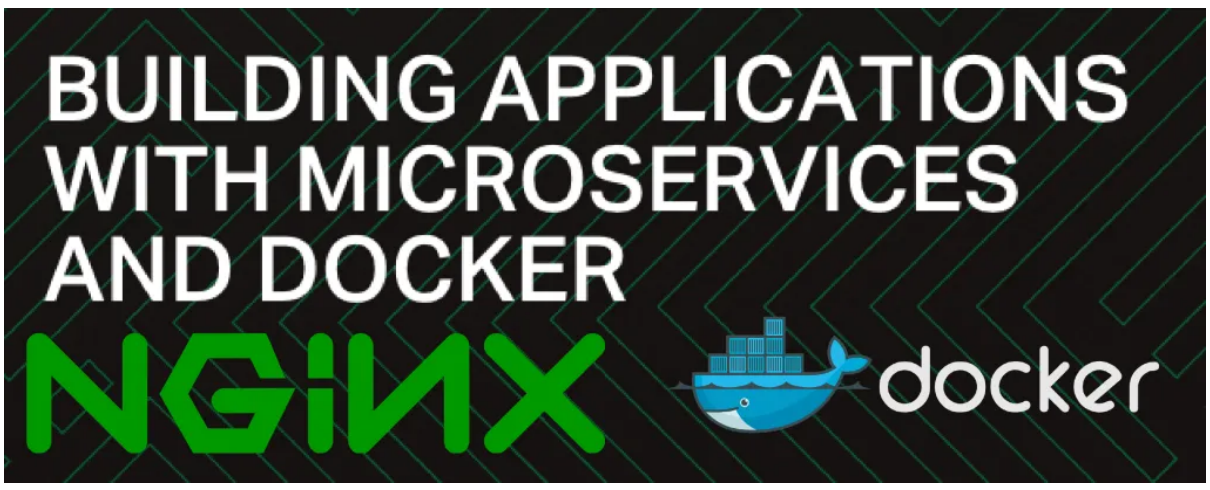
Tips for Deploying NGINX (Official Image) with Docker

By Mario Ponticello *April 21 2015*

container, Logging, microservices, nginx, Official image

In the past year alone, the Docker community has created 100,000+ images and over 300+ million images have been pulled from Docker Hub to date. Over 20 million of these pulls came from the 70+ Official Images that Docker develops in conjunction with upstream partners, like Oracle, CentOS, and NGINX.

NGINX is used by over 40% of the world's busiest websites and is an open-source reverse proxy server, load balancer, HTTP cache, and web server. The official image on Docker Hub has been pulled over 3.4 million times and is maintained by the NGINX team. This post from NGINX provides a walkthrough on using the Docker Image to deploy the open-source version of NGINX.



Attend the webinar titled "Building Applications with Microservices and Docker"

NGINX Solutions Architect Rick Nelson and Docker Engineer Jerome Petazzoni discuss the best approach to developing applications with microservices.

Date: April 22rd

Time: 10:00 am PDT / 1:00 pm EDT

Here is a Tutorial on How to use the NGINX Official Docker Image

To create an instance of NGINX in a Docker container, search for and pull the NGINX official image from Docker Hub. Use the following command to launch an instance of NGINX running in a container and using the default configuration.

```
 # docker run --name mynginx1 -P -d nginx

fcd1fb01b14557c7c9d991238f2558ae2704d129cf9fb97bb4fadf673a58580d
```

This command creates a container named mynginx1 based on the NGINX image and runs it in detached mode, meaning the container is started and stays running until stopped but does not listen to the command line. We will

discuss later how to interact with the container.

The NGINX image exposes ports 80 and 443 in the container and the `-P` option tells Docker to map those ports to ports on the Docker host that are randomly selected from the range between 49153 and 65535. We do this because if we create multiple NGINX containers on the same Docker host, we create a conflict on ports 80 and

443. The port mappings are dynamic and are set each time the container is started or restarted. If you want the port mappings to be static, set them manually with the `-p` option. The long form of the Container Id will be returned.

