Proxmox Essentials – From Media Server to IT Applications and Testing Workflows

A Journey with Anthony Pierre

Preface

I joined Ten10 looking for a change in career. I had a significant history working in the hospitality sector. As such, the technical aspects of working within an IT company can be...challenging! So, it was up to me, and I suppose all of us who have undertaken the Ten10 journey through the Academy, to try to understand as much as we are able, to better fit into this new environment, thus improving our prospects within our client roles.

On a personal level, I had some experience with hardware as there was a time that I liked to build PC's. Now I am very much a fan of smaller '1 litre' PCs and have started to use various iterations of them as my day to day desktops and at this point I have converted a couple of them to home servers, as they are quiet and powerful.

I realised at quite an early point of my academy training, that programming was not my strength, and I was grateful to the trainers and operations, that they managed to find me a suitable client role. But to increase my comfort levels in my new working environment, I decided that I would attempt to undertake some small technical projects during my bench time after completing the client specific training, and the writing of this tutorial is to consolidate the learning that I acquired in this endeavor. I recognise that I still have a lot to learn, but I am happy to share what I have learnt so far.

Many of these install instructions are of course available online, however, in the future as I go into more advanced configurations, I found that information was scattered over numerous locations, so this is an attempt to consolidate everything I have learned so far into one document, while also adding a range of tips and explanations that will help with the installation while guiding less experienced users. My first 'project' was to see if I could use Proxmox to install a media server.

What are Virtual Environments?

During my academy training, while in a session at the Leeds office, one of the trainers mentioned using VirtualBox. VirtualBox is an open-source virtualisation platform that allows users to create and run virtual machines (VMs) on their computers. It's designed for developers and IT teams, and can be used to:

- **Test software**: Allowing the running of multiple operating systems simultaneously on a single device.
- **Create isolated environments**: Creating isolated environments for development and experimentation.

I had used VirtualBox but in a limited capacity, just playing around and experimenting with Linux installs. By keeping the linux within the VM it meant that I would not have to disturb

my Windows install. VirtualBox is a useful and comprehensive software, however and could have been my choice to continue this project. But I decided for my needs, I would use the Proxmox Virtual Environment (PVE).

What is Proxmox?

Proxmox is an open-source virtualisation platform designed to efficiently manage and deploy virtual machines (VMs) and containers. Introduced in 2008 by Proxmox Server Solutions GmbH, PVE is built on Debian Linux and leverages the KVM (Kernel-based Virtual Machine) hypervisor alongside Linux Containers (LXC) to provide a comprehensive solution for server virtualisation. Proxmox has steadily gained traction for its ease of use, cost-effectiveness, and flexibility, particularly in the enterprise and homelab markets which is where I came across it while searching for a way to understand the different flavours of Linux.

Key Features of Proxmox

Proxmox offers a robust set of features that make it a popular choice:

- **Web-Based Interface:** A fully-featured GUI simplifies the management of VMs, containers, storage, and networking.
- **Cluster Management:** Users can create a cluster of multiple nodes for high availability and resource scaling.
- **Backup and Restore:** Integrated backup and restore capabilities support incremental backups and flexible scheduling.
- **Support for Containers:** In addition to VMs, Proxmox supports LXC, which offers lightweight virtualisation for isolated applications.
- **Software Defined Storage (SDS):** Includes support for ZFS, Ceph, NFS, and iSCSI storage systems.
- **Third-Party Integrations:** Offers APIs and tools like Proxmox Backup Server for advanced backup strategies.

Rival Software in the Virtualisation Space

Proxmox competes with several prominent virtualisation solutions:

1. **VMware vSphere/ESXi:** A commercial hypervisor with advanced enterprise features, often seen as Proxmox's main competitor. While VMware is feature-rich, its licensing costs are a drawback for smaller operations.
2. **Microsoft Hyper-V:** A hypervisor bundled with Windows Server, popular in enterprise environments where Microsoft ecosystems are predominant.
3. **Oracle VirtualBox:** A free, cross-platform hypervisor primarily used for desktop virtualisation, not as robust for enterprise-level deployments.
4. **Citrix Hypervisor (formerly XenServer):** Known for its strong focus on application virtualisation but increasingly less competitive in the broader hypervisor market.

Unlike many of its rivals, Proxmox's open-source model and community-driven development make it a highly accessible option for small and medium-sized enterprises (SMEs), non-profits, and individual enthusiasts. Like myself!

