Home Server Applications

Reference information for various server-based software and self-hosted applications.

Network Applications

Reference information for my network related applications

NGINX Proxy Manager

What is Nginx Proxy Manager?

Nginx Proxy Manager is a Docker application that lets you quickly and easily expose your selfhosted services to the outside world. NPM includes Letsencrypt SSL certificate management, which permits you to obtain free SSL certificates for secure hosting of your sites.

Installation

NGINX Proxy Manager (NPM) is installed as a Docker container.

You must have Docker and Docker Compose installed to use NPM. I am currently using Docker CE (community edition).

You also have a choice of databases to use with NPM. The default database installed is SQLite. I chose to utilize MariaDB instead of the default as it is open-source and MySQL compatible but with a richer feature set and better performance than either MySQL or SQLite.

> Please note, that DB_MYSQL_* environment variables will take precedent over DB_SQLITE_* variables. So if you keep the MySQL variables, you will not be able to use SQLite. #

This installation guide is for NPM with MariaDB (MySQL).

Using MariaDB Database with NPM

If you opt for the MariaDB configuration you will have to provide the database server yourself. The current minimum supported version is:

- MariaDB v10.2.7+

It's easy to use another docker container for your database also and link it as part of the docker stack, so that's what the following examples are going to use.

Here is my `docker-compose.yml` using a MariaDB container. You can use it as example :

```
version: '3'
services:
  app:
    image: 'jc21/nginx-proxy-manager:latest'
    ports:
      - '80:80'
      - '81:81'
      - '443:443'
    environment:
      DB_MYSQL_HOST: "db"
      DB_MYSQL_PORT: 3306
      DB_MYSQL_USER: "your MySQL username"
      DB_MYSQL_PASSWORD: "your MySQL password"
      DB_MYSQL_NAME: "nginx"
      TZ: America/New_York
    volumes:
      - /localpathtoyourNPMdata:/data
      - /localpathtoyourNPMletsencryptcertificatedata:/etc/letsencrypt
  db:
    image: 'mariadb'
    environment:
      MYSQL_ROOT_PASSWORD: 'your MySQL root password'
      MYSQL_DATABASE: 'nginx'
      MYSQL_USER: 'your MySQL username'
      MYSQL_PASSWORD: 'your MySQL password'
      TZ: America/New_York
    volumes:
      - /localpathtoyourNPMdatabase:/var/lib/mysql
```

> Make sure you change DB_MYSQL_USER, DB_MYSQL_PASSWORD and MYSQL_ROOT_PASSWORD to whatever username and passwords you intend to use.

>

> Make sure you change the local path of your volumes to the path you intend to use to store NGINX Proxy Manager data, certificates and the database.

> Also, make sure you change the timezone (TZ) parameter to reflect your timezone as it affects the certificate timestamps you get from Let's Encrypt. You can find your timezone from here: Wikipedia TZ Database

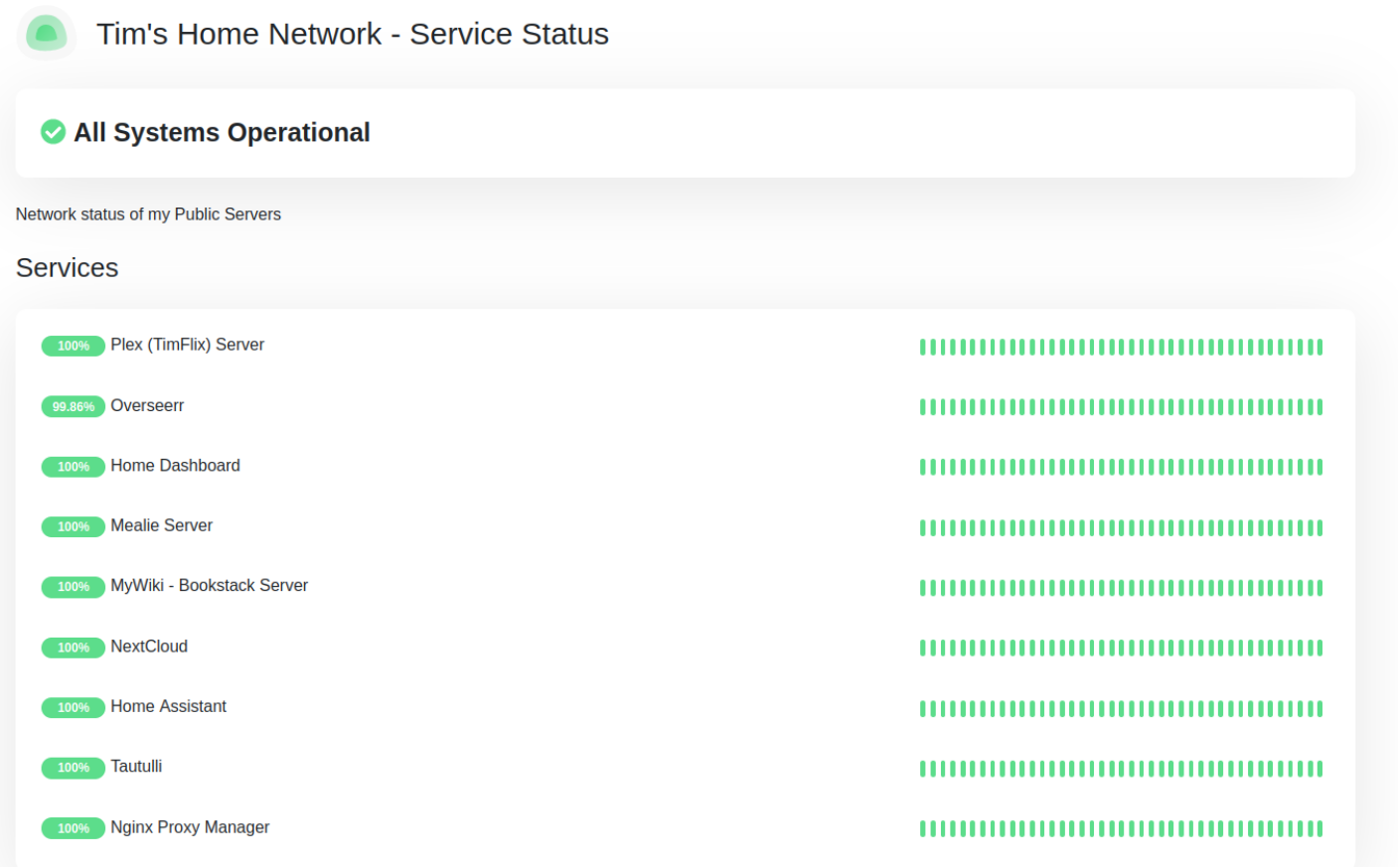
Network Applications

Uptime Kuma

[Uptime Kuma] is a self-hosted, open source, fancy uptime monitoring and alerting system. It can monitor HTTP, HTTP with keyword, TCP, Ping, and DNS systems.

