedor  header=Firma con Cl@ve Firma  # Descripcion a mostrar en la opcion seleccionable del proveedor  description=Utilice un certificado de firma de Cl@ve o realice su solicitud si no dispone de &eacute;l.  # Texto a mostrar cuando se detecte que el usuario no esta registrado en el proveedor  noregistered=El usuario no est&aacute; dado de alta en Cl@ve, por lo que no tiene disponible esta opci&oacute;n.  # Indica si el proveedor necesita JavaScript para funcionar  needjavascript=false |

## Dependencias Externas

FIRe delega o permite delegar algunas operaciones en diversos sistemas externos. Por defecto, incluye la lógica para la conexión con estos sistemas, pero también permite a los desarrolladores implementar su propia lógica de conexión con sistemas equivalentes para permitir ampliar la funcionalidad de FIRe. Los tipos de sistemas con los que permite conectar son los siguientes:

* Proveedores de firma en la nube: Estos proveedores almacenan en la nube claves de firma de usuarios y sirven de pasarela para firmar con dichas claves. FIRe se distribuye con los conectores para el uso de los siguientes proveedores:
  + Cl@ve Firma.
  + CERES Cloud ID.
* Sistemas de validación y promoción de firmas: Son sistemas que permiten validar las firmas generadas por FIRe, independientemente del proveedor utilizado para firmar, y promocionar estas firmas a formatos longevos incorporándoles la información de validación (sellos de tiempo, información de revocación, etc.). FIRe se distribuye con el conector para el uso de los siguientes sistemas de validación y promoción de firmas:
  + @firma.
* Sistemas de gestión de alarmas: Son sistemas a los que FIRe puede notificar la detección de errores críticos, como la falta de un fichero de recursos, la pérdida de conexión con la base de datos o que no se puede acceder a algún sistema externo. FIRe se distribuye con el conector para el envío de errores a los siguientes sistemas de gestión de alarmas:
  + GrayLog.
* Orígenes de documentos y firmas: Pueden ser sistemas externos de los que FIRe pueda obtener los documentos a firmar y almacenar las firmas generadas. FIRe define el acceso a los mismos a través de lo que llama gestores de documentos. FIRe incorpora por defecto varios gestores de documentos, pero ninguno de ellos está orientado a acceder a sistemas externos, por lo que no requieren que se habiliten accesos de red específicos para los mismos.

Los administradores del sistema en el que se despliegue el componente central deberán habilitar el acceso de red a aquellos sistemas mencionados que se habiliten en FIRe, además de todos aquellos para los que se desarrollen conectores propios para dar servicio a las funciones de FIRe o a las aplicaciones.

A continuación, se detallan los sistemas a los que FIRe permite dar acceso con los conectores incorporados por defecto:

* Cl@ve Firma:
  + Es el componente desplegado en los entornos de la Gerencia Informática de la Seguridad Social que gestiona el acceso a los almacenes de claves del Cuerpo Nacional de Policía.
  + Se deberá dar acceso sólo si se configura en FIRe el proveedor de Cl@ve Firma.
  + La URL de acceso será proporcionada por la GISS una vez apruebe la conexión del organismo.
  + Adicionalmente, los ciudadanos pueden verse redirigidos al dominio “*clave-dninbpcert.policia.gob.es*” del CNP para realizar determinadas operaciones (generación de certificados, cambio de contraseñas…). Si los usuarios de su aplicación deben acceder a través de la red del organismo, asegúrese de que tienen acceso a este dominio.
* CERES Cloud ID:
  + Es el sistema de firma en la nube de la Fábrica Nacional de Moneda y Timbre, utilizado para permitir la firma con los certificados en la nube de funcionario de esta entidad.
  + Se deberá dar acceso sólo si se configura el proveedor CERES Cloud ID.
  + La URL de acceso será proporcionada por la FNMT una vez apruebe la conexión del organismo.
  + Durante el proceso de firma con este proveedor, los usuarios serán redirigidos al mismo dominio en el que se encuentra el servicio de firma proporcionado por la FNMT. Si los usuarios de su aplicación deben acceder a través de la red del organismo, asegúrese de que tienen acceso a ese dominio.
* @firma:
  + Se utiliza para la mejora de las firmas generadas (creación de firmas longevas).
  + El organismo deberá tener acceso a un despliegue de este sistema para hacer uso del mismo. No hay un despliegue único al que cualquiera pueda tener acceso.
  + El acceso a esta plataforma es opcional. Sólo se accederá a ella si se solicita la creación de firmas longevas desde la aplicación.
  + La URL y el resto de datos de acceso serán proporcionados por el responsable del despliegue de @firma que se desee utilizar.
* GrayLog
  + Es un sistema para la gestión de logs al que se enviaran los errores críticos detectados durante la ejecución de FIRe.
  + El organismo deberá proporcionar su propia instancia de GrayLog a través de la cual quiera gestionar las alarmas de la aplicación. En FIRe se deberá configurar el acceso al *stream* al que enviar los avisos.
  + El uso de este sistema es opcional. Ya se configure o no, FIRe almacenará las trazas de cualquier error en sus ficheros de log.
  + La URL y los datos de configuración para el envío de alertas deberá proporcionarlos el responsable del despliegue de GrayLog al que se deseen enviar.

## Base de datos

FIRe utiliza base de datos para la gestión de las aplicaciones autorizadas para el uso de FIRe, la gestión de usuarios, el almacenamiento de estadísticas y el acceso a sistemas de consulta de logs. Queda a elección del integrador crear una nueva base de datos para el sistema o utilizar una ya existente.

El Sistema de Gestión de Base de Datos queda a elección del integrador, pero se debe tener encuentra que el componente central deberá poder acceder al mismo a través de un controlador JDBC. Tanto el driver JDBC a utilizar como la cadena de conexión deben quedar reflejadas en el fichero de configuración del componente central.

### Creación de la Base de datos

Para la creación de las tablas de la base de datos se proporcionan scripts orientados a su uso con los SGBD MySQL y ORACLE. No obstante, el administrador de la base de datos deberá comprobar que estas no entran en conflicto con otras tablas ya creadas o con la configuración concreta de su base de datos. También podrá el administrador crear sus propias sentencias para la creación de las tablas si dispone de algún otro SGBD en el que estos scripts no funcionen directamente.

### Modelo de datos

A continuación, se proporcionan las definiciones de las distintas tablas del modelo de datos de FIRe:

|  |
| --- |
| **Tabla de aplicaciones** |
| Nombre de la tabla: tb\_aplicaciones  Campos:   * Nombre=id, Tipo=cadena 48 caracteres, NO NULO, Clave Primaria. * Nombre=nombre, Tipo=cadena 45 caracteres, NO NULO. * Nombre=fecha\_alta, Tipo=fecha y hora, NO NULO. * Nombre= fk\_certificado, Tipo= entero 11 cifras, NO NULO, Clave Ajena con la tabla tb\_certificados. * Nombre=habilitado, Tipo=booleano, Valor por defecto= true |

|  |
| --- |
| **Tabla de certificados** |
| Nombre de la tabla: tb\_certificados  Campos:   * Nombre= id\_certificado, Tipo= entero 11 cifras auto-incremental, Clave Primaria. * Nombre= nombre\_cert, Tipo= cadena 45 caracteres, NO NULO. * Nombre= fec\_alta, Tipo= fecha y hora, Valor por defecto= fecha Actual, NO NULO. * Nombre= cert\_principal, Tipo= cadena 5000 caracteres. * Nombre= cert\_backup, Tipo= cadena 5000 caracteres. * Nombre= huella\_principal, Tipo= cadena 45 caracteres. * Nombre= huella\_backup, Tipo= cadena 45 caracteres. |

|  |
| --- |
| **Tabla de usuarios** |
| Tabla: tb\_usuarios  Campos:   * Nombre= id\_usuario, Tipo= entero 11 cifras auto-incremental, Clave Primaria. * Nombre= nombre\_usuario, Tipo= cadena 30 caracteres, NO NULO. * Nombre= clave, Tipo= cadena 2000 caracteres, NO NULO. Clave única. * Nombre= nombre, Tipo= cadena 45 caracteres, NO NULO. * Nombre= apellidos, Tipo= cadena 120 caracteres, NO NULO. * Nombre= correo\_elec, Tipo= cadena 45 caracteres, Valor por defecto= NULO * Nombre= telf\_contacto, Tipo= cadena 45 caracteres. * Nombre= fk\_rol, Tipo= entero 11 cifras, NO NULO. Clave ajena. * Nombre= fec\_alta, Tipo= fecha y hora, Valor por defecto= fecha Actual, NO NULO. * Nombre= usu\_defecto, Tipo= booleano, Valor por defecto= false * Nombre= cod\_renovacion, Tipo= cadena 100 caracteres. * Nombre= fec\_renovacion, Tipo= fecha y hora. * Nombre= rest\_clave, Tipo= booleano, Valor por defecto= false |

|  |
| --- |
| **Tabla de roles** |
| Tabla: tb\_roles  Campos:   * Nombre= id, Tipo= entero 11 cifras auto-incremental, Clave Primaria. * Nombre= nombre\_rol, Tipo= cadena 45, NO NULO. Clave única. * Nombre= permisos, Tipo= cadena 45 caracteres. |

|  |
| --- |
| **Tabla de relación de Responsables de aplicaciones** |
| Tabla: tb\_responsable\_de\_aplicaciones  Campos:   * Nombre= id\_responsables, Tipo= entero 11 cifras, Clave Primaria. * Nombre= id\_aplicaciones Tipo= cadena 48, Clave Primaria. |

|  |
| --- |
| **Tabla de servidores de logs** |
| Tabla: tb\_servidores\_log  Campos:   * Nombre= id\_servidor, Tipo= entero 11 cifras auto-incremental, Clave Primaria. * Nombre= nombre Tipo= cadena 45, NO NULO. Clave única * Nombre= url\_servicio\_log, Tipo= cadena 500 caracteres, NO NULO. Clave única. * Nombre= clave, Tipo= cadena 45 caracteres, NO NULO. * Nombre= verificar\_ssl, Tipo= booleano, Valor por defecto= true |

|  |
| --- |
| **Tabla de información estadística de firma** |
| Tabla: tb\_firmas  Campos:   * Nombre= id, Tipo= entero 11 cifras auto-incremental, Clave Primaria. * Nombre= fecha, Tipo= fecha y hora, Valor por defecto= fecha Actual, NO NULO. * Nombre= aplicacion Tipo= cadena 45, NO NULO. * Nombre= formato Tipo= cadena 20, NO NULO. * Nombre= formato\_mejorado Tipo= cadena 20, Valor por defecto= NULO. * Nombre= algoritmo Tipo= cadena 20, NO NULO. * Nombre= proveedor Tipo= cadena 45, NO NULO. * Nombre= navegador Tipo= cadena 20, NO NULO. * Nombre= correcta, Tipo= booleano, Valor por defecto= false * Nombre= total, Tipo= entero 11 cifras, NO NULO. |

|  |
| --- |
| **Tabla de información estadística de transacciones** |
| Tabla: tb\_transacciones  Campos:   * Nombre= id, Tipo= entero 11 cifras auto-incremental, Clave Primaria. * Nombre= fecha, Tipo= fecha y hora, Valor por defecto= fecha Actual, NO NULO. * Nombre= aplicación, Tipo= cadena 45, NO NULO. * Nombre= operación, Tipo= cadena 10, NO NULO. * Nombre= proveedor, Tipo= cadena 45, NO NULO. * Nombre= proveedor\_forzado, Tipo= booleano, Valor por defecto= false * Nombre= correcta, Tipo= booleano, Valor por defecto= false * Nombre= tamanno, Tipo= entero 11 cifras, Valor por defecto= 0. * Nombre= total, Tipo= entero 11 cifras, Valor por defecto= 0. |

Opcionalmente, pueden crearse también en el esquema las siguientes tablas para dar soporte a la funcionalidad de compartición de sesiones y documentos entre nodos por medio de base de datos. Los valores para la configuración de este comportamiento se describen en las propiedades “sessions.dao” y “sessions.documents.dao” descritas en el apartado “4.4.1 Fichero config.properties”.

|  |
| --- |
| **Tabla de sesiones compartidas** |
| Tabla: tb\_sesiones  Campos:   * Nombre= id, Tipo= cadena 64 caracteres, Clave Primaria. * Nombre= f\_modificacion, Tipo= fecha y hora, NO NULO. * Nombre= sesion, Tipo= BLOB. |

|  |
| --- |
| **Tabla de documentos compartidos** |
| Tabla: tb\_documentos  Campos:   * Nombre= id, Tipo= cadena 64 caracteres, Clave Primaria. * Nombre= f\_modificacion, Tipo= fecha y hora, NO NULO. * Nombre= datos, Tipo= BLOB. |

