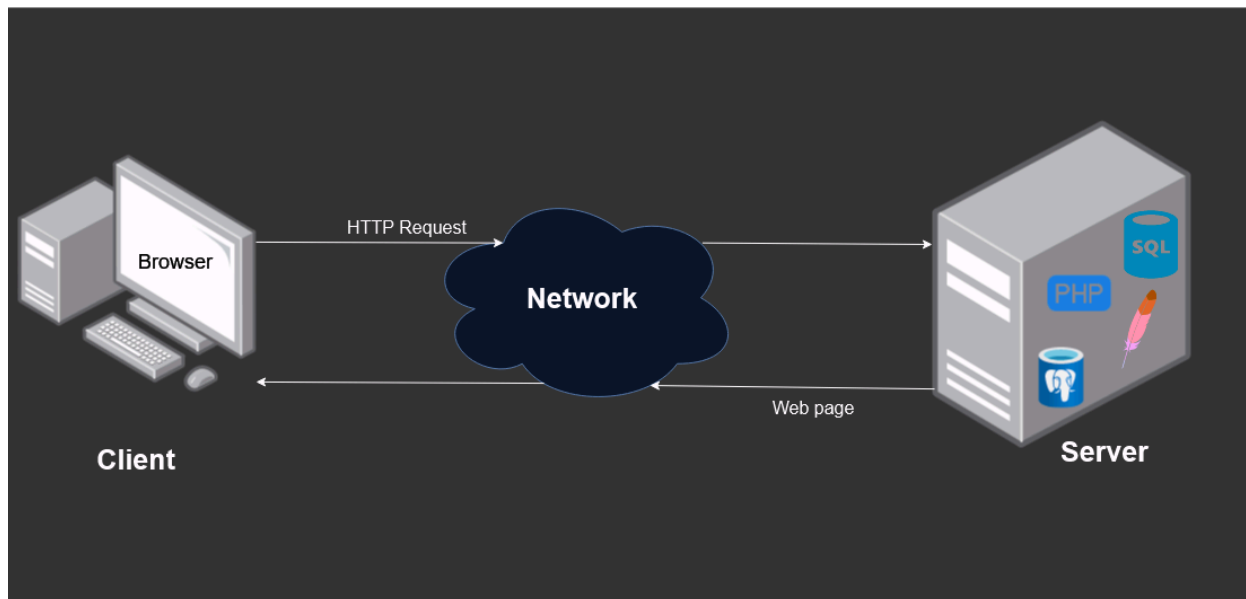










S2.03

Setting up a Web server

By
Tristan Fontanière

B2



Introduction 	2
Set up the Virtual Machine system 	3
Installing the virtual machine	4
Installing the operating system	7
Make secure connection to root 	10
Apache installation 	13
• Apache on HTTP	13
• Upgrade to HTTPS	15
PostgreSQL installation 	21
• Getting a working postgresql server	21
• Manage the connections to PostgreSQL	22
• Starting to fill your base	24
PHP installation 	30
PhpPgAdmin installation 	32
Summary 	35

Introduction



This document is a guide that will teach you how to install a Debian 12 server with functioning Apache, PostgreSQL and PHP. I will be going through every step of the installation and I will take into account every possible issue. I will also add some recommendations especially about security and useful tools.

I recommend you to make every step in the given order considering some services work with one another. Also, some steps are optional and will be explicitly marked as such.

Every important step will be illustrated with screenshots of my personal installation, just adapt it to your own namings.

In the end you should have a functioning server (in this case a virtual machine) that you can access only by giving your personal key and test on any browser on your host machine.

Somes requirements

- ☐ Have your computer running on a linux system.
- ☐ Check your available storage.
- ☐ Read about norms on hosting websites.
- ☐ Learn about some basic linux commands.

Some precisions

- I will use some abbreviations such as **vm** (virtual machine) **psql** (postgresql).
- The use of **\$** means that the command is in a non-root shell.
- The use of **#** means that the command is in a root shell.
- The commands highlighted like `this` are shell commands.
- The commands highlighted like `this` are psql commands.
- When I use for example [username] make sure to replace it with your own case.
- Titles marked with **&&** are optional parts and will not affect the usability of your vm.

Set up the Virtual Machine system



First of all you will need to have a virtual machine running on your computer, notice that the server works only when the virtual machine is on. For this guide, I use a virtual machine to make it easier, as so I will do every manipulation in a localhost situation, this is a way to fake a connection with a real server.

Here I will complete the whole installation on a QEMU virtual machine. So first we need that program.

On a Debian system just run this command to install qemu :

```
# apt-get install qemu-system
```

If you're on a different Operating System or Linux distribution you can check the installation here : <https://www.qemu.org/download/>

Now that Qemu is successfully installed, we will complete the full installation of your virtual machine by using command lines.

Installing the virtual machine

Now you will need to have an iso image for the virtual machine, you can find the official iso image on the debian site right here :

<https://cdimage.debian.org/cdimage/release/current/amd64/iso-cd/>

Here you are going to have to download the file named :

`debian-12.5.0-amd64-netinst.iso`

Our installation is focused on Debian 12 but notice that you can use any version.

Before using it on your virtual machine, check the integrity of the image you have downloaded with :

```
$ sha512sum debian-12.5.0-amd64-netinst.iso
```

Check if the result of this command is the exact same character sequence as the one in the file `SHA256SUMS` in the site.

Now we will have to install the virtual machine with the image. Place yourself in the directory where you decided to store the iso image and use the following command

```
$ qemu-img create [your image].img 4G
```


For some explanations : qemu creates a new image on the file you give. The parameter “4G” specifies that this image will be created with a size of 4 gigabytes.

Your machine is not running yet but don't worry we are almost there ! The last step here is to start the qemu virtual machine, we will set up all the settings we want.

In my example I will use different settings on my virtual machine but don't hesitate to look for these settings if you want to change some values (especially about the ports forwarding).

The following command helps you start your virtual machine :

```
$ qemu-system-x86_64
```

 But don't use it alone !

We will now go through all the important arguments to specify and I will give you the command to paste ;


- **-machine q35** (it is a kind of vm used by qemu that allows us to specify all the others arguments)
- **-cpu host** (it tells the vm to use the same cpu features as the host)
- **-m 4G** (it tells the vm to use 4 gigabyte of memory from the host)
- **-enable-kvm** (it allows the vm to use a specific feature of the cpu that greatly upgrade the performances of the vm)
- **-device VGA,xres=1024,yres=768** (it allows the vm to emulate a gpu and display graphical applications; the rest is the resolution)
- **-display gtk,zoom-to-fit=off** (it tells the qemu to use gtk+ to display the vm; also disable the resize)
- **-drive format=raw,file=[your_image.img],discard=unmap** (it tells the vm that the image used is raw, file is an absolute path to your image, and unmap is useful to clean some space on the disk)
- **-device e1000,netdev=net0** (e1000 is a kind of network card and net0 is the name given to the virtual network device and linked to e1000)
- **-netdev**

user,id=net0,hostfwd=tcp::2222-:22,hostfwd=tcp::4443-:443,hostfwd=tcp::8080-:80,hostfwd=tcp::5432-:5432

(the argument "user" allows the vm to directly access to the host's network, it also links this configuration to the virtual network created before. All the `hostfwd` are tcp ports forwarding, for example, the port 2222 of the vm will use the port 22 of the machine)

And with all of that, the command is :

```
$ qemu-system-x86_64 -machine q35 -cpu host -m 4G -enable-kvm -device VGA,xres=1024,yres=768 -display gtk,zoom-to-fit=off -drive format=raw,file=[your_image.img],discard=unmap -device e1000,netdev=net0 -netdev user,id=net0,hostfwd=tcp::2222-:22,hostfwd=tcp::4443-:443,hostfwd=tcp::8080-:80,hostfwd=tcp::5432-:5432
```

 Don't forget to replace `your_image` and check for potential spaces (never more than two) !

And that's it, your virtual machine should be starting !

Since it's kind of a difficult and long command, I suggest you create a script to start your vm. To do a script you can simply type this (in your host) :

```
$ echo "[your_command]" >> virtual-machine-start.sh
```

And then make the script usable by adding `"#!/bin/bash"` at the beginning typing :

```
$ chmod +x virtual-machine-start.sh
```

Just restart your shell and you can use your script by typing the path to it, just like this :

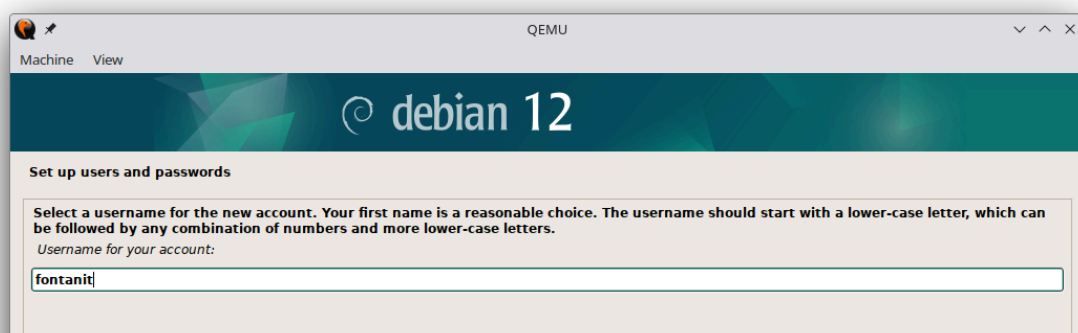
```
$ ./virtual-machine-start.sh
```

Installing the operating system

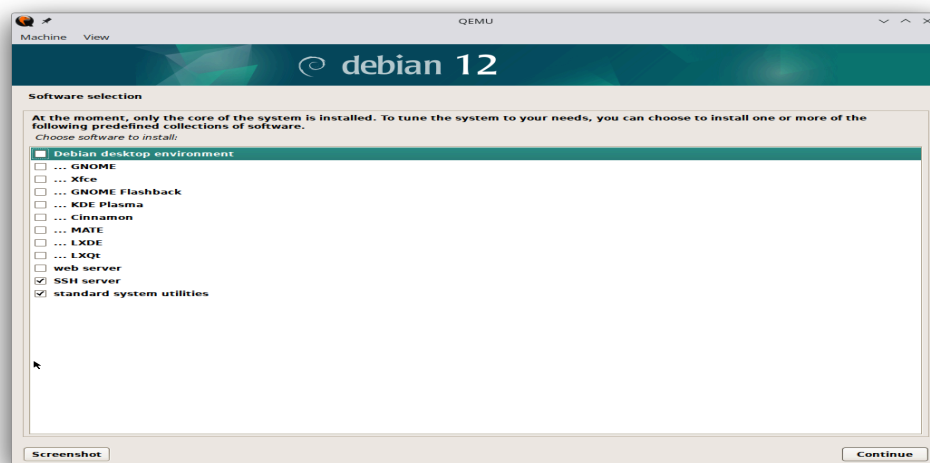
We can now focus on setting up and installing the operating system with the right settings.