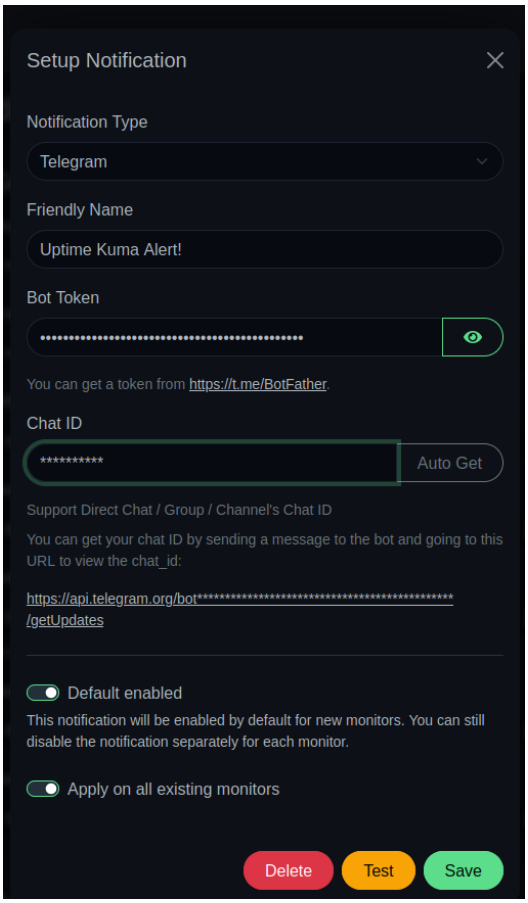
Uptime Kuma is an easy way to know if your systems are up and running. You can even add your favorite Internet sites (i.e., Facebook, Amazon, Twitter, etc.) if you want to be sure they are working. I use Uptime Kuma primarily for my home LAN components and any websites/applications I host from home.

Uptime Kuma even permits me to provide status pages for my users so they can know at a glance if any of my services are down or under maintenance.



In addition, you can setup any number of ways to be notified if any monitored service becomes unavailable. Uptime Kuma supports notifications via email, SMS and more. I currently use Telegram for notifications from Kuma to my phone.

> As Kuma also supports **Apprise** (which supports 50+ notification methods by itself), I will likely move to that notification platform in the future.



Installation

There are both stand-alone and container installation methods for Uptime Kuma. I chose the container method as I am heavily invested in Docker and Docker-Compose on my systems.

To install via Docker, use the following code for the default values.

```
docker run -d --restart=always -p 3001:3001 -v uptime-kuma:/app/data --name uptime-kuma louislam/uptime-kuma:1
```

Otherwise, you can specify your port and data storage (volume) using this docker template.

```
docker run -d --restart=always -p <YOUR_PORT>:3001 -v <YOUR_DIR OR VOLUME>:/app/data --name uptime-kuma louislam/uptime-kuma:1
```

If you use Docker Compose, use the following instructions:

> Shell instructions.

```
mkdir uptime-kuma
cd uptime-kuma
```

```
touch docker-compose.yml
nano docker-compose.yml # copy the contents from the docker-compose.yml example below
mkdir data
ls
docker-compose up -d --force-recreate
```

> Docker compose file contents. This goes in the docker-compose.yml file you created above.

```
---
version: "3.1"

services:
  uptime-kuma:
    image: louislam/uptime-kuma:1
    container_name: uptime-kuma
    volumes:
      - <Uptime Kuma data volume>:/app/data
    ports:
      - <Uptime Kuma port>:3001
    restart: unless-stopped
    security_opt:
      - no-new-privileges:true
```

> Make sure you replace <Uptime Kuma data volume> with the path on your local machine that you want to save your Kuma configuration and data files. For example, /var/lib/docker/volumes/uptime-kuma

> Also make sure you change <Uptime Kuma port> to whatever port you want to use. The default port is 3001. I use the default port because it doesn't conflict with any other software on the system where I have Kuma installed. Your system may be different.

So, as an example, if you wanted to use the path and port as listed above, your docker-compose.yml would look like this:

```
---
version: "3.1"

services:
  uptime-kuma:
```

```
image: louislam/uptime-kuma:1
container_name: uptime-kuma
volumes:
  - /var/lib/docker/volumes/uptime-kuma:/app/data
ports:
  - 3001:3001
restart: unless-stopped
security_opt:
  - no-new-privileges:true
```

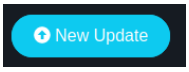
Accessing Uptime Kuma

Once you have started your Uptime Kuma container, you should now be able to access from a browser by going to

http://<your server IP>:<your port>

where **<your server ip>** is the IP address of the server where you installed Uptime Kuma and **<your port>** is the port you chose to use in the docker-compose.yml file. So if your server's IP address is 192.168.1.10 and you chose port 3001, the URL for your browser should look like: http://192.168.1.10:3001

Updating Uptime Kuma

When a new version of Uptime Kuma becomes available, you will see 

appear at the top right of your Uptime Kuma dashboard. Clicking this will take you to the Uptime Kuma Github page where you can see what changes have been made to the new version.

If you installed Uptime Kuma using Docker or Docker-Compose (not stand-alone) you may not yet be able to update your instance until the image is built. This is explicitly stated on the Kuma Update page...

