Local and Remote Server Management

- Coolify Setup and Usage
  - Install Coolify and Confgure
- Server Management Tools
  - Install Termix
  - Install Nexterm

Coolify Setup and Usage

Install Coolify and Confgure

https://www.youtube.com/embed/cg28Ztt4-os

What is Coolify

Coolify is an application that can be used to deploy applications, databases, and services easily and quickly to your local server, or to remote server hosts. Think if it as the Application / Service part of a service like Linode, Digital Ocean, Vultr, etc.; but without the monthly fee if you run it on your hardware.

Why Coolify for Homelabbers?

We, as the homelab community, don't use our computers like your average, everyday consumer. We use our computers to service our needs, wants, and desires. We do extra work now to make things easier in the long term.

I think of my journey over the last fiver years with all of you, and I see what I've built (it's a giant mess physically) but incredibly powerful and beautiful when I really look at what I have. Allow me to elaborate for a moment:

I have a media server that serves media to my entire family on all of our devices with Jellyfin. It's amazing, and a normal word in my home. Hey, dad, can you buy this show, and rip it to Jellyfin for me? Hey, son, I was wanting this movie, can you get it and put it on Jellyfin? Dad, jellyfin loggged me out!

I run my entire network which isn't your average home network, with open source OpenWRT firmware. I have a Main Router, 3 wall plate Wifi Hotspots, that all have 4 VLANs serving my home with various networks for various reasons. I have almot 100 devices on my network (when you include VMS and Containers). It all just runs. I really don't have to mess with it.

I have NextCloud as my syncing service for myself, my wife, my mom, my daughter, and some nieces who don't even live with us. They all just use it, because I've made it the norm.

We play an Online MMORPG that I host locally, and my nieces can play from their house over the open source VPN Wireguard just like they are here with us.

We can start a voice call through Mumble running here on my servers. We chat using myown Matrix server. My family prefers Linux to Windows.

I say all of that to say, Coolify might just be my next big thing in Homelab and Professional work. Deploying all of these applications, providing best in class alternatives to all of the costly, proprietary, subscription based crapola out there now...that's the dream, and I'm moving toward it steadily.

How Can I Use this Professionally?

I believe that a tool like Coolify, combined with tools like Proxmox, or LXConsole and Incus Containers, can make spinnin up and down servers and services for your clients a very simple process. Here are the basic steps behind my thoughts.

1. Create a base VM / Container (I'll use Incus as my example)
2. Make sure you have Docker, Docker Compose, OpenSSH, Git, Curl, WGet installed on this base container.
3. Use this as your lowest level template container, which you then create a new container base per client you serve.
4. This new container base for each client will then have a separate SSH key that you generate for the client on it. This will separate it from the other client containers. Each container I make for this client will use this base as a starting point.
5. Coolify can then use this container to deploy any and all services, Applications to. We'll only need 1 coolify install. Coolify can reach out to all of these servers, and take actions for us.
6. So, now we have the DO type experience, but with our own infrastructure. We could even run this and manage client installs on DO, Linode, Hetzner, and so on.
7. Create the server (just like in DO), Put an Application on it, or several if we want.
8. We manage our clients DNS / Domains / Name Servers, so we just need to create reverse proxy entries for all of their services.
9. Backups of Incus containers are extremely portable, and make it great for keeping our clients data safe in case of DR needs as well.

Install and Setup Coolify

As an Managed Service Provider you will be doing many different tasks to make sure your clients are always up and running. There are some absolutely amazing tools that can help with this out there, and we've been covering them little by little.

One of the tasks is setting up services for your clients to run. Maybe they want a specific application, or maybe their IT person needs a database setup that they can use and destroy later, or perhaps they need it for longer term use.

Now, you could go, every time a new request for a new service comes in, and create a new VPS in whatever your favorite provider is, then add that VPS to your Netbird VPN, and make sure it's all setup with all the right groups and permissions, then add yourself and any of your employees to the list of allowed users, and do all of the other things you need to do just to get the server setup, secure, and ready; or you could use something like Coolify.

This is a service provisioning and management system. This has a ton of 1-click setups for databases, applications, and services. It also has the ability to pull from a git resource, and deploy that way, or to use Docker, and Docker Compose to deploy that way instead.

