

Using Portainer with Docker and Docker Compose

10 minute read Updated: April 17, 2023



James Walker

We're [Earthly](#). We make building software simpler and therefore faster. If you're interested in a [simple way to build containers](#) then [check us out](#).

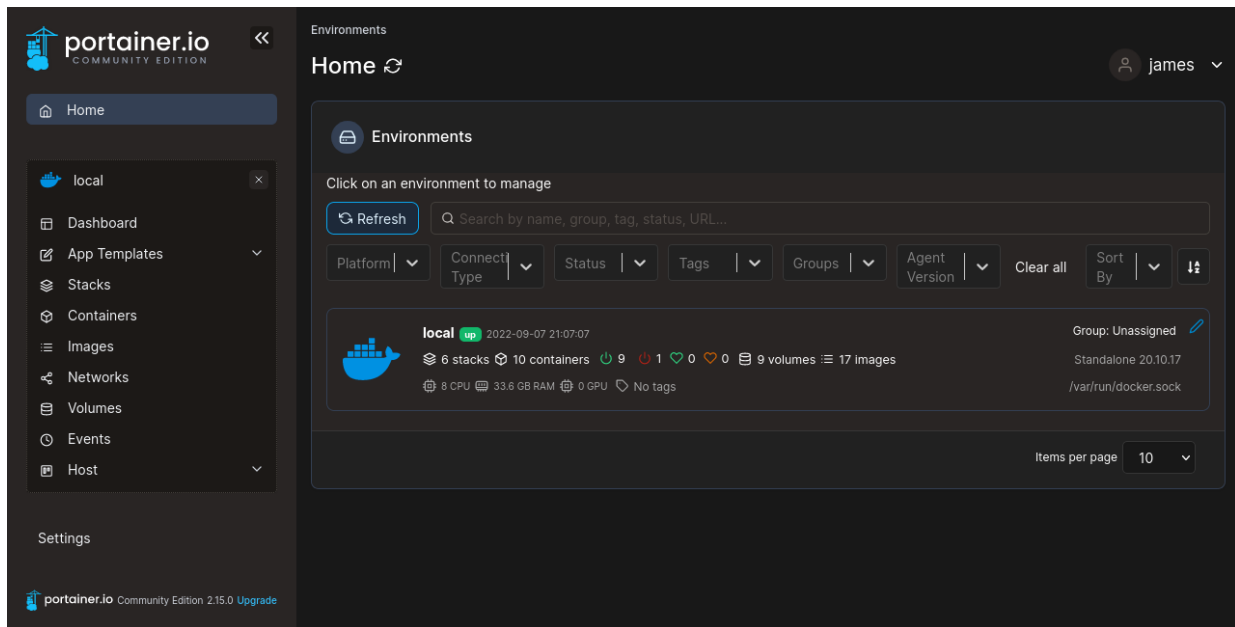
Docker's CLI and API are powerful tools, but they can be unwieldy when you're working with large container fleets or looking for a more visual experience. [Portainer](#), a web-based Docker management system that provides a convenient graphical user interface (GUI), lets you take charge of your containers, images, volumes, and other resources, without memorizing long terminal commands.

Portainer can be used to monitor your Docker installation, interact with containerized apps, and deploy new stacks with minimal effort. A single Portainer instance can connect to multiple Docker hosts, centralizing your container management around one application. It also supports other container environments beyond [Docker](#), including [Kubernetes](#) clusters and [Azure Container Instances](#).

This article will show you how to set up and start using Portainer. You'll also learn the benefits of some of Portainer's headline features, such as how to deploy apps with built-in templates and your own Compose files.

What Is Portainer?

[Portainer](#) is a [container](#) management interface. It started out as a GUI for Docker but has expanded to support several other container environments. It has more than 1 million users and over [22,000 GitHub stars](#). Two versions are available: the free and open source Community Edition (CE) and a paid Enterprise Edition (EE).



Portainer dashboard

You can use Portainer whenever you want to interact with your containers from a graphical interface. CLI commands and API endpoints are often handy in development but less ideal for managing production applications. With Portainer, you can easily monitor multiple endpoints and allow team members to access a shared deployment environment.

Implementing Portainer

Portainer is usually deployed in its own container. This article assumes you're using Docker, but you can also [run Portainer directly in Kubernetes](#) by deploying with the official [Helm chart](#).

Here's an overview of the steps required to get Portainer running:

- Install Docker
- Create a new container that runs Portainer
- Log into the Portainer UI to set up your initial user account
- Use Portainer or the Docker CLI to manage your Docker environment

The following sections will detail each of these steps in turn.

Installing Docker

Before you go any further in this tutorial, you'll need to install Docker. If you're using Windows or Mac, download, and run [the latest version of the Docker Desktop](#) installer. Linux users can try [the experimental version](#) of Desktop for Linux or use the following steps to install [Docker Engine](#).

Docker Engine is distributed in the package repositories of all major Linux distributions. It's also available as a direct download in DEB or RPM format. You can obtain detailed instructions for each method and platform [from the official Docker documentation](#). The following steps assume you're installing from the repository on a Debian-based system.

To begin, install the dependencies required by running the following commands:

```
$ sudo apt-get update
$ sudo apt-get install ca-certificates curl gnupg lsb-release
```

Next, add the GPG key used to sign the Docker repository:

```
$ sudo mkdir -p /etc/apt/keyrings
$ curl -fsSL https://download.docker.com/linux/debian/gpg | \
sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

This lets the **apt** package manager verify the source of your download. Now add the repository to your package list with the following command:

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

The interpolated commands allow automatic selection of the correct list for your system.

Docker can now be installed with the following command:

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

The **docker** CLI requires root privileges by default. You can avoid prefixing commands with **sudo** by adding yourself to the **docker** group:

```
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
```

Log out and log back in to apply the change.

Finally, test your installation by starting a container with the Hello World image:

