DigitalOcean

Subscribe

How To Run Nginx in a Docker Container on Ubuntu 14.04

36

Posted October 28, 2015 315.9k DOCKER NGINX UBUNTU

By: Thomas Taege

Introduction

This tutorial shows how to deploy Nginx in a Docker container.

By containerizing Nginx, we cut down on our sysadmin overhead. We will no longer need to manage Nginx through a package manager or build it from source. The Docker container allows us to simply replace the whole container when a new version of Nginx is released. We only need to maintain the Nginx configuration file and our content.

Nginx describes itself as:

> nginx [engine x] is an HTTP and reverse proxy server, a mail proxy server, and a generic TCP proxy server, originally written by Igor Sysoev.

In practice many sysadmins use Nginx to serve web content, from flat-file websites to upstream APIs in NodeJS. In this tutorial we will serve a basic web page, so we can focus on configuring Nginx with a Docker container.

Docker containers are a popular form of a relatively old operations practice: containerization. Containerization differs from virtualization in that virtualization abstracts away the hardware, while containerization abstracts away the base operating system, too. In practical terms this means we can take an application (or group of applications) and wrap them in a container (or containers) to make them modular, portable, composable, and lightweight.

This portability means you can install the Docker Engine (also referred to as Docker Core, and even just Docker) on a wide variety of operating systems, and any functional container written by anyone will run on it.

If you want to learn more about Docker you can check out an introductory Docker tutorial .

For the purposes of this article we will be installing the Docker Engine on Ubuntu 14.04.

We will be installing the current stable version of Docker for Ubuntu, which is 1.8.1.

This tutorial is aimed at Nginx users who are new to Docker. If you want just the bare commands for setting up your Nginx container, you can do Step 1 and then jump to Step 5.

If you want to build up to your container step by step and learn about port mapping and detached mode, follow the whole tutorial.

Prerequisites

To containerize Nginx, please complete the following:

- Set up an Ubuntu 14.04 server, preferably with SSH keys for security
- Set up a sudo user
- Verify your kernel version

Docker 1.8.1 relies on some fairly recent kernel features, so make sure the kernel is at **3.10** or above. A fresh image will be running a fairly new kernel, but if you need to check, just run `uname -r` .

```
$ uname -r
```

We've included the output from a fresh Ubuntu 14.04 Droplet below, which is over 3.10, so you shouldn't have anything to worry about unless you're running this on an older image.

```
Output
3.13.0-57-generic
```

Step 1 — Installing Docker

Docker hosts a startup script to get Docker up and running on your machine. We can simply run the command:

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

> In general, you shouldn't pipe random scripts from the internet to your shell (`| sh`), as they could pretty much be doing anything. Take a look at get.docker.com if you want to know what you're getting yourself into.

Once this completes you'll see the installed version like as shown below (your readout may be newer; this is fine) and some instructions for running as non-root/without sudo. We're running through this tutorial as with a sudo user, though, so no need to worry about this for the purposes of this tutorial.

Output

```
Client:
 Version:      1.8.3
 API version:  1.20
 Go version:   go1.4.2
 Git commit:   f4bf5c7
 Built:        Mon Oct 12 05:37:18 UTC 2015
 OS/Arch:      linux/amd64

Server:
 Version:      1.8.3
 API version:  1.20
 Go version:   go1.4.2
 Git commit:   f4bf5c7
 Built:        Mon Oct 12 05:37:18 UTC 2015
 OS/Arch:      linux/amd64
```

Optional: Run the `hello-world` container to make sure everything in working as expected.

```
$ sudo docker run hello-world
```

You should see output similar to that shown below.

Output

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
535020c3e8ad: Pull complete
af340544ed62: Already exists
library/hello-world:latest: The image you are pulling has been verified. Important: ima
Digest: sha256:d5fbd996e6562438f7ea5389d7da867fe58e04d581810e230df4cc073271ea52
Status: Downloaded newer image for hello-world:latest
```

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

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:
 https://hub.docker.com

For more examples and ideas, visit:
 https://docs.docker.com/userguide/
```

With this complete, we're able to dive in to the basics of Docker.

(Optional) Step 2 — Reviewing Container Basics: Run, List, Remove

This section shows how to run a basic container and then remove it. If you already know how to use Docker in general, and want to skip to the Nginx part, go to Step 5.