Applications and Uses of Proxmox

Proxmox is versatile, lending itself to a range of use cases:

- **Enterprise Virtualisation:** Companies use Proxmox to host business-critical applications on VMs while maintaining resource efficiency through containerisation.
- **Homelabs and Learning Environments:** Its cost-free model and flexibility make it ideal for hobbyists and IT professionals experimenting with virtualisation.
- **High-Availability Clusters:** Proxmox is often employed in clusters to ensure uptime for essential services.
- **Development and Testing:** Developers benefit from creating isolated environments for testing applications and workflows.
- **Self-Hosting and Cloud Infrastructure:** Proxmox supports a variety of self-hosted services, from media servers like Plex to web applications and databases.

After understanding a little more about why Proxmox is so well received in the homelab space, I decided on a ‘Bare Metal Install’. I have various HP Small Form Factor (SFF) PCs lying around my home and wanted to do something with the most powerful. This was an HP Mini Elite G8. This has a 12th generation Intel processor, 16 GB of ram and 512Gb of hard drive space. I deduced that by using an LXC, to begin with and expanding my range later on by utilising Docker I could optimise my hard drive space without compromising on functionality, as LXC;s are lightweight in operation as opposed to having to run a full operating system in order to run an individual item of software. The container encapsulates **just** the software and its dependencies and resources to run. This is very similar to how Docker manages its containers for deployment.

Why Choose Proxmox for Your Server?

Proxmox Virtual Environment (VE) is designed to function as a dedicated hypervisor, providing a robust and efficient platform for managing virtual machines (VMs) and containers. Unlike traditional operating systems like Windows or desktop Linux distributions, which are built for multi-purpose use, Proxmox excels in running virtualised environments directly on the hardware—a method often referred to as a bare-metal installation.

A bare-metal installation means that Proxmox will be the sole operating system installed on the machine, fully utilising its resources to optimise the performance of virtual machines and containers. While it is technically possible to run Proxmox as an application within another OS or even dual boot it, these setups are impractical for most users and can compromise the system's stability and efficiency.

By dedicating your hardware to Proxmox, you unlock its full potential as a hypervisor, enabling advanced features such as high availability, clustering, and seamless virtualisation. Once installed, Proxmox provides the tools to deploy and manage an entire virtualised infrastructure with ease, whether you're running lightweight Linux Containers (LXC), full virtual machines, or testing IT applications.

With this in mind, the first step is to prepare your system for Proxmox installation by creating a bootable ISO image. Let's begin.

Part 1: Installing Proxmox VE

1. **Download Proxmox VE ISO**:
    - Visit Proxmox VE Downloads.
    - Download the latest ISO installer.

2. **Create a Bootable USB Drive**:
    - Use a tool like Rufus (Windows) or Balena Etcher (Linux/Mac).
    - Flash the Proxmox ISO onto a USB drive.

3. **Install Proxmox VE**:
    - Boot your system using the USB drive.
    - Follow the on-screen instructions:
        - Select the target disk for installation.
        - Configure your time-zone and root password.
        - Assign a hostname and network configuration.

4. **Access Proxmox Web Interface**:
    - Open a browser and navigate to http://<Proxmox_IP>:8006

Tip – When you see the < > these are not to be entered. So, to access the Proxmox installation after this installation for example it would be http://192.168.0.123:8006. This would be placed into the address line of your browser of choice.

    - Log in using the root credentials.

This takes care of the main install.

Let's look at Step 4 in a little more detail. The IP address is automatically given to Proxmox during the install. This is provided from the internet router using the Dynamic Host Configuration Protocol system. (DHCP). The router uses its own determination to assign the address, and this address is written to a configuration file within the Proxmox. We used the example of http://192.168.0.123:**8006** for instance.

There is nothing wrong with this address as this was the address that was available at the time, although there is a chance that this address could change after a reboot, this would mean our http://192.168.0.123:**8006** could change to http://192.168.0.**124**:**8006** for example. But there is a chance that the router would continue to reserve this address for you based on the DHCP server renewing the lease for the same address, or the router assigns the same address based on its configuration.

