4. Web Server Administration

Miquel Àngel París i Peñaranda

Web Application Deployment

2nd C-VET Web Application Development

Index of contents

1. Goals.

1. Install and configure web servers on virtual machines and/or the cloud.
2. Perform functional tests on web and application servers.
3. Document the installation and configuration processes performed on web and application servers.

"Apache Administrator's Survival Manual" Author: Miguel Jaque Barbero (November 10, 2006)

Updated: Miquel Àngel París Peñaranda (November 2024) DAW2 – Web Application Deployment

2. Introduction.

The Apache web server is the most popular on the Internet and therefore the one we will use to implement this service on a test server.

2.1 Installation

With Debian/Ubuntu, installing Apache is very simple. Simply execute the order:

```
$ apt install apache2
```

And you'll have the web server installed.

2.2. Configuration

The Debian / Ubuntu configuration, by default, includes:

The Apache configuration files in `/etc/apache2`

The root directory for documents in `/var/www`

2.3. Start and Stop

Apache runs as a background service (daemon). To start it, just execute one of the commands:

`$ service apache2 start` or `$ systemctl start apache2`

And to stop it: `$ service apache2 stop` or $ systemctl stop apache2

Taking a look at the script (`/etc/init.d/apache2`) you will see that the command that actually starts and stops Apache is `apache2ctl`. Checking the man page (`man apache2ctl`) you will see that it has many more options than start and stop.

We can also restart the service if it is already in operation and we want it to load new modules, for example:

`$ service apache2 restart` or `$ systemctl restart apache2`

If we have enabled new content to make it accessible, we can reload it with:

`$ service apache2 reload` or `$ systemctl reload apache2`

Finally, we can see if the server is up and running with:

`$ service apache2 status` or `$ systemctl status apache2`

We can validate the configuration files with: `$ apachectl configtest`

2.4. Processes

Although Apache initially runs as root, on a Linux system only root is allowed to open sockets on ports below 1000, and Apache listens, by default, on the traditional **port 80 of the HTTP protocol**.

Starting Apache will start the root process other child processes that are already running with another user and group that is less dangerous than root. By default, these child processes run with the **www-data** user and the **www-data** group. Make sure that this user has permission to read the documents that Apache should serve.

The initial process will not address any client requests to avoid security issues. Instead, it will be the child processes that are responsible for serving the pages.

You can check all of this with the `$ ps -aux` command to see the processes running on your system.

3. Basic Server

The Apache configuration file that Debian/Ubuntu installs by default (`/etc/apache2/apache2.conf`), although it is very good, is also very complicated and does not help us to explain the most elementary concepts of Apache, also in the latest versions some parameters of the basic configuration have been established in configuration files, in the configuration of the modules that apply them or in the configuration files of the Web.

So let's change it (saving a backup first) for this one:

```
# SIMPLE Configuration File (/etc/apache2/apache2.conf)
# Name by which the Server knows itself
ServerName "wad.2daw.iesjc"
# Directory with Apache configuration files
ServerRoot "/etc/apache2"
# Directory of published documents
DocumentRoot "/var/www"
# File where the Apache process number is saved
PidFile /var/run/apache2/apache2.pid
#User and group that Apache will run with
User www-data
Group www-data
# Log File for Errors
ErrorLog /var/log/apache2/error.log
# Apache Listen Port
Listen 80
# Include the enabled modules on our server
IncludeOptional mods-enabled/*.load
IncludeOptional mods-enabled/*.conf
IncludeOptional conf-enabled/*.conf
# Include the config. ports on which the server listens
Include ports.conf
# Include the websites enabled on our server
IncludeOptional sites-enabled/*.conf File 
/etc/apache2/mods-enabled/dir.conf
# List of files that can serve as directory indexes
DirectoryIndex index.html index.htm
# File with the MIME type list
TypesConfig /etc/mime.types
```

In the previous content:

1. The configuration files are plain text.

2. Each line contains a configuration policy.

3. Lines beginning with # are comments.

3.1. Basic Directives

- **ServerName**

  Sets the name by which the server knows itself. This directive is used for redirects. That is, when Apache has to indicate to the client (the browser) another address to which it has to go.

  It is important that this name can be resolved by DNS. Otherwise, the customer will not be able to access the redirected page.

- **ServerRoot**

  Sets the directory in which the Apache configuration files are located. And if Apache doesn't know the configuration directory, how can it access this configuration file? The answer to this paradox is that Apache can be booted by passing it a configuration file as a parameter. But it may happen that in the configuration file we refer to other files that must be included.

- **DocumentRoot**

  Sets the directory in which the files that Apache will serve to clients are located (HTML pages, PHP scripts, CGIs, etc.). Each virtual server has its directory defined with this policy.

- **PidFile**

  Sets the file in which the Apache process number will be saved. This file is the one that is read when the process has to be stopped/killed.

- **User**

  Sets the user with which Apache will run. Well, actually the Apache process runs as root, because it usually has to open a listening socket for port 80 and, in POSIX, only root can open ports below 1000. However, after starting that first process as root, Apache creates several child processes that run with the user set in this directive (www-data in Debian). These processes will be the ones who will really attend to the requests of the users.

- **Group**

  Sets the pool that Apache will run with (its listening processes).

- **ErrorLog**

  Sets the Apache error log file. This file will record access failures, attempts to access resources without authorization, pages not found, etc.

- **Listen**

  Set the IP address and port on which Apache will listen. By default, Apache will listen on all enabled IP addresses on the machine. It is defined in the ports.conf file.

- **DirectoryIndex**

  Sets the names of files that will serve as indexes when accessing a directory without specifying any specific resources. Thus, if a user requests www.myweb.org/dir1/ Apache will search that directory for files with the name indicated in this directive to serve them.

