# NGinX Proxy Manager


- Securing NGinX Proxy Manager
  - Securing NGinX Proxy Manager
- Install and Setup NGinX Proxy Manager
  - Using NginX Proxy Manager for proper Website Routing

# Securing NGinX Proxy Manager

# Securing NGinX Proxy Manager

https://www.youtube.com/embed/UfCkwlPIozw

I've covered NGinX Proxy Manager, a web GUI for NGinX Web Server in multiple videos.  I essentially use it anytime I want to give a web site a public URL with Ssl encryption (so basically always).

One of the questions I get fairly often is about how to Secure the admin portal of NGinX Proxy Manager itself.  So, in this video I go through the actions with securing the Admin Portal and show you how you can also use the tools in NPM to secure  your various self hosted web sites and web applications with even more than just their respective login screens.

# Installing NGinX Proxy Manager

Again, I've discussed this, and done this in multiple videos, as well as in multiple posts on this site, but let me cover it again here, so you're not jumping around looking for bits and pieces.

To install NPM you need to install docker and docker-compose, and create a new folder on the server you want to run it in.  Next, you'll create two files inside that folder:

- config.json
- docker-compose.yml

Inside the config.json file, you'll put the following:

```
{
  "database": {
    "engine": "mysql",
    "host": "db",
    "name": "npm",
    "user": "<your desired username>",
    "password": "<a strong password>",
    "port": 3306
  }
}
```

And inside the docker-compose.yml file you'll put:

```
version: '3'
services:
  app:
    image: 'jc21/nginx-proxy-manager:latest'
    ports:
      - '80:80'
      - '81:81'
      - '443:443'
    volumes:
      - ./config.json:/app/config/production.json
      - ./data:/data
      - ./letsencrypt:/etc/letsencrypt
  db:
    image: 'jc21/mariadb-aria:10.4'
    environment:
      MYSQL_ROOT_PASSWORD: 'npm'
      MYSQL_DATABASE: 'npm'
      MYSQL_USER: '<username from config.json>'
      MYSQL_PASSWORD: '<strong password from config.json>'
    volumes:
      - ./data/mysql:/var/lib/mysql
```

Make sure to replace the items with < and > around it in each file, and that the username and passwords in each file match.

Now run the command:

`docker-compose up -d`

Give it a minute to pull down everything, and get started, and then in your browser go to the IP address of your server.  You should get a Congratulations screen.

if you go tot he IP address at port 81 (http://192.168.1.x:81), you'll be prompted to login to NPM.

Default credentials are:

username: admin@example.com
passwrod: changeme

Make sure to update the email and password, from the default values, then log out, and back in usign the new values you entered.

Now, you're ready to start proxying traffic.

# Securing NGinX Proxy Manger Admin Console

The simples and most direct way is to secure NPM to itself.  Yep, you just make a loop so that when you ask for a specific URL that you'll have created an A Record for, you get your NGinX Proxy Manager install will proxy the traffic to it's port 81 admin console.

Let's add a new Host entry, and on the dtails page enter the URL you want to use to access the admin console.

In my case, I called it "manage" and created an A Record to point to my public IP address, which is port forwarded 80, 443 to my NPM server.

In my Details tab, I'll enter "manage.example.com" (replace example.com with your domain of course).

Next, enter the IP address of your docker0 interface.  You can find this with either:

```
ifconfig
```

or

```
ip addr show docker0
```

Next, enter 81 for the port number.

Turn on the "Block Common Exploits" option, and Save.

We Save now so we can test it and make sure we are routed properly to our Admin login page.  This is still unencrypted, so don't log in, but make suer you get to the Admin login page by visiting your URL.

If you get to the page successfully, you can go back to NPM via the IP nad Port, and click the 3 dot icon at the right end of your 'manage' row.  Select 'Edit' andn move to the 'SSL' tab.

Choose "Request a New Certificate" from the first dropdown.

Turn on "Force SSL".

Fill in your email address.  LetsEnctrypt uses this to let you know if your Certificates have issues or will expire soon and haven't been renewed.

Select to 'Accept the Terms of Service'.

Click 'Save'.

Now, attempt again, to reach your URL.  You should be routed to an SSL encrypted site, and you can now login to your Admin console.

# What About Keeping the Rest of the Internet Out?

Yep, you just created a publicly accessible URL for your NPM admin console.  Not to fear.  You can still secure it further.  Go to the "Access Lists" tab, and create a new Access List.

Give your new List a name, and then move to the Authorization tab.  Enter as many emails and passwords for users you want to have access to the site.  If you want to restrict to http basic auth, then save, and close your browser (Note: you may have to clear your cache).  Then, re-open, and visit your site, and you should get a prompt for Credentials before you get to the main login screen of NPM itself.

Want more than just basic authentication?

You can also add public IP addresses that will be allowed to access the site.  Edit your Access List, and move to the Access tab.  Enter yoru public IP address, then try to access the site from a machine not on your LAN, and you'll find you won't be able to.

