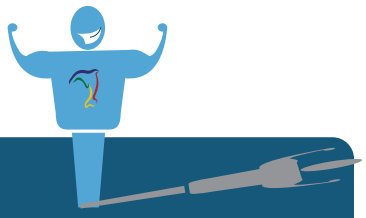
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

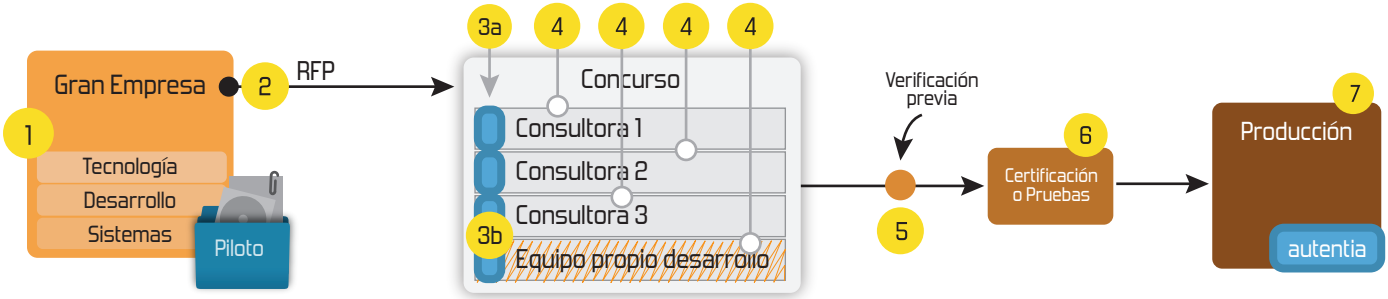
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Hosting Patrocinado por enREDados.com

Tutorial desarrollado por: Alejandro Perez García 2003-2005, nuestro experto en J2EE, Linux y optimización de aplicaciones empresariales.

Si te gusta lo que ves, **puedes contratarme** para impartir **cursos presenciales** en tu empresa o para ayudarte en proyectos (Madrid).

Contacta: alejandropg@autentia.com.

autentia real business solutions

Descargar este documento en formato PDF apache_secure_debian.pdf

Oferta Alarma Hogar
Instale su Alarma este mes por 79€ Llámenos Gratis ¡ahora mismo!
www.securitasdirect.es

Free SSL Certificates
Issued in mins installed in seconds Low price single root 128/256 bit
www.rapidssl.com

Anuncios Goooooogle Anunciarse en este sitio

Como configurar la seguridad del servidor Web Apache en GNU / Linux (Debian)

Fecha: 05-04-2004

Versión: 1.0

Índice de contenidos

1. Introducción

La seguridad, ya sea en Apache o en cualquier otro servidor, es un tema bastante amplio y muy serio. Así que este tutorial sólo pretende ser una introducción a como podemos configurar la seguridad en nuestro servidor Web Apache.

En este tutorial se verá una forma de configurar la seguridad, pero hay muchas otras. Cada uno debe buscar la que más se ajuste a sus necesidades.

Se recomienda encarecidamente leer la documentación que proporciona Apache en: http://httpd.apache.org/docs-2.0/ssl/.

2. Entorno

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil Ahtex Signal X-9500M (Centrino 1.6 GHz, 1024 MB RAM, 60 GB HD).

- Sistema Operativo: GNU / Linux, Debian Sid (unstable), Kernel 2.6.4, KDE 3.2
- Apache 2.0.48 instalado en /usr/local/apache2
- PHP 4.3.4 instalado en /usr/local/php4
- MySQL 4.1.1 instalado en /usr/local/mysql
- openssl 0.9.7d-1

Para saber como se instaló el entorno (MySQL, Apache, PHP) ver el tutorial 'Como instalar MySQL, Apache y PHP en GNU / Linux (Debian)'

Supondremos que todos los archivos que descargamos de Internet los guardaremos en /download.

