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.

- Docker
- Docker Compose
- Portainer
- Watchtower

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
```

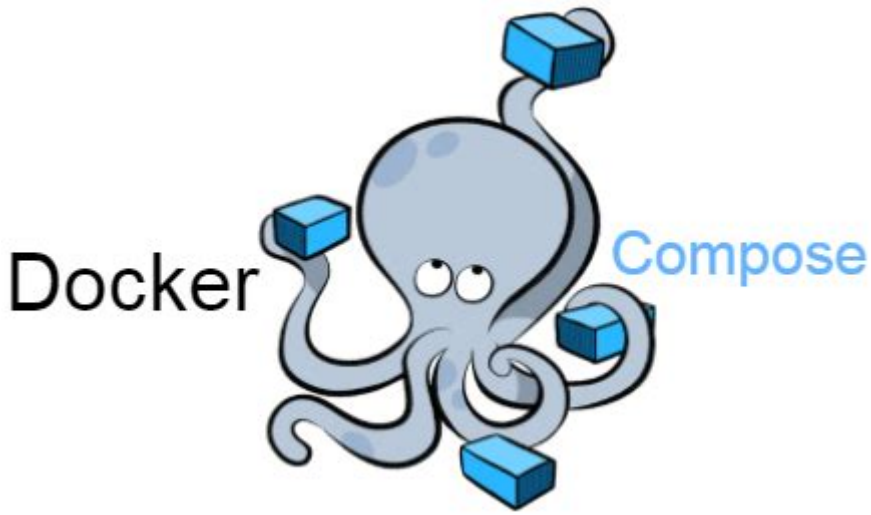
The output should be similar to the following, showing that the service is active and running:

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.

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 v