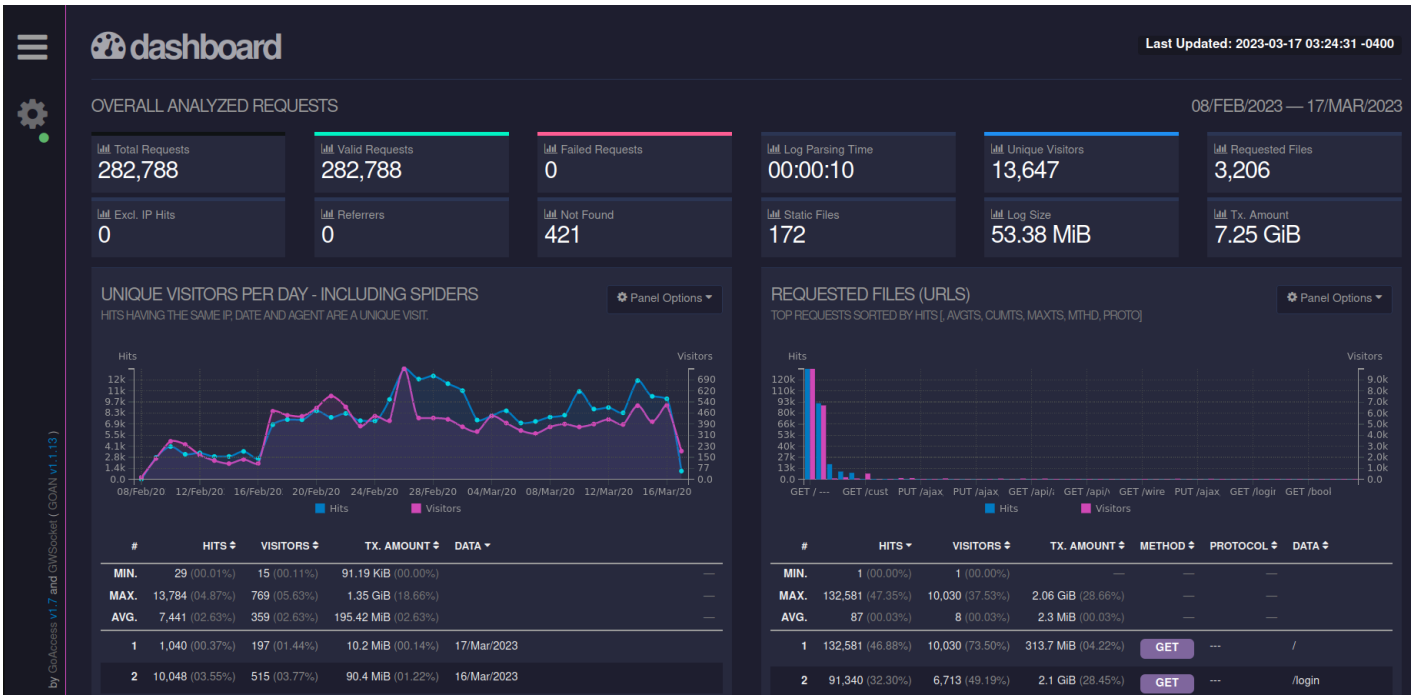
> For every new release, it takes some time to build the docker image, please be patient if it is not available yet

I currently use **Watchtower** to monitor and automatically update all my Docker images. If you do not have some method installed to automate updating your Docker images you may use the excellent instructions provided by the Uptime Kuma author louislam to manually update your Uptime Kuma container.

Network Applications

GoAccess

GoAccess Web Server Statistics



GoAccess is an open source **real-time web log analyzer** and interactive viewer that runs in a **terminal** in *nix systems or through your **browser**.

It provides **fast** and valuable HTTP statistics for system administrators that require a visual server report on the fly.

I use GoAccess to monitor my self-hosted websites traversing my reverse proxy and Cloudflare.

Installation

GoAccess installation methods can be found by going to the **official GoAccess website**

I use a Docker image specifically designed for GoAccess to pull logs from an instance of **Nginx Proxy Manager**. If you have that setup, then this is my recommended Docker Compose file setup.

```
version: '3.3'
services:
    goaccess:
```

```
    image: 'xavierh/goaccess-for-nginxproxymanager:latest'
    container_name: goaccess
    restart: always
    ports:
        - '7880:7880'
    environment:
        - TZ=America/New_York
        - SKIP_ARCHIVED_LOGS=False #optional
        - DEBUG=False #optional
        - BASIC_AUTH=False #optional
        - BASIC_AUTH_USERNAME=user #optional
        - BASIC_AUTH_PASSWORD=pass #optional
        - EXCLUDE_IPS=127.0.0.1 #optional - comma delimited
        - LOG_TYPE=NPM #optional - more information below
    volumes:
    - /path-to-your-nginxproxymanager/logs:/opt/log
    - /path/to/host/custom:/opt/custom #optional, required if using log_type = CUSTOM
```

> Be sure to replace "/path-to-your-nginxproxymanager/logs" with your actual path to your Nginx Proxy Manager logs. This docker-compose.yml also assumes port 7880 is available on your server. If not, change to an available port of your choosing.

Explanations of the "optional" portions of the docker compose file can be found by going to **the Docker image author's Github repository**.

Productivity Applications

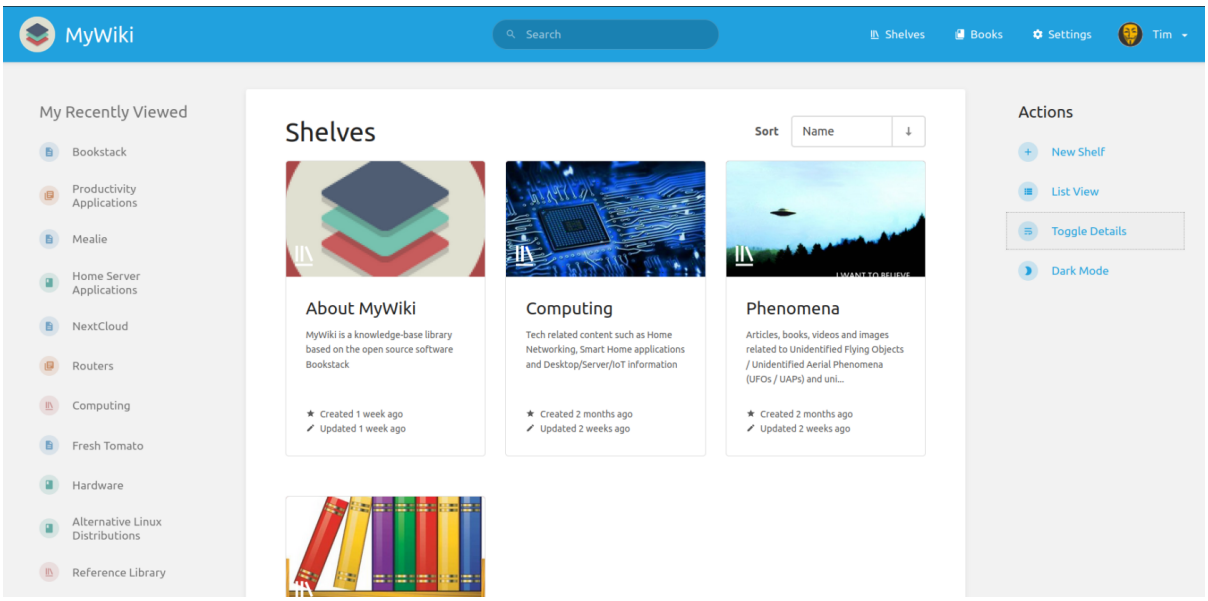
Reference information for client/server productivity applications

Productivity Applications

Bookstack

BookStack is a simple, self-hosted, easy-to-use platform for organising and storing information.

