Installation guide of a Debian 12 LAMP Server

With Apache, PostgreSQL and PHP

Timothé Jacob

IUT2 Université Grenoble Alpes

Table of contents

Introduction **2**

Setting up the virtual machine **3**

Installing the virtual machine **4**

First boot **7**

Check the internet connectivity 7

Check if Xorg is installed 8

Check if the SSH server is working properly 8

Install the softwares **10**

Install Apache 10

Install PostgreSQL 11

Installation 11

Basic configuration 12

Open an external access 13

Last checks 15

Install PHP 16

Install PhpPgAdmin 17

Quick security analysis **19**

Annex documents **20**

Introduction

Hello and welcome to this guide on setting up a **Debian 12 server** with **Apache**, **PHP** and **PostgreSQL** installed.

Throughout this tutorial, we will **explain** each step in depth so that you can understand what you are doing, allowing you to modify the system or extend its functionalities later if you wish.

This type of installation is called a **LAMP** installation. It stands for **Linux**, **Apache**, **MySQL**, **PHP**. In this tutorial, we'll use PostgreSQL instead of MySQL, but the principle remains the same.

You might ask, "**Why** this installation? Why choose these softwares?"

First, **Debian** is a recognized OS for its efficiency, its security and its speed. It's perfect for hosting softwares that needs to run all-day long. Also, it can be installed **without any graphical interface**, which makes everything a lot faster.

Apache is the most popular **web-server**, known for its performance. It's essential.

PHP will allow us to generate dynamic contents for web applications. In our case, we'll use it to manage a database on a web page.

And finally, **PostgreSQL** has advanced features for creating and managing databases, and is also powerful.

In this guide, I'm using a **virtual machine**, but this works the same way for an actual server. You'll just have to adjust the IP Addresses (especially the "localhost" ones).

Setting up the virtual machine

First, we need to set up the virtual machine. To perform this, we will use **QEMU**, an open-source emulation software. **Install it.**
When QEMU is installed, open a **terminal**. From there, you have to type a command, where you can customize the parameters to adjust it to your needs.