Las tablas de la base de datos estarán inicialmente vacías a excepción del listado de roles y de los datos del usuario administrador. Los datos que se deben incluir para esto son:

|  |
| --- |
| **Listado de roles** |
| Tabla=tb\_roles  Registros:   * id= 1, nombre\_rol= 'admin', permisos= '1,2' * id= 2, nombre\_rol= 'responsable', permisos= '2' * id= 3, nombre\_rol= 'contact', permisos= NULL |

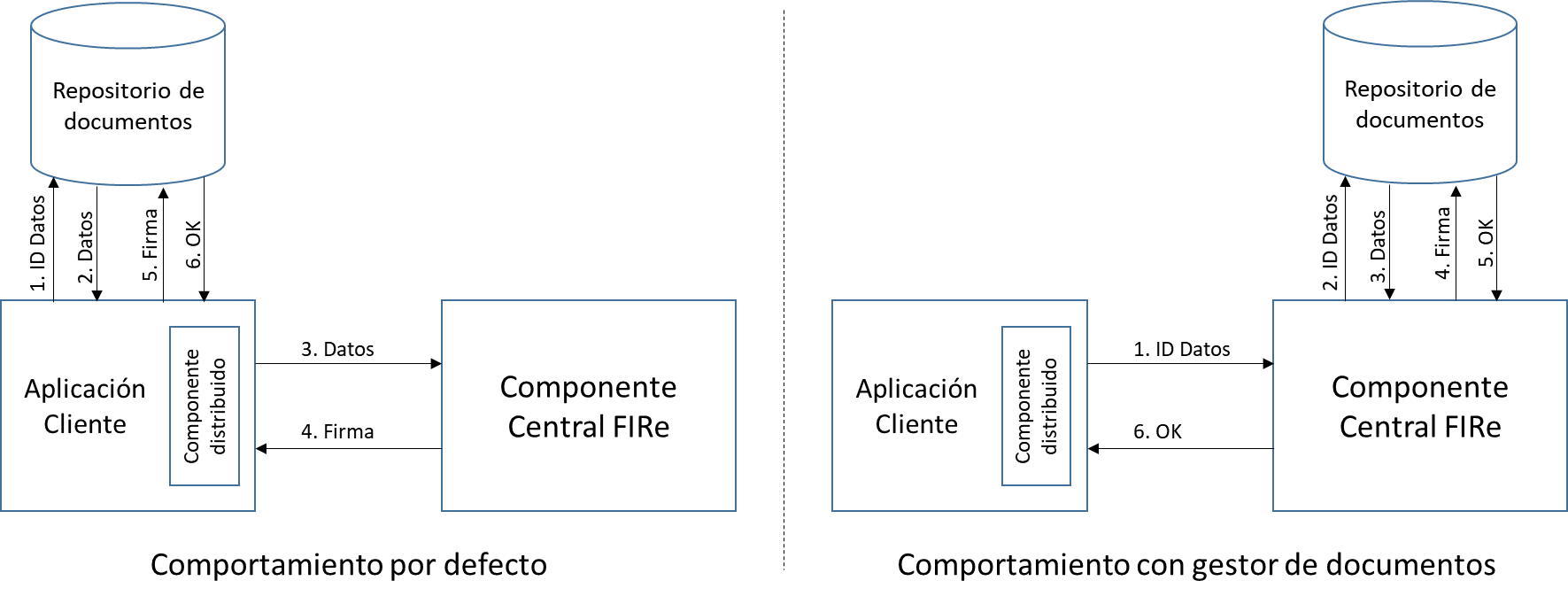
|  |
| --- |
| **Usuario administrador** |
| Tabla=tb\_usuarios  Registro:   * nombre\_usuario= 'admin', clave= '$2y$12$JfP4bTV0i29Mnb3XBPOQl.L8JdbTrpn4fQljv8EEJKIp6NRZLB5TC', nombre= 'default name', apellidos= 'default surnames', fk\_rol= 1, usu\_defecto= 1 |

El alta de los nuevos usuarios, certificados y aplicaciones se realizará a través del componente de administración. Consulte el manual de administración de FIRe para saber cómo acceder al módulo de administración, dar de alta nuevas aplicaciones y cambiar la contraseña del administrador.

## Clases gestoras de documentos

El funcionamiento por defecto de FIRe requiere que la aplicación cliente cargue los datos que desea firmar, los envíe al componente central para solicitar su firma y, posteriormente, recupere la firma electrónica generada para almacenarla o tratarla como corresponda. Sin embargo, FIRe también soporta el escenario en el que una aplicación cliente le indica al componente central qué datos son los que se deben firmar y es el propio componente central el que carga los datos y trata y guarda la firma para, finalmente, devolver un resultado a la aplicación que le indique como ha terminado el proceso.

A continuación, se muestran dos diagramas simplificados en los que se muestra el envío de datos en cada uno de los escenarios presentados:



Como diferencias clave entre el escenario por defecto y el que utiliza una clase gestora de documentos, el uso de la clase gestora conlleva las siguientes ventajas y desventajas:

* Ventajas:
  + Se reduce significativamente el tráfico de datos entre las aplicaciones cliente y el componente central ya que, aunque se producen el mismo número de conexiones, los datos y la firma no viajan entre la aplicación cliente y el componente central de FIRe.
  + Cuando existen varias aplicaciones que realizan el mismo tratamiento de datos antes y después de firmar, se puede implementar este comportamiento una única vez en el componente central en lugar de hacerlo en cada una de las aplicaciones.
* Desventajas:
  + Se traslada la carga de recuperar los datos y almacenar la firma al servidor en donde esté desplegado FIRe, lo cual puede afectar a su rendimiento.

Para el uso del nuevo escenario es necesario implementar una “clase gestora de documentos”. Esta clase será la que defina de dónde se deben obtener los datos indicados por la aplicación cliente y cómo tratar y almacenar la firma resultante.

Al implementar una clase gestora también se puede optar por definir un modelo mixto entre los dos presentados, en el que:

* Se espere que la aplicación envíe los datos al componente central, pero sea la clase gestora la que se encargue de guardar la firma. Este modelo es útil cuando los datos están por defecto en la aplicación cliente. Por ejemplo, cuando se trata de un formulario relleno por el usuario o cuando se quiera firmar un documento que la aplicación cliente ya tiene cargado porque haberlo usado con otra finalidad.
* Se desee recuperar los datos a través de la clase gestora, pero la firma deba devolverse a la aplicación cliente. Esto es útil cuando se desee procesar la firma de alguna manera antes de guardarla y no es posible hacerlo desde el componente central.

### Desarrollo de una clase gestora de documentos

El desarrollo de la clase gestora de documentos deberá realizarlo el desarrollador de la aplicación cliente, ya que es el que conoce la lógica intrínseca de su aplicación, los sistemas en la que se encuentran los datos y dónde se almacenan las firmas. Esta lógica, sin embargo, no se ejecutará desde la aplicación cliente sino desde el componente central, para lo cual el administrador del sistema FIRe deberá desplegar y configurar esta clase de tal forma que el componente central tenga acceso a ella.

Si un integrador desease conectarse con FIRe y utilizar una clase gestora de documentos, debería, primeramente, solicitar el permiso del administrador de FIRe, ya que este es el que debe habilitar el uso de esa clase gestora.

Antes de dar su visto bueno, el administrador deberá valorar:

* El beneficio que obtendrá la aplicación o aplicaciones cliente frente a la carga adicional que recibirá el componente central por llevar a cabo esta tarea.
* La confianza que se tiene en el integrador y el desarrollo que haga de la clase gestora. Hay que tener en cuenta que este desarrollo se ejecutará en el servidor del componente central, con los mismos permisos que tenga el usuario con el que se ejecute el servidor de aplicaciones, y tendrá acceso a sus recursos.
* Que el desarrollo realizado no incluya dependencias incompatibles con las de FIRe, el servidor de aplicaciones o alguna otra clase gestora, ya que estas deberán desplegarse también como parte del componente central.
* La posibilidad y conveniencia de dar acceso a los recursos de red que necesite la clase gestora para la carga de los datos y el tratamiento y guardado de la firma.

En caso de dar el visto bueno al uso de la clase gestora, el integrador deberá proporcionar al administrador esta clase (empaquetada en forma de JAR), el nombre completo de la clase, las bibliotecas de las que dependa, el listado exacto de recursos y entornos a los que se deberá tener acceso (para habilitar los permisos del sistema y permisos de red necesarios) y, si procede, el fichero de configuración de la clase y las instrucciones para establecer sus valores.

El detalle de la implementación de nuevas clases gestoras de documentos se puede encontrar en el manual del integrador de FIRe.

### Alta de la clase gestora

Para dar de alta la clase gestora, el administrador deberá:

1. Introducir el JAR de la clase gestora y sus dependencias dentro del directorio de bibliotecas del WAR del componente central (fire-signature.war\WEB-INF\lib).
2. Agregar al fichero de configuración config.properties una entrada que relacione el nuevo gestor de documentos con la clase que lo implementa. La forma de esta entrada deberá ser:
   * docmanager.NOMBRE=NOMBRE\_CLASE

Donde:

* + NOMBRE: Es el nombre que queramos darle al gestor para identificarlo claramente y diferenciarla de cualquier otro que demos de alta.
  + NOMBRE\_CLASE: Nombre completo de la clase gestora de documentos (incluyendo el paquete) proporcionada por el integrador.

Por ejemplo:

* + docmanager.bbddMinisterio=es.minhap.ddbb.MinisterioDocManager

1. Si se ha proporcionado un fichero de configuración, se deberá copiar al directorio en el que se encuentra el fichero de configuración del componente central (config.properties) y renombrarlo según el patrón:
   * docmanager.NOMBRE.properties

Donde:

* + NOMBRE: Es el nombre que le hemos dado al gestor en el paso anterior para identificarlo claramente y diferenciarlo de cualquier otro que demos de alta.

Por ejemplo:

* + docmanager.bbddMinisterio.properties

1. Si el integrador ha proporcionado instrucciones para configurar el fichero de configuración, seguir dichas instrucciones.
2. Habilitar el acceso a los recursos y entornos de red necesarios para el funcionamiento de la clase lectora indicados por el integrador.
3. Restringir, si procede, el acceso a la clase gestora por parte de las aplicaciones. Una clase gestora da acceso a los documentos de una aplicación y puede que estos no deban ser accesibles desde todas las aplicaciones que utilicen nuestra instancia de FIRe. Es posible restringir qué aplicaciones son las que pueden utilizar este gestor de documentos configurando la propiedad “docmanager.*NOMBRE*.requestor.valid” del fichero “config.properties” o indicar que aplicaciones son las que no pueden usarlo mediante la propiedad “docmanager.*NOMBRE*.requestor.invalid” del mismo fichero. Puede consultar el detalle de estas propiedades el apartado 4.3.1 Fichero config.properties. Por ejemplo:
   * docmanager.bbddMinisterio.requestor.valid=2D171F3CC342,64FBE9226FF6
4. Proporcionar al integrador el nombre que hemos designado para referenciar al gestor de documentos. Por ejemplo:
   * bbddMinisterio

## Clases de notificación de errores críticos

FIRe puede enviar automáticamente una notificación cuando se produce algún error crítico en la aplicación. La lógica para la notificación de estos errores se define a través de una clase de una clase Java y el administrador del sistema puede configurar en el fichero “config.properties” qué clase debe usarse para el envío de las notificaciones. Si no se configura ninguna clase, los errores no se notificarán de una forma especial y únicamente aparecerán junto a las trazas de la aplicación.

FIRe incluye una implementación de clase notificadora que permite el envío de los errores críticos a un servidor GrayLog, pero un desarrollador puede implementar una clase que notifique los errores de forma distinta. Consulte los apartados “4.3.1 Fichero config.properties” (propiedad “alarms.notifier”) y “4.3.3 Fichero alarms\_config.properties” para saber más de como configurar el envío de notificaciones a GrayLog.

El listado de errores que se puede notificar está ya predefinido y no es posible agregar otro nuevos.

### Desarrollo de una clase de notificación

La lógica para la notificación de los errores en FIRe se define a través de una clase que implementa la interfaz “es.gob.fire.alarms.AlarmNotifier”, localizada en la biblioteca “fire-alarms.jar”. Un desarrollador puede crear una clase Java que implemente esta interfaz y definir así nuevos modos de enviar o procesar las notificaciones de FIRe. Por ejemplo, envío de un correo electrónico al administrador, guardado de los registros de error en fichero, etc.

