Lab Assignment – Introduction to Apache2 web server configuration

Course outline:

1.Introduction

Web servers are used to serve solicited resources requested by web clients. These clients, named *browsers*, are typically graphical applications with hypertext capabilities, able to make HTTP request and to visualize the web contents of the HTTP responses. Some examples of browsers are Google Chrome, Firefox and Microsoft Internet Explorer.

Users make an HTTP request through the web browser by introducing an URL (Uniform Resource Locator), which identifies the web server's FQDN (Fully Qualified Domain Name) and a path to the resource requested. For example, to access the UPM main web page, user should introduce http://www.upm.es. However, for information about "Estudiantes", user should access the corresponding web resource, through the link http://www.upm.es/institucional/Estudiantes. In this case the directory "institutional" has an HTML page named "Estudiantes".

Apache is a software foundation with a wide variety of open source software for the web, which includes Apache2 web server. It works under a GNU Public License and is free of charge, being available for various operating systems, namely Linux, Windows, MacOS and *nix platforms.

Apache2 web server, currently in the stable version 2.4.12, has a set of interesting features that makes it the most used web server worldwide, as depicted in Figure 1.

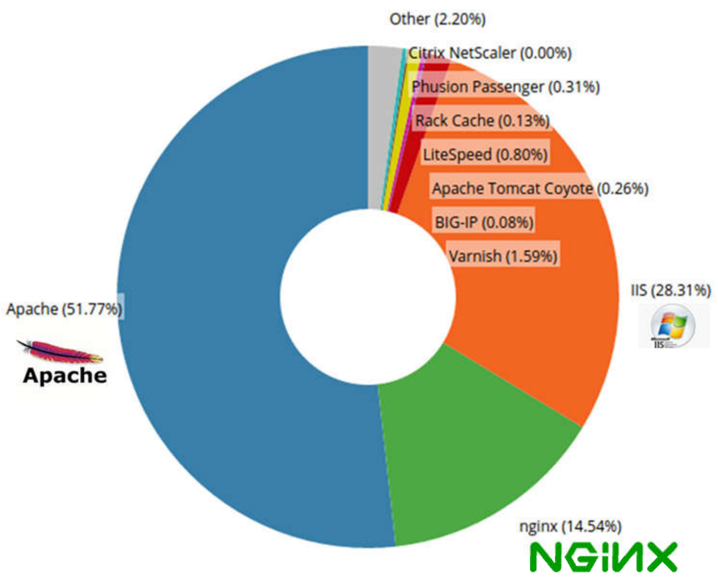

Figure 1- Market share of web servers [`http://trends.builtwith.com/`].

At a first glance we may observe that Apache2, NGIX and Microsoft IIS altogether represent about 94.6% of the market share in web servers, according to the "Built With Trends" website[1] in April 2015. The most commonly used web transfer protocol is Hyper Text Transfer Protocol (HTTP). Apache2 also support HTTP connections under an SSL/TLS tunnel (HTTPS) and File Transfer Protocol (FTP).

Apache2 web servers running in a Linux system are commonly used with a database engine (e.g. MySQL) and scripting languages for SysAdmin operations. This framework that ally **L**inux, **A**pache, **M**ySQL and a scripting language like **P**erl, **P**ython and/or **P**HP, are commonly named LAMP. Other different flavors of *AMP architecture are available for other operating systems rather than Linux. Some examples are XAMP for cross-platforms operating systems, WAMP for Windows and MAMP for MAC OS.

2. Lab setup

This class was prepared in a lab environment in which the PCs have a specification similar to the one illustrated on Figure 2.

The host system can be any operating system (e.g. Linux or Windows). We only need a browser and a virtual machine hypervisor and player, like `vmware` or `VirtualBox`. The virtual machine is an Ubuntu Server 14.04 LTS for 64 bits, configured with 512 Mbytes of RAM and 20 Gbytes of disk. It is also a good practice to have two different Network Interface Cards (NIC) configured. One should be configured as "Host Only" to connect directly with the host, being the other NIC configured as NAT to access to the Internet.