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 $drive -device
e1000,netdev=net0 -netdev user,id=net0,hostfwd=tcp::2222-:22,
hostfwd=tcp::4443-:443,hostfwd=tcp::8080-:80,hostfwd=tcp::5432-:5432
```

To make it clear, each parameter is bold. Here's the explanation of this command :

- The base of the command is **`qemu-system-x86_64`**
- **`-machine`** is to select the virtual machine model. In our case, it's preferable to use the `Q35` model, which is modern and fast.
- **`-cpu`** is the parameter to select the CPU. In most cases, `host` is the best solution, since you probably don't have multiple CPUs.
- **`-m`** is to set how much memory you want to allocate to your machine. Adjust this value to your needs. For the tutorial, 4 gigabytes is largely enough.
- **`-enable-kvm`** will enable hardware virtualization support KVM (Kernel-based Virtual Machine), thus providing improved virtualization performance.
- -**`device`** is used to add and configure virtual devices within the VM. Here, we're specifying a VGA graphics adapter for the virtual machine with a resolution of 1024x768 pixels
- **`-display`** configures the graphical display. We are using the GTK graphical interface, and disabling zoom-to-fit, to avoid graphical conflicts.
- **`-drive`** is used to set on which drive you're storing your machine. Replace `$drive` by your drive name. Make sure a Debian image is in.
- **`-device`**, that we're using again to set a virtual network interface.
- **`-netdev`** is used to configure a virtual network. "`user`" is the type, and "`net0`" is the name. Then, each "`hostfwd`" is used to forward ports, which is essential if you do it on a VM, because port conflicts could happen.

Execute this prompt, and wait for the virtual machine to set up. Then start it.

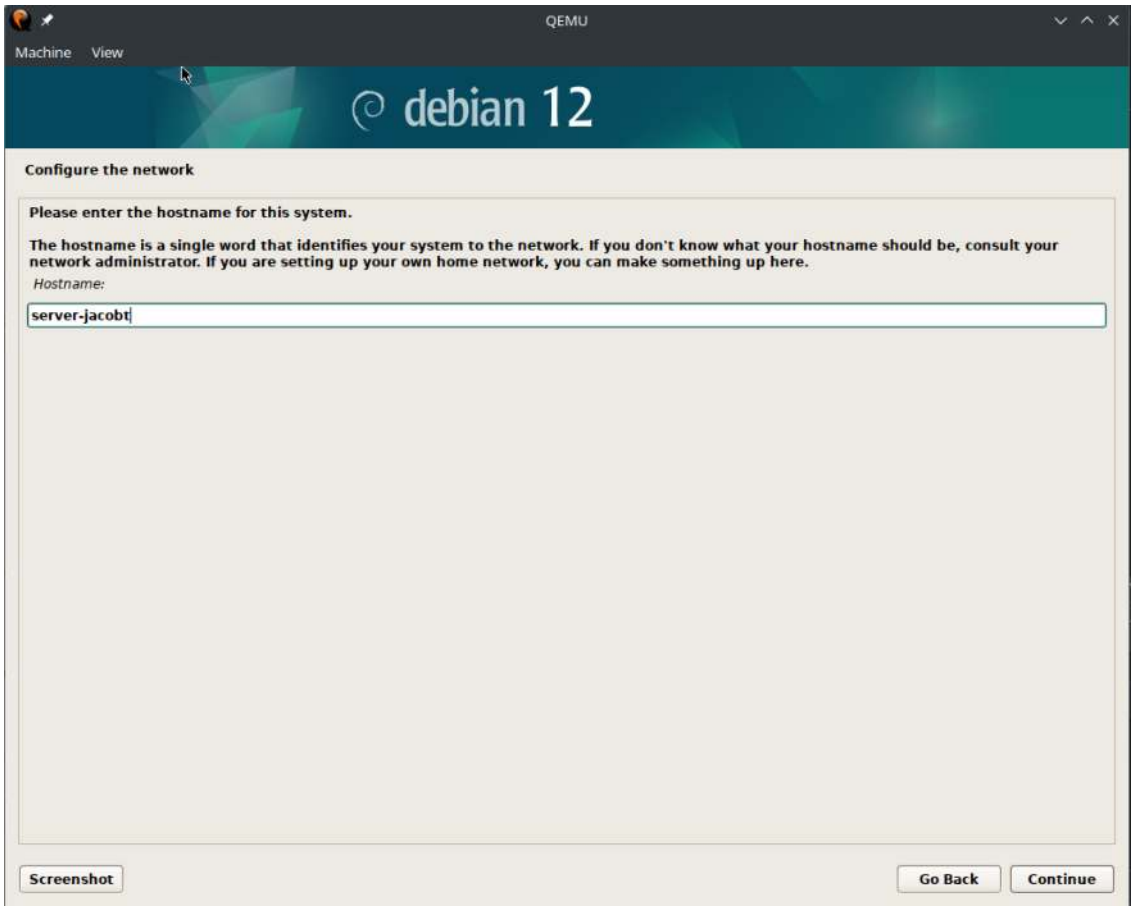
Installing the virtual machine

Once the virtual machine is created and started, you should see the **Debian installer** screen. If not, make sure a Debian image is in the drive you set up. You can download it here.

I recommend you choose the **graphical installation**. It's more comfortable. The graphical part is **only** for the installation. You'll have the choice later to enable or not the GUI (graphical user interface).