You can setup multiple servers, and have those servers all be managed by a single Coolify installation (ideal setup), and using secure SSH keys and authentication, Coolify can do the work on your behalf.

I want to show you how we can use this, and how we can manage everything our clients need, as well as a vast majority of the tools we'll be using with Coolify.

Buckle up, this one is a bit long, and will take some work, but it will be 1000% worth it in the end.

Update, Upgrade, and Install Dependency Software

First, we should always update and upgrade our host OS to ensure we have the lasted security updates and any fixes in place.

```
sudo apt update && sudo apt upgrade -y
```

Next, we need to install our dependency software. Since Coolify uses Docker, Docker Compose, and can also run projects directly from a Git repository, we'll want to install Docker, Docker Compose, and Git. While we are at it, let's make sure we have Curl and WGet installed as well as openssh server. OpenSSH will be used by Coolify to connect to remote server resources.

```
sudo apt install git wget curl openssh-server -y
```

After those complete, we'll need to install Docker and Docker Compose. Luckily there is a nice one liner that can get Docker-CE and Docker Compose installed for us, and it should work on almost any Linux distribution.

```
curl -sSL https://get.docker.com/ | sh
```

Let that run, and you should now have all of the necessary software installed.

Unfortunately, Coolify is currently setup to be installed by root. This isn't ideal, and should change in the future, but for now we'll go with it, so we can learn about it and be ready when this changes to allow non-root users with sudo privileges.

If you're not currently logged in as the root user, you can become the root user in most Linux based systems by doing the command:

```
sudo su
```

Enter your sudo password if prompted. Next, we'll run the script to install Coolify.

Technically this script tries to install all of the dependencies, including Docker and Docker Compose as well, but it should run pretty quickly since we've already done most of that.

```
curl -fsSL https://cdn.coollabs.io/coolify/install.sh | bash
```

Allow it to finish, then let's give our server a quick reboot.

DNS is Key

As an MSP, it's important that you can make necessary DNS additions, edits, deletions and so on for your clients. There are literally a dozen ways to do this depending on how you want things to work. The way I currently work in my homelab is I have a wildcard DNS A record that points to my public IP for my reverse proxy. Something like

```
A      *.mysuperdomain.com  -->     210.31.48.134
```

This is a perfectly solid DNS entry to have, but sometimes you have things running in different places, and may not want your proxy manager to handle those items. In those cases, you can just create a specific DNS A record as needed.

Setup our Primary Reverse Proxy

I figured I could proxy through Nginx Proxy Manager to my Coolify system, but I knew it was going to be quite as simple as just adding a new host entry, and it wasn't. That said, after a bit of thought, and some light research, I found a method that works nicely.

But why not just use Coolfiy for all of the apps?

Great question, and honestly, had I known about Coolify when I started my homelab, I may have done that. I didn't, and now have my systems all setup and running, and running well. They take almost no attention at all, and just do what they are supposed to do.

I don't want to have a mass migration effort that I know will 100% frustrate me, and make me want to give up on the whole thing. No, instead I'd rather make the two systems symbiotic. I can then take my time, learn, and move things over slowly... if I want to move them at all.

Back to our Reverse Proxy setup:

I initially thought, ohhh, I can just proxy like I always do, and point my domain to the IP for Coolify containers...but that's not exactly how coolify exposes things...I mean, it can be done as I show with setting the ports options directly in the docker compose files in Coolify, but I don't get the feeling that's how it was intended, and in the end I want to keep Coolify in mind for running services for my clients, so I can give back to Coolify and keep it going.

So, my next thought was, *how do I split my port forward from my firewall*? Right now, port 80 and 443 are forwarded to my NGinX Proxy Manager...but I need some things to go there, and other things to go to my Coolify system. If only there was a way to tell each request for a different

domain or subdomain which machine to go to...then I smacked myself in the forehead, and said "DUH!". This is literally what NGinX Proxy Manager does now. It takes the request for each subdomain and routes it to the right machine and port. I was back where I started...so I kept thinking.

Okay, Let me simplify my thoughts a bit. What if my traffic for <some-sub-domain>.mygreatdomain.com all go to whatever host I need them to, but through NPM as they do today (e.g. nextcloud.mygreatdomain.com, and music.mygreatdomain.com, and media.mygreatdomain.com, and so on).

