
Debian 13 Server

Installation Guide

Apache · PostgreSQL · PHP · PhpPgAdmin

Ugurlu Furkan

UGA – IUT2 – Departement Informatique

BUT-1A – S2.03 – 2025/2026

Contents

Introduction	3
1 Preparing the ISO Image	4
1.1 Finding the ISO	4
1.2 Verifying the checksum	4
2 Installing Debian 13	4
2.1 Starting the installer	4
2.2 Language and locale	4
2.3 Hostname	4
2.4 User accounts	5
2.5 Disk partitioning	5
2.6 Software selection	6
2.7 GRUB	6
2.8 Shutting down cleanly	6
3 Moving the Disk Image to Erebus	7
3.1 Port forwarding	7
4 Connecting via SSH	7
4.1 How the forwarding works	7
4.2 Opening a session	7
4.3 Switching to root	8
4.4 Checking the SSH service	8
4.5 Filesystem layout	8
4.6 Checking that no desktop was installed	9
5 Installing Apache	10
5.1 Installation	10
5.2 Checking the service	10
5.3 Quick test without a browser	10
5.4 Default page in Firefox	11
6 Installing PostgreSQL	11
6.1 Installation	11
6.2 Checking the service	11
6.3 Listing the default databases	11
6.4 Creating a user and a database	12
6.5 Creating a table and inserting data	12
6.6 Allowing remote connections	13
6.6.1 postgresql.conf – making PostgreSQL listen on the network	13
6.6.2 pg_hba.conf – defining who can connect	13
6.6.3 Restarting to apply the changes	14
6.7 Testing from the workstation	14
6.8 Checking that passwords are properly hashed	14

7	Installing PHP	15
7.1	Installation	15
7.2	Checking it works	15
7.3	Deploying the SAE PHP page	15
8	Installing PhpPgAdmin	16
8.1	Installation	16
8.2	Allowing access from the workstation	16
8.3	Logging in	16
9	Disk Usage	17
10	SSH Key Authentication	18
10.1	Generating a key pair	18
10.2	Copying the key to the VM	18
10.3	Testing it	18
11	Root SSH Access with a Key	18
11.1	Copying the key into the root account	18
11.2	Updating sshd_config	18
11.3	Testing	19
12	HTTPS with a Self-Signed Certificate	19
12.1	Enabling SSL	19
12.2	Testing in Firefox	19
13	Security	20
	Conclusion	21

Introduction

This document describes how I set up a Debian 13 “Trixie” server inside a Qemu/KVM virtual machine during the S2.03 practical sessions, and how I configured Apache, PostgreSQL, PHP and PhpPgAdmin on it.

The goal is that someone else could follow these steps and end up with the same working environment. All services are accessible from the host machine via port forwarding: SSH on port 2222, HTTP on port 8080 and PostgreSQL on port 5432.

Wherever you see `yourlogin`, replace it with your own UGA login.

Warning. Everything gets installed through APT. Compiling from source is not recommended here – it makes updates harder to manage.

Prerequisites

Before starting, make sure you have the following:

- A Linux workstation with Qemu/KVM and the S2.03 scripts (`S2.03-lance-installation`, `S2.03-lance-machine-virtuelle`, `S2.03-deplace-image-disque-sur-erebus`)
- Your UGA login, which will be used as the system username and in the host-name
- Some familiarity with the Linux command line (module R1.04)
- Around 5 GB of free disk space on the workstation
- Access to the IUT2 network so Erebus is reachable

PART 1 DEBIAN 13 INSTALLATION

1. Preparing the ISO Image

1.1 Finding the ISO

The Debian 13 netinst image should already be in the shared folder. You can check with:

```
$ ls /usr/local/images-ISO/  
  debian-13.4.0-amd64-netinst.iso
```

1.2 Verifying the checksum

Before doing anything with the image, it is worth checking that it downloaded correctly. Run `sha512sum` and compare the result with the hash listed on <https://cdimage.debian.org>. They need to be identical.

```
$ sha512sum debian-13.4.0-amd64-netinst.iso
```

Warning. If the hashes do not match, do not use the image. Download it again.

2. Installing Debian 13

2.1 Starting the installer

The following script boots the VM from the ISO:

```
$ S2.03-lance-installation
```

Note. It is worth opening `S2.03-commun` to see the exact Qemu/KVM parameters being passed: RAM size, number of CPUs, the disk image path, and how port forwarding is set up.

2.2 Language and locale

In the installer, select these options one by one:

```
Language -> English  
Location -> Other > Europe > France  
Locales  -> United States, en_US.UTF-8  
Keyboard -> French
```