- **TypesConfig**

  Set the file with the Mime type list. Mime types are a standard that relates file types to their extensions and allows Apache to inform the browser of the type of file it is delivering. Thus, the browser decides how to present it (displaying a web page, running a plugin, saving it to disk...)

3.2. Activate web space for system users

Apache has a module that allows users of the system to have their website on the server, but with the advantage that the space available to host this page is within the personal directory of each user, so that each one can manage their own page without having to have more privileges. To enable the module you have to run:

```
$ a2enmod userdir
```

Once the module is enabled, any user can put their website inside the directory public_html in their home directory (/home/user_name/public_html).

Normally this folder will not be created, but it can be created by the user himself, he just has to take into account to give him permissions to make it accessible. It would be best to give the 755 permissions so that the user could make modifications and the rest could only access to see the stored objects.

The objects inside this folder that will make up the web page (html documents, images, videos...) should have the 644 permissions because, unlike the directory that contains them, they do not need the execute permission. Obviously if you have subdirectories you will also need this permission.

Like the rest of the locations, Apache will try to display the page called index, so if this page exists, to display it, you just have to put the URL in the browser:

```
http://server_address/~user_name
```

3.3. Directory Configured Server

On the simple server we just saw, the settings are the same for all the documents we're publishing. In some cases we may want to have different configurations for different directories and even for different files. To do this, we can use a configuration file like the following:

```
# BLOCK CONFIGURATION File
ServerName "wad.2daw.iesjc"
ServerRoot "/etc/apache2"
DocumentRoot "/var/www"
PidFile /var/run/apache2/apache2.pid
User www-data
Group www-data
ErrorLog /var/log/apache2/error.log
Listen 80
TypesConfig /etc/mime.types
# set the configuration of each directory
<Directory "/var/www">
      # Don't allow automatic indexing anywhere (except explicitly 
      allowed)
      Options -Indexes
```

```
</Directory>
<Directory "/var/www/downloads">
      # In this directory, we allow automatic indexes, but we do not
      allow symbolic links
      Options +Indexes -FollowSymLinks
      # Don't allow anyone to access the .htaccess files in this
      directory
      <Files .htaccess>
            order allow,deny
            deny from all
      </Files>
</Directory>
<Location /server-status>
      # In this address, we display information about the status of the
      server
      SetHandler server-status
</Location>
```

3.3.1. Policies for Block Configuration

When applying different configurations by directory, by file or by location, we say that we are applying "Block Configuration". We have used the following directives:

- **<Directory>**

  Indicates that the configuration block it covers (between <Directory> and </Directory>) applies to the specified directory and its subdirectories. There is also a <DirectoryMatch> directive that allows regular expressions to be used. Thus, the same configuration block can be applied to several directories.

- **Options**

  Modifies the options that apply to a directory. The options are added (with +) or removed (with -) with respect to those that are currently applied to the directory.

  The following options can be added/removed using this policy: All, ExecCGI, FollowSymLinks, Includes, IncludesNOEXEC, Indexes, Multiviews, and SymLinksIfOwnerMatch.

- **<File>**

  Indicates that the configuration block it covers (between <File> and </File>) will be applied to files whose name matches the name indicated.

There is also a <FileMatch> directive that allows you to use regular expressions. Thus, the same configuration block can be applied to several files.

- **<Location>**

  Indicates that the configuration block it spans (between <Location> and </Location>) will be applied to locations that match the one listed. The "location" is the address requested by the customer. Note that it is not the same as the file system (on which <Directory> and <File> work). Using redirects and aliases, it is possible for a client to request a certain "location" and for the page delivered to have a completely different path than the one he specified. There is also a <LocationMatch> directive that allows you to use regular expressions. Thus, the same configuration block can be applied to several locations.

- **SetHandler**

  Sets the handler that will be used to serve requests to a directory, file type, or location. A handler is an internal Apache representation of an action that is to be executed when there is a file call. Generally, files have implicit handlers, based on the type of file in question. Normally, all files are simply served by the server, but some types of files are treated differently. The possible handlers are: default-handler, send-as-is, cgi-script, imap-file, server-info, server-status and type-map.

4. Virtual Servers

Having an entire Apache server to serve just one website is a waste of resources. Apache is capable of serving from a single machine to a whole set of websites. That is, we can serve at the same time requests for www.depinfo.iesjc , www.depadm.iesjc, teachers.iesjc , students.iesjc...

This is done using "Virtual Servers".

4.1. Architecture Options

To serve multiple websites, we must first get client requests for those URLs to our server. This is a DNS configuration issue. You can hava a look at the following link How to Set Up Virtual Hosts on Apache2 and Create a DNS Zone Using Bind9 - HubPages

Another option would be for the teacher to configure the DHCP + DNS service of the classroom server so that VMs in bridge mode can be configured automatically and their name is registered in the DNS service, so that classmates' websites can be accessed.

In the case of DNSMASQ, /etc/dnsmasq.conf will need to be configured:

- Uncomment the DHCP part line: dhcp-ignore-names, so that it takes the names of the client systems and uses them to add them to DNS.
- Secondly, if we want the client systems to have more than one name, we must also add in /etc/dnsmasq.conf the CNAME statements for each client so that we can create at least two Apache virtual servers using the hostname. The syntax to be used by each client system will be: `cname = alias_name,system_name` only one alia per line.

Once this is done, we will have several options to configure our virtual hosts. First, we can set multiple IP addresses and assign one to each virtual host. We will call this "virtual servers over IP".

Second, we can set different listening ports for website. Yes, this is somewhat complicated because it involves telling the customer which port they should go to. But it can be useful for internal networks. We will call this "Virtual Servers per Port".

And thirdly, the HTTP 1.1 protocol allows the client to indicate to us, by means of a header, the name of the website they want to access. As there are hardly any browsers that do not support the HTTP 1.1 protocol, this is the most used option.

