How to create Docker Images with a Dockerfile

Docker is operating-system-level virtualization mainly intended for developers and sysadmins. Docker makes it easier to create and deploy applications in an isolated environment. A **Dockerfile** is a script that contains collections of commands and instructions that will be automatically executed in sequence in the docker environment for building a new docker image.

In this tutorial, I will show you how to create your own docker image with a dockerfile. I will explain the dockerfile script in detail to enable you to build your own dockerfile scripts.

Prerequisite

- A Linux Server - I will use Ubuntu 16.04 as the host machine, and Ubuntu 16.04 as the docker base image.
- Root Privileges.
- Understanding Docker command

Introduction to the Dockerfile Command

A dockerfile is a script which contains a collection of dockerfile commands and operating system commands (ex: Linux commands). Before we create our first dockerfile, you should become familiar with the dockerfile command.

Below are some dockerfile commands you must know:

FROM

The base image for building a new image. This command must be on top of the dockerfile.

MAINTAINER

Optional, it contains the name of the maintainer of the image.

RUN

Used to execute a command during the build process of the docker image.

ADD

Copy a file from the host machine to the new docker image. There is an option to use a URL for the file, docker will then download that file to the destination directory.

ENV

Define an environment variable.

CMD

Used for executing commands when we build a new container from the docker image.

ENTRYPOINT

Define the default command that will be executed when the container is running.

WORKDIR

This is directive for CMD command to be executed.

USER

Set the user or UID for the container created with the image.

VOLUME

Enable access/linked directory between the container and the host machine.

Now let's start to create our first dockerfile.

Step 1 - Installing Docker

Login to your server and update the software repository.

```
ssh root@192.168.1.248
apt-get update
```

Install docker.io with this apt command:

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

When the installation is finished, start the docker service and enable it to start at boot time:

```
systemctl start docker
systemctl enable docker
```

Docker has been installed and is running on the system.

Step 2 - Create Dockerfile

In this step, we will create a new directory for the dockerfile and define what we want to do with that dockerfile.

Create a new directory and a new and empty dockerfile inside that directory.

```
mkdir ~/myimages
cd myimages/
touch Dockerfile
```

Next, define what we want to do with our new custom image. In this tutorial, I will install Nginx and PHP-FPM 7 using an Ubuntu 16.04 docker image. Additionally, we need Supervisord, so we can start Nginx and PHP-FPM 7 both in one command.

Edit the 'Dockerfile' with vim:

```
nano Dockerfile
```

On the top of the file, add a line with the base image (Ubuntu 16.04) that we want to use.

```
#Download base image ubuntu 16.04
FROM ubuntu:16.04
```

Update the Ubuntu software repository inside the dockerfile with the 'RUN' command.

```
# Update Ubuntu Software repository
RUN apt-get update
```

Then install the applications that we need for the custom image. Install Nginx, PHP-FPM and Supervisord from the Ubuntu repository with apt. Add the RUN commands for Nginx and PHP-FPM installation.

```
# Install nginx, php-fpm and supervisord from ubuntu repository
RUN apt-get install -y nginx php7.0-fpm supervisor && \
    rm -rf /var/lib/apt/lists/*
```

At this stage, all applications are installed and we need to configure them. We will configure Nginx for handling PHP applications by editing the default virtual host configuration. We can replace it our new configuration file, or we can edit the existing configuration file with the 'sed' command.

In this tutorial, we will replace the default virtual host configuration with a new configuration by using the 'COPY' dockerfile command.

```
#Define the ENV variable
ENV nginx_vhost /etc/nginx/sites-available/default
ENV php_conf /etc/php/7.0/fpm/php.ini
```

```
ENV nginx_conf /etc/nginx/nginx.conf
ENV supervisor_conf /etc/supervisor/supervisord.conf

 # Enable php-fpm on nginx virtualhost configuration
COPY default ${nginx_vhost}
RUN sed -i -e 's/;cgi.fix_pathinfo=1/cgi.fix_pathinfo=0/g' ${php_conf} && \
    echo "\ndaemon off;" >> ${nginx_conf}
```