2.3 Hostname

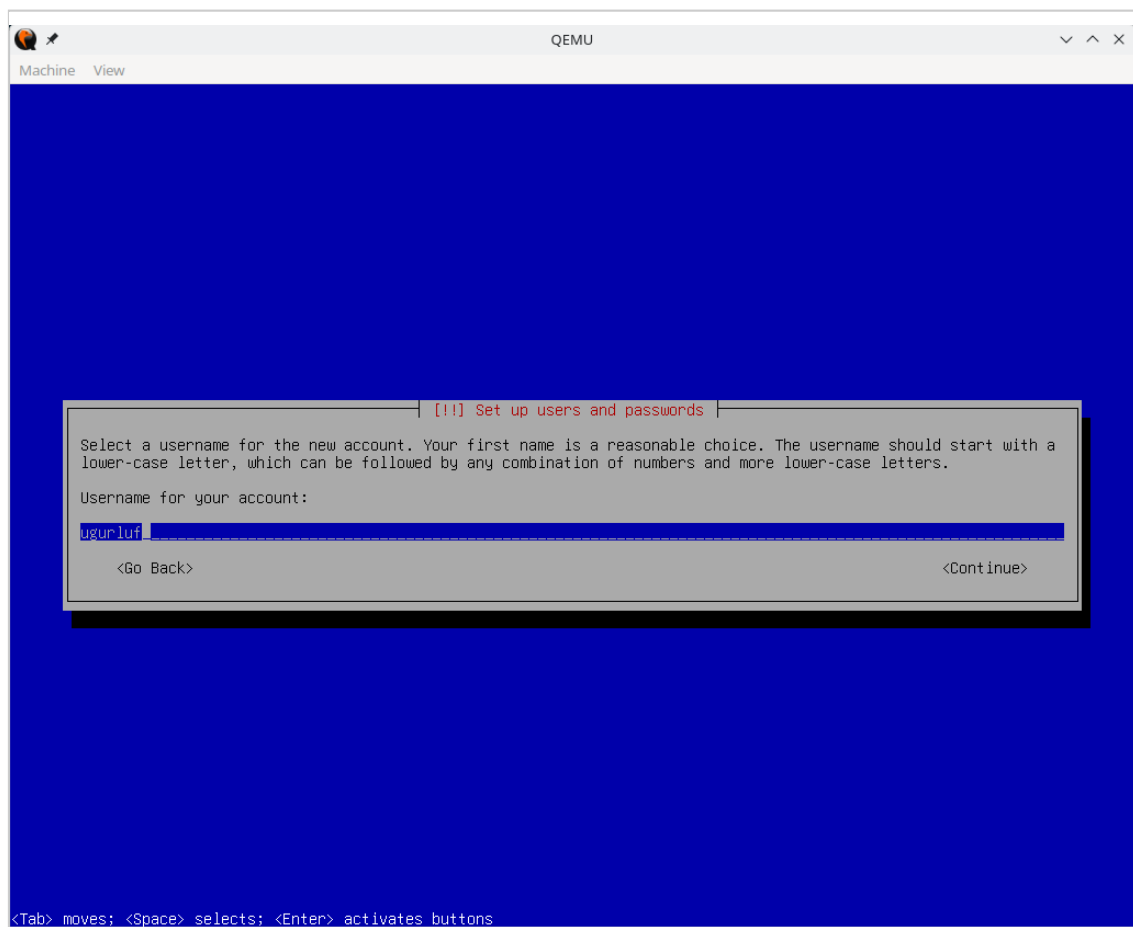
I used `server-yourlogin` as the hostname, following the course convention.

```
Hostname -> server-yourlogin
```

2.4 User accounts

The installer asks you to create a root account and a normal user. Use your UGA login as the username. I recommend ticking **Show Password** every time you type one, since a mistake here means you get locked out and have to reinstall.

```
Root password -> (choose something you will remember)
Full name      -> Your Name
Username       -> yourlogin
User password  -> (choose something)
```



2.5 Disk partitioning

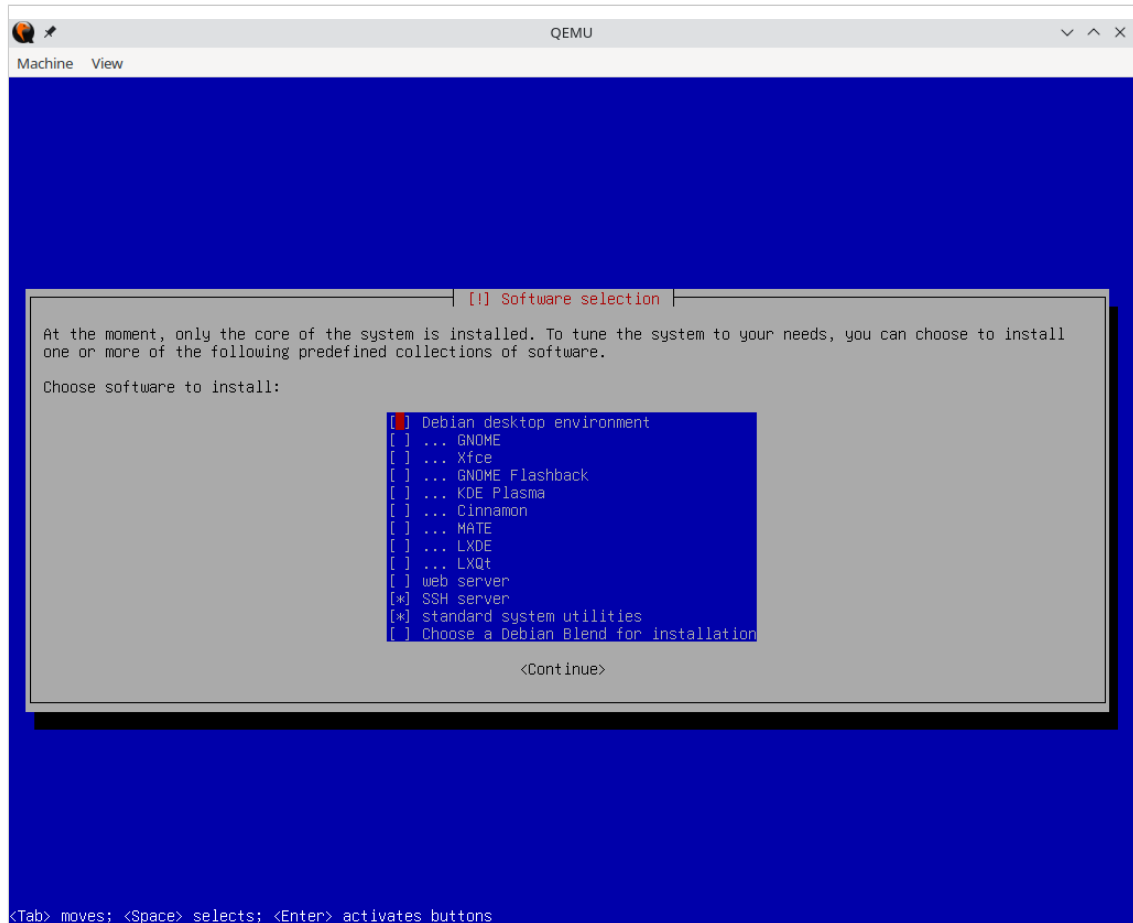
For this kind of setup, one partition is plenty:

```
Method -> Guided -- use entire disk
Scheme -> All files in one partition
Confirm -> Yes
```