3. Comprobación del entorno

Antes de empezar debemos comprobar que nuestro Apache está compilado con los módulos de seguridad que vamos a utilizar.

Para ver listar los módulos podemos ejecutar:

```
# cd /usr/local/apache2/bin
# ./httpd -l
Compiled in modules:
  core.c
  mod_access.c
  mod_auth.c
  mod_auth_digest.c
  mod_include.c
  mod_log_config.c
  mod_env.c
  mod_setenvif.c
  mod_ssl.c
  prefork.c
  http_core.c
  mod_mime.c
  mod_status.c
  mod_autoindex.c
  mod_asis.c
  mod_cgi.c
  mod_negotiation.c
  mod_dir.c
  mod_imap.c
  mod_actions.c
  mod_userdir.c
  mod_alias.c
  mod_so.c
```

En la lista deben aparecer:

- mod_auth.c: para autenticación básica.
- mod_auth_digest.c: para autenticación con el método digest (usuario y password encriptados).
- mod_ssl.c: para poder activar SSL (https).

mod_auth.c debería aparecer, ya que se trata de una opción por defecto. Si los otros dos módulos no aparecen tendremos que recompilar Apache.

Para compilar Apache con soporte SSL es necesario tener instalado los fuentes de openssl, ya que al compilar se necesitan los ficheros de cabecera (.h). También necesitamos tener instalado openssl para poder trabajar con los certificados.

Para instalar ambos paquetes basta con ejecutar:

```
# apt-get install openssl libssl-dev
```

Ahora ya estamos listos para recompilar nuestro Apache. Nos situamos en el directorio donde tentemos los fuentes de Apache y usamos los siguientes comando:

```
# ./configure --prefix=/usr/local/apache2 --with-mpm=prefork --enable-so --enable-auth-digest
--enable-ssl
# make
# make install
```

4. Autenticación HTTP

Podemos configurar Apache de forma que para acceder a cierto directorio (y subdirectorios) sea necesario introducir un usuario y una clave.

Dentro de esta posibilidad vamos a ver dos método:

- Basic: cuando en el cliente se introduce el usuario y la clave, estos viajan al servidor sin cifrar.
- Digest: el usuario y la clave van cifrados del cliente al servidor.

Estos métodos tienen la ventaja de que es Apache quien se encarga de comprobar las correctas credenciales del cliente, por lo que no tenemos que hacer ningún tipo de comprobación en nuestra aplicación.

Hay que tener en cuenta que estos dos métodos sólo sirven para autenticar a un usuario cuando intenta acceder a un determinado recurso. Es decir, Apache identifica si se trata de un usuario válido, y en tal caso le deja acceder al recurso. Pero los datos que posteriormente se envíen de cliente al servidor, o viceversa, no tienen ningún tipo de cifrado. Estos métodos sólo sirven para controlar el acceso, no para proteger los datos una vez se ha comprobado que el acceso es válido.

4.1. Autenticación HTTP Basic

Para este modo de autenticación se utiliza el módulo mod_auth.

Este método tiene la ventaja de que lo soportan todos los navegadores, pero tiene la desventaja de que el nombre de usuario y la clave no van cifrados del cliente al servidor. Esto hace que no sea un método recomendado para entornos donde la seguridad es clave.

4.1.1. Configuración de httpd.conf

En el fichero /usr/local/apache2/conf/httpd.conf basta con añadir las siguientes líneas:

```
<Directory "/usr/local/apache2/htdocs/private">
   AuthName "privatefiles"
   AuthType Basic
   AuthUserFile /usr/local/apache2/conf/passwd_basic
   Require valid-user
</Directory>
```

Veamos cada una de las directivas:

- Directory: estamos diciendo a Apache que las directivas que vienen a continuación tiene efecto sobre el directorio /usr/local/apache2/htdocs/private, y sus subdirectorios. Es decir, que vamos a proteger este directorio y sus subdirectorios.
- AuthName: es el nombre del dominio de autenticación. También es el texto que aparecerá en la ventana que pide el el usuario y la clave.