```
$ docker run hello-world

Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:7d246653d0511db2a6b2e0436cfd0e52ac8c066000264b3ce63331ac66dca625
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
...
```

Installing Docker Compose

Although not required to use Portainer, **Docker Compose** is a popular utility that makes it easier to manage containers in your terminal. Compose will be used in the next step to start Portainer.

Docker Compose used to be an independent binary but has now been integrated into Docker as a plugin. It's included with Docker Desktop and can be added to the Docker Engine installation configured earlier by running the following command:

```
$ sudo apt-get install docker-compose-plugin
```

You should now be able to use **docker compose** in your terminal:

```
$ docker compose version
Docker Compose version v2.6.0
```

Deploying Portainer

Portainer has a few dependencies that must be supplied when you start your container:

- It requires a volume to store persistent data.

- Your host's Docker socket should be mounted into the container so that Portainer can access and interact with the Docker daemon.
- You need to bind a port to the container so you can access the web UI.

This requires several flags to be used when you start Portainer with **docker run** :

```
$ docker run -d \  
-p 9443:9443 \  
--name portainer \  
--restart unless-stopped \  
-v data:/data \  
-v /var/run/docker.sock:/var/run/docker.sock \  
portainer/portainer-ce:latest
```

A better way to start Portainer is to use Docker Compose. This lets you write the container's configuration into a file so you can bring up the app with a single command. To do so, save the following file as **docker-compose.yml** in your working directory:

```
version: "3" \  
services: \  
  portainer: \  
    image: portainer/portainer-ce:latest \  
    ports: \  
      - 9443:9443 \  
    volumes: \  
      - data:/data \  
      - /var/run/docker.sock:/var/run/docker.sock \  
    restart: unless-stopped \  
volumes: \  
  data:
```

This encapsulates all the flags given to the **docker run** command in the previous example.

Here, the **image** field is set to **portainer/portainer-ce:latest** to use the latest Portainer CE release from Docker Hub. Change this to **portainer/portainer-ee:latest** if you've purchased an Enterprise Edition license.

The **ports** field sets up a port binding from your host to the container. You'll be able to access the Portainer UI by visiting **https://localhost:9443** . Portainer provides a self-signed HTTPS certificate, which you can override by mounting your own into the container.

The **volumes** field sets up a **data** volume that's mounted to **/data** inside the container. Portainer will write your settings to this location, allowing them to persist after the container

restarts. The host's Docker socket, `/var/run/docker.sock`, is **bind mounted** straight into the container so Portainer can manage the Docker installation it's running within.

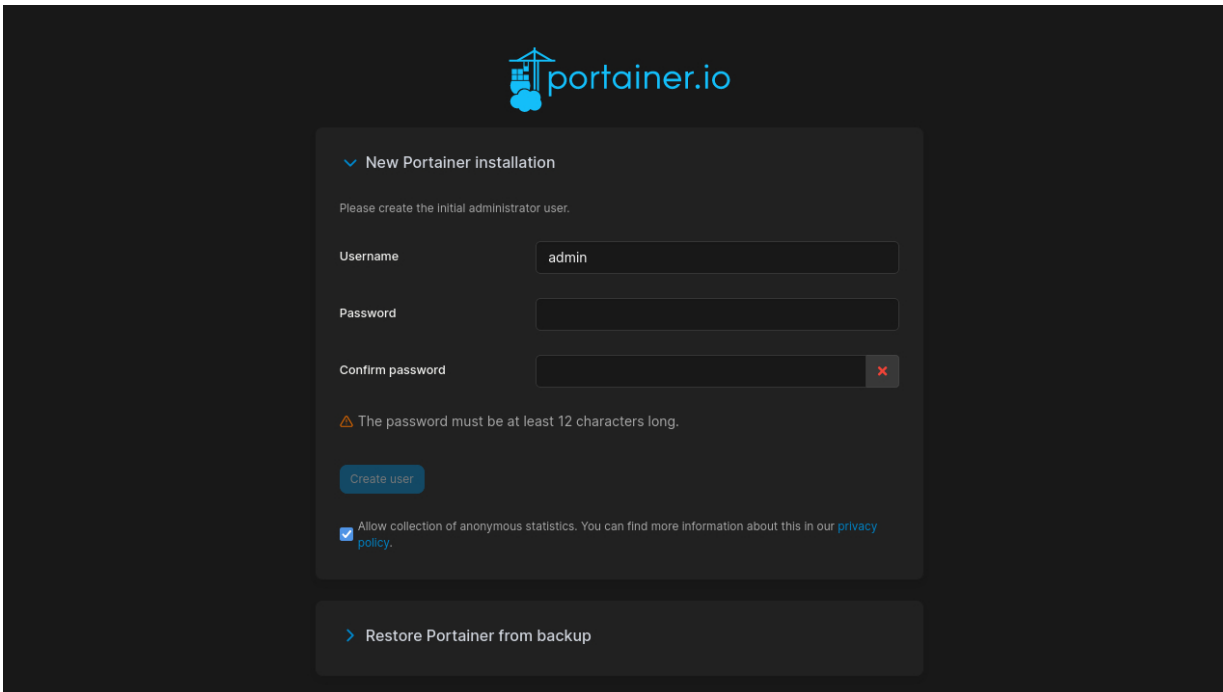
Finally, the `restart` field is set to `unless-stopped`, so Docker automatically starts Portainer after the host reboots unless you manually stop the container first.

Now you can use this Compose file to bring up Portainer:

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

Next, head to <https://localhost:9443> in your browser. You'll see a security prompt if you're using Portainer's built-in SSL certificate. This configuration shouldn't be used in production or when Portainer is exposed on a public network, but this is safe for local use.