And finally, we can do a mix with all of these options.

4.2. Virtual Servers over IP

Let's first assume that our server serves two IP addresses and that we assign each of them to a website.

The configuration file would be the following and would be located in `/etc/apache2/sites-available/web_name.conf`:

```
# Configuration #Fichlero for VIRTUAL HOSTS OVER IP
ServerRoot "/etc/apache2"
DocumentRoot "/var/www"
PidFile /var/run/apache2/apache2.pid
User www-data
Group www-data
ErrorLog /var/log/apache2/error.log
TypesConfig /etc/mime.types

DirectoryIndex index.html index.htm

# EstablecemosSetting up the configuration for each Virtual Host
<VirtualHost 192.168.2.166>
      # EachCada one has its own name
      ServerName www.depinfo.iesjc
      DocumentRoot /var/www/depinfo

      # SplittingDividimos log files
      ErrorLog /tmp/www1_ERROR.log
      TransferLog /tmp/www1_ACCESS.log
</VirtualHost>

<VirtualHost 192.168.2.200>
      ServerName www2.depinfo.iesjc
      DocumentRoot /var/www/depinfo2
      ErrorLog /tmp/www2_ERROR.log
      TransferLog /tmp/www2_ACCESS.log
</VirtualHost>
```

4.3. Virtual Servers by Port

Let's do the same thing, but on one of the IP addresses, let's use two different TCP/IP ports to serve two different sites.

In the file `/etc/apache2/ports.conf` we must add: `Listen 8080` so that the server also listens on this new port.

The configuration file would be as follows, with the same location as in the previous case:

```
# Configuration Filerfor VIRTUAL HOSTS OVER IP AND PORT
ServerRoot "/etc/apache2"
DocumentRoot "/var/www"
PidFile /var/run/apache2/apache2.pid
User www-data
Group www-data
ErrorLog /var/log/apache2/error.log
TypesConfig /etc/mime.types

# Indicating the port of each website
<VirtualHost 192.168.2.166:80>
      ServerName teachers.depinfo.iesjc
      DocumentRoot /var/www/teachers
      ErrorLog /tmp/teachers_ERROR.log
      TransferLog /tmp/teachers_ACCESS.log
</VirtualHost>

<VirtualHost 192.168.2.166:8080>
      ServerName students.depinfo.iesjc
      DocumentRoot /var/www/students
      ErrorLog /tmp/students_ERROR.log
      TransferLog /tmp/students_ACCESS.log
</VirtualHost>

<VirtualHost 192.168.2.200>
      ServerName www.depinfo.iesjc
      DocumentRoot /var/www/depinfo
      ErrorLog /tmp/www_ERROR.log
      TransferLog /tmp/www_ACCESS.log
</VirtualHost>
```

4.4. Virtual Servers by Name

And now, let's set up the server using the names of each website:

The configuration file would be as follows, with the same location as in the previous case:

```
# Configuration File for VIRTUAL HOSTS BY NAME
ServerRoot "/etc/apache2"
DocumentRoot "/var/www"
PidFile /var/run/apache2/apache2.pid
User www-data
Group www-data
ErrorLog /var/log/apache2/error.log
TypesConfig /etc/mime.types

# Configuration of virtual hosts by name applies to
# request received on this IP
NameVirtualHost 192.168.2.166

# Configuration referring to each site by name
<VirtualHost www.depinfo.iesjc>
      ServerName www.depinfo.iesjc
      DocumentRoot/var/www/depinfo
      ErrorLog /tmp/www1_ERROR.log
      TransferLog /tmp/www1_ACCESS.log
</VirtualHost>

<VirtualHost www2.depinfo.iesjc>
      ServerName www2.depinfo.iesjc
      DocumentRoot /var/www/depinfo2
      ErrorLog /tmp/www2_ERROR.log
      TransferLog /tmp/www2_ACCESS.log
</VirtualHost>
```

4.5. Virtual Servers by Name, IP and Port

To complete the example, we configure the server using all the options:

```
# Virtual Host Configuration File
ServerRoot "/etc/apache2"
DocumentRoot "/var/www"
PidFile /var/run/apache2/apache2.pid
User www-data
Group www-data
ErrorLog /var/log/apache2/error.log
TypesConfig /etc/mime.types

NameVirtualHost 192.168.2.166

<VirtualHost www.depinfp.iesjc>
      ServerName www.depinfo.iesjc
      DocumentRoot /var/www/depinfo
      ErrorLog /tmp/depinfo_ERROR.log
      TransferLog /tmp/depinfo_ACCESS.log
</VirtualHost>

<VirtualHost www2.depinfo.iesjc>
      ServerName www2.depinfo.iesjc
      DocumentRoot /var/www/depinfo2
      ErrorLog /tmp/depinfo2_ERROR.log
      TransferLog /tmp/depinfo2_ACCESS.log
</VirtualHost>

<VirtualHost 192.168.2.200:80>
      ServerName teachers.depinfo.iesjc
      DocumentRoot /var/www/teachers
      ErrorLog /tmp/teachers_ERROR.log
      TransferLog /tmp/teachers_ACCESS.log
</VirtualHost>

<VirtualHost 192.168.2.200:8080>
      ServerName students.depinfo.iesjc
      DocumentRoot /var/www/students
      ErrorLog /tmp/students_intranet_ERROR.log
      TransferLog /tmp/students_intranet_ACCESS.log
</VirtualHost>
```

4.6. Dynamic Virtual Servers

But what if we have to configure tens, hundreds or even thousands of websites as happens in an ISP? For that we have the option of Dynamic Virtual Hosts.