AuthType: el tipo de autenticación.

- AuthUserFile: el fichero donde están los usuarios y las claves.
- Require: con esta directiva indicamos que usuarios tienen acceso, tenemos varias posibilidades:
  - valid-user: cualquier usuario que esté en el fichero de claves.
  - user <lista de usurios>: lista de usuarios, separados por espacios, que pueden acceder.
  - group <lista de grupos>: lista de grupos, separados por espacios, que pueden acceder. Ojo, porque si usamos esta opción también necesitamos usar la directiva AuthGroupFile para indicar donde se encuentra el fichero con la definición de los grupos.

4.1.2. Creación de un usuario

Para crear los usuarios para el método de autenticación 'Basic', usaremos la aplicación htpasswd. Por ejemplo:

```
# cd /usr/local/apache2/bin
# ./htpasswd /usr/local/apache2/conf/passwd_basic admin
```

De esta forma estamos añadiendo el usuario 'admin' al fichero de claves /usr/local/apache2/conf/passwd_basic. El programa nos pedirá la clave y luego nos la vuelve a preguntar para confirmarla.

Si el fichero no existe (para la primera vez), es necesario lanzar el comando con la opción -c:

```
# ./htpasswd -c /usr/local/apache2/conf/passwd_basic admin
```

También podemos especificar la clave en la linea de comandos, a continuación del nombre. En este caso tendremos que utilizar la opción -b:

```
# ./htpasswd -b /usr/local/apache2/conf/passwd_basic admin laclave
```

Si ejecutamos el comando sin argumentos, obtendremos la ayuda de las posibles opciones.

4.1.3. Creación de un grupo

Para especificar los grupos basta con crear un fichero de texto con el siguiente formato para cada línea:

```
nombreGrupo: user1 user2 user3 ...
```

Luego basta con usar la directiva AuthGroupFile para indicar la ruta completa donde se encuentra el fichero que hemos creado con la definición de los grupos.

Cada usuario del grupo lo añadiremos al fichero de claves tal y como se describió en el apartado anterior.

4.2. Autenticación HTTP Digest

Para este modo de autenticación se utiliza el módulo mod_auth_digest.

Este método tiene la gran ventaja de que el usuario y la clave van cifradas del cliente al servidor. Pero tiene el inconveniente de que podemos encontrarnos con versiones antiguas de navegadores, que no lo soporten.

4.2.1. Configuración de httpd.conf

En el fichero /usr/local/apache2/conf/httpd.conf basta con añadir las siguientes líneas:

```
<Directory "/usr/local/apache2/htdocs/private">
   AuthName "privatefiles"
   AuthType Digest
   AuthDigestFile /usr/local/apache2/conf/passwd_digest
   Require valid-user
</Directory>
```

Vemos que es prácticamente igual que cuando usábamos el método Basic. Hemos cambiado:

AuthName: esta vez indicamos que el método a usar es 'Digest'.

AuthDigestFile: esta directiva sustituye a AuthUserFile, pero tiene la misma finalidad: especificar donde se encuentra el fichero de claves. Hay que tener presente que el fichero de claves tiene que ser distinto que el usamos con el método Basic, ya que tiene formatos distintos.

4.2.2. Creación de un usuario

Para crear los usuarios para el método de autenticación 'Digest', usaremos la aplicación htdigest. Por ejemplo:

```
# cd /usr/local/apache2/bin
# ./htdigest /usr/local/apache2/conf/passwd_digest privatefiles admin
```

De esta forma estamos añadiendo el usuario 'admin' al fichero de claves /usr/local/apache2/conf/passwd_digest. El programa nos pedirá la clave y luego nos la vuelve a preguntar para confirmarla.

