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

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: http://faq.asphosthelpdesk.com/article.php?id=257

Docker in Brief

The **docker project** offers higher-level tools, working together, which are built on top of some Linux kernel features. The goal is to help developers and system administrators port applications - with all of their dependencies conjointly - and get them running across systems and machines -*headache free*.

Docker achieves this by creating safe, LXC (i.e. Linux Containers) based environments for applications called "docker containersâ€•. These containers are created using docker images, which can be built either by executing commands manually or automatically through **Dockerfiles**.

Nginx in Brief

Nginx is a very high performant web server / (reverse)-proxy). It has reached its popularity due to being light weight, relatively easy to work with, and easy to extend (with add-ons / plug-ins). Thanks to its architecture, it is capable of handling*a lot*of requests (virtually unlimited), which - depending on your application or website load - could be really hard to tackle using older alternatives. It can be considered*the*tool to choose for serving static files such as images, scripts or style-sheets.

Installing Docker on Ubuntu (Latest)

With its most recent release (**0.7.1.** dating 5 Dec.), docker can be deployed on various Linux operating systems including Ubuntu / Debian and CentOS / RHEL.

Installation Instructions for Ubuntu

sudo aptitude update

```
sudo aptitude -y upgrade
```

Make sure aufs support is available:

```
sudo aptitude install linux-image-extra-`uname -r`
```

Add docker repository key to apt-key for package verification:

```
sudo sh -c "wget -qO- https://get.docker.io/gpg | apt-key add -"
```

Add the docker repository to aptitude sources:

```
sudo sh -c "echo deb http://get.docker.io/ubuntu docker main\
> /etc/apt/sources.list.d/docker.list"
```

Update the repository with the new addition:

```
sudo aptitude    update
```

Finally, download and install docker:

```
sudo aptitude install lxc-docker
```

Ubuntu's default firewall (**UFW**: Uncomplicated Firewall) denies all forwarding traffic by default, which is needed by docker.

Enable forwarding with UFW:

Edit UFW configuration using the nano text editor.

```
sudo nano /etc/default/ufw
```

Scroll down and find the line beginning with **DEFAULT_FORWARD_POLICY**.

Replace:

```
DEFAULT_FORWARD_POLICY="DROP"
```

With:

```
DEFAULT_FORWARD_POLICY="ACCEPT"
```

Press **CTRL+X** and approve with **Y** to save and close.

Finally, reload the UFW:

```
sudo ufw reload
```

Basic Docker Commands

Running the docker daemon and CLI Usage

Upon installation, the docker daemon should be running in the background, ready to accept commands sent by the docker CLI. For certain situations where it might be necessary to manually run docker, use the following:

Running the docker daemon:

```
sudo docker -d &
```

docker CLI Usage:

```
sudo docker [option] [command] [arguments]
```

Note:docker needs sudo privileges in order to work.

Docker Commands

Here is a summary of currently available (version **0.7.1**) docker commands:

```
attach: Attach to a running container
build:  Build a container from a Dockerfile
commit: Create a new image from a container's changes
cp:     Copy files/folders from the containers filesystem to the host path
diff:       Inspect changes on a container's filesystem
events: Get real time events from the server
export: Stream the contents of a container as a tar archive
history:    Show the history of an image
images: List images
import: Create a new filesystem image from the contents of a tarball
info:   Display system-wide information
insert: Insert a file in an image
inspect:    Return low-level information on a container
kill:       Kill a running container
load:   Load an image from a tar archive
login:  Register or Login to the docker registry server
logs:   Fetch the logs of a container
port:   Lookup the public-facing port which is NAT-ed to PRIVATE_PORT
ps:     List containers
pull:       Pull an image or a repository from the docker registry server
push:   Push an image or a repository to the docker registry server
restart:    Restart a running container
rm:     Remove one or more containers
rmi:        Remove one or more images
run:        Run a command in a new container
save:   Save an image to a tar archive
search: Search for an image in the docker index
start:  Start a stopped container
stop:   Stop a running container
tag:        Tag an image into a repository
top:        Lookup the running processes of a container
version:    Show the docker version information
```

Let"s Begin!

Building a Docker Container With Nginx Installed

After having installed docker and having quickly gone over its commands, we are ready to start with the

actual work to create our docker container running Nginx.

Note: Although after following this section we will have a running docker container with Nginx installed, it is definitely not the recommended method due to its complexity. However, it is here to offer you a chance to learn how to work with a live container and get familiarized with the commands we will need to define later to automate the process. To create a docker image with Nginx installed in a much better way, see the next section: **Creating a Dockerfile to Automatically Build Nginx Image**.