[1] http://trends.builtwith.com/web-server

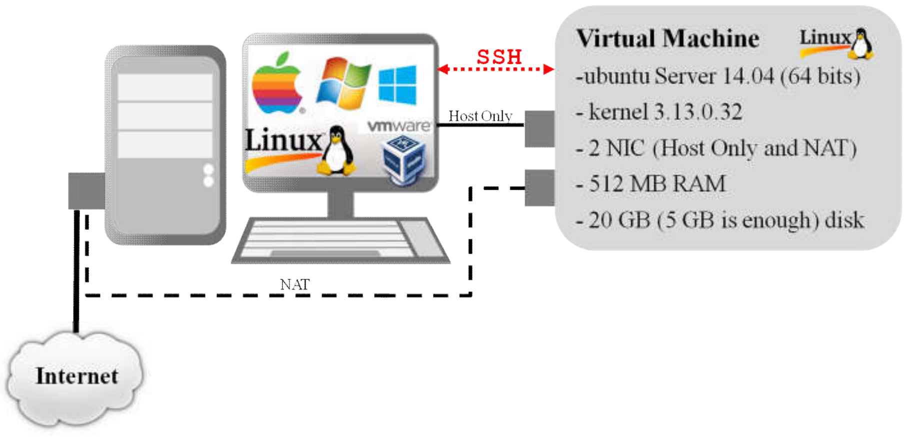
Virtual Machine
Linux
-ubuntu Server 14.04 (64 bits)
- kernel 3.13.0.32
- 2 NIC (Host Only and NAT)
- 512 MB RAM
- 20 GB (5 GB is enough) disk
SSH
Host Only
NAT
Linux
vmware
Internet

Figure 2- Lab setup specification

The NIC configured as “Host Only” can be used to remotely administrate the virtual machine from the host system through a secure SSH connection, without using the internet access for administration tasks. To have SSH running we need to install `openssh-server` package and initiate the service with the default parameters, by executing as `root` the following command: `apt-get install openssh-server`. You may also intend to change any parameters on the `ssh` configuration file `/etc/ssh/sshd_config`. Any changes to the `ssh` configuration implies to restart the service, by issuing the command `service ssh start`.

After installing the virtual machine with Ubuntu 14.04 operating system, we have to download and install all the patches and security updates available on the official repositories, by executing as `root` the following commands:

- `apt-get install update`
- `apt-get install upgrade`

It is recommended that you reboot the Linux server after the upgrade process, by executing one of the command following commands:

- `shutdown -r 0`
- `reboot`

3.Apache2 installation and startup procedures

Apache2 web server is available under the Ubuntu repositories. To install the Apache2 web server, its documentation and some useful tools, please use the following command line: `sudo apt-get install apache2 apache2-doc apache2-utils`.

After the installation process is finished, the Apache2 daemon should be running. You may check if the `apache2` service/daemon is up and running, by executing as `root` one of the following commands:

- `service apache2 {start|status|restart|stop|...}`
- `/etc/init.d/apache2 {start|status|restart|stop|...}`
- `apache2ctl {start|status|restart|stop|...}`

There are other ways to identify if the web server is running and to analyze any error or warning that may have occurred during the startup. A list of verification and validation commands can be found below:

- `ps -ef | grep apache2`
- `ps -eLf | grep apache2`
- `tail /var/log/syslog`
- `tail /var/log/apache2/error.log`
- `netstat -ant`
- `telnet <ip_address> 80` or `telnet localhost 80`

A tools set of scripts and commands to administrate the Apache2 web server is also installed, namely `a2query`, `a2ensite`, `a2dissite`, `a2enmod`, `a2dismod` and `htpasswd`. These scripts are described in Section 4 and some of them will be used during this course.

4. Directories and files

The Apache2 web server configuration is made through the edition of text configuration files located by default in the directory `/etc/apache2/`.

`apache2.conf`	`Main configuration file that handle global definitions.`
`envvars`	File with environment variables used in the Apache2 configuration files.
`mods-available`	This directory holds the configuration of all the available Apache2 modules that can be activated in Apache2.
`mods-enabled`	After enabling a module, this directory holds a link to the module configuration file located in directory `mods-available`. Modules are enabled by executing the command `a2enmod` and further reloading (or restarting) Apache2 web server. (`service apache2 reload|restart`).
`ports.conf`	File with the directives related with TCP ports to be used by Apache2 for HTTP and HTTPS connections.

`sites-available`	This directory contains virtual hosts configuration files. Virtual hosts allow Apache2 to have a set of different web sites hosted in the same web server, each one with distinct configurations.
`sites-enabled`	After enabling a site (virtual host), this directory holds a link to the site configuration file located in directory `sites-available`. Sites are enabled by executing the command `a2ensite` and further reloading (or restarting) Apache2 web server. (`service apache2 reload|restart`).