Then I can create a special subdomain just for the Coolify apps to run on, like coolify.mygreatdomain.com. All of the apps in Coolify will then have a subdomain of <some-sub-domain>.coolify.mygreatdomain.com . This let's me identify those apps quickly in Nginx Proxy Manager, and may give me some freedom in creating the apps I want. Is it necessary? No. Will it help me keep things straight? Yes.

Alternate Method (if you have Coolify Running on a VPS)

If you have opted to run Coolify on a VPS that has it's own public IP, then you can instead just setup the base domain in Coolify, instead of setting a base sub-domain like we are doing above. In other words, in the Coolify Settings >> Configuration >> Instance's Domain, you'd put a sub-domain to reach your Coolify Management UI (something like coolify.mygreatdomain.com).

But, under Servers you can select a server, and in the General section for that Server you'll have a "Wildcard Domain". This is where we want to set the base domain as mygreatdomain.com.

Now, if you are using multiple servers (e.g. with SSH) and managing all their services through Coolify, we'll want to set the "Wildcard Domain" for each of those. We may set them all the same, and use routing to get our clients to their services, or we may set a sub-domain on them that helps us keep them straight.

Let's pretend we have three clients. For each client we manage their domain names:

Client	Domain Name	A-Recrod
MicroTime	microtime.org	*.microtime.org
Burside Accounting	burnaccounts.com	*.burnaccounts.com
Greg and Greg Law Partners	gaglaw.net	*.gaglaw.net

We can setup a server for each client, then setup Coolify to be the management system for all of their services.

We want to set their base domain for each of their own servers. E.g. (Server 1 Wildcard Domain = microtime.org, Server 2 Wildcard Domain = burnaccounts.com, and Server 3 Wildcard Domain = gaglaw.net). This all works out great, and there are more tools in Coolify we can use to separate these even more if needed, but it allows us to deploy applications and get the subdomain for that

app with the clients' own domain.

If the client doesn't have their own domain, we can always create servers for their apps, and use our domain as the base, with their client name as the subdomain base for their server. So for MicroTime we'd set their server to use the Wildcard Domain of "microtime.mygreatdomain.com", and for Burnside Accounting, we would set the Wildcard Domain to be "burnaccounts.mygreatdomain.com", and so on.

We have some flexibility, no matter what we need / want to do here.

Setup our Coolify System

So, first things first, let's setup our Coolify Web Management Interface with a subdomain so we can connect to it securely with https (SSL).

Go to Settings >> Configuratiaon, and set the Instance Domain to your preferred domain / subdomain. I'm going to set mine to "https://coolify.mygreatdomain.com".

Click 'Save', and head back to Servers. We have the option when we first start Coolify to decide if we'll be using it for localhost, or only for remote servers.

Regardless, we can add more servers after our initial setup. Just note that setting localhost will allow the use of local resources to run applications / services, and store their data, and this could create performance issues running Coolify at some point.

HomeLab Setup

If you are doing this for a homelab, you can definitely just setup the Coolify software on the same host you want to run your applications / services. Know that the Coolify application on it's own, at the time of writing this, requires a certain amount of CPU and RAM in order to run properly. That's before you get your other applications installed, so keep that in mind as you add more apps and services on top of it.

The base requirements are:

- 2 CPUs (or vCPUs)
- 2 GB RAM
- 30 GB Storage Space

Again, this is just to run Coolify, and have room for logging, and so on as it runs over time.

In the Coolify setup wizard after you create your initial account, you can select "localhost" as your option if this is how you want to run. Then you'll want to go in and set your Coolify domain (if you want to access it via a domain name), and your localhost base wildcard domain. You do that under servers, then click localhost, then set it in the General tab.

Once you've done that, you can create your first Project. Give it a name, then add a resource to the project. Resources are the Databases, Applicatioins, and services you can use to setup whatever it is you want to run or deploy.

You can opt to use a pass-through service hosted by Coolify. It runs kind of like ngrok as far as I can tell, where it creates a randomized subdomain on your localhost IP, and then adds their domain sslip.io to it. You, in theory should be able to reach that from the internet. This of course depends on Coolify to keep the service running, so not my preference.

Professional Setup for Services

I believe Coolify can absolutely help us setup services and applications for our MSP clients. Particularly when we combine it with things like Netbird, Authentik (hopefully they'll add that soon), and LXConsole (Incus Web UI), and / or Proxmox.