Vemos que en la línea de comandos hemos introducido como segundo parámetro 'privatefiles'. Este parámetro debe coincidir exactamente con el nombre del dominio de autenticación que hemos especificamos con la directiva AuthName. Es decir, cuando añadimos un usuario, lo hacemos a un dominio de autenticación concreto. De forma que si con la directiva Require especificamos 'valid-user', se consideran sólo usuarios válidos los que pertenecen al dominio de autenticación especificado en AuthName.

Si el fichero no existe (para la primera vez), es necesario lanzar el comando con la opción -c:

```
# ./htpasswd -c /usr/local/apache2/conf/passwd_digest privatefiles admin
```

Si ejecutamos el comando sin argumentos, obtendremos la ayuda de las posibles opciones.

4.3. Logout automático

El único inconveniente que tiene la autenticación por HTTP es que no podemos conseguir un logout automático, pasado un determinado periodo de tiempo.

Esto es debido a que una vez nos hemos autenticado, el navegador cachea la credencial de forma que aunque fijemos un periodo de validez, una vez transcurrido este, si el navegador intenta acceder a otro recurso de la zona protegida, el servidor le devolverá un error 401 (Authorization Required), y el navegador automáticamente volverá a mandar las credenciales cacheadas al servidor.

La única forma que tenemos para hacer logout es:

- Cerrar el navegador.
- Sobreescribir las credenciales. Para esto podemos crear un identificador de usuario válido, pero sin privilegios, y poner una URL de logout a la que sólo puede acceder este usuario. Esta URL se convierte en un botón de logout.

En cualquiera de estos dos casos hay que instruir y convencer al usuario para que los use.

Si para nuestra aplicación es vital esta funcionalidad, necesitamos manejar la sesión, e implementar nuestro propio sistema de login.

5. SSL

Trabajar con SSL nos permite que todos los datos que se transfieren entre el cliente y el servidor vayan cifrados.

Antes de hablar más sobre SSL vamos a definir algunos términos:

- RSA Private Keys: fichero digital que podemos usar para descifras mensajes que nos mandan. Tiene una parte pública (que distribuimos con nuestro certificado), que permite a la gente cifrar los mensajes que nos manda. Este mecanismo de clave asimétrica nos asegura que los mensajes cifrados con la clave pública (que distribuimos a mucha gente) sólo pueden ser descifrados con la clave privada (que sólo conocemos nosotros).
- Certificate Signing Request (CSR): es un fichero digital que contiene nuestra clave pública y nuestro nombre.
- Certification Authority (CA): entidad de confianza encargada de firmar certificados (CSR).
- Certificate (CRT): Una vez la CA a firmado el CSR, obtenemos un CRT. Este fichero contiene nuestra clave pública, nuestro nombre, el nombre de al CA, y está firmado digitalmente por la CA. De esta forma otras entidades pueden verificar esta firma para comprobar la veracidad del certificado. Es decir, si obtenemos un certificado que esta firmado por una CA que nosotros consideramos de confianza, podemos confiar también en la autenticidad del certificado.

Ahora que ya tenemos un poco más claros los conceptos, podemos ver que hay varias posibilidades para configurar SSL. Por ejemplo:

- Cualquier cliente puede conectarse a una URL determinada, usando https. En este caso el servidor enviará su certificado al cliente para que este pueda descifrar la información que le llega del servidor y cifrar la que envía hacia el servidor.
- También podríamos hacer que sólo los clientes que tengan un determinado certificado puedan conectarse a una determinada URL.
- Otra posibilidad (la que vamos a ver en este tutorial) es combinar el primer ejemplo con las técnicas de autenticación que hemos visto antes. De forma que cuando intentemos acceder a una determinada URL usando https tendremos que autenticarnos primero.

5.1. Creando nuestra propia CA

Existen varias CAs que, previo pago, pueden firmar nuestro CSR. Estas CAs son mundialmente conocidas de forma que cualquier cliente podrá conectarse con confianza a nuestro servidor.