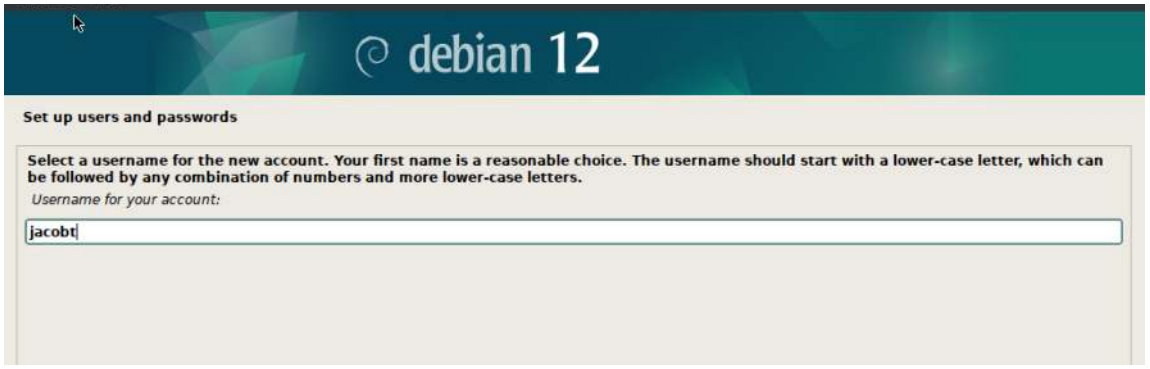
Note that if you choose any other option, the interface might differ from the screenshots in this guide. However, the content should remain the same.

- Choose the language, the location, the locales and the keyboard.
- Type an **hostname**. For this guide, we'll set "server-jacobt".

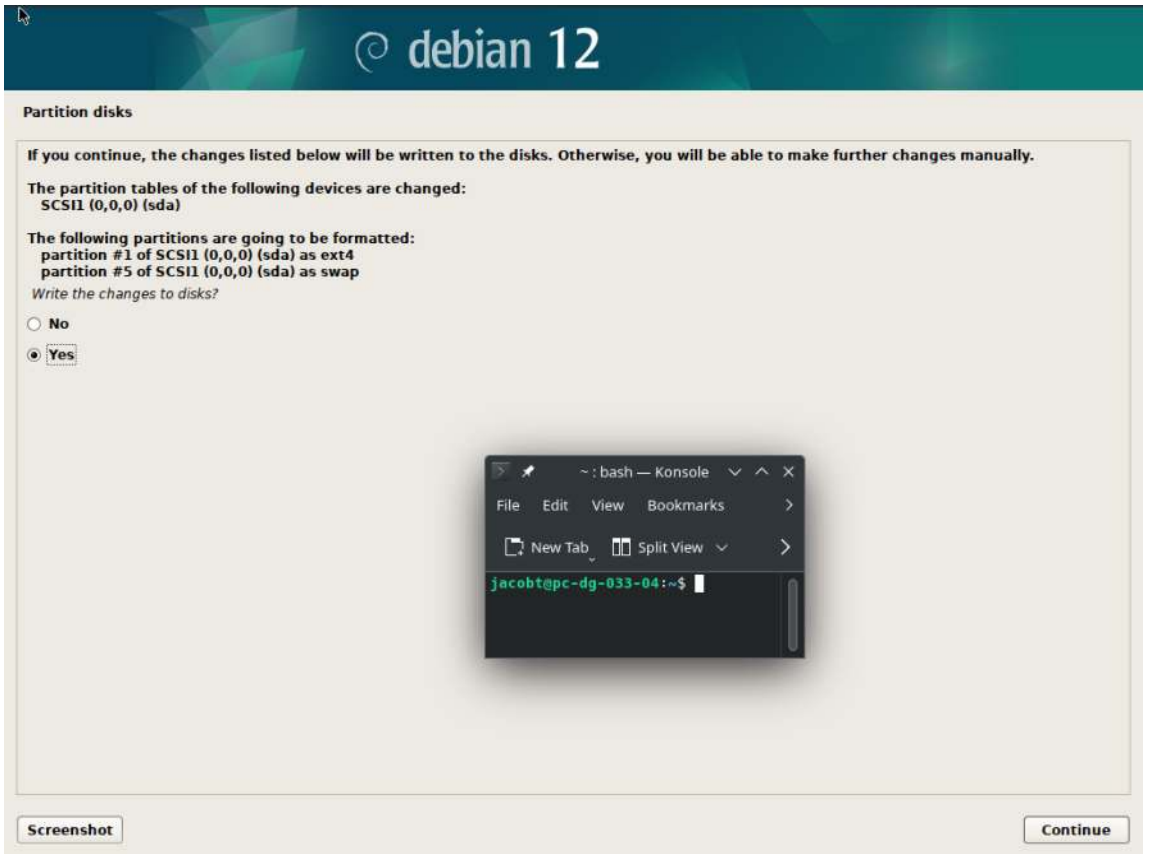


- Set a **root password**. This password is for the root user account, which is a **super-user**. It has all the permissions. We will use this user to install the softwares, and modify the configurations.

- Set your **username**, your full name and your password. In my case, I'm using "jacobt". This is the username you'll see in this tutorial.

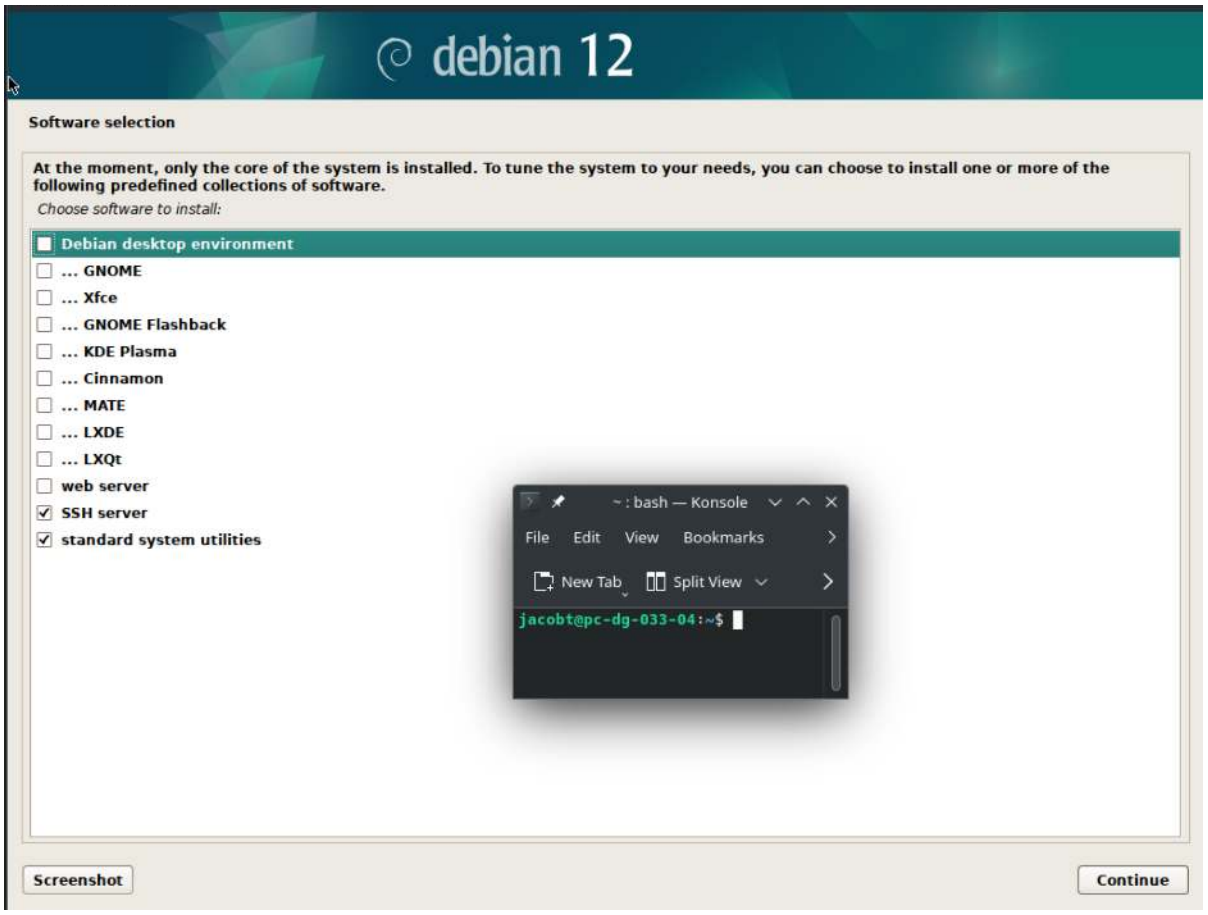


- Next, you'll have to set the **disk partitioning**. Adjust the settings to your needs and to your experience with Linux. It is recommended to use a **single partition**.



- Select the `deb.debian.org` **package manager**. It's the official Debian package manager. It works really well, and is enough for what we need. You can change these settings later if you want.

- **Really important step** : select carefully the **softwares**. You have to **disable** the Debian desktop environment, and enable **only** the SSH server, and the standard system utilities. This allows us to only use a CLI (Command Line Interface), and not a GUI (Graphical User Interface).



- Install **GRUB**, and, on the next page, set the device for boot loader with **`/dev/sda`**

When all these steps are done, the Debian installation should start. It might take some time, but when it's done, it should reboot.

First boot

If everything's done right, your system should have booted with **no interface**, just some text, waiting for you to write something.

So the **first thing to do** is to **log in** with the username and the password you specified in the install. Now, you should be able to enter some commands. We will use these commands to navigate through the system, and manipulate what we need.

Check the internet connectivity

Now that you're logged in, we want to **check** if we **can reach external networks** and ensure our VM is connected to the internet.

First, type `ip addr` to check your **IP address**. Here's what you should see :