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

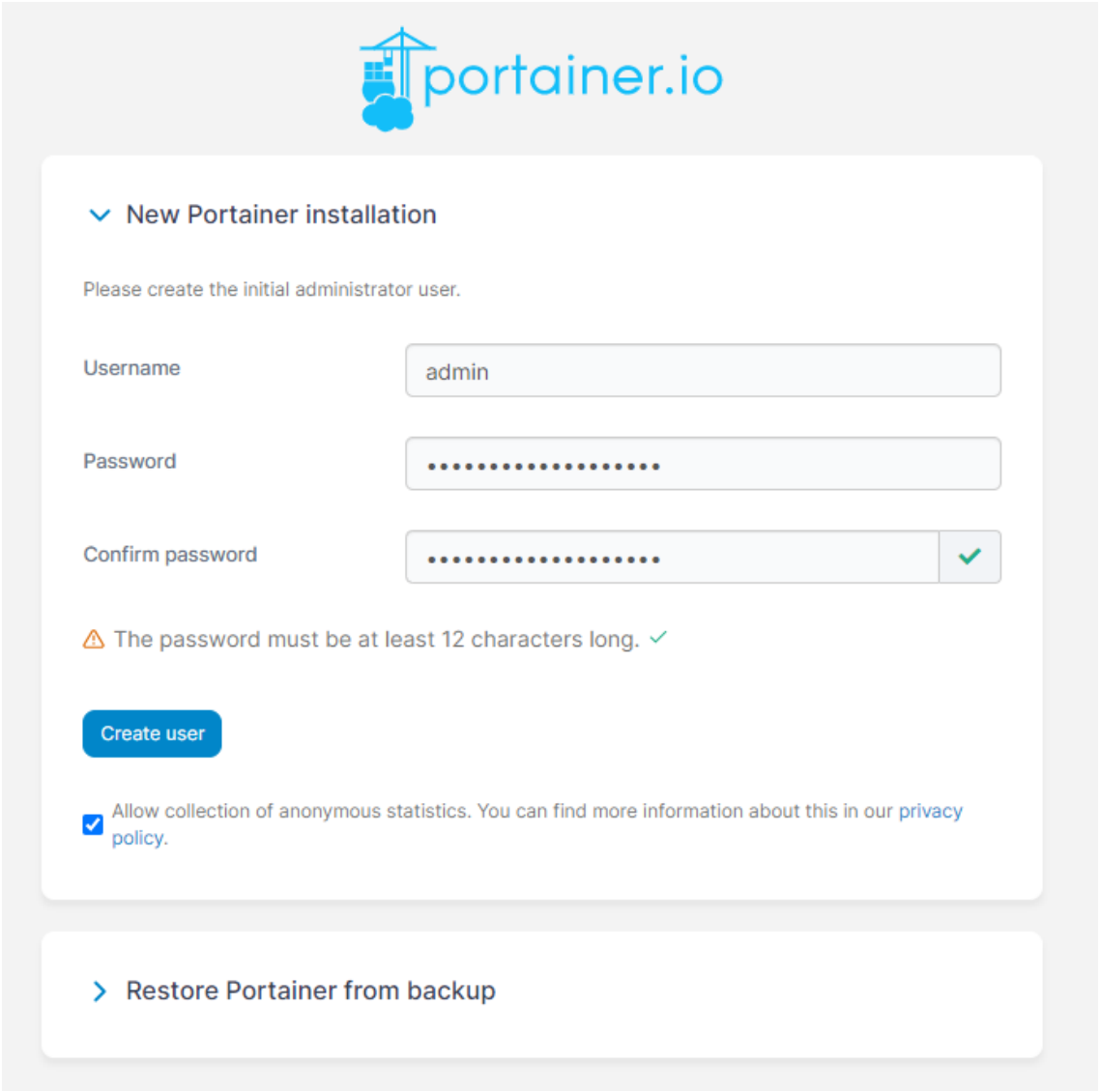
Replace `localhost` with the relevant IP address or FQDN if needed, and adjust the port if you changed it earlier.

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

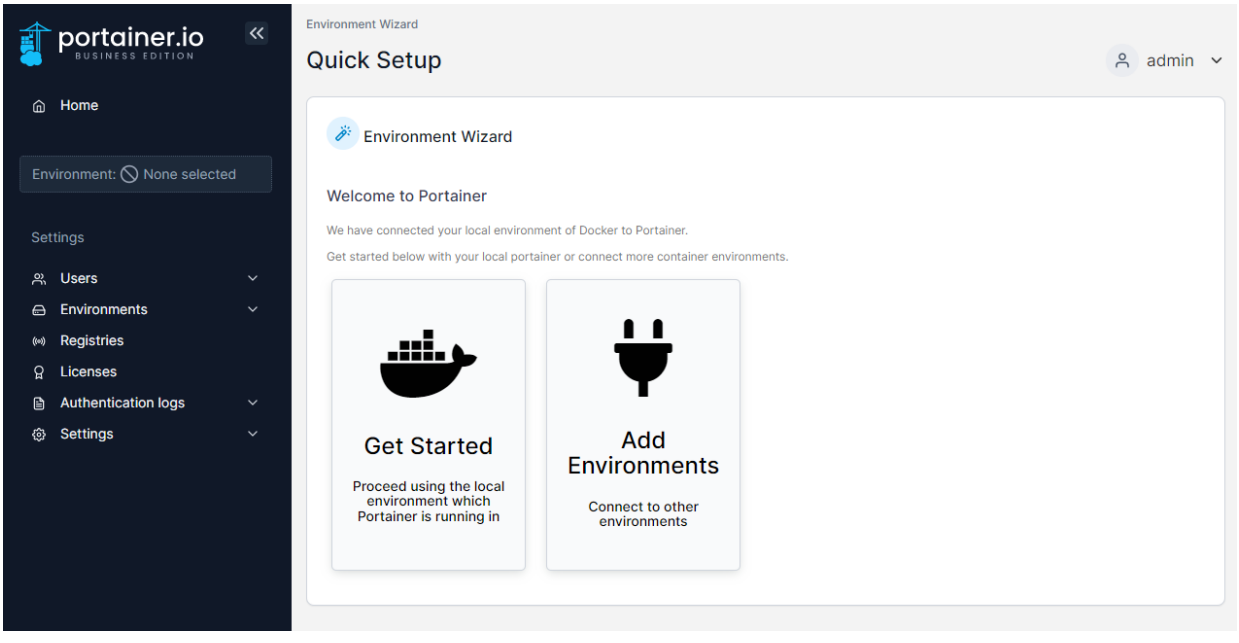
Initial Setup

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.

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.