We've installed the Docker Client as part of our Docker installation, so we have access to the command line tool that allows us to interact with our containers.

If we run the following command;

```
$ sudo docker ps -a
```

You should get output similar to the following;

```
Output
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
a3b149c3ddea        hello-world         "/hello"            3 minutes ago       Exited (
```

We can see some basic information about our container.

You'll notice it has a nonsensical name like `nostalgic_hopper`; these names are generated automatically if you don't specify one when creating the container.

We can also see in that the `hello-world` example container was run 3 minutes ago and exited 3 minutes ago.

If we run this container again with this command (replacing `nostalgic_hopper` with your own container name):

```
$ sudo docker start nostalgic_hopper
```

Then run the command to list containers:

```
$ sudo docker ps -a
```

We should now see that the container has run recently;

```
Output
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
a3b149c3ddea        hello-world         "/hello"            4 minutes ago       Exited (0) 9
```

By default, Docker containers run their assigned commands and then exit.

Some containers will be set up to run through a list of tasks and finish, while others will run indefinitely.

Now that we've gone through some Docker basics, let's remove the `hello-world` image, as we won't be needing it again (remember to replace `nostalgic_hopper` with your container name, or use your container ID).

```
$ sudo docker rm nostalgic_hopper
```

Next we'll start using Nginx.

(Optional) Step 3 — Learning How to Expose the Port

In this section we'll download the Nginx Docker image and show you how to run the container so it's publicly accessible as a web server.

By default containers are not accessible from the Internet, so we need to map the container's *internal* port to the Droplet's port. That's what this section will teach you!

First, though, we'll get the Nginx image.

> Step 5 contains the final commands to deploy the full container, so if you aren't overly concerned with the implementation details you can skip straight there.

Run the following command to get the Nginx Docker image:

```
$ sudo docker pull nginx
```

This downloads all the necessary components for the container. Docker will cache these, so when we run the container we don't need to download the container image(s) each time.

Docker maintains a site called Dockerhub, a public repository of Docker files (including both official and user-submitted images). The image we downloaded is the official Nginx one, which saves us from having to build our own image.

Let's start our Nginx Docker container with this command:

```
$ sudo docker run --name docker-nginx -p 80:80 nginx
```

- `run` is the command to create a new container
- The `--name` flag is how we specify the name of the container (if left blank one is assigned for us, like nostalgic_hopper from Step 2)
- `-p` specifies the port we are exposing in the format of `-p local-machine-port:internal-container-port`. In this case we are mapping Port 80 in the container to Port 80 on the server
- `nginx` is the name of the image on dockerhub (we downloaded this before with the pull command, but Docker will do this automatically if the image is missing)

That's all we need to get Nginx up! Paste the IP address of your Droplet into a web browser and you should see Nginx's "Welcome to nginx!" page.

You'll also notice in your shell session that the log for Nginx is being updated when you make requests to your server, because we're running our container interactively.

Let's hit the break shortcut `CTRL+C` to get back to our shell session.

If you try to load the page now, you'll get a "connection refused" page. This is because we shut down our container. We can verify this with this command:

```
$ sudo docker ps -a
```

You should see something similar to the output shown below.

```
Output
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
05012ab02ca1        nginx               "nginx -g 'daemon off"   57 seconds ago      Exited
```

We can see that our Docker container has exited.

Nginx isn't going to be very useful if we need to be attached to the container image for it to work, so in the next step we'll show you how to detach the container to allow it to run independently.

Remove the existing `docker-nginx` container with this command:

```
$ sudo docker rm docker-nginx
```

In the next step we'll show you how to run it in detached mode.

(Optional) Step 4 — Learning How to Run in Detached Mode

Create a new, detached Nginx container with this command:

```
$ sudo docker run --name docker-nginx -p 80:80 -d nginx
```

We added the `-d` flag to run this container in the background.

The output should simply be the new container's ID.

If we run the list command:

```
$ sudo docker ps
```

We're going to see a couple things in the output we haven't seen before.

```
Output
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATU
b91f3ce26553        nginx               "nginx -g 'daemon off"   About a minute ago   Up Ab
```

We can see that instead of `Exited (0) X minutes ago` we now have `Up About a minute`, and we can also see the port mapping.

If we go to our server's IP address again in our browser, we will be able to see the "Welcome to nginx!" page again. This time it's running in the background because we specified the `-d` flag, which tells Docker to run this container in detached mode.