The installation is very simple, just follow the steps given by Debian and enter your own informations. Here are the fields that are important :

- Root Password : choose a simple one since it's a vm.
- Username : the name of the user you will use.(see the screen below)



- User Password : once again, a simple one.
- Partition disk : **YES**
- Software selection : make sure to uncheck “debian desktop environment” and check “ssh server” (just like the screen below).



- Device for boot loader : /dev/sda

All the other fields are at your own goodwill.

Before doing anything, restart the vm by connecting to root (the username is “root” and the password is the one you defined) or if you already connected to your user use :

```
$ su -
```

Enter the root password.

Before using the vm, we need to restart it, use this :

```
# poweroff
```

Now restart your vm by using the script or the command we saw.

Now you can connect with your user. First thing first, we can try to make connections with the outside, for example you can use :

```
$ ping 8.8.8.8 -c 10
```

This will send 10 ICMP packets to the server of google, if you see answers scrolling by, this is good you can connect outside your vm !
If not, there may be issues in the command you use to start your vm.

We will now make some tests to check if ssh works correctly.
First let's connect **from your host** on your vm, we need to do that in non-root so just use :

```
$ ssh [your_username]@localhost -p 2222
```

You can now use this command to switch to the root shell (while being connected to your vm on ssh) :

```
$ su -
```

Just use the password you defined at first. And now to check if everything is working, install any package by using :

```
# apt install [package_name]
```

You can also take note of your network characteristics by using :

```
# ip addr
```

Also check that you have no Xorg server on your machine, since this is a vm that is supposed to be a server, a graphical server would take too much space and drastically reduce the security of the machine. To check this :

```
# dpkg -l | grep xorg
```

You should get no result.

This is it for the setup of the machine we are going to use. Now let's focus on the different services that we need and want to run for our server. I will go through all of them with every step of their setup.


Make secure connection to root

In this part we will focus on adding a bit of security to our server, what we will do is to allow connections to the root shell by using a unique private key. With that you will be able to access the shell root directly from your host without having to use your user as a gateway.

First you can check if ssh is running correctly, because we will need it a lot, by using :

```
# systemctl status ssh
```

This is an example of what answer you should get if it works correctly.



```
fontanit@server-fontanit:~$ systemctl status ssh
• ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; preset: enabled)
   Active: active (running) since Fri 2024-05-03 16:25:31 CEST; 11min ago
     Docs: man:sshd(8)
           man:sshd_config(5)
   Main PID: 468 (sshd)
      Tasks: 1 (limit: 4644)
     Memory: 6.6M
        CPU: 82ms
    CGroup: /system.slice/ssh.service
            └─468 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

Warning: some journal files were not opened due to insufficient permissions.
fontanit@server-fontanit:~$ _
```

If the service hasn't start correctly, you can easily start it with this command :

```
# systemctl start ssh
```

Now let's focus on setting up the private key. **On your host machine** place yourself in the directory of your choice (by convention it's in `~/ .ssh/`) and generate a key with :

```
$ ssh-keygen -t rsa
```

For the file where you want to save the key, if you do not enter anything it will be created with the default name. Same for the passphrase, press enter to create no passphrase.

You now have two files : your `id_rsa` (private key) and `id_rsa.pub` (public key).

The next step is to copy the public key in the server for example use :

```
$ scp id_rsa.pub [username]@localhost:~/
```

This will put the file in the space of your user. Now go back on your vm and this is what you wanna do :

1. Connect to root with `$ su -`
2. Copy the key to the `authorized_keys` of the root user with :

```
# cat /home/[username]/id_rsa.pub >>  
/root/.ssh/authorized_keys
```

Well done ! Now let's just simplify the connection with a script **on your host**. In the directory you want (for example create a `~/scripts`) and write the following script :

```
#!/bin/bash  
ssh-add /home/username/scripts/id_rsa  
ssh root@localhost
```

Just paste this text in a file like `connect.sh`

Here the username is the one of your host and the **root** is the user you want to connect with this script. And yes, add the **private** key not the public in this script !

Make the script usable by running :

```
$ chmod +x connect.sh
```

Now restart your shell to take this into account, and then use the script. If you are connected on root after entering the passphrase (or not), you have succeeded !

To make the ssh connections even better we can turn off connections by password on the root user. We just need to change a rule in a configuration file, edit this file while being on root in your vm :

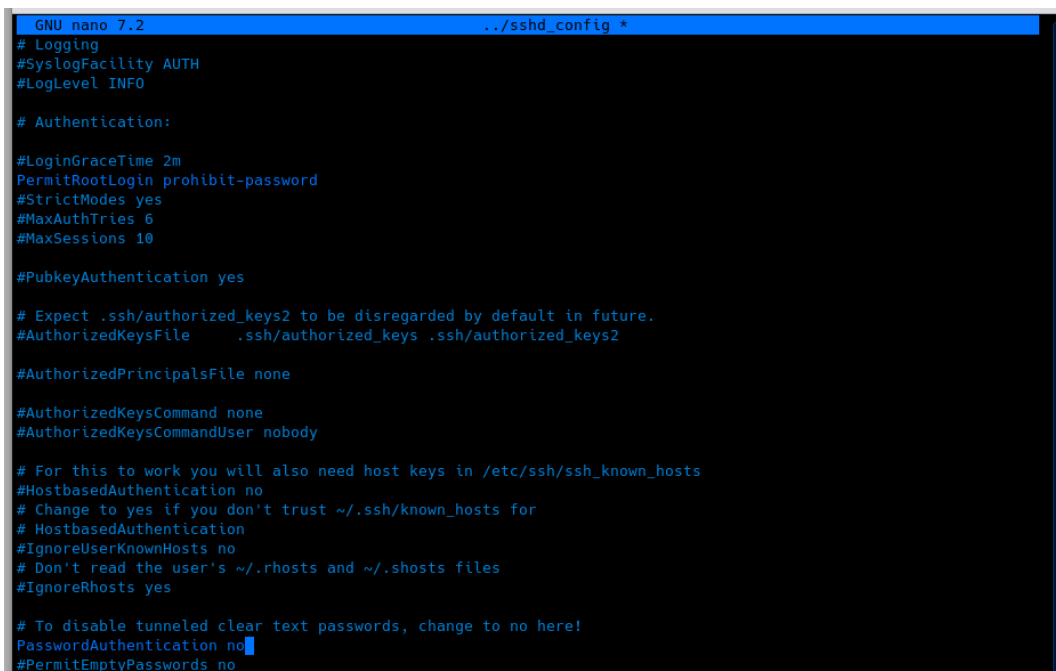
```
# nano /etc/ssh/sshd_config
```

And then you need to update or uncomment these two lines :

```
PermitRootLogin  
PasswordAuthentication
```

The first one needs to be set as “**prohibit-password**” and the second one needs to be set as “**no**”. Their use is pretty obvious, I will let you understand it.

Now your file should be like this :



```
GNU nano 7.2          ../sshd_config *  
# Logging  
#SyslogFacility AUTH  
#LogLevel INFO  
  
# Authentication:  
#LoginGraceTime 2m  
PermitRootLogin prohibit-password  
#StrictModes yes  
#MaxAuthTries 6  
#MaxSessions 10  
  
#PubkeyAuthentication yes  
  
# Expect .ssh/authorized_keys2 to be disregarded by default in future.  
#AuthorizedKeysFile      .ssh/authorized_keys .ssh/authorized_keys2  
  
#AuthorizedPrincipalsFile none  
  
#AuthorizedKeysCommand none  
#AuthorizedKeysCommandUser nobody  
  
# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts  
#HostbasedAuthentication no  
# Change to yes if you don't trust ~/.ssh/known_hosts for  
# HostbasedAuthentication  
#IgnoreUserKnownHosts no  
# Don't read the user's ~/.rhosts and ~/.shosts files  
#IgnoreRhosts yes  
  
# To disable tunneled clear text passwords, change to no here!  
PasswordAuthentication no  
#PermitEmptyPasswords no
```

This is it for the ssh connections, you can also change the root by your vm username to connect to a user only environment and not a root environment by ssh.

Apache installation



- Apache on HTTP

Apache is the service that allows you to host websites on HTTP. We almost only see HTTPS websites so in this part I will also cover the way to easily host a https website.