First, let's look at what we have to work with from the series so far.

1. VPN using Netbird. This will be extremely important in keeping our client services secure. We run them inside the VPN, and don't expose them directly to the internet. (You can also accomplish this with a VPS).
2. A way to provision small servers quickly (Proxmox and LXConsole give us great tooling around deploying a powerful, ut small Containerized VM with a few clicks).
    1. We can even create our base template, and save that to make things quicker each time thereafter.
3. Authentik for single sign-on identity and authentication.
4. Zabbix for monitoring all the things we want / need to monitor about these systems. (Could even be a part of our template, if we build it just right possibly).
5. Ansible - we can use this to help keep our servers up to date, and make sure our client services are never outdated.
6. Technitium - a powerful way to help our clients quickly setup access to their services using their own domain.
7. Purelymail - Setting up and managing client email using their domains.?

Given all of the items we have above, Coolify can supplement our build out for setting up, managing, and monitoring services for our clients.

There is a ton of functionality already built into Coolify, and I'm just scratching the surface, so dig in a bit, ask questions, jump over to the Coolfiy social apps and get their input and feedback as well.

If you're interested in a follow up video, make sure to leave a like on this video, and a comment letting me know you'd like to see more of what Coolify can do.

Support My Channel and Content

Support my Channel and ongoing efforts through Patreon:

https://www.patreon.com/awesomeopensource

Buy me a Beer / Coffee:
https://paypal.me/BrianMcGonagill

Server Management Tools

Server Management Tools

Install Termix

https://pt.opensourceisawesome.com/videos/embed/532sScp1rxURUcZKMUoYeo

Install Termix - Cross Platform Terminal Emulator

FIRST AND FOREMOST - in the video, I showed changing the permissions **should not** be set to 777, but instead make sure the 'data' folder owner is $USER:docker with: `chown -R $USER:docker ./data`.

Termix is a terminal emulator built to run as a desktop application, mobile application, and / or in your browser on Windows, MacOS, Linux, Android, and iOS / iPadOS.

It’s got a ton of great features like remembering connections, addition of ssh keys, grouping connections, snippets, and so much more. It’s powerful for a whole host of reasons, two of which are it’s incredible accessibility when you are essentially anywhere with an internet connection, and it’s open source!

Setup a Server

Install Updates

Update and upgrade your server's packages using the following commands:

- For Ubuntu/Debian: `sudo apt update && sudo apt upgrade -y`
- For RedHat/CentOS/Fedora/Alma/Rocky: `sudo dnf update -y`

Create a Non-Root User

Create a non-root user with superuser (sudo) privileges:

1. Add a new user using `adduser <username>`
2. Set the password for this user.
3. Enter the relevant information (optional)
4. Enter 'Y', then press Enter.
5. Add the user to the "sudo" group:
   - For Ubuntu/Debian: `usermod -aG sudo <username>`
   - For RedHat/CentOS/Fedora/Alma/Rocky: `usermod -aG wheel <username>`

Ubuntu / Debian

```
usermod -aG sudo <username>
```

RedHat / CentOS / Fedora / Alma / Rocky

```
usermod -aG wheel <username>
```

Now, you can log out of the system, and log back in as your new non-root super user.

Install Docker and Docker Compose

Install Docker and Docker Compose on your server:

1. Install the `curl` utility:
   - For Ubuntu/Debian: `sudo apt install curl -y`
   - For RedHat/CentOS/Fedora/Alma/Rocky: `sudo dnf install curl -y`
2. Run the command to install Docker and Docker Compose:
   `curl https://get.docker.com | sh`

Add Your User to the Docker Group

Add your non-root user to the docker group so you can use Docker commands without sudo:

```
sudo usermod -aG docker <username>
```

Now log out and back into your system, and you are ready to setup your application.

Install Termix with Docker and Docker Compose

First, let’s create our folder structure. We’ll create a parent ‘docker’ folder, with a ‘termix’ folder inside of it.

```
mkdir -p docker/termix
```

Next, we'll move into our new `termix` folder with

`cd termix`

Now we need to create a configuration file called 'compose.yaml'.