Next, configure Supervisord for Nginx and PHP-FPM. We will replace the default Supervisord configuration with a new configuration by using the 'COPY' command.

```
#Copy supervisor configuration
COPY supervisord.conf ${supervisor_conf}
```

Now create a new directory for the php-fpm sock file and change the owner of the /var/www/html directory and PHP directory to www-data.

```
RUN mkdir -p /run/php && \
    chown -R www-data:www-data /var/www/html && \
    chown -R www-data:www-data /run/php
```

Next, define the volume so we can mount the directories listed below to the host machine.

```
# Volume configuration
VOLUME ["/etc/nginx/sites-enabled", "/etc/nginx/certs", "/etc/nginx/conf.d", "/var/lo
```

Finally, setup the default container command 'CMD' and open the port for HTTP and HTTPS. We will create a new start.sh file for default 'CMD' command when container is starting. The file contains the 'supervisord' command, and we will copy the file to the new image with the 'COPY' dockerfile command.

```
# Configure Services and Port
COPY start.sh /start.sh
CMD ["./start.sh"]

 EXPOSE 80 443
```

Save the file and exit.

Here is the complete Dockerfile in one piece:

```
#Download base image ubuntu 16.04
FROM ubuntu:16.04

 # Update Software repository
RUN apt-get update

 # Install nginx, php-fpm and supervisord from ubuntu repository
RUN apt-get install -y nginx php7.0-fpm supervisor && \
    rm -rf /var/lib/apt/lists/*

 #Define the ENV variable
ENV nginx_vhost /etc/nginx/sites-available/default
```

```
ENV php_conf /etc/php/7.0/fpm/php.ini
ENV nginx_conf /etc/nginx/nginx.conf
ENV supervisor_conf /etc/supervisor/supervisord.conf

 # Enable php-fpm on nginx virtualhost configuration
COPY default ${nginx_vhost}
RUN sed -i -e 's/;cgi.fix_pathinfo=1/cgi.fix_pathinfo=0/g' ${php_conf} && \
    echo "\ndaemon off;" >> ${nginx_conf}

 #Copy supervisor configuration
COPY supervisord.conf ${supervisor_conf}

 RUN mkdir -p /run/php && \
    chown -R www-data:www-data /var/www/html && \
    chown -R www-data:www-data /run/php

 # Volume configuration
VOLUME ["/etc/nginx/sites-enabled", "/etc/nginx/certs", "/etc/nginx/conf.d", "/var/lo

 # Configure Services and Port
COPY start.sh /start.sh
CMD ["./start.sh"]

 EXPOSE 80 443
```

Now inside our 'Dockerfile' directory, create a new configuration file for the virtual host named 'default', a supervisord configuration file 'supervisord.conf' and a service configuration script 'start.sh'.

```
vim default
```

Paste default virtual host configuration below:

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

     root /var/www/html;
    index index.html index.htm index.nginx-debian.html;

     server_name _;

     location / {
        try_files $uri $uri/ =404;
    }

     location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php7.0-fpm.sock;
    }

     # deny access to .htaccess files, if Apache's document root
    # concurs with nginx's one
    #
    #location ~ /\.ht {
    #    deny all;
    #}
```

```
}
```

Supervisord configuration file:

```
vim supervisord.conf
```

Paste configuration below:

```
[unix_http_server]
file=/dev/shm/supervisor.sock   ; (the path to the socket file)

 [supervisord]
logfile=/var/log/supervisord.log ; (main log file;default $CWD/supervisord.log)
logfile_maxbytes=50MB        ; (max main logfile bytes b4 rotation;default 50MB)
logfile_backups=10           ; (num of main logfile rotation backups;default 10)
loglevel=info                ; (log level;default info; others: debug,warn,trace)
pidfile=/tmp/supervisord.pid ; (supervisord pidfile;default supervisord.pid)
nodaemon=false               ; (start in foreground if true;default false)
minfds=1024                  ; (min. avail startup file descriptors;default 1024)
minprocs=200                 ; (min. avail process descriptors;default 200)
user=root             ;

 ; the below section must remain in the config file for RPC
; (supervisorctl/web interface) to work, additional interfaces may be
; added by defining them in separate rpcinterface: sections
[rpcinterface:supervisor]
supervisor.rpcinterface_factory = supervisor.rpcinterface:make_main_rpcinterface

 [supervisorctl]
serverurl=unix:///dev/shm/supervisor.sock ; use a unix:// URL  for a unix socket

 ; The [include] section can just contain the "files" setting.  This
; setting can list multiple files (separated by whitespace or
; newlines).  It can also contain wildcards.  The filenames are
; interpreted as relative to this file.  Included files *cannot*
; include files themselves.