Para crear una nueva clase notificadora, puede crearse un proyecto Java que importe la biblioteca “fire-alarms.jar”. Si se crea el proyecto con Maven, puede importarse esta clase usando la dependencia:

|  |
| --- |
| <dependency>  <groupId>es.gob.fire</groupId>  <artifactId>fire-alarms</artifactId>  <version>NUMERO\_VERSION\_FIRE</version>  </dependency> |

La interfaz AlarmNotifier define los siguientes métodos que será necesario implementar:

* void init(Properties config) throws InitializationException;
  + Método para la inicialización de la clase gestora. Se ejecutará una única vez antes de enviar por primera vez una notificación de error.
  + Recibe como parámetro las propiedades extraídas del fichero de configuración “alarms\_config.properties”.
* void setModule(String module)
  + Método que informa del módulo desde el que se realizarán las notificaciones. Se ejecutará una vez tras la inicialización de la clase.
  + Los parámetros recibidos en este método serán:
    - module: Nombre del módulo que genera el error.
  + Actualmente, la clase notificadora sólo se utiliza desde el componente central de FIRe y ésta siempre llamará a este método proporcionando como parámetro el valor “MODCENTRAL”.
* void notify(AlarmLevel level, Alarm alarm, String… source) throws IOException;
  + Método para la notificación de un error en el sistema. Este método puede enviar/registrar la información proporcionada por parámetro, así como cualquier otra información obtenida dinámicamente del entorno o del fichero de configuración.
  + Los parámetros recibidos en este método serán:
    - level: Nivel de alama. Este parámetro se establece mediante un enumerado. Los valores disponibles son:
      * DEBUG: Nivel de depuración.
      * INFO: Nivel informativo.,
      * WARNING: Nivel de advertencia.
      * ERROR: Nivel de error.
      * CRITICAL: Nivel de fallo crítico.
    - alarm: Tipo de error que se ha producido. Este parámetro puede adoptar los valores:
      * RESOURCE\_NOT\_FOUND: No se encuentra un fichero de configuración. En el parámetro "source" se indicará el nombre del fichero.
      * RESOURCE\_CONFIG: Se ha detectado un error en una propiedad de un fichero de configuración. En el parámetro "source" se indicarán el nombre de la propiedad y el del fichero.
      * LIBRARY\_NOT\_FOUND: Biblioteca o dependencia no encontrada. En el parámetro "source" se indicará el nombre de la clase que no se ha podido cargar.
      * CONNECTION\_DB: No se puede conectar con la base de datos.
      * CONNECTION\_VALIDATION\_PLATFORM: No se puede conectar con la plataforma de validación de firmas.
      * CONNECTION\_SIGNATURE\_PROVIDER: No se puede conectar con un proveedor de firma en la nube. En el parámetro "source" se indicará el nombre asignado al proveedor.
      * CONNECTION\_DOCUMENT\_MANAGER: No se puede conectar con un gestor de documentos. En el parámetro "source" se indicará el nombre de la clase de conexión con el gestor de documentos en cuestión.
    - source: Fuente del error. El valor que aquí se proporcione depende del tipo de error declarado en el parámetro “alarm”. Por ejemplo, en el caso del no poder acceder a un fichero, será el nombre del fichero; si hay una propiedad mal configurada, se indicará el nombre de la propiedad y el nombre del fichero en el que se encuentra; si ocurre un error de acceso a base de datos, será nulo; etc.
  + En caso de producirse un error durante el envío o registro de la notificación, se lanzará una excepción de tipo java.lang.IOException.

La implementación de la clase de envío de notificaciones a GrayLog puede servir como inspiración para el desarrollo de cualquier clase similar. Puede encontrarse esta en el módulo “fire-alarms-graylog”.

La clase de notificación puede hacer uso de otras clases y recursos que se distribuyan junto a ella. También puede hacer uso de bibliotecas externas. Sin embargo, en este último caso, habrá que consultar con el administrador del despliegue de FIRe que las bibliotecas que deseamos utilizar no introducen incompatibilidades con FIRe, las bibliotecas del servidor de aplicaciones en el que se despliega o las bibliotecas que acompañen a de alguna otra clase importada.

Tenga en cuenta que, si desde su clase de notificación accede a sistemas externos al componente central, deberá notificarlo al administrador de FIRe para que disponga el acceso a los servicios y recursos necesarios.

### Alta de la clase de notificación

Para poder utilizar una nueva clase de notificación de errores, el administrador deberá:

1. Introducir el JAR en el que se encuentre la nueva clase y todas las dependencias de la misma dentro del directorio de bibliotecas del WAR del componente central (fire-signature.war\WEB-INF\lib).
2. Actualizar el fichero de configuración config.properties para establecer en la propiedad “alarms.notifier” el nombre completo de la nueva clase. Por ejemplo:
   * alarms.notifier=es.gob.minhap.fire.MiClaseNotifier
3. Si el responsable de la clase ha proporcionado una serie de propiedades de configuración, establecer los valores de éstas en el fichero alarms\_config.properties.
4. Realizar cualquier otra actuación para la configuración de la clase que haya definido el desarrollador.
5. Reiniciar el servidor de aplicaciones.

# Componente Distribuido

El componente distribuido consiste en una biblioteca que permite acceder a las funcionalidades proporcionadas por el componente central de FIRe.

Este componente se integra en las aplicaciones cliente que desean utilizar los servicios del componente central de FIRe. Si no se es responsable de las aplicaciones que van a hacer uso de los servicios de FIRe, no es necesario revisar este apartado.

Se distribuyen tres implementaciones diferentes del componente distribuido para facilitar su integración en las distintas aplicaciones cliente: Java, .NET y PHP.

## Java

Se requiere un entorno de ejecución de Java 1.6 o superior, en 32 o 64 bits.

La versión Java del componente distribuido se distribuye en forma de JAR firmado digitalmente y acompañado de otras bibliotecas con las que mantiene dependencias.

Las bibliotecas son:

* Componente distribuido Java:
  + fire-client-2.4.jar
* Dependencias:
  + javax.json-api-1.0.jar
  + javax.json-1.0.4.jar
  + slf4j-api.1.7.25.jar

El primer paso para el uso del componente distribuido es agregar estas bibliotecas como dependencias de la aplicación cliente. Puede hacerlo agregando las bibliotecas directamente al *classpath* de la aplicación o mediante algún gestor de dependencias, como Apache Maven.

Para agregar el paquete como dependencia de Maven, utilice la siguiente referencia:

|  |
| --- |
| <dependency>  <groupId>es.gob.fire</groupId>  <artifactId>fire-client</artifactId>  <version>2.4</version>  </dependency> |

El componente distribuido permite que se le proporcione la configuración durante la construcción de la clase principal o mediante un fichero de propiedades. En caso de usarse un fichero de propiedades, este deberá llamarse client\_config.properties y situarse en el directorio que se haya proporcionado al servidor de aplicaciones mediante la variable de entorno “fire.config.path” o, si no se ha configurado, en el *classpath* del propio servidor de aplicaciones.

Ya sea que se le pasen directamente al API las opciones de configuración o se establezcan a través del fichero de propiedades, las propiedades que permite configurar el API son las siguientes:

* fireUrl
  + URL del servicio del componente central.
* javax.net.ssl.keyStore (Opcional)
  + Ruta del almacén de claves para la autenticación mediante certificado con el componente central. Este certificado debe estar dado de alta en la base de datos del componente central, asignado al identificador de la aplicación cliente en la que se esté integrando el componente distribuido.
  + Si se omite este parámetro se usará la configuración establecida a nivel global en la JRE.
* javax.net.ssl.keyStorePassword (Opcional)
  + Contraseña del almacén de claves de autenticación SSL.
* javax.net.ssl.keyStoreType (Opcional)
  + Tipo del almacén de claves del certificado de autenticación SSL: “JKS” (almacén de Java) o “PKCS12” (almacén PKCS12/PFX).
  + Por defecto, se considera que el almacén es de tipo JKS.
* javax.net.ssl.trustStore (Opcional)
  + Ruta del almacén de certificados de confianza SSL. Esto se usa cuando el certificado con el que se ha montado el SSL del componente central no está en el almacén de confianza de Java y se desea establecer un almacén de confianza alternativo.
  + En caso de querer desactivar la comprobación del certificado SSL del servidor, se puede configurar el valor “all”.
  + Si se omite este parámetro se usará la configuración establecida a nivel global en la JRE. Por defecto, esto es que se confiará en los certificados dados de alta en el almacén “cacerts”.
* javax.net.ssl.certAlias (Opcional)
  + Alias del certificado del almacén de claves que se debe utilizar para la autenticación de la aplicación.
  + Si se omite, se utilizará el primer certificado encontrado en el almacén.
* javax.net.ssl.trustStorePassword (Opcional)
  + Contraseña del almacén de confianza.
* javax.net.ssl.trustStoreType (Opcional)
  + Tipo del almacén de confianza: “JKS” (almacén de Java) o “PKCS12” (almacén PKCS12/PFX).
  + Por defecto, se considera que el almacén es de tipo JKS.

Ejemplo de objeto de configuración o contenido de client\_config.properties:

fireUrl=https://localhost:8443/fire-signature/fireService

javax.net.ssl.keyStore=app\_fire.jks

javax.net.ssl.keyStorePassword=11111111

javax.net.ssl.keyStoreType=JKS

Consulte la documentación de su servidor de aplicaciones o contenedor de aplicaciones Web Java para conocer como proporcionarle propiedades o como introducir ficheros de propiedades dentro del *classpath* de las aplicaciones.

Consulte el manual de integración de FIRe para más información sobre cómo hacer uso de FIRe desde sus aplicaciones.

### Trazas de registro del Componente Distribuido Java

El componente distribuido Java utiliza SLF4J como biblioteca para la generación de trazas de registro (*logs*). Esta biblioteca permite que el integrador enlace las trazas del componente distribuido de FIRe con los del resto de su aplicación por medio de la “biblioteca puente” correspondiente a su sistema de logs (Log4J, Log4J 2, Apache Logging API…). Para más información, consulte el manual del integrador de FIRe.

Todos los logs del componente distribuido Java de FIRe se imprimen con el nombre de la clase que los genera. Así, pueden recogerse todos utilizando el nombre “es.gob.fire.client”.

## .Net

Se requiere un entorno de ejecución de Microsoft .NET versión 4.5 o superior.

La versión .NET del componente distribuido se encuentra en formato de biblioteca de enlace dinámico (DLL) de .NET (construida con C#). Esta biblioteca se llama FIRe.dll.

Para el uso de la biblioteca se necesita que esta se encuentre en uno de los directorios de carga de bibliotecas de la aplicación que la utilice, que típicamente son:

1. El mismo directorio que la aplicación (EXE, DLL, etc.) que use el componente distribuido.
2. Un directorio dentro del PATH del sistema.
3. El directorio de bibliotecas del sistema, normalmente %SystemRoot%/System32

Consulte la documentación de su entorno de desarrollo para saber cómo usar durante el desarrollo de su aplicación.

En Microsoft Visual Studio, por ejemplo, necesitará importar la referencia a la biblioteca. Puede hacerlo tal como se indica en el siguiente enlace:

* https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2008/7314433t(v=vs.90)

La configuración del cliente distribuido se puede realizar durante su construcción, mediante un diccionario con las distintas opciones, o a través de la clave “HKEY\_CURRENT\_USER\Software\FIRe” del registro de Windows.

Ya se use un medio u otro, las opciones de configuración son las siguientes:

* fire\_service
  + URL del servicio de FIRe.
* ssl\_client\_pkcs12
  + Ruta absoluta del almacén PKCS#12/PFX con las clave y certificado para la autenticación cliente SSL contra el componente central.
* ssl\_client\_pass
  + Contraseña del almacén PKCS#12/PFX para la autenticación cliente SSL contra el componente central.
* ssl\_client\_alias (Opcional)
  + Alias del certificado para la autenticación cliente SSL contra el componente central del almacén PKCS#12/PFX seleccionado.
  + Si no se indica, se tomará el primer certificado que se encuentre.
* admit\_all\_certs (Opcional)
  + Al indicar el valor “true”, se configura que se debe obviar la autenticación del certificado SSL. Si se indica “false”, se establece que debe comprobarse la validez y confianza del certificado (la fecha de caducidad, que se encuentre en el almacén de confianza de Windows, etc.). Por defecto, sí se realizarán las comprobaciones de seguridad.