```
jacobt@server-jacobt:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc 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: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc 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 86100sec preferred_lft 86100sec
    inet6 fec0::5054:ff:fe12:3456/64 scope site dynamic mngtmpaddr
       valid_lft 86106sec preferred_lft 14106sec
    inet6 fe80::5054:ff:fe12:3456/64 scope link
       valid_lft forever preferred_lft forever
jacobt@server-jacobt:~$ _
```

Then we want to make sure we can access an **external server**. To do this, type `traceroute 8.8.8.8`
This command will basically send a packet to the IP specified (in our case, the Google DNS), and show you each "checkpoints" it passes.

You should see something like this :

```
jacobt@server-jacobt:~$ traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  10.0.2.2 (10.0.2.2)  3.252 ms  0.889 ms  0.085 ms
 2  _gateway (192.168.141.19)  4.861 ms  4.934 ms  4.816 ms
 3  rt-wan.iut2.upmf-grenoble.fr (193.55.51.1)  1.251 ms  1.445 ms  1.936 ms
 4  r-viallet1.grenet.fr (193.54.184.185)  0.516 ms  1.022 ms  0.975 ms
 5  tigre1.grenet.fr (193.54.185.17)  12.171 ms  12.133 ms  12.026 ms
 6  hu0-2-0-2-ren-nr-grenoble-rtr-091.noc.renater.fr (193.51.181.94)  1.290 ms  2.228 ms  1.771 ms
 7  hu0-3-0-3-ren-nr-lyon2-rtr-091.noc.renater.fr (193.55.204.14)  10.605 ms  10.159 ms  6.746 ms
 8  et-3-1-0-ren-nr-marseille2-rtr-131.noc.renater.fr (193.51.177.254)  8.241 ms  8.199 ms  8.161 ms
 9  72.14.218.132 (72.14.218.132)  6.468 ms  6.427 ms  6.389 ms
10  192.178.105.171 (192.178.105.171)  6.420 ms 192.178.105.27 (192.178.105.27)  8.010 ms 192.178.105.91 (192.178.105.91)  6.699
 ms
11  142.251.78.89 (142.251.78.89)  6.596 ms 209.85.243.243 (209.85.243.243)  7.667 ms 72.14.232.43 (72.14.232.43)  7.627 ms
12  dns.google (8.8.8.8)  6.424 ms  6.988 ms  6.886 ms
jacobt@server-jacobt:~$
```

If the last line is `dns.google (8.8.8.8)`, it means that your machine is connected. If you see nothing, it means that there's a connection problem. Check if your ethernet cable or Wi-Fi interface is connected.

Check if Xorg is installed

Next, we want to check if **Xorg** is installed. Xorg is an open-source software used to manage graphical elements. It's essential for GUIs. Since we don't have a GUI, and we don't want any waste of performance, Xorg should **not** be **installed**.

The prompt to check that is `dpkg -l | grep xorg`
This command will list any installed packages related to Xorg. If no results are returned, it confirms that there is no graphical interface installed, which is suitable for a LAMP server setup.

If you see something, you might have skipped [this step.](#) You need to reinstall the server.

Check if the SSH server is working properly

Because we want to access the VM or the server from another machine, we need to verify if the **SSH server** is alive, and working.

- First thing to do is to **log in as root**. To do this, type `su -` and enter the root password you set during the installation.
- Type `systemctl status ssh` to ensure the SSH software is up.
  `systemctl` is a command we'll use to control the status of any process.
  You should have something like this :

```
root@server-jacobt:~# 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 10:07:52 CEST; 34min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
   Main PID: 473 (sshd)
      Tasks: 1 (limit: 4645)
     Memory: 8.3M
        CPU: 185ms
     CGroup: /system.slice/ssh.service
             └─473 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

May 03 10:07:51 server-jacobt systemd[1]: Starting ssh.service - OpenBSD Secure Shell server...
May 03 10:07:52 server-jacobt sshd[473]: Server listening on 0.0.0.0 port 22.
May 03 10:07:52 server-jacobt sshd[473]: Server listening on :: port 22.
May 03 10:07:52 server-jacobt systemd[1]: Started ssh.service - OpenBSD Secure Shell server.
May 03 10:09:44 server-jacobt sshd[510]: Accepted password for jacobt from 10.0.2.2 port 59220 ssh2
May 03 10:09:44 server-jacobt sshd[510]: pam_unix(sshd:session): session opened for user jacobt(uid=1000) by (uid=0)
May 03 10:09:44 server-jacobt sshd[510]: pam_env(sshd:session): deprecated reading of user environment enabled
root@server-jacobt:~#
```

  - If it's **not** running, type `systemctl start ssh`
  - If the SSH process is not found, type `apt install openssh-server`, and then, start the process using `systemctl start ssh`

Now, we will **test** the SSH access. On your **host machine**, open a **terminal**.

If your install is on a **VM**, type `ssh username@localhost -p 2222`

If your install is on a **server**, and you are accessing it on another machine, **on the same network**, type `ssh username@localip`. If using **port forwarding**, add `-p`, followed by the port at the end of the command, like above.

Don't forget to replace `username` by your username, and `localip` by the local IP address of the server.

The `-p 2222` that you see on the command is because we configured a **port forwarding** when initializing the VM.

Usually, the default SSH port is **22**. But, because we're on a VM, and we don't want **conflicts** on our network, we need to set up a **port forwarding**.

By making this, each time your VM receives a packet for the port 2222, it is forwarded to port 22, which is the SSH port. This process is done **locally** so it won't cause any conflicts. It allows us to keep the softwares working, while maintaining the stability of the network.

Now that you're logged in SSH, from an external machine, we will **install** a software to ensure that it's properly working.

- **Login as root** (`su -`)
- Install the **sudo** software, which is an essential software to grant **admin permissions** to some users. `apt install sudo`
- Add your user to the **sudo group** `sudo usermod -aG sudo [username]`
- Reboot your system `reboot` and log through SSH again.
- **Do not** log as root, and type `sudo apt install micro`
  It should ask for your password, and install this text-editor software.

If everything worked, it means that your **SSH server** and the **sudo** software are **working**.

Install the softwares

Now that the **essentials** are running, and our server is **operational**, we can start installing the basics softwares to make a LAMP Debian Server.

Install Apache

Apache is one of the most used web server software program packages within the world. Developed and maintained with the aid of using the Apache Software Foundation, Apache is an **open-source**, cross-platform **server** designed to supply **web content** material reliably and securely. Its **robustness**, **flexibility**, and huge network guide make it a famous preference for web website hosting web sites and net packages.

It's also a **must-have** for a LAMP server. Let's install it.

- On your VM/Server, **login as root** (`su -`)
- **Install** Apache with `apt install apache2`
- **Start** the Apache service (`systemctl start apache2`)
- **Check** the status of the service (`systemctl status apache2`)
  You should see something like this :