The official Apache documentation is available here :

<https://httpd.apache.org/>

First thing we need to install Apache, since we are using a Debian system we just need these 2 commands to install and start Apache :

```
# apt install apache2
# service apache2 start
```

(note : when the system asks for confirmation, just type “y” to accept)

Check if Apache was correctly started with :

```
# systemctl status apache2
```

Just like ssh you should get that kind of answer..

```
Machine View
QEMU
root@server-fontanit:~# systemctl status apache2
• apache2.service - The Apache HTTP Server
  Loaded: loaded (/lib/systemd/system/apache2.service; enabled; preset: enabled)
  Active: active (running) since Fri 2024-05-03 16:31:59 CEST; 53s ago
  Docs: https://httpd.apache.org/docs/2.4/
  Main PID: 1001 (apache2)
  Tasks: 55 (limit: 4644)
  Memory: 9.4M
  CPU: 34ms
  CGroup: /system.slice/apache2.service
          └─1001 /usr/sbin/apache2 -k start
            └─1003 /usr/sbin/apache2 -k start
              └─1004 /usr/sbin/apache2 -k start

May 03 16:31:59 server-fontanit systemd[1]: Starting apache2.service - The Apache HTTP Server...
May 03 16:31:59 server-fontanit apachectl[1000]: AH00558: apache2: Could not reliably determine the server's fully qualified do
May 03 16:31:59 server-fontanit systemd[1]: Started apache2.service - The Apache HTTP Server.
lines 1-16/16 (END)
```

If the service hasn't start correctly, you can easily start it with this command :

```
# systemctl start apache2
```

You can now check if the apache server answers correctly. We will need to make a simple http query by using `telnet`, in order to do this we are going to use the localhost on the port 80 (default port). Just do the following steps :

```
$ telnet localhost 80
```

The shell is now waiting for you to type something, write this :

```
HEAD / HTTP/1.0
```

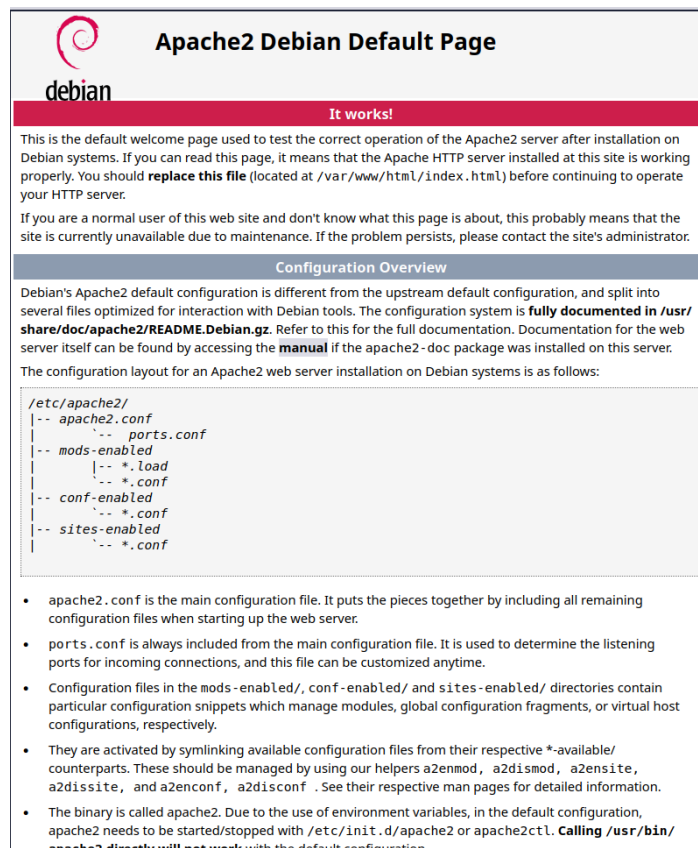
And then press enter two times.

If Apache is working, you are going to get the following answer from the command :

```
HTTP/1.1 200 OK
```

Since we redirected the port 80 of the server on the port 8080 of the host machine, we can access the apache menu on a web browser at this address : <http://localhost:8080>

You should get this page displayed if it's working.



Apache2 Debian Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Debian's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Debian tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Debian systems is as follows:

```

/etc/apache2/
|-- apache2.conf
|   |-- ports.conf
|-- mods-enabled
|   |-- *.load
|   |-- *.conf
|-- conf-enabled
|   |-- *.conf
|-- sites-enabled
|   |-- *.conf

```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the `mods-enabled/`, `conf-enabled/` and `sites-enabled/` directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective `*-available/` counterparts. These should be managed by using our helpers `a2enmod`, `a2dismod`, `a2ensite`, `a2dissite`, and `a2enconf`, `a2disconf`. See their respective man pages for detailed information.
- The binary is called `apache2`. Due to the use of environment variables, in the default configuration, `apache2` needs to be started/stopped with `/etc/init.d/apache2` or `apache2ctl`. Calling `/usr/bin/apache2` directly will not work with the default configuration.

● Upgrade to HTTPS

For security purposes, most of the websites nowadays are on HTTPS (the exchange of messages from the protocol are encrypted). Here I am going to cover the way to get on HTTPS **unofficially** by creating an auto-signed certificate.

There are ways to make it official but for our local vm we are going to stay in this simpler method.

We need to first go in `/var/www/` and change some rights. In order to check the rights use :

```
# ls -l
```

The default rights are displayed on the left but we need to change them. They need to belong to the user `www-data`, which is the user identified by `apache2`. You should have this display for now :


```
root@server-fontanit:/var/www# ls -l
total 4
drwxr-xr-x 2 root root 4096 May 28 08:43 html
root@server-fontanit:/var/www#
```

Here html belongs to root.

and then use these commands :

```
# chown -R www-data:www-data html/
# chmod -R u+rwX html/
# chmod -R g-rwx html/
# chmod -R o-rwx html/
```

These commands do two different things : first give the property of html to www-data, and then rewrite the good rights. The -R means “recursive”.

You should now get this display :

```
root@server-fontanit:/var/www# chown -R www-data:www-data html/
root@server-fontanit:/var/www# chmod -R u+rwX html/
root@server-fontanit:/var/www# chmod -R g-rwx html/
root@server-fontanit:/var/www# chmod -R o-rwx html/
root@server-fontanit:/var/www# ls -l
total 4
drwxrwx--- 2 www-data www-data 4096 May 28 08:43 html
root@server-fontanit:/var/www#
```

Now www-data and its linked group have all the rights on the directory that contains the site.

I will now take care of the configuration required to get the auto signed https. I will do it for the default site given by Apache which is `/var/www/html/` but it always works the same for any site you host.

First we need to set up ssl keys. In order to do that, go to `/etc/ssl/`
In this directory, type the following command :

```
# openssl genrsa -out index.key 4096
```

This will create a key that will be used for the certificate.
Now let's create the certificate itself :

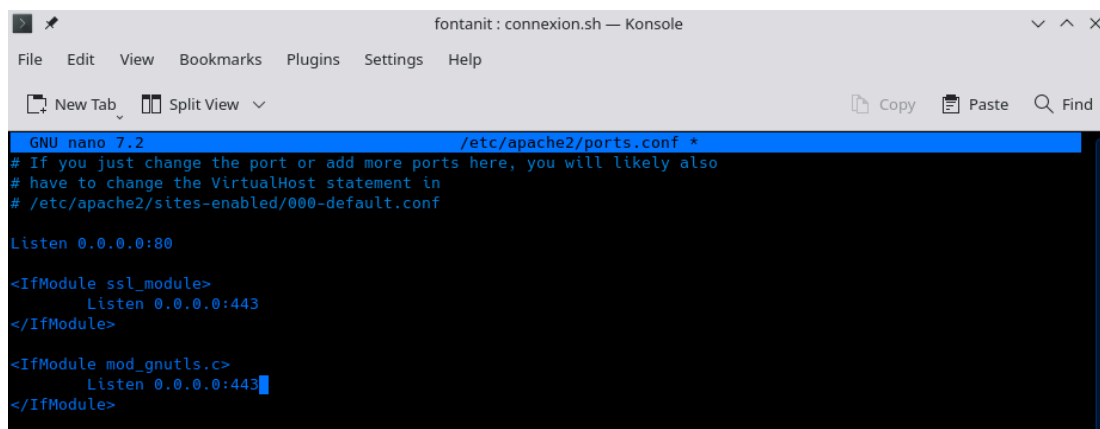
```
# openssl req -new -x509 -days 3650 -key  
index.key -out index.pem
```

No information is truly important knowing we are on a vm, the only thing
you need to type right is the **Common name** here you need to type
localhost , for other uses you will need to type your domain name.

Then we need to edit the file that defines the different ports of apache,
as always do :

```
# nano /etc/apache2/ports.conf
```

Normally you just need to add the different **0.0.0.0:** , just make it like the
screen :



```
fontanit : connexion.sh — Konsole
File Edit View Bookmarks Plugins Settings Help
New Tab Split View Copy Paste Find
GNU nano 7.2 /etc/apache2/ports.conf *
# If you just change the port or add more ports here, you will likely also
# have to change the VirtualHost statement in
# /etc/apache2/sites-enabled/000-default.conf
Listen 0.0.0.0:80

<IfModule ssl_module>
    Listen 0.0.0.0:443
</IfModule>

<IfModule mod_gnutls.c>
    Listen 0.0.0.0:443
</IfModule>
```

Now let's create the configuration files for our supposed site.
Use this command to go in the right directory we need to work on :

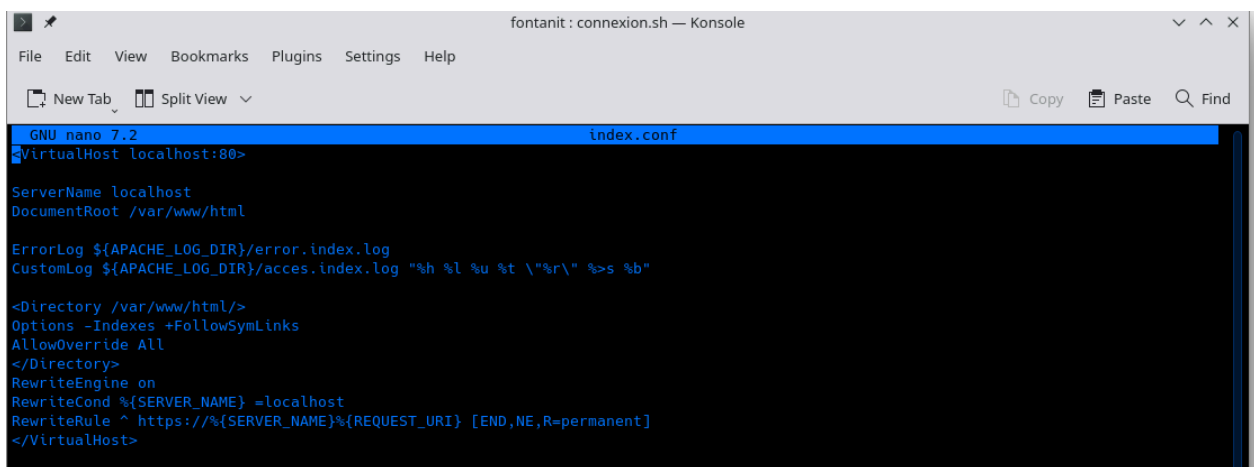
```
# cd /etc/apache2/sites-available/
```

We need to create two configuration files, one which is the default one, and one which is specifically for the https. Follow these two commands :

```
# echo " " >> index.conf
# echo " " >> index-ssl.conf
```

We need now to fill these files with the right configuration rules, I will give them to you but not go through the details. It's just about Apache configuration and not important to understand in our case.

