NGINX

BLOG TECH Rick Nelson of NGINX, Inc. October 22, 2014

Deploying NGINX and NGINX Plus with Docker

NGINX Plus, Docker, containers, CentOS

Note*: This post was updated in November 2017 to make the Docker commands comply with current standards and to provide a better and safer-to-use NGINX Plus Dockerfile.*

Docker is an open platform for building, shipping, running, and orchestrating distributed applications. As software applications, the open source NGINX product and the enhanced and commercially supported version, NGINX Plus, are great use cases for Docker, and we make an NGINX image available on Docker Hub, the repository of Docker images. This article describes how you can deploy the open source NGINX software using this image from Docker Hub, or create and deploy your own Docker image of NGINX Plus.

Introduction

The Docker open platform includes the Docker Engine – the open source runtime that builds, runs, and orchestrates containers – and Docker Hub, a hosted service where Dockerized applications are distributed, shared, and collaborated on by the entire development community or within the confines of a specific organization.

Docker containers enable developers to focus their efforts on application “content” by separating applications from the constraints of infrastructure. Dockerized applications are instantly portable to any infrastructure – laptop, bare-metal server, VM, or cloud – making them modular components that can be readily assembled and reassembled into fully featured distributed applications and continuously innovated on in real time.

For more information about Docker, see What is Docker? or the full Docker documentation.

Using the NGINX Docker Image

You can create an NGINX instance in a Docker container using the NGINX image from Docker Hub.

Let’s start with a very simple example. To launch an instance of NGINX running in a container and using the default NGINX configuration, run this command:

```
# docker run --name mynginx1 -p 80:80 -d nginx
fcd1fb01b14557c7c9d991238f2558ae2704d129cf9fb97bb4fadf673a58580d
```

This command creates a container named **mynginx1** based on the NGINX image. The command returns the long form of the container ID, which is used in the name of log files; see Managing Logging.

The `-d` option specifies that the container runs in detached mode, which means that it continues to run until stopped but does not respond to commands run on the command line. We discuss later how to interact with the container.

The `-p` option tells Docker to map the ports exposed in the container by the NGINX image – port 80 – to the specified port on the Docker host. The first parameter specifies the port in the Docker host, while the second parameter is mapped to the port exposed in the container.

To verify that the container was created and is running, and to see the port mappings, we run `docker ps`. (We've split the output across multiple lines here to make it easier to read.)

```
# docker ps
CONTAINER ID  IMAGE         COMMAND               CREATED
STATUS        ...
fcd1fb01b145  nginx:latest  "nginx -g 'daemon of  16 seconds ago  Up
15 seconds ...

    ... PORTS              NAMES
    ... 0.0.0.0:80->80/tcp mynginx1
```

The `PORTS` field in the output reports that port 80 on the Docker host is mapped to port 80 in the container. Another way to verify that NGINX is running is to make an HTTP request to that port. The code for 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="https://www.nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Working with the NGINX Docker Container

So now we have a working NGINX Docker container, but how do we manage the content and the NGINX configuration? And what about logging?

A Note About SSH

It is common to enable SSH access to NGINX instances, but the NGINX image does not have OpenSSH installed, because Docker containers are generally intended to be for a single purpose (in this case running NGINX), and for normal operations there is no need to have shell access directly to the NGINX container. Instead we'll 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 several ways you can manage both the content served by NGINX and the NGINX configuration files. Here we cover a few of the options.

Option 1 – 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, which uses **/usr/share/nginx/html** as the container's root directory and puts configuration files in **/etc/nginx**. For a Docker host with content in the local directory **/var/www** and configuration files in **/var/nginx/conf**, run this command (which appears on two lines here only for legibility):

```
# docker run --name mynginx2 --mount type=bind
source=/var/www,target=/usr/share/nginx/html,readonly --mount
source=/var/nginx/conf,target=/etc/nginx/conf,readonly -p 80:80 -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 `readonly` option means these directories can be changed only on the Docker host, not from within the container.

Option 2 – Copy 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 during container creation. Once a container is created, the files are maintained by creating a new container when files change or by modifying the files in the container. A simple way to copy the files is to create a **Dockerfile** with commands that are run during generation of a new Docker image based on the NGINX image from Docker Hub. For the file-copy (`COPY`) commands in the **Dockerfile**, the local directory path is relative to the build context where the **Dockerfile** is located.

In our example, the content is in the **content** directory and the configuration files are in the **conf** directory, both subdirectories of the directory where the **Dockerfile** is located. The NGINX image has the default NGINX configuration files, including **default.conf**, in the **/etc/nginx/conf.d** directory. Since we instead want to use the configuration files from the host, we include `RUN` commands that delete the default files:

```
FROM nginx
RUN rm /etc/nginx/conf.d/default.conf
COPY content /usr/share/nginx/html
COPY conf /etc/nginx
```

We create our own NGINX image by running the following command from the directory where the **Dockerfile** is located. Note the period (".") at the end of the command. It defines the current directory as the build context, which contains the **Dockerfile** and the directories to be copied.