```
root@server-jacobt:~# 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 10:19:59 CEST; 4min 5s ago
       Docs: https://httpd.apache.org/docs/2.4/
   Main PID: 1163 (apache2)
      Tasks: 55 (limit: 4645)
     Memory: 9.4M
        CPU: 61ms
     CGroup: /system.slice/apache2.service
             ├─1163 /usr/sbin/apache2 -k start
             ├─1166 /usr/sbin/apache2 -k start
             └─1167 /usr/sbin/apache2 -k start

May 03 10:19:59 server-jacobt systemd[1]: Starting apache2.service - The Apache HTTP Server...
May 03 10:19:59 server-jacobt apachectl[1153]: AH00558: apache2: Could not reliably determine the server's fully qualified domain >
May 03 10:19:59 server-jacobt systemd[1]: Started apache2.service - The Apache HTTP Server.
lines 1-16/16 (END)
```

- **Check** if the server is working. To do this, enter `telnet localhost 80`, then type `HEAD / HTTP/1.0`, and break the line **twice**.
  It should **close** the connection, and return `HTTP OK`. If it is not working, try rebooting your system.
- On the **host machine**, open a web browser and go to http://localhost:8080

You should see something like this :

You might know that the default **HTTP port** is **80**. And you also might have noticed that we used the **8080 port**.
It's the same principle. To not create any conflicts, we made a **port forwarding** there. You can see it here, in the VM Initialization command.

Install PostgreSQL

PostgreSQL is an alternative to SQLite or MySQL. These are **DBMS** (DataBase Management System). They allow you to create and administer databases. It's also an essential part of our LAMP Server.

⚠️ During this process, replace **only** what's between the curly brackets. Also, at the end of each command you'll type in SQL, don't forget the semicolon (**;**). It's necessary when it comes to the SQL language.

Installation

To install it, follow these steps :

- On your VM/Server, **login as root** (`su -`)
- **Install** PostgreSQL with `apt install postgresql`

- **Start** the PostgreSQL service (`systemctl start postgresql`)

- **Check** the status of the service (`systemctl status postgresql`)
  You should see something like this :

```
root@server-jacobt:~# systemctl status postgresql
● postgresql.service - PostgreSQL RDBMS
     Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; preset: enabled)
     Active: active (exited) since Fri 2024-05-03 10:35:49 CEST; 45s ago
   Main PID: 2904 (code=exited, status=0/SUCCESS)
        CPU: 2ms