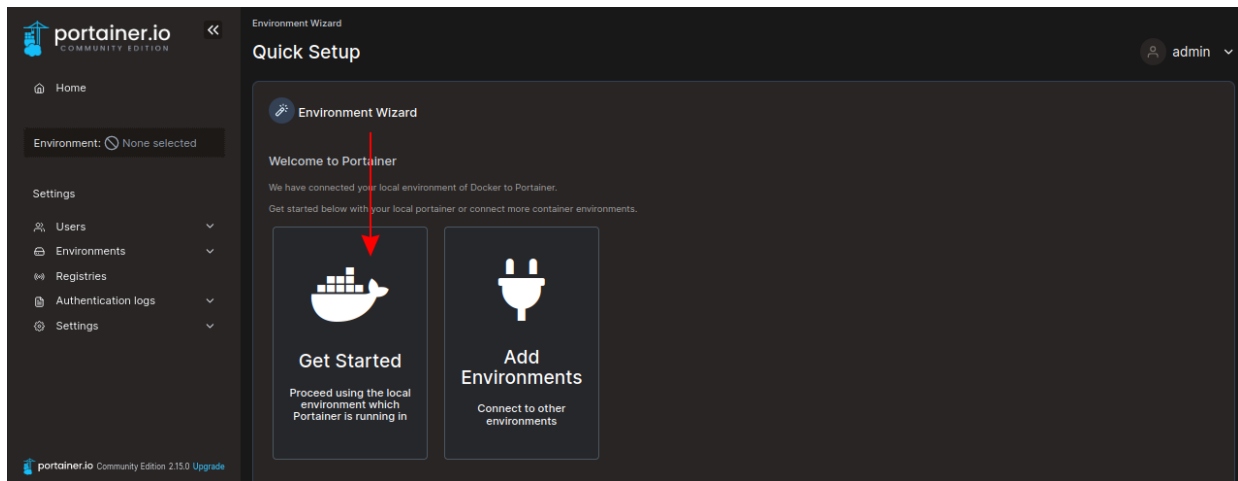
Once you've acknowledged the prompt, you'll get to Portainer's first run screen. Create your initial user account by entering a username and password and pressing **Create user**:



The screenshot shows the Portainer.io logo at the top. Below it, the heading "New Portainer installation" is followed by the instruction "Please create the initial administrator user." The form contains three input fields: "Username" with the value "admin", "Password", and "Confirm password" with a red "x" icon on the right. A warning triangle icon is next to the text "The password must be at least 12 characters long." Below the form is a blue "Create User" button. At the bottom of the form, there is a checked checkbox for "Allow collection of anonymous statistics. You can find more information about this in our [privacy policy](#)." Below the form is a link that says "> Restore Portainer from backup".

Creating an initial Portainer user account

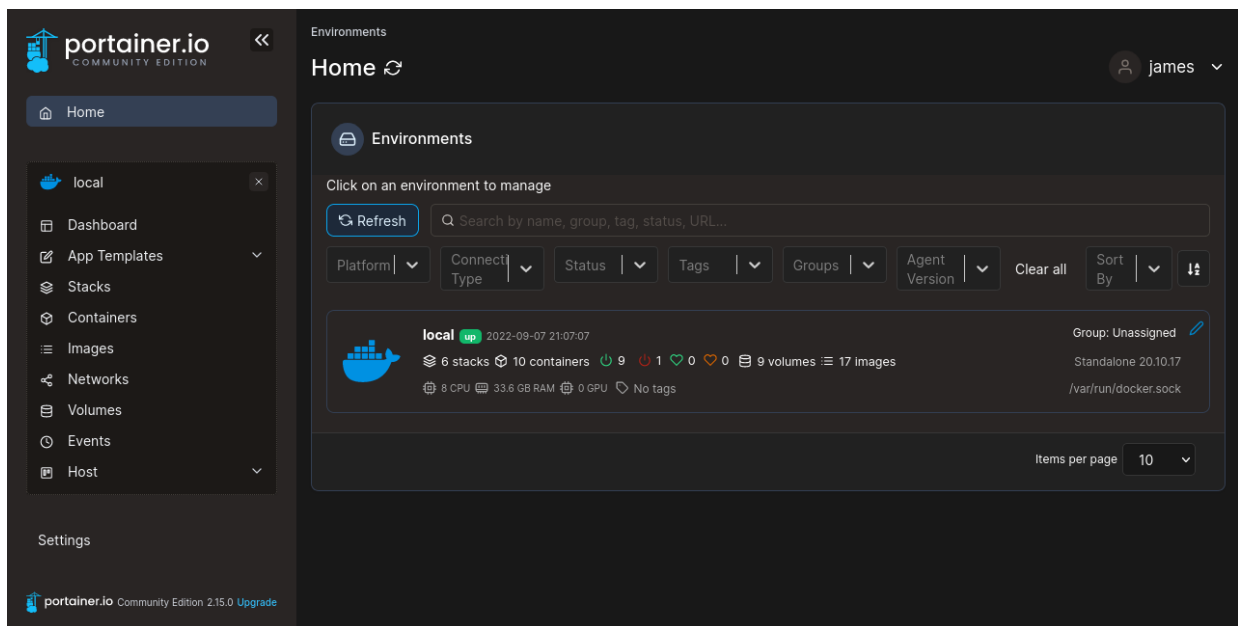
You'll be taken to the environment setup wizard. This is where you connect Portainer to your containerization systems. Click the **Get Started** button to continue with the local Docker socket mounted into the container from your host, and you'll end up on the Portainer dashboard:



Portainer’s environment setup screen

Touring the Portainer Dashboard

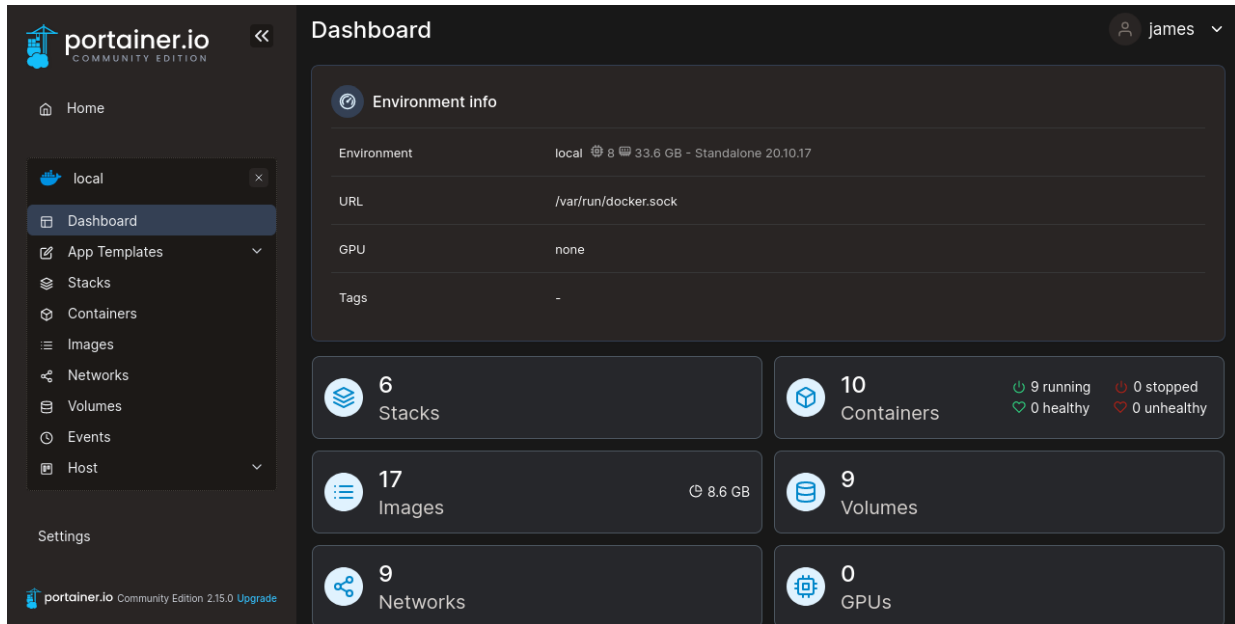
The dashboard provides an overview of all the environments you’ve added to Portainer. Although there’s only your **local** environment at the moment, you could add Kubernetes clusters and other remote Docker hosts in the future:



Portainer dashboard

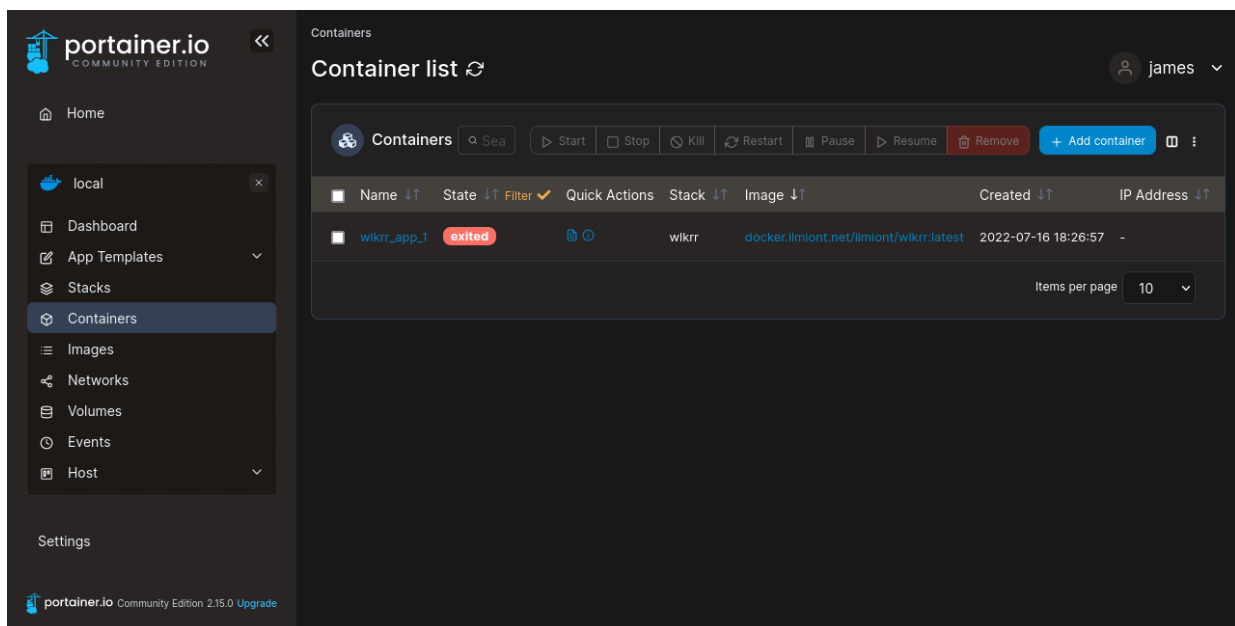
Each environment gets a summary tile, giving quick insights into the number of running, stopped, and healthy containers, as well as counts of the images and volumes available. The