```
# docker build -t mynginx_image1 .
```

Now we run this command to create a container called **mynginx3** based on the **mynginx_image1** image:

```
# docker run --name mynginx3 -p 80:80 -d mynginx_image1
```

If we want to make changes to the files in the container, we use a helper container as described in Option 3.

Option 3 – Maintain Files in the Container

As mentioned in A Note About SSH, we can't use SSH to access the NGINX container, so if we want to edit the content or configuration files directly we have to create a helper container that has shell access. For the helper container to have access to the files, we must create a new image that has the proper Docker data volumes defined for the image. Assuming we want to copy files as in Option 2 while also defining volumes, 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 mynginx_image2 .
```

Now we run this command to create an NGINX container (**mynginx4**) based on the **mynginx_image2** image:

```
# docker run --name mynginx4 -p 80:80 -d mynginx_image2
```

We then run the following command to start a helper container **mynginx4_files** that has a shell, enabling us to access the content and configuration directories of the **mynginx4** container we just created:

```
# docker run -i -t --volumes-from mynginx4 --name mynginx4_files
debian /bin/bash
root@b1cbbad63dd1:/#
```

The new **mynginx4_files** helper container runs in the foreground with a persistent standard input (the `-i` option) and a tty (the `-t` option). All volumes defined in **mynginx4** are mounted as local directories in the helper container.

The `debian` argument means that the helper container uses the Debian image from Docker Hub. Because the NGINX image also uses Debian (and all of our examples so far use the NGINX image), it is most efficient to use Debian for the helper container, rather than having Docker load another operating system. The `/bin/bash` argument means that the `bash` shell runs in the helper container, presenting a shell prompt that you can use to modify files as needed.

To start and stop the container, run the following commands:

```
# docker start mynginx4_files
# docker stop mynginx4_files
```

To exit the shell but leave the container running, press `Ctrl+p` followed by `Ctrl+q`. To regain shell access to a running container, run this command:

```
# docker attach mynginx4_files
```

To exit the shell and terminate the container, run the `exit` command.

Managing Logging

You can configure either default or customized logging.

Using 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`; all messages from both logs are then written to the file **/var/lib/docker/containers/*container-ID*/*container-ID*-json.log** on the Docker host. The ***container-ID*** is the long-form ID returned when you create a container. To display it, run this command:

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

You can use both the Docker command line and the Docker Engine API to extract the log messages.

On the command line, run this command:

```
# docker logs container-name
```

To use the Docker Remote API, issue a `GET` request using the Docker Unix sock:

```
curl --unix-sock /var/run/docker-sock
http://localhost/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 Engine API documentation (search for “Get container logs” on that page).

Using Customized Logging

If you want to implement another method of log collection, or if you want to configure logging differently in certain configuration blocks (such as `server{}` and `location{}`), define a Docker volume for the directory or directories in which to store the log files in the container, create 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 can start with the **Dockerfile** from Option 3 and simply add a `VOLUME` definition 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 use it to create an NGINX container and a helper container that have access to the logging directory. The helper container can have any desired logging tools installed.

Controlling NGINX

Since we do not have direct access to the command line of the NGINX container, we cannot use the `nginx` command to control NGINX. Fortunately we can use signals to control NGINX, and Docker provides the `kill` command for sending signals to a container.

To reload the NGINX configuration, run this command:

```
# docker kill -s HUP container-name
```

To restart NGINX, run this command to restart the container:

```
# docker restart container-name
```

Deploying NGINX Plus with Docker

So far we have discussed Docker for the open source NGINX software, but you can also use it with the commercial product, NGINX Plus. The difference is you first need to create an NGINX Plus image, because as a commercial offering NGINX Plus is not available at Docker Hub. Fortunately, this is quite easy to do.

Note: Never upload your NGINX Plus images to a public repository such as Docker Hub. Doing so violates your license agreement.

Creating a Docker Image of NGINX Plus

To generate an NGINX Plus image, first create a **Dockerfile**. The example we provide here uses Debian 9 (Stretch) as the base Docker image. Before you can create the NGINX Plus Docker image, you have to download your version of the **nginx-repo.crt** and **nginx-repo.key** files. NGINX Plus customers can find them at the customer portal, **https://cs.nginx.com**; if you are doing a free trial of NGINX Plus, they were provided with your trial package. Copy the files to the directory where the **Dockerfile** is located (the Docker build context).

As with open source NGINX, by default the NGINX Plus access and error logs are linked to the Docker log collector. No volumes are specified, but you can add them if desired, or each **Dockerfile** can be used to create base images from which you can create new images with volumes specified, as described previously.

By default, no files are copied from the Docker host as a container is created. You can add `COPY` definitions to each **Dockerfile**, or the image you create can be used as the basis for another image as described above.

NGINX Plus Dockerfile

```
#For Debian 9
FROM debian:stretch-slim

LABEL maintainer="NGINX Docker Maintainers <docker-maint@nginx.com>"

# Download certificate and key from the customer portal (https://cs.nginx.com)
# and copy to the build context
COPY nginx-repo.crt /etc/ssl/nginx/
COPY nginx-repo.key /etc/ssl/nginx/

# Install NGINX Plus
RUN set -x \
  && apt-get update && apt-get upgrade -y \
  && apt-get install --no-install-recommends --no-install-suggests -y apt-transport-https
  && \
  NGINX_GPGKEY=573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62; \
  found=''; \
  for server in \
    ha.pool.sks-keyservers.net \
    hkp://keyserver.ubuntu.com:80 \
    hkp://p80.pool.sks-keyservers.net:80 \
    pgp.mit.edu \
  ; do \
    echo "Fetching GPG key $NGINX_GPGKEY from $server"; \
    apt-key adv --keyserver "$server" --keyserver-options timeout=10 --recv-keys "$NGINX_G
  done; \
  test -z "$found" && echo >&2 "error: failed to fetch GPG key $NGINX_GPGKEY" && exit 1; \
  echo "Acquire::https::plus-pkgs.nginx.com::Verify-Peer \"true\";" >> /etc/apt/apt.conf.d
  && echo "Acquire::https::plus-pkgs.nginx.com::Verify-Host \"true\";" >> /etc/apt/apt.con
  && echo "Acquire::https::plus-pkgs.nginx.com::SslCert     \"/etc/ssl/nginx/nginx-repo.cr
  && echo "Acquire::https::plus-pkgs.nginx.com::SslKey      \"/etc/ssl/nginx/nginx-repo.ke
  && printf "deb https://plus-pkgs.nginx.com/debian stretch nginx-plus\n" > /etc/apt/sourc
  && apt-get update && apt-get install -y nginx-plus \
  && apt-get remove --purge --auto-remove -y gnupg1 \
  && rm -rf /var/lib/apt/lists/*

# Forward request logs to Docker log collector
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
  && ln -sf /dev/stderr /var/log/nginx/error.log

EXPOSE 80

STOPSIGNAL SIGTERM

CMD ["nginx", "-g", "daemon off;"]
```

Dockerfile hosted with ❤ by GitHub view raw

Creating the NGINX Plus Image

With the **Dockerfile**, **nginx-repo.crt**, and **nginx-repo.key** files in the same directory, run the following command there to create a Docker image called **nginxplus** (as before, note the final period):

```
# docker build --no-cache -t nginxplus .
```

Note the `--no-cache` option, which tells Docker to build the image from scratch and ensures the installation of the latest version of NGINX Plus. If the **Dockerfile** was previously used to build an image and you do not include the `--no-cache` option, the new image uses the version of NGINX Plus from the Docker cache. (We purposely do not specify a version in the **Dockerfile** so that the file does not need to change at every new release of NGINX Plus.) Omit the `--no-cache` option if it's acceptable to use the NGINX Plus version from the previously built image.

Output like the following from the `docker images nginxplus` command indicates that the image was created successfully:

```
# docker images nginxplus
REPOSITORY  TAG     IMAGE ID      CREATED        VIRTUAL SIZE
nginxplus   latest  ef2bf65931cf  6 seconds ago  91.2 MB
```

To create a container named **mynginxplus** based on this image, run this command:

```
# docker run --name mynginxplus -p 80:80 -d nginxplus
```

You can control and manage NGINX Plus containers in the same way as NGINX containers.

Summary

NGINX, NGINX Plus, and Docker work extremely well together. Whether you use the open source NGINX image from the Docker Hub repository or create your own NGINX Plus image, you can easily spin up new instances of NGINX and NGINX Plus in Docker containers. You can also easily create new Docker images from the base images, making your containers even easier to control and manage. Make sure that all NGINX Plus instances running in your Docker containers are covered by your subscription. For details, please contact the NGINX sales team.

There is much more to Docker than we have been able to cover in this article. More information is available at www.docker.com.

12 Comments NGINX 1 Login

Recommend Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

Bradley Cummins • 3 years ago

Is this statement still correct?

"Since we do not have direct access to the command line of the NGINX container"

You can access the nginx container with

# docker exec -it my-nginx bash

Once inside you can gracefully restart using

# kill -HUP `cat /var/run/nginx.pid`

7 • Reply • Share ›

Faisal Memon Mod Bradley Cummins • 3 years ago

You definitely can shell into a container, but you don't need to just to restart the nginx process. Good post on this related topic: https://jpetazzo.github.io/...

• Reply • Share ›

benneil • 2 years ago

Are their any plans on havng the nginx-plus package available on alpine linux?

1 • Reply • Share ›

Faisal Memon Mod benneil • 2 years ago

Thanks for the feedback, we've gotten that request and we are evaluating it.

• Reply • Share ›

grosser • 7 months ago

Shouldn't it be `STOPSIGNAL SIGQUIT` ? ... or at least add a comment saying "Use SIGQUIT here to gracefully shut down" here to gracefully shut down

Reply • Share ›

Mayank Patel • a year ago

Can Nginx Team not provide Nginx Plus official image with different Options like Alpine, Ubuntu, CentOS, etc? Image could ask user to pass in there CRT and KEY file.
Also, to pass in volume for all the configuration files. If there is official Nginx Plus image available and I am not aware then please guide.

Reply • Share ›

acheshkov • 2 years ago

Hi, can i use different port then 80 or 443
to run nginx inside nginx:latest docker image ?
it looks like all other ports a closed

Reply • Share ›

> **Faisal Memon** Mod → acheshkov • 2 years ago
>
> You should be able to override the EXPOSE directive and list additional ports there.
>
> Reply • Share ›

Justin Morse • 2 years ago

So how would I go about creating a working directory in Windows? I get errors against valid repository as well as invalid volume specification if I try and sneak C:\, //C:/, //C:\, etc.

Reply • Share ›

Joshua T Kalis • 2 years ago

This was a great help. Thank you.

Please pardon the n00b question but, what next then? How do I use it?

Reply • Share ›

И.К. • 3 years ago

Hi.
Here is my issue.
Can you please help?
Windows 8.1

PS C:\Users\Developer\desktop\testdocker> docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
5de9db992795 nginx "nginx -g 'daemon off" 15 seconds ago Up 7 seconds 0.0.0.0:32773->
80/tcp, 0.0.0.0:32772->443/tcp jovial_goodall
PS C:\Users\Developer\desktop\testdocker> curl http://localhost:32773
curl : can't connect to remote server
line:1 item:1

+ curl http://localhost:32773

+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~

+ CategoryInfo : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException

+ FullyQualifiedErrorId :
WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestComn

Thanks

Reply • Share ›

> **И.К.** → И.К. • 3 years ago
>
> nvm. I was trying to connect to the docker from outside the docker-machine.
>
> Reply • Share ›

Subscribe Add Disqus to your siteAdd DisqusAdd Disqus' Privacy PolicyPrivacy PolicyPrivacy

TRY NGINX PLUS FOR FREE

ASK US A QUESTION

Products

NGINX Plus
NGINX Controller
NGINX Unit
NGINX Amplify
NGINX Web Application Firewall

NGINX on Github

NGINX Open Source
NGINX Unit
NGINX Amplify
NGINX Kubernetes Ingress Controller
nginMesh
NGINX Microservices Reference Architecture
NGINX Crossplane

Solutions

ADC / Load Balancing
Microservices
Cloud
Security
Web & Mobile Performance
API Gateway

Resources

Documentation
Ebooks
Webinars
Datasheets
Success stories
Blog
FAQ
Learn
Glossary

Support

Professional Services
Training
Customer Portal Login

Partners

Amazon Web Services
Google Cloud Platform
IBM
Microsoft Azure
Red Hat
Find a Partner
Certified module program

Company

About NGINX
Careers
Leadership
Press
Events

Connect With Us

STAY IN THE LOOP