There are also several scripts used to help on managing the Apache2 web server configuration. Some of them are described below:

`a2query`	A script used to query the Apache2 web server about the actual configuration. For example it is possible to know if a specific module or site is enabled.
`a2enmod`	A script used to activate a specific module, by creating a symbolic link between the module in the directory `/etc/apache2/mods-enabled` to the directory `/etc/apache2/mods-available`.
`a2dismod`	A script to deactivate a specific module, which consists on removing the symbolic link created in `/etc/apache2/mods-enabled`.
`a2ensite`	A script used to activate a specific site, by creating a symbolic link between the module in the directory `/etc/apache2/sites-enabled` to the directory `/etc/apache2/sites-available`.
`a2dissite`	A script to deactivate a specific site, which consists on removing the symbolic link created in `/etc/apache2/sites-enabled`.
`apache2ctl`	A set of management tasks for Apache2 web server, namely start, stop and look at the status, among others.
`htpasswd`	A script used to manage authentication in Apache2, by maintaining a file with the users' credentials valid to authenticate on Apache2 web server.

More information about the use of these commands can be found in the corresponding Linux manual page. For example, `man apache2ctl` or `man htpasswd`.

5. General configuration

Apache2 web server configuration is made through a set of configuration directives, spread by the files described previously on Section 4. These directives can have a global meaning or be included in a specific configuration context. Contexts are delimitated by specific tags with the following format:

```
<Context options>
    Directive_1 value_1
    Directive_2 value_2
    Directive_n value_n
</Context>
```

An example of a context is `<Directory /dir/test> … </Directory>`.

The full list of directives and corresponding explanation can be found in Apache2 documentation web page in `http://httpd.apache.org/docs/`. Examples of some most commonly used directives and the corresponding syntax can be found below.

`ServerName [scheme://]fully-qualified-domain-name[:port]`
Name by which the (virtual) server responds to HTTP requests. For example `www.upm.es`.

`ServerAdmin email-address|URL`
SysAdmin's e-mail. For example `webmaster@upm.es`.

`Listen [IP-address:]portnumber [protocol]`
TCP ports and optionaly IP addresses in which the server will listen for HTTP connections. By default, HTTP port is 80.

`ServerRoot directory-path`
Directory where the Apache2 configuration files and subdirectories are located.

`DocumentRoot directory-path`
Directory containing the contents that will be made available on the web.

`UserDir directory-filename [directory-filename]`
Directory name where the user files to be made available on the Internet are located. By default this variable is "`public_html`".

`AccessFileName filename [filename]`
File name that will allow to change the remote access perissions to a directory. By default, the file is named `.htaccess`.

`Redirect [status] URL-path URL`
It allows the redirection of HTTP requests to an external URL or to a different directory.

`ErrorDocument error-code document`
For a specific error type it is possible to have a customized HTML file.

`MaxClients number`
Maximum number of processes that can be created to process client requests.

`MaxRequestsPerChild number`
Maximum number of requests an Apache2 server can handle. After reaching this number the process dies, reducing memory leaks and the number of running processes.

`ListenBacklog backlog`
Maximum number of requests that can be in waiting queue.

`StartServers numbers`
Maximum number of Apache2 servers that are created on the startup phase.

`TimeOut seconds`
Maximum amount of seconds that Apache2 server has to wait to receive a valid GET request between a POST or PUT and between each ACK transmission.

`KeepAlive On|Off`
Enable or disable support for persistent HTTP/1.1 connections.

`MaxKeepAliveRequests number`
Maximum requests that can use the same persistent TCP connection.

`KeepAliveTimeout seconds`
Seconds that Apache2 has to wait before closing a persistent connection.

`ErrorLog file-path|syslog[:facility]`
File to where Apache2 will write errors, like a page not found event.

