< !-- Awin verification 001 -->

Nginx Inside Docker with Ubuntu 16.04 HOWTO

Run Nginx inside a Docker Container with Ubuntu 16.04

> *“Live boldly. Push yourself. Don’t settle.” — Jojo Moyes, Me Before You*

Contents [hide]

1. Introduction

Docker is a very cool technology for running whole operating systems inside a process. You run a command on the host system and a whole new virtual system springs into existence, ready to operate as you have configured it. This capability offers unprecedented power and flexibility to build and throw away infrastructure as you wish. You could use this capability in test environments, running applications with differing system requirements in the same host, and a lot more.

We have previously covered installing the Nginx web server and enabling it with SSL support. We have also talked about hardening the Nginx SSL installation. These activities were performed on a new operating system build. Let us now demonstrate how to run the Nginx web server with SSL inside of a docker container. We also show you how to use this running container to take over the duties of web serving on the host. The advantage of hosting a web server inside a docker container are many; for example, you could separate the web serving part from the database, with the database also running inside a docker container. (We will demonstrate this configuration in a future article.)

2. Preparation

Let us now see what preparatory steps are required to run Nginx inside a docker container. Obviously, the first thing we want to do on a new system is to install docker. Assuming Ubuntu 16.04 host server, run the following command as the **root** user to install docker.

```
apt-get install docker.io
```

Assuming you are smart enough not to do all development as the **root** user, set up a non-privileged user account. In our case this account is called **aurora**. Pick whatever name suits you.

```
adduser aurora
usermod -aG docker aurora
```

Now the user **aurora** can log in (after setting password, or enabling password-less ssh access) and run docker commands.

3. A Docker Primer

The Docker application is a server that runs on a system and is controlled by a command called **docker**. The docker can run instances of an *image* which is a complete operating system packed into a file. This running instance is known as a **container**, as in a container for an operating system. You can start and stop these containers are per demand. You can also completely delete containers and images from the docker system.

To get started, you can build an image from a base operating system image, and configure it to your specifications. To ease the repeated building and configuration of images, Docker uses a build file known as the *Dockerfile*. Here is the complete reference to the Dockerfile, and the directives it can include.

4. Preparing the Dockerfile

We will now use a Dockerfile to prepare the image to can run the Nginx web server.

First, we choose Ubuntu 16.04 as the base server image using the Dockerfile clause *FROM*. The *MAINTAINER* should list your name and email address.

```
FROM ubuntu:16.04
MAINTAINER Jack Black "jack.black@example.com"
```

Next, we run a few commands to configure the base image.

First is to update the server OS.

```
RUN DEBIAN_FRONTEND=noninteractive apt-get update && \
    DEBIAN_FRONTEND=noninteractive apt-get -y dist-upgrade
```

We obviously need to install *nginx*. Also require *net-tools* which contains the *netstat* command.

```
RUN DEBIAN_FRONTEND=noninteractive apt-get -yq install net-tools nginx
```

Add a non-privileged user for normal operation.

```
RUN useradd -ms /bin/bash aurora
```

Next, we adjust the configuration of Nginx by removing some unneeded files.

```
RUN rm -f /etc/nginx/fastcgi.conf /etc/nginx/fastcgi_params && \
    rm -f /etc/nginx/snippets/fastcgi-php.conf
/etc/nginx/snippets/snakeoil.conf
```

Docker supports its own networking inside the container. By default, listen ports opened inside the running container are not accessible from outside the container. This is a security feature. Opened ports must be explicitly declared to be accessible from outside. Since Nginx opens ports 80 and 443, and we want these accessible from the outside, we need to explicitly tell Docker to expose them.

```
EXPOSE 80
EXPOSE 443
```

We have configured Nginx for SSL (HTTPS) and these are the files that are being used. We copy these files from the build environment to the correct path inside the image. We discuss contents of these files below. You can also refer to this article for the complete details.

```
COPY nginx/ssl /etc/nginx/ssl
COPY nginx/snippets /etc/nginx/snippets
COPY nginx/sites-available /etc/nginx/sites-available
```

The last directive in our Dockerfile is to have Nginx run in the foreground inside the container.

```
ENTRYPOINT ["/usr/sbin/nginx", "-g", "daemon off;"]
```

5. Configuring Nginx

Note, we have a detailed article describing how to configure Nginx for SSL. Below we present only the changes required to the stock Nginx installation to enable SSL.