2.6 Software selection

This is probably the most important step of the installation. If you accidentally install a desktop environment, the VM will waste a lot of resources on graphics it never uses. Make sure it looks like this:

```
[ ] Debian desktop environment    <- leave unchecked
[*] SSH server                   <- this one must be checked
[*] Standard system utilities    <- leave checked
```



Note. Servers do not need a graphical interface. Leaving it out reduces RAM usage, shrinks the number of packages that need security updates, and makes the machine boot faster.

2.7 GRUB

```
Install GRUB?  ->  Yes
Boot device    ->  /dev/sda
```

2.8 Shutting down cleanly

Once the machine reboots, log in as root and shut it down before moving the disk image:

```
# systemctl poweroff
```

Warning. Do not close the QEMU window with the X button. This can corrupt the disk image. Always use `systemctl poweroff`.

3. Moving the Disk Image to Erebus

The disk image is saved locally on the workstation. To be able to access the VM from any machine, move it to Erebus first.

Warning. Make sure the VM is fully shut down before doing this.

```
$ S2.03-deplace-image-disque-sur-erebus
```

Once that is done, start it from Erebus:

```
$ S2.03-lance-machine-virtuelle
```

3.1 Port forwarding

The launch script automatically sets up these port redirections between the VM and the workstation:

Service	VM port	Host port	How to use it
SSH	22	2222	<code>ssh yourlogin@localhost -p 2222</code>
HTTP	80	8080	<code>http://localhost:8080/</code>
HTTPS	443	4443	<code>https://localhost:4443/</code>
PostgreSQL	5432	5432	<code>psql -h localhost -U yourlogin</code>

4. Connecting via SSH

Since the VM has no desktop, everything is done through SSH from the workstation.

4.1 How the forwarding works

The launch script maps port 2222 on the workstation to port 22 inside the VM. So when you connect to `localhost:2222`, the traffic goes straight into the VM. You never need to know its actual IP address.

4.2 Opening a session

Open a terminal on the workstation and type:

```
$ ssh yourlogin@localhost -p 2222
```

The first time you do this, SSH will warn you that the host is unknown. That is expected – just type `yes`:

```
The authenticity of host '[localhost]:2222' can't be established.
ED25519 key fingerprint is SHA256:xxxx...
Are you sure you want to continue connecting (yes/no)? yes
```

```
yourlogin@server-yourlogin:~$
```

If you see that prompt, you are inside the VM.

Note. Root SSH login is disabled on Debian by default, which is the right call. Always connect as your regular user and then switch to root.

4.3 Switching to root

```
yourlogin@server-yourlogin:~$ su -
Password:
root@server-yourlogin:~#
```

The # prompt means you are root. Any command in this guide shown with # needs to be run as root.

4.4 Checking the SSH service

```
# systemctl status ssh
```

```
ugurluf@server-ugurluf:~$ systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/usr/lib/systemd/system/ssh.service; enabled; preset: enabled)
   Active: active (running) since Wed 2026-05-06 14:04:49 CEST; 3min 27s ago
  Invocation: 30eb314a72ee4b81874bdbcdb7df420
     Docs: man:sshd(8)
           man:sshd_config(5)
   Process: 681 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
    Main PID: 716 (sshd)
      Tasks: 1 (limit: 4639)
     Memory: 6.9M (peak: 24.7M)
        CPU: 78ms
    CGroup: /system.slice/ssh.service
            └─716 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

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

4.5 Filesystem layout

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

```
ugurluf@server-ugurluf:~$ 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=9335e3069-3840-4071-b408-afc68e92f73e / ext4 errors=remount-ro 0 1
# swap was on /dev/sda5 during installation
UUID=50668943-0f2a-47a0-859f-2b9f4577eae7 none swap sw 0 0
/dev/sr0 /media/cdrom0 udf,iso9660 user,noauto 0 0
ugurluf@server-ugurluf:~$
```

4.6 Checking that no desktop was installed

Both commands below should return nothing. If they do, X11 or Wayland libraries are present and should be removed.

```
# dpkg -l | grep libx11  
# dpkg -l | grep wayland
```

PART 2 APACHE – POSTGRESQL – PHP – PHPPGADMIN

5. Installing Apache

5.1 Installation

All of the following is done inside the VM via SSH.