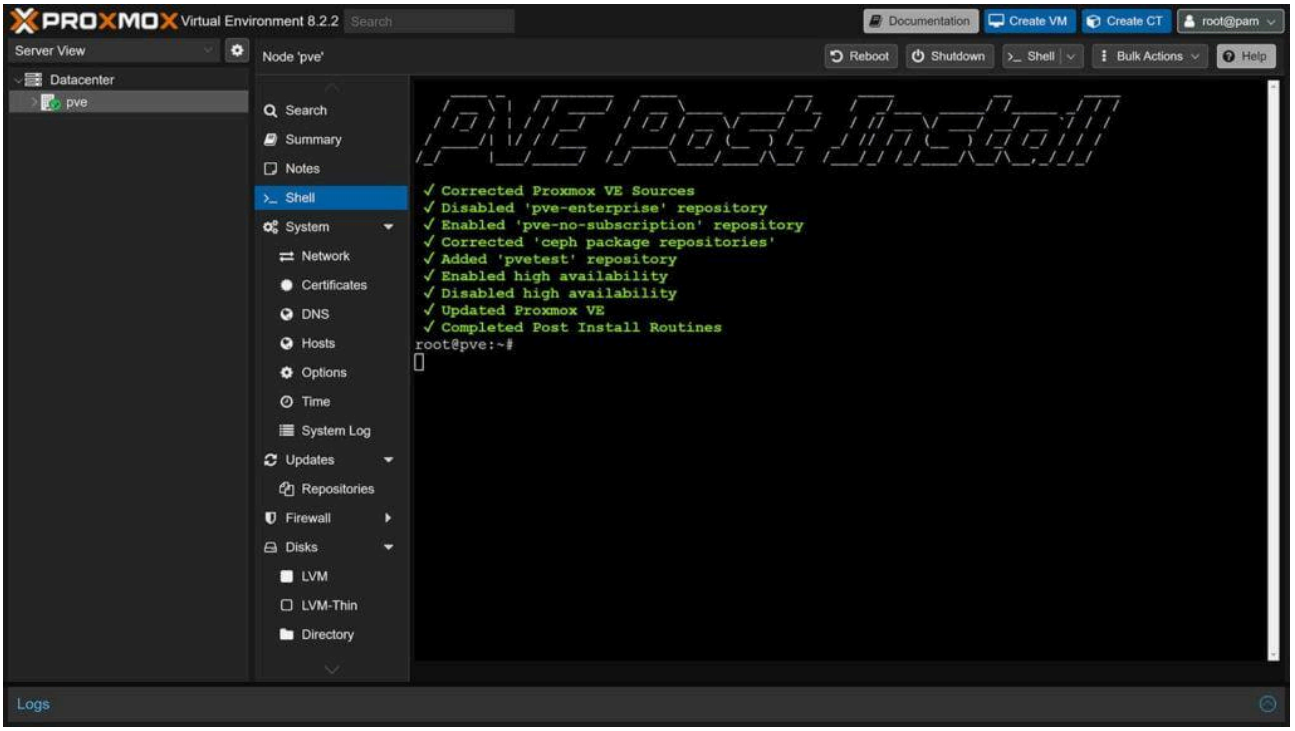
This system is ‘headless’. Meaning that after install I will have no monitor or keyboard attached, and access the system using my laptop. To prevent the potential loss of IP address I would configure Proxmox to use a ‘static IP’ meaning the address would not change.

Tip - You can see that I have highlighted the port 8006. I do this to show how important these additional port numbers are at this stage, as they are used to open the local address of this and future installs. This will be discussed in more depth in a future tutorial.

Configuring a Static IP for Proxmox

This is not a mandatory step.

As explained previously, by default, Proxmox uses DHCP to obtain an IP address, but a static IP ensures the host has a consistent network identity. Accessing the shell of the PVE brings you to the root command line. (See picture below.)



1. **Locate Proxmox's Network Configuration File**:
    - The network configuration file is stored in a folder at /etc/network/interfaces.
2. **Edit the Network Configuration File**:
    - Open the file using the inbuilt text editor called nano:

nano /etc/network/interfaces

3. **Change** dhcp **to a Static IP Configuration**:
    - Replace the DHCP configuration with a static IP setup. For example:

```
auto vmbr0
iface vmbr0 inet static
    address ~~192.168.0.123~~ 192.168.0.10
    netmask 255.255.255.0
    gateway 192.168.0.1
    bridge-ports enp0s31f6
    bridge-stp off
    bridge-fd 0
    dns-nameservers 8.8.8.8 8.8.4.4
```

- **address**: The static IP you want for Proxmox. I have used 192.168.0.10 in this example totally removing and replacing the previous 192.168.0.123 entry.
- **netmask**: Your network's subnet mask.
- **gateway**: The default IP addresses list of your router.