Para agregar estos valores al registro de Windows, debe crear valores de tipo cadena con el nombre de la opción y el valor que se le desee asignar. Esto puede hacer directamente editando el registro o por medio de un fichero REG. A continuación, se muestra un ejemplo:

[HKEY\_CURRENT\_USER\Software\FIRe]

"fire\_service"="https://servidor.com/fire-signature/fireService"

"ssl\_client\_pkcs12"="C:/users/usuario/Documents/client\_ssl.pfx"

"ssl\_client\_pass"="12345678"

"admit\_all\_certs"="true"

En este ejemplo, se configura la URL de acceso del servicio localizado en el dominio de ejemplo “servidor.com”. Además, las peticiones al servicio se autenticarán con el certificado cliente del almacén PKCS#12 “client\_ssl.pfx”, con contraseña “12345678”. No se comprobará la validez y confianza del certificado SSL del servidor (admit\_all\_certs=true).

Dado que la configuración del registro se almacena en la rama de usuario del registro, la importación debe hacerse con el mismo usuario que se utilice para ejecutar la aplicación que haga uso de la DLL del componente distribuido.

## PHP

Para el uso del componente distribuido PHP se requiere un entorno de ejecución compatible con PHP 5 o superior y la extensión “php\_curl” habilitada. Si se va a utilizar la función auxiliar parse\_certificate(), también será necesario activar la extensión “php\_openssl”.

Este componente se presenta como un único fichero PHP llamado “fire\_client.php”. Su integración requiere únicamente incluir en su página PHP una referencia al mismo. Esto se puede hacer con el comando include.

Por ejemplo:

<html>

<head>

<title>Ejemplo de importacion</title>

</head>

<body>

<?php

**include** 'fire\_client.php';

…

Dentro de “fire\_client.php” se incluyen las variables de configuración con la URL del servicio del componente central y la lógica de comunicación necesaria para conectar con él. Para configurar el acceso al componente central, deberemos modificar el fichero “fire\_client.php”

La variable para la configuración de las URL de los servicios del componente central es:

* SERVICEURL
  + URL del servicio del componente central.

Un ejemplo de configuración de esta variable es:

<?php

// Definimos las URL con las que invocar cada servicio

**define ("SERVICEURL","https://servidor.com/fire-signature/fireService");**

…

En este ejemplo, se configura la URL de acceso del servicio localizado en el dominio “servidor.com”.

También será necesario configurar la conexión SSL cliente contra el componente centralizado a través de cURL. Para esto, estableceremos las propiedades necesarias de cURL en el listado “$client\_ssl\_curl\_options” que se encuentra al principio del fichero.

La configuración que se proporciona como ejemplo en el fichero es la siguiente:

**$client\_ssl\_curl\_options = array(**

**CURLOPT\_SSLCERT => "ssl\_client\_cert.pem",** // Ruta certificado SSL cliente

**CURLOPT\_SSLCERTTYPE => "PEM",** // Formato del certificado

**CURLOPT\_SSLKEY => "ssl\_client\_key.pem",** // Ruta clave del certificado

**CURLOPT\_SSLKEYTYPE => "PEM",** // Formato de clave privada

**CURLOPT\_SSLKEYPASSWD => "password",** // Contraseña de la clave privada

**CURLOPT\_SSL\_VERIFYPEER => 0** // Verificar certificado SSL del servidor

**);**

Se puede consular la información completa sobre estos y otros parámetros de configuración de cURL en la web: <https://curl.haxx.se/libcurl/c/easy_setopt_options.html>

# Componente de Administración

El módulo de administración de FIRe permite gestionar qué aplicaciones cliente tienen permiso para acceder a él y cuáles son sus certificados de acceso, generar informes estadísticos, consultar los logs del componente central y gestionar usuarios y roles. Para el uso del módulo de administración es obligatorio que este módulo y el componente central utilicen el mismo esquema de base de datos.

El componente de administración parte con un único usuario administrador:

* Usuario: **admin**
* Contraseña: **1111**

El administrador de FIRe debería acceder al módulo de administración, cambiar esta contraseña por defecto y, opcionalmente, dar de alta otros usuarios para el resto de administradores si los hubiera.

Si no se modifica el contexto de despliegue del módulo de administración, se debería poder acceder al mismo a través de una URL del tipo:

* https://DOMINIO/fire-admin-web/

Para saber más de las opciones de administración de FIRe, consulte el manual del módulo de administración.

## Despliegue

Para poder desplegar el módulo de administración hay que realizar los siguientes pasos:

1. Configuración de variables de arranque del servidor. Será necesario definir las siguientes variables que permitan establecer la configuración del entorno:
   1. server.config.dir: El valor de esta variable, será la ruta absoluta hacia la carpeta donde ubicaremos los ficheros de configuración del componente de administración.
   2. spring.config.location: El valor de esta variable, será la ruta hacia el fichero de configuración de Spring, ya que el componente de administración hace uso de esta tecnología. Este fichero suele recibir el nombre “application.properties”.
   3. log4j.configuration: El valor de esta variable, será la ruta hacia el fichero de configuración de “Log4j” para la gestión de los logs de la herramienta de administración. Este fichero recibe el nombre “fire-log4j.xml”.
   4. file.encoding: El valor de esta variable se refiere a la codificación de caracteres que utiliza el sistema. Su valor será siempre: UTF-8
2. Será necesario copiar una serie de recursos a la carpeta de configuración definida en punto 2.1. de este apartado.
   * Carpeta “messages”: Esta carpeta contiene los mensajes de log externalizados.
   * Fichero de configuración de Spring: “application.properties”.
   * Fichero de configuración de Log4j: “fire-log4j.xml”.
   * Fichero de configuración del servidor de correos: “mail.properties”. Esta configuración es usada por la herramienta para el envío de correos de recuperación de contraseña.
   * Fichero de configuración de mapeo de entidades de base de datos para MySql: “mysql-orm.properties”.
   * Fichero de configuración de conexiones con base de datos: “persistence.properties”. La configuración podrá ser para Oracle o MySql y la propiedad que establece el dialecto, será usada para que el sistema cargue o no el fichero de configuración de MySql.
3. Desplegar en el servidor de aplicaciones el archivo WAR del servicio de pruebas:
   * fire-admin-web.war

## Configuración del componente

Para que el componente de administración pueda operar será necesario configurar varios ficheros:

* persistence.properties.
* mail.properties.

En principio no será necesario configurar ninguna propiedad en el fichero de Spring “application.properties” ya que los valores de las variables no deberían cambiar según el entorno.

### Fichero persistence.properties

Este fichero permite la configuración de la conexión con la base de datos. Las propiedades que establece son las siguientes:

* datasource.driver
  + Clase del driver JDBC que se desea utilizar. Casos comunes son:
    - oracle.jdbc.driver.OracleDriver (Clase JDBC de Oracle)
    - com.mysql.cj.jdbc.Driver (Clase JDBC de MySQL)
* datasource.url
  + Cadena de conexión a la base de datos del sistema.
* datasource.user
  + Usuario para la conexión con la base de datos.
* datasource.pass
  + Contraseña para la conexión con la base de datos.
* hibernate.dialect
  + Clase del dialecto de base de datos. Casos comunes son:
  + org.hibernate.dialect.Oracle10gDialect (Para Oracle)
  + org.hibernate.dialect.MySQLDialect (Para MySQL)

La contraseña de usuario, se almacenará en base de datos utilizando la codificación “bcrypt”. Esta función de “hashing” de claves, es la utilizada por defecto por el framework “Spring Security” e implementada por la clase “*org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder*”.

### Configuración del servidor de correo

Este fichero permite la configuración del servidor de correo electrónico. Actualmente, esta configuración se utiliza para los siguientes fines:

* Recuperación de contraseña de los usuarios.

Las propiedades que establece son las siguientes:

* mail.smtp.starttls.enable
  + Indica si requiere conectarse por TLS. Puede ser “true” o “false”.
* mail.smtp.host
  + Domino del servidor SMTP desde el que se van a enviar los correos electrónicos.
* mail.smtp.port
  + Puerto del servidor SMTP desde el que se van a enviar los correos electrónicos.
* mail.smtp.mail.sender
  + Dirección de correo que aparecerá en las notificaciones por correo electrónico.
* mail.from.name
  + Nombre el emisor que aparecerá en las notificaciones por correo electrónico.
* mail.smtp.auth
  + Indica si requiere autenticación.
* mail.smtp.user
  + Nombre de usuario de la cuenta para el envío de correo electrónico.
* mail.smtp.password
  + Contraseña de la cuenta para el envío de notificaciones por correo electrónico.
* mail.password.expiration
  + Tiempo en milisegundos que puede transcurrir antes de que caduque un proceso de cambio de contraseña. Una vez un usuario ha solicitado cambiar la contraseña, sólo tiene el tiempo aquí indicado antes de que se deshabilite la posibilidad de cambiarla y deba volver a solicitarlo.

### Ficheros de ejemplo

Ejemplo de establecimiento de variables de arranque, en “catalina.bat” de Tomcat:

|  |
| --- |
| set "JAVA\_OPTS=%JAVA\_OPTS% -Dserver.config.dir=%CATALINA\_HOME%/conf/"  set "JAVA\_OPTS=%JAVA\_OPTS% -Dspring.config.location=%CATALINA\_HOME%/conf/application.properties"  set "JAVA\_OPTS=%JAVA\_OPTS% -Dlog4j.configuration=file:/%CATALINA\_HOME%/conf/fire-log4j.xml -Dfile.encoding=UTF-8" |

Un ejemplo de fichero de configuración de base de datos “persistence.properties”, sería:

|  |
| --- |
| datasource.driver = com.mysql.cj.jdbc.Driver  datasource.url = jdbc:mysql://127.0.0.1:3306/fire\_db?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC  datasource.user = fireownermysql  datasource.pass = 1111  hibernate.dialect = org.hibernate.dialect.MySQLDialect |

Un ejemplo de fichero de configuración de base de datos “mail.properties”, sería:

|  |
| --- |
| # Protocolo  mail.protocol=smtp  #Host del correo  mail.smtp.host=smtp.gmail.com  # Indica si requiere conectarse por tls  mail.smtp.starttls.enable=true  # Puerto  mail.smtp.port=587  # Indica el emisor del correo  mail.smtp.mail.sender=pruebas@gmail.com  # Usuario  mail.smtp.user=pruebas@gmail.com  # Contraseña  mail.smtp.password=prueba  # Indica si requiere autenticacion  mail.smtp.auth=true  # Indica el nombre del emisor del correo  mail.from.name=Administrador FIRe  # Indica el tiempo de caducidad del restablecimiento de contrasena (milisegundos)  mail.password.expiration=1800000 |

# Simulador de Cl@ve Firma

Junto con FIRe se distribuye un servicio que simula el comportamiento del proveedor Cl@ve Firma, permitiendo así probar a firmar desde nuestras aplicaciones sin necesidad de tener conexión con ningún proveedor real de firma en la nube.

## Despliegue

Para poder desplegar y conectar nuestro componente central con el servicio simulador de Cl@ve Firma hay que realizar los siguientes pasos:

1. Desplegar en el servidor de aplicaciones el archivo WAR del servicio de pruebas:
   * clavefirma-test-services.war
2. Agregar al directorio determinado por la variable de entorno “fire.config.path” o, en su defecto, al *classpath* del servidor de aplicaciones el fichero de configuración del simulador:
   * test-backend.properties

## Configuración del componente

Para configurar el servicio de simulación de Cl@ve Firma será necesario editar el fichero “test-backend.properties” para configurarle las siguientes propiedades:

* tmp\_dir
  + Directorio en el que se almacenarán los ficheros temporales necesarios durante el proceso de firma. El servicio debe tener permisos para crear, modificar y eliminar ficheros en este directorio. Si no se establece un valor, se usará el directorio de temporales del sistema.
* url\_service
  + URL base del servicio de pruebas. Esta URL debe ser accesible desde el equipo del usuario. Es decir, debe indicarse el nombre de dominio o IP necesario para que el equipo del usuario pueda acceder directamente a las páginas y servicios del simulador de pruebas. En caso de no indicarse, se interpretará que el despliegue de cliente y servidor se realizan en el mismo equipo y se buscarán los recursos en la dirección:
    - <http://localhost:8080/clavefirma-test-services>

## Uso del servicio de simulación de Cl@ve Firma

Para permitir que nuestra aplicación pueda hacer uso del servicio de simulación de Cl@ve Firma deberemos habilitar el uso del conector encargado de conectar con este servicio y configurar dicho conector.