 [include]
files = /etc/supervisor/conf.d/*.conf

  [program:php-fpm7.0]
command=/usr/sbin/php-fpm7.0 -F
numprocs=1
autostart=true
autorestart=true

 [program:nginx]
command=/usr/sbin/nginx
numprocs=1
autostart=true
autorestart=true
```

Start.sh file.

```
vim start.sh
```

Paste configuration below:

```
#!/bin/sh

 /usr/bin/supervisord -n -c /etc/supervisor/supervisord.conf
```

Save and exit

Make start.sh executable with chmod command:

```
chmod +x start.sh
```

Save the file and exit.

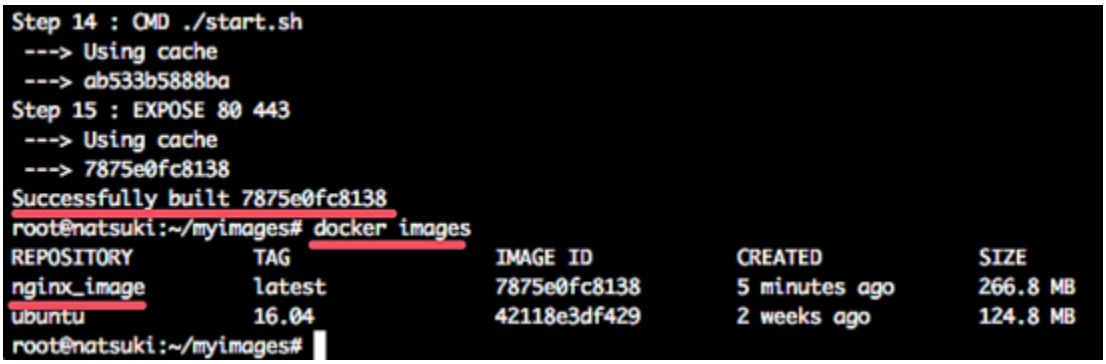
Step 3 - Build New Docker Image and Create New Container Based on it

The Dockerfile and all required config files have been created, now we can build a new docker image based on Ubuntu 16.04 and our dockerfile with the docker command below:

```
docker build -t nginx_image .
```

When the command completed successfully, we can check the new image 'nginx_image' with the docker command below:

```
docker images
```



Then we can try to create a new container based on nginx_images. And before create new container, we can create new directory on the host machine for the webroot data.

```
mkdir -p /webroot
```

Now run the new container with the command below:

```
docker run -d -v /webroot:/var/www/html -p 80:80 --name hakase nginx_image
```

Then we can check that the new container with name hakase based on 'nginx_image' is running:

```
docker ps
```

```
root@natsuki:~/myimages# mkdir -p /webroot
root@natsuki:~/myimages# docker run -d -v /webroot:/var/www/html -p 80:80 --name hakase nginx_image
7ddd15694ab6e6620c173f037d367039a6b1513c98511e0ea5e758035313be6a
root@natsuki:~/myimages# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                          NAMES
7ddd15694ab6        nginx_image         "./start.sh"        3 seconds ago       Up 2 seconds        0.0.0.0:80->80/tcp, 443/tcp    hakase
root@natsuki:~/myimages#
```

Note:

- *--name hakase nginx_image* = We create a new container with the name 'hakase', based on docker image 'nginx_images'.
- *-p 80:80* = hakase container running on port 80 on the host machine.
- *-v /webroot:/var/www/html* = /webroot directory on the host machine rewrite the /var/www/html directory on the container.

The new container based on the nginx_image is running without error.

Step 4 - Testing Nginx and PHP-FPM in the Container

Try to create a new index.html file in the /webroot directory with echo:

```
echo '<h1>Nginx and PHP-FPM 7 inside Docker Container</h1>' > /webroot/index.html
```

Testing with curl command by accessing the host machine ip address.