Creating a Base Docker Container From Ubuntu

Using docker's RUN command, we will begin with creating a new container based on the Ubuntu image. We are going to attach a terminal to it using the "-tâ€• flag.

```
sudo docker run -i -t -p 80:80 ubuntu /bin/bash
```

Note: After executing this command, docker might need to *pull* the Ubuntu image before creating a new container for you.

Remember: You will be attached to the container you create. In order to detach yourself and go back to your main terminal access point, run the escape sequence: CTRL+P followed by CTRL+Q. Being attached to a docker container is like being connected from inside another.

To attach yourself back to this container:

1. List all running containers using **sudo docker ps**
2. Find its ID
3. Use **sudo docker attach [id]** to attach back to its terminal

Important: Please do not forget that since we are in a container, all the following commands will be executed there, without affecting the host.

Preparing the Base Container for Nginx Installation

In order to install Nginx and the tools we are going to need for the process, the relevant application repository must be available for downloads.

Let's append Ubuntu's *universe* to the default list of the base image.

```
echo "deb http://archive.ubuntu.com/ubuntu/ raring main universe" >>
/etc/apt/sources.list
```

Update the list with the newly added source.

```
apt-get update
```

Before we proceed to install Nginx, there are some tools we should have installed such as nano - just in case.

```
apt-get install -y nano \
```

```
                wget \
                dialog \
                net-tools
```

Installing Nginx

Thanks to having it available in the repository, we can simply use apt-get to download and install nginx.

```
apt-get install -y nginx
```

Configuring Nginx

Using the text editor nano, which we have installed in the previous step, let's create a sample Nginx configuration to proxy connections to application servers.

```
# Delete the default configuration
rm -v /etc/nginx/nginx.conf
# Create a blank one using nano text editor
nano /etc/nginx/nginx.conf
```

First, on top of the file, a line must be added to **not** to have Nginx spawn its processes and then quit.

The reason we cannot allow this to happen is because docker depends on a single process to run (which can even be a process manager nonetheless) and when that process stops (i.e. quitting after spawning workers), the container stops.

Start with the following as the first line of the`nginx.conf`:

```
daemon off;
```

We will use a simple sample configuration to have Nginx run as a reverse proxy. Copy-and-paste the following after the`daemon off;`instruction.

```
worker_processes 1;
events { worker_connections 1024; }
http {
    sendfile on;
    gzip              on;
    gzip_http_version 1.0;
    gzip_proxied      any;
    gzip_min_length   500;
    gzip_disable      "MSIE [1-6]\.";
    gzip_types        text/plain text/xml text/css
                      text/comma-separated-values
                      text/javascript
                      application/x-javascript
                      application/atom+xml;
    # List of application servers
    upstream app_servers {
```

```
        server 127.0.0.1:8080;
    }
    # Configuration for the server
    server {
        # Running port
        listen 80;
        # Proxying the connections connections
        location / {
            proxy_pass         http://app_servers;
            proxy_redirect     off;
            proxy_set_header   Host $host;
            proxy_set_header   X-Real-IP $remote_addr;
            proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header   X-Forwarded-Host $server_name;
        }
    }
}
```

Save and exit pressing CTRL+X and confirming with Y.

To run Nginx, you can execute the following:

```
service nginx start
```

And that's it! We now have Nginx running in a docker container, accessible from the outside world on port **80** as we set using the `-p 80:80` flag.

Remember: This Nginx file, albeit configured correctly, will not do anything since there are currently no application servers running on the server. Instead of this one, you can copy and use another example which simply works as a forward proxy for testing HTTP headers until you have your application server(s) installed and working.

Creating the Dockerfile to Automatically Build the Image

As we have mentioned in the previous step, it is certainly not the recommended way to create containers this way for scalable production. The right way to do can be considered as using Dockerfiles to automate the build process in a structured way.

After having gone through the necessary commands for downloading and installing Nginx inside a container, we can use the same knowledge to compose a Dockerfile that docker can use to build an image, which then can be used to run Nginx instances easily.

Before we start working on the Dockerfile, let's quickly go over the basics.

Dockerfile Basics

Dockerfiles are scripts containing ***commands*** declared successively which are to be executed in that order by docker to automatically create a new docker image. They help greatly with deployments.

These files always begin with defining an base image using the `FROM`command. From there on, the*build process* starts and each following action taken forms the final image which will be committed on the host.

Usage:

```
# Build an image using the Dockerfile at current location
# Tag the final image with [name] (e.g. *nginx*)
# Example: sudo docker build -t [name] .
sudo docker build -t nginx_img .
```

Dockerfile Commands Overview

- ADD: Copy a file from the host into the container
- CMD: Set default commands to be executed, or passed to the ENTRYPOINT
- ENTRYPOINT: Set the default entrypoint application inside the container
- ENV: Set environment variable (e.g. key = value)
- EXPOSE: Expose a port to outside
- FROM: Set the base image to use
- MAINTAINER: Set the author / owner data of the Dockerfile
- RUN: Run a command and commit the ending result (container) image
- USER: Set the user to run the containers from the image
- VOLUME: Mount a directory from the host to the container
- WORKDIR: Set the directory for the directives of **CMD** to be executed

Creating the Dockerfile

To create a Dockerfile at the current location using the nano text editor, execute the following command:

```
sudo nano Dockerfile
```

Note: Append all the following lines one after the other to form the Dockerfile to be saved and used for building.