Habilitaremos el uso del conector del servicio de pruebas desde el fichero “config.properties“ del componente central, en donde deberemos establecer las siguientes propiedades:

* provider.clavefirmatest
  + Esta propiedad debe establecer la clase del conector con el servicio de pruebas: “es.gob.fire.server.connector.test.TestConnector”.
* providers
  + Debe incluir entre los proveedores listados el nombre del proveedor del servicio de pruebas: “clavefirmatest”.
  + Por ejemplo:
    - providers=clavefirmatest,local

Configuraremos el conector para el uso del servicio simulador mediante el fichero “provider\_clavefirmatest.properties” del componente central. La configuración de este fichero se describe en el apartado 4.4.5 Fichero provider\_clavefirmatest.properties.

## Usuarios de prueba

El servicio simulador de Cl@ve Firma incluye por defecto varios usuarios de prueba:

* Usuario con certificado para la prueba de las operaciones de firma:
  + Usuario: **00001**
  + Contraseña: **1111**
* Usuario sin certificado para la prueba de las operaciones de generación de certificado.
  + Usuario: **00002**
  + Contraseña: **1111**
    - Durante la prueba de este servicio se dará al usuario la posibilidad de modificar la contraseña. Dado que este servicio tiene como único objetivo ayudar a la integración del servicio, los datos introducidos por el usuario se ignorarán y la contraseña siempre será la indicada.
* Usuario con certificado bloqueado:
  + Usuario: **00003**
* Usuario con registro débil/no fehaciente:
  + Usuario: **00004**

Adicionalmente, el servicio simulador permite utilizar el mismo certificado de pruebas cuando se le pide el certificado de una serie de usuarios, que son los utilizados como usuarios de en Cl@ve Permanente. Esto permitiría crear una aplicación que utilizase un usuario de prueba de Cl@ve Permanente y firmase con un certificado de prueba de FIRe. El listado de usuarios es el siguiente:

* **54500442X**
* **24540447E**
* **92920000T**
* **Y5010343Z**
* **Y5010344S**
* **Y5010345Q**
* **Y5010346V**
* **Y5010347H**

La contraseña para todos ellos es: **1111**

### Agregar nuevos usuarios de prueba

Si se desea añadir nuevos usuarios de prueba al servicio, hay que seguir los siguientes pasos:

1. Crear un fichero de propiedades con la configuración del nuevo certificado de pruebas. El nombre del fichero debe ser el identificador que se desee para el usuario y su extensión “.properties”. Por ejemplo, si deseamos dar de alta al usuario “00000000T”, deberemos crear el fichero “00000000T.properties”.

El fichero debe contener las siguientes propiedades:

* + state: Estado en el que deseamos que se encuentre el usuario de pruebas. Esta propiedad debe tener asignado uno de los siguientes valores:
    - OK: El usuario tiene un certificado válido y puede firmar. Este es el valor por defecto y requiere que se proporcione un almacén de claves en el paso 3 y se establezca su contraseña en la propiedad “password”.
    - NOCERT: Es un usuario correcto, pero sin certificados, por lo que deberá generarlos durante la operación de firma. Este caso requiere que se proporcione un almacén de claves en el paso 3 y se establezca su contraseña en la propiedad “password”.
    - BLOCKED: Es un usuario con su certificado bloqueado. En este caso no es necesario proporcionar un almacén de certificados ni indicar una contraseña.
    - WEAK\_REGISTRY: El usuario realizo un registro débil y no puede tener certificados de firma. En este caso no es necesario proporcionar un almacén de certificados ni indicar una contraseña.
  + password: Contraseña del almacén PKCS#12 con el certificado de firma. Esta propiedad sólo es necesaria si se estableció la propiedad “state” con el valor OK o NOCERT.

Un ejemplo de fichero sería:

state=OK

password=1111

1. Mover el fichero de propiedades creado a un directorio “testservice” dentro del *classpath* del servidor de aplicaciones. Como alternativa, también es posible abrir el WAR del fichero simulador de Cl@ve (clavefirma-test-services.war) y agregar el fichero de propiedades al subdirectorio “\WEB-INF\classes\testservice”.
2. Si se configuró el estado OK o NOCERT, se deberá agregar el almacén PKCS#12 con el certificado del nuevo usuario en el mismo directorio que el fichero de propiedades. El nombre de este fichero debe ser el identificador del usuario al cual pertenece, terminado con la extensión “.p12”. Por ejemplo, “00000000T.p12”. La contraseña de este almacén debe ser la indicada en el fichero de propiedades correspondiente.

Puede hacer uso del servicio de prueba desde su propia aplicación o desde la aplicación de ejemplo que se distribuye con FIRe. Recuerde que ni el servicio simulador de clave ni la aplicación de ejemplo distribuida con FIRe deberían desplegarse en entornos productivos.

# Componentes adicionales

Los componentes relevantes de FIRe se han descrito en los apartados anteriores, pero pueden existir otros componentes que den soporte a alguna de las funcionalidades agregadas a FIRe, como conectores para el uso de proveedores en la nube o gestores de documentos para la recuperación y guardado de los datos procesados por FIRe.

En este apartado se describen los componentes adicionales que se distribuyen junto a FIRe. Considere que estos componentes sirven a un fin concreto y, según el despliegue o la funcionalidad configurada en FIRe, es posible que no sea necesario su despliegue.

## Servicio auxiliar del conector de la FNMT

Este servicio permite al conector de CERES Cloud ID procesar las peticiones *Oauth* para la autorización del usuario. Este servicio sólo necesita desplegarse cuando se configure el conector de CERES Cloud ID en el componente central.

El servicio se distribuye en el archivo “fnmt-fire-service.war” y debe desplegarse en un contenedor de *servlets* compatible. Para el funcionamiento de este servicio es necesario disponer de una JRE 1.7 o superior. Este servicio puede desplegarse en los mismos nodos en los que se despliegue el componente central.

La configuración del servicio se realiza mediante el fichero “asyncnotifier.properties”. Este es el mismo fichero que utiliza el conector de la CERES Cloud ID del componente central y que debe situarse en el directorio configurado mediante la variable de entorno “fire.config.path”. En este fichero se definen las siguientes propiedades:

* tmpdir
  + Ruta absoluta del directorio temporal en el que almacenar los ficheros de compartición de datos entre el servicio auxiliar de la FNMT y el conector de CERES Cloud ID.

## Servicio para la consulta de logs desde el módulo de administración

El módulo de administración de FIRe incluye un apartado desde el que es posible realizar consultas sobre los logs de FIRe de los distintos nodos en los que se haya desplegado, e incluso permite la descarga de estos logs. Este servicio sólo debe desplegarse cuando se despliega también el módulo de administración y el componente central de FIRe se conecte a base de datos.

Para permitir la consulta de logs desde el módulo de administración de FIRe es necesario que en todos los nodos en los que se despliegue el componente central se despliegue también el archivo “log-consumer-service.war”. Este servicio es el que atiende a las consultas realizadas sobre los ficheros de log de un directorio concreto del servidor. El servicio no tiene dependencias con ningún contenedor de *servlets* concreto.

El servicio se configura por medio del fichero “logconsumer.properties” que debe encontrarse en el directorio configurado mediante la variable del entorno “logconsumer.config.path” o en el *classpath* del contenedor de servlets o la JRE que lo ejecuta. La variable se configura igual que la variable “fire.config.path”, con la que se indica a FIRe donde están sus ficheros de configuración.

Las propiedades que se deben configurar en este fichero son:

* auth.key
  + Clave para la autenticación con el servicio. Es la clave que deben indicar los servicios de consulta de logs para poder acceder a los logs de FIRe. Esta clave se deberá configurar en la interfaz de administración de FIRe cuando se dé de alta el servidor de log.
  + La clave debe ser cualquier clave AES válida codificada en base 64.
    - **NOTA**: Si se utiliza una clave AES-256, será necesario tener habilitado el uso de los algoritmos de alta seguridad de Java. Esto se encuentra habilitado por defecto en la JDK 7u171 y superiores, JDK 8u161 o en cualquier JDK 9 o superior. Si está usando una versión que no lo tiene habilitado por defecto, deberá instalar la *Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy* (<https://www.oracle.com/java/technologies/javase-jce-all-downloads.html>).
  + Por seguridad, cada nodo de FIRe en el que se despliegue este servicio debería usar una clave distinta.
  + Ejemplo:
    - auth.key= wbJqwyrBND6q+tyz+q0F/y/4pJToDLz58IREaYOtDjc=
* logs.dir
  + Ruta absoluta del directorio en el que se generan los ficheros de log de FIRe.
  + El servicio de logs sólo permitirá consultar los ficheros que haya en este directorio.
  + Debería ser el mismo directorio que se configura en la propiedad “logs.dir” del fichero “config.properties” del componente central de FIRe.
  + Ejemplo:
    - logs.dir= /users/prueba/fire/logs

# Configuración de los componentes de FIRe

FIRe utiliza múltiples ficheros para la configuración del comportamiento de sus distintos módulos y la comunicación entre ellos. El uso y contenido de estos ficheros se ha explicado en los apartados correspondientes a los módulos a los que afectan. En este apartado se explicarán los aspectos que afectan a todos estos ficheros, a excepción de los ficheros de los componentes adicionales, que no pertenecen al propio proyecto FIRe.

Los criterios que siguen todos los ficheros de configuración de FIRe son los siguientes:

* Son ficheros de propiedades en texto plano, lo que hace posible modificarlos mediante cualquier editor de textos, como el Bloc de Notas de Windows o el “vi” de UNIX.
* Utilizan el juego de caracteres UTF-8.
* Se buscarán en el directorio configurado mediante la variable de entorno “fire.config.path” proporcionada a la JVM. Esto puede hacerse comúnmente a través de la variable JAVA\_OPTS que carga el servidor de aplicaciones, proporcionándole el valor -Dfire.config.path=”/*directorio*”. Si no se configura esta variable, los ficheros se buscarán en el *classpath* del servidor de aplicaciones.
* Permiten utilizar variables de entorno para la configuración del valor o parte del valor de sus propiedades. Este punto se describe en detalle en el apartado 9.1 Uso de variables de entorno.
* Permiten escribir de forma cifrada el valor o parte del valor de sus propiedades. Este punto se describe en detalle en el apartado 9.2 Cifrado de propiedades.
  + El cifrado de propiedades no se admite en los ficheros de configuración de los componentes de pruebas (test-app.properties y test-backend.properties).

## Uso de variables de entorno

Los ficheros de configuración de FIRe permiten cargar variables de entorno como valores o parte de los valores de sus propiedades. Esto se puede hacer estableciendo en los ficheros el nombre de la variable delimitada por las partículas “${” y “}”. Al cargar el fichero, FIRe identificará que debe utilizar como valor el indicado en una variable de entorno, la cargará y sustituirá la variable por su valor en la configuración interna.

Algunos ejemplos de propiedades son:

* temp.dir=${temp\_dir}
  + FIRe detectará que se quiere utilizar como valor de la propiedad “temp.dir” el valor existente en la variable de entorno “temp\_dir”. Esta debería ser una variable que el administrador del sistema proporciona al servidor de aplicaciones durante el despliegue y en el que hubiese establecido el directorio en el que FIRe debe almacenar los ficheros temporales.
* javax.net.ssl.keyStore=${catalina.home}/conf/certs/client\_ssl.jks
  + FIRe detectará que se quiere utilizar como parte del valor de la propiedad “javax.net.ssl.keyStore” el valor existente en la variable de entorno “catalina.home”. Esta variable de entorno es una que siempre establece Apache Tomcat durante su arranque y en la que almacena la ruta del propio servidor de aplicaciones.

## Cifrado de propiedades

FIRe permite que los valores de las propiedades de los distintos ficheros de configuración se indiquen cifrados y codificados en base64. El mecanismo de cifrado es ajeno a FIRe. Es el administrador del sistema el que debe cifrar los datos, configurar el valor cifrado en el fichero de configuración y proporcionar a FIRe una forma de descifrarlos en el momento en el que vaya a necesitarlos.

El desarrollo del mecanismo de cifrado corre a cuenta del organismo que desee emplearlo. Este mecanismo debe desarrollarse en lenguaje Java, mediante la implementación de una interfaz específica que varía según se vaya a utilizar en un módulo servidor o en el componente distribuido Java, y configurarse para su uso.