I rebranded my Bookstack instance to **MyWiki** and originally intended to use it only for my personal reference and possibly some journalling. I still use it for referencing but I am building it out as a "knowledge-base library" so others may benefit. If you're here, you probably figured that out already



Installation

To install Bookstack, you will need Docker installed at a minimum and optimally Docker Compose as well. This installation guide assumes you are installing Bookstack on Ubuntu 22.04 and you have both Docker and Docker Compose installed.

If you do not have Docker and Docker Compose installed, you can follow my guides for doing so by clicking these links:

- **How to install Docker**
- **How to install Docker Compose**

> You don't have to install Bookstack using containers. It's just my preference. You can see all the installation methods by going to **Bookstack's official installation web page**.

I am using a pre-built container from **LinuxServer.io**.

You gain access to the relevant container images via a Docker Compose YAML file. You will need to edit various parameters to configure Bookstack and your Bookstack database.

Container images are configured using parameters passed at runtime. These parameters are separated by a colon and indicate `<external>:<internal>` respectively. For example, `-p 8080:80` would expose port `80` from inside the container to be accessible from the host's IP on port `8080` outside the container.

Step 1 - Create a directory to store your docker-compose.yml file.

```
sudo mkdir /docker/bookstack
```

Step 2 - Create and open your docker-compose.yml for editing

```
cd /docker/bookstack
touch docker-compose.yml
nano docker-compose.yml
```

Step 3 - Copy and paste the following Docker Compose file template into your docker-compose.yml file.

```
---
version: "2"
services:
  bookstack:
    image: lscr.io/linuxserver/bookstack
    container_name: bookstack
    environment:
      - PUID=1000
      - PGID=1000
      - APP_URL=
      - DB_HOST=bookstack_db
      - DB_PORT=3306
```

```
      - DB_USER=bookstack
      - DB_PASS=<yourdbpass>
      - DB_DATABASE=bookstackapp
    volumes:
      - /path/to/data:/config
    ports:
      - 6875:80
    restart: unless-stopped
    depends_on:
      - bookstack_db
  bookstack_db:
    image: lscr.io/linuxserver/mariadb
    container_name: bookstack_db
    environment:
      - PUID=1000
      - PGID=1000
      - MYSQL_ROOT_PASSWORD=<yourdbpass>
      - TZ=Europe/London
      - MYSQL_DATABASE=bookstackapp
      - MYSQL_USER=bookstack
      - MYSQL_PASSWORD=<yourdbpass>
    volumes:
      - /path/to/data:/config
    restart: unless-stopped
```

Step 4 - Edit the file, changing the relevant portions of the template as outlined below:

- Change the PUID & GUID if you want another user or group to manage Bookstack. Leave it as it is in the template to use your current username and group.
- The APP_URL variable is for specifying the IP:port or URL your application will be accessed on (ie. `http://192.168.1.1:6875` or `https://bookstack.mydomain.com`). If nothing is specified, the installation computer's IP address will be the default.
- Change the TZ environment variable to reflect your timezone.

> Go **HERE** to find your timezone

- Change DB_USER and MYSQL_USER to whatever you want, but both need to be the same user OR you can leave both as 'bookstack'.
- Change DB_PASS and MYSQL_PASSWORD to a long, strong password. Both variables need to match one another.

- Under the volumes section, change '/path/to/data' to the path of the directory where you will store your Bookstack configuration data.

> Make sure the user and group you specified in your PUID & GUID has read and write permissions to the path and directory you specify.

- You can also change the port you will use to access Bookstack. The default port is 6875. You really only need to change this if that port is already in use on the computer on which you are installing Bookstack.

Once you have your docker-compose.yml configured and saved, you can start your Bookstack instance by typing:

```
docker compose up -d
```

or, if using an older version of Docker Compose:

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

> You must run the command from the directory where you saved your docker-compose.yml file.

Access to Bookstack

From a web browser on the computer where you installed Bookstack, enter the following URL:

http://localhost:6875

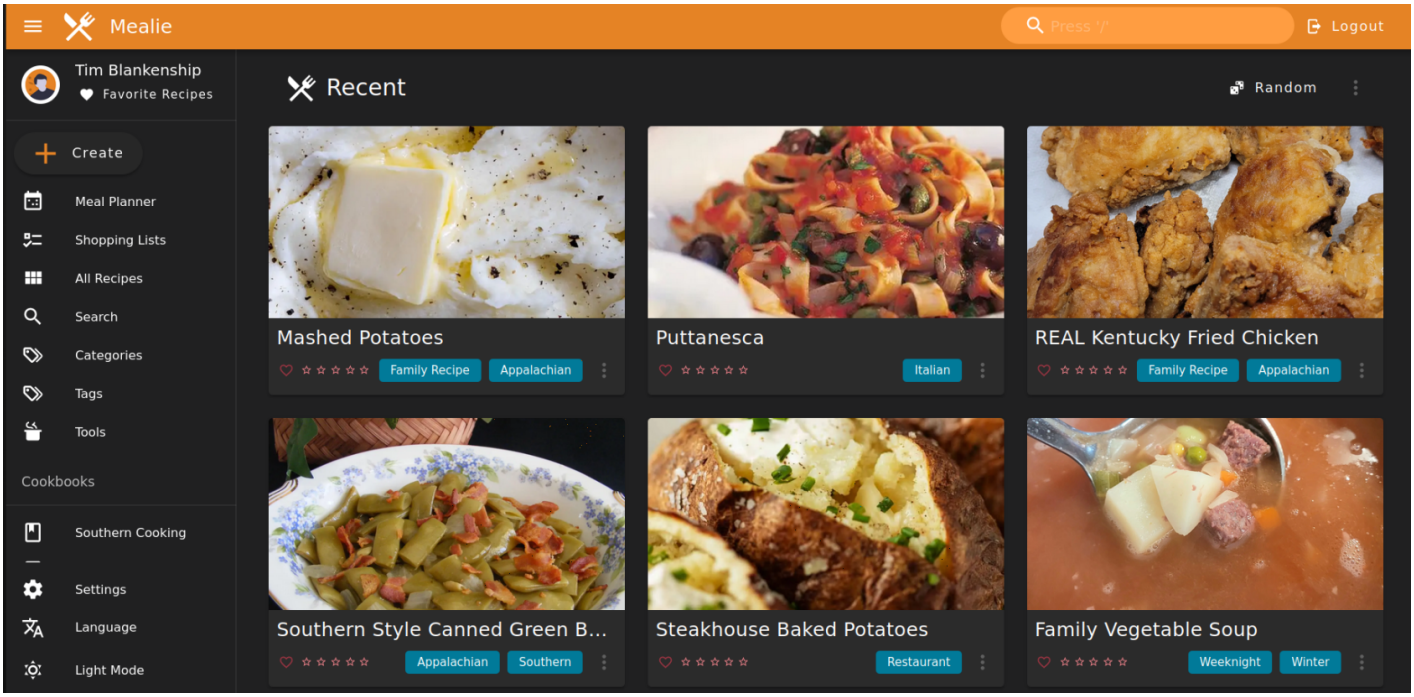
> If you changed the default port from 6875 to something else, you will need to specify it in your URL

If you are on a different computer than the one where you installed bookstack. change *localhost* to reflect the IP address or domain name of the computer where Bookstack is installed.

Productivity Applications

Mealie

Mealie is a self hosted recipe manager and meal planner with a RestAPI backend and a reactive frontend application built in Vue for a pleasant user experience for the whole family. Easily add recipes into your database by providing the url and Mealie will automatically import the relevant data or add a family recipe with the UI editor. Mealie also provides an API for interactions from 3rd party applications.



Key Features

- □□ Fuzzy search
- □□ Tag recipes with categories or tags to flexible sorting
- □□ Import recipes from around the web by URL
- □□ Progressive Web App
- □□ Create Meal Plans
- □□ Generate shopping lists
- □□ Easy setup with Docker
- □□ Customize your interface with color themes layouts
- □□ Export all your data in any format with Jinja2 Templates, with easy data restoration from the user interface.
- □□ localized in many languages

- Plus tons more!
  - Flexible API
    - Custom key/value pairs for recipes
    - Webhook support
    - Interactive API Documentation
  - Raw JSON Recipe Editor
  - Migration from other platforms
    - Chowdown
    - Nextcloud Cookbook
  - Random meal plan generation

Productivity Applications

NextCloud

Nextcloud is the industry-leading, fully open-source, on-premises (self-hosted) content collaboration platform. Teams access, share and edit their documents, chat and participate in video calls and manage their mail and calendar and projects across mobile, desktop and web interfaces.

Hub integrates the four key Nextcloud products Files, Talk, Groupware and Office into a single platform, optimizing the flow of collaboration. Eliminate the confusing hodgepodge of different SaaS tools and the compliance, security, cost and productivity issues that come with it and standardize on a single solution with Nextcloud Hub.

Productivity Applications

Vaultwarden

Vaultwarden is an alternative self-hosted implementation of Bitwarden. It is compatible with all upstream Bitwarden clients

Bitwarden is an open-source password management service that stores sensitive information such as website credentials in an encrypted vault. The platform offers a variety of client applications including a web interface, desktop applications, browser extensions, mobile apps, and a command-line interface.

Installation

Pull the docker image and mount a volume from the host for persistent storage:

```
docker pull vaultwarden/server:latest
docker run -d --name vaultwarden -v /vw-data/:/data/ -p 80:80 vaultwarden/server:latest
```

Change `/vw-data/` to the path where you will store your vaultwarden data locally.

Change the port if you are already using port 80 for another service on the installation computer. For example, from `80:80` to `8181:80`

If you are installing on a Linux server, you can verify what ports are in use by using this command

```
sudo ss -ltn
```

> It is highly recommended to secure your Vaultwarden server using a TLS certificate. If you have an available domain name, you can get HTTPS certificates with Let's Encrypt, or you can generate self-signed certificates with utilities like mkcert. Some proxies automate getting certificates, like **NGINX Proxy Manager**.

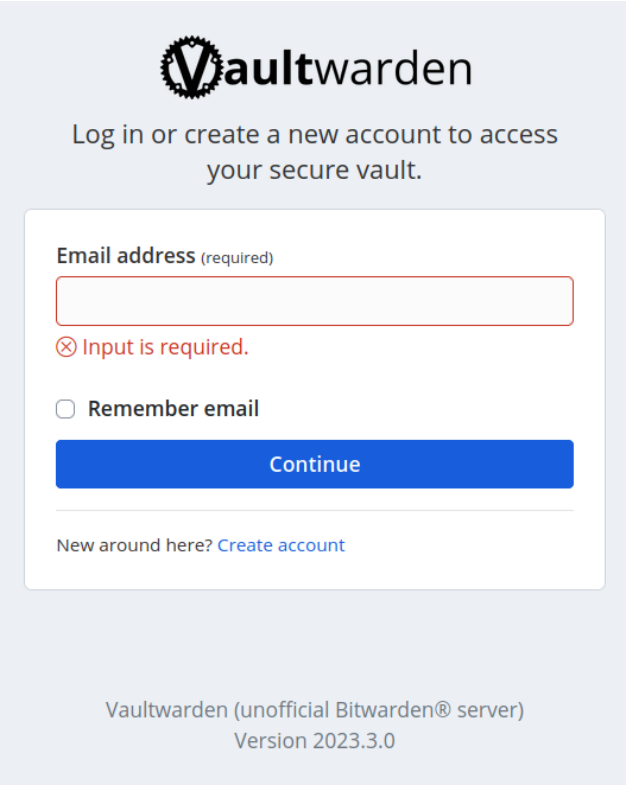
Accessing Vaultwarden

Browser

To access your Vaultwarden server from a web browser, locally

`http://localhost:port` where 'localhost' is the IP address of your Vaultwarden server and 'port' is the port number you defined in your docker container if it is any port other than port 80.

