S103 - Report

Installation of a development post

Baptiste DESLOGE
Olivier GABELLE
Doryan JOVANOVIC
Université Paris-Saclay - IUT d'Orsay

16/01/2023

Table des matières

1. Introduction

Welcome to this report. The following report tackles a project that was carried out in groups of three. The group responsible is Olivier GABELLE.

2. Objectives

The purpose of this project is to set up an Apache Server and configure it on a completely fresh Raspberry Pi 400, with nothing in it out of the box. We will document all the steps that are required to reproduce the process. We are intended to be capable of reproducing these steps, and this is why we are evaluated.

3. Installation Steps

3.1. General Setup

First and foremost, we flashed the MicroSD Card by connecting it to an external computer. With the terminal, we typed the command `sudo rpi-imager` to launch the raspberry flashing software with all rights. RPI-Imager is an external software that does not come in a regular OS installation. It needs to be installed before being able to launch it.

We used this software to put the Debian Operating System on the micro-SD card. Once done, we connected it to the raspberry PI 400 in order to boot on it. After having booted, we created a user, set up a password and time zones.

The user is named `olivier`; the password is `toto`.

Once on the desktop, it was necessary to have an internet connection. Thanks to the ethernet cable and the script that was setup by teachers, we were able to be connected to the internet. However, the raspberry is unable to browse the internet if the clock isn't synchronized to the real time zone we are on.

We used the following command `sudo date -s ‘DATE’` in order to set it up manually (necessary at each boot). The date is in the format `‘AAAA-MM-JJ hh:mm:ss’`. We are now able to go on the internet, which we used to install LibreOffice with the command `sudo apt install libreoffice` in order to write this report. We also performed a full update of the software thanks to the commands `sudo apt update` and `sudo apt upgrade`.

3.2. Apache Installation

We now need to install Apache. To do so, we gather the package with the following command, that we can eventually run at super user (`sudo`):

```
sudo apt install apache2
```

After having installed Apache, here are a few information that are useful to know: websites that are recognizable by the Apache Server are stored by default in the following directory:

```
/var/www/html
```

It is actually possible to change it, but for now, we do not need to. By default, there is a custom homepage that Apache hosts on the IP address `127.0.0.1:80` (note that for us, `localhost` is 127.0.0.1, so you may see localhost instead of the IP address is this report).

This page is named `index.html` and is located in the previous folder: `/var/www/html`. We are going to add our personal websites: for the sake of this project, we are going to use Olivier's and Doryan's COIN Project designed earlier in the year, as well as Baptiste's COIN Project and Doryan R102's Project in order to have 3 websites in total.

Using the web browser Chromium, we downloaded the archive and extracted it with the following command in the terminal: `unzip archive.zip`

In order to add more websites: we created three more folders located in `/var`:

```
/var/olivier
/var/doryan
/var/baptiste
```

Once done, we moved all the necessary files in the folders `/var/doryan`, `/var/baptiste` and `/var/olivier`. However, since `/var` is a protected folder of the system, this manipulation needed to be realized in the terminal itself, as a super user (`sudo`). With a simple command, we can move our file:

```
mv myFolder /var/folderName
```

The default website is now accessible through our IP address and port 127.0.0.1:80 because port 80 is configured by default (for other ports, we will need to add some configuration files). Now, we need to set up a VirtualHost configuration to actually have multiple sites at the same time.

We need to tell Apache that it needs to look here. Each website needs to be ran on a different port. Remember that the landing website is hosted on port 80, thus we have chosen ports `81`, `82` and `83` for our websites.

First, we need to edit the port configuration file to tell Apache that a total of four (default + 3) ports are running on the web server. It is located at:

```
/etc/apache2/ports.conf
```

We are adding three lines to this file for each port.

```
NameVirtualHost *:81
Listen 81
NameVirtualHost *:82
Listen 82
NameVirtualHost *:83
Listen 83
```

Now, for each website, a configuration file is required. There are located at this path:

```
/etc/apache2/sites-available
```

We created two more configuration files based on the model of the default one. They are called `doryan.conf`, `baptiste.conf` and `olivier.conf`.

These files start with an anchor named `<VirtualHost *:80>` (default value). We change it respectively to `81`, `82` and `83` for each configuration file. Further in the file, there is a line displaying

```
DocumentRoot /var/www/html
```

It is the default path where Apache is looking in order to find the website associated with this port. We change it to `/var/baptiste`, `/var/doryan` and `/var/olivier` for each website.

Apache needs to take into account these configuration files. Thus, we do the following command to enable these files:

```
sudo a2ensite olivier.conf
sudo a2ensite baptiste.conf
sudo a2ensite doryan.conf
```

We reload Apache with this command:

```
service apache2 reload
```

Apache is now told about our configuration files. We now need to add the websites' directories in the `apache2.conf` file. It is located at the following path:

```
/etc/apache2/apache2.conf
```

A few lines down in the file, there are anchors `<Directory></Directory>` where we can specify some arguments for each website located at a specific directory.

We add an attribute in the opening anchor `<Directory /path/folder>`. Then, we need to specify the option `Require all granted` that allows every user to access our website. Otherwise, they will encounter a 403 Error telling "access denied".

We reload Apache, and now our websites are up.

3.3. Adding Passwords

We can now add a password for each website to secure the access.