May 03 10:35:49 server-jacobt systemd[1]: Starting postgresql.service - PostgreSQL RDBMS...
May 03 10:35:49 server-jacobt systemd[1]: Finished postgresql.service - PostgreSQL RDBMS.
root@server-jacobt:~#
```

Basic configuration

Now that the software is installed, let's try to **connect** to the database.

- **Log** in as the `postgres` user, type `su - postgres`
- **Connect** to the database with `psql`

You should be **connected** with the `postgres` user. If you can't connect, try **rebooting** your system. But now, we need to create a user account to securely manage the databases. Let's do this :

- **Create** a new **user**
  `CREATE USER {username} WITH PASSWORD '{password}';`

  The username you're setting needs to be the **same** as your Debian username.

- **Create** a new **database**
  `CREATE DATABASE {name} WITH OWNER {username};`
- **Disconnect** of the **database** (`\q`) and **log out** of root user (type `exit` twice)
- **Re-connect** to the **database** (`psql -d {database_name}`)
  On this command, the `-d` is an option to tell psql on which database you'd like to be connected. In this case, carefully input the name of the database you created.

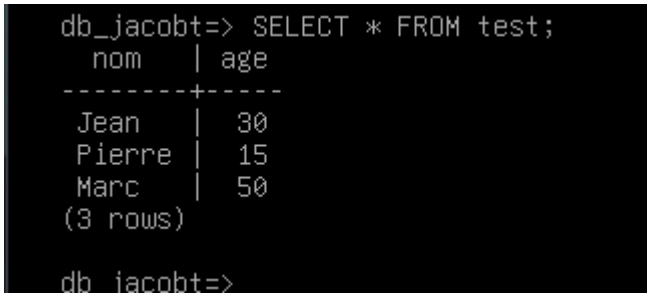
At this point, you should be successfully **connected** to your database.

Then, let's **create** your **first table** in your database. We will use SQL commands on this part. Here's a quick cheat sheet of the SQL basics commands to help you understand.

- **Create** a table. For the example, we'll create this one :
```
CREATE TABLE test(
      name VARCHAR PRIMARY KEY,
      age NUMERIC
);
```

- **Insert** a few lines in your table :
```
INSERT INTO test VALUES ('Jean', 30);
INSERT INTO test VALUES ('Pierre', 15);
INSERT INTO test VALUES ('Marc', 50);
```

- **Select** everything in your table : everything we inserted should appear.
```
SELECT * FROM test;
```



Open an external access

For the next step, we will make an access as you can **connect** to the database from the **host machine**. This involves editing the PostgreSQL configuration files to allow remote connections and setting up the appropriate rules.

- If not done yet, **disconnect** from the **database** (`\q`)

- **Login** as **root** (`su -`)

- **Navigate** to the configuration files. Type `cd /etc/postgresql/15/main`

  Be careful there, the "**15**" you can see in the command is related to the **PostgreSQL version**. If the command doesn't work, it's probably that you have a different version. You can check your PostgreSQL version by following [this guide](#).

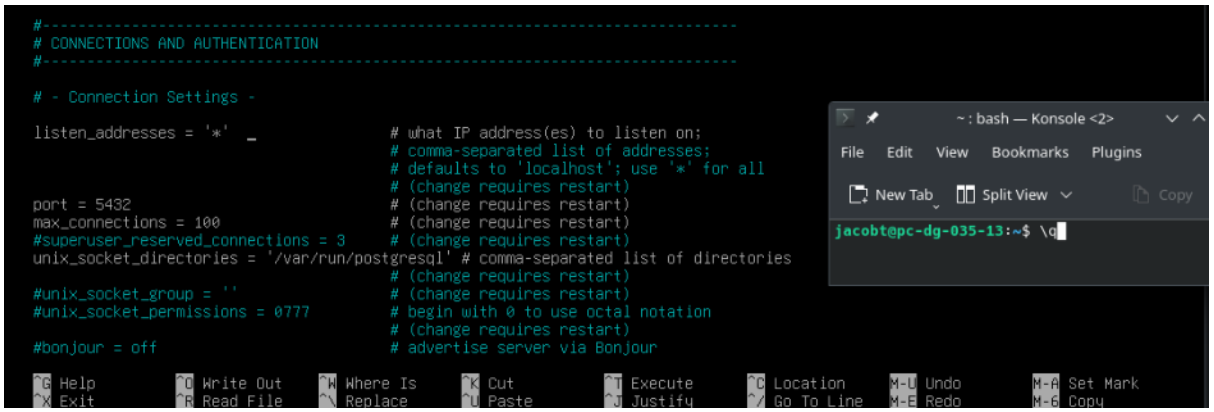
- Type `ls`, you should see a file named `postgresql.conf`

- **Edit** the file (`nano postgresql.conf`)

- **Scroll down** using the arrow keys until you see `CONNECTIONS AND AUTHENTICATION`, and remove the **#** on the **`listen_addresses`** line.

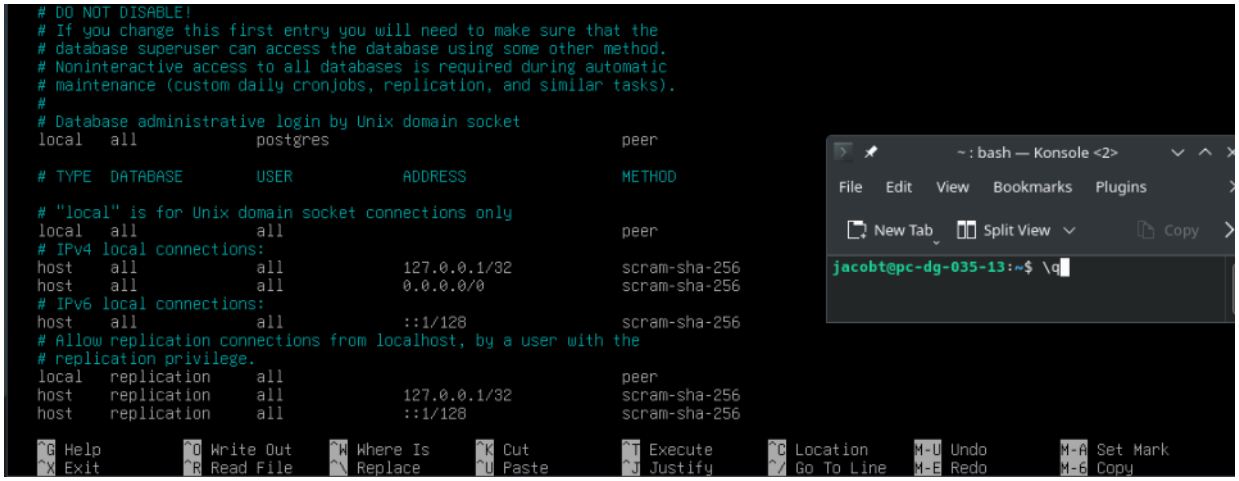
  Any # present at the start of a line means that it should be ignored. The system won't treat it. So removing it will make the line active, and enable the configuration setting.

- On the **`listen_adresses`** line, **change** the "**`localhost`**" by "**`*`**"
  Now, you should have something looking like this :



- **Press** Ctrl+S to **save**, and Ctrl+X to **quit**.
- Once back on the shell, **edit** the `pg_hba.conf` file (`nano pg_hba.conf`)
- **Scroll** all the way **down**, and under the `#ipv4 local connections` line, add the following line :
  `host  all  all  0.0.0.0/0  scram-sha-256`

  You should have that :



- **Press** Ctrl+S to **save**, and Ctrl+X to **quit**.
- **Restart** the service (`systemctl restart postgresql`)

Now, we made an access. Let's hop back on the **host machine** to test this out.

- Get back on the **host machine**.
- On a terminal, **type** `psql -h localhost -U {username} {database_name}`.

  In this command, the **-h** option specifies the host to connect to (in this case, `localhost`), the **-U** option indicates the username to use for the connection, and the final argument is the name of the database you want to connect to.

- **Enter** your **password**. You should be connected.
- **Select** everything in the table we created before, using the same SQL command `SELECT * FROM test;`

```
jacobt@pc-dg-035-13:~$ psql -h localhost -U jacobt db_jacobt
Password for user jacobt:
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.

db_jacobt=> SELECT * FROM test;
  nom   | age
--------+-----
 Jean   |  30
 Pierre |  15
 Marc   |  50
(3 rows)

db_jacobt=>
(jacobt) localhost:2222 ×   jacobt : psql ×
```

Last checks

Now that everything is functional, let's do quick **checks** on your databases.

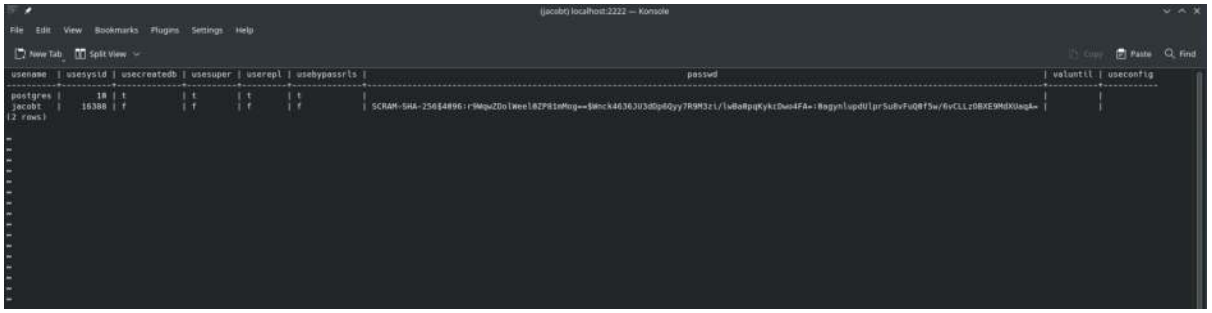
- If not, **connect to your database**
- **Type** `\d` to have a view of all your databases

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

db_jacobt=>
(jacobt) localhost:2222 ×   jacobt : psql ×
```

Finally, **check** if the passwords are encrypted with SHA-256 protocol :

- On your **server/VM**, on the **shell**, login first as **root** (`su -`), and then as **postgres** (`su - postgres`)
- **Connect** to the database with `psql`
- **Type** `SELECT * FROM pg_shadow;`
- **Check** if the `passwd` column contains entries that start with `SCRAM-SHA-256`



We made it! **PostgreSQL** is **installed** and fully **functional**. You can administer your databases on your host machine, and manage it without issues.