We need to load the module: `$ sudo a2enmod vhost_alias` Let's look at an example:

```
# Configuration File for Dynamic Virtual Hosts
ServerName "iesjc"
ServerRoot "/etc/apache2"
DocumentRoot "/var/www"
PidFile /var/run/apache2/apache2.pid
User www-data
Group www-data
ErrorLog /var/log/apache2/error.log
TypesConfig /etc/mime.types
ErrorLog /tmp/iesjc_ERROR.log
#  We instruct Apache to use the name of the request received as its name
(ServerName)
UseCanonicalName Off
# Setting the DocumentRoot for each website we serve.
VirtualDocumentRoot /var/www/%1.0
```

But, if we have many websites, it can also be useful for each of them to have its own configuration file (we will give permission to their administrators to manage it themselves). We can achieve this by uploading all the configuration files of the virtual sites from a directory.

We could do this by using the following line in the Apache configuration file:

```
Include /etc/apache2/sites-enabled/[^.#]*
```

This directive includes in the configuration file all the files that it finds in the indicated directory.

4.7. Policies for Configuring Virtual Servers

Let's look at the directives we've used:

- **<VirtualHost>**

  Indicates that the configuration block it spans (between <VirtualHost> and </VirtualHost>) applies to the listed website.

  Each virtual host can be identified by IP, IP:Port, or by name.

- **NameVirtualHost**

  Sets the IP address on which virtual hosts will be configured.

- **VirtualDocumentRoot**

  Dynamically sets the root of documents (DocumentRoot) for virtual hosts. To set the name of the directory use % which refers to the name we are to access our server and the number that goes with the % indicates that part of this name is used to identify your directory:

  - %0 The entire name is used, so if our website is identified by www.depinfo.iesjc, the root directory will be /var/www/www.depinfo.iesjc/
  - %1 takes the first part of the name, www and the root directory would be in /var/www/www %2 takes the second part.

The following list shows how it works as combinations can be made:

- %% insert a %
- %p inserts the port number of the virtual server
- %N.M where N indicates the part of the name and M indicates how many characters are taken from it. The value for N:
    - 0 Full name →%0
    - 1 the first part → %1
    - 2 the second part →%2
    - -1 The last part →%-1
    - -2 the penultimate part →%-2
    - 2+ the second and all subsequent parts → %2+
    - -2+ the penultimate and all the preceding parts → %-2+
    - 1+ and -1+ the same as zero 0

- **UseCanonicalName**

  Determines how Apache will know its own name (for redirects).

- **TransferLog**

  Sets the Apache access log file. This file will record accesses to the pages served by Apache.

5. Secure access with OpenSSL (HTTPS)

Secure connections are achieved using the https protocol. This protocol is similar to http, but uses SSL encryption algorithms (in our case those provided by the OpenSSL package).

To get Apache to send us encrypted pages, a simple configuration has to be made, which broadly consists of generating the certificate that will be presented to the client when it connects (on a "professional" server this certificate would be requested from a certifying agency) and creating a virtual server that listens on port 443 (by default of the https protocol) and that presents the certificate.

5.1. Certificate generation

First of all, you have to create the directory where the certificates will be saved. This directory will be created where the rest of the Apache configuration is, although it is also common for them to be stored in the directories that Linux distributions have (In Debian / Ubuntu they are in /etc/ssl/certs/ for certificates and /etc/ssl/private/ for keys):

```
$ sudo mkdir /etc/apache2/ssl
```

The certificate will be generated with the command:

```
$ sudo /usr/sbin/make-ssl-cert /usr/share/ssl-cert/ssleay.cnf
/etc/apache2/ssl/apache.pem
```

The certificate will be in the /etc/apache2/ssl/apache.pem file. This file contains the keys necessary for the exchange of encrypted information, so all that is missing is the configuration of the virtual server.

5.2. Secure Virtual Server

For the correct operation we only need to configure a virtual server that listens on port 443 and presents the certificate that we have generated for the encryption of the connection. To do this, you just have to follow the same steps as before to create the virtual server and add a few lines:

```
<VirtualHost *:443>
ServerAdmin webmaster@localhost
ServerName www.smx.org
SSLEngine On
SSLCertificateFile /etc/apache2/ssl/apache.pem
DocumentRoot /var/www
<Directory / >
      Options FollowSymLinks
      AllowOverride None
</Directory>
<Directory /var/www/>
      Options Indexes
      FollowSymLinks MultiViews
      AllowOverride None
</Directory>
ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
<Directory "/usr/lib/cgi-bin">
      AllowOverride None
      Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
</Directory>
ErrorLog /var/log/apache2/error.log
#Possible values: debug, info, notice, warn, error, alert, emerg.
LogLevel warn
CustomLog /var/log/apache2/access.log combined
Alias /doc/ "/usr/share/doc/"
(…)
</VirtualHost>
```

Basically, the only difference with the other virtual servers that has been created is the use of port 443 (first line), which activates the SSL engine and indicates where the file with the certificate is (second and third) and finally that the directory is asked to use SSL.

In the `/etc/apache2/ports.conf` file you must have the `Listen 443` directive for it to work correctly, but under normal circumstances this directive will already be in place from the Apache installation and activated at the time the SSL module is loaded.

5.3. Enabling Secure Access

When all the rest of the configuration is done, all that remains is to activate the SSL module, so the command will be executed:

```
$ sudo a2enmod ssl
```

And the Apache is restarted so that it loads all the new configuration:

```
$ sudo systemctl restart apache2
```

As we will see when loading our website with its certificate in the browser, it tells us that the website is not secure, that the associated certificate is not trusted. This is because the certificate has not been issued by a recognized Certificate Authority, the root certificate of the Certificate Authority is not installed in the browser and therefore should not be trusted. Let's see below another way to generate certificates from a Certificate Authority (CA), which we can install in the client browsers and we will verify that it now trusts the certificates signed with our CA.