These configuration directives are grouped in contexts, depending on their purpose. The general configuration directives are in the file `/etc/apache2/apache2.conf`. Edit the file and make sure that the following main directives are present.

```
#Server's directory root
ServerRoot "/etc/apache2"
#PID File
PidFile ${APACHE_PID_FILE} #variable in envvar file
#Allow persistent connectiobns?
KeepAlive on
#How many persistent connections?
MaxKeepAliveRequests 100
#Time out for the next connection from the same client.
MaxKeepAliveTimeout 5
#Error log file
ErrorLog ${APACHE_LOG_DIR}/error.log
```

```
#Grant/Deny access to a content directory
<Directory /var/www/>
      Options Indexes FollowSymLinks
      AllowOverride None
      Require all granted
</Directory>
#File with additional configurations for directories
AccessFileName .htaccess
#Include files to be loaded at Apache2 server startup
IncludeOptional mods-enabled/*.load
IncludeOptional mods-enabled/*.conf
Include ports.conf
IncludeOptional conf-enabled/*.conf
IncludeOptional sites-enabled/*.conf
```

There is a slight difference between directives `Include` and `IncludeOptional`. The former is mandatory and if the file to be included does not exist then this event is reported in the logging file. The later means that if the file does not exist, this directive will be silently ignored instead of causing an error. In the default configuration file "`ports.conf`" is mandatory for the Apache2 server to know in which port it should listen to requests, while the virtual hosts configuration files are not mandatory, since there is also a default virtual host enabled.

6. Multi-Processing Module (MPM)

There are several modules available to be configured on Apache2. Their configuration files are available at the directory `/etc/apache2/mods-available/`. One of these modules is `MPM`, which is responsible by the customization of multi-processing environment in which Apache2 will run. This module is enabled by default, but we may want to adjust some parameters for better performance of the server. Edit the file `/etc/apache2/mods-enabled/mpm_event.conf` and identify the following parameters and their corresponding values. We may want to test different values to tune the web server performance.

```
<IfModule mpm_event_module>
      StartServers            2
      MinSpareThreads         25
      MaxSpareThreads         75
      ThreadLimit             64
      ThreadsPerChild         25
      MaxRequestWorkers       150
```

```
        MaxConnectionsPerChild   0
</IfModule>
```

7. Virtual Hosts

The way to have multiple web sites hosted in the same Apache2 web server is by configuring virtual hosts. To do so we have to follow some configuration tasks, described below.

1) Let's assume we intend to configure the site "`www.spain_portugal.com`". Let's also presume that the following operations are executed as `root` or similar privileges (e.g. `sudo`).
2) Copy the default virtual host template file, named `/etc/apache2/sites-available/000-default.conf`, to a new configuration file. To better understand the configuration files meaning it is a good idea to name it with the site name and the extension "`.conf`". That is:

   ```
   cd /etc/apache2/sites-available/
   cp 000-default.conf spain_portugal.conf
   ```
3) Create a new directory to host the contents of the new site. Assuming that `DocumentRoot` directive will point to `/var/www/`, then this new site may be hosted on `/var/www/spain_portugal`. So:

   ```
   mkdir –p /var/www/spain_portugal
   chown -R www-data.www-data /var/www/spain_portugal
   mkdir –p /var/log/apache2/spain_portugal
   ```
4) Edit the file `/etc/apache2/sites-available/spain_portugal.conf` and change the directives that will take effect on the way the website will process the requests. Below you can find a minimal proposal for the configuration of some common directives in a virtual host:

   ```
   <VirtualHost *:80>
         ServerName www.spain_portugal.com
         ErrorLog ${APACHE_LOG_DIR}/spain_portugal/error.log
         DocumentRoot /var/www/spain_portugal
         DirectoryIndex Index.html
         <Directory /var/www/spain_portugal>
               Options Indexes FollowSymLinks
               Order allow,deny
               Allow from all
         </Directory>
    <VirtualHost>
   ```

Variable `{APACHE_LOG_DIR}` is defined in file `/etc/apache2/envvar`.

5) In `/var/www/spain_portugal` we have to put the web site documents, like HTML pages and other resources. For testing purposes we might create a file `Index.html` with the following content:

```
<html>
  <head>
    <title>www.spain_portugal.com</title>
  </head>
  <body>
    <h1>web site www.spain_portugal.com successfully configured! </h1>
  </body>
</html>
```

6) Activate the site in Apache2 web server and force the Apache2 daemon to reread the configuration file, by restarting it. To do that, please execute the following procedure:

```
a2ensite spain_portugal

apache2ctl configtest

service apache2 restart
```

7) The new web site is now available through the previously created virtual host. To test the configuration, we have to access to the site by its corresponding URL, that is: `www.spain_portugal.com`.