Firstly, index.conf (with `nano` of course) :



```
fontanit : connexion.sh — Konsole
File Edit View Bookmarks Plugins Settings Help
New Tab Split View
GNU nano 7.2 index.conf
VirtualHost localhost:80>
ServerName localhost
DocumentRoot /var/www/html
ErrorLog ${APACHE_LOG_DIR}/error.index.log
CustomLog ${APACHE_LOG_DIR}/access.index.log "%h %l %u %t \"%r\" %>s %b"
<Directory /var/www/html/>
Options -Indexes +FollowSymLinks
AllowOverride All
</Directory>
RewriteEngine on
RewriteCond %{SERVER_NAME} =localhost
RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]
</VirtualHost>
```

```
<VirtualHost localhost:80>

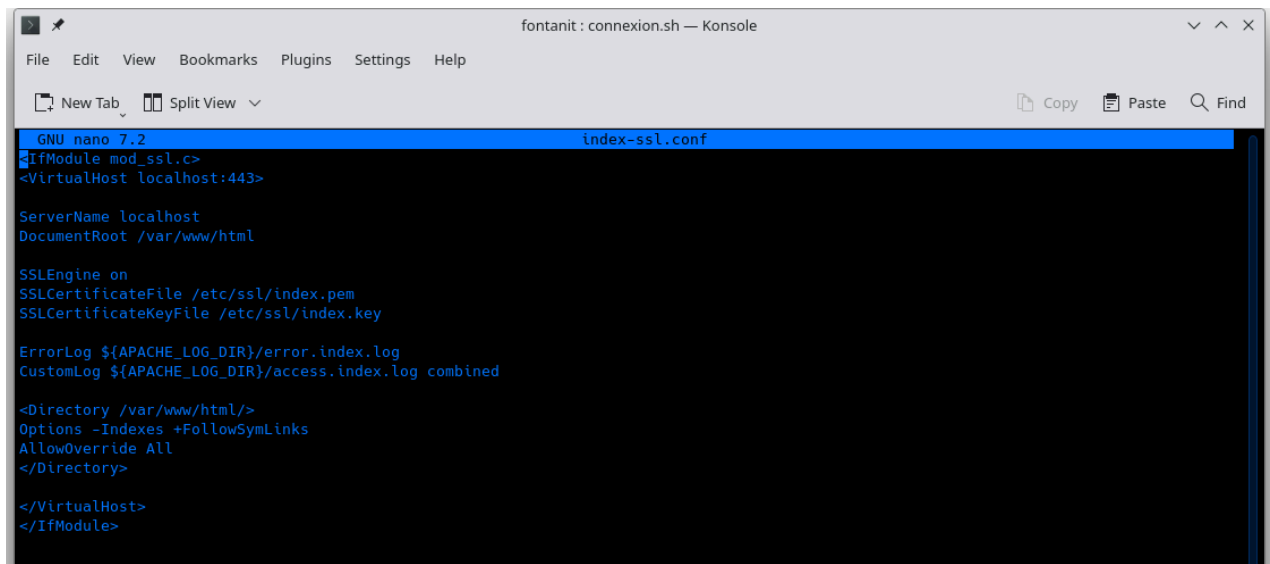
ServerName localhost
DocumentRoot /var/www/html

ErrorLog ${APACHE_LOG_DIR}/error.index.log
CustomLog ${APACHE_LOG_DIR}/access.index.log "%h %l %u %t \"%r\" %>s %b"

<Directory /var/www/html/>
Options -Indexes +FollowSymLinks
AllowOverride All
</Directory>
RewriteEngine on
```

```
RewriteCond %{SERVER_NAME} =localhost
RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI}
[END,NE,R=permanent]
</VirtualHost>
```

Secondly, index-ssl.conf :



The screenshot shows a terminal window titled 'fontanit : connexion.sh — Konsole'. The window contains the nano 7.2 editor editing the file 'index-ssl.conf'. The content of the file is as follows:

```
GNU nano 7.2 index-ssl.conf
<IfModule mod_ssl.c>
<VirtualHost localhost:443>

ServerName localhost
DocumentRoot /var/www/html

SSLEngine on
SSLCertificateFile /etc/ssl/index.pem
SSLCertificateKeyFile /etc/ssl/index.key

ErrorLog ${APACHE_LOG_DIR}/error.index.log
CustomLog ${APACHE_LOG_DIR}/access.index.log combined

<Directory /var/www/html/>
Options -Indexes +FollowSymLinks
AllowOverride All
</Directory>

</VirtualHost>
</IfModule>
```

```
<IfModule mod_ssl.c>
<VirtualHost localhost:443>

ServerName localhost
DocumentRoot /var/www/html

SSLEngine on
SSLCertificateFile /etc/ssl/index.pem
SSLCertificateKeyFile /etc/ssl/index.key

ErrorLog ${APACHE_LOG_DIR}/error.index.log
CustomLog ${APACHE_LOG_DIR}/access.index.log combined

<Directory /var/www/html/>
Options -Indexes +FollowSymLinks
AllowOverride All
</Directory>

</VirtualHost>
</IfModule>
```

I used Rewrite on certain Apache rules, this is a mod that is not enabled by default on a machine, we need to activate it by using :

```
# a2enmod rewrite
```

Now that this is made, we should be able to enable our site by using this command :

```
# a2ensite index-ssl.conf
```

The last step is to restart apache to apply everything we did :

```
# systemctl restart apache2
```

Now just try to access your site at the new address which should work : <https://localhost:4443/>

You will get an error page on some browsers because it's an unofficial certificate but just accept it anyways.

Yes we need to specify 4443 because we made a port forwarding when we started the machine.

PostgreSQL installation

- Getting a working postgresql server

PostgreSQL is a database service that allows you to manipulate data easily, it can also be used in web services as we will see later. Its official documentation is available here :

<https://www.postgresql.org/docs/>

On Debian you can once again install this service with a simple command :

```
# apt install postgresql
```

To check if postgres was successfully installed you can try to connect on the postgres user with :

```
# su - postgres
```

Use `exit` right after.

To check if PostgreSQL service has correctly started :

```
# systemctl status postgresql
```

And just like other services you should get this.

```
root@server-fontanit:/home/fontanit/.ssh# cd
root@server-fontanit:~# systemctl status postgresql
• postgresql.service - PostgreSQL RDBMS
   Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; preset: enabled)
   Active: active (exited) since Mon 2024-05-06 12:27:04 CEST; 23min ago
   Process: 591 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
   Main PID: 591 (code=exited, status=0/SUCCESS)
   CPU: 1ms

May 06 12:27:04 server-fontanit systemd[1]: Starting postgresql.service - PostgreSQL RDBMS...
May 06 12:27:04 server-fontanit systemd[1]: Finished postgresql.service - PostgreSQL RDBMS.
root@server-fontanit:~#
```

If the service hasn't start correctly, you can easily start it with this command :

```
# systemctl start postgresql
```

In order to check if the database is working correctly, we will add some users and tables into a specific database a bit later. First connect to the database with (on the postgres user) :

```
# psql
```

If you get connected to postgresql, the databases are accessible for the postgres user, that's good but now we are going to manage connections first.

- Manage the connections to PostgreSQL

In order to allow all users you are going to create in the future to use the database and allow connections from the host machine, we will have to edit some configuration files and change their settings. This part is really precise and will work only by following the steps.