```
# apt update
# apt install apache2
# systemctl enable apache2
# systemctl start apache2
```

5.2 Checking the service

```
# systemctl status apache2
```

```
ugurluf@server-ugurluf:~$ systemctl status apache2
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/apache2.service; enabled; preset: enabled)
   Active: active (running) since Wed 2026-05-06 14:04:50 CEST; 4min 9s ago
     Invocation: 29c8323379124265b45ca73e8bcc9345
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 676 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 735 (apache2)
      Tasks: 6 (limit: 4639)
     Memory: 23.7M (peak: 23.9M)
        CPU: 88ms
    CGroup: /system.slice/apache2.service
            └─735 /usr/sbin/apache2 -k start
              └─751 /usr/sbin/apache2 -k start
                └─752 /usr/sbin/apache2 -k start
                  └─753 /usr/sbin/apache2 -k start
                    └─754 /usr/sbin/apache2 -k start
                      └─755 /usr/sbin/apache2 -k start

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

5.3 Quick test without a browser

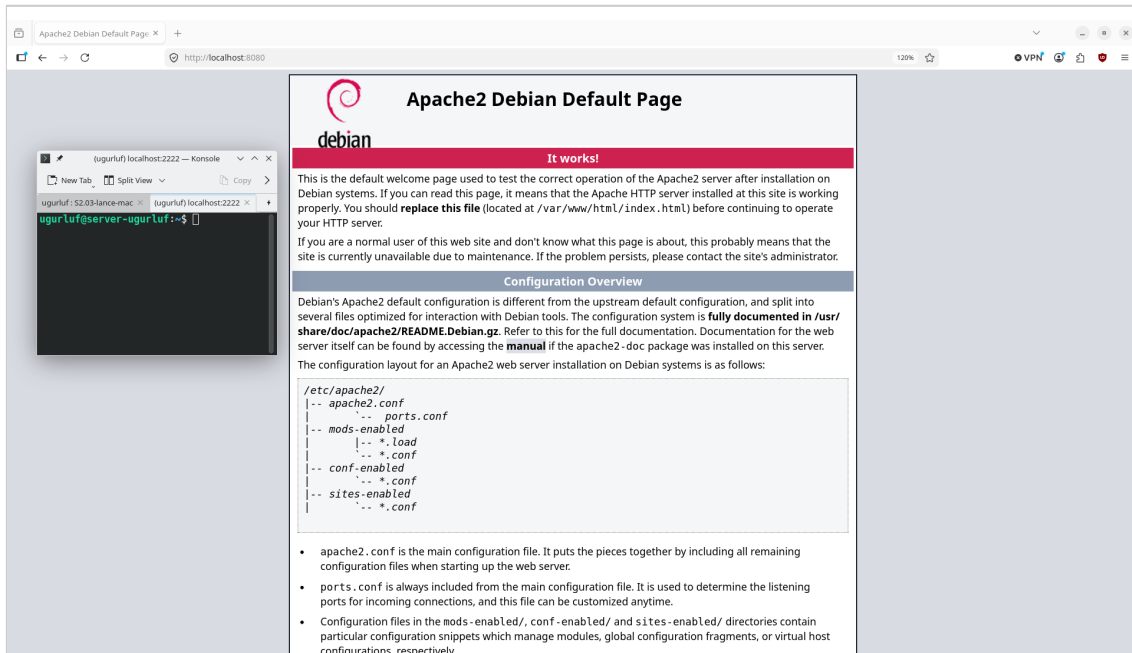
Since the VM has no browser installed, I use `netcat` to check that port 80 is open:

```
# from inside the VM:
$ netcat -zv localhost 80
localhost [127.0.0.1] 80 (http) open

# from the workstation in a separate terminal:
$ netcat -4zv localhost 8080
Connection to localhost 8080 port succeeded!
```

5.4 Default page in Firefox

Open `http://localhost:8080` on the workstation. The Apache default page appearing means everything is working correctly.



6. Installing PostgreSQL

6.1 Installation

```
# apt install postgresql
```

6.2 Checking the service

```
# systemctl status postgresql
```

```
ugurluf@server-ugurluf:~$ systemctl status postgresql
● postgresql.service - PostgreSQL RDBMS
   Loaded: loaded (/usr/lib/systemd/system/postgresql.service; enabled; preset: enabled)
   Active: active (exited) since Wed 2026-05-06 14:04:52 CEST; 4min 41s ago
  Invocation: 2be5a903d39c4f74b7e0ca08284f18e4
    Process: 803 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
   Main PID: 803 (code=exited, status=0/SUCCESS)
    Mem peak: 1.8M
       CPU: 4ms

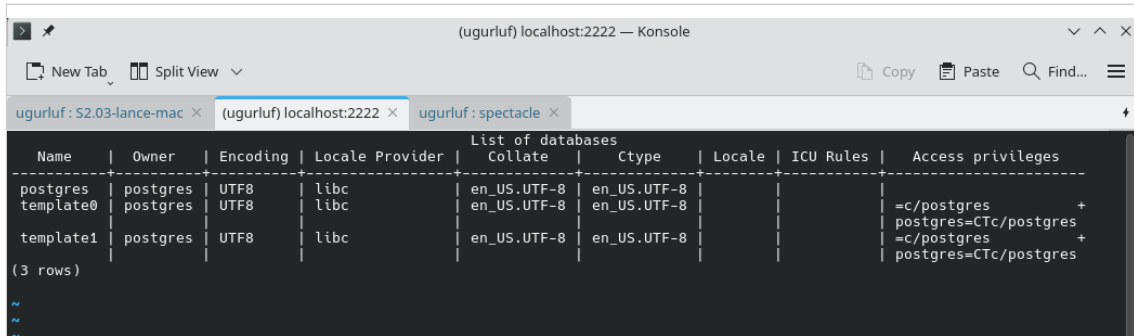
Warning: some journal files were not opened due to insufficient permissions.
ugurluf@server-ugurluf:~$
```

6.3 Listing the default databases

PostgreSQL creates a dedicated system account called `postgres` during installation. You switch to it to do local database administration.

```
# su - postgres
$ psql -l
```

Three databases appear by default: `postgres`, `template0` and `template1`.