- **nginx/ssl**: This is a directory containing the SSL certificate (bundled) and the private key used to sign the CSR.
- **nginx/snippets**: This is a directory which includes a single file called *ssl.conf*. ***We have discussed here how the various clauses which enhance the SSL setup of your Nginx web server.*** It looks like this:

```
ssl_certificate /etc/nginx/ssl/example.com.bundle.cer;
ssl_certificate_key /etc/nginx/ssl/example.com.keynopass;

ssl_dhparam /etc/nginx/ssl/dhparam.pem;

ssl_session_timeout 10m;

# Use this Block to support IE < 9, Android < 2.2 or Java < 6
ssl_ciphers "ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-
SHA256:DHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:ECDHE-
RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-
SHA:ECDHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES128-
SHA256:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA:ECDHE-RSA-DES-CBC3-
SHA:EDH-RSA-DES-CBC3-SHA:AES256-GCM-SHA384:AES128-GCM-SHA256:AES256-
SHA256:AES128-SHA256:AES256-SHA:AES128-SHA:DES-CBC3-
SHA:HIGH:!aNULL:!eNULL:!EXPORT:!DES:!MD5:!PSK:!RC4:!0x6b";
ssl_protocols TLSv1 TLSv1.1 TLSv1.2; ssl_prefer_server_ciphers on;
ssl_session_cache shared:SSL:10m; add_header Strict-Transport-
Security "max-age=63072000; includeSubDomains"; add_header X-Frame-
Options DENY; add_header X-Content-Type-Options nosniff;
ssl_stapling on; # Requires nginx >= 1.3.7
ssl_stapling_verify on; # Requires nginx => 1.3.7
resolver 8.8.4.4 8.8.8.8 valid=300s;
resolver_timeout 5s;
```

- **nginx/sites-available**: This contains a single file called *default* which provides the configuration for the default site. For our case, we have added these lines for configuring SSL.

  ```
  listen 443 ssl default_server;
  listen [::]:443 ssl default_server;
  include /etc/nginx/snippets/ssl.conf;
  ```

6. Building the Docker Image

Now that we have a Dockefile, we can build our image from it using the following command. It tells docker to build the image called *mynginx* with the tag *latest* and replace any old images with the freshly built one. The last argument tells docker to pick up relative file names from the current directory. Specifically, the directory contains an *nginx* directory with the above listed files.

```
docker build --rm=true --force-rm=true -t mynginx:latest .
```

After you run this command, docker goes through the process of downloading the base Ubuntu 16.04 image and configuring it. After it is done, it shows something like:

```
...
Removing intermediate container 79cb250e4a03
Successfully built 4620b73d1fcd
```

You can check that the image has been added to Docker using the command:

```
docker images
# prints
```

```
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
mynginx             latest              4620b73d1fcd        6 minutes
ago        212 MB
ubuntu              16.04               0458a4468cbc        5 weeks
ago          112 MB
```

To discard the image (for re-building):

```
docker rmi mynginx:latest
docker rmi ubuntu:16.04
```

For a complete cleanup of all intermediate images, use the following command:

```
if [[ $(docker images -f "dangling=true" -q | wc -l) -gt 0 ]]; then
docker rmi $(docker images -f "dangling=true" -q); fi
```

7. Running the Docker Image

Now that we have built our Docker Nginx image, we try to run it.

```
docker run -p 80:80 -p 443:443 -it mynginx:latest
```

This command run the *mynginx* image with the *latest* tag, and maps the exposed ports as follows: port 80 from the container is mapped to port 80 on the host, and likewise for port 443.

When this command completes successfully, you will see that the host is listening on ports 80 and 443. The requests are routed automatically to Nginx running inside the docker container.

```
netstat -an -t tcp | grep LISTEN
# prints
tcp6       0      0 :::80                   :::*
LISTEN
tcp6       0      0 :::443                  :::*
LISTEN
...
```

At this point, you can visit your site using a browser and it should just work. Check both *HTTP* and *HTTPS*.

7.1. Stopping the Container

To stop a container, you need the container id. Look it up by listing the containers.

```
docker ps
# prints
CONTAINER ID        IMAGE               COMMAND                  ...
aaebfd873d7a        mynginx:latest      "/usr/sbin/nginx -..."   ...
```

You can stop the docker container using:

```
docker stop aaebfd873d7a
```

7.3. Removing the Container

Remove the container from docker using the following command. Again you can lookup the container id if needed.

```
docker rm aaebfd873d7a
```

8. The Dockerfile

For reference, here is the complete Dockerfile.

```
FROM ubuntu:16.04
MAINTAINER Jack Black "jack.black@example.com"
RUN DEBIAN_FRONTEND=noninteractive apt-get update && \
    DEBIAN_FRONTEND=noninteractive apt-get -y dist-upgrade && \
    DEBIAN_FRONTEND=noninteractive apt-get -yq install net-tools nginx &&
\
    useradd -ms /bin/bash aurora && \
    rm -f /etc/nginx/fastcgi.conf /etc/nginx/fastcgi_params && \
    rm -f /etc/nginx/snippets/fastcgi-php.conf
/etc/nginx/snippets/snakeoil.conf
EXPOSE 80
EXPOSE 443
COPY nginx/ssl /etc/nginx/ssl
COPY nginx/snippets /etc/nginx/snippets
COPY nginx/sites-available /etc/nginx/sites-available
ENTRYPOINT ["/usr/sbin/nginx", "-g", "daemon off;"]
```

Conclusion

In this article, we learned how to install the Nginx web server in a Docker container and run it to turn the host into a web server. We have also made changes to Nginx configuration to SSL-enable it.

Jay Sridhar / March 2, 2018 / Sysadm / docker, nginx, ssl, web