Now, let's say you want to access the site with one or the other, Username and passsword, or Public IP.  Then on the Details tab of your Access List, and enable the option for 'Satisfy Any'.  This makes it so either User or IP will be allowed.  When this is disabled, then you must have both User credentials and be on an allowed Public IP.

# Conclusion

You have amazing open source tools at your fingertips, and making them more secure is highly recommended.  Please use the tools and capabilities to run securely.

# Install and Setup NGinX Proxy Manager

# Using NginX Proxy Manager for proper Website Routing

https://www.youtube.com/embed/RBVcnxTiIL0

## What's NginX?

NginX (pronounced Engine-X) is a web-server, and reverse proxy.  Basically, it can serve up web pages, and can provide a proxy service for incoming web requests.  It all sounds a bit generic, but that's because it is.  NginX can serve web pages, but can also direct requests for Web pages, Web Services, and Web Applications to the right place.  Essentially, it's a hub and router.  It receives a request for any number of web services, and routes the requests to the proper location.

NginX in and of itself, if configured through the terminal in configuration files.  It's not pretty, it's not overly difficult, but you definitely have to have a good feel for all of the options that can be set.

NginX Proxy Manager is a nice Graphical User Interface (GUI) for the user to utilize vs. having to edit and create a bunch of configurations.

## Workflow of NginX Proxy Manager

When you want to route a user to a specific web page or site, NginX is a great tool for the job.  NginX will listen on your server, and once it receives a request, will route the request to the appropriate service, server, page, or application.

### Clarification of the term "Router"

I need to be clear about something here.  NginX is not an application router.  Many applications use routers to move you from page, or portion of a page or application, to another.  These are application level routers.  Also, NginX is not a hardware router, like the one on your home network that routes all of your network traffic to various machines, smart devices, etc.

NginX is a Proxy Router.  It acts as a proxy for the requested web page or site, and forwards that request on to the appropriate site on your server, then returns the response information to the browser.

### Workflow Continued

Requests for websites come into a server on a standard port (generaly 80 or 443). When those requests are received, NginX will parse the request by name, and look through it's configuration files to see if any of them match for the request being made.

If I request `fixitdelrio.com`, NginX will look for a configuration file that tells it what to do with requests for that site.  If it finds a match it will then use the other information in that confiuration to push that request along to the appropriate server or service.

The configuration might tell NginX, "Hey, when you see `fixitdelrio.com`, send it to the IP 10.20.30.40 please."  So, since we asked so nicely, NginX does as requested.

For another site being run on the same server, like `opensourceisawesome.com`, NginX will send the request along to the same IP, but a different port.  While we make the request on the standard port 80, NginX knows that really that site is running on port 24356, and has in it's confguration file to push our request along to 10.20.30.40:24356.  Thus, we ask for `opensourceisawesome.com`, and don't have to know it's running on a special port.

# What about SSL and Encryption?

NginX can also deal with SSL and Encryption, and can be quite helpful with it as well.

If I want to run `lubbocklug.org` on https instead of http, I can use NginX to help me do that.  I can use NginX-Proxy-Manager to tell the request for http://lubbocklug.org to always force the requestor over to https://lubbocklug.org, thus they never go to my site without encryption.

This is huge in today's world of unethical hackers.  Protecting our users is one of the most important steps we can take as self-hosters.

# Okay, I'm tired of the Intro...tell me how to do it.

Information sourced from https://nginxproxymanager.com/setup/

First, you want to install Docker.  Docker CE (Community Edition) is a wonderful tool that uses a very lightweight virtualisation engine to run applications, web sites, and services.  You can hose many containers (a virtualized application server) on a single Docker install.

Instructions for Installing Docker CE will vary from Operating System to Operating System, so it will be better if you Google "How to Install Docker CE" and find the best instructions for your OS.

Once you have Docker installed, you will want to install NginX Proxy Manager.  This part is fairly straight-forward, so let's look at how it's done.

# docker-compose.yml

Next, we need a docker-copose.yml (pronounced yamuhl) file.  This file tells docker what images to pull, what containers to start, what to call them, how they connect to each other if there is more than one, and all kinds of other information.  It's, again, pretty straight-forward, so let's jump into it.

Still in our `nginx_proxy_manager` folder, we now want to create a file called `docker-compose.yml`.  So enter the command

`nano docker-compose.yml`

Copy the code below using CTRL+C (Win, Linux, Unix) or CMD+C (MacOS).

```
version: '3.8'
services:
  app:
    image: 'jc21/nginx-proxy-manager:latest'
    restart: unless-stopped
    ports:
      - '80:80'
      - '81:81'
      - '443:443'
    volumes:
      - ./data:/data
      - ./letsencrypt:/etc/letsencrypt
```

Now paste the text into the blank text editor window.  Use CTRL+Shift+V for Linux, Unix, Win; and usse CMD+V for MacOS.

Again, we need to edit come values in this file.  We want a couple of these values to match the values we changed in our config.json.  Using your arrow keys move down to the section titled `# environment:`.  Under that section you need to either remove the `#` symbol in front of the second line, or remove this entire section (3 lines).