Defining the Fundamentals

Let's begin our Dockerfile by defining the basics (fundamentals) such as the`FROM`image (i.e. Ubuntu) and the `MAINTAINER`.

```
############################################################
# Dockerfile to build Nginx Installed Containers
# Based on Ubuntu
############################################################
# Set the base image to Ubuntu
FROM ubuntu
# File Author / Maintainer
MAINTAINER Maintaner Name
```

Installation Instructions for Nginx

Following our steps from the previous section, let's form the block to have Nginx installed.

```
# Install Nginx
# Add application repository URL to the default sources
RUN echo "deb http://archive.ubuntu.com/ubuntu/ raring main universe" >>
/etc/apt/sources.list
# Update the repository
RUN apt-get update
# Install necessary tools
RUN apt-get install -y nano wget dialog net-tools
# Download and Install Nginx
RUN apt-get install -y nginx
```

Bootstrapping

After adding the instructions for installing Nginx, let's finish off with configuring Nginx and getting Dockerfile to replace the default configuration file with one we provide during build.

```
# Remove the default Nginx configuration file
RUN rm -v /etc/nginx/nginx.conf
# Copy a configuration file from the current directory
ADD nginx.config /etc/nginx/
# Append "daemon off;" to the beginning of the configuration
RUN echo "daemon off;" >> /etc/nginx/nginx.conf
# Expose ports
EXPOSE 80
# Set the default command to execute
# when creating a new container
CMD service nginx start
```

Final Dockerfile

In the end, this is what the Dockerfile should look like:

```
################################################################
# Dockerfile to build Nginx Installed Containers
# Based on Ubuntu
################################################################
# Set the base image to Ubuntu
FROM ubuntu
# File Author / Maintainer
MAINTAINER Maintaner Name
# Install Nginx
# Add application repository URL to the default sources
RUN echo "deb http://archive.ubuntu.com/ubuntu/ raring main universe" >>
/etc/apt/sources.list
```

```
# Update the repository
RUN apt-get update
# Install necessary tools
RUN apt-get install -y nano wget dialog net-tools
# Download and Install Nginx
RUN apt-get install -y nginx
# Remove the default Nginx configuration file
RUN rm -v /etc/nginx/nginx.conf
# Copy a configuration file from the current directory
ADD nginx.config /etc/nginx/
# Append "daemon off;" to the beginning of the configuration
RUN echo "daemon off;" >> /etc/nginx/nginx.conf
# Expose ports
EXPOSE 80
# Set the default command to execute
# when creating a new container
CMD service nginx start
```

Again save and exit the file by pressing CTRL+X and confirming with Y.

Using the Dockerfile to Automatically Build Nginx Containers

As we first went over in the "basics" section, Dockerfiles' usage consists of calling them with "docker buildâ€• command.

Since we are instructing docker to copy a configuration (i.e. nginx.conf) from the current directory to replace the default one, we need to make sure to have it alongside this Dockerfile before starting the build process.

Note: The above explained procedure of copying *in* an Nginx configuration allows you great flexibility and saves a lot of time by not dealing with attaching and detaching yourself from containers to create configuration files. Now you can simply use one to directly build and run an image.

Create a sample`nginx.conf`using the text editor nano:

```
sudo nano nginx.conf
```

And replace its contents to use it as a forward proxy for testing:

```
worker_processes 1;
events { worker_connections 1024; }
http {
    sendfile on;
    server {
        listen 80;
        location / {
            proxy_pass http://httpstat.us/;
            proxy_set_header  X-Real-IP  $remote_addr;
        }
    }
}
```

Let's save and exit the nginx.conf the same way by pressing CTRL+X and confirming with Y.

This docker image will allow us to port all our progress and quickly create containers running Nginx with a single command.

To start using it, build a new container image with the following:

```
sudo docker build -t nginx_img_1 .
```

And using that image - which we tagged as nginx_img_1 - we can run a new container:

```
sudo docker run -name nginx_cont_1 -p 80:80 -i -t nginx_img_1
```

Now you can visit the IP address and your Nginx running docker container shall do its job, forwarding you to the HTTP status testing page.

Example:

```
# Usage: Visit http://[my ip]
http://95.85.10.236/200
```

Sample Response:

```
200 OK
```