To allow users who are not postgres to connect to the database, make sure to be on the root user. First we will edit the file :

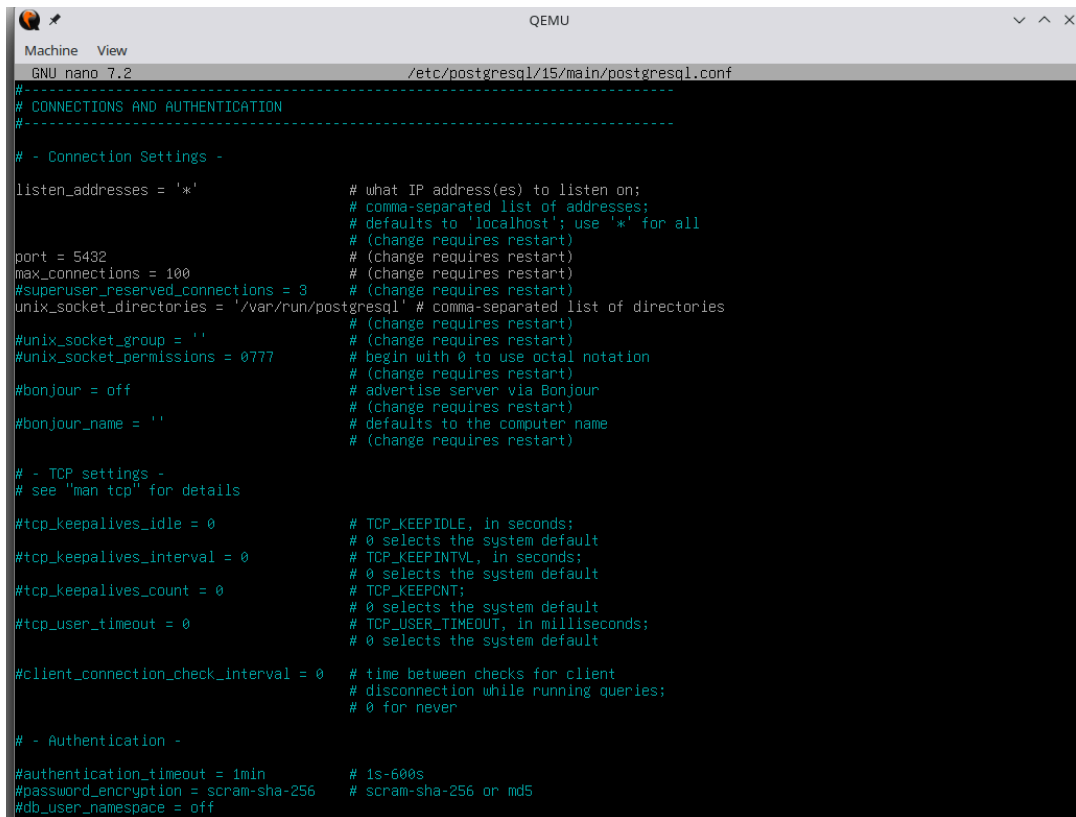
`/etc/postgresql/15/main/postgresql.conf`

You can use any shell text editor but I recommend using `nano`. Use this command :

```
# nano /etc/postgresql/15/main/postgresql.conf
```

Look for the line where you can read "listen_adresses= localhost" and edit it to : `listen_adresses= '*'`

Just like that.



```

Machine View
GNU nano 7.2 /etc/postgresql/15/main/postgresql.conf
#
# CONNECTIONS AND AUTHENTICATION
#
# - Connection Settings -
listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for all
                                # (change requires restart)
port = 5432                     # (change requires restart)
max_connections = 100           # (change requires restart)
#superuser_reserved_connections = 3 # (change requires restart)
unix_socket_directories = '/var/run/postgresql' # comma-separated list of directories
                                # (change requires restart)
#unix_socket_group = ''         # (change requires restart)
#unix_socket_permissions = 0777 # begin with 0 to use octal notation
                                # (change requires restart)
#bonjour = off                  # advertise server via Bonjour
                                # (change requires restart)
#bonjour_name = ''              # defaults to the computer name
                                # (change requires restart)
#
# - TCP settings -
# see "man tcp" for details
#tcp_keepalives_idle = 0        # TCP_KEEPIDL, in seconds;
                                # 0 selects the system default
#tcp_keepalives_interval = 0    # TCP_KEEPINTVL, in seconds;
                                # 0 selects the system default
#tcp_keepalives_count = 0       # TCP_KEEPCNT;
                                # 0 selects the system default
#tcp_user_timeout = 0           # TCP_USER_TIMEOUT, in milliseconds;
                                # 0 selects the system default
#client_connection_check_interval = 0 # time between checks for client
                                # disconnection while running queries;
                                # 0 for never
#
# - Authentication -
#authentication_timeout = 1min   # 1s-600s
#password_encryption = scram-sha-256 # scram-sha-256 or md5
#db_user_namespace = off

```

This will allow postgres to “listen” to any ip addresses, even those that are not local.

The second step is to define an authentication rule, in this case postgres will accept every user created with a password. Edit the next file `/etc/postgresql/15/main/pg_hba.conf`

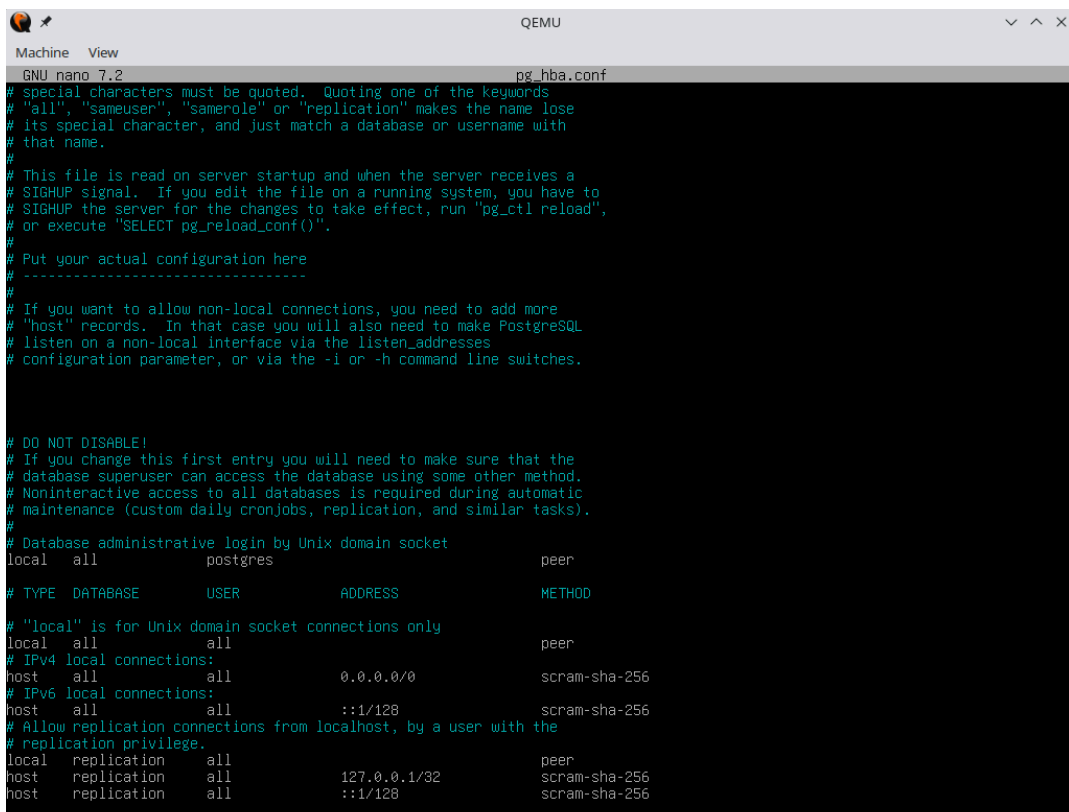
Once again use `nano` to edit it.

```
# nano /etc/postgresql/15/main/pg_hba.conf
```

Look for the line saying “IPv4”

And edit it to : `host all all 0.0.0.0/0 scram-sha-256`

Like this.

A screenshot of a QEMU window titled 'QEMU' with a 'Machine View' tab. Inside, a terminal window titled 'GNU nano 7.2' shows the 'pg_hba.conf' file. The file contains configuration for PostgreSQL authentication, including comments about special characters, server startup, and a table of connection settings. The table has columns for TYPE, DATABASE, USER, ADDRESS, and METHOD. The settings include local connections, IPv4 and IPv6 host connections, and replication connections.

```
Machine View
pg_hba.conf
# special characters must be quoted.  Quoting one of the keywords
# "all", "sameuser", "samerole" or "replication" makes the name lose
# its special character, and just match a database or username with
# that name.
#
# This file is read on server startup and when the server receives a
# SIGHUP signal.  If you edit the file on a running system, you have to
# SIGHUP the server for the changes to take effect, run "pg_ctl reload",
# or execute "SELECT pg_reload_conf()".
#
# Put your actual configuration here
# -----
#
# If you want to allow non-local connections, you need to add more
# "host" records.  In that case you will also need to make PostgreSQL
# listen on a non-local interface via the listen_addresses
# configuration parameter, or via the -i or -h command line switches.
#
# DO NOT DISABLE!
# If you change this first entry you will need to make sure that the
# database superuser can access the database using some other method.
# Noninteractive access to all databases is required during automatic
# maintenance (custom daily cronjobs, replication, and similar tasks).
#
# Database administrative login by Unix domain socket
local  all             postgres             peer
# TYPE  DATABASE      USER      ADDRESS              METHOD
# "local" is for Unix domain socket connections only
local  all             all             peer
# IPv4 local connections:
host   all             all             0.0.0.0/0             scram-sha-256
# IPv6 local connections:
host   all             all             ::1/128               scram-sha-256
# Allow replication connections from localhost, by a user with the
# replication privilege.
local  replication     all             peer
host   replication     all             127.0.0.1/32          scram-sha-256
host   replication     all             ::1/128               scram-sha-256
```

After all these modifications just restart postgres to apply them :

```
# systemctl restart postgresql
```

You should now be able to connect to the database with any user, but to achieve this you need first to perform the next part where we will create a user for yourself.

- Starting to fill your base

In order to check if the database is working correctly, we will add some users and tables into a specific database. First connect to the postgres database while being on the postgres user with :

```
# psql
```

Now we will create a user for yourself, we try to avoid using postgres user for future manipulations due to the fact that he has too many rights;

```
CREATE USER [your_username] WITH CREATEDB  
CREATEROLE PASSWORD '[your_password]';
```

Here you created a user who has your name, he can create a database, create roles and has a specific password.

Now you can create the database that user will own :

```
CREATE DATABASE mybase WITH  
OWNER = '[your_username]';
```

You can now leave the postgres user by using `exit`.