Install PHP

Now that PostgreSQL is installed, let's install **PHP** :

Back-end web development requires the **server-side scripting** language **PHP**. It makes it possible to create **dynamic** and **interactive** web pages. This robust language can handle tasks like creating dynamic page **content**, controlling **sessions**, and interacting with **databases**, which is exactly why it's an important part of a LAMP system.

To proceed, follow these instructions :

- On your VM/Server, **login as root** (`su -`)
- **Install** PHP with `apt install php-common libapache2-mod-php php-cli`
- **Restart** the Apache service (`systemctl restart apache2`)

PHP is now **installed**.

Then, let's **enable** a basic configuration to see if everything is properly working.

- While logged as **root**, type `cd /var/www`
- **Edit** the `info.php` file (`nano info.php`)
- In the **nano** editor, add this to the file :
```
<?php
phpinfo();
phpinfo(INFO_MODULES);
?>
```
- **Press** Ctrl+S to **save**, and Ctrl+X to **quit**

Now that you've done this, on your **host machine**, open a web browser and access http://localhost:8080/info.php
You should see a **web page** with **information** about **PHP**.

Note that the URL could change depending on the IP Address and the web port of your server. You might also need to restart the Apache service, or reboot the whole server to make it work.

Install PhpPgAdmin

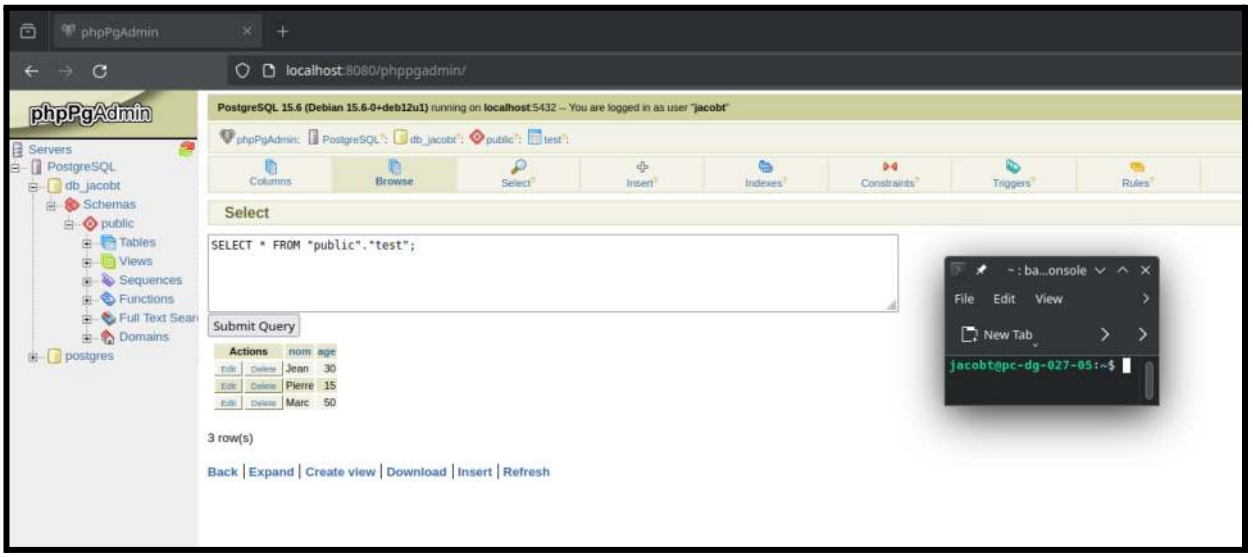
Now that **PHP** is installed, we will install a **module** that will allow us to manage the database we created on a web page.

It will make the **administration easier** for everyone.

- On your VM/Server, **login as root** (`su -`)
- **Install** PhpPgAdmin with `apt install phppgadmin`
- **Enter** `nano /usr/share/phppgadmin/classes/databases/Connection.php` (pay attention at the capital letter on **C**onnection.php)
- In the **nano** editor, find the line that contains `case '14'` and change it with `case '15'`
  ⚠️ It's the **version** of **PostgreSQL**. So it might be **different** than 15. Consider that.
- **Press** Ctrl+S to **save**, and Ctrl+X to **quit**
- **Enter** `nano /etc/apache2/conf-available/phppgadmin.conf`
- In the **nano** editor, find the line that contains `require local` and change it with `require all granted`
- **Press** Ctrl+S to **save**, and Ctrl+X to **quit**

Now, everything should be **installed** and **configured**. We should be able to manage our database from the host machine. Let's test that :