Un ejemplo de este tipo de entidades de certificación puede ser VISA o VeriSign.

En nuestro caso, como estamos probando, nos crearemos nuestra propia CA.

Para facilitar este tipo de tareas el paquete openssl nos proporciona el script CA.sh (o CA.pl en Perl). Ejecutaremos:

```
# cd /usr/local/apache2
# /usr/lib/ssl/misc/CA.sh -newca
```

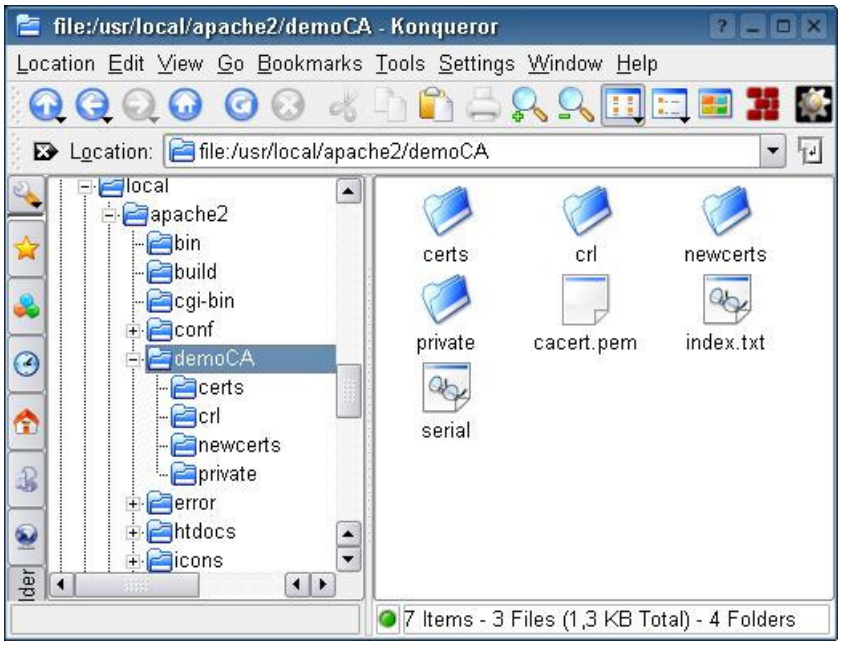
Nos hará las siguientes preguntas:

1. Nombre del certificado de la CA o que pulsemos 'enter' para crearlo. En nuestro caso pulsamos 'enter' para crear uno nuevo.

2. Una 'pass phrase', que nos vuelve a preguntar para confirmarla. Esta será la clave para acceder a la clave privada (la clave privada se guarda cifrada). Deberíamos poner más de una palabra.

3. Cierta información para añadir al certificado: código del país, provincia, localidad, nombre de la organización, de la unidad, ...

Una vez finaliza la ejecución del script podemos ver que en el directorio /usr/local/apache2 nos ha aparecido un nuevo directorio: demoCA. Este tiene la siguiente estructura:



El fichero cacer.pem es el certificado de la CA, que luego usaremos para firmar nuestro certificado.

Los subdirectorios están vacíos, a excepción de private, donde encontramos el fichero cakey.pem. Este fichero es la clave de la CA.

5.2. Creamos nuestro CSR

Ahora vamos a crear el CSR que debería firmar una auténtica CA (nosotros lo firmaremos con nuestra propia CA, la que hicimos en el apartado anterior). Para esto seguimos los siguientes pasos:

1. Creamos la clave para nuestro servidor Apache (la clave será triple-DES y en formato PEM):

   ```
   # openssl genrsa -des3 -out server.key 1024
   ```

   El comando nos pedirá un 'pass phrase' y nos la vuelve a preguntar para confirmarla.