The next step is to fill in the database a bit for further tests, starting by connecting to the database we created previously with your user with this syntax :

```
$ psql mybase
```

note : the user you created in postgres must have the same username as your linux user for this command to work, if not, when you want to connect use this syntax :

```
$ psql -U [postgresql_username] mybase
```

To change the current database while being connected in postgres you can use this postgres command :

```
\c [base_name] [username]
```

To fill the base I suggest you to use a simple script, before connecting to postgresql, and while being connected to your linux user, type this in whatever directory :

```
$ echo "CREATE TABLE test
(name varchar PRIMARY KEY,
id int);
INSERT INTO test values('Jean',1);
INSERT INTO test values('Homer',2);"
>> create.sql
```

This script is just an example, you can change whatever values and names. Now stay in the directory and connect to postgres properly, with this command or the one on the previous page :

```
$ psql mybase
```

You can now execute the script you wrote by using :

```
\i create.sql
```

Now let's try to make some requests from the vm and from the host machine to test our database.

First on your virtual machine connect to your base :

```
$ psql mybase
```

Then, make a simple request on the tables we created such as :

```
SELECT * FROM test;
```

You should get all the data you entered in your script just like the screen underneath.



```
fontanit@server-fontanit:~$ psql mabase
psql (15.6 (Debian 15.6-0+deb12u1))
Type "help" for help.

mabase=> SELECT * FROM test;
 nom | id 
-----+---
 Jean |  1 
 Homer |  2 
(2 rows)

mabase=>
```

If you don't get the expected results, try to :

1. Check your script.
2. Check if the table is really in the right base (with `\d` for example).

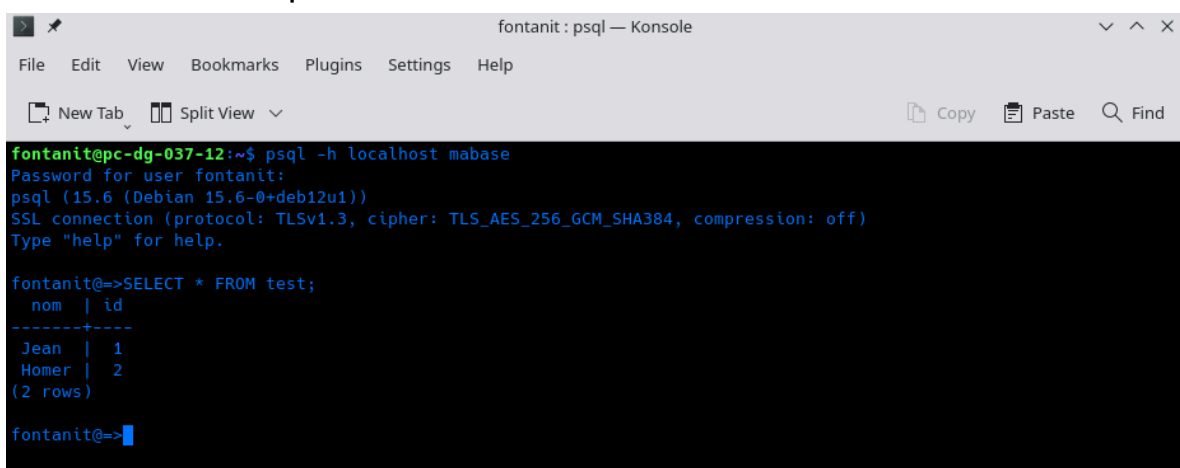
We will now repeat the same tests on our host. Start a shell.

To connect to the database **from the host** use :

```
$ psql -h localhost -U [username]
```

The username must be the one you created a postgres user for.

Use the same request we used in the vm. The result should be this :



```
fontanit@pc-dg-037-12:~$ psql -h localhost mabase
Password for user fontanit:
psql (15.6 (Debian 15.6-0+deb12u1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

fontanit@=>SELECT * FROM test;
 nom | id 
-----+---
 Jean |  1 
 Homer |  2 
(2 rows)

fontanit@=>
```

If you are unable to connect, try to :

1. Check the configuration files.
2. Check if your host is correctly connected to the vm.
3. Install postgres on your host if it's not the case.

Your base is now created, you can now check some things to make sure you have done everything correctly and that your postgres is well organized.

First use this command in your database to check if your user is really the owner (this time from the vm) :

```
\d
```

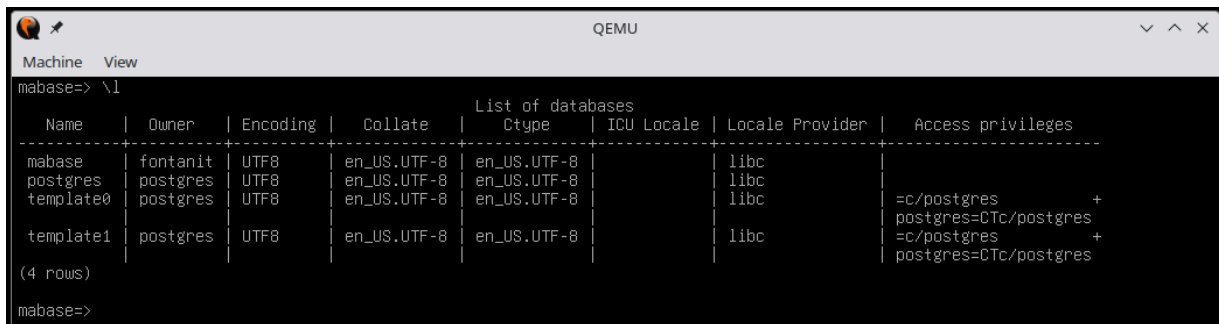
```
fontanit@server-fontanit:~$ psql mabase
psql (15.6 (Debian 15.6-0+deb12u1))
Type "help" for help.

mabase=> \d
      List of relations
 Schema | Name | Type  | Owner
-----+-----+-----+-----
 public | test | table | fontanit
(1 row)
```

You can also check the list of bases on your server with :

```
\l
```

Do not modify the default bases that postgres created. The important columns are “name”, “owner” for our use, and I won’t insult you by explaining their use ;-)



```
QEMU
Machine View
mabase=> \l
      List of databases
 Name | Owner | Encoding | Collate | Ctype | ICU Locale | Locale Provider | Access privileges
-----+-----+-----+-----+-----+-----+-----+-----
 mabase | fontanit | UTF8 | en_US.UTF-8 | en_US.UTF-8 | en_US.UTF-8 | libc | 
 postgres | postgres | UTF8 | en_US.UTF-8 | en_US.UTF-8 | en_US.UTF-8 | libc | =c/postgres +
 template0 | postgres | UTF8 | en_US.UTF-8 | en_US.UTF-8 | en_US.UTF-8 | libc | postgres=CTc/postgres
 template1 | postgres | UTF8 | en_US.UTF-8 | en_US.UTF-8 | en_US.UTF-8 | libc | =c/postgres +
(4 rows)
mabase=>
```

You can also check and see all the authorized users with their passwords in a specific table (The passwords are encrypted for security purposes).

This table can be consulted only if you have the right privileges, I suggest you use the user postgres just as I did underneath.

```
SELECT * FROM pg_shadow;
```

As an example, this is what I get.

```
postgres=# SELECT * FROM pg_shadow ;
username | usesysid | usecreatedb | usesuper | use repl | usebypassrls |          | valuntil | useconfig
-----+-----+-----+-----+-----+-----+-----+-----+-----
postgres |      10 | t           | t        | t        | t          |          |          |
fontanit |   16387 | t           | f        | f        | f          | SCRAM-SHA-256$4096:BAua8Aw3SXdpfhD8Et010A==$aWN0nFd1faI
3h0xNI1Fd1NuamE53Hp25pAcorDA2CrgM=:b+wp9ughKj18nYLRR/9R2/zMiDug2VmVYoLt9KDCdRY= |          |
(2 rows)
```

nb : postgres has no password to limit the access to this user, indeed he is the superuser and has too many rights.

PHP installation

Php is a web programming language made for dynamic pages, it is often used to have interaction between a server and a client.

Its official documentation is available here :

<https://www.php.net/docs.php>

In this part we will add php service to our apache2 service to make sure our server can read and execute php files for any website we could host.

Once again the installation is pretty simple on Debian :

```
# apt install php-common libapache2-mod-php  
php-cli
```

To test the good functioning of php we will try to open a php file that we write in `/var/www/html/` , write the following command to create a php file :

```
# echo "<?php phpinfo();phpinfo(INFO_MODULES);  
?>" >> info.php
```

A quick note : you must be in `/var/www/html/` because this is the place where apache reads the different base sites.

In this case we only host one site, it would be more complicated with multiple sites we would need different configuration files. Also, this directory is owned by the user **www-data** as we saw in the https part.

Now, on your **host** machine try to access : <http://localhost:8080/info.php>

If you can access it, php is working !

Just another example of php test I made is to display on a web page informations about my system using different shell commands such as :

whoami (current username), **who** (currently connected user), **/sbin/blkid** (display disks), **ip addr** (networks interfaces), **dpkg -l | grep "apache"** (display all the packages of apache), **systemctl status apache2** (display the status of apache2 service), **systemctl status postgresql** (display the status of postgresql service), and then dpkg and systemctl for ssh service.