Tip – Ensure that the ip address that you choose is not in use. You will have to determine the best method to find out that information depending on your individual hardware.

4. **Apply the Changes**
    - Restart the networking service

```
systemctl restart networking
```

- Alternatively, reboot the server

```
reboot
```

5. **Verify the Static IP**
    - After reboot, check the IP address with

```
ip addr show vmbr0
```

Setting Up a Container

Proxmox is written using Debian, but also plays well with Ubuntu as well. For this reason I use either Debian or Ubuntu based templates as the base for my containers. Normally there may be a preference noted in the software that would be installed in the container. The templates can be created using any Linux distribution if there is a need for a particular base environment, but Debian and Ubuntu seemed to be the most popular for my use case. Even Windows can be made into a template, but there are additional steps needed at the creation, installation and usability stages to be compliant with Windows licensing requirements. Linux is a mostly free software platform, with only certain enterprise distributions such as Red Hat Linux and some consumer-focused distributions, such as Zorin or Elementary asking for payment, but this is normally for premium support or additional bundled items.

When we think of the template you should see it as a base image of nearly everything that is needed to get a piece of software up and running. For further visualisation think that the current Windows install is around 64GB! Remember that is the base installation, just to have the operating system running at a standard. Programs can be shaved out of the install and there are a few open source projects in order to get windows down to as small a size as possible, but this operates in a legal grey area, as Windows is a proprietary software.

Which Template Should I be Using?

In order to make this tutorial relevant to the working environment at Ten10, I am going to answer this question based on testing, which is probably still the bread and butter of our business, and how we would deploy Proxmox for use in this specific area.

When deciding between a Debian or Ubuntu-based template in Proxmox for use in a testing environment, the choice depends on the specific requirements of your testing scenario. Factors like software compatibility, ease of deployment, resource usage, and frequency of updates play a significant role. Below is a detailed comparison with an emphasis on how these templates would function within a testing environment with some examples and use cases.

1. Base System

- **Debian**
    - Debian provides a solid, minimal base ideal for system-level testing, where the focus is on stability, reproducibility, and resource efficiency.
    - In a testing environment, Debian's ‘testing’ branch can be used if access to newer packages is needed without the overhead of frequent Ubuntu updates.
    - For example testing custom scripts, server configurations, or applications on a reliable and lean OS.
- **Ubuntu**:

    - Ubuntu's broader feature set and pre-installed utilities make it a good choice for application-level testing, where the OS is a means to deploy and test software quickly. i.e testing modern applications that require newer libraries or dependencies.

2. Release Cycle

- **Debian**
    - Stable: Best for testing long-term compatibility and reliability of software in production-like conditions.
    - Testing/Unstable: Great for testing the latest software in development or pre-production environments, mimicking cutting-edge scenarios.
    - Use Case - If you're developing software for production environments that use Debian, the 'stable' branch ensures compatibility, while the 'testing' branch allows you to catch potential issues with upcoming updates.
- **Ubuntu**
    - LTS (Long-Term Support): Ideal for creating predictable and stable test environments.
    - Interim Releases: Useful for testing with the latest software stacks, especially if you're working with rapidly evolving tools (e.g., Kubernetes, Docker, or AI/ML frameworks).
    - Use Case: For the testing of software that targets cloud environments or requires modern tooling not available in Debian 'stable'.

3. Performance in Testing

- **Debian**
    - Lower resource overhead makes Debian templates ideal for resource-constrained environments or running multiple concurrent tests.
    - Use Case - The setting up of multiple lightweight containers to simulate different nodes in a distributed system.
- **Ubuntu**
    - Slightly higher resource usage, but the availability of newer libraries and tools can speed up deployment.
    - Use Case - Running a Continuous Integration (CI) and Continuous Deployment/Delivery (CD) pipeline in containers where quick deployment and updates matter more than resource constraints.

4. Software Availability

- **Debian**
    - Ideal for testing software stability with well-tested versions of packages.
    - The 'testing' or 'unstable' branches can be used if the goal is to ensure compatibility with the latest software versions before they are officially released in 'stable'.