2. Creamos el CSR (estará en formato PEM), usando la clave generada en el punto anterior:

   ```
   # openssl req -new -key server.key -out server.csr
   ```

   Lo primero que hará será pedirnos la 'pass phrase' que le pusimos a la clave en el punto anterior. Luego nos pedirá los datos que queremos añadir a nuestro certificado. Aquí es importante tener en cuenta que cuando nos pregunte por el 'Common Name' debemos poner el nombre completo del dominio del servidor, por ejemplo, si vamos a acceder usando https://www.foo.dom/, tendremos que poner www.foo.dom.

Con esto hemos conseguido generar el fichero server.csr.

5.3. Creamos nuestro CRT

El último paso es firmar el CSR para conseguir el CRT. Para esto volveremos a usar el script CA.sh. El problema que tiene este script es que trabaja con unos nombres fijos de ficheros, así que antes de ejecutarlo tenemos que renombrar el fichero server.csr (en mi caso he preferido hacer un enlace simbólico).

```
# ln -s server.csr newreq.pem
# CA.sh -signreq
```

Nos pedirá la 'pass phrase' de la clave que hemos usado para generar el certificado de la CA (ver apartado 5.1. paso 2).

Nos mostrará la información de nuestro certificado, y nos preguntará si queremos firmar el certificado. Por supuesto, contestamos que sí.

Nos preguntará si queremos hacer 'comit' de los certificados firmados, es decir si queremos confirmar la operación. De nuevo, contestamos que sí.

Nos habrá generado el fichero newcert.pem. Este fichero lo renombramos por server.crt.

```
# mv newcert.pem server.crt
```

5.4. Configuramos Apache

- Lo primero es poner los ficheros donde corresponde.

```
# mkdir conf/ssl.key
# mkdir conf/ssl.crt
# mv server.key conf/ssl.key/
# mv server.crt conf/ssl.crt/
```

- Ahora añadimos al fichero /usr/local/apache2/conf/httpd.conf:

```
<Directory "/usr/local/apache2/htdocs/private">
   SSLRequireSSL
   AuthName "privatefiles"
   AuthType Basic
   AuthUserFile /usr/local/apache2/conf/passwd_basic
   Require valid-user
</Directory>
```

  Vemos que es la misma configuración que usamos en el apartado 4.1. salvo que hemos añadido la directiva SSLRequireSSL. Esta directiva prohíbe el acceso a no ser que sea HTTP sobre SSL (HTTPS).

  Con esto conseguimos que para acceder al directorio /usr/local/apache2/htdocs/private sea obligatoriamente usando HTTPS, y previa autenticación.

- Por último sólo nos queda arrancar Apache. Hay que tener en cuenta que para arrancar Apache con soporte SSL hay que utilizar:

```
# /usr/local/apache2/bin/apachectl startssl
```

Vemos que al arrancar Apache nos pide una 'pass phrase'. Esta es la que usamos al crear nuestro certificado (ver apartado 5.2. punto 1.). Esto lo hace porque la clave esta guardada cifrada.

Si queremos evitar tener que escribir la 'pass phrase' cada vez que arranquemos Apache podemos guardar la clave sin cifrar, pero ojo, esto no es nada recomendable, sobre todo en producción. Podemos tener un grave problema de seguridad.

Para generar una clave sin cifrar podemos hacer.

```
# cd /usr/local/apache2/conf/ssl.key/
# openssl rsa -in server.key -out server.unencrypted.key
```

Luego es conveniente restringir los permisos al máximo.

```
# chmod 400 server.unencrypted.key
```

Otra manera para no tener que escribir la 'pass phrase' es usar la directiva SSLPassPhraseDialog exec: ... Donde cambiaremos los puntos suspensivos por un programa que saque por la salida estándar la 'pass phrase'.

Hay que tener en cuenta que este método no tiene por que ser más seguro que el anterior. En general no recomendaría ninguno de los dos para un sistema en producción.

