SERVERS

Part One! Don't miss next issue, subscribe on page 26!

How to build the Linux Format server

Start running your own server with the help of **David Rutland** and our new series on what to do with one once you've got it!

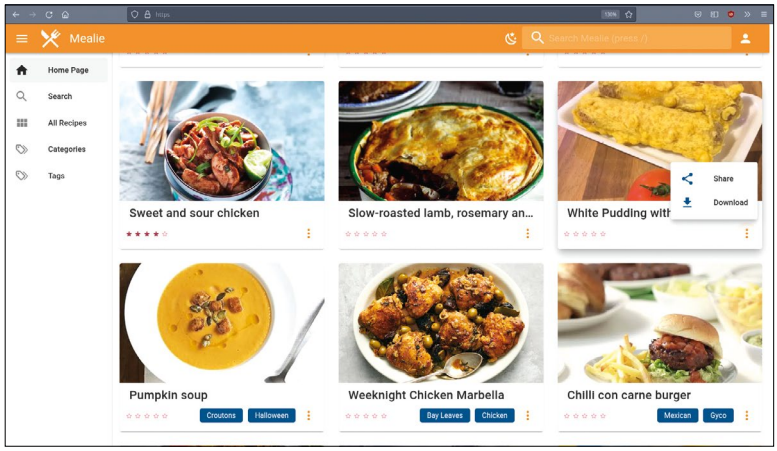
OUR EXPERT

David Rutland is a tinkerer and a dilettante. He buys domains on a whim and runs them from a Raspberry Pi behind the couch.

Okay, so you've decided you need a Virtual Private Server in your life. It's a great idea, and will allow you to run a whole bunch of self-hosted, web-facing software without the noise, expense and energy bill associated with setting up a server in the cupboard under the stairs at home. The magical orphans you already have stashed there will be pleased.

If you're a regular reader of these hallowed pages, you've probably already seen our VPS feature in **LXF281**, which covered such essentials as what a VPS actually is, why you'd want one, how to choose a provider and initial setup. In case you missed it, we've been kind enough to provide a PDF copy of the article which you can find at **https://bit.ly/lxf281lxfserver**. Go and read it then come back.

Up to speed? Good. But there is so much more you need to know, and choices you need to make. This article will help to guide you through some of the basics you need to keep your spanky new VPS safe, secure and working as it should. Don't worry, they're not difficult choices, but they will have an effect on how you install and use software.



This excellent web-facing recipe manager can be up and running with a single command thanks to Docker.

QUICK TIP

Running software services on a Virtual Private Server is as easy or as hard as you want it to be. Deploying a raft of Docker containers is simple, but you'll miss out on what's going on under the bonnet. Try doing things the long way where you can – it will give you a better idea of how your services work.

Actual server software

Your VPS is a virtual machine running in a datacentre in a different part of the world. You're going to be accessing the services you run on it via one or more domain names. But how do you ensure that you connect to the right service?

For instance, if you've installed *Jellyfin* to manage your media, *Mealie* to keep track of your recipes, *Wger* to track your personal fitness and *FreshRSS* to aggregate the latest news on your behalf, you'll need four separate domain names, or at the very least sub-domains (many self-hosted sites need to function from a domain root). And this collection of names will need to point at the static IP address of your VPS.

With all of these incoming requests, your VPS needs to know how to serve the right content. It would be less than useful if, while trying to plan a balanced meal for your upcoming marathon, you were instead served up the latest depressing news titbits from *The Economist*.

Thankfully there's software out there which can help your server to direct traffic to the right directory or port, and return a response that is both coherent and appropriate to what you are trying to achieve.

If you followed part one of our VPS series you will have already installed Apache Server. Apache is old-school cool, and straight out of the box, it can do exactly what we want it to.

Apache employs user-created **.conf** files to tell it what it's supposed to be doing, and in their most basic form they are pretty simple to understand. They contain the name of the virtual server, what ports to listen on and the Document Root location, which is where the files you want to serve up are kept.

Apache conf files should be created in **/etc/apache2/sites-available/**. For instance, the **.conf** file which relates to our fabulous *Linux Format* URL shortening engine (as it was first set up) looks like this:

```
<VirtualHost *:80>
ServerName lxf.by
DocumentRoot /var/www/lxf/public/
</VirtualHost>
```

That's easy to understand. Apache is listening on

port 80 for HTTP requests to **lxf.by**. When any requests to **lxf.by** come in, they are directed to the contents of the directory **/var/www/lxf/public/**.