Los pasos concretos para el uso del cifrado en los ficheros de configuración son:

1. Implementación del método de descifrado:

* En el caso de los módulos servidor (componente central y de administración), se implementará la interfaz “es.gob.fire.server.decipher.PropertyDecipher” y su método “decipher”. El método de descifrado recibirá el valor decodificado del Base 64 indicado en el fichero de configuración. Esta clase se encuentra en el módulo “fire-signature-decipher”.

|  |
| --- |
| <dependency>  <groupId>es.gob.fire</groupId>  <artifactId>fire-signature-decipher</artifactId>  <version>2.4</version>  </dependency> |

* En el caso del componente distribuido Java, sólo se podrán indicar cifradas las distintas contraseñas de las que hace uso. Para permitir al componente descifrarlas se implementará la interfaz “es.gob.fire.client.PasswordDecipher” y su método “decipher”. El método de descifrado recibirá el valor decodificado del Base 64 indicado en el fichero de configuración. Esta clase se encuentra en el módulo “fire-client”.

|  |
| --- |
| <dependency>  <groupId>es.gob.fire</groupId>  <artifactId>fire-client</artifactId>  <version>2.4</version>  </dependency> |

1. La clase de descifrado deberá agregarse al *classpath* del servidor de aplicaciones, por ejemplo, empaquetándola en un JAR y agregándolo al WAR del componente en el que se vaya a utilizar.
2. Indicar a FIRe cuál es la clase para el descifrado.

* En los módulos servidor se hará configurando la propiedad “cipher.class” en el fichero de configuración que la contenga (config.properties para todos los ficheros de configuración del componente central y admin\_config.properties para los ficheros del módulo de administración). Por ejemplo:

cipher.class=es.gob.miapp.fire.MiDecipher

* En el componente distribuido Java se proporcionará una instancia de la clase de descifrado al construir el cliente. Por ejemplo:

PasswordDecipher **decipher** = **new** es.gob.miapp.fire.MiDecipher();

FireClient **client** = **new** FireClient(appId, config, **decipher**);

1. Establecer los valores o fragmentos cifrados y codificados en base 64 en las propiedades que se deseen de los distintos ficheros de configuración. Esto se hará mediante una cadena del tipo:

{@ciphered: CADENA\_CIFRADA\_B64 }

Por ejemplo:

bbdd.conn=jdbc:mysql://127.0.0.1:3306/fire\_db?user=miusuario&password={@ciphered: aDbb+4nmBHk7ifT= }

Para saber cómo configurar expresamente la clase de descifrado en un componente, consulte con el apartado expreso de dicho componente.

# Nuevos proveedores de firma

FIRe permite configurar múltiples proveedores de firma en la nube. En la distribución básica de FIRe se distribuyen los conectores para hacer uso de varios proveedores, pero se pueden agregar otros nuevos, que se distribuyan de forma independiente o sean desarrollados por su organismo.

Es muy importante comprobar que los conectores que se desean incorporar a FIRe proceden de entidades de confianza. Estos conectores se ejecutarán dentro del servidor de aplicaciones de su organismo, con lo cual pueden llegar a afectar a FIRe u otras aplicaciones desplegadas, así como acceder a información de los usuarios de FIRe y los documentos que se procesan. No despliegue en FIRe conectores de dudosa procedencia o que crea que pueden suponer un riesgo para la seguridad de su información o de sus usuarios.

Tenga en cuenta que el despliegue de un nuevo conector implica modificar el despliegue que exista actualmente y debería realizarse con el servidor de aplicaciones detenido para garantizar el correcto funcionamiento. Prepare todo de antemano antes de realizar el despliegue e intente realizarlo en un momento de baja carga de trabajo del servidor. En un despliegue sobre múltiples nodos puede ir abordando esta tarea de forma secuencial para no interrumpir el servicio.

En este apartado se describen los pasos genéricos para el despliegue de nuevos conectores en el componente central de FIRe. Es probable que a estos pasos se agreguen otros específicos para cada conector particular. Consulte la documentación de los conectores que desee integrar para conocer las necesidades concretas de estos y las opciones de configuración que soportan.

La sucesión de pasos a seguir es:

1. Agregar al componente central de FIRe las bibliotecas del nuevo conector.
   * El componente central de FIRe consiste en un archivo WAR que debe desplegarse en un servidor de aplicaciones. Este WAR contiene todos los JAR que son dependencias del componente central y de los conectores que incorpora por defecto.
   * Para desplegar el nuevo conector, introduzca en el WAR “afirma-signature.war” los JAR del nuevo conector y sus dependencias. Esto puede hacerlo mediante una herramienta de compresión de ficheros, ya que un fichero WAR es un fichero ZIP con otra extensión. Estos ficheros JAR deberán copiarse en el subdirectorio “WEB-INF\lib”.
   * Compruebe que las dependencias del nuevo conector no incluyen bibliotecas que puedan generar incompatibilidades con alguna dependencia de cualquier otro conector desplegado. Por ejemplo, que haya varias bibliotecas iguales con distinto nombre (si las hay, elimine todas salvo una de ellas), varias versiones de una misma biblioteca (elimine todas salvo la versión más nueva), o bibliotecas que se sepan incompatibles. En el caso de que no se puedan solventar estas incompatibilidades, consulte la documentación del conector por si existiese un modo alternativo de despliegue.
2. Desplegar los servicios auxiliares y recursos externos.
   * En caso de que su conector cuente con servicios auxiliares externos para su funcionamiento, despliéguelos normalmente en el servidor de aplicaciones del componente central o en algún otro servidor accesible desde este.
   * Si se requiere del uso de un recurso externo, un directorio temporal o similar, dispóngalos siguiendo las pautas dadas en el manual del conector.
3. Preparar el fichero de configuración del conector
   * Cree en el directorio de ficheros de configuración de FIRe un nuevo fichero llamado *PROVEEDOR*\_config.properties, en donde *PROVEEDOR* será el nombre que se desea asignar al conector. Este fichero será un Properties en el que se deberán configurar todas las propiedades determinadas en la documentación del conector.
4. Habilitar el conector
   * Edite el fichero config.properties de FIRe para agregar la propiedad “provider.*PROVEEDOR*”, en donde *PROVEEDOR* será el nombre de proveedor que ya usamos en el paso anterior. Como valor de esta propiedad se establecerá el nombre de la clase conectora. El nombre completo de esta clase se le debe proporcionar en la documentación del conector.
   * Edite la propiedad “providers” del fichero config.properties para agregar al listado el nombre del nuevo proveedor.

# Despliegue en entornos balanceados

Los componentes distribuidos y de administración de FIRe no almacenan datos temporales o de sesión en el servidor, por lo que pueden desplegarse en un entorno balanceado sin necesidad de realizar ninguna actuación adicional para garantizar su funcionamiento.

El componente centralizado y el servicio simulador de Cl@ve Firma, en cambio, almacenan datos temporales en disco. El componente centralizado, además, trabaja con sesiones y dispone tanto de servicios utilizados por la aplicación cliente como de páginas web a las que acceden los usuarios finales, por lo que es común que en la ejecución de un mismo trámite participen distintos nodos que deberían compartir unidad de disco y objetos en memoria.

Para simplificar el despliegue en entornos balanceados, el componente central permite la compartición de sesiones y documentos mediante varios mecanismos. Estos son:

* Guardado en disco.
  + Permite almacenar temporalmente los recursos a compartir en una unidad de disco accesible desde todos los nodos del componente central.
* Guardado en base de datos.
  + Permite almacenar temporalmente los recursos a compartir en la base de datos configurada en la componente central.

El componente central de FIRe gestiona por separado la compartición de sesiones y documentos entre los nodos. De esta forma, es posible configurar que, por ejemplo, las sesiones se compartan a través de base de datos mientras que los documentos temporales se compartan a través de disco.

Para el uso de los mecanismos de compartición se deberán seguir los siguientes pasos:

1. Disponer de almacenamiento compartido por todos los nodos que desplieguen el componente central, ya sea disco o base de datos.
2. Configurar que los objetos temporales del componente central se almacenen en el almacenamiento compartido.
3. Mantener la hora de todos los nodos sincronizados en la medida de lo posible. Cada uno de los nodos puede ejecutar revisiones periódicas de los ficheros temporales para eliminar aquellos que se consideren caducados. En caso de desincronización, un nodo podría guardar un recurso y, otro nodo, con una hora más, eliminarlo de inmediato por considerarlo caducado según su reloj.
4. Configurar los gestores requeridos para compartir las sesiones y documentos a través del fichero “config.properties” del componente central.
   * Se puede establecer el gestor de sesiones a través de la propiedad “sessions.dao” del fichero. Los valores aceptados son:
     + es.gob.fire.server.services.internal.sessions.FileSystemSessionsDAO
       - Gestor que comparte las sesiones a través de disco. Las sesiones se almacenan en el directorio configurado en la propiedad “temp.dir”, que deberá encontrarse en el almacenamiento compartido por los nodos.
       - Si no se configura la compartición de documentos, al establecer este valor se entenderá que los documentos también se compartirán a través del sistema de ficheros.
     + es.gob.fire.server.services.internal.sessions.DBSessionsDAO
       - Gestor que comparte las sesiones a través de base de datos. Las sesiones se almacenarán en la base de datos configurada a través de la propiedad “bbdd.conn”.
       - Requiere que el esquema de la base de datos tenga creada la tabla “tb\_comp\_sesiones”.
       - Si no se configura la compartición de documentos, al establecer este valor se entenderá que estos también se deben compartir a través de base de datos.
   * Se puede habilitar la compartición de documentos a través de la propiedad “sessions.documents.dao” del fichero. Los valores aceptados son:
     + es.gob.fire.server.services.internal.sessions.FileSystemTempDocumentsDAO
       - Gestor que comparte los documentos a través de disco. Los documentos se almacenan en el directorio configurado en la propiedad “temp.dir”, que deberá encontrarse en un almacenamiento compartido por los nodos.
     + es.gob.fire.server.services.internal.sessions.DBTempDocumentsDAO
       - Gestor que comparte los documentos a través de base de datos. Los documentos se almacenarán en la base de datos configurada a través de la propiedad “bbdd.conn”.
       - Requiere que el esquema de la base de datos tenga creada la tabla “tb\_comp\_documentos”.
   * No se debería configurar la compartición de sesiones o documentos si no se está realizando un despliegue en un entorno balanceado, ya que su uso implica la realización de tareas innecesarias cuando el despliegue se realiza en un único nodo.
5. Configurar que los objetos temporales del servicio simulador se almacenen en el almacenamiento compartido. Esto se realizará a través de la propiedad “tmp\_dir” del fichero “test-backend.properties”, mediante la cual se configura la ruta del directorio de ficheros temporales de la aplicación.
   * Esta medida sólo es necesaria en entornos de prueba, ya que el servicio simulador no debería desplegarse en producción.
6. Migración a FIRe 2.4
   1. Migración desde FIRe 2.1 / 2.1.1
      1. Migración de la base de datos

Versiones posteriores de FIRe introdujeron la posibilidad de gestionar una mayor cantidad de elementos desde su módulo de administración, lo que implicó cambios en su base de datos y en cómo el componente central accede a la información de las aplicaciones.

Se distribuyen junto a FIRe diversos scripts SQL para actualizar las bases de datos Oracle y MySQL desde versiones anteriores de FIRe. Estos scripts crearán las nuevas tablas de base de datos necesarias y migrarán los datos de su base de datos al nuevo modelo para que no tenga que hacer nada de forma manual.

Para la actualización a la última versión de la base de datos, es necesario ejecutar ordenadamente los siguientes scripts:

1. Migracion\_fire\_2\_-\_2\_2.sql
2. Migracion\_fire\_2\_2\_-\_2\_4.sql

Algunos datos de las nuevas tablas, se rellenarán con datos por defecto por no haber existido hasta el momento. Consulte el manual de administración de FIRe para conocer las nuevas capacidades del módulo de administración y como, opcionalmente, completar estos datos.

La información de las aplicaciones de la antigua base de datos se conservará en la tabla “tb\_aplicaciones\_old”.

Si tiene problemas al migrar a la nueva versión de FIRe, pruebe a eliminar el resto de tablas de la base de datos, ejecutar los scripts de su anterior versión de FIRe y copiar los datos de la tabla “tb\_aplicaciones\_old” a “tb\_aplicaciones”.

Una vez complete satisfactoriamente la migración a la última versión de FIRe, ya puede eliminar la tabla “tb\_aplicaciones\_old”.