```
curl 192.168.1.250
curl -I 192.168.1.250
```

We will see the results below.

```
ubuntu@hiroyuki:~$ curl 192.168.1.248
<h1>Nginx and PHP-FPM 7 inside Docker Container</h1>
ubuntu@hiroyuki:~$ curl -I 192.168.1.248
HTTP/1.1 200 OK
Server: nginx/1.10.0 (Ubuntu)
Date: Thu, 11 Aug 2016 16:26:05 GMT
Content-Type: text/html
Content-Length: 53
Last-Modified: Thu, 11 Aug 2016 16:25:37 GMT
Connection: keep-alive
ETag: "57aca701-35"
Accept-Ranges: bytes

ubuntu@hiroyuki:~$
```

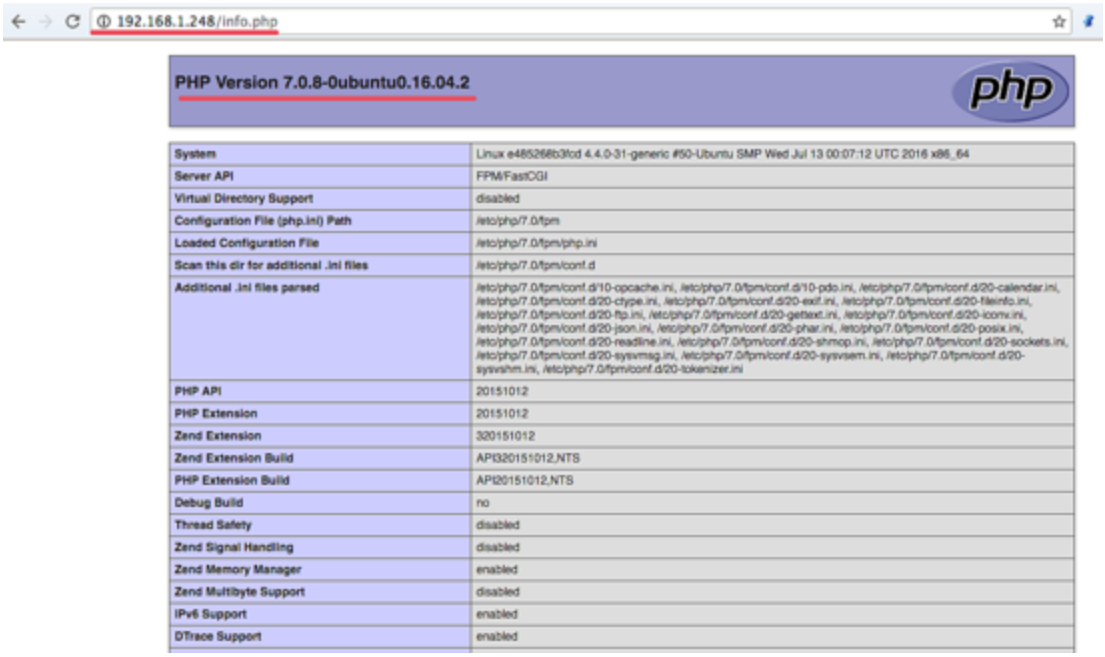
Next, test that PHP-FPM 7.0 is running by creating a new phpinfo file in the /webroot directory on the host machine.

```
echo '<?php phpinfo(); ?>' > /webroot/info.php
```

Open the web browser and type the host machine IP address:

http://192.168.1.248/info.php

Now you can see the output of the phpinfo file.



the new docker image 'nginx_image' has been successfully created, now we can create more containers based on that image.

Reference

- https://docs.docker.com/engine/reference/builder/
- https://github.com/ngineered/nginx-php-fpm
- https://github.com/docker-library/wordpress/

×

This feature is only available to subscribers. Get your subscription here.