Apache **.conf** files will, later on, contain more interesting details such as SSL certificate locations, internal port mapping and access controls. You don't need to worry about those right this second.

With Apache, for each service you use you will need to create a **.conf** file providing the relevant details; activate it by typing

```
sudo a2ensite /etc/apache2/sites-available/yournewservice.conf
```

and then restart apache with:

```
sudo service apache2 restart.
```

When directing traffic to individual domains, Apache is acting as a reverse proxy. But Apache isn't the only server software in town which can manage your traffic for you. There are dozens of projects which are more or less specialised, easier to use or more comprehensive.

Apache's chief competitor is Nginx, which, like Apache, serves around one quarter of the entire traffic on the internet. It's faster and more lightweight than Apache, but it's more difficult to install and configure, and more importantly, doesn't allow admins to override system-wide access settings on a per-file basis.

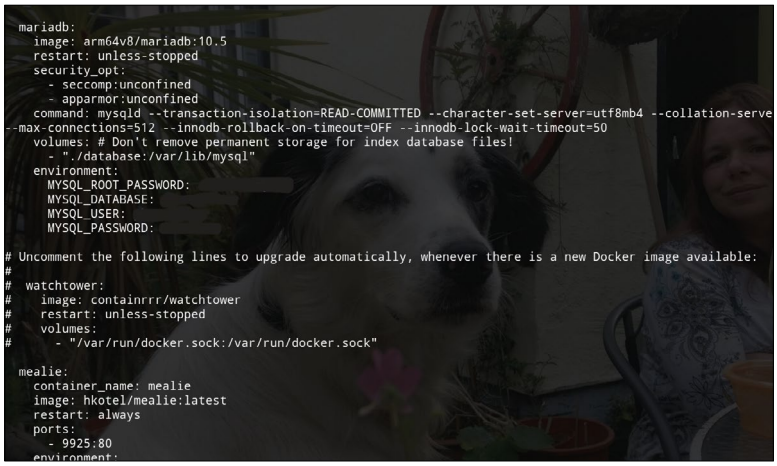
Nginx is also embroiled in an ongoing ownership dispute in the Russian Federation. We don't feel comfortable recommending a piece of software – regardless how good – which could be yoinked from the world at any moment.

tOther server software capable of running as a reverse proxy includes Hiawatha, WEBrick and Cherokee. Although each of these have their own benefits and advantages, they are not nearly as popular as Apache or Nginx, so it will be considerably more difficult to find online support from fellow users.

Security matters

See that little padlock up in the corner of your address bar? Sometimes, it will say secure, or TLS, or SSL. It means that the traffic between your machine and the webpage you're connected to is encrypted.

In theory, no-one can eavesdrop on exactly what you're doing, and even your ISP can only see what site you're visiting – not what content you view while you're there. It also means that the site is exactly what the address bar says it is. The content at **https://linuxformat.com** is created by us here at *Linux Format* – there's no man-in-the-middle injecting malware into the HTML headers or siphoning off the credit card details of our users. In addition to the tiny padlock and three letter abbreviations, you'll note that the address starts with HTTPS rather than HTTP. The S stands for secure.



```
  mariadb:
    image: arm64v8/mariadb:10.5
    restart: unless-stopped
    security_opt:
      - seccomp:unconfined
      - apparmor:unconfined
    command: mysqld --transaction-isolation=READ-COMMITTED --character-set-server=utf8mb4 --collation-serve
--max-connections=512 --innodb-rollback-on-timeout=OFF --innodb-lock-wait-timeout=50
    volumes: # Don't remove permanent storage for index database files!
      - "./database:/var/lib/mysql"
    environment:
      MYSQL_ROOT_PASSWORD:
      MYSQL_DATABASE:
      MYSQL_USER:
      MYSQL_PASSWORD:

# Uncomment the following lines to upgrade automatically, whenever there is a new Docker image available:
#
#  watchtower:
#    image: containrrr/watchtower
#    restart: unless-stopped
#    volumes:
#      - "/var/run/docker.sock:/var/run/docker.sock"

  mealie:
    container_name: mealie
    image: hkotel/mealie:latest
    restart: always
    ports:
      - 9925:80
    environment:
```

The docker-compose.yml file can be used to start and manage dozens of services. The only required skill is a mastery of Ctrl+C and Ctrl+V.