Follow the instructions below:

Key for CA certificate with password:

```
$ sudo openSSL genrsa -out private_key_ca.pem -des3 2048
```

Genrsa operation --> Generates an RSA private key

-des3 2048 --> Triple-DES encryption with 2048 bits in length, we can use Blowfish encryption instead.

Create the certificate of the Certificate Authority:

```
$ openssl req -new -x509 -key private_key_ca.pem -out cacert.pem -days 
365
```

req operation → management for x509 certificate signing requests

$ man req: (which puts the in manual) -new --> Generates a new certificate request

-x509 --> x509 certificate management

-key --> -Key to use

-out --> Output file with certificate -days --> Validity period of the certificate, in this case 365 days.

Key for our server certificate:

```
$ sudo openssl genrsa -out server_private_key_cert.pem 2048
```

It creates the key, but since it doesn't have the encryption option, it doesn't ask for a password.

Request for an unsigned and passwordless server certificate so that the service does not stop when booting while waiting for the key.

```
$ sudo openssl req -new -nodes -key server_private_key_cert.pem -out 
serv_cert_request.pem -days 365
```

With -nodes → This option indicates that if a private key is created, it should not be encrypted.

Create this file: `config_cert.txt`

```
basicConstraints=critical,CA:FALSE
extendedKeyUsage=serverAuth,emailProtection
```

The first line indicates that you cannot sign other certificates with this one. The second that we'll use it as a server certificate.

Signing the Server Certificate with the CA Certificate:

```
$ sudo openssl x509 -CA cacert.pem -CAkey private_key_ca.pem -req -in 
serv_cert_request.pem -days 365 -sha512 -CAcreateserial -out 
signed_serv_cert.pem -extfile config_cert.txt
```

x509 Operation → x509 Certificate Data Management

$man x509: (what it says in the manual) -CA → Specifies the certificate of authority to be used to sign.

-CAkey → Specifies the private key used to sign the certificate

-req → Indicates that a certificate request is to be used instead of a certificate

-in → Indicates the input file with the certificate to be processed

-sha512 → A hash function used to create the certificate digest to verify its integrity.

-CAcreateserial → Creates a file with the serial number for CA, it is necessary if the -CA option is used.

-extfile → Specifies a file with extensions used in the certificate

Now the file with the CA certificate (cacert.pem) is copied to the client systems and in the browser in the configuration we will add it in the Certificates section in the "Issuing Entities" area in Chrome or "Authorities" in Firefox. If we now load our website, the connection will appear as secure since we have indicated that it trusts this CA.

But when performing this step in browsers such as Brave or Chrome, they show us a certificate error in which they say that CN is invalid. This is because they expect a certificate in which the server names appear in a list of alternative names and that although they are completely different they refer to the same domain, as a list of aliases. To achieve this we will use a certificate with SAN (Subject Alternative Name) support. The steps are basically the ones we have already seen, changing only the commands to generate the server's certificate request and the server's signature to obtain the certificate itself. Let's look at the steps.

First of all, we create a file with all the configuration and data of the certificate to be generated. Here we will put the information related to the server for which we create the certificate, the data that in the previous example we entered from the command line, we will also put the configuration of the certificate, that we will not be able to sign other certificates with it and that we will use it to authenticate servers and finally we will also add the alternative names that we can use in the URL of the browsers to access this server. Let's look at an example of this file, called `san.txt`:

```
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req prompt = no

[req_distinguished_name]
C = ES
ST = Castellón
L = Benicarlo
O = iesjc
OU = depinfo
CN = ubnt2024
[v3_req]
basicConstraints=critical,CA:FALSE
extendedKeyUsage = serverAuth
subjectAltName = @alt_names

[alt_names]
```

```
DNS.1 = ubnt2024
DNS.2 = ubnt2024.depinfo
DNS.3 = ubnt2024.depinfo.iesjc
```

Now the commands in which we use the previous file to generate the certificate request first and the signing of the request with the CA certificate to obtain the certificate below are the following, the previous orders to these are the same as we have seen above:

```
$ sudo openssl req -sha512 -new -out serv_cert_request.csr -key
server_private_key_cert.pem -config san.txt
$ sudo openssl x509 -days 365 -req -sha512 -in serv_cert_request.csr
CAcreateserial -CA cacert.pem -CAkey private_key_ca.pem -out
signed_serv_cert.crt -extensions v3_req -extfile san.txt
```

Once we have the `signed_serv_cert.crt` certificate and its key, we put them in the configuration of the virtual server as we have seen before

6. Authentication

On many occasions we will want to restrict access to some resources (pages). With Apache we can establish username and password mechanisms, to limit access. In addition, users can be included in groups and set permissions for the groups. BUT BEWARE! The transmission of information has very weak encryption. Do not mount any of the mechanisms that we will see below if you do not also use SSL. Any sniffer can easily steal your passwords.

6.1. How does it work?

Once we establish that a certain resource on our server requires authentication, when we try to access it, Apache returns a 401 (Authentication Required) message to the client. Normally, in current browsers, if there is a saved user/key pair for the resource, it will be sent to the server automatically. If not, they will ask the user. From the server side, there are two authentication mechanisms.

6.2. Basic Authentication

With this mechanism, the password is sent clearly over the network, or at most, with a base 64 encryption (easy, easy...).