5.5. Limpia de archivos

Los ficheros que necesita Apache sólo son:

- Clave: /usr/local/apache2/conf/ssl.key/server.key.
- Certificado firmado por la CA: /usr/local/apache2/conf/ssl.crt/server.crt.

Así que si queremos podemos borrar el resto de los archivos que hemos generado para firmar nuestro certificado.

```
# cd /usr/local/apache2/
# rm server.csr newreq.pem
# rm -Rf demoCA/
```

6. Conclusiones

Con esto hemos terminado este tutorial.

Espero que os sirva como introducción para configurar, de forma sencilla, sitios seguros dentro de vuestro Apache.

Y vuelvo a recomendaros que os leáis la documentación que proporciona el propio Apache para informaros bien sobre todas las posibilidades y directivas de configuración que están a vuestro alcance.

7. Sobre el autor

Alejandro Pérez García

Dir. Implantación y Rendimiento

mailto:alejandropg@autentia.com

Autentia Real Business Solutions S.L.

http://www.autentia.com

Si desea contratar formación, consultoria o desarrollo de piezas a medida puede contactar con

Gestión de contenidos

Autentia S.L. Somos expertos en:
J2EE, C++, OOP, UML, Vignette, Creatividad ..
y muchas otras cosas

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
Acceso seguro a CVS a través de SSH	Os mostramos como segurizar los accesos a CVS a través de SSH, utilizando herramientas gratuitas
Protege tu PC	Os mostramos como proteger tu máquina de ataques mientras estés conectado a una red o Internet
Administrar UNIX mediante Webmin	En este artículo os mostramos como administrar una máquina UNIX, mediante un interfaz web, con Webmin
Activar SSL en IIS	Os mostramos comos activar el soporte de https en IIS, creando vuestros propios certificados autofirmados, usando OpenSSL
Instalar un Web en tu máquina Windows	Si trabajais habitualmente en plataforma Windows, en este tutorial podreis encontrar todos los pasos necesarios para instalar un servidor Web en vuestra propia máquina. Esto puede ser el primer paso para crear una Intranet.
Seguridad en Tomcat	Os mostramos como proteger de un modo básico el acceso a recursos dentro de vuestro servidor de componentes Tomcat
Compartir impresoras y ficheros con Linux	Cesar Crespo Martín y Alejandro Perez García nos enseñan como realizar la compartición de impresoras y ficheros con Linux, CUPS y SAMBA con clientes Windows.
Construir un Servidor Web en Java	En este tutorial os enseñamos los principios de las aplicaciones multi-hilo a través de la creación de un servidor web básico en Java. Podremos ver en un ejemplo real el uso de sockets, threads, excepciones, etc.
MySql en Windows	MySql es una de las principales bases de datos "gratuitas" que podemos encontrar en Internet. En este tutorial aprendereis a instalarlo en Windows
Activar soporte SSL en Tomcat	Os mostramos como activar el acceso SSL en Tomcat, utilizando certificados generados por Keygen (java)

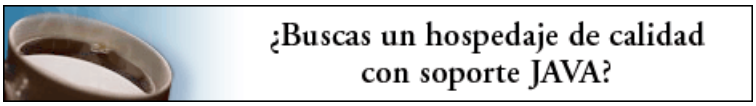
Nota: Los tutoriales mostrados en este Web tienen como objetivo la difusión del conocimiento.

Los contenidos y comentarios de los tutoriales son responsabilidad de sus respectivos autores.

En algún caso se puede hacer referencia a marcas o nombres cuya propiedad y derechos es de sus respectivos dueños. Si algún afectado desea que incorporemos alguna reseña específica, no tiene más que solicitarlo.

Si alguien encuentra algún problema con la información publicada en este Web, rogamos que informe al administrador rcanales@adictosaltrabajo.com para su resolución.

Patrocinados por enredados.com .... Hosting en Castellano con soporte Java/J2EE



www.AdictosAlTrabajo.com Opimizado 800X600