Following that, once again I place this file in `/var/www/html/` , then I need to use `/sbin/blkid` , and then open the file on http://localhost:8080/page_sae_S2.03.php

And this is the result I get.

```
Bonjour

Je suis www-data

Qui est connecté ?

fontani1 tty1      Jun  5 08:00
root              Jun  5 09:46 (10.0.2.2)

Mes disques sont

Mes interfaces

1: lo: mtu 65536 qlist noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s2: mtu 1500 qlist fq_codel state UP group default qlen 1000
    Link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s2
        valid_lft 8631sec preferred_lft 70705sec
    inet6 fe80::5054:ff:fe12:3456/64 scope site dynamic enp0s2addr
        valid_lft 8631sec preferred_lft 16271sec
    inet6 f080::5054:ff:fe12:3456/64 scope link
        valid_lft forever preferred_lft forever

My apache install is

ii apache2                2.4.59-1-deb12u1      amd64      Apache HTTP Server
ii apache2-bin            2.4.59-1-deb12u1      amd64      Apache HTTP Server (modules and other binary files)
ii apache2-data           2.4.59-1-deb12u1      all        Apache HTTP Server (common files)
ii apache2-utils          2.4.59-1-deb12u1      amd64      Apache HTTP Server (utility programs for web servers)
ii libapache2-mod-php     8.2.2+0-1
ii libapache2-mod-php8.2  8.2.18-1-deb12u1     amd64      server-side, HTML-embedded scripting language (Apache 2 module)

My apache status is

* apache2.service - The Apache HTTP Server
Loaded: loaded (/lib/systemd/system/apache2.service; enabled; preset: enabled)
Active: active (running) since Wed 2024-06-05 09:23:04 CEST; 3min ago
Docs: https://httpd.apache.org/docs/2.4/
Process: 2284 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
Main PID: 2288 (apache2)
Tasks: 10 (limit: 4044)
Memory: 10.0M
CPU: 31ms
CGroup: /system.slice/apache2.service
├─2288 /usr/sbin/apache2 -k start
├─2289 /usr/sbin/apache2 -k start
├─2290 /usr/sbin/apache2 -k start
├─2291 /usr/sbin/apache2 -k start
├─2292 /usr/sbin/apache2 -k start
├─2293 /usr/sbin/apache2 -k start
├─2294 /usr/sbin/apache2 -k start
├─2295 /usr/sbin/apache2 -k start
├─2441 /usr/sbin/apache2 -k start
├─2562 sh -c 'systemctl status apache2'
└─2563 systemctl status apache2

My postgresql install is

ii postgresql              15-248              all        object-relational SQL database (supported version)
ii postgresql-15          15.6-0-deb12u1      amd64      The World's Most Advanced Open Source Relational Database
ii postgresql-client-15   15.6-0-deb12u1      amd64      front-end programs for PostgreSQL 15
ii postgresql-client-common 248                all        manager for multiple PostgreSQL client versions
ii postgresql-common      248                all        PostgreSQL database-cluster manager

My postgresql status is

* postgresql.service - PostgreSQL RDWMS
Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; preset: enabled)
Active: active (exited) since Wed 2024-06-05 08:03:11 CEST; 1h 50min ago
Process: 681 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
Main PID: 681 (code=exited, status=0/SUCCESS)
CPU: 2ms

My ssh install is

ii libssh2-1:amd64        1.10.0-3+b1         amd64      SSH2 client-side library
ii openssh-client         1:9.7p1-2+deb12u2   amd64      secure shell (SSH) client, for secure access to remote machines
ii openssh-server         1:9.7p1-2+deb12u2   amd64      secure shell (SSH) server, for secure access from remote machines
ii openssh-sftp-server    1:9.7p1-2+deb12u2   amd64      secure shell (SSH) sftp server module, for SFTP access from remote machines
ii task-ssh-server        3.15                all        SSH server

My ssh status is

* ssh.service - OpenSSH Secure Shell server
Loaded: loaded (/lib/systemd/system/ssh.service; enabled; preset: enabled)
Active: active (running) since Wed 2024-06-05 08:03:28 CEST; 1h 50min ago
Docs: man:ssh(8)
     man:ssh_config(5)
Process: 475 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
Main PID: 507 (sshd)
Tasks: 1 (limit: 4044)
Memory: 0.0M
CPU: 47ms
CGroup: /system.slice/ssh.service
├─507 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"
```


PhpPgAdmin installation



In this part we will focus on installing and setting up PhpPgAdmin, it is a service that allows you to control your databases (with PostgreSQL) directly in a webpage. It is used to make your life easier while developing a website which needs to access databases, for example to manage different users.

The official documentation of this service is available here :

<https://github.com/phpPgAdmin/phpPgAdmin>

On a debian system you can easily install the packages with :

```
# apt install phpPgAdmin php-pgsql -y
```

In order to allow the use of PhpPgAdmin on a web interface, there are a couple of modifications we need to do.

- You need to access the file `Connection.php` to check its location you can use :

```
# find / -name Connection.php
```

This command will look for the file in the whole tree structure of your machine.

Normally the file is in `/usr/share/phpPgAdmin/classes/database/`
Just as we did before edit the file :

```
# nano  
/usr/share/phpPgAdmin/classes/databse/  
Connection.php
```

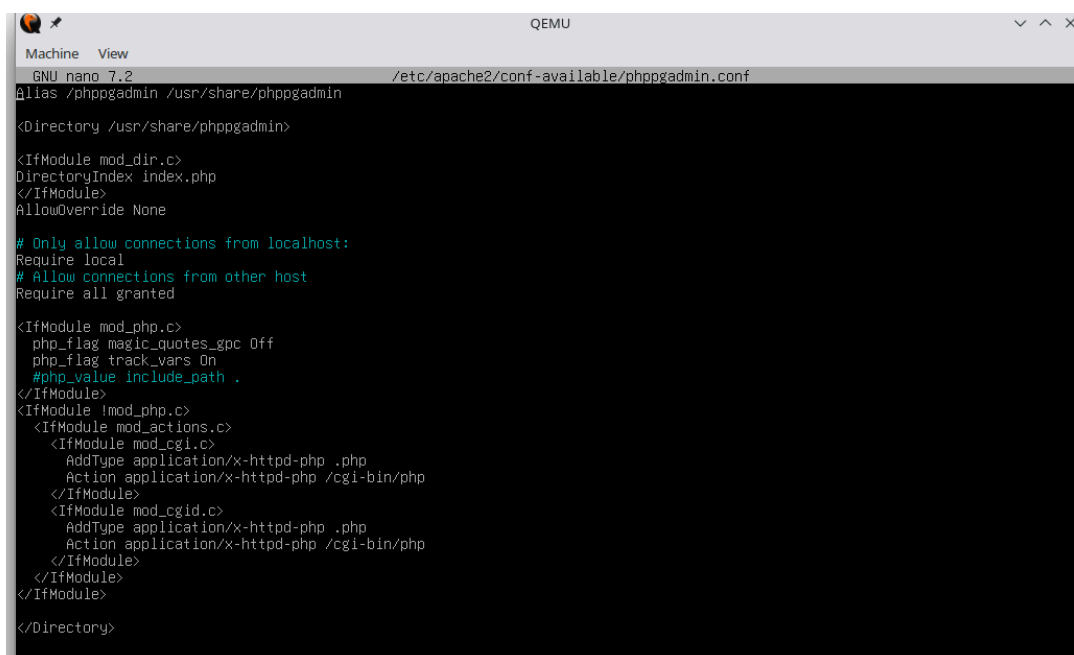
Look for the line saying `case '14' : return 'Postgres'; break;` and change the “14” to “15”. This will allow the service to take into account the version 15 of postgres.

- Now we need to authorize remote connections. The file we need for this is in `/etc/apache2/conf-available/`

Precisely there should be a file called `phppgadmin.conf`, so once again edit it :

```
# nano
/etc/apache2/conf-available/phppgadmin.conf
```

In the file there should be a line saying `Require local`, just under it add a new line saying `Require all granted`.



```
Machine View
GNU nano 7.2 /etc/apache2/conf-available/phppgadmin.conf
Alias /phppgadmin /usr/share/phppgadmin

<Directory /usr/share/phppgadmin>

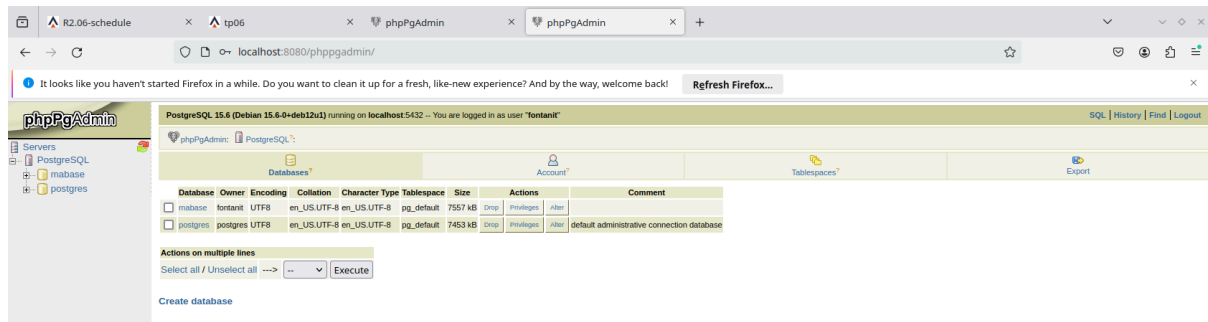
<IfModule mod_dir.c>
DirectoryIndex index.php
</IfModule>
AllowOverride None

# Only allow connections from localhost:
Require local
# Allow connections from other host
Require all granted

<IfModule mod_php.c>
php_flag magic_quotes_gpc Off
php_flag track_vars On
#php_value include_path .
</IfModule>
<IfModule !mod_php.c>
<IfModule mod_actions.c>
<IfModule mod_cgi.c>
AddType application/x-httpd-php .php
Action application/x-httpd-php /cgi-bin/php
</IfModule>
<IfModule mod_cgid.c>
AddType application/x-httpd-php .php
Action application/x-httpd-php /cgi-bin/php
</IfModule>
</IfModule>
</IfModule>
</Directory>
```