We must enable the module: `a2enmod auth_basic` Let's see an example:

```
<VirtualHost teachers.depinfo.iesc>
      ServerName teachers.depinfo.iesjc
      DocumentRoot /var/www/teachers
      ErrorLog /tmp/teachers_ERROR.log
      TransferLog /tmp/teachers_ACCESS.log
      <Directory /var/www/teachers>
            # Setting the type of Access Control to be used
            # to access this directory
            AuthType Basic
            # Givinge a name to the private environment we protect
            AuthName "Teacher Information"
            # File with the information of authorized users and
            # their (encrypted) passwords.
            AuthUserFile /etc/apache2/passwd/passwd_teachers
            # File with information from groups and their member users.
            AuthGroupFile /etc/apache2/passwd/teachers_group
            #Setting the security level for this directory.
            require valid-user
            # Others possible values are:
            # require user1 user2 ... list with the Authorized Users
            # require group group1 grupo2 ... list with the Authorized 
            Groups
            # require user usu1 usu2 group gr1 gr2 ... a combination of 
            both
      </Directory>
</VirtualHost>
```

6.3. User and Group Files

The information with the usernames and passwords is stored in the file indicated by the AuthUserFile directive. This file will look something like: usu1:$apr1$Ym2fv...$kGAt 2g1y6dOSt404Xfka2. usu2:$apr 1$0D0nG/.. $cv 8eAB0Fykj/PnoIP/0x6. And, naturally, it should be where no web server user can get to read it.

Since few people are able to write the encrypted password directly, Apache includes an application to manage this file. This is htpasswd.

To create a file like the one in the example, we will have to execute the following commands:

```
$ sudo htpasswd -cm /etc/apache2/passwd/passwd_teachers usr1
$ sudo htpasswd -m /etc/apache2/passwd/passwd_teacher usr2
```

The first one creates the file and adds the user usu1. Option c indicates that a new file must be created and option m that the password will be encrypted with MD5.

The second command simply adds the user us2 to the already created file.

In both cases we will be asked for the password.

The group file contains the information of these, indicating which users belong to each group. It does not require any command, as it is a simple text file, with a structure like the following: teachers: usr1 usr2 admin: adm1 adm2

Note that the users in each group are not separated by commas (as is done in /etc/group), but by white spaces.

6.4. Digest Authentication (Encryption)

We can improve security somewhat by using Digest. Instead of sending the password in the clear (or base-64 encrypted) over the network, with Digest the passwords do not travel over the network. Instead, when you request a protected resource, the server sends the client a number (noonce). Using that "unique" number, the URI of the requested resource, and the password, the browser digests. That is, a calculation based on MD5 that gives you a very rare number and difficult to obtain without the password (very difficult). The server checks the digest using the stored information. Due to the mathematical peculiarities of this type of operation, the server does not need to save the password (not even an encrypted one). You just have to save another digest based on the same password to check it out.

Certainly, this mechanism reduces the risk of password capture. Let's look at an example:

```
# Configuration file with Digest Authentication
# Load the module we need with a2enmod auth_digest
<VirtualHost teachers.depinfo.iesjc>
     ServerName teachers.depinfo.iesjc
     ServerAdmin administrator@depinfo.iesjc
     DocumentRoot /var/www/teachers
     ErrorLog /tmp/teachers_ERROR.log
     TransferLog /tmp/teachers_ACCESS.log
     <Directory /var/www/teachers>
          AuthType Digest AuthName teachers
          # Indicates the person in charge of supplying the Digest
          # by default file. Or, using the modules
          # appropriate, use databases where they are stored
```

```
                # Usernames and passwords
                AuthDigestProvider file # File with authorized users' 
                Digests
                AuthUserFile /etc/apache2/passwd/digest_teachers require 
                valid-user
        </Directory>
</VirtualHost>
```

Digest file

The management of the digest file on the server is carried out with the htdigest application. To add the same users we had, we must place the following orders:

```
$ sudo htdigest -c /etc/apache2/passwd/digest_teachers teachers 
usr1
$ sudo htdigest /etc/apache2/passwd/digest_teachers teachers usr2
```

Again, the c option in the first command will create the file.

Note that it is necessary to pass the name of the protected environment (teachers) to the command, this field is called realm and its value corresponds to the one we have used in AuthName.

6.5. Authentication by Networks, Domains and Hosts

But sometimes we need to set permissions not only by user, but also by source IP address, subnet, etc. For this we have the Require ip or Require host directives provided by the mod_authz_host module that is usually enabled by default.

Let's look at an example:

```
# Configuration file with Basic Authentication and Allow
ServerName "iesjc"
ServerRoot "/etc/apache2"
DocumentRoot "/var/www"
PidFile /var/run/apache2/apache2.pid
User www-data
Group www-data
ErrorLog /var/log/apache2/error.log
TypesConfig /etc/mime.types
ErrorLog /tmp/iesjc_ERROR.log
NameVirtualHost 192.168.0.51
<VirtualHost teachers.depinfo.iesjc>
        ServerName teachers.depinfo.iesjc
        DocumentRoot "/var/www/teachers"
        ErrorLog /tmp/teachers_ERROR.log
        TransferLog /tmp/teachers_ACCESS.log
        <Directory /var/www/teachers>
            AuthType Basic
            AuthName "Teacher Information"
            AuthUserFile/etc/apache2/passwd/passwd_teachers
            AuthGroupFile /etc/apache2/passwd/grupo_ teachers
            require valid-user
        </Directory>
        <Directory /var/www/stu_wad> # Only allows access from this 
        subnet
            Require ip 192.168.18
        </Directory>
    </VirtualHost>
```

The operation of Require can be established through IP's or through subdomains and hostnames, for example:

`Require host iesjc.org aula18.build_a.net teacherserver` Allows access to systems belonging to these domains and the teacherserver system.

Requires ip 10.2 192.168.100.100 Allows access to systems that are in this subnet and to the one that has that IP.

It can also be used with negation by:

`Require not host iesjc.org aula18.build_a.net teacherserver`

Require not ip 10.2 192.168.100.100

which will have the opposite effect, however, the Rerequire statements, if there are several or there is a denial, are included in directives such as: <RequireAll>... </RequireAll>, <RequireAny> ... </RequireAny> or <RequireNone> ... </RequireNone>. So that the first one must meet all the conditions to grant access, in the second that some of the conditions are met and in the last that none of the directives contained in it are met.