Si se desease realizar la migración de forma manual, se deberían seguir los siguientes pasos:

1. Traspasar los datos de usuario por defecto creado en la tabla tb\_configuracion, a tb\_usuarios donde el campo parametro se corresponderá con el campo nombre\_usuario y el campo valor se corresponde con el campo clave. Los campos nombre y apellidos al ser obligatorios deberán rellenarse con algún dato por defecto al insertarlos.
2. Traspasar los datos de los certificados de la tabla tb\_aplicaciones a la nueva tabla tb\_certificados. Los campos cer y huella a los campos cert\_principal y huella\_principal respectivamente. Se debe tener en cuenta que, al crear los registros en la nueva tabla, el campo id\_certificado que es auto-numérico se corresponderá con la nueva tabla tb\_aplicaciones con el campo fk\_certificado y deberá tener un nombre para el certificado, en los ejemplos de los scripts de traspaso para MySQL y ORACLE el nombre del certificado se compone de “CERT\_” + campo id de la tabla tb\_aplicaciones.
3. Traspasar los datos de la anterior tabla de tb\_aplicacione a la nueva tabla, sin los datos anteriormente mencionados de certificados (cer y huella). En dicho traspaso se deberán también indicar el dato numérico en el campo fk\_certificado correspondiente al campo id\_certificado creado anteriormente en la tabla de certificados.

Adicionalmente, a partir de FIRe 2.4 se permite configurar la compartición de sesiones entre nodo a través de base de datos en lugar de a través de disco. Si se desease hacer uso de esta funcionalidad, se debería ejecutar también el script “Opcional\_sesiones\_en\_bd\_ddl.sql”.

* + 1. Migración de la configuración del componente central

FIRe 2.2 introdujo nuevas opciones de configuración, así como el nuevo sistema multiproveedor que permite agregar nuevos proveedores y obliga a que estos se configuren de forma independiente.

Para reutilizar la configuración de su FIRe 2.1/2.1.1 deberá aplicar los siguientes cambios:

* Fichero config.properties
  + La propiedad “temp.clavefirma.timeout” ha cambiado de nombre. Renómbrela como “temp.fire.timeout”.
  + La propiedad “backendClassName” desaparece, ya que ahora es posible configurar varios proveedores simultáneos. Para migrar su configuración actual, realice las siguientes acciones según el conector que tuviese configurado:
    - Si tenía configurado el conector del simulador de pruebas:
      * Renombre la propiedad “backendClassName” como “provider.clavefirmatest”.
      * Agregue a su fichero de configuración la propiedad “providers” con el valor “clavefirmatest,local”.

providers=clavefirmatest,local

provider.clavefirmatest=es.gob.fire.server.connector.test.TestConnector

* + - Si tenía configurado el conector del simulador de pruebas:
      * Renombre la propiedad “backendClassName” como “provider.clavefirma”.
      * Agregue a su fichero de configuración la propiedad “providers” con el valor “clavefirma,local”.

providers=clavefirma,local

provider.clavefirma=es.gob.fire.server.connector.clavefirma.ClaveFirmaConnector

* + La propiedad de configuración correspondiente al conector de Cl@ve Firma se deberá trasladar al fichero “provider\_clavefirma.properties”. Esta es:
    - clavefirma.providerName

Consulte el punto correspondiente al fichero “provider\_clavefirma.properties” para más información.

* + Las propiedades de configuración correspondientes al conector del simulador de Cl@ve Firma se deberán trasladar al fichero “provider\_clavefirmatest.properties”.
    - test.endpoint
    - test.ssl.keystore
    - test.ssl.keystorePass
    - test.ssl.keystoreType
    - test.ssl.truststore
    - test.ssl.truststorePass
    - test.ssl.truststoreType

Consulte el punto correspondiente al fichero “provider\_clavefirmatest.properties” para más información.

* + Se agrega la propiedad “validator.class” para configurar la clase conectora con el sistema de validación y actualización de firmas.
    - Para mantener el funcionamiento de versiones anteriores de FIRe (validación mediante la Plataforma @firma), esta propiedad debe establecerse con el valor por defecto, “es.gob.fire.upgrade.afirma.AfirmaValidator”.
  + Se traslada la propiedad “afirma.appId” al fichero “platform.properties”.
  + Si se tiene establecida la propiedad “sessions.dao” con el valor “filesystem” para indicar la compartición de sesiones entre nodos debe sustituirse su valor por “es.gob.fire.server.services.internal.sessions.FileSystemTempDocumentsDAO”
* Fichero gatewayapi.properties
  + Este fichero desaparece y sus propiedades deben trasladarse al fichero “provider\_clavefirma.properties”.
  + Consulte el punto correspondiente al fichero “provider\_clavefirma.properties” para más información.
* Fichero provider\_clavefirma.properties
  + Este fichero configura el comportamiento del conector de Cl@ve Firma y debe contener la propiedad “clavefirma.providerName” extraída del fichero “config.properties” y todas las propiedades del fichero “gatewayapi.properties”.
  + Este fichero también permite configurar nuevas propiedades. Consulte el apartado Fichero provider\_clavefirma.properties para conocerlas.
  + Por ejemplo:

clavefirma.providerName=MI\_PROVIDER\_NAME

URL\_GATEWAY=https://direccionclavefirma.gob.es:452/servicio

AUTH\_STORE=RUTA\_ABSOLUTA\_ALMACEN\_PKCS12

AUTH\_STORE\_PASS=CONTRASEÑA\_ALMACEN\_PKCS12

* Fichero provider\_clavefirmatest.properties
  + Este fichero configura el comportamiento del conector del simulador de Cl@ve Firma para pruebas y debe contener las propiedades que comenzaban por “test.” del fichero “config.properties”.
  + Por ejemplo:

test.endpoint=https://127.0.0.1:8443/clavefirma-test-services

test.ssl.keystore=C:/Users/usuario/SSL/client\_ssl.jks

test.ssl.keystorePass=12345678

test.ssl.keystoreType=JKS

test.ssl.truststore=all

#test.ssl.truststorePass=

#test.ssl.truststoreType=

Consulte el apartado Fichero config.properties para identificar si la nueva versión de FIRe incluye alguna nueva propiedad de configuración que pueda ser de interés para su despliegue.

* + 1. Migración de las aplicaciones

Las aplicaciones que utilicen FIRe no deberán realizar ningún cambio en su código. Sin embargo, es necesario que actualicen el componente distribuido que estén utilizando (Java, .NET o PHP) por el correspondiente de la versión de FIRe que se haya desplegado.

* + - 1. Actualización del componente distribuido JAVA

En el caso de utilizar el componente distribuido Java, también será necesario importar en su aplicación la biblioteca de SLF4J (versión 1.7.25) y la biblioteca correspondiente al sistema de log que se desee utilizar. Consulte el manual del integrador de FIRe para más información.

* + - 1. Actualización del componente distribuido PHP

Opcionalmente, durante la migración a la nueva versión del componente distribuido PHP, puede sustituirse el uso de “fire\_api.php” en favor del nuevo “fire\_client.php”, incorporado en FIRe 2.4. Todas las nuevas funcionalidades que se agreguen a FIRe se implementarán sólo en esta nueva biblioteca.

Mientras que “fire\_api.php” definía directamente los métodos para la llamada a las distintas operaciones de FIRe, el nuevo “fire\_client.php” define una clase “FireClient” mediante la que se pueden llamar a todos esos mismos métodos. La principal diferencia al usar un objeto FIReClient es que es en su constructor cuando proporcionamos el identificador de nuestra aplicación, por lo que este es un dato que ya no aparece en ninguno de los métodos de las operaciones a FIRe.

* 1. Migración desde FIRe 2.2
     1. Migración de la base de datos

Las nuevas versiones de FIRe han agregado nuevas tablas de base de datos para el funcionamiento de sus nuevas capacidades.

Se distribuye junto a FIRe un script SQL para actualizar las bases de datos Oracle y MySQL desde versiones anteriores de FIRe (“Migracion\_fire\_2\_2\_-\_2\_4.sql”). Este script creará las nuevas tablas de base de datos necesarias, que inicialmente podrán estar vacías.

Adicionalmente, a partir de FIRe 2.4 se permite configurar la compartición de sesiones entre nodo a través de base de datos en lugar de a través de disco. Si se desease hacer uso de esta funcionalidad, se debería ejecutar también el script “Opcional\_sesiones\_en\_bd\_ddl.sql”.

Si se utiliza un SGBD distinto a Oracle y MySQL, su administrador (DBA), deberá generar las sentencias SQL necesarias para construir las tablas del nuevo modelo de datos. Consulte el modelo de datos de la nueva versión de FIRe en el apartado 4.5.2 Modelo de datos.

* + 1. Migración de la configuración del componente central

La nueva versión de FIRe introduce nuevos ficheros y opciones de configuración, algunas de las cuales son necesarias para replicar el comportamiento de esta versión del componente:

* Fichero config.properties
  + Se agrega la propiedad “validator.class” para configurar la clase conectora con el sistema de validación y actualización de firmas.
    - Para mantener el funcionamiento de versiones anteriores de FIRe (validación mediante la Plataforma @firma), esta propiedad debe establecerse con el valor por defecto, “es.gob.fire.upgrade.afirma.AfirmaValidator”.
  + Se traslada la propiedad “afirma.appId” al fichero “platform.properties”.
  + Si se tiene establecida la propiedad “sessions.dao” con el valor “filesystem” para indicar la compartición de sesiones entre nodos debe sustituirse su valor por “es.gob.fire.server.services.internal.sessions.FileSystemTempDocumentsDAO”
    1. Migración de las aplicaciones

Las aplicaciones que utilicen el componente distribuido de FIRe 2.2 no deberán realizar ningún cambio en su código.

Los componentes distribuidos de FIRe 2.2 son compatibles con la nueva versión de FIRe, por lo que tampoco es obligatoria su actualización, aunque sí es aconsejable actualizarlos para aprovechar todas las características de la nueva versión, identificar mejor algunos errores y simplificar la migración a futuras versiones. Para actualizar al nuevo componente distribuido sólo es necesario sustituir el antiguo componente por el nuevo (JAR, DLL o fichero PHP).

* + - 1. Actualización del componente distribuido JAVA

En el caso de utilizar el componente distribuido Java, también será necesario importar en su aplicación la biblioteca de SLF4J (versión 1.7.25) y la biblioteca correspondiente al sistema de log que se desee utilizar. Consulte el manual del integrador de FIRe para más información.

* + - 1. Actualización del componente distribuido PHP

Opcionalmente, durante la migración a la nueva versión del componente distribuido PHP, puede sustituirse el uso de “fire\_api.php” en favor del nuevo “fire\_client.php”, incorporado en FIRe 2.4. Todas las nuevas funcionalidades que se agreguen a FIRe se implementarán sólo en esta nueva biblioteca.

Mientras que “fire\_api.php” definía directamente los métodos para la llamada a las distintas operaciones de FIRe, el nuevo “fire\_client.php” define una clase “FireClient” mediante la que se pueden llamar a todos esos mismos métodos. La principal diferencia al usar un objeto FIReClient es que es en su constructor cuando proporcionamos el identificador de nuestra aplicación, por lo que este es un dato que ya no aparece en ninguno de los métodos de las operaciones a FIRe.

* 1. Migración desde FIRe 2.3 / 2.3.1
     1. Migración de la base de datos

Las nuevas versiones de FIRe han agregado nuevas tablas de base de datos para el funcionamiento de sus nuevas capacidades.

Se distribuye junto a FIRe un script SQL para actualizar las bases de datos Oracle y MySQL desde versiones anteriores de FIRe. El script a utilizar para la migración desde FIRe 2.3 es el mismo que el de la migración desde FIRe 2.2 (“Migracion\_fire\_2\_2\_-\_2\_4.sql”). Este script creará las nuevas tablas de base de datos necesarias, que inicialmente podrán estar vacías.

Adicionalmente, a partir de FIRe 2.4 se permite configurar la compartición de sesiones entre nodo a través de base de datos en lugar de a través de disco. Si se desease hacer uso de esta funcionalidad, se debería ejecutar también el script “Opcional\_sesiones\_en\_bd\_ddl.sql”.

Si se utiliza un SGBD distinto a Oracle y MySQL, su administrador (DBA), deberá generar las sentencias SQL necesarias para construir las tablas del nuevo modelo de datos. Consulte el modelo de datos de la nueva versión de FIRe en el apartado 4.5.2 Modelo de datos.