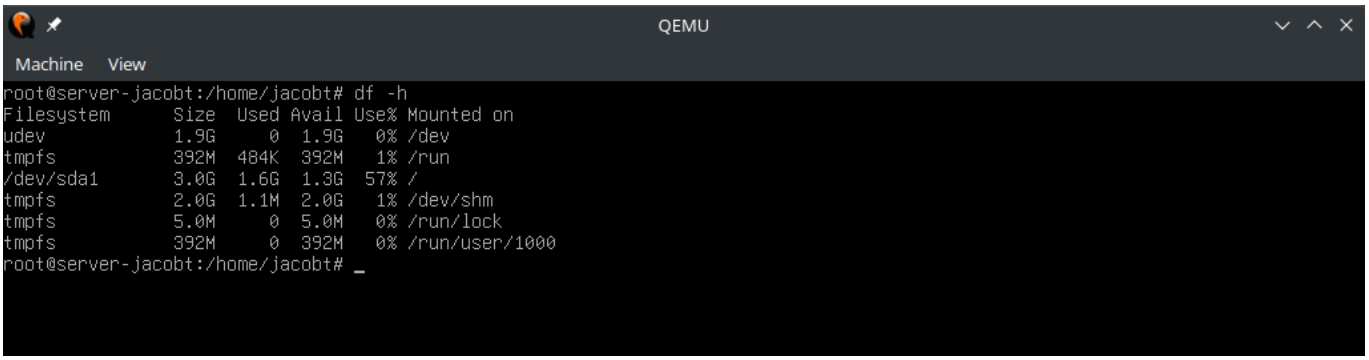
- On the **host machine**, open a web browser and access http://locahost:8080/phppgadmin
- You should see an **administration** page of the **databases** of your server
- On the **left menu**, select your database, and **log in** with your credentials
- **Navigate** until you find your table, and interrogate it with an SQL prompt. You should see something like this :



Everything is now correctly set up ! You can now **use** your LAMP server and extend its functionalities by installing other softwares.

Quick final command : to see your available storage after all these manipulations, on your VM, enter `df -h`

You should get this interface :



On my machine, I have approximately 6G available.

Quick security analysis and tips

To ensure your system stays **secure**, you should regularly run the **updates**. To do so, log in as **root**, and type these three commands :

- `apt update` – to update the databases
- `apt upgrade` – to apply the new updates
- `apt clean` – to clean the install packages

Also, you can **install** and **enable** a **firewall** to block unwanted connections :

- `apt install ufw` – to install the firewall (log as root before)
- `ufw allow ssh` – to authorize the connections to these protocols
- `ufw allow http`
- `ufw allow https`
- `ufw enable` – to enable the firewall

You can disable the **root login** on the **SSH** :

- As **root** on the VM, type `nano /etc/ssh/sshd_config`
- In the **nano** editor, set the `PermitRootLogin` line to `False`

To run **admin commands** while logged in with **SSH**, just use **sudo**, the software we installed at the beginning of this guide. Make sure your user is in the sudo group.

And finally, you can install a **package** that automatically **runs security updates** :

- `apt install unattended-upgrades`
- `dpkg-reconfigure --priority=low unattended-upgrades`

Annex screenshots

`cat /etc/fstab` command :

```
jacobt@server-jacobt:~$ 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=449c06dc-0c19-4b69-90a6-4aa95c87e1c8 /               ext4    errors=remount-ro 0       1
# swap was on /dev/sda5 during installation
UUID=d3ae5753-80fb-49f3-b7f5-8d0191043499 none            swap    sw              0       0
/dev/sr0        /media/cdrom0   udf,iso9660 user,noauto     0       0
jacobt@server-jacobt:~$ _
```

Final website screenshot :

```
Bonjour

Je suis www-data

Qui est connecté ?

jacobt   tty1         May 27 10:11

Mes disques sont

/dev/sda5: UUID="d3ae5753-80fb-49f3-b7f5-8d0191043499" TYPE="swap" PARTUUID="d513a6f3-05"
/dev/sda1: UUID="449c06dc-0c19-4b69-90a6-4aa95c87e1c8" BLOCK_SIZE="4096" TYPE="ext4" PARTUUID="d513a6f3-01"

Mes interfaces

1: lo:  mtu 65536 qdisc 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 qdisc 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 83145sec preferred_lft 83145sec
    inet6 fec0::5054:ff:fe12:3456/64 scope site dynamic mngtmpaddr
       valid_lft 86212sec preferred_lft 14212sec
    inet6 fe80::5054:ff:fe12:3456/64 scope link
       valid_lft forever preferred_lft forever

My apache install is

ii  apache2                  2.4.57-2                    amd64        Apache HTTP Server
ii  apache2-bin              2.4.57-2                    amd64        Apache HTTP Server (modules and other binary files)
ii  apache2-data             2.4.57-2                    all          Apache HTTP Server (common files)
ii  apache2-utils            2.4.57-2                    amd64        Apache HTTP Server (utility programs for web servers)
ii  libapache2-mod-php       2:8.2+93                    all          server-side, HTML-embedded scripting language (Apache 2 module) (default)
ii  libapache2-mod-php8.2    8.2.7-1~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 Mon 2024-05-27 11:04:16 CEST; 15s ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 2843 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 2848 (apache2)
      Tasks: 8 (limit: 4645)
     Memory: 15.1M
        CPU: 103ms
     CGroup: /system.slice/apache2.service
             |-2848 /usr/sbin/apache2 -k start
             |-2849 /usr/sbin/apache2 -k start
             |-2850 /usr/sbin/apache2 -k start
             |-2851 /usr/sbin/apache2 -k start
             |-2852 /usr/sbin/apache2 -k start
             |-2853 /usr/sbin/apache2 -k start
             |-2866 sh -c "systemctl status apache2"
             `-2867 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 RDBMS
     Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; preset: enabled)
     Active: active (exited) since Mon 2024-05-27 10:10:18 CEST; 54min ago
    Process: 589 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
   Main PID: 589 (code=exited, status=0/SUCCESS)
        CPU: 1ms

My ssh install is

ii  libssh2-1:amd64              1.10.0-3+b1                 amd64        SSH2 client-side library
ii  openssh-client               1:9.2p1-2+deb12u2           amd64        secure shell (SSH) client, for secure access to remote machines
ii  openssh-server               1:9.2p1-2+deb12u2           amd64        secure shell (SSH) server, for secure access from remote machines
ii  openssh-sftp-server          1:9.2p1-2+deb12u2           amd64        secure shell (SSH) sftp server module, for SFTP access from remote machines
ii  task-ssh-server              3.73                        all          SSH server

My ssh status is

* ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; enabled; preset: enabled)
     Active: active (running) since Mon 2024-05-27 10:10:15 CEST; 54min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
    Process: 458 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Main PID: 507 (sshd)
      Tasks: 1 (limit: 4645)
     Memory: 6.4M
        CPU: 377ms
     CGroup: /system.slice/ssh.service
             `-507 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"
```