   It is important to note that DNS should be able to resolve the URL, otherwise it'll not be possible to contact the Apache2 web server and the corresponding virtual host. A simple way to test this exercise in the lab setup is to change the "hosts" file of the system from where you intend to access. Mind that "hosts" file is a text file that is located in different places depending on the operating system. In Linux is located on `/etc/hosts` and in Windows is usually on `C:\Windows\System32\drivers\etc\hosts`. Below you may find an example of the line you should add to the "hosts" file, in order to have access to the web site created previously:

```
x.y.z.w  www.spain_portugal.com
```

   in which `x.y.z.w` corresponds to the IP address of the interface card accepting HTTP requests. The DNS resolution is always made firstly by accessing to the "hosts" file and then by trying to resolve through the DNS server.

   Another important issue is the access through IP address instead of by the fully qualified domain name. To do that, we have to configure the virtual host by IP address, replacing the virtual host definition with the following line: `<VirtualHost x.y.z.w:80>`. Again, in this case we have also to have DNS resolution implemented correctly.

8) To create another virtual host in the Apache2 web server, for example `www.portugal_spain.com`, we have to follow the steps from 2 to 7 previously described.

8. Access to user's home directory

One useful feature of web servers is to provide access to users' homepage, by using the following URL type: `http://www.spain_portugal.com/~user1`, in which `~user1` points to the home directory of the username `user1`. To do that Apache2 web server take advantage of the module `UserDir` that is configured as follow:

1) Edit the file `/etc/apache2/mods-available/userdir.conf`, in which we have to include the directives related with the `UserDir` module.
2) In this module we may restrict the access to some users and specify which directory contains each user's web contents. It is also a good practice to restrict the access to `root`'s home directory through a web browser.
3) An example of this configuration can be found below:

```
<IfModule mod_userdir.c>
    UserDir disabled root
    UserDir enabled user1 user2
    UserDir "public_html" "public"
</IfModule>
```

   In this case the the users `user1` and `user2` are enabled to have remote access to their directories and user's home directory may be named as "`public`" or "`public_html`". That is, the users' web page is located in subdirectories "`public`" or "`public_html`" under their home directories.
4) We have then to enable module `userdir`, by executing the command `a2enmod userdir`. Next we have to reload the Apache2 web server, since we have made changes to its configuration files.
5) It is now possible to type `http://www.spain_portugal.com/~user1` in the web browser to access `Index.html` file located in `~/user1/public_html`. To test the access, you have to create the file `~/user1/public_html`.
6) The remote access to home directories is disabled for user `root`, which is a good practice.

9. Remote Authentication

Remote authentication is usually applied to a directory in which only authorized users are able to access. We may apply remote authentication under the `<Directory>` context, which can be defined itself in a virtual host configuration. The remote authentication configuration has the following main tasks:

1) To set up a file with the credentials:

```
htpasswd -c /etc/apache2/.htpasswd user01
```

In this example the credentials file is `/etc/apache2/.htpasswd` and is created with the option "`-c`". In this execution, the user `user01` is also added to the file with the encrypted password asked in the command line.

2) To add other users' credentials you may execute the command `htpasswd` for each new user. For example:

   ```
   htpasswd /etc/apache2/.htpasswd user02
   htpasswd /etc/apache2/.htpasswd user03
   ```

   It is important to note that these users don't have to exist in the Linux system. These credentials are only used for authentication on the web server. Even for existing Linux users the credentials may be different. These credentials may also be centralized for example in an LDAP server. To do so, we have to configure and to enable Apache2 `ldap` module, available at `/etc/apache2/mods-available/ldap.conf`.

3) The example below enables Apache2 remote authentication in a specific directory: `/var/www/spain_portugal/private`. It uses the file `.htpasswd` that stores the authentication credentials and a basic (plain text) authentication method. This configuration should be included in virtual host configuration file, under the `<VirtualHost>` context.