Now we have a running instance of Nginx in a detached container!

It's not useful enough yet, though, because we can't edit the config file, and the container has no access to any of our website files.

Stop the container by running the following command:

```
$ sudo docker stop docker-nginx
```

Now that the container is stopped (you can check with `sudo docker ps -a` if you want to be sure), we can remove it by running the following command;

```
$ sudo docker rm docker-nginx
```

Now we'll get to the final version of our container, with a quick stop to generate a custom website file.

Step 5 — Building a Web Page to Serve on Nginx

In this step, we'll create a custom index page for our website. This setup allows us to have persistent website content that's hosted outside of the (transient) container.

Let's create a new directory for our website content within our home directory, and move to it, by running the commands shown below.

```
$ mkdir -p ~/docker-nginx/html
$ cd ~/docker-nginx/html
```

Now let's create an HTML file (we show the commands for Vim, but you can use any text editor you like).

```
$ vim index.html
```

Enter insert mode by pressing `i`. Paste in the content shown below (or feel free to add your own HTML markup).

```
<html>
  <head>
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css" rel=
    <title>Docker nginx Tutorial</title>
  </head>
  <body>
    <div class="container">
      <h1>Hello Digital Ocean</h1>
      <p>This nginx page is brought to you by Docker and Digital Ocean</p>
    </div>
  </body>
</html>
```

If you're familiar with HTML, you'll see that this is a super basic web page. We've included a `<link>` tag that is pointed to a CDN for Bootstrap (a CSS framework that gives your web page a collection of responsive styles). You can read more about Bootstrap.

We can save this file now by pressing `ESC`, and then `:wq` and `ENTER`:

- write (`w`) tells Vim to write the changes to the file
- quit (`q`) tells Vim to exit

We've now got a simple index page to replace the default Nginx landing page.

Step 6 — Linking the Container to the Local Filesystem

In this section, we'll put it all together. We'll start our Nginx container so it's accessible on the Internet over Port 80, and we'll connect it to our website content on the server.

Background information about volumes; that is, linking to permanent server content from your container:

Docker allows us to link directories from our virtual machine's local file system to our containers.

In our case, since we want to server web pages, we need to give our container the files to render.

We could copy the files into the container as part of a Dockerfile, or copy them into the container after the fact, but both of these methods leave our website in a static state inside the container. By using Docker's data volumes feature, we can create a symbolic link between the Droplet's filesystem and the container's filesystem. This allows us to edit our existing web page files and add new ones into the directory and our container will automatically access them. If you want to read more about Docker and volumes check out the data volumes documentation.

The Nginx container is set up by default to look for an index page at `/usr/share/nginx/html`, so in our new Docker container, we need to give it access to our files at that location.

Making the link:

To do this, we use the `-v` flag to map a folder from our local machine (`~/docker-nginx/html`) to a relative path in the container (`/usr/share/nginx/html`).

We can accomplish this by running the following command:

```
$ sudo docker run --name docker-nginx -p 80:80 -d -v ~/docker-nginx/html:/usr/share/nginx/h
```

We can see that the new addition to the command `-v ~/docker-nginx/html:/usr/share/nginx/html` is our volume link.

- `-v` specifies that we're linking a volume
- the part to the left of the `:` is the location of our file/directory on our virtual machine (`~/docker-nginx/html`)
- the part to the right of the `:` is the location that we are linking to in our container (`/usr/share/nginx/html`)

After running that command, if you now point your browser to your DigitalOcean Droplet's IP address, you should see the first heading of **Hello Digital Ocean** (or whatever web page you created in Step 5).

If you're happy with the other Nginx defaults, you're all set.

You can upload more content to the `~/docker-nginx/html/` directory, and it will be added to your live website.

For example, if we modify our index file, and if we reload our browser window, we will be able to see it update in realtime. We could build a whole site out of flat HTML files this way if we wanted to. For example, if we added an `about.html` page, we could access it at `http://your_server_ip/about.html` without needing to interact with the container.

(Optional) Step 7 — Using Your Own Nginx Configuration File

This section is for advanced users who want to use their own Nginx config file with their Nginx container. Please skip this step if you don't have a custom config file you want to use.

Let's go back a directory so we aren't writing to our public HTML directory:

```
$ cd ~/docker-nginx
```

If you'd like to take a look at the default config file, just copy it using the Docker copy command:

```
$ sudo docker cp docker-nginx:/etc/nginx/conf.d/default.conf default.conf
```

Since we're going to be using a custom `.conf` file for Nginx, we will need to rebuild the container.