We can run `docker ps` to verify that the container was created and is running, and to see the port mappings:

```
 # docker ps

CONTAINER ID  IMAGE         COMMAND               CREATED         STATUS

PORTS                                     NAMES

fcd1fb01b145  nginx:latest  "nginx -g 'daemon of  16 seconds ago  Up 15 seconds

0.0.0.0:49166->443/tcp, 0.0.0.0:49167->80/tcp mynginx1
```

We can also verify that NGINX is running by making an HTTP request to port 49167 (reported in the output from the preceding command as the port on the Docker host that is mapped to port 80 in the container); the default NGINX welcome page appears:

```
 # curl http://localhost:49167

<!DOCTYPE html>

<html>

<head>

<title>Welcome to nginx!</title>

<style>

body {

width: 35em;

margin: 0 auto;

font-family: Tahoma, Verdana, Arial, sans-serif;

}

</style>

</head>

<body>

<h1>Welcome to nginx!</h1>

<p>If you see this page, the nginx web server is successfully installed and working.
Further configuration is required.</p>

<p>For online documentation and support please refer to

<a href="http://nginx.org/">nginx.org</a>.<br/>

Commercial support is available at

<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>

</body>

</html>
```

Working with the NGINX Docker Container

Now that we have a working NGINX Docker container, how do we manage the content and the NGINX configuration? And what about logging? It is common to have SSH access to NGINX instances, but Docker containers are generally intended to be for a single purpose (in this case running NGINX) so the NGINX image does not have OpenSSH installed and for normal operations there is no need to get shell access directly to the NGINX container. We will use other methods supported by Docker. For a detailed discussion of alternatives to

SSH access, see Why you don't need to run SSHd in your Docker Containers.

Managing Content and Configuration Files

There are multiple ways you can manage the NGINX content and configuration files and we will cover a few options:

Maintain the Content and Configuration on the Docker Host

When the container is created we can tell Docker to mount a local directory on the Docker host to a directory in the container. The NGINX image uses the default NGINX configuration, so the root directory for the container is/usr/share/nginx/html and the configuration files are in /etc/nginx. If the content on the Docker host is in the local directory /var/www and the configuration files are in /var/nginx/conf, we run the command:

```
# docker run --name mynginx2 -v /var/www:/usr/share/nginx/html:ro \

-v /var/nginx/conf:/etc/nginx:ro -P -d nginx
```

Now any change made to the files in the local directories /var/www and /var/nginx/conf on the Docker host are reflected in the directories /usr/share/nginx/html and /etc/nginx in the container. The `:ro` option causes these directors to be read only inside the container.

Copy the Files from the Docker Host

Another option is to have Docker copy the content and configuration files from a local directory on the Docker host when a container is created. Once a container is created, the files are maintained by creating a new container when the files change or by modifying the files in the container. A simple way to copy the files is to create a Dockerfile to generate a new Docker image, based on the NGINX image from Docker Hub. When copying files in the Dockerfile, the path to the local directory is relative to the build context where the Dockerfile is located. For this example, the content is in the content directory and the configuration files are in the conf directory, both in the same directory as theDockerfile. The NGINX image has the default NGINX configuration files, including default.conf and example_ssl.conf in/etc/nginx/conf.d. Since we want to use the configuration files from the host, we include commands in the followingDockerfile to delete the default files:

```
FROM nginx

RUN rm /etc/nginx/conf.d/default.conf

RUN rm /etc/nginx/conf.d/examplessl.conf

COPY content /usr/share/nginx/html

COPY conf /etc/nginx
```

We can then create our own NGINX image by running the following command from the directory where the Dockerfileis located:

```
# docker build -t mynginximage1.
```

Note the period (".") at the end of the command. This tells Docker that the build context is the current directory. The build context contains the Dockerfile and the directories to be copied. Now we can create a container using the image by running the command:

```
# docker run --name mynginx3 -P -d mynginximage1
```

If we want to make changes to the files in the container, we use a helper container as described below.

Maintain Files in the Container

As mentioned previously, we are not able to get SSH access to the NGINX container, so if we want to edit the content or configuration files directly we can use a helper container that has shell access. In order for the helper container to have access to the files, we must create a new image that has the proper volumes specified for the image. Assuming we want to copy the files as in the example above, while also specifying volumes for the content and configuration files, we use the following Dockerfile:

```
FROM nginx

 COPY content /usr/share/nginx/html

 COPY conf /etc/nginx

 VOLUME /usr/share/nginx/html

 VOLUME /etc/nginx
```

We then create the new NGINX image by running the following command (again note the final period):

```
# docker build -t mynginximage2 .
```

Now we create an NGINX container using the image by running the command:

```
# docker run --name mynginx4 -P -d mynginximage2
```

We then start a helper container with a shell and access the content and configuration directories of the NGINX container we created in the previous example by running the command:

```
# docker run -i -t --volumes-from mynginx4 --name mynginx4files debian /bin/bash
```

```
 root@b1cbbad63dd1:/#
```