   ```
   <Directory /var/www/spain_portugal/private>
         AuthUserFile /etc/apache2/.htpasswd
         AuthName "Restricted Area"
         AuthType Basic
         Require valid-user
   </Directory>
   ```

The authentication may also be encrypted, by changing the `AuthType` to `Digest`. In this case, the password has an associated "realm" and gives an extra protection on the authentication process. To setup and manage the digest authentication file we must use the command `htdigest`, according to the following example:

```
htdigest -c /etc/apache2/.htpasswd_digest 'Private' 'user01'
```

In this case we create the file `/etc/apache2/.htpasswd_digest` with the realm string "`Private`" and in the same command line we generate the password for user `user01`. The configuration of the directory to be authenticated by the method "digest" looks like the following:

```
  <Directory /var/www/spain_portugal/private>
        AuthUserFile /etc/apache2/.htpasswd_digest
        AuthName "Private"
        AuthType Digest
        Require valid-user
</Directory>
```

After changing the Apache2 configuration file we have to enable the module auth_digest and restart the Apache2 web server, by executing the following commands:

```
a2enmod auth_digest
service apache2 restart
```

It is important to note that the `AuthName` directive must have the same name as the realm string used to generate the passwords. In this case both are named "`Private`". It is also worth mention that this authentication method doesn't prevent password from being captured in the network. The best way to protect the authentication to a specific directory is by configuring an HTTP connection in an SSL/TLS tunnel. This method crypt the HTTP connection and the data are not able to be analyzed.

10. Remote Access

In Apache2 it is possible to define different methods to handle with remote access to directories and files. In directories we need to create the file in which we define the access configurations. This file is a hidden special file to the Apache2 web server and will not be present in the directory listing, if available. By default this file is named `.htaccess` and can be changed by the directive `AccessFileName` in Apache2 main configuration file. Some security issues related with remote access to files through the web browser are described below:

1) In the `<Directory>` context we may want to restrict the access to contents through symbolic links, by not using the directive `Options FollowSymLinks`.
2) We may also want to avoid the access to the directory listing, by not using the directive `Options Indexes`.
3) We may want to allow the access to a particular directory only for a specific range of computers in a particular IP range. This configuration is achieved by using the directives `Allow` and `Deny`.
4) The same configurations can be applied to specific files under the context `<File>`. For example we may want to have specific restrictions on accessing files with the extension "`.GIF`". Since the `<Directory>` and `<File>` may exist in a virtual host, we are able to have different access combinations for the set of virtual hosts configured.
5) An example of the remote access configuration for directories and files can be observed below:

```
<Directory /var/www/a.com/docs>
       Options Indexes FollowSymLinks
       Order Deny,Allow
       Deny from all
       Allow from 192.168.1.129
</Directory>
```

```
<Files "*.gif">
       Order deny,allow
       Deny from all
       Allow from localhost 192.168.0
 </Files>
```

The use of `.htaccess` file for remote access control should be avoided due to performance constraints. The configuration directives that we might put in the `.htaccess` file to control the access to a directory are the same that we may include in the `<Directory>` or `<File>` contexts. However, this method can be very helpful when we don't have access to the Apache2 main configuration files and need to restrict the access to a specific directory.

11. Redirect

There are many reasons to redirect a request to a different URL or directory. The redirect mechanism can be done by using the directive `Redirect` with the URL or directory to which the requests have to be redirected. The redirects can be permanent or temporary (default). The most appropriate place to apply directive `Redirects` is under the context `<VirtuaHost>` but it may also be applied in the `<Directory>`.

Below you may find two example of redirection:

1) Redirection to an external URL:

```
<VirtualHost *:80>
     [...]
     Redirect "/docs" "http://example.com/two"
</ VirtualHost >
```

In this case the access to the directory `/var/www/docs` will cause an HTTP redirect to the URL `http://example.com/two`.

2) Redirection to a different directory in the same web server:

```
< VirtualHost *:80>
     [...]
     Redirect "/docs" "/docs_new"
</ VirtualHost >
```

In this example the access to the directory `/var/www/docs` will cause an HTTP redirect to the directory `/var/www/docs_new`, in the same web server.

12. HTTP with Secure Socket Layer

The file `/etc/apache2/ports.conf` stores the numeric ports for HTTP requests and for HTTPS requests if module SSL is enabled. Apache2 web server accepts by default HTTP and HTTPS connections on ports 80 and 443, respectively.

To accept HTTPS connections we need to generate a valid certificate and put it available on the Apache2 web server. Apache2 is installed with a certificate for testing purposes, but it is also possible to generate a different signed one, by issuing the command `make-ssl-cert`. You can find below the configuration steps to setup Apache2 to accept HTTPS connections:

1) File `/etc/apache2/ports.conf` has the following typical configuration:

   ```
   Listen 80
   Listen 3333
   <IfModule ssl_module>
       Listen 443
   </IfModule>
   ```

   In this example the Apache2 web server accepts HTTP connections on ports 80 and 3333. If the module `ssl_module` is enabled then Apache2 also listens for HTTPS requests on port 443.