You should see a screen similar to this:



You will need to create an account to use it. Just click the 'Create account' link on the page.

> It is also worth noting that you can export your current Bitwarden data (if you already had a Bitwarden account) directly into Vaultwarden by using any Bitwarden client's 'Export Vault' function while logged with your Bitwarden account. You can export into .json or csv formats. You then log out of your Bitwarden account and and login using your Vaultwarden account on that same client and import your data using the 'Import Vault' function. Your data will then automatically sync to any other Bitwarden client where you have your Vaultwarden account set up.

Clients

You can access you Vaultwarden server using any of the clients supported by Bitwarden. To set your Bitwarden clients to point to your Vaultwarden server, follow these instructions:

Browser Extension or Mobile App

To connect a browser extension or mobile app to your Vaultwarden server:

1. Log out of your Bitwarden browser extension or mobile app.
2. On the login screen, select the ⚙ **Settings** icon.
3. In the **Server URL** field, enter the domain name for your server with `https://` (for example, `https://my.bitwarden.domain.com`).
4. Select **Save**.

> You must have your Vaultwarden server setup with a domain name and a TLS or self-signed certificate for this to work

Desktop Client

Each account that's logged in to your desktop app can be connected to a different server. For example, you can have one account that connects to a Bitwarden server in the cloud and another account that connects to a private Vaultwarden server.

To connect your desktop client to point to your Vaultwarden server:

1. Log out of your account in your Bitwarden desktop app **OR** select **+ Add Account**.
2. On the login screen, select the ⚙ **Settings** icon.
3. In the **Server URL** field, enter the domain name for your server with `https://` (for example, `https://my.bitwarden.domain.com`).
4. Select **Save**.

Container Applications

Containerization is a form of virtualization where applications run in isolated user spaces, called containers, while using the same shared operating system (OS). One of the benefits of containerization is that a container is essentially a fully packaged and portable computing environment.

A container is standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

This chapter details the container management software I currently use to manage containerized applications.

Container Applications

Docker

Docker is an application that simplifies the process of managing application processes in *containers*. Containers let you run your applications in resource-isolated processes. They're similar to virtual machines, but containers are more portable, more resource-friendly, and more dependent on the host operating system.

Installation

In this guide, you will install Docker Community Edition (CE) on Ubuntu 22.04.

To follow this tutorial, you will need the following:

- One Ubuntu 22.04 server, including a `sudo` non-**root** user and a firewall.
- An account on **Docker Hub** if you wish to create your own images and push them to Docker Hub.

The Docker installation package available in the official Ubuntu repository may not be the latest version. To ensure we get the latest version, we'll install Docker from the official Docker repository. To do that, we'll add a new package source, add the GPG key from Docker to ensure the downloads are valid, and then install the package.

First, update your existing list of packages:

```
sudo apt update
```

Next, install a few prerequisite packages which let `apt` use packages over HTTPS:

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

Then add the GPG key for the official Docker repository to your system:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg
```

Add the Docker repository to APT sources:

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-
keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null
```

Update your existing list of packages again for the addition to be recognized:

```
sudo apt update
```

Make sure you are about to install from the Docker repo instead of the default Ubuntu repo:

```
apt-cache policy docker-ce
```

You'll see output like this, although the version number for Docker may be different:

Output of apt-cache policy docker-ce

```
docker-ce:
  Installed: (none)
  Candidate: 5:20.10.14~3-0~ubuntu-jammy
  Version table:
     5:20.10.14~3-0~ubuntu-jammy 500
        500 https://download.docker.com/linux/ubuntu jammy/stable amd64 Packages
     5:20.10.13~3-0~ubuntu-jammy 500
        500 https://download.docker.com/linux/ubuntu jammy/stable amd64 Packages
```

Notice that `docker-ce` is not installed, but the candidate for installation is from the Docker repository for Ubuntu 22.04 (`jammy`).

Finally, install Docker:

```
sudo apt install docker-ce
```

Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it's running:

```
sudo systemctl status docker
```

The output should be similar to the following, showing that the service is active and running:

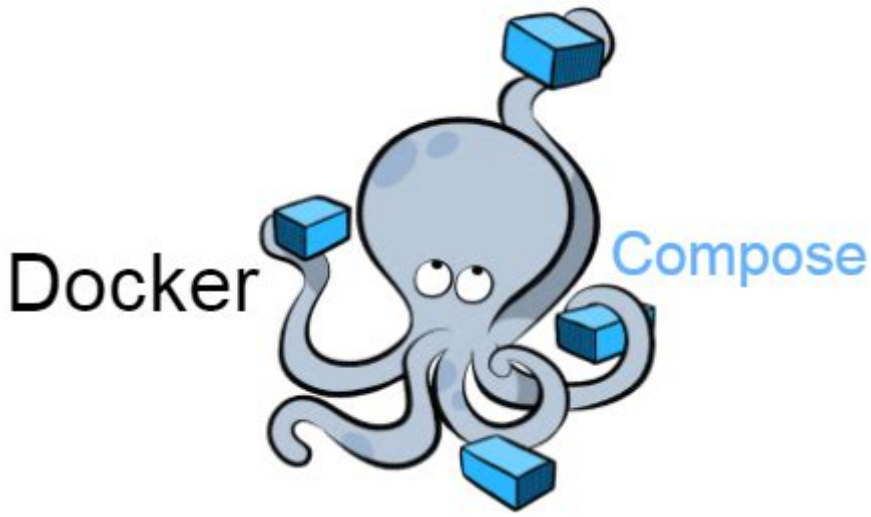
Output

```
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Fri 2022-04-01 21:30:25 UTC; 22s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 7854 (dockerd)
      Tasks: 7
     Memory: 38.3M
        CPU: 340ms
     CGroup: /system.slice/docker.service
             └─7854 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

Installing Docker now gives you not just the Docker service (daemon) but also the `docker` command line utility, or the Docker client.

Container Applications

Docker Compose



Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

I heavily utilize Docker Compose in my environment. I find it much easier to deploy Docker containers and, of greater importance, to document, update and make changes to my production containers and stacks.

I also use **Portainer** for some container management. However, I use it primarily for restarting or stopping containers and for development and testing purposes.

Installation

This guide assumes you are installing Docker Compose on Ubuntu 22.04 LTS.

To make sure you obtain the most updated stable version of Docker Compose, you'll download this software from its **official Github repository**.

You can confirm the latest version available in their **releases page**. The latest release at the time of this writing is version 2.16.0

> **Note:** Starting with Docker Compose v2, Docker has migrated towards using the `compose` CLI plugin command, and away from the original `docker-compose`. The actual usage involves dropping the hyphen from `docker-compose` calls to become `docker compose`

1. Use the following command to download v2.16.0:

```
mkdir -p ~/.docker/cli-plugins/
curl -SL https://github.com/docker/compose/releases/download/v2.16.0/docker-compose-linux-x86_64 -o ~/.docker/cli-plugins/docker-compose
```

> If the version has changed since this guide was written, simply replace "2.16.0" in the command above with the new version number.

2. Set the correct permissions so that the `docker compose` command is executable:

```
chmod +x ~/.docker/cli-plugins/docker-compose
```

3. Verify that the installation was successful by running the following:

```
docker compose version
```

The terminal should return the version of Docker Compose you selected.

Portainer

Portainer Community Edition is a lightweight service delivery platform for containerized applications that can be used to manage Docker, Swarm, Kubernetes and ACI environments. It is designed to be as simple to deploy as it is to use. The application allows you to manage all your orchestrator resources (containers, images, volumes, networks and more) through a 'smart' GUI and/or an extensive API.

Portainer consists of a single container that can run on any cluster. It can be deployed as a Linux container or a Windows native container.

Introduction

Portainer consists of two elements, the *Portainer Server*, and the *Portainer Agent*. Both elements run as lightweight Docker containers on a Docker engine. This document will help you install the Portainer Server container on your Linux environment. To add a new Linux environment to an existing Portainer Server installation, please refer to the **Portainer Agent Installation** section of this guide.

To get started, you will need:

- The latest version of Docker installed and working
- sudo access on the machine that will host your Portainer Server instance
- By default, Portainer Server will expose the UI over port `9443` and expose a TCP tunnel server over port `8000`.
  - The latter is optional and is only required if you plan to use the Edge compute features with l

Deployment

First, create the volume that Portainer Server will use to store its database:

```
docker volume create portainer_data
```

Then, download and install the Portainer Server container:

```
docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always -v
/var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-
ce:latest
```

> By default, Portainer generates and uses a self-signed SSL certificate to secure port `9443`. Alternatively you can provide your own SSL certificate during installation or via the Portainer UI after installation.

> If you require HTTP port `9000` open for legacy reasons, the following to your `docker run` command: **add -p 9000:9000**

Portainer Server has now been installed. You can check to see whether the Portainer Server contain

```
docker ps
```

If all is well, you should see container is Up

```
root:~# docker ps
CONTAINER ID   IMAGE                           COMMAND                  CREATED
STATUS
PORTS
NAMES
de5b28eb2fa9   portainer/portainer-ce:latest   "/portainer"             2 weeks ago   Up 9
days   0.0.0.0:8000->8000/tcp, :::8000->8000/tcp, 0.0.0.0:9443->9443/tcp, :::9443->9443/tcp
portainer
```

Logging In

Now that the installation is complete, you can log into your Portainer Server instance by opening a w