On the internet, there's a chain of trust that begins with a root certificate provider, continues through resellers and ends up with you – the server admin and end user. From the certificate provider downwards, everyone in the chain needs to prove that they are who they say they are, and that a site is not masquerading as another completely different site.

That kind of reassurance is good to have, especially when connecting to your own private content on your own virtual private server. You may not have anything to hide, but you probably don't want anyone else looking at it anyway.

If you've ever bought a hosting package, you may have noticed among the available extras there's the option to pay extra for SSL. Namecheap, for instance, charges oddly specific amounts of between £7.23 per year and £114.86 per year for the privilege of having the little padlock, and the peace of mind which goes along with it.

Thankfully, as owner and admin of your very own Virtual Private Server, you can provision your various sites with as many security certificates as you desire – at zero cost! The certificate provider we'll be using is Let's Encrypt and the tool is called *Certbot*. Like all good Linux programs it does one thing and does it very well.

QUICK TIP

Remember that a Virtual Private Server is just that – virtual. If you mess up to the extent that you need to wipe it and start again, you haven't lost a lot. Treat it as a learning experience.

» A PATCHY SERVER

Last issue we went into depth on selecting and standing up a bare VPS running Ubuntu Server (**https://bit.ly/lxf281lxfserver**).

If you missed it go back and catch up, else on a fresh install get Apache up and running this way. Ssh in as root, eg. `root@XXX.YUR.IPX.HRE` to your new server. Create a new, non-root user with its own home directory skeleton using the *useradd* command:

```
useradd -m <anew_username>
```

followed by a new password for your new user using the *passwd* command:

```
passwd <anew_username>
```

Grant your new user sudo powers so you can actually get things done by typing the following:

```
usermod -a -G sudo <anew_username>
```

After this you should log out as root and log back in as your new user. Update and upgrade the already installed packages using `sudo apt update` and `sudo apt upgrade` commands and start adding packages which will make your VPS journey possible. As a bare minimum, you should have server software such as Apache or Nginx (we're sticking with the former). You'll also need PHP and a database such as MariaDB.

You can install Apache by typing `sudo apt install apache2`, PHP by typing `sudo apt install php`, and MariaDB by typing `sudo apt install mariadb-server` then `sudo mysqlsecureinstallation` and following the prompts. See, super easy!

QUICK TIP

Certbot can be used to create wildcard certificates which will cover any new subdomain you create. This saves time and can improve security through obscurity, but comes with its own set of risks.

First of all you need to add the certbot repository:

```
sudo add-apt-repository ppa:certbot/certbot
```

then `sudo apt update` and then install *certbot* and dependencies by inputting:

```
sudo apt-get install python3-certbot-apache
```

While it's tempting to just:

```
sudo apt install certbot
```

don't do this. Yes, *certbot* will install, but it won't know how to interact with your server, and you'll be embarrassed when you have to re-consult this section to do it properly.

A dead cert

Congrats. *Certbot* is now installed on your system and ready to get to work! For this next part, we're going to assume that have a website on your server with a domain or subdomain tied to it, such as **test.lxfby**, and that you've created and activated an Apache **.conf** file with the correct configuration.

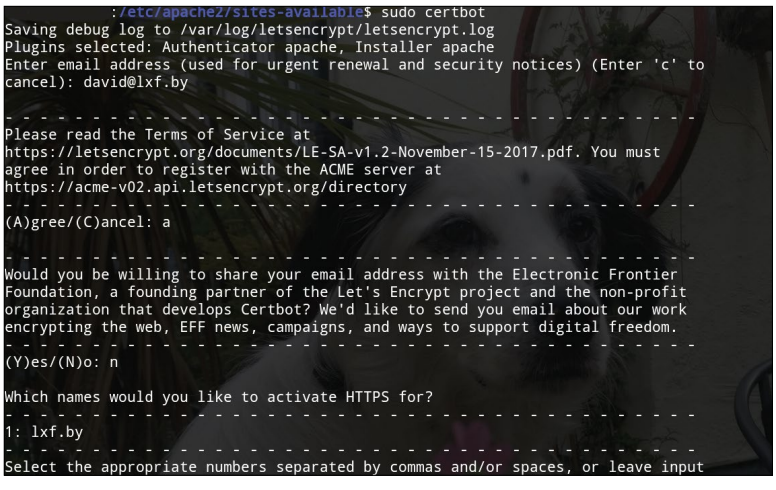
It doesn't matter what the site is, so long as it has at least one page, and is accessible from the internet. The default Apache page will do fine. Running:

```
sudo certbot --apache -d yourspankynewdomain.com
```

results in *certbot* showing you a list of domain names for which you have active **.conf** files. At this point, it's likely that you only have one of them, so it will be a very short list.