The screenshot shows a terminal window titled "(ugurluf) localhost:2222 — Konsole". It displays the output of the `\l` command in PostgreSQL, which lists the databases. The output is as follows:

Name	Owner	Encoding	Locale Provider	Collate	Ctype	Locale	ICU Rules	Access privileges
postgres	postgres	UTF8	libc	en_US.UTF-8	en_US.UTF-8			
template0	postgres	UTF8	libc	en_US.UTF-8	en_US.UTF-8			=c/postgres +
template1	postgres	UTF8	libc	en_US.UTF-8	en_US.UTF-8			postgres=CTc/postgres +

(3 rows)

Type `\q` to quit `psql`, then `exit` to go back to root.

6.4 Creating a user and a database

Still as the `postgres` Unix account:

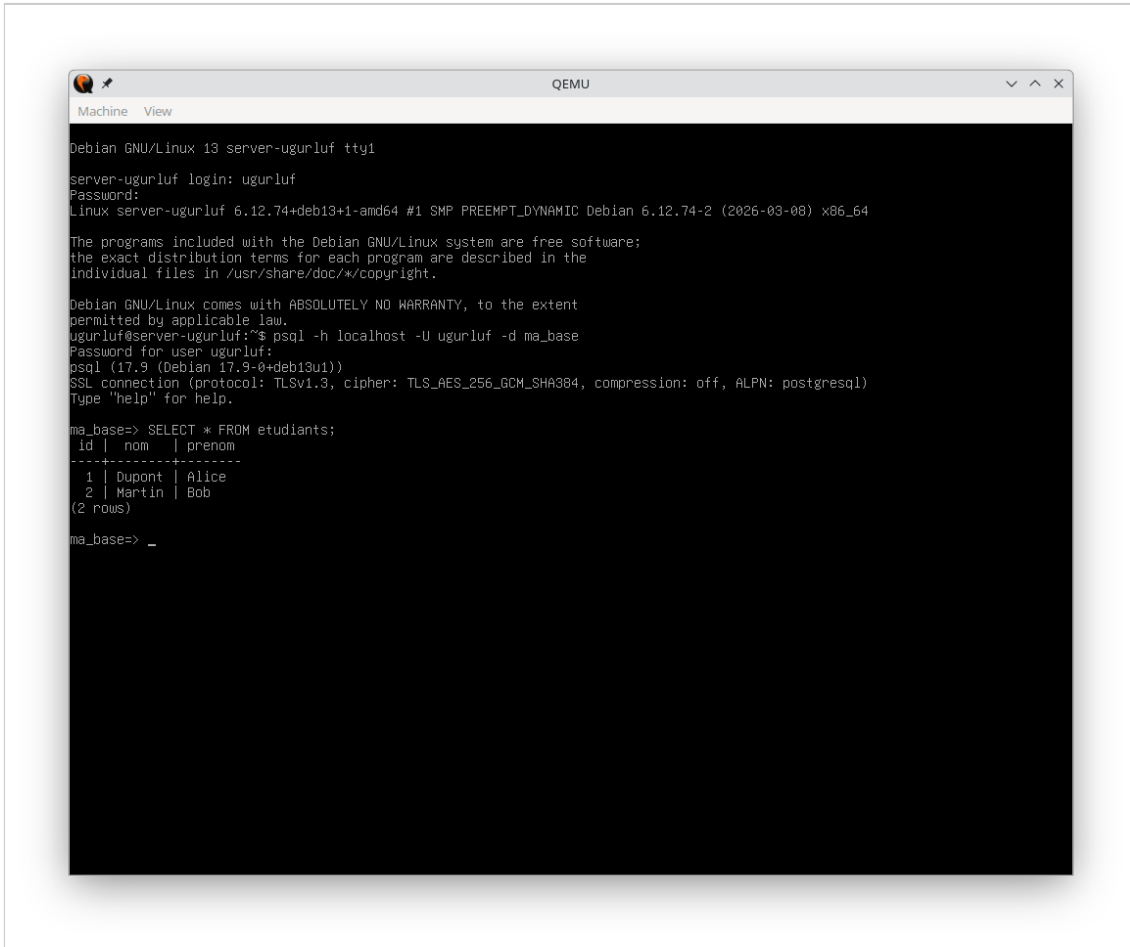
```
$ psql

postgres=# CREATE USER yourlogin WITH PASSWORD 'yourpassword';
postgres=# CREATE DATABASE ma_base OWNER yourlogin;
postgres=# \q
```

6.5 Creating a table and inserting data

```
# psql -h localhost -U yourlogin -d ma_base

ma_base=> CREATE TABLE etudiants (
    id      SERIAL PRIMARY KEY,
    nom     VARCHAR,
    prenom  VARCHAR
);
ma_base=> INSERT INTO etudiants (nom, prenom) VALUES
    ('Dupont', 'Alice'), ('Martin', 'Bob');
ma_base=> SELECT * FROM etudiants;
ma_base=> \q
```



6.6 Allowing remote connections

Out of the box, PostgreSQL only accepts local connections. Getting it to accept connections from the workstation requires editing two configuration files.

6.6.1 postgresql.conf – making PostgreSQL listen on the network

Switch to the `postgres` account and open the config file:

```
# su - postgres
$ nano /etc/postgresql/17/main/postgresql.conf
```

Use **Ctrl-W** to search for `listen_addresses`, then uncomment the line and change its value to `'*'`:

```
listen_addresses = '*'
```

Save with **Ctrl-O** and close with **Ctrl-X**.

6.6.2 pg_hba.conf – defining who can connect

```
$ nano /etc/postgresql/17/main/pg_hba.conf
```

Add this at the end of the file. It allows connections from any IP as long as the password is verified using SCRAM-SHA-256:

```
# IPv4 remote connections:
host all all 0.0.0.0/0 scram-sha-256
```

Note. The `postgres` superuser has no password set, so this rule does not apply to it – it can only connect locally.