* + 1. Migración de la configuración del componente central

La nueva versión de FIRe introduce nuevos ficheros y opciones de configuración, algunas de las cuales son necesarias para replicar el comportamiento de esta versión anterior del componente:

* Fichero config.properties
  + Se agrega la propiedad “validator.class” para configurar la clase conectora con el sistema de validación y actualización de firmas.
    - Para mantener el funcionamiento de versiones anteriores de FIRe (validación mediante la Plataforma @firma), esta propiedad debe establecerse con el valor por defecto, “es.gob.fire.upgrade.afirma.AfirmaValidator”.
  + Se traslada la propiedad “afirma.appId” al fichero “platform.properties”.
  + Si se tiene establecida la propiedad “sessions.dao” con el valor “filesystem” para indicar la compartición de sesiones entre nodos debe sustituirse su valor por “es.gob.fire.server.services.internal.sessions.FileSystemTempDocumentsDAO”
    1. Migración de las aplicaciones

Las aplicaciones que utilicen el componente distribuido de FIRe 2.3 no deberán realizar ningún cambio en su código.

Los componentes distribuidos de FIRe 2.3 son compatibles con la nueva versión de FIRe, por lo que tampoco es obligatoria su actualización, aunque sí es aconsejable actualizarlos para aprovechar todas las características de la nueva versión, identificar mejor algunos errores y simplificar la migración a futuras versiones. Para actualizar al nuevo componente distribuido sólo es necesario sustituir el antiguo componente por el nuevo (JAR, DLL o fichero PHP).

* + - 1. Actualización del componente distribuido PHP

Opcionalmente, durante la migración a la nueva versión del componente distribuido PHP, puede sustituirse el uso de “fire\_api.php” en favor del nuevo “fire\_client.php”, incorporado en FIRe 2.4. Todas las nue-vas funcionalidades que se agreguen a FIRe se implementarán sólo en esta nueva biblioteca.

Mientras que “fire\_api.php” definía directamente los métodos para la llamada a las distintas operaciones de FIRe, el nuevo “fire\_client.php” define una clase “FireClient” mediante la que se pueden llamar a todos esos mismos métodos. La principal diferencia al usar un objeto FIReClient es que es en su constructor cuando proporcionamos el identificador de nuestra aplicación, por lo que este es un dato que ya no aparece en ninguno de los métodos de las operaciones a FIRe.

1. Despliegue de demostración sobre Apache Tomcat

En el kit de integración se ofrece un despliegue de demostración realizado sobre Apache Tomcat para que sirva de referencia y de ayuda en el uso de los diferentes ficheros de configuración.

Este Tomcat:

* Tiene desplegado:
  + El componente central (fire-signature.war).
  + La aplicación de ejemplo (fire-test-jsp.war)
  + El servicio simulador de Cl@ve Firma (clavefirma-test-services.war)
  + El módulo de administración (fire-admin-jsp.war).
* Se ha agregado al fichero “catalina.bat” la opción “fire.config.path” a la variable JAVA\_OPTS para que se carguen los ficheros de configuración de FIRe desde el subdirectorio “*conf/fire\_config*” del servidor. Todos los ficheros de configuración de FIRe se pueden encontrar en este directorio.
* Se ha creado un directorio “fire\_temp” en el directorio base del servidor y se han configurado todos los ficheros de configuración de FIRe para que almacenen los temporales en el mismo.
* Tiene configurado en el fichero “server.xml” el uso de SSL con autenticación cliente y un *TrustManager* a medida para que deje pasar peticiones independientemente del certificado utilizado. El certificado SSL servidor se incluye en el propio Tomcat.
* En el directorio de bibliotecas de Tomcat se ha incluido el JAR “fire-trustManagerClassName-1.0.jar”, para permitir que el servidor deje pasar cualquier certificado cliente de autenticación y sea la propia aplicación la que decida si lo acepta o rechaza. También se encuentra desplegada cierta versión de la biblioteca JDBC de MySQL. Si se va a configurar el despliegue para que use otro SGBD o una versión distinta de MySQL, será necesario sustituir esta biblioteca por la que corresponda.

Para hacer funcionar la aplicación de prueba de FIRe sobre este despliegue es necesario:

1. Agregar al directorio de bibliotecas del Tomcat de pruebas (lib) el controlador JDBC de su base de datos.
2. Configurar en el fichero “config.properties” el driver y la cadena de conexión a la base de datos.
3. Configurar en el fichero “admin\_config.properties” el driver y la cadena de conexión a la base de datos.
4. Dar de alta a través del módulo de administración, el certificado “client\_ssl.cer” (incluido en el subdirectorio “conf/certs” del servidor) y dar de alta una aplicación de pruebas asociándole ese certificado recién creado.
   * En la configuración por defecto, se puede acceder al módulo de administración a través de la URL:
     + <https://127.0.0.1:8443/fire-admin-jsp/Login.jsp>
5. Configurar en el fichero “test-app.properties” la propiedad “appId”, el identificador que la herramienta de administración ha asignado a la aplicación.
6. **ADVERTENCIA**: El componente de firma local AutoFirma incluye como medida de seguridad no firmar cuando se ejecuta desde una web desplegada en local (127.0.0.1 o localhost). Para poder utilizar las funciones de firma con certificado local en esta aplicación de ejemplo, se deberá optar por una de las siguientes opciones:
   * Modificar el fichero “fireenv.bat” situado en el directorio “bin” del servidor de aplicaciones y establecer ahí la IP de red del equipo local.
   * Agregar un alias a la dirección 127.0.0.1 en el fichero “hosts” del sistema y utilizar este alias como nombre de dominio en el fichero “fireenv.bat” indicado en el punto anterior.
   * Realizar el despliegue en un equipo remoto y modificar todos los ficheros de configuración con la IP o nombre de dominio correspondiente.

Una vez realizados estos cambios, si se usó la IP por defecto para el despliegue (127.0.0.1), se podrá acceder a la página de pruebas a través de la URL: <https://127.0.0.1:8443/fire-test-jsp/Login.jsp>

Otras cosas a tener en cuenta:

* Para utilizar otros proveedores desde la aplicación de pruebas (como el de Cl@ve Firma o CERES Cloud ID), será necesario modificar el valor de la propiedad “providers” del fichero “config.properties” y agregar los ficheros de configuración apropiados con los datos para conectar con los servicios de los proveedores.

1. Problemas conocidos
   1. Error en las firmas XAdES en despliegues sobre JBoss

El servidor de aplicaciones JBoss, en algunas de sus versiones, incluye las bibliotecas de Xerces como extensiones accesibles por las aplicaciones. Estas bibliotecas sustituyen a la versión de las mismas incluidas en Java, lo que provoca que FIRe se encuentre con una versión distinta a la esperada. Este problema deriva en que las firmas XAdES fallen a la hora de construir el XML de la firma.

En algunas versiones de JBoss, como con JBoss 7, este problema se puede solventar mediante un mecanismo incluido en el propio FIRe. Por medio de la propiedad “signature.alternativeXmldsig”, incluida en el fichero de configuración config.properties, se puede notificar al componente para que intente utilizar correctamente la nueva versión de la biblioteca.

Sin embargo, esto no resuelve todos los casos de error detectados. En algunos casos, como con JBoss FUSE, será necesario indicar expresamente al servidor de aplicaciones que no utilice estas bibliotecas. En ese caso, podemos dejar la propiedad “signature.alternativeXmldsig” con el valor por defecto (false) y seguir los siguientes pasos:

1. En el fichero “etc/config.properties” de JBoss, se deben eliminar los paquetes “org.apache.xerces.\*” de la propiedad “org.osgi.framework.bootdelegation” para así evitar la carga de Xerces.
2. En el fichero “etc/jre.properties” se deben agregar las siguientes referencias para permitir su carga desde las bibliotecas de la JRE:
   * com.sun.org.apache.xerces.internal.dom
   * org.jcp.xml.dsig.internal.dom
3. Volcado de estadísticas a base de datos

FIRe permite la generación de datos estadísticos diarios en disco, que se pueden volcar en base de datos para su explotación desde el módulo de administración. En el caso de generarse los datos estadísticos, puede configurarse FIRe para que los vuelque automáticamente cada día a base de datos o para que nunca los vuelque, de tal forma que será el administrador del sistema (no necesariamente un administrador de FIRe) quien tenga que ejecutar activamente el proceso de volcado.

El comportamiento de FIRe se define mediante las propiedades statistics.policy, statistics.dir y statistics.dumptime del fichero de configuración “config.properties”. Consulte el apartado 4.3.1 Fichero config.properties para saber más acerca de estas propiedades.

Para aquellos casos en los que se configure que se generen los datos pero que no se haga el volcado automático a base de datos, se ha creado la aplicación FireStatisticsDbLoader.jar. Este ejecutable permite volcar en base de datos todos los ficheros de datos estadísticos de FIRe que no se hayan volcado ya.

En un directorio de datos estadísticos encontraremos los ficheros:

* FIRe\_STATISTICS.log
  + Fichero que indica la fecha de los datos que se volcaron por última vez en base de datos.
  + Si el fichero no existe, se entenderá que ninguno de los datos del directorio se ha volcado ya en base de datos.
* FIRE\_SIGNATURE\_YYYY-MM-DD.log
  + Fichero con la información de las firmas realizadas el día YYYY-MM-DD (fecha en formato año-mes-día).
  + Se generará un fichero de este tipo por cada día en el que se generen firmas con FIRe.
* FIRE\_TRANSACTION\_YYYY-MM-DD.log
  + Fichero con la información de las transacciones realizadas el día YYYY-MM-DD (fecha en formato año-mes-día).
  + Se generará un fichero de este tipo por cada día en el que se ejecuten operaciones con FIRe.

El ejecutable FireStatisticsDbLoader.jar leerá el fichero FIRE\_STATISTICS.log para comprobar los datos de hasta qué día se han volcado en base de datos y leerá y volcará los ficheros FIRE\_SIGNATURE\_YYYY-MM-DD.log y FIRE\_TRANSACTION\_YYYY-MM-DD.log de los días siguientes a ese.

Debe tenerse en cuenta que, según el sistema de gestión de base de datos utilizado, la aplicación requerirá un controlador JDBC específico para conectar con ella. Es por ello que este controlador debe estar disponible para que la herramienta pueda hacer el volcado. Para que la aplicación pueda encontrar el controlador JDBC necesario existen 2 opciones:

1. Incluir el controlador JDBC en algún directorio del *classpath* de la JRE utilizada o incluir el JAR de la JDBC en el *classpath* por medio de la llamada a la aplicación.
2. Copiar el JAR del controlador JDBC en el mismo directorio en el que se encuentra el ejecutable FireStatisticsDbLoader.jar y cambiarle el nombre a “jdbc.jar”.

La herramienta FireStatisticsDbLoader no tiene interfaz gráfica y debe ejecutarse por consola. Para saber cómo usarlo se puede ejecutar el JAR con el parámetro “-help”. Se deja aquí copia de la salida del comando:

|  |
| --- |
| Aplicacion para la carga inmediata de estadisticas de FIRe en base de datos.  Dependencias:  Esta aplicacion requiere de un controlador JDBC para conectar con su base de  datos. Configure su ClassPath para tener acceso al controlador desde esta  aplicacion o copielo al mismo directorio que este JAR con el nombre 'jdbc.jar'.  Uso:  java -jar FireStatictDbLoader.jar [Opciones]  Carga los datos usando las opciones proporcionadas.  java -Dfire.config.path="DIRECTORIO\_CONFIGURACION" -jar FireStatictDbLoader.jar  Carga los datos usando las propiedades definidas en el fichero  config.propeties que se encuentra en el directorio configurado.  java -cp "Ruta\_JDBC;FireStatictDbLoader.jar" es.gob.fire.statistics.cmd.StatisticsDBLoader [Opciones]  Carga los datos usando las propiedades definidas en el fichero  config.properties que se encuentra en el directorio configurado y  agregando al classpath ficheros JAR adicionales, entre los que se  puede encontrar el del controlador de base de datos.  Ruta\_JDBC:  Ruta al JAR controlador JDBC de nuestra base de datos.  Opciones:  -useConfigFile  Carga los datos en BD usando el fichero de configuracion de FIRe.  Si no se usa, se deben usar las opciones -dir, -driver y -conn.  -dir  Ruta absoluta del directorio con los ficheros de datos estadisticos.  -driver  Clase controladora JDBC para el acceso a la base de datos.  -conn  Cadena de conexion con la base de datos.  -processCurrentDay  El proceso de carga incluye los datos de hoy (por defecto, no se  hace). Esto provoca que despues no se puedan registrar los datos  generados el resto del dia.  Ejemplos:  java -jar FireStatictDbLoader.jar -dir "/usr/fire/statistics"  –driver oracle.jdbc.driver.OracleDriver  –conn jdbc:oracle:thin:Fire/1111@12.34.56.78:1521:FIRE\_DB  java -Dfire.config.path="/usr/fire/config" -jar FireStatictDbLoader.jar |