    - Example - Validating a server application against stable library versions to prevent regressions.
- **Ubuntu**
    - Preferred for testing new features or modern applications that require up-to-date libraries, drivers, or frameworks.
    - Example - Testing a web app using the latest Node.js or Python version without needing to compile or add custom repositories.

5. Customisation and Ecosystem

- **Debian**
    - Highly customisable, making it ideal for testing environments that need tailored OS configurations.
    - Example: Setting up minimal containers with only essential services for performance benchmarking.
- Ubuntu
    - Easier to set up with built-in utilities and wider driver support.
    - Example: Rapidly deploying multiple environments with pre-installed tools for testing end-user scenarios.

6. Network Testing

- Debian
    - The lightweight nature of Debian makes it ideal for testing network configurations, such as firewalls, VPNs, and routing.
    - Example - Simulating a small-scale network with multiple containers acting as routers and clients.
- Ubuntu
    - Better suited for application-level network testing, such as testing web servers or APIs under different conditions.
    - Example - Load testing a web application stack using Ubuntu containers with modern HTTP/3 support.

7. Long-Term Testing and Compatibility

- Debian
    - Use Debian ‘stable’ for long-term testing environments that need consistency over months or years.
    - Example - Creating a testing environment for a legacy application to ensure compatibility during updates.
- Ubuntu
    - Use Ubuntu LTS for environments that need to mimic enterprise systems with vendor-supported configurations.
    - Example - Testing software integrations in a hybrid cloud setup with a mix of on-premises and cloud deployments.

8. Practical Scenarios

Scenario	Debian Template Usage	Ubuntu Template Usage
System-Level Testing	Testing minimal OS setups, kernel tweaks, or configurations.	Less suitable due to added utilities and overhead.
Application Compatibility Testing	Stable branch ensures a reliable base for regression tests.	Ideal for newer applications requiring recent dependencies.
Performance Benchmarks	Minimal resource usage is perfect for stress testing systems.	Better for testing end-user applications under real-world conditions.
CI/CD Pipelines	Lightweight containers speed up environment preparation.	Wider software support streamlines deployment workflows.
Multi-Node Simulations	Best for creating resource-efficient simulated nodes.	Useful if the nodes need modern libraries or proprietary drivers.

- **Choose Debian**:
  - When stability and minimalism are key (e.g., system-level testing, compatibility checks for production environments).
  - For scenarios requiring resource efficiency or when testing environments need to mimic traditional server setups.
- **Choose Ubuntu**:
  - For rapid prototyping and testing of modern applications or software stacks that depend on newer packages.
  - When testing in environments that require frequent updates or mimic enterprise/cloud systems.

Exploring Advanced CI/CD Tools

To conclude this dense section, as you continue your journey in building scalable and efficient environments with Proxmox, integrating advanced CI/CD tools like **J**enkins, GitLab CI, and GitHub Actions can further enhance your workflows. These tools allow you to automate repetitive tasks, ensure consistent testing, and deploy applications seamlessly across your Proxmox infrastructure.

- **Jenkins** - Known for its flexibility, Jenkins is ideal for organisations that need highly customisable pipelines or manage multiple environments. It can integrate with Proxmox to automate builds, tests, and deployments across containers and virtual machines.
- **GitLab CI** - If your code repository resides on GitLab, its built-in CI/CD pipelines offer a streamlined solution. GitLab Runners can be deployed in Proxmox containers to handle the entire build-test-deploy cycle efficiently.
- **GitHub Actions** - For projects hosted on GitHub, GitHub Actions provides an intuitive way to define workflows, integrate with third-party tools, and automate deployments to your Proxmox environment via SSH or APIs.

Each of these tools aligns well with Proxmox's flexibility, making them valuable additions to your testing, staging, or production setups, all of them can be installed either directly from the command line or through using Docker.

Case Study: Plex Media Server on Proxmox

One of the practical applications I explored was setting up a Plex Media Server on Proxmox. Plex is a powerful media management tool, and deploying it on Proxmox demonstrates the platform's capability to host lightweight and efficient applications.

Step 1: Download the Debian & Ubuntu Templates & Create the Container

Proxmox uses templates to create LXC containers. These templates are available from the Proxmox VE Content Library.