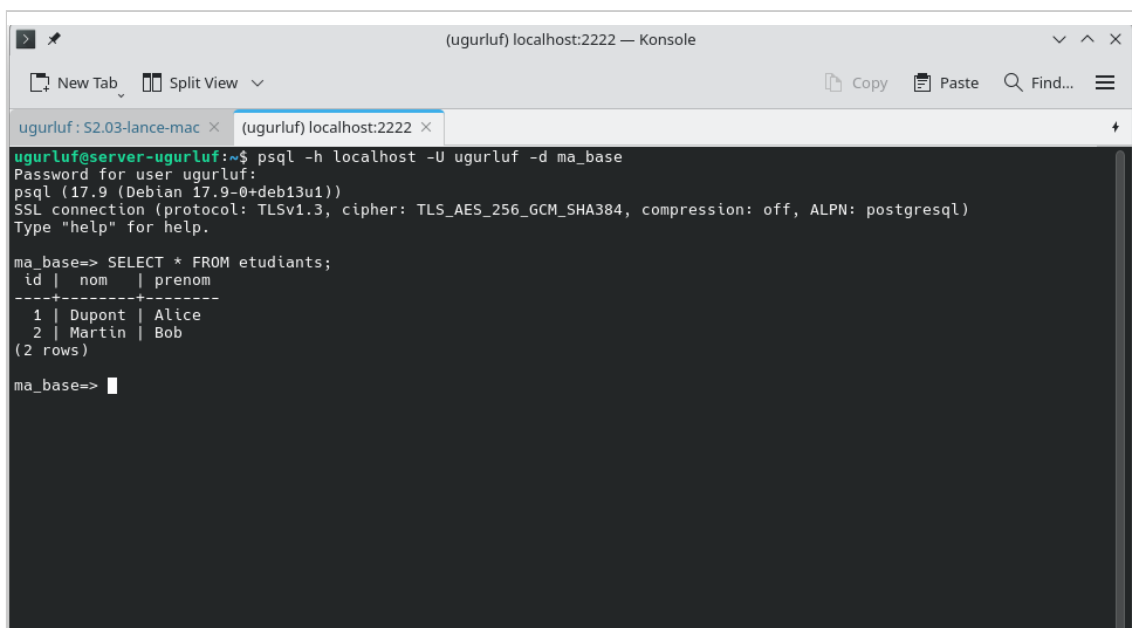
6.6.3 Restarting to apply the changes

```
$ exit # back to root
# systemctl restart postgresql.service
```

6.7 Testing from the workstation

Open a new terminal on the workstation (not inside the SSH session) and connect directly:

```
$ psql -h localhost -U yourlogin -d ma_base
ma_base=> SELECT * FROM etudiants;
```



The screenshot shows a terminal window titled "(ugurluf) localhost:2222 — Konsole". The user "ugurluf" is connected to the "ma_base" database on "localhost". The prompt is "ma_base=>". The user enters the command "SELECT * FROM etudiants;". The output shows two rows of data:

id	nom	prenom
1	Dupont	Alice
2	Martin	Bob

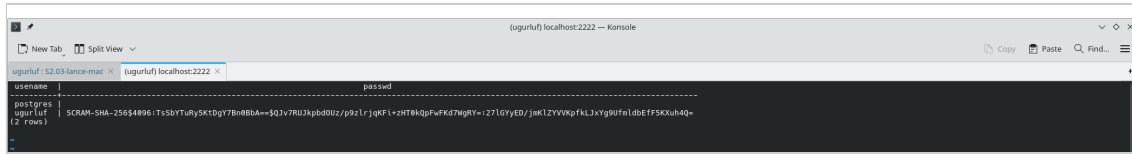
The output also indicates "(2 rows)".

6.8 Checking that passwords are properly hashed

PostgreSQL stores passwords as SCRAM-SHA-256 hashes. You can verify this by looking at the `pg_shadow` system table:

```
# su - postgres
$ psql
postgres=# SELECT username, passwd FROM pg_shadow;
```

The password column for your user should start with `SCRAM-SHA-256$`. . . . The `postgres` row being empty is normal – it uses OS-level authentication instead.



7. Installing PHP

7.1 Installation

```
# apt install php-common libapache2-mod-php php-cli
# systemctl restart apache2
```

7.2 Checking it works

Create a small test file:

```
# nano /var/www/html/info.php

<?php
phpinfo();
phpinfo(INFO_MODULES);
?>
```

Open `http://localhost:8080/info.php` in Firefox. A purple page full of PHP configuration details means it is working.

Warning. Delete that file once you have checked it. It exposes information about the server that you do not want to leave public.

```
# rm /var/www/html/info.php
```