I suggest if you are not going to use IPv6, then make that section look like this.

```
environment:
      # Uncomment this if IPv6 is not enabled on your host
      DISABLE_IPV6: 'true'
```

If you intend to use IPv6, then change `true` to `false`, and ensure it's still surrounded by single quotes '.

Now use your arrow keys to move down to the section titled `db:`.  Below it, we want to chagne three values.

First let's change the value for MYSQL_ROOT_PASSWORD.  We, again, want to make this a strong, but different password from the config file earlier.

After that, change the MYSQL_USER value to match what we entered in our config.json file for "user", and our MYSQL_PASSWORD value to match what we entered in our config.json file for "password".

Once those changes are made, save the file with CTRL+O, then press Enter / Return.  Next press CTRL+X to exit the nano editor.

## Start the NginX Proxy Manager

Finally, we will use our docker-compose.yml file to fetch the docker images, and start our containers (yep, there are 2 conttainers - 1 for NginX Proxy Manager, and 1 for the MySql database for configs).

In the same terminal window, enter the command:

```
docker-compose up -d
```

if your user is not part of the `docker ` group, you may have to use

```
sudo docker-compose up -d
```

Then, enter your sudo password when prompted.

If all goes well, you should be able to browse to your server URL or IP address on port 81 to see the NginX-Proxy-Manager admin portal.

something like `http://opensourceisawesome.com:81` <– of course using your own domain or IP.

If you see the admin portal, congratulations! You've got it setup!  Now NginX is listening on port 80 and port 443 for web-requests.

The next part is setting up various sites for NginX to proxy.

# Proxying Site Traffic with NGinX Proxy Manager

Now that NGinX Proxy Manager is up and running, let's setup a site.  Click on 'Proxy Hosts' on the dashboard.  The card will likely have a 0, and the view will be empty, or should, so we need to add a new host.



Dashboard View

Now click on the 'Add Proxy Host' button on the upper right of the Proxy Hosts view.



You should see a modal (pop-up) window like the one below.



Add Proxy Host Modal Window

Enter the domain name you want NGinX to listen for in the "Domain Name" field. Domain names should be entered wtihout http or https on the front.  so only enter something like `billybobsbassboatsandboots.com` or if you are listening for a subdomain `inventory.billybobsbassboatsandboots.com`

Next, enter the hostname or IP address of the server where the site you entered in the previous step is running.  Finally, enter the port number on which that site is listening / hosting it's traffic.  If you're using Docker to host these sites, then you can see any port mapping using the `docker ps` or `sudo docker ps` command.

Click 'Save'.

If all goes well, the modal (pop-up) window will close, and you should see an entry in your Proxy Hosts view.

Now you can click on the domain name to have it open in a new tab.  If everything is setup properly, you should see your web site.

# But what about SSL?

SSL is absolutely an option, and pretty easy to get setup with NginX Proxy Manager.  You do have to make sure that you've setup your domain to be reached on port 80.  Don't get confused.  The port you entered in the last step above, does not need to be 80, but the Domain Name you entered should not have a port added on the end of it...that's all it means.

Essentially, you need to be able to get to `billybobsbassboatandboots.com` without having to add a specific port number.  So we don't want to have to do `billybobsbassboatandboots.com:11232` or anything.

Once you are sure that your site comes up on port 80, you'll want to click on the three vertical dot icon at the right end of the line with your domain on it.



Click the 3-dot Menu Option

Select 'Edit' from the menu that is shown, and we'll edit our NginX entry.  In the modal (pop-up) window, we want to move to the second tab "Custom Locations".  In this section we just want to re-type the same domain name we entered on the first tab, but in the 'Location' field.

Next, click the drop-down menu under 'Scheme', and select "https".  Now enter the IP or Hostname address of the server we are proxying the traffic to (usually the same as what we entered on the first tab as well.  Finally, enter the port you mapped to 443 in the Port field.

Now we want to move to the third tab, "SSL".  Here you want to click where it says "None", and select "Request a new SSL Certificate".  If you want to force users to always go to the secure version of your site (which you almost always do), turn on the switch next to the "Force SSL" option.

Now enter your email address into the email field, and turn on the option next to "I agree to the LetsEncrypt Terms of Service".



Options to Request a LetsEncrypt SSL Certificate for your site.

Now click "Save".  It may take a minute or so, but if you are returned to the Proxy Hosts view, and no errors are displayed, then your site should now be available on https.

# Conclusion

This process of proxying traffic through a single endpoint, is useful for controlling not only the traffic to and from your home or server, but also for allowing you to run multiple web-sites / hosts on a single server install.

Repeat the above steps for each site you are hosting, and over time you'll get a full list of sites being proxied by NginX.  You can add custom options as you become more familiar with NginX right inside the Edit modal as well (tab 4).

I hope this is helpful to you, and please subscribe to my channel so you'll know when I release new Open Source, Self-hosted videos.