This creates an interactive container named mynginx4_files that runs in the foreground with a persistent standard input (-i) and a tty (-t) and mounts all the volumes defined in the container mynginx4 as local directories in the newmynginx4_files container. This container uses the Debian image from Docker Hub, which is the same operating system used by the NGINX image. Since all of the examples shown so far use the NGINX image and therefore Debian, it is more efficient to use Debian for the helper container rather then having Docker load another operating system. After the container is created, it runs the bash shell, which presents a shell prompt for the container that you can use to modify the files as needed. You can also install other tools, such as Puppet or Chef, in the container to manage these files. If you exit the shell by running the `exit` command, the container

terminates. If you want to exit while leaving the container running, use `Control-p` followed by `Control-q`. The container can be started and stopped with the following commands:

```
# docker start mynginx4files
```

and

```
# docker stop mynginx4files
```

Shell access can be regained to a running container with the command:

```
# docker attach mynginx4files
```

Managing Logging

Default Logging

The NGINX image is configured to send the main NGINX access and error logs to the Docker log collector by default. This is done by linking them to stdout and stderr, which causes all messages from both logs to be stored in the file/var/lib/docker/containers/\<container id\>/\<container id\>-json.log on the Docker Host. \<container id\> is the long-form Container Id returned when you create a container. You can display the long-form Id for a container with the command:

```
# docker inspect --format '{{ .Id }}' <container name>
```

You can use both the Docker command line and the Docker Remote API to extract the log messages. From the command line run the command:

```
# docker logs <container name>
```

To enable the Docker Remote API, add the following line to the file /etc/default/docker:

```
DOCKEROPTS='-H tcp://0.0.0.0:4243 -H unix:///var/run/docker.sock'
```

When Docker is restarted, it listens for HTTP API requests on port 4243 (you can specify a different port) and also on a socket. To get all the messages, you can issue the GET request:

```
http://<docker host>:4243/containers/<container name>/logs?stdout=1&stderr=1
```

To include only access log messages in the output, include only `stdout=1` ; to limit the output to error log messages, include only `stderr=1` . To learn about other available options, see the Docker documentation.

Customized Logging

If you want to implement another method of log collection, or if you want to configure logging differently at various levels in the NGINX configuration (such as servers and locations), you can use a volume for the directory or directories in which to store the log files in the container. You can then use a helper container to access the log files and use whatever logging tools you like. To implement this, create a new image that contains the volume or volumes for the logging files. For example, to configure NGINX to store log files in /var/log/nginx/log, we start with the Dockerfile shown in the earlier example of copying files from the Docker host to the container and simply add a volume declaration for this directory:

```
FROM nginx
COPY content /usr/share/nginx/html
COPY conf /etc/nginx
VOLUME /var/log/nginx/log
```

We can then create an image as described above and using this image create an NGINX container and a helper container that have access to the log directory. This helper container can have any desired logging tools installed.

Controlling NGINX

Since we do not have access to the command line of the NGINX container directly, we cannot use the `nginx` command to control NGINX. Fortunately NGINX can be controlled by signals and Docker provides `kill` command for sending signals to a container. For example, to reload the NGINX configuration run the command:

```
docker kill -s HUP <container name>
```

If you want to restart the NGINX process, restart the container by running the command:

```
docker restart <container name>
```

Summary

NGINX and Docker are tools that work extremely well together. By using the NGINX open-source image from the Docker Hub repository, you can easily spin up new instances of NGINX in Docker containers. You can also take these images and easily create new Docker images from them to give you even more control of your containers and how you manage them.

Learn more about NGINX by visiting their website and download the official image from Docker Hub today.

Learn More about Docker

- New to Docker? Try our 10 min online tutorial
- Share images, automate builds, and more with a free Docker Hub account
- Read the Docker 1.6 Release Notes
- Subscribe to Docker Weekly
- Attend upcoming Docker Meetups
- Attend upcoming Docker Online Meetups
- Register for DockerCon 2015
- Start contributing to Docker

Tips for Deploying NGINX (Official Image) with Docker

By Mario Ponticello

Related Posts

Docker Enterprise: The First DISA STIG'ed Container Platform!

By Andy Clemenko October 08 2019

Top Questions Answered: Docker and Kubernetes? I Thought You Were Competitors!

By Jim Armstrong October 07 2019

Top Questions: Containers and VMs Together

By Jim Armstrong October 01 2019

Feedback

14 thoughts on "Tips for Deploying NGINX (Official Image) with Docker"

Jean-Francois Nadeau

April 21, 2015 at 11:39 am

Our web site code is open on GitHub and provides an example of building a new container from the nginx base image & deployment on CoreOS. If you're interested, here is the github repo: https://github.com/mailsquad/website

Reply

James Kitto

June 24, 2015 at 2:42 pm

Very useful guide, however I was thrown at first where it says “Note the period (“.”) at the end of the command.” as the command provided is actually missing the period.

