NGINX

HOW I DOCKERISED MY BLOG

30TH OCTOBER 2017

SHARE ON

5 MIN READ

Docker is a fantastic tool and great if you're running a VPS like me. However a common question is... ***What if I want to be able to run multiple sites from a single VPS?***

Most websites running in containers listen on port 80 by default. Only one of them can be bound to that port at a time, so whats the answer?

Well, unless you want to access your websites using addresses such as `www.example.com:1234` . Then a ***reverse proxy*** is the answer.

Nginx will be our reverse proxy. It will take all incoming requests and route them to the correct container via Dockers `VIRTUAL_HOST` variable.

PREREQUISITES

You will need to install Docker and Docker Compose. Both have guides which I have linked below:

- Docker
- Docker Compose

NOTE: The Docker install guide is for Ubuntu but there are guides available for other Linux flavors as well as Windows and Mac.

DOCKER NETWORKS

Docker creates three default networks upon install. They are `bridge` , `none` and `host` . You can view the available Docker networks at any time using the command `docker network ls` .

If you would like to know more about Docker network please check out the official docs.

CREATING A DOCKER NETWORK

The first thing we are going to do is create a new Docker network for all of the containers to connect on. Type the command `docker network create nginx-proxy` .

When we start any containers we will be explicitly connecting them to this network. If we don't they won't be able to connect to each other, which would be bad.

THE THREE AMIGOS: NGINX, DOCKER-GEN & LETSENCRYPT

In this day and age we should all be running SSL enabled sites. And with the great work of LetsEncrypt there is no excuse not to go SSL.

With this in mind we are going to bring together three great tools to not only allow our sites to run https. But to also allow it to happen automatically.

To do this we will use Docker Compose. Which for those not aware, is a tool for defining and running multi-container solutions. We are going to use it to run these 3 images:

1. nginx

2. docker-gen
3. letsencrypt-nginx-proxy-companion

STEP 1

The first thing we are going to do is create some folders. I have setup the following structure but feel free to define your own.

```
  apps/
   - nginx-proxy/
     - files/
     - config/
```

The `files` folder is going to hold the nginx config files that will be generated by docker-gen when a new container is registered. As well as the certificates generated by LetsEncrypt.

Why do we need this you may ask?
As we are running containers now, all files within live and die with the container. We will be linking a directory in our container with the one above in order to persist our data files. We define these links as `volumes`. If we didn't do this, when we wanted to upgrade our images all our configuration files would be lost.

STEP 2

Now we have all our folders in place we need to add a few files.

Starting in the `apps/nginx-proxy/config/` folder run the following command:
`curl https://raw.githubusercontent.com/jwilder/nginx-proxy/master/nginx.tmpl > nginx.tmpl`

This is going to download the latest version of the `nginx.tmpl`, which is used to generate our nginx configs when registering new containers.

Next up is to create a `docker-compose.yml` in the same directory and paste in the following code:

```
version: '3'
services:
  nginx:
    image: nginx
    container_name: nginx
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /apps/nginx-proxy/files/conf.d:/etc/nginx/conf.d
      - /apps/nginx-proxy/files/vhost.d:/etc/nginx/vhost.d
      - /apps/nginx-proxy/files/html:/usr/share/nginx/html
      - /apps/nginx-proxy/files/certs:/etc/nginx/certs:ro

  nginx-gen:
    image: jwilder/docker-gen
    command: -notify-sighup nginx -watch -wait 5s:30s /etc/docker-gen/templates/nginx.tmpl /etc/
    container_name: nginx-gen
    restart: unless-stopped
    volumes:
      - /apps/nginx-proxy/files/conf.d:/etc/nginx/conf.d
      - /apps/nginx-proxy/files/vhost.d:/etc/nginx/vhost.d
      - /apps/nginx-proxy/files/html:/usr/share/nginx/html
      - /apps/nginx-proxy/files/certs:/etc/nginx/certs:ro
      - /var/run/docker.sock:/tmp/docker.sock:ro
      - ./nginx.tmpl:/etc/docker-gen/templates/nginx.tmpl:ro

  nginx-letsencrypt:
    image: jrcs/letsencrypt-nginx-proxy-companion
    container_name: nginx-letsencrypt
    restart: unless-stopped
    volumes:
      - /apps/nginx-proxy/files/conf.d:/etc/nginx/conf.d
      - /apps/nginx-proxy/files/vhost.d:/etc/nginx/vhost.d
      - /apps/nginx-proxy/files/html:/usr/share/nginx/html
      - /apps/nginx-proxy/files/certs:/etc/nginx/certs:rw
      - /var/run/docker.sock:/var/run/docker.sock:ro
    environment:
      NGINX_DOCKER_GEN_CONTAINER: "nginx-gen"
      NGINX_PROXY_CONTAINER: "nginx"

networks:
  default:
    external:
      name: nginx-proxy
```

STEP 3

With all the configuration files in place then we should be able to fire up our new Docker application using this command: `docker-compose up -d`

If all is well you should be able to run `docker ps` and see the 3 containers running.

REGISTERING CONTAINERS

Congratulations! You now have a reverse proxy all ready to go. But how do we register new containers with it?

I am going to show you how to register a new Ghost container as that is what I use for my blog. But the general principals should apply to most containerised applications.

STEP 4

To start I'm going to add a couple of new folders to the previous structure.

```
   apps/
    - nginx-proxy/
      - files/
      - config/
    - blog/
      - content/
```

I'm then going to add a new file, `blog.yml`. And paste in the following code.

```
version: '3'

services:

   blog:
     image: ghost
     container_name: blog
     restart: unless-stopped
     expose:
           - "443"
     volumes:
       - /apps/blog/content:/var/lib/ghost/content
     environment:
       VIRTUAL_HOST: myblog.com,www.myblog.com
       VIRTUAL_PORT: 2368
       LETSENCRYPT_HOST: myblog.com,www.myblog.com
       LETSENCRYPT_EMAIL: me@myblog.com

networks:
    default:
       external:
         name: nginx-proxy
```

Much like before data needs to be persisted outside of the container. And as before we achieve this by defining `volumes` .

I have defined a few `environment` variables and these are the key to the automatic registration with out proxy.

`VIRTUAL_HOST`
Our `docker-gen` container will watch for containers started with this variable set and will automatically generate a configuration for them in nginx.

`VIRTUAL_PORT`
This port will be registered with nginx so it know what port to connect to the container on.

`LETSENCRYPT_HOST` & `LETSENCRYPT_EMAIL`
These two variables are used by the `jrcs/letsencrypt-nginx-proxy-companion` container to either create or renew the SSL certificate for the container.

You'll also notice that once again I have defined the Docker network that the container should connect on.

STEP 5

All thats left to do is run `docker-compose up -d` followed by `docker ps` . And you should now see a ghost container running along with our previous 3 containers.

You should now be able to open a browser and type in the blog containers address and view your shiny new dockerised blog, or whatever container you started.

SUMMING UP

In 5 steps we have created a reverse proxy that we can auto register new containers with SSL. And we have proven it by registering our first container. Not bad if I do say so myself.

I've included the links to all the projects used throughout this post. I did just want to mention docker-compose-letsencrypt-nginx-proxy-companion by Evert Ramos. This was very helpful with getting my setup working. And while I have modified it a bit for my uses the bulk of credit should go to him.

SHARE ON