6.6. Policies for Authentication

Let's look at the directives we've used:

- **AuthType** Sets the type of user authentication to use. The possible values are Basic and Digest.
- **AuthName** Sets the name for the sandbox.
- **AuthUserFile** Sets the server file that stores users' information and their encrypted passwords (for Basic authentication).
- **AuthDigestFile** Sets the server file that stores user information and their digests (for Digest authentication).
- **AuthGroupFile** Sets the server file that stores user group information (for Basic authentication).
- **Require** Determines which users will be able to access the protected resource. The possible values are: user, group, valid-user, or any combination of them. When using user and/or group, a space-separated list with the names of authorized users and/or groups must be added.
- **Require [not] ip** Sets the system that can access the resource using a network address or subnet.
- **Require [not] host** Sets the name of a host or subdomain to the systems that will be able to access the resource.
- **AccessFileName** Sets the name for distributed configuration files.
- **AllowOverride** Sets which directives are allowed in distributed configuration files. Possible values are: AuthConfig, FileInfo, Indexes, Limit, and Options.
- **AuthDigestProvider** Indicates the person in charge of supplying the default Digest file. Or, using the appropriate modules, use databases where usernames and passwords are stored.
- **RequireAll** includes a set of authorization policies of which none must fail, and at least one must be met in order for it to grant access to the resource.
- **RequireAny** includes a group of authorization policies of which at least one must be met in order to grant access to the resource.

- **RequireNone** includes a group of authorization policies that must not be met in order for you to grant access to the resource.

7. .htaccess Distributed Configuration

With the .htacces files you can modify the access policies to the directory you are in, it would have the same effect as if these policies are set in the virtual server settings within the <Directory> policy. Apache advises against using these types of configurations if we can edit the configuration file where the directory is used. .htaccess files slow down the server's response.

Policies included in a .htaccess files are applied in the directory where you are located and in its subdirectories, note that several of these files can be cascaded by rewriting policies established in previous configurations.

In order to use distributed configuration files, the AllowOverride directive must be set in the Directory definition in the virtual server configuration file. AllowOverride indicates what will take effect on the directory of those included in the .htaccess. For example, if you want to modify the default character set in the contents of a directory using .htaccess, we will put the AddDefaultCharset utf-8 directive in it, but it will only take effect if the directive appears in the Directory definition of this virtual server: AllowOverride FileInfo, otherwise, it will not allow you to override this feature and change the character set to the one we have indicated in .htaccess. To allow all kinds of changes you can set AllowOverride All. Another typical use of .htaccess is to establish authentication of users with access to the contents of the directory, for this AllowOverride AuthConfig must be set and in the .htaccess we will put for example:

```
AuthType Digest
AuthName teachers
AuthDigestProvider file
AuthUserFile /etc/apache2/passwd/digest_teachers
require valid-user
```

8. Indexes

Sometimes it can be impossible to keep a directory's index page up to date. For example, if it is a download directory in which several users can upload information for others to access. Also, why are we going to work making indexes in HTML if Apache can do them for us.

The normal operation of Apache2 is that if when it receives a request from a client in the access directory it does not find any file of the type DirectoryIndex then it will show the list of contents of the directory, this default configuration can be changed if we do not want the contents of the directory to be displayed by removing Indexes in the Options directive or by putting:

```
Options -Indexes
```

Below, we'll look at how we can handle the way this content is displayed. Apache has directives not only to make very powerful indexes, but also to customize them by adding or removing options. Let's see how to set them up:

```
<VirtualHost www.depinfo.iesjc>
      ServerName "www.depinfo.iesjc"
      DocumentRoot "/var/www/depinfo"
      <Directory /var/www/curso/depinfo/forms>
            # Setting nice indexes for this directory
            IndexOptions FancyIndexing
            #Adding a description for some resources
            AddDescription "Contact Form" contact.html
            # Doing some items not to appear in the index
            IndexIgnore *.jpg
            IndexIgnore..
      </Directory>
</VirtualHost>
```

There are many policies that will allow us to configure the indexes, you can consult them in https://httpd.apache.org/docs/2.4/mod/mod_autoindex.html. Here are a few:

- **IndexOptions** Sets the options for the indexes. They can be, among others: DescriptionWidth, FancyIndexing, FoldersFirst, IconsAreLinks, IconHeight, IconWidth, IgnoreCase, ScanHTMLTitles ...
- **IndexIgnore** Determines which files will not appear in the index.
- **AddDescription** Adds a description for one or more files in the index.

- **AddIcon** Associates an icon with a file or file type.
- **IndexOptions FancyIndexing** Allows you to sort the list by clicking on the column headers.
- **SuppressColumnSorting** Removes links from headers that allow content sorting

9. Redirects

Sometimes it is necessary to move things around. You have to move directories from one place to another, move files... and we want our customers to continue to access as they did before. The mod_rewrite module is used to rewrite the URLs of requests received by our server so that it can serve the pages even if the requested content has been moved, either within the current server or to a different server.

To achieve this, we have three mechanisms:

9.1. Redirects with Alias

Alias allows us to give access to resources that are not at the address indicated by the DocumentRoot directive.

For example

```
<VirtualHost www.depinfo.iesjc>
      ServerName www.depinfo.iesjc
      DocumentRoot /var/www/depinfo
      # Giving access to a directory that is NOT in DocumentRoot
      Alias /exams_scheduling /var/www/calendar
      # Also we can create DocumentRoot directory accesses
      Alias /exam1 /var/www/depinfo/development/exam1/test
</VirtualHost>
```

9.2. Redirects with Redirect

Redirect associates an old URL with a new one. The new URL is notified to the client to make the connection to it.