1. **Access the Content Library**:
    - Log in to the Proxmox web interface.
    - Navigate to **Datacenter > Node > Storage**.
    - Select a storage option (e.g., local or local-lvm).
    - Click the **Content** tab.
2. **Download the Debian Template**:
    - Click the **Templates** button at the top.
    - Search for the desired template.
    - Select the template and click **Download**.
    - The template will be saved in your selected storage under local.
3. **Start the Container Creation Wizard**:
    - In the Proxmox web interface, click **Create CT** at the top-right corner of the screen.
    - Provide the following information in the wizard:

      Node: Select your Proxmox node.

      Hostname: plex-server.

Password: Set a strong password for the container (e.g., password123 for this example). Ensure you note it down.

Template: Under the Template tab, select the downloaded Ubuntu template.

Resources: Configure the following default values:

CPU: 2 cores.

Memory: 2048 MB (2 GB).

Disk: 16 GB.

Network: Assign a static IP address under the Network tab:

IP Address: 192.168.0.100/24.

Gateway: Your network gateway (e.g., 192.168.0.1).

DNS Server: Your preferred DNS server (e.g., 8.8.8.8 for Google DNS).

Confirm

Review the settings and click Finish to create the container.

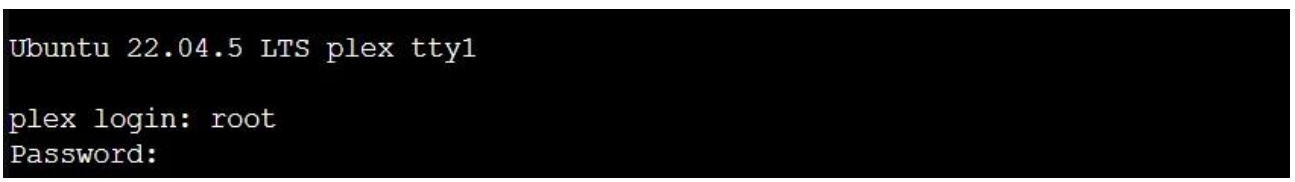
Explanation - These initial steps have created an LXC container with Ubuntu as its base template. To do this we selected the Proxmox node. We only have one so it was an easy choice, but the power of the software is that multiple nodes (PC's/Proxmox instances) can be daisy chained together for failover implementations. We named the container plex-server, set a password and chose the appropriate template. We then allocated cpu cores, this can vary as to your system needs. We set memory and disk space. Finally we set network parameters by allocating a static ip, in a different range from the Proxmox host. We then set both network and gateway and dns servers. 8.8.8.8 is used or another popular DNS server is 1.1.1.1 provided by cloudflare.

Step 2: Start and Access the Container

- **Start the Container**:
  - In the Proxmox interface, locate your newly created container.
  - Right-click the container and select **Start**.
- **Access the Console**:
  - Right-click the container again and select **Console** to access the terminal.

Step 3: Installing Plex Media Server

On entering the container, and on selecting the console the console window will ask for a login name.



This is the ‘root’ login as used for Proxmox and the password would be the one used at the container creation. This will take you to the main root command line interface.

This has now taken us into a fresh template install and it is now ready for the steps to be taken to install Plex. The text in the boxes below are the commands that would be entered into the command line. I have also tried to provide a little more depth as to what the commands are actually doing. This was my effort to try to gain a greater understanding of the Proxmox environment.

```
apt update && apt upgrade -y
apt install -y curl gnupg
```

apt update && apt upgrade -y

This line performs two operations in sequence, separated by && (which means the second command will only run if the first one completes successfully).

- apt update

This command updates the list of available packages and their versions. It fetches the package lists from the repositories configured on your system (e.g., Debian or Ubuntu repositories). This ensures your system knows about the latest software versions, security patches, and new packages. Running apt update ensures that your system is aware of any updates to the packages you have installed and that you are working with the latest available information.

- apt upgrade -y

This command upgrades all installed packages on your system to their latest available versions, based on the package lists updated by apt update. The -y at the end automatically answers "yes" to any prompts that might ask you whether you want to proceed with the upgrades. This is useful in automated scripts or when you don't want to be prompted to confirm the upgrade process.

apt install -y curl gnupg

- apt install -y

This command installs new packages or updates existing ones to the latest version available in your system’s repositories. The -y flag, like in the previous line, automatically answers "yes" to prompts that ask whether you want to proceed with the installation.