We will create three distinct users: `baptiste`, `olivier` and `doryan`. Each password is `toto`.

To do so, we need to perform a prior installation to gather an additional package for Apache: `sudo apt install apache2-utils`

Please note that this package is susceptible to be already installed, in which case this command will be useless. The Apache server needs to read a file that contains our encrypted passwords, each one associated with a username.

```
sudo htpasswd -c /etc/apache2/.htpasswd myUsername
```

The keyword `htpasswd` is a command specific to Apache: we use the option `-c` to create the file if it is our first time. You will not need to add this option for the next users. `.htpasswd` is a hidden file since it is prefixed by a dot (.).

For our project, we will add three users with the following commands:

```
sudo htpasswd -c /etc/apache2/.htpasswd olivier
sudo htpasswd /etc/apache2/.htpasswd doryan
sudo htpasswd /etc/apache2/.htpasswd baptiste
```

After each command, Apache will ask you to provide a password. We put `toto` for each user since we do not need any kind of security regarding this machine, but you should be careful.

If we display the content of the `.htpasswd` file, we can see our three encrypted passwords (you can display it in the terminal with `cat .htpasswd`).

Now, we need to tell Apache that our passwords are located in this prior file. We will need to edit every configuration file that we created to add a small block of information so Apache can target our file. Our configuration files are located at `/etc/apache2/sites-available` as said before.

Using nano, we edit every configuration file needed by adding these lines:

```
<Directory "websitePath">
      AuthType Basic
      AuthName "Message to display to the user"
      AuthUserFile /etc/apache2/.htpasswd
      Require valid-user
</Directory>
```

All highlighted elements should be changed depending on your installation. For instance, the path in the anchor directory should be targeting where your website(s) is/are located. The parameter `AuthType Basic` specifies that the authentication type is the basic, default one of Apache (simple username-password). The `AuthName` can be used to specify information regarding the authentication of the user. Besides, `AuthUserFile` should be targeting the password file that we created earlier, by default in the prior path. Finally, the line `Require valid-user` ensures that the user should be known by Apache in order to access the website wanted.

To apply all these modifications, restart Apache. We have now set up passwords for our websites. Please note that every user present in the `.htpasswd` file can access every website that was configured to target this same `.htpasswd` file.

3.4. Enable SSL for our pages

First and foremost, to use SSL, it needs to be installed on the machine: it is often the case by default on Debian or popular distributions. Then, we need to enable the SSL plugin of Apache with the following command. Restart Apache right after.

```
sudo a2enmod ssl
```

In order to provide a secure connection with the HTTPS norm, we need to have a certificate and give it to Apache. We launch the following command:

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -
keyout /etc/ssl/private/apache-selfsigned.key -out
/etc/ssl/certs/apache-selfsigned.crt
```

Let's explain what is happening here:

- `openssl`: it is the package that we use to launch the command: OpenSSL is a "software" used to manage and create certificates.
- `-req -x509`: is a key for a specific version of our certificate.
- `-nodes`: is used to skip a verification about the security of our certificate.
- `-days 365`: set the time during the certificate will be considered as valid.
- `-newkey rsa:2048`: means that we created a certificate key because we didn't get one before. The key will be created with the RSA encryption method and its size will be defined as 2048 bits.
- `-keyout`: designs the path where the key has been created.
- `-out`: designs the path where the certificate has been created.

This command will ask you for information in the terminal: we need to fill them appropriately, especially `CommonName` which needs to match our `ServerName` specified in our configuration files: for us, it is `localhost`. This command will create a certificate (`.crt`) and a key linked to it (`.key`), that are stocked at `/etc/ssl`.

In Apache, SSL is working through port `443`: we leave it like it. In the default configuration file `000-default.conf`, we are changing the VirtualHost port to `443`: we also need to provide a `ServerName` that is identical to the one provided in the certificate (for us `localhost`). At this point, this file should look like this:

```
<VirtualHost *:443>
        ServerName localhost
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

We are going to add three command lines to enable SSL:

- The first is to enable the module.
- The second is to tell Apache where our certificate is.

- The third is to tell it where our certificate key is.

Add the commands between the `<VirtualHost>` anchor, with the rest:

```
SSLEngine on
SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt
SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key
```

Since port 80 is the default one, port 443 is directly affecting it, which means that we can now access localhost with the prefix HTTPS.

If you want to secure more pages with SSL, you need to add the three SSL command lines to every configuration file associated with a website. For example, we did it for our configuration files `doryan.conf`, `olivier.conf` and `baptiste.conf`.

4. Conclusion

To sum up, we need to remember that the purpose of this project was to set up an Apache server and document each step of the process. But more importantly, we were intended to learn and be able to reproduce this manipulation.

We were a very organized team: since we knew everyone needed to know these steps, we divided the tasks as efficiently as possible. Multiple tasks were repeated three times and since we were three, we decided that every member of the group absolutely had to execute it. Besides, we did every research together in order to cross our sources and verify instructions, mainly to avoid misinformation. Finally, we wrote this report together, by supporting each other’s ideas and adding small things here and there to upgrade its quality. Regarding manipulation, research and recording steps, each member of the group did ⅓ of the work: we spread it evenly. Overall, everything was fine to realise, however we tried to setup a redirection on the HTTP addresses to link them directly to the HTTPS ones, but we did not manage to do it successfully.