sidebar to the left of the screen is where you can navigate between environments, resource types, and application-level global settings:



Portainer’s environment-specific dashboard

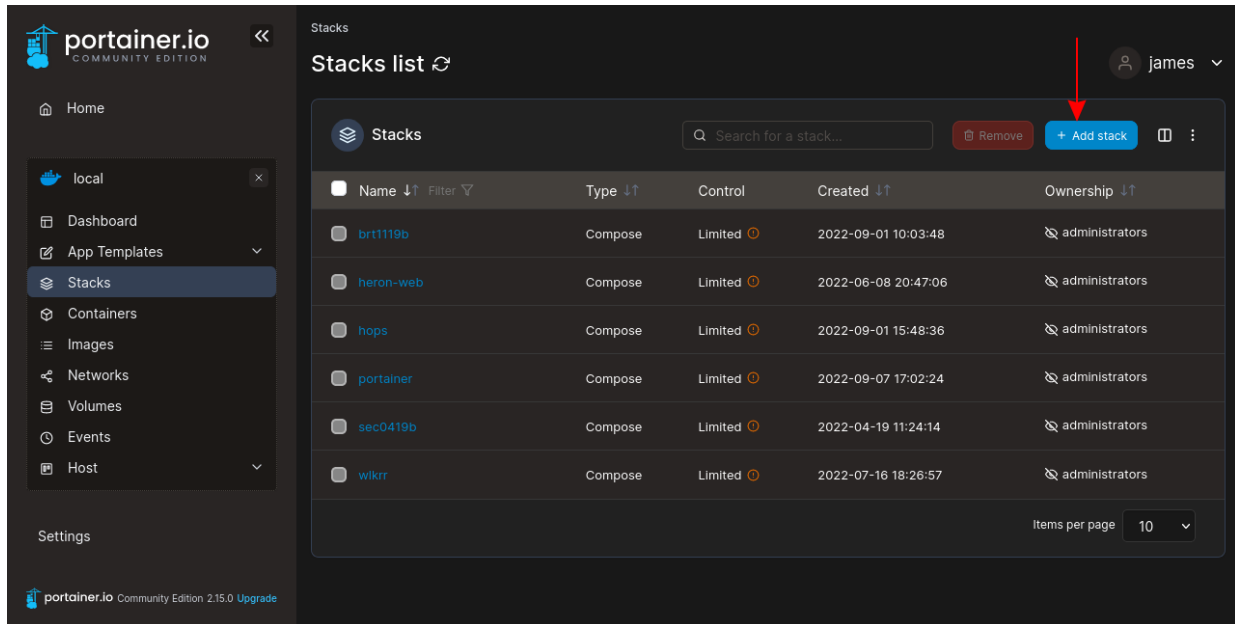
Clicking into an environment takes you to its own dashboard that summarizes the number of available resources. Clicking any resource type displays a table enumerating all the objects in the environment. Action buttons at the top of the screen are available to perform context-specific functions, such as stopping a container or deleting an image:



Viewing containers in Portainer

Deploying an Application with Portainer Stacks

Portainer provides several options for deploying new applications. One of these is **stacks**, a thin wrapper around Docker Compose functionality. A stack is a collection of one or more containers that collectively provide a complete application. You could have a stack consisting of an API, a database, and a frontend web UI:



The **Stacks** screen in Portainer

To create a new stack, click the **Stacks** menu item on the left sidebar and then press the **Add stack** button on the top-right. There are four ways to define a stack:

- **Web editor:** This lets you type out a Docker Compose file manually.
- **Upload:** This lets you upload an existing Docker Compose file from your machine.
- **Repository:** This automatically loads a Compose file directly from a Git repository.
- **Custom template:** This lets you can create your own reusable templates by heading to **App Templates > Custom Templates** on the left sidebar.

Here's a sample Compose file you can try:

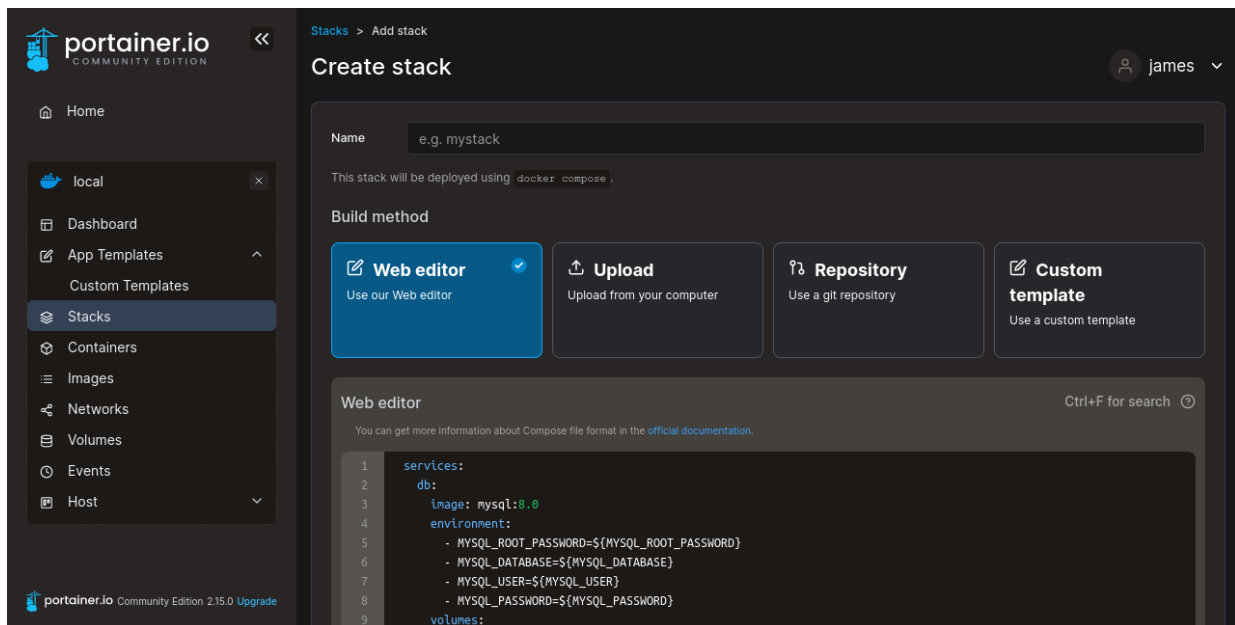
```
services:
  db:
    image: mysql:8.0
    environment:
      - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
```