`nano compose.yaml`

Copy the following code block, and paste it into the new file we just created:

```
services:
  termix:
    image: ghcr.io/lukegus/termix:latest
    container_name: termix
    restart: unless-stopped
    ports:
      - "8080:8080"
    volumes:
      - ./termix-data:/app/data
    environment:
      PORT: "8080"
```

If your host machine (where you are running docker) happens to have port 8080 in use for another service already, then you can change the left side of the port mapping to something less common, such as 8022, or any other unused port number.

The port mapping above should then look like:

```
    ports:
      - "8022:8080"
```

Once, you have any changes needed, save the file with CTRL + O, then press Enter to confirm, and exit the nano text editor with CTRL + X.

Next, we'll pull our images, and start our container application with the command:

`docker compose up -d && docker compose logs -f`

The above is really two commands in one, first telling docker compose to pull the images and start the associated container(s), and the second telling docker compose to display the logs for the containers as the system is up and running.

You can stop viewing the logs with CTRL + C.

Once everything is up and running you can navigate to the IP of your host machine on port 8080 (or the port you set on the left side of the port mapping) to see the Web User Interface for Termix.

Here you can create your initial user, which will also be the administrative user of the site. You can add more users, as well as disable registration in the settings. There are also SSO login options for the browser version so you can have your team utilize your SSO IdP if desired.

Setup a Reverse Proxy

First and foremost, you need to own a domain name. I own the domain 'opensourceisawesome.com'. You need to have a domain you own, and can set DNS A and AAAA Recrods for.

You can get a cheap domain at places like NameCheap, or by going to Hover. I use hover, and I like their control panel for managing my domains. If you use my link to sign up, I get a credit, if you don't, then I don't. No big deal, and no pressure, just an option.

Next, you need to have some reverse proxy software setup. As sttated above, I've done videos on setting up NGinX Proxy Manager, and Pangolin, which both have clean user interfaces for setting up your reverse proxy.

Once you have your reverse proxy installed, you can add your subdomain to it.

For instance, you may want to use termix.mygreatdomain.com as your subdomain. In that case you need to have an A record in your domain's DNS settings that points the subdomain 'termix' to the public IP address of your server, or the private network in which your server is setup.

Then, in your reverse proxy, you'll add an entry for 'termix.mygreatdomain.com' and tell it to send the traffic along to the host server and port you setup Termix to run on.

The way this flows is as follows:

A user requests your Termix instance at https://termix.mygreatdomain.com → which asks DNS the IP for the site → which sends you to the public IP address of your host or private network → which forwards you to the reverse proxy → which sends you to the local or private address and port of your Termix instance.

Seems like a lot, but it works, it's reliable, and it's repeatable.

Once complete, you should now be able to get to your Termix instance via it's secure URL, and start accessing your servers and hosts from anywhere you have a browser and internet access.

Support this Channel and Content

Become a Patron at Patreon

Check me out on Patreon

Be me a Coffee or Beer at Paypal

Paypal Support for Awesome Open Source

Get the Awesome Open Source Merch

Great Merch for Open Source Advocates!

Server Management Tools

Install Nexterm

https://pt.opensourceisawesome.com/videos/embed/xhjJgPPWTnAATSHLe5y986

Install Nexterm - Browser based Server Access and Management

Nexterm is an amazing open source project designed to enable secure, powerful server and host management via the web browser. Tools like SSH, SFTP, RDP, VNC, and the use of snippets and code repositories make it a power house for managing your homelab or your business, clients, and cloud infrastructure with ease.

Setup a Server

Install Updates

Update and upgrade your server's packages using the following commands:

- For Ubuntu/Debian: `sudo apt update && sudo apt upgrade -y`
- For RedHat/CentOS/Fedora/Alma/Rocky: `sudo dnf update -y`

Create a Non-Root User

Create a non-root user with superuser (sudo) privileges:

1. Add a new user using `adduser <username>`
2. Set the password for this user.
3. Enter the relevant information (optional)
4. Enter 'Y', then press Enter.
5. Add the user to the "sudo" group:
    - For Ubuntu/Debian: `usermod -aG sudo <username>`

- For RedHat/CentOS/Fedora/Alma/Rocky: `usermod -aG wheel <username>`

Ubuntu / Debian

`usermod -aG sudo <username>`

RedHat / CentOS / Fedora / Alma / Rocky

`usermod -aG wheel <username>`

Now, you can log out of the system, and log back in as your new non-root super user.