7.3 Deploying the SAE PHP page

First run `blkid` on the VM (required by the course):

```
# blkid
```

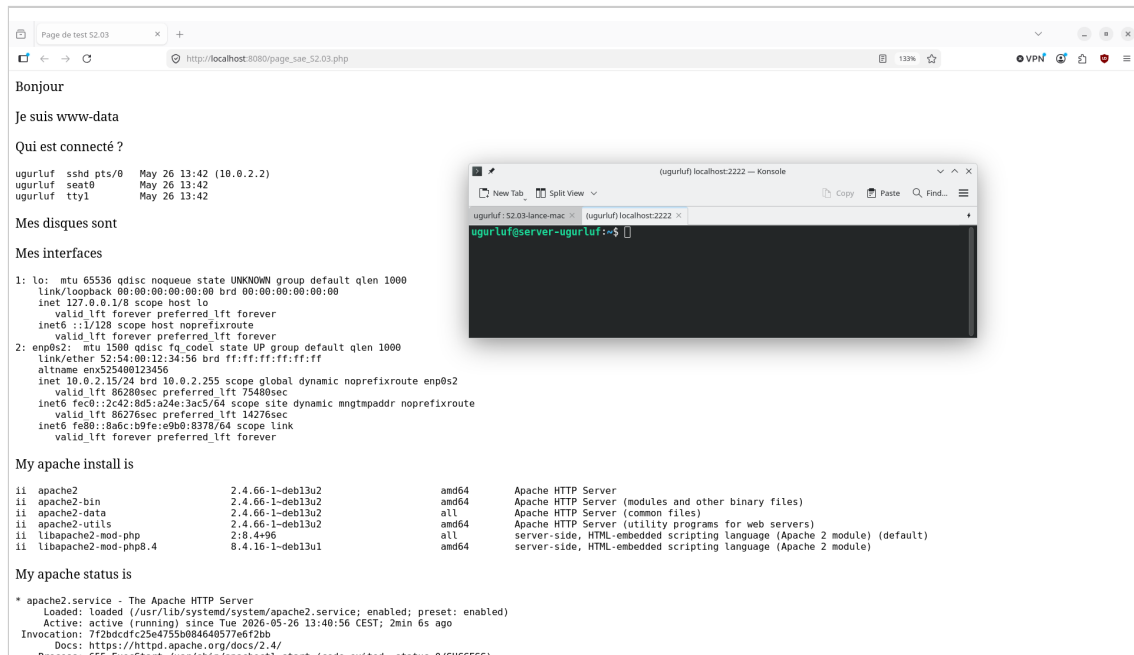
The PHP file lives on the workstation's intranet, not inside the VM, so copy it over with `scp`. Open a new terminal on the workstation and run:

```
$ scp -P 2222 \
  /users/info/www/intranet/enseignements/S2.03/page_sae_S2.03.php \
  yourlogin@localhost:/tmp/
```

Then from inside the VM, move it to the Apache document root:

```
# cp /tmp/page_sae_S2.03.php /var/www/html/
```

Open `http://localhost:8080/page_sae_S2.03.php` in Firefox.



8. Installing PhpPgAdmin

PhpPgAdmin is a web interface for PostgreSQL – useful for browsing tables and running queries without going through the terminal each time.

8.1 Installation

```
# apt install phppgadmin
```

8.2 Allowing access from the workstation

By default it only accepts local connections. Open its config and comment out the restriction:

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

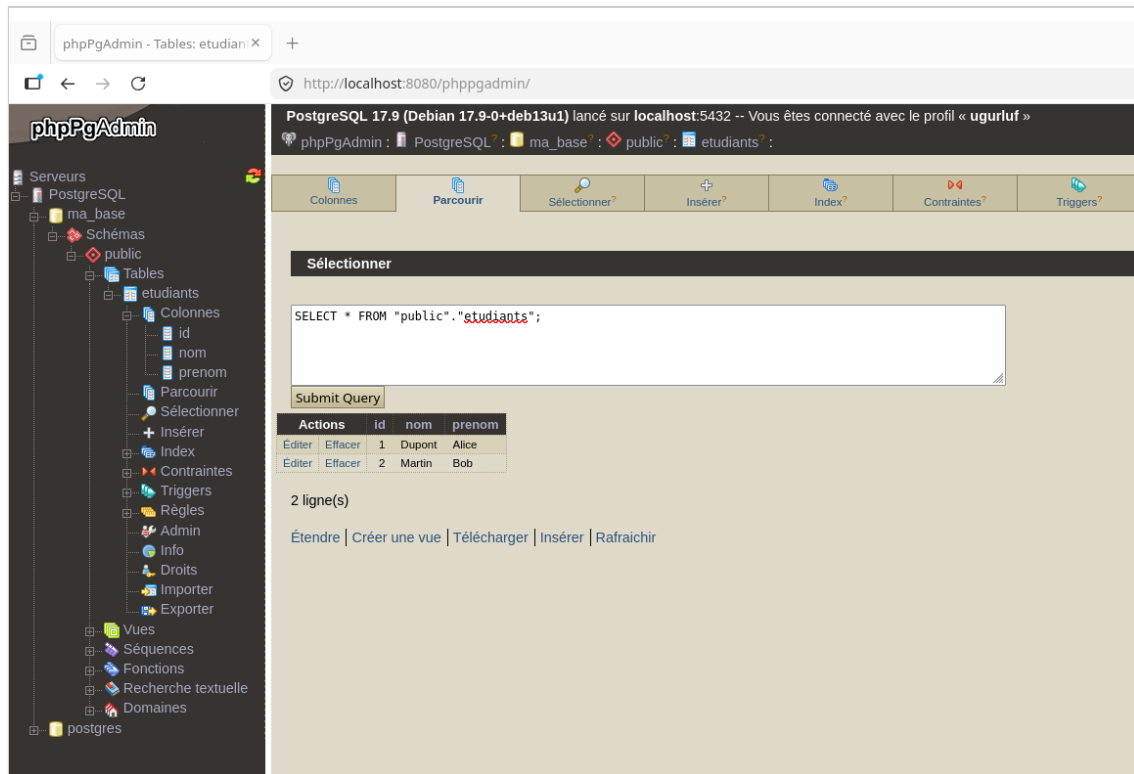
# Require local <- comment this line out