Type in the number of the domain name you want a certificate for and hit Enter. After checking that the site actually exists, and it is on the same VPS running *certbot*, the certificate authority Let's Encrypt will issue a certificate and an encryption key which *certbot* will store on your system. *Certbot* will pester you to add your email address so that the Electronic Frontier Foundation can send its monthly newsletter to your inbox, but don't feel pressured to say yes.

Using certbot to get the certificate and keys which make your VPS-based site secure is super-simple.



The last thing you need to do is to choose whether to have *certbot* modify your **.conf** files so that HTTP traffic is automagically redirected to the more secure HTTPS. After all that effort, it would be a shame not to.

And that's it. Certificates last for a maximum of three months, but *certbot*, like the good little bot it is, will renew them for you without intervention.

D is for Database

Setting up a VPS is all about choice, and nowhere is that choice more evident than with the availability of databases you can deploy.

Why do you need a database at all, we hear you ask! The answer is that the services you will be creating on your VPS are going to creating and retrieving vast quantities of data, and they need to store it efficiently.

A web-facing jukebox, as a simple example, would need to know the whereabouts of every track on every album, plus the cover art in three different sizes. It needs to be able to show you the release date, band members, bitrate and duration of the media. It needs to be able to present all of that to you without diving into the ID3 tags for every track every time you refresh the page, which would slow things down massively.

Databases help with this by keeping the data neatly structured and instantly accessible to whichever program needs them. If you've not used databases before, the names can be baffling. Who, exactly, is Maria? Does Postgres have something to do with vibrant modern art? Wasn't Mongo the standby drummer in a knock-off Beatles cover band? No-one knows, or more accurately, cares.

Ideally every self-hosted service you install should be able to use your database of choice to safely store its own data, without being able to access or alter any of the data put there by other programs. *Jellyfin*, for instance, shouldn't be able to access details about your *Jitsi* call logs or check out the private photos in your *Photoprism* instance.

Occasionally, the self-hosted software you plan on running will recommend that a particular database be used, and sometimes that will be a hard requirement. Among the most commonly employed are MariaDB/MySQL, Mongo and Postgres. They all work differently, and there are valid reasons for software developers to favour one over another.

Databases fall into one of two categories: relational databases and non-relational databases. MySQL, MariaDB and PostgreSQL fall into the first category ,

» DIFFERENT STROKES

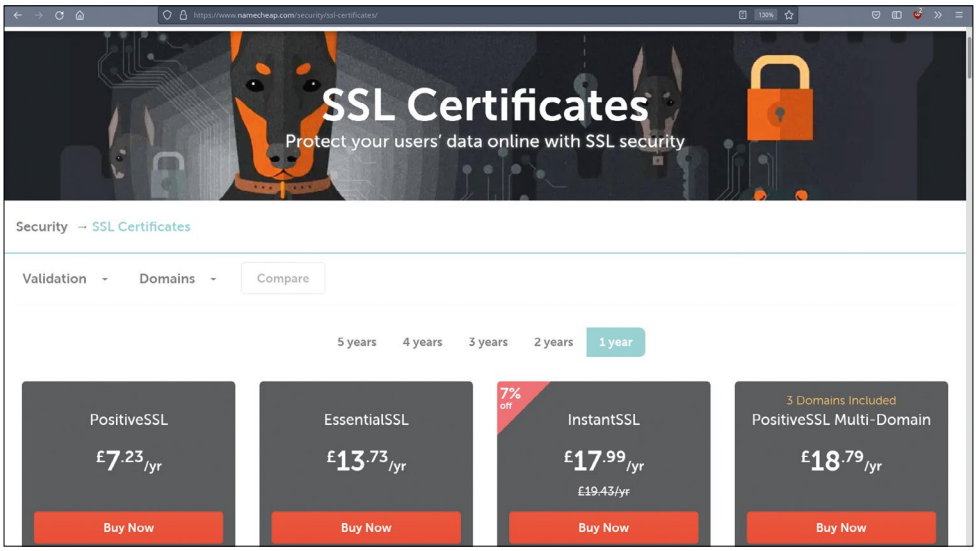
Thousands of people host their own servers, either at home or on a VPS, and their reasons for doing so vary from the need to have everything completely under their own control to trying to push the tech giants out of their life altogether.