Install Docker and Docker Compose

Install Docker and Docker Compose on your server:

1. Install the `curl` utility:
   - For Ubuntu/Debian: `sudo apt install curl -y`
   - For RedHat/CentOS/Fedora/Alma/Rocky: `sudo dnf install curl -y`
2. Run the command to install Docker and Docker Compose:
   `curl https://get.docker.com | sh`

Add Your User to the Docker Group

Add your non-root user to the docker group so you can use Docker commands without sudo:

`sudo usermod -aG docker <username>`

Now log out and back into your system, and you are ready to setup your application.

Install Nexterm with Docker and Docker Compose

First, let’s create our folder structure. We’ll create a parent ‘docker’ folder, with a ‘termix’ folder inside of it.

`mkdir -p docker/nexterm`

Next, we’ll move into our new `nexterm` folder with

`cd nexterm`

Now we need to create a configuration file called ‘compose.yaml’.

`nano compose.yaml`

Copy the following code block, and paste it into the new file we just created:

```
services:
  nexterm:
    environment:
      ENCRYPTION_KEY: "< openssl rnad -hex 32 >"
    ports:
      - "6989:6989"
    restart: unless-stopped
    volumes:
      - ./nexterm:/app/data
    image: germannewsmaker/nexterm:latest
```

If you prefer not to use port 6989 as the port you access Nexterm through, you can change the left side of the port mapping to any port number which is not in use or reserved on your host system. For instance, if you wanted to use port 8022 as the port you access Nexterm through, then you would setup the port mapping as below.

The port mapping above should then look like:

```
    ports:
      - "8022:6989"
```

Once, you have any changes needed, save the file with CTRL + O, then press Enter to confirm, and exit the nano text editor with CTRL + X.

Next, we'll pull our images, and start our container application with the command:

`docker compose up -d && docker compose logs -f`

The above is really two commands in one, first telling docker compose to pull the images and start the associated container(s), and the second telling docker compose to display the logs for the containers as the system is up and running.

You can stop viewing the logs with CTRL + C.

Once everything is up and running you can navigate to the IP of your host machine on port 6989 (or the port you set on the left side of the port mapping) to see the Web User Interface for Nexterm.

Here you can create your initial user, which will also be the administrative user of the site. You can add more users, as well as disable registration in the settings. There are also SSO login options for

the browser version so you can have your team utilize your SSO IdP if desired.

Setup a Reverse Proxy

First and foremost, you need to own a domain name. I own the domain ‘opensourceisawesome.com’. You need to have a domain you own, and can set DNS A and AAAA Recrods for.

You can get a cheap domain at places like NameCheap, or by going to Hover. I use hover, and I like their control panel for managing my domains. If you use my link to sign up, I get a credit, if you don’t, then I don’t. No big deal, and no pressure, just an option.

Next, you need to have some reverse proxy software setup. As stated above, I’ve done videos on setting up NGinX Proxy Manager, and Pangolin, which both have clean user interfaces for setting up your reverse proxy.

Once you have your reverse proxy installed, you can add your subdomain to it.

For instance, you may want to use `nexterm.mygreatdomain.com` as your subdomain. In that case you need to have an A record in your domain’s DNS settings that points the subdomain ‘nexterm’ to the public IP address of your server, or the private network in which your server is setup.

Then, in your reverse proxy, you’ll add an entry for ‘nexterm.mygreatdomain.com’ and tell it to send the traffic along to the host server and port you setup Nexterm to run on.

The way this flows is as follows:

A user requests your Nexterm instance at https://nexterm.mygreatdomain.com → which asks DNS the IP for the site → which sends you to the public IP address of your host or private network → which forwards you to the reverse proxy → which sends you to the local or private address and port of your Nexterm instance.

Seems like a lot, but it works, it’s reliable, and it’s repeatable.

Once complete, you should now be able to get to your Nexterm instance via it’s secure URL, and start accessing your servers and hosts from anywhere you have a browser and internet access.

Support this Channel and Content

Become a Patron at Patreon

Check me out on Patreon

Be me a Coffee or Beer at Paypal

Paypal Support for Awesome Open Source

Get the Awesome Open Source Merch

Great Merch for Open Source Advocates!