Reply

Adam Herzog

March 3, 2016 at 1:30 pm

Just fixed! Thanks for letting us know

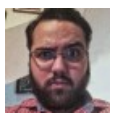
Reply

Rob Wyrick

June 18, 2017 at 11:53 pm

But the command is still wrong… there should be a space before the period, no?

Reply



Eduardo

July 28, 2017 at 1:56 pm

Yes, it should have a space before the period. Indicating the current folder.

docker build -t mynginximage1 .

Seems like it's a typo, because you can see the same command after the incorrect one but without the error.

Reply

Utah Sites

February 1, 2016 at 12:15 am

This tutorial is indeed a big help for me personally. Glad to know everything about it now after spending time researching about it. I know a lot better now because of the step by step tutorial you’ve shared. And thanks for giving some input and tips as well for deploying images with docker.

Reply

Amandeep

March 31, 2016 at 10:53 am

Hi,
I have query regarding helper container.
I have created a nginx container from custom docker image exposing port 80 and 443 and few volumes also and one helper container named nginx4files where nginx.conf is mounted automatically.
Problem statement -: from nginx4files container, i have modified the port of nginx from 80 to 8085 and reload the nginx configuration using nginx -s reload command. I checked the nginx.conf file in nginx container which is getting updated as well but when i am running docker ps the changed port (80 to 8085)is not getting mapped to host dynamic port. Therefore, even container is running, i am not able to access nginx landing page from host system. Please advise, do i need to stop the nginx container to reflect the configuration of i am missing something here.

Thanks
Amandeep

Reply

Travis Runyard

February 21, 2017 at 9:55 am

What is the general consensus? Do people keep their configs in a central location and update them to update all containers that read them or the other method mentioned?

Reply

SEO National

April 30, 2017 at 8:42 pm

This tutorial is very a big help. Through this we can then create an image as described in this page and using this image create an NGINX container and a helper container that have access to the log directory.

Reply

jeremy rutman

June 4, 2017 at 1:55 am

I followed this blog's instructions but wind up with a server that shows only the 'wecome to nginx ' splash page – no changes to /var/www seem to get seen . I used docker run –name mynginx2 -v /data/www:/usr/share/nginx/html:ro -v /var/nginx/conf:/etc/nginx:ro -p8090:80 -d nginx and also without -d flag to no avail (my web content is in /data/www)

Reply

Dean Christian Armada

July 22, 2017 at 8:34 pm

restarting the container is not advisable when you initialize Docker Swarm because it may remove the nginx service. So if you need an alternative aside docker restart; You can go inside the container and just run nginx -s reload

Reply

prachi

August 22, 2017 at 2:59 am

Can we configure nginx to process header Expect

Reply

Ram

June 2, 2018 at 11:01 pm

Running it on mac:
docker run –name mynginx2 -v /tmp/var/www:/usr/share/nginx/html:ro \
-v /tmp/var/nginx/conf:/etc/nginx:ro -P -d nginx

my data is in /tmp/var/www and nginx conf in /tmp/var/nginx
when i try to run curl I am getting 403 error

Reply

Ram Dhakne

June 2, 2018 at 11:07 pm

Running on mac

docker run –name mynginx2 -v /tmp/var/www:/usr/share/nginx/html:ro \
-v /tmp/var/nginx/conf:/etc/nginx:ro -P -d nginx

data in /tmp/var/www
and conf in /tmp/var/nginx/conf

getting 403 when trying to access the nginx web UI

✗ ram.dhakne@Rams-MBP □ /tmp/var □ curl http://localhost:32770/
<html>
<head><title>403 Forbidden</title></head>
<body bgcolor="white">
<h1>403 Forbidden</h1>
<hr> nginx/1.13.12
</body>
</html>

Reply

Leave a Reply

Your name (required)

Your name

Your email address(required, but will not be published)

Your email address

Your website if you have one (not required)

Your website url

Your comment

Your comment

Post comment

Sign up for our newsletter.

Company Email

Select Country...

Submit

docker

Products

Product Overview

Product Offerings

Docker Enterprise
Docker Hub

Product Solutions

Docker Enterprise Solutions

Technologies

Developer Tools
Desktop
Container Runtime
Kubernetes
Docker App
Image Registry
Container Management

Solutions

Use Cases

Modern Apps
Microservices
CI/CD
Big Data/Data Science
Edge Computing
Cloud Migration
Digital Transformation
Windows Server Migration

Industry

Government
Higher Education
Financial Services
Insurance
Healthcare/Pharma

Customers

Case Studies

Company

About Us
What is a Container
Containerization Strategy
Management
Newsroom
Careers
Partners
Contact Us

Resources

Learn

Content Library
Docker Blog
Community
Open Source

Attend

DockerCon
Events and Webinars

Tools

Documentation
Training
Customer Success
Engineering Blog