2) File `/etc/apache2/mods-available/ssl.conf` has the configuration directives related with SSL operations. Among the several directives dedicated to SSL configuration, we may highlight those related to the management of X.509 digital certification keys, namely `SSLCertificateFile`, `SSLCertificateKeyFile`.

3) File `/etc/apache2/sites-available/default-ssl.conf` stores the configuration about the virtual host used for HTTPS access. This virtual host is used when the HTTP request starts by `https://`. The configuration directives used for SSL are similar to those explained previously on section 7.

4) After editing the configuration files for Apache2 web server to cope with SSL connections, we need to enable the modules and, finally to reload the Apache2 configuration:

   ```
   a2enmod ssl
   a2ensite default-ssl
   service apache2 reload|restart
   ```

13. Webmin – a web based administration tool

Webmin is a web-based system administration tool for Unix-like systems. It provides an easy alternative to command line system administration and can be used to manage various aspects of a system, such as users and services, through the use of the provided Webmin modules.

To install Webmin via `apt-get`, you must first add the Webmin repository to your sources.list file, by editing the file `/etc/apt/sources.list` and adding the following lines:

```
    deb   http://download.webmin.com/download/repository sarge contrib
    deb   http://webmin.mirror.somersettechsolutions.co.uk/repository
 sarge contrib
```

Now add the Webmin GPG key to `apt`, so the source repository you added will be trusted. To do that, execute the following command:

```
wget -q http://www.webmin.com/jcameron-key.asc -O- | sudo apt-key add -
```

Next, before installing Webmin, you must update package lists with the command:

```
sudo apt-get update
```

Now run this apt-get command to install Webmin:

```
sudo apt-get install webmin
```

After the installation is complete, the Webmin service will start automatically. To test the access to Webmin tool, you have to access to the URL `https://x.y.z.w:10000`, in which `x.y.z.w` is the IP address of the Linux system and `10000` is the default port for webmin.

14. Exercises

Before starting the proposed exercises it'd be important to validate the following check list:

☐ The setup to the exercises is done, with a similar configuration to the one described on Section 2.

☐ Apache2 installation procedure is done (Section 3)

☐ Apache2 daemon is working properly and log files do not have any errors that deserve our attention.

☐ Directories and configuration files were observed in general and there is a good understanding about their meaning and why we need them (Section 4).

☐ There is also a good understanding about the most commonly configuration directives and contexts in Apache2. The official documentation manual (Apache2 documentation web site) is at hand and easily accessible.

☐ The topics covered in this class, described in Sections 6 to 12 are assimilated and there is a good understanding on how to configure them in an Apache2 web server.

The lab exercise described below aims to consolidate the topics covered in this class. Suppose you have a just installed and clean Apache2 web server and you have to configure three sites on it, with different configurations for each one. Below you find some general configuration features and others more specific that should be applied to each site individually:

- Web sites should be accessible by the following FQDN: www.one.com, www.two.com and www.three.com.

- Each site has its own directory to store the web contents.
- In all the sites there should be possible to access to users' home directory, except for user `root`.
- Files with names that match the criteria "`backup*.log`" should be only accessible by IP addresses `192.168.1.0/24`.
- We want to deny remote access to site www.one.com to systems configured with IP addresses in the range `192.168.5.0/24`.
- Site www.two.com should have access via HTTPS. If the user access to http://www.two.com he should be redirected to https://www.two.com and present a secure web page.
- On site www.three.com there is a private area that is only accessible by user "`private`".

15. Bibliography

- **[DOCS]**: Apache HTTP Server: `http://httpd.apache.org/docs/`.
- **[DOCS]**: Full list of books about Apache software foundation (ASF) `http://www.apachebookstore.com/`
- **[BOOK]** *"Apache Cookbook: Solutions and Examples for Apache Administrators"*; Rich Bowen; O'Reilly; 2008 (new edition November 2015)
- **[DOWNLOAD]** VMware Virtualization for Desktop; Server, Application, Public; Hybrid Clouds | United States; `http://www.vmware.com/`
- **[DOWNLOAD]** Download Ubuntu Server | Download | Ubuntu; `http://www.ubuntu.com/download/server`
- **[DOWNLOAD]** Oracle VM VirtualBox; `https://www.virtualbox.org/`