# systemctl restart apache2
```

8.3 Logging in

Go to `http://localhost:8080/phppgadmin` in Firefox. Log in with your login and your database password. Then navigate to `ma_base` → `etudiants` → **Browse**.

Note. Logging in as `postgres` is blocked by default in PhpPgAdmin, which is intentional.

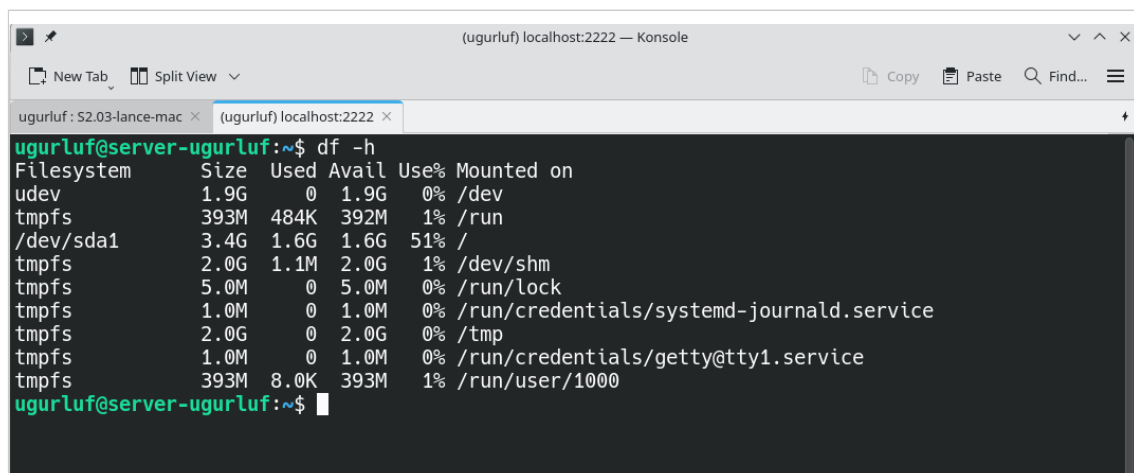


9. Disk Usage

Just to check there is still space left on the VM:

```
# df -h
```

The / line shows total, used and available space on the main partition.



PART 3 EXTENSIONS

10. SSH Key Authentication

Using a password to log in over SSH is not ideal – it can be brute-forced. Switching to key authentication removes that risk and is actually more convenient once it is set up.

10.1 Generating a key pair

Run this on the workstation, not inside the VM:

```
$ ssh-keygen -t ed25519 -C "yourlogin@iut2"
# press Enter three times
```

This creates two files in `~/.ssh/`: the private key `id_ed25519` (keep it to yourself) and `id_ed25519.pub` (the public key, which you share).

10.2 Copying the key to the VM

```
$ ssh-copy-id -p 2222 yourlogin@localhost
```

10.3 Testing it

```
$ ssh yourlogin@localhost -p 2222
# should connect without asking for a password
```

Tip. Once this is working, you can disable password login entirely by adding `PasswordAuthentication no` to `/etc/ssh/sshd_config` and restarting `sshd`.

11. Root SSH Access with a Key

By default you cannot SSH directly into root on Debian. Here is how to allow it, but only via key – not password.

11.1 Copying the key into the root account

```
# mkdir -p /root/.ssh
# cp /home/yourlogin/.ssh/authorized_keys /root/.ssh/
# chmod 700 /root/.ssh
# chmod 600 /root/.ssh/authorized_keys
```

11.2 Updating `sshd_config`

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

```
PermitRootLogin prohibit-password
PubkeyAuthentication yes

# systemctl restart sshd
```

The `prohibit-password` option lets root log in with a key but blocks password login.

11.3 Testing

```
$ ssh root@localhost -p 2222
# should connect as root without a password prompt
```

12. HTTPS with a Self-Signed Certificate

By default Apache only serves over HTTP. Adding TLS means the traffic between the browser and the server is encrypted.

12.1 Enabling SSL

Debian includes a self-signed certificate out of the box. Enable it with:

```
# a2enmod ssl
# a2ensite default-ssl
# systemctl restart apache2
```

The certificate files are at:

```
/etc/ssl/certs/ssl-cert-snakeoil.pem
/etc/ssl/private/ssl-cert-snakeoil.key
```

12.2 Testing in Firefox

Go to `https://localhost:4443/`. Firefox will show a certificate warning because it is self-signed – click **Advanced** → **Accept the Risk** to continue.

Tip. For a production server, use a proper certificate from Let's Encrypt: `apt install certbot`, then `certbot -apache`.

13. Security

The setup described above works fine for a lab environment. Before exposing any of this to a real network, the following issues should be addressed:

Issue	Fix	Priority
Packages not updated	Run <code>apt update && apt upgrade</code> regularly and consider enabling <code>unattended-upgrades</code> .	High
SSH uses passwords	Switch to key authentication and set <code>PasswordAuthentication no</code> .	Critical
Root SSH enabled	Set <code>PermitRootLogin prohibit-password</code> at minimum.	Critical
PostgreSQL open to all IPs	Replace <code>0.0.0.0/0</code> with the actual subnet in <code>pg_hba.conf</code> .	Medium
No HTTPS	See Section 12. Use Let's Encrypt for production.	Medium
<code>info.php</code> still on server	Remove it immediately after testing.	High
postgres accessible remotely	That account should never be reachable from outside the VM.	Medium

Keeping packages up to date

```
# apt update && apt upgrade
# apt install unattended-upgrades
# dpkg-reconfigure unattended-upgrades
```

Restricting PostgreSQL access

```
# In /etc/postgresql/17/main/pg_hba.conf, replace:
host all all 0.0.0.0/0      scram-sha-256
# with your actual subnet, for example:
host all all 192.168.1.0/24 scram-sha-256
```

Conclusion

After following these steps, the server is running Debian 13 without any graphical environment, with Apache, PostgreSQL 17, PHP and PhpPgAdmin all accessible from the workstation.

The most important things to remember: use APT for everything, switch to SSH keys as soon as possible, and keep the system updated. Security tends to be the thing people think about last and regret not thinking about first.