Their setups will vary as much as their reasons for having them! We've gone with an Ubuntu server install for the *Linux Format* VPS, purely because it's the most common option, and there is a tonne of support out there for Ubuntu. Likewise, we're using Apache for our proxy server and MariaDB for our main database. You don't need to follow us exactly. You should have fun with your VPS and configure it in the way that makes the most sense to you.

Want to run the latest Arch build? Go for it. Prefer Nginx, and to eschew databases altogether because you have a better idea? That's entirely up to you. It's all about customisation and building your own ideal machine. If you really, really want to, you can even deploy a licensed copy of Windows Server 2019 on your VPS. You shouldn't though.

while MongoDB is in the second. A relational database stores data in tables containing specific pieces and types of data. For example, a photo gallery could store details of pictures filenames and paths in one table and details such as GPS coordinates, camera type, exposure and so on in another. This form of data storage is called structured data.

A non-relational database is different in that it stores its data in a non-tabular form. Instead, non-relational databases tend be based on data structures like documents, which can be highly detailed while containing a range of different types of information in different formats.



Seven per cent off may seem like a great deal for SSL certs from namecheap, but less so when you can use certbot to help you get a free certificate from Let's Encrypt.

On your VPS, you'll probably want one of each type. For applications which require database software, we'll talk you through setting up users and tables as and when we come to it.

Docking around

Of course, life would be a lot easier if developers simply packaged everything you would ever need to use their programs into one neat file or command which would set itself up with a minimum of input from you, and include ready-to-roll preconfigured databases, default logins and all dependencies.

Fortunately, such technology already exists, and in the world of self-hosted software, Docker is where it's at. With a single command, Docker can pull an application, its dependencies, libraries and configuration files from Docker Hub, and have them running in a virtualised container accessible through a virtual port on your Virtual Private Server in minutes. It's that simple.

The Docker application itself is super-easy to install from the default Ubuntu repositories with:

```
sudo apt install docker.io
```

Docker should be run when your VPS starts up. To do this you need to:

```
sudo systemctl start docker
```

and then:

```
sudo systemctl enable docker
```

As an example of how incredibly simple it is to get something up and running with Docker, try typing this into your terminal:

```
docker run -p 9925:80 -v pwd:'/app/data/' hkotel/mealie:latest
```

All done? Provided you don't have any errors (you shouldn't), you'll find that if you navigate to port 9925 on your VPS (such as **VPS.IP.ADDRESS:9925**), there will be an instance of *Mealie* – a fully self-hosted recipe manager – waiting for you. All set up and ready for the default admin user name and password! You didn't need to set up the database: you didn't need to worry about initial authentication. It's a one-line wonder!

But pulling or running Docker images one at a time can be a chore – especially if your VPS is running multiple services. That's time that could be better spent scouring the internet for more self-hosted projects to make your life both easier and more complicated.

That's where Docker Compose comes in. It enables you to create a text file which will run and manage multiple containers at once. With it, you can set persistent volumes, choose which version of the software you want to run, and get more involved in the configuration of your services.

To install Docker-Compose, first install Docker as above and then:

```
sudo apt install docker-compose
```

Boom! Done

The Docker Compose file is called **docker-compose.yml**, and is written in YAML, which stands for YAML Ain't Markup Language. That's because in the early 2000s, if your project didn't employ counter-intuitive recursive backronyms, it was considered uncool, or possibly even LAME.

To edit your compose file use `nano docker-compose`. You'll notice it's completely empty, and waiting for you to fill it with the details of whatever project takes your fancy. On the project page of *Mealie*, **https://hay-kot.github.io/mealie**, you'll see two sample docker compose files: one using SQLite, and one using Postgres. Make your selection, then copy and paste the contents into **docker-compose.yml**. Save the file (Ctrl+O if you're using *nano*), then from your home directory do:

```
docker-compose pull
```

This will pull the relevant images onto your VPS, and `docker-compose up -d` will start them as containerised applications in the background.

The beauty of **docker-compose.yml** is that more and more services can be appended to it. Fancy trying out the *Photoprism* gallery app? Simply get the contents of the default **docker-compose** file from the project page, and stick it into your own. You can download, launch and manage dozens of applications this way. Hundreds! Thousands (*steady now!–Ed*), even!

Do remember, though, that you only have limited storage space on your server, so don't get *too* carried away until you really know what you're doing. LXF