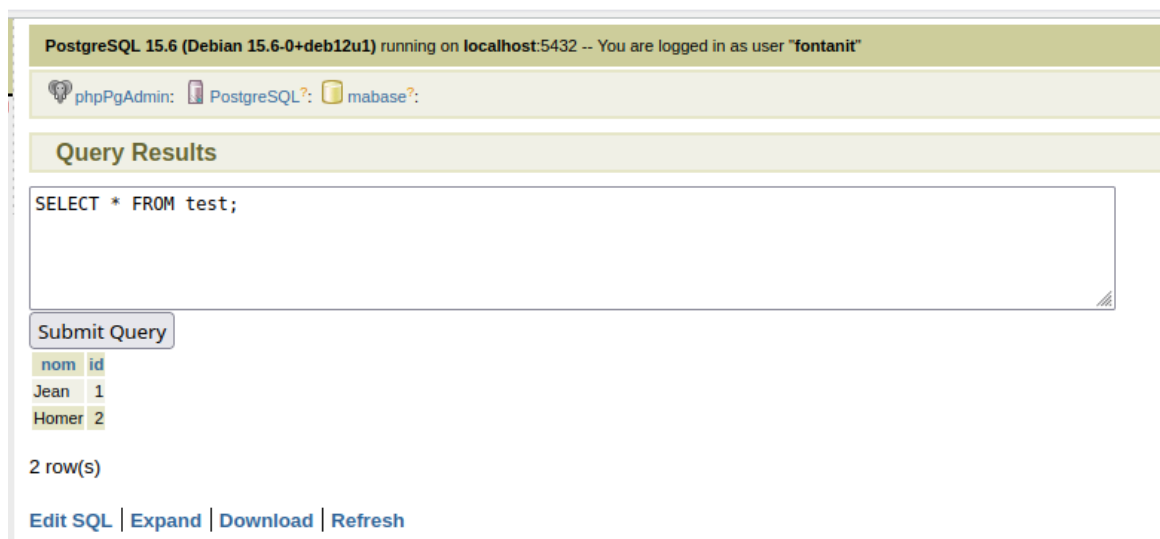
You can now access your PhpPgAdmin space at <http://localhost:8080/phppgadmin/>

When arriving at this address, in the left menu, click on “**PostgreSQL**”, the one with a red cross. Enter your username with its password. You should get this page displayed.



Just click on “mybase” to enter your database. Then, click on the first column named “schema”. Finally click on “browse” in the column named “actions”. This is the default place to make queries in your database.

And thanks to PhpPgAdmin you can now make queries in your base from the graphical interface just like this.



This really makes the database easier to manipulate !

Summary



As a general idea, here is the space taken on my vm after all the installations and manipulations I did :

```

Machine View
QEMU

Debian GNU/Linux 12 server-fontanit tty1
server-fontanit login: fontanit
Password:
Linux server-fontanit 6.1.0-18-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.76-1 (2024-02-01) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri May 24 15:07:22 CEST 2024 on tty1
fontanit@server-fontanit:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            1.9G   0    1.9G   0% /dev
tmpfs           392M  476K  392M   1% /run
/dev/sda1       3.0G  2.5G  335M  89% /
tmpfs           2.0G  1.1M  2.0G   1% /dev/shm
tmpfs           5.0M   0    5.0M   0% /run/lock
tmpfs           392M   0    392M   0% /run/user/1000
fontanit@server-fontanit:~$ _

```

To check your space you can use :

```
$ df -h
```

You can also check the assembly and partitions by using :

```
# cat /etc/fstab
```

Here is what I have in this file.

```

root@server-fontanit:~# cat /etc/fstab
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# systemd generates mount units based on this file, see systemd.mount(5).
# Please run 'systemctl daemon-reload' after making changes here.
#
# <file system> <mount point> <type> <options> <dump> <pass>
# / was on /dev/sda1 during installation
UUID=b0307148-d4c1-40e0-9648-3496e7218355 / ext4 errors=remount-ro 0 1
# swap was on /dev/sda5 during installation
UUID=ab664790-2b0f-4353-8873-0db9b41c6e91 none swap sw 0 0
/dev/sr0 /media/cdrom0 udf,iso9660 user,noauto 0 0
root@server-fontanit:~# _

```

One last thing concerning security, I suggest you to install this debian package :

```
# apt install lynis
```

It's a security tool that can display all the potential dangerous services running on your machine. You can use it by running the command :

```
# lynis audit system
```

And then it's pretty simple to read, the services marked in red are potentially dangerous, at the end you have the “**warnings**” which are really bad for the machine. It also gives information about how to solve those problems. I will let you go through this service on your own because it would need an entire guide !

This is what is displayed at the end of the command :

```
-[ Lynis 3.0.8 Results ]-

Warnings (1):
-----
! Couldn't find 2 responsive nameservers [NETW-2705]
  https://cisofy.com/lynis/controls/NETW-2705/

Suggestions (54):
-----
* This release is more than 4 months old. Check the website or GitHub to see if there is an update available.
[LYNIS]
  https://cisofy.com/lynis/controls/LYNIS/

* Install libpam-tmpdir to set $TMP and $TMPDIR for PAM sessions [DEB-0280]
  https://cisofy.com/lynis/controls/DEB-0280/

* Install apt-listbugs to display a list of critical bugs prior to each APT installation. [DEB-0810]
  https://cisofy.com/lynis/controls/DEB-0810/

* Install needrestart, alternatively to debian-goodies, so that you can run needrestart after upgrades to determine which daemons are using old versions of libraries and need restarting. [DEB-0831]
  https://cisofy.com/lynis/controls/DEB-0831/

* Install fail2ban to automatically ban hosts that commit multiple authentication errors. [DEB-0880]
  https://cisofy.com/lynis/controls/DEB-0880/

* Set a password on GRUB boot loader to prevent altering boot configuration (e.g. boot in single user mode without password) [BOOT-5122]
  https://cisofy.com/lynis/controls/BOOT-5122/

* Consider hardening system services [BOOT-5264]
  - Details : Run '/usr/bin/systemd-analyze security SERVICE' for each service
  https://cisofy.com/lynis/controls/BOOT-5264/

* If not required, consider explicit disabling of core dump in /etc/security/limits.conf file [KRNL-5820]
  https://cisofy.com/lynis/controls/KRNL-5820/

* Check process listing for processes waiting for IO requests [PROC-3614]
  https://cisofy.com/lynis/controls/PROC-3614/

* Configure password hashing rounds in /etc/login.defs [AUTH-9230]
  https://cisofy.com/lynis/controls/AUTH-9230/

* Install a PAM module for password strength testing like pam_cracklib or pam_passwdqc [AUTH-9262]
```

This is just an example, Lynis displays a lot more infos that I will not detail.

You can also make sure to have frequent security updates for example by using crontab. It is a service that runs specific tasks in the background at regular periods of time that you can set up. It should be installed by default but if not, you know the song :

```
# apt install cron
```

Then write a script for security updates, for example :

```
# echo "apt update && apt upgrade -y && apt  
clean && apt autoclean" >> security.sh
```

The "-y" is an argument to automatically say yes to every confirmation.

Then edit it with `nano` to add **#!/bin/bash** at the beginning that will make it an understandable script by your machine. Like this :



```
GNU nano 7.2 security.sh  
#!/bin/bash  
apt update && apt upgrade -y && apt clean && apt autoclean
```

Don't forget to make it executable with `chmod` as always.

And now with `crontab` use :

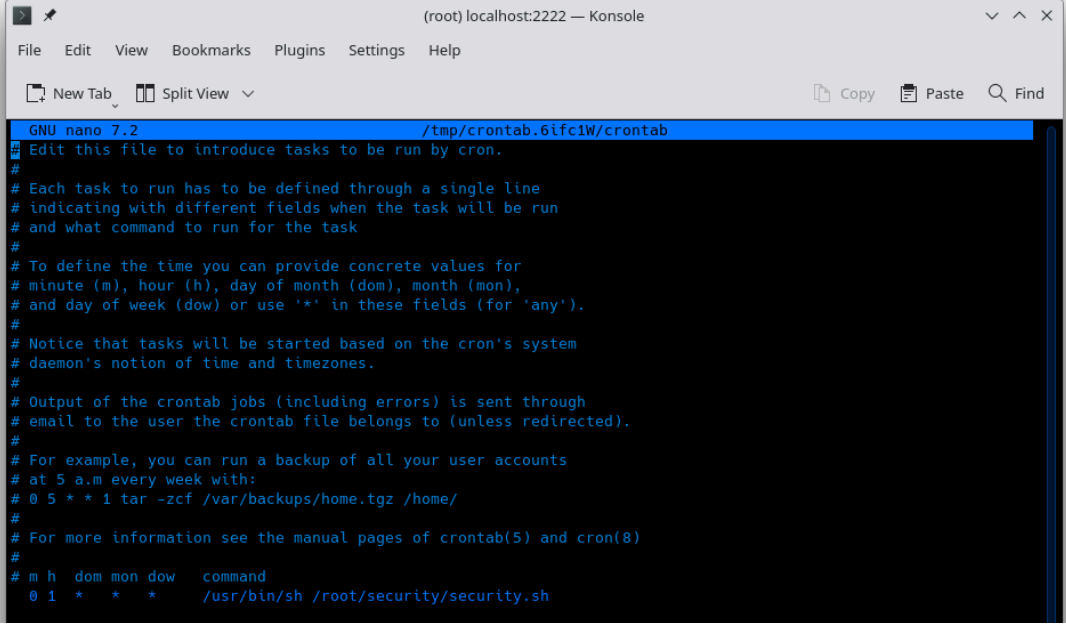
```
# crontab -e
```

After using this command, crontab will ask you to choose an editor, choose the one you want, I chose nano as always.

Here you wanna add a line at the very end, it should look like this :

```
# m h dom mon dow    command  
0 1 * * * /usr/bin/sh /root/scripts/security.sh
```

Personally my file looks like this :



```
GNU nano 7.2 /tmp/crontab.6ifc1W/crontab
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
0 1 * * * /usr/bin/sh /root/security/security.sh
```

Don't forget it's a script so make it executable as I explained earlier !

Some explanations : this configuration says that every day at 1am, the script will be executed and make the security updates. In this case it's a vm so it's not that useful, but for a server that runs 24/7 it would work !

You are now ready to set up your own vm to run any type of website no matter the language used. You can also manage your own databases either on a graphical web interface or by using commands. You also have a way to connect to your vm from your host and a little bit of security !