```

- MYSQL_DATABASE=${MYSQL_DATABASE}
- MYSQL_USER=${MYSQL_USER}
- MYSQL_PASSWORD=${MYSQL_PASSWORD}
volumes:
  - db:/var/lib/mysql
wordpress:
  image: wordpress:latest
  ports:
    - 8880:80
  environment:
    - WORDPRESS_DB_HOST=db
    - WORDPRESS_DB_USER=${MYSQL_USER}
    - WORDPRESS_DB_PASSWORD=${MYSQL_PASSWORD}
    - WORDPRESS_DB_NAME=${MYSQL_DATABASE}
volumes:
  db:

```

This Compose file includes two services that run a basic WordPress site. Enter a name for your stack at the top of the screen, then paste the WordPress Compose file into the editor:

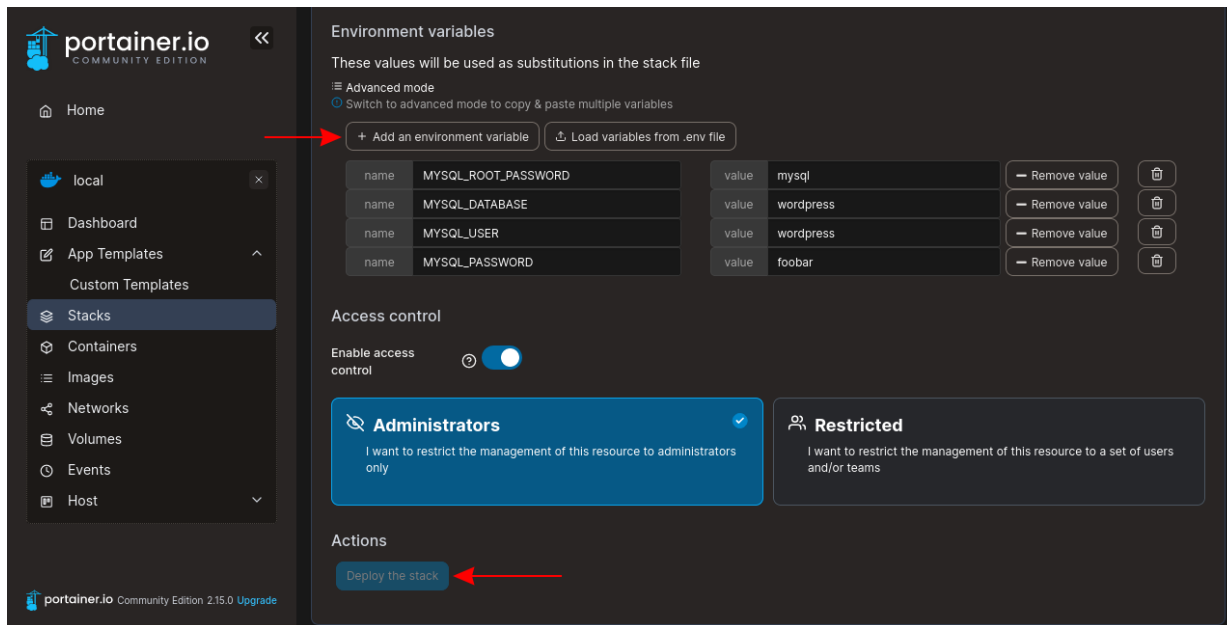


Portainer's stack editor

The Compose file uses environment variable substitution with **`${VARIABLE}`** syntax to configure the database connection. You need to supply values for these variables when you start your stack. To do this, scroll down the page and press the **Add an environment variable** button to create a new key-value pair. Repeat this for the four required variables:

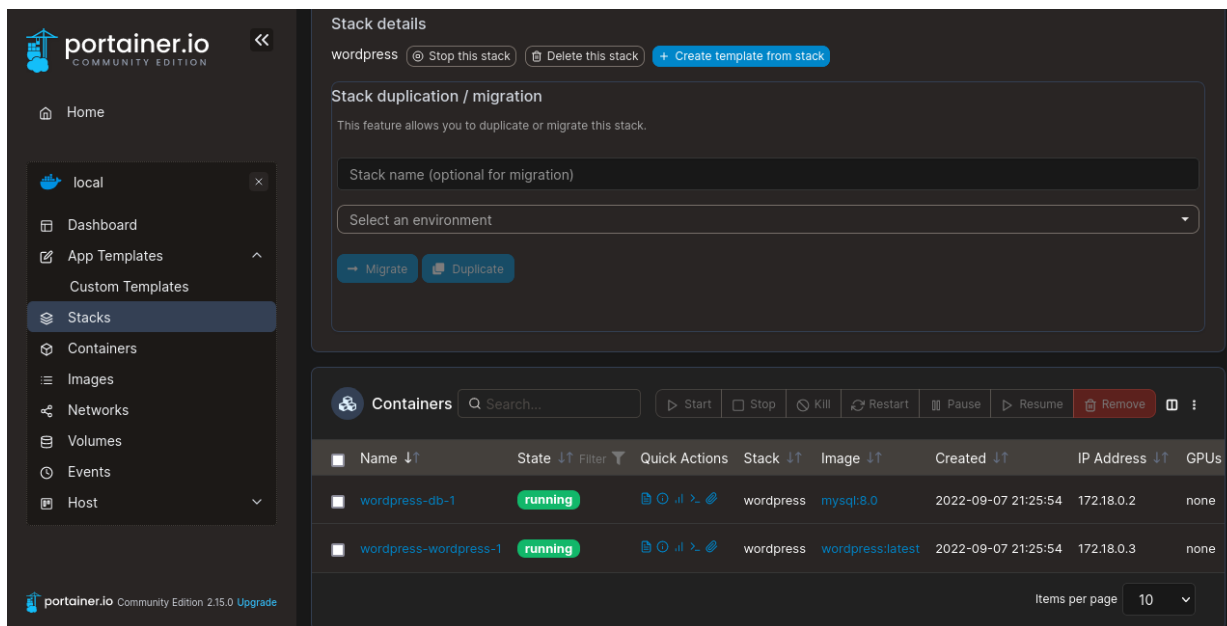
- **MYSQL_ROOT_PASSWORD**
- **MYSQL_DATABASE**
- **MYSQL_USER**

- **MYSQL_PASSWORD**



Setting Portainer stack environment variables

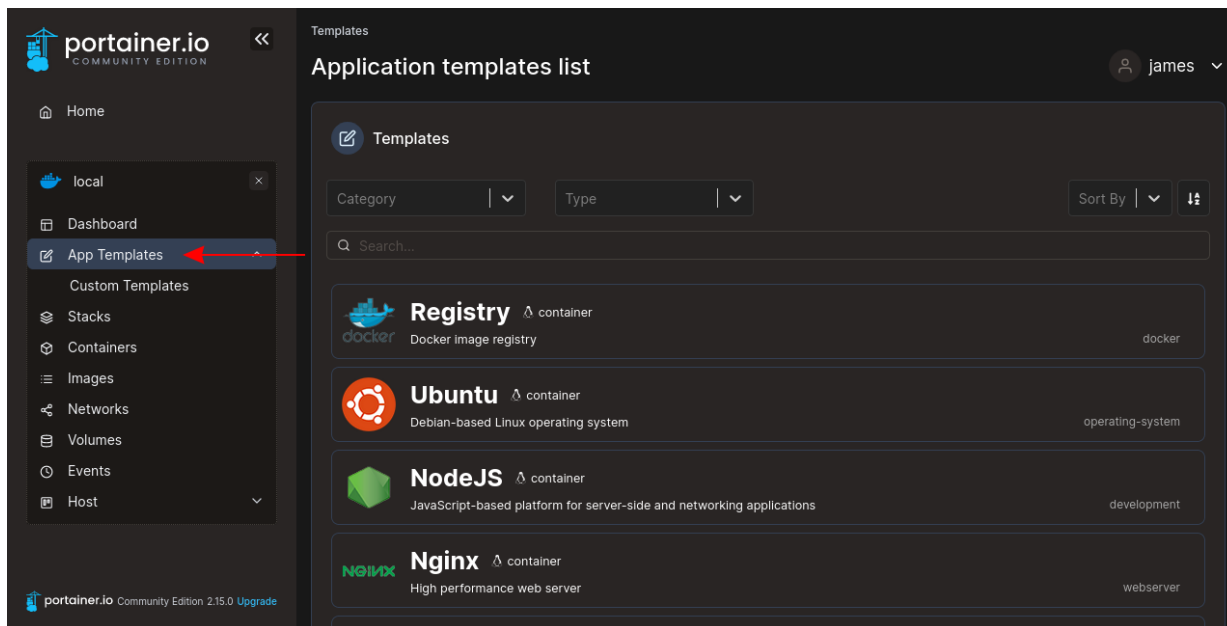
Press the blue **Deploy the stack** button at the bottom of the screen to start your services. It may take a few minutes for Portainer to pull the required images and create your containers. You'll then be taken to the stack's page, which shows the details of the running containers. You can access the created WordPress site by heading to <http://localhost:8880> in your browser:



Viewing a running stack in Portainer

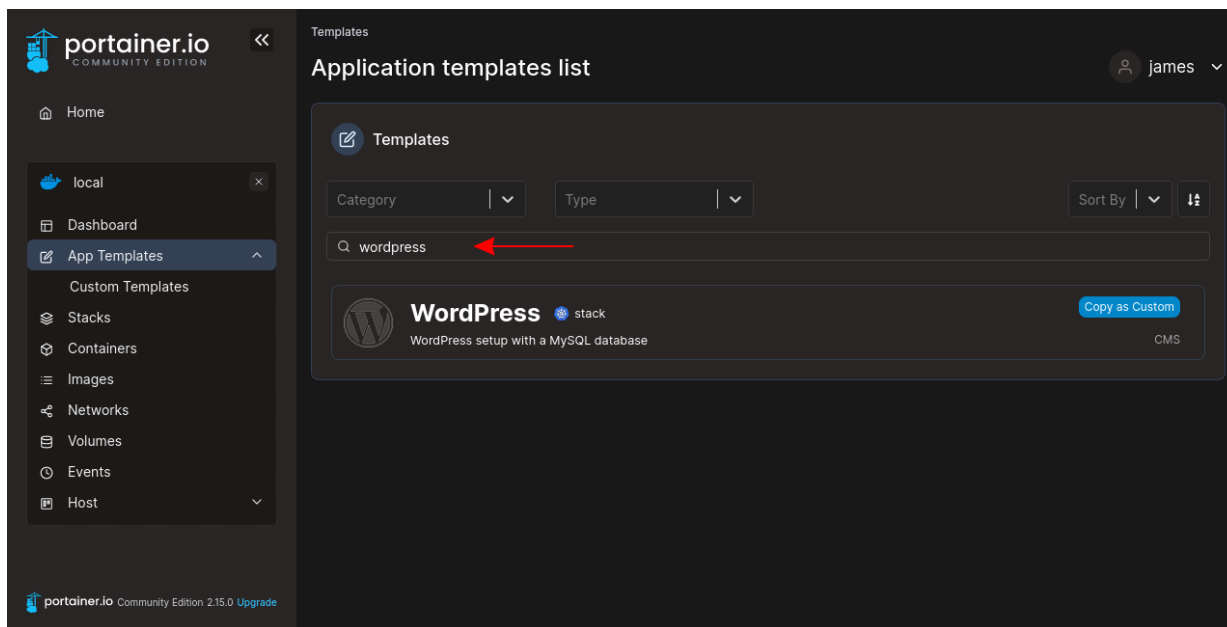
Deploying a Portainer Template

Templates are an even easier way to launch new application instances. Portainer comes with a set of built-in templates for popular apps. These can be reached by heading to **App Templates** on the left sidebar. You can also create your own templates based on Compose files:



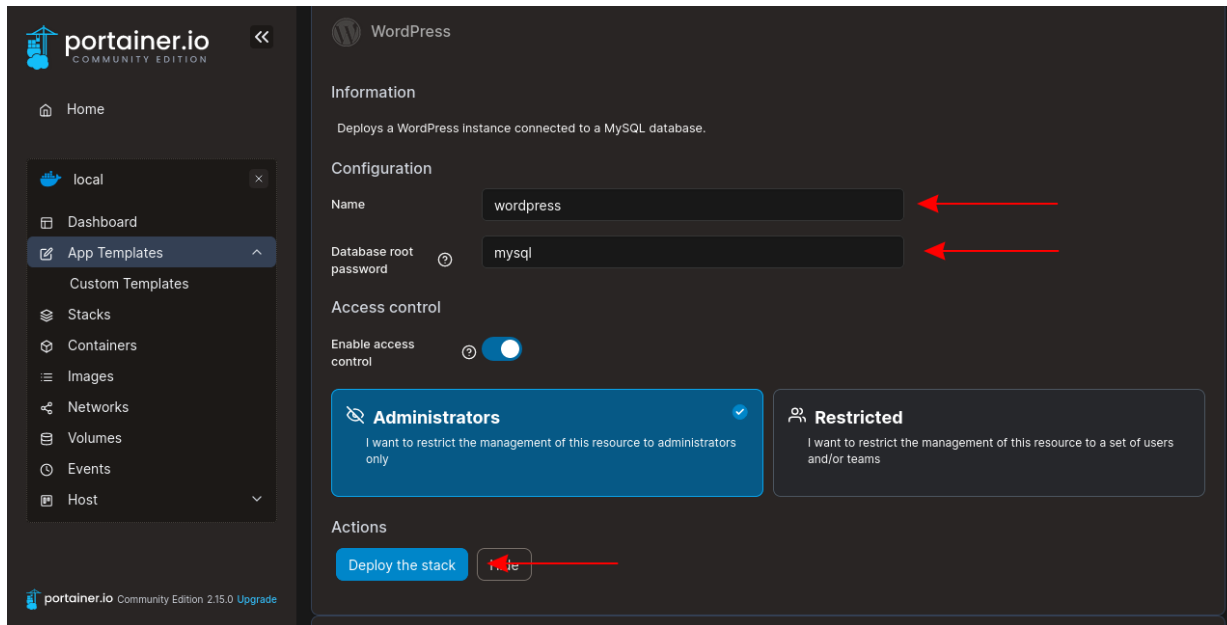
Portainer’s built-in app templates

You could replicate the WordPress site created earlier by using the official WordPress template. Head to **Add Templates** and enter “wordpress” into the search bar at the top of the screen:



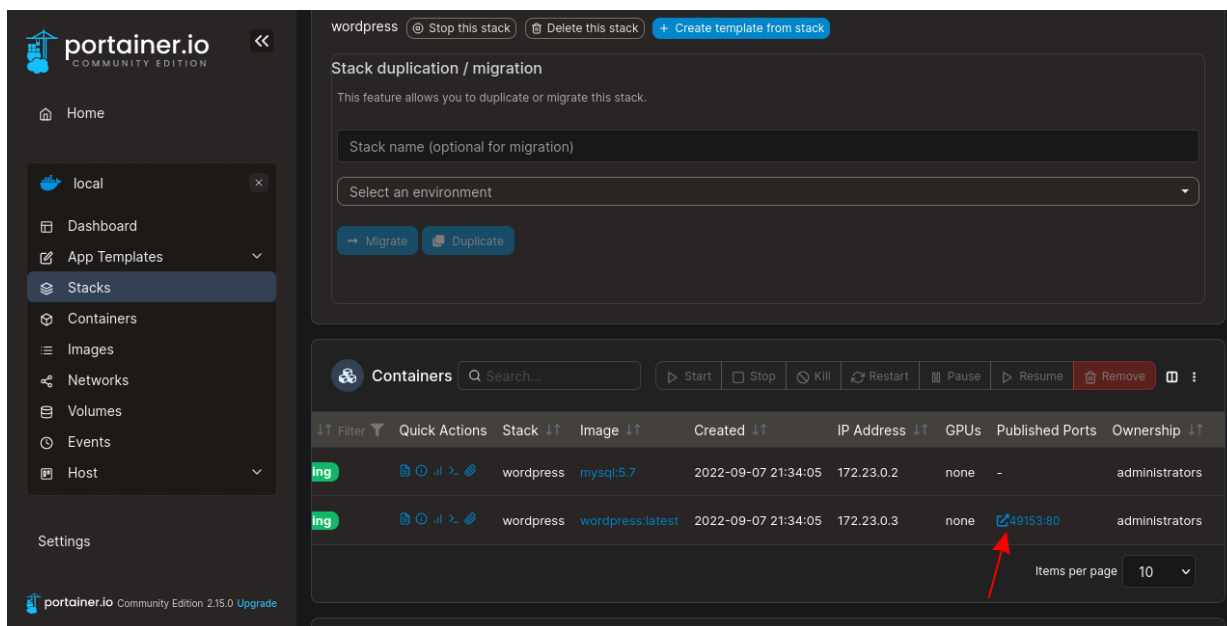
Searching for the WordPress app template in Portainer

The template comes preconfigured with the services you need to run a WordPress site. You only have to supply a name for your stack and the root password to set it on the MySQL database server. Enter these into the fields at the top of the page, and then press the **Deploy** the stack button at the bottom:



Deploying the WordPress app template in Portainer

Wait while Portainer pulls your images and creates your containers. The container will be assigned a random port by default. You can find it by navigating to the stack's details page and then scrolling the Containers table so you can view the port published by the WordPress service. This example is accessible on **localhost : 49153** :

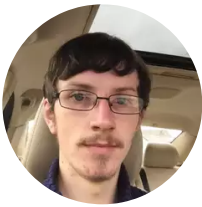


Conclusion

Portainer is a convenient and feature-rich interface for Docker containers and other environments. It brings almost all the capabilities of the Docker UI to your web browser, letting you perform management operations on any device.

Portainer is ideal for many different use cases, from your local development workstation to production app monitoring. You can also use it to track containers and images used by CI/CD pipelines and build systems, preventing excess resources from accumulating on your Docker host.

James Walker



James Walker is the founder of Heron Web, a UK-based software development studio providing bespoke solutions for SMEs. He's experienced in delivering custom software using engineering workflows built around modern DevOps methodologies. James is also a freelance technical writer and has written extensively about the software development lifecycle, current industry trends, and DevOps concepts and technologies.

Writers at Earthly work closely with our talented editors to help them create high quality tutorials. This article was edited by:



Bala Priya C

Bala is a technical writer who enjoys creating long-form content. Her areas of interest include math and programming. She shares her learning with the developer community by authoring tutorials, how-to guides, and more.

Updated: April 17, 2023

Published: November 26, 2022