For example:

```
      <VirtualHost www.depinfo.iesjc>
            ServerName www.depinfo.iesjc
```

```
        DocumentRoot "/var/www/depinfo"
        # Redirect customers to the new website
        Redirect /info http://www.iesjoancoromines.org
</VirtualHost>
```

9.3. Rewrite

All of the above, and much more, can be done with Rewrite. Rewrite is a very extensive and comprehensive module that basically applies a pattern to a URL and makes substitutions to it.

For example:

We load the module we need: `$ sudo a2enmod rewrite`

```
NameVirtualHost 192.168.0.51
<VirtualHost www.depinfo.iesjc>
        ServerName www.depinfo.iesjc
        DocumentRoot "/var/www/depinfo"
        # Turning on the rewrite engine
        RewriteEngine on
        # Setting the log file
        RewriteLog /tmp/log_rewrite
        # Setting the log level
        RewriteLogLevel 9
        # Redirect www.depinfo.iesjc/info/ to iesjoancoromies.org
        RewriteRule ^/info/$ http://www.iesjoancoromines.org
</VirtualHost>
```

But there are much more complicated and useful rewrite rules. For example, let's imagine that we have a web page whose content depends on a parameter (typical of PHP, Perl, etc. pages). The problem is that neither humans nor search engines like to remember a URL of the http://www.depinfo.iesjc/prog.php?opcion=1 type. We prefer direct URLs.
What can we do?

We can rewrite the requests that come to us to an "easy" address. For example, if option 1 gives us information about the school and option 2 gives us information about the school calendar, we can create two "fictitious" addresses that are http://www.depinfo.iesjc/centro/ and http://www.depinfo.iesjc/calendario/ and put the following directives in the configuration file:

```
RewriteRule ^/center/$ prog.php?option=1 [L]
RewriteRule ^/calendar/$ prog.php?option=2 [L]
```

Look at option [L] at the end. It signals to the process that it is a final rule. If it is applied, no further rules will apply.

Rewrite rules run in cascade. The inbound URL for the second rule is the outbound URL for the first. This allows us to use chained rules.

The options are almost limitless: we can pass parts of the URL as parameters, we can set conditions, chains of rules... Any redirection problem, no matter how complicated, can be solved with this module.

9.4. Policies for Redirect Configuration

We have used the following directives:

- **Alias** Associates URLs with system files.
- **Redirect** Sends an external redirect instructing the client to load another URL.
- **RewriteEngine** Enables/Disables the rewrite engine.
- **RewriteLog** Sets the log file for the rewrite engine.
- **RewriteLogLevel** Sets the debug level for the rewrite engine.
- **RewriteRule** Sets a rule for the rewrite engine.

10. Logging - Activity Log

Log files are a critical tool for the security of our server. In this chapter we will see how to configure them so that Apache saves the information we need, both to improve its performance and to solve problems for users, know which sections they access the most, unauthorized access attempts, etc.

But it is not enough to correctly configure the logs so that they store the information we want. We also need to consult it regularly. Otherwise, any information we might record will be useless.

There are many programs that can help you inspect and manage Apache logs.

You can find them in the apache2-utils, scanerrlog, visitors, vlogger, webdruid...

10.1. Log Files

Although we have already used them, we are going to mention them explicitly.

Apache uses two log files:

- **ErrorLog** – To log errors that occur when accessing Apache resources.
- **TransferLog** – To record accesses that do not fail.
- **CustomLog** Sets the file and format for a custom log.

These policies can be set in the Apache2 configuration file, but most commonly set them on each virtual server to record their activity.

10.2. ErrorLog

The contents of ErrorLog look something like this:

```
[Tue, Nov 21, 11:20:14, 2006] [notice] caught SIGTERM, shutting down
[Tue Nov 21 11:20:15 2006] [notice] Apache/2.0.55 (Debian) configured
-- resuming normal operations
[Tue Nov 21 11:20:18 2006] [error] [client 127.0.0.1] unknown directive
"fsze="ssi.shtml"" in parsed doc /var/www/ssi.shtml
[Tue Nov 21 12:12:36 2006] [error] [client 192.168.0.51] File does not
exist: var/www/course/cityhall/not_exist
```

In it we can see the messages of Apache closure, restart, SSI policy errors, pages not found, etc.

10.3. TransferLog

The contents of TransferLog will look something like this:

```
192.168.0.51 - - [21/Nov/2006:10:22:32 +0100] "GET /cgi/aym.cgi
HTTP/1.1" 200 51
127.0.0.1 - - [21/Nov/2006:11:07:30 +0100] "GET /ssi.shtml HTTP/1.1"
200 130
192.168.0.51 - - [21/Nov/2006:11:43:20 +0100] "GET /index.shtml
HTTP/1.1" 200 348
192.168.0.51 - - [21/Nov/2006:12:12:36 +0100] "GET /not_exist
HTTP/1.1" 404 207
```

In it we can see from which address our server has been accessed, at what time, what request was received, result, etc.

10.4. Policies for Log Configuration

We can use the following directives for the configuration of log files:

- **ErrorLog** Sets the file that stops the error log. Optionally, this can be a program that, for example, logs errors in a database.
- **LogLevel** Sets the level of detail for the error log. The possible values are: emerg, alert, crit, error, warn, notice, info, and debug.
- **TransferLog** Sets the file that stops the access log. Optionally, it can be a program that, for example, records accesses in a database.
- **LogFormat** Defines a log format for the access log or for a custom log.
- **CustomLog** Sets the file and format for a custom log.

Web Links

In this section, you will find the relevant links of interest necessary to expand and explore the contents of the unit.

- Apache HTTP Server Version 2.4 Documentation - Apache HTTP Server Version 2.4