- curl

Curl is a command-line tool used for transferring data using various network protocols, such as HTTP, HTTPS, FTP, etc. It’s commonly used for downloading files, fetching URLs, or making requests to APIs. It's a very versatile tool often used in scripts to download files from the internet, interact with APIs, or fetch external resources.

- gnupg

gnupg is a tool for secure communication. It is the GNU Privacy Guard (GPG) software used for encryption, signing, and verification of files and messages. It's often used for securely downloading and verifying software packages, especially when adding third-party repositories or keys, like we are doing with Plex. Many software providers require GPG to securely add their signing keys to your system to verify the integrity of packages or software you install. This is especially critical when adding external repositories to your package manager (e.g., for installing Docker, third-party repositories, etc.)

```
curl https://downloads.plex.tv/plex-keys/PlexSign.key | apt-key add -
```

curl https://downloads.plex.tv/plex-keys/PlexSign.key | apt-key add

- curl https://downloads.plex.tv/plex-keys/PlexSign.key:

This command uses curl to download the GPG key used to sign Plex packages from the specified URL. This key is required to verify the authenticity of the Plex Media Server packages you install, ensuring that they haven't been tampered with.

- | apt-key add -

This part takes the downloaded key (from curl) and pipes it directly into the apt-key add - command, which adds the key to the list of trusted keys on your system. By adding the key, you're telling apt that it can trust packages signed with this key. Without this step, your system would refuse to install Plex packages as it wouldn't recognise the key.

```
echo "deb https://downloads.plex.tv/repo/deb public main" >
/etc/apt/sources.list.d/plexmediaserver.list
```

```
echo "deb https://downloads.plex.tv/repo/deb public main" >
/etc/apt/sources.list.d/plexmediaserver.list
```

- echo "deb https://downloads.plex.tv/repo/deb public main"

creates a line of text that specifies the Plex repository. The URL https://downloads.plex.tv/repo/deb public main tells the system where to find the Plex Media Server packages. The deb indicates this is a Debian-based repository, and public main is the repository's distribution channel and section. You need to add the Plex repository to your system so that apt knows where to download the Plex Media Server packages from.

- > /etc/apt/sources.list.d/plexmediaserver.list

This redirects the output of the echo command (the repository URL) to a new file in /etc/apt/sources.list.d/plexmediaserver.list. By saving this line into a .list file, you add the Plex repository to your package manager's sources list. This allows apt to locate Plex packages when you run apt update and apt install.

```
apt update
apt install -y plexmediaserver
```

- apt update

This command updates the list of available packages from all repositories listed in your system's sources. It checks the repositories (including the newly added Plex repository) to see if there are any new or updated packages available.

- apt install -y plexmediaserver

This command installs the Plex Media Server package from the Plex repository you added earlier. The -y flag automatically confirms the installation, so you don't have to manually approve it. This is the actual step where Plex Media Server gets installed onto your system. After running this command, the server will be ready to use.

```
systemctl status plexmediaserver
```

systemctl status plexmediaserver

This command checks and displays the current status of the Plex Media Server service. It will tell you whether Plex is running, inactive, or failed. Checking the status helps you ensure that the Plex Media Server service has started correctly after installation.

```
systemctl start plexmediaserver
```

This command manually starts the Plex Media Server service if it's not already running. After installation, the service may not automatically start. Running this command ensures that the Plex server begins running and is ready to accept connections.

After verifying that Plex has started it is now time to access the software from its unique URL. In this instance the URL will be

http://192.168.0.100:32400/web

This too can be broken down. 192.168.0.100 is the static IP address you've assigned to the container (the one running Plex), and 32400 is the default port for accessing Plex's web interface. This is the address you use to access Plex's configuration page through a web browser. Once you're on this page, you can finish setting up Plex, add media libraries, and manage your server settings.

And with that installation is complete!

I will not go into the uses of Plex as this was just a proof of concept and a stepping stone into the uses of Proxmox. As I have stated this was just my first step and I used something that would be of interest to me to make the learning more fun.

The next step I took was to use Proxmox as my repository for my Docker builds. I have been baffled by Docker for a long time, but with some application on my part, I was finally able to make some sense of it and now keep a number of Docker containers running various programs inside an LXC container in Proxmox! I also use a program called Portainer which makes management of the Docker containers more visual.

I will be happy to share that journey in my next tutorial!

If I have made any major errors please contact me anthony.pierre@ten10.com