First stop the container:

```
$ sudo docker stop docker-nginx
```

Remove it with:

```
$ sudo docker rm docker-nginx
```

Now you can edit the default file locally (to serve a new directory, or to use a `proxy_pass` to forward the traffic to another app/container like you would with a regular Nginx installation). You can read about Nginx's configuration file in our Nginx config file guide.

Once you've saved your custom config file, it's time to make the Nginx container. Simply add a second `-v` flag with the appropriate paths to give a fresh Nginx container the appropriate links

to run from your own config file.

```
$ sudo docker run --name docker-nginx -p 80:80 -v ~/docker-nginx/html:/usr/share/nginx/html
```

This command also still links in the custom website pages to the container.

Please note that you will need to restart the container using a `docker restart` command if you make any changes to your config file after starting the container, since Nginx does not hot reload if its config file is changed:

```
$ sudo docker restart docker-nginx
```

Conclusion

You now have a running Nginx container serving a custom web page.

From here, we recommend reading up on Docker's container linking if you want to learn about linking containers together for the purposes of using Nginx as a reverse proxy for serving other container-based web apps.

If you wanted to manage a group of containers, such as an app container, a database container, and this Nginx container, take a look at Docker Compose.

By: Thomas Taege

Editor:
Sharon Campbell

Related Tutorials

Docker Explained: How To Containerize and Use Nginx as a Proxy

How to Use Traefik as a Reverse Proxy for Docker Containers on Ubuntu 18.04

How to Deploy a Symfony 4 Application to Production with LEMP on Ubuntu 18.04

How To Provision and Manage Remote Docker Hosts with Docker Machine on Ubuntu 18.04

Webinar Series: Building Blocks for Doing CI/CD with Kubernetes

7 Comments

Leave a comment...

Log In to Comment

digitaldragon *October 30, 2015*

Hello,

0

I receive the following error when I run command:

```
sudo docker run --name docker-nginx -p 80:80 nginx
```

Error Message:

```
The name "docker-nginx" is already in use by container 9730ffe1c2a8. You have to delete (or rename) that container to be able to reuse that name.
```

```
xxxx@service2:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             ST
xxxx@service2:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VI
nginx               latest              914c82c5a678        38 hours ago        13
xxxx@service2:~$
```

digitaldragon *October 31, 2015*

0 This command worked for me:

```
sudo docker run --name mynginx -P -d nginx
```

digitaldragon *October 31, 2015*

0 Problem solved. I forgot to add the `-a` option after `sudo docker ps` . The `-a` option is required to show all containers. Then I just removed the stopped container that was using the name `docker-nginx` .

joenarvaez *December 17, 2015*

0 Thanks for the great article!

However, I'm unsuccessful in using proxy_pass.

here is my container app mapping to the host:

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
cd5c4c1bdd0f        drfms_web           "python server.py"       3 minutes ago
```

here's my default.conf

```
server {
    listen       80;
    server_name  drfms.jndesigns.io;

    location / {
        proxy_pass http://127.0.0.1:8080;
    }
}
```

here's my command to start the ngnix container:

```
sudo docker run --name docker-nginx -p 80:80  -p 5000:8080  -v ~/docker-nginx/defa
```

I get this error:

```
failed to create endpoint dockernginx on network bridge: Bind for 0.0.0.0:5000 fai
```

Is there a way to specify that I want to forward a port directly to another container?

---

prachayasu *February 16, 2016*

0 Hi, Great article. But I have a question about nginx. Is it necessary to install Nginx on a Droplet that run on an external docker? or Can I install only Docker?

Aery *March 11, 2016*

0 You simply need to create your droplet with the desired operating system, install Docker, and then install the Docker image. The image, from my understanding, is a collection of software that can be mapped to ports on the system, such as Nginx. You specific a command to allow TCP/80 access to the software, in this event, Nginx, so you can serve files to the web.

In short, there's no need to create Nginx on your base operating system if you simply want to run a web server using Docker.

---

paulvanbladel *June 13, 2017*

0 Very nice article, thanks for sharing.
What's your recommended approach when the server hosting docker is used for multiple web sites?

Would you create for each web site a separate nginx docker container or would you manage them all in one nginx docker container?