```
https://localhost:9443
```

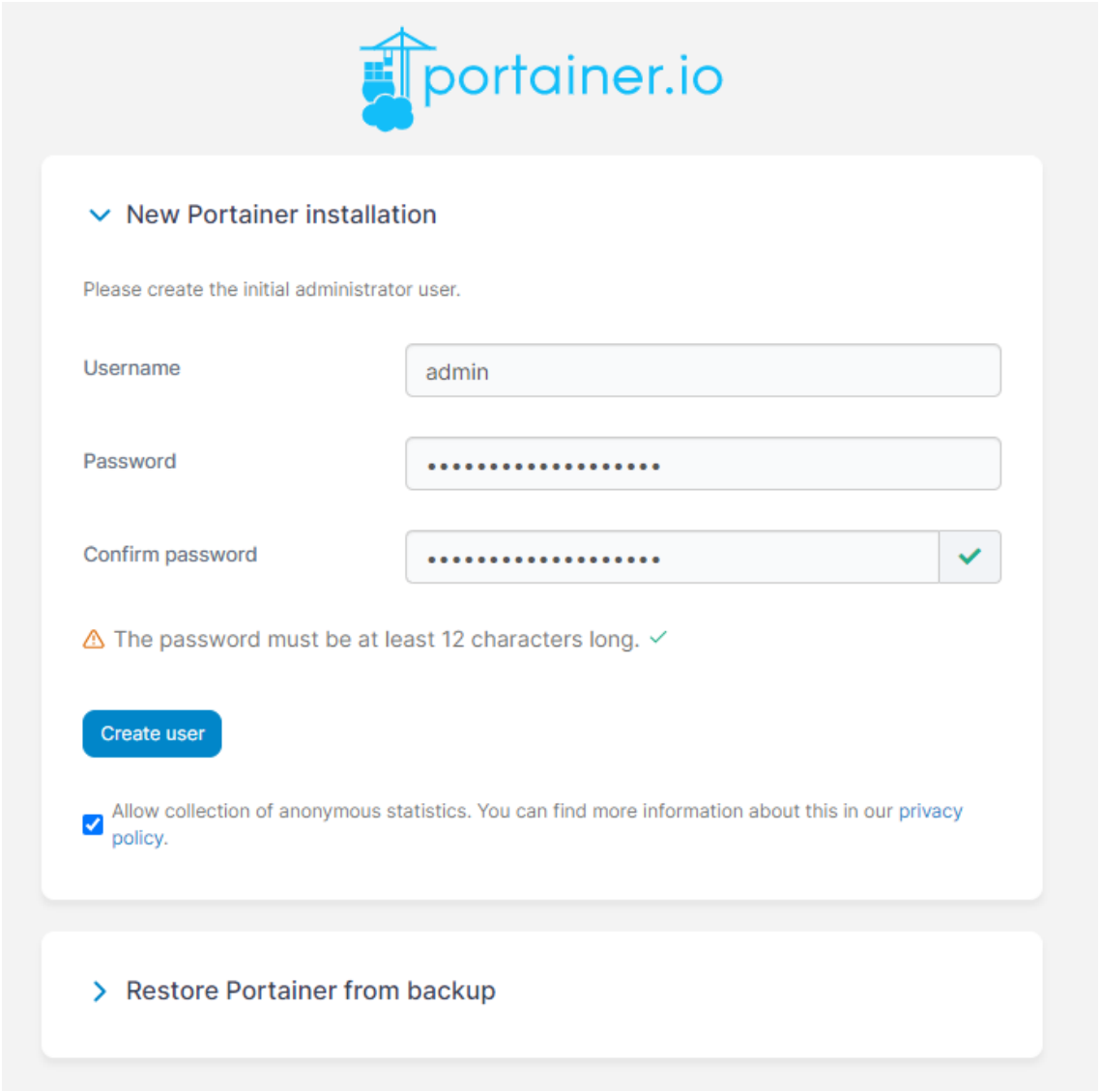
Replace `localhost` with the relevant IP address or FQDN if needed, and adjust the port if you changed it earlier.

You will be presented with the initial setup page for Portainer Server.

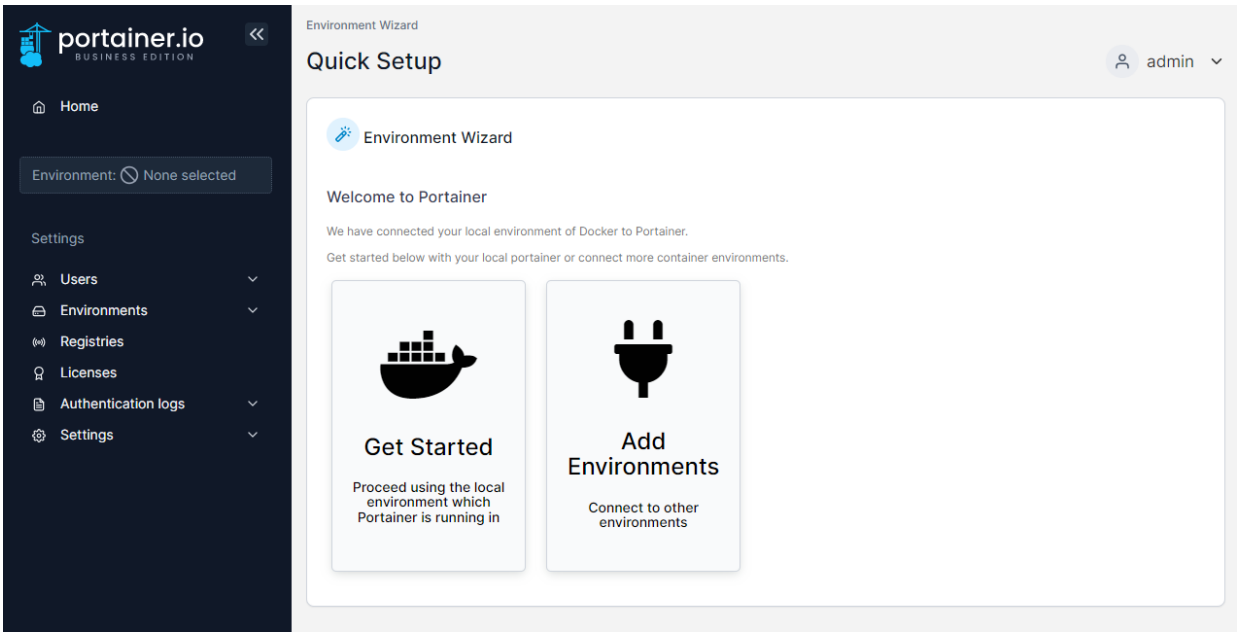
Initial Setup

Your first user will be an administrator. The username defaults to `admin` but you can change it if you prefer.

The password must be at least 12 characters long and meet the listed password requirements.



Once the admin user has been created, the **Environment Wizard** will automatically launch.



The installation process automatically detects your local environment and sets it up for you.
If you want to add additional environments to manage with this Portainer instance, click Add Enviror

Otherwise, click Get Started to start using Portainer!

Portainer Agent Installation

Portainer uses the *Portainer Agent* container to communicate with the *Portainer Server* instance and provide access to the node's resources.
On each computer that is running Docker containers that you want to manage, you will need to install the Portainer agent by executing the following:

```
docker run -d -p 9001:9001 --name portainer_agent --restart=always -v
/var/run/docker.sock:/var/run/docker.sock -v /var/lib/docker/volumes:/var/lib/docker/volumes
portainer/agent:latest
```

Once the agent has been installed you are ready to add the environment to your Portainer Server installation.

Container Applications

Watchtower

Watchtower is an application that will monitor your running Docker containers and watch for changes to the images that those containers were originally started from. If watchtower detects that an image has changed, it will automatically restart the container using the new image.

With watchtower you can update the running version of your containerized app simply by pushing a new image to the Docker Hub or your own image registry. Watchtower will pull down your new image, gracefully shut down your existing container and restart it with the same options that were used when it was deployed initially.

Installing Watchtower

I am using Docker Compose to run my Watchtower instances.

> You need to run an instance of Watchtower on each server where you run Docker containers.

Follow these steps to get Watchtower up and running:

- Make a directory for your Watchtower project and then navigate into it:

```
mkdir ~/watchtower
cd ~/watchtower
```

- Create a new YAML file named `docker-compose.yml` using `nano` or your preferred text editor:

```
nano docker-compose.yml
```

- Insert the following into `docker-compose.yml`:

```
version: "3"
services:
  watchtower:
    container_name: watchtower
    image: containrrr/watchtower
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    restart: unless-stopped
    environment:
      - TZ=America/New_York
      - WATCHTOWER_LIFECYCLE_HOOKS=1 # Enable pre/post-update scripts
    command: --debug true --cleanup true dockerimage1 dockerimage2 dockerimage3
```

Where "dockerimage" 1, 2, and 3 are the names of the docker images I want to monitor and update when a change occurs to the original image.

> Make sure to put a "space" between the names of the images you want to monitor

- Save and exit your file. If you used `nano`, you can do this by pressing `CTRL+O`, `ENTER`, then `CTRL+X`. Now you can start your containers using `docker compose up`. Add the `-d` flag to prevent Docker from taking over your terminal:

```
docker compose up -d
```

> Passing a list of containers to monitor, which does not include the watchtower container, will disable the monitoring of watchtower. By adding it to the argument list, it will start automatically updating itself.