How to Deploy a Django Application with Docker

In this tutorial, we'll apply Docker to website development. We'll learn how to deploy a Django application with Docker on an Alibaba Cloud ECS instance.

by Esther Vaati MVB · May. 17, 18 · Web Dev Zone · Tutorial

Written by Esther Vaati, Alibaba Cloud Tech Share author. Tech Share is Alibaba Cloud's incentive program to encourage the sharing of technical knowledge and best practices within the cloud community.

In this tutorial, we are going to learn about Docker and how to apply it to website development. We will be deploying a Django application with Docker on an Alibaba Cloud ECS instance.

What is Docker?

Docker is a technology that makes it easier to create, deploy, and run applications by using containers. Containers allow developers to package applications with all the components required by the applications and later ship them out as packages. It also makes it possible to get more apps running on the same server.

With Docker, you can be assured of a higher level of security since applications that are running on containers are isolated from each other. In addition, Docker ensures that each container has its own resources and therefore an application will only use the resources that are assigned to it.

Prerequisites

Before you begin this guide you'll need the following:

- An Alibaba Cloud ECS Linux instance. If you haven't yet set up your Linux instance, this article shows you various ways to set it up.
- Docker
- Python 2.7

Install Docker

Login to your server using the ssh command.

`$ ssh root@47.88.220.88`

Update Ubuntu packages.

`$ sudo apt-get update`

Install the latest version of Docker with the following command.

`$ sudo apt-get install docker`

To verify that Docker has installed correctly run the following command.

`$ sudo docker run hello-world`

If performed correctly, the above commands should let your instance download a test image and run it in a container.

Containers and Images in Docker

On an Alibaba Cloud ECS instance, you can use images to create ECS clusters with identical configurations. Similarly, Docker containers have images. Conceptually, they are very similar. Based on the official Docker documentation:

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings.

You can view running containers by running `$ sudo docker ps`.

An image, on the other hand, is an inert, immutable, file that's essentially a snapshot of a container. Images are created with the build command, and they'll produce a container when started with the run command.

You can view images by running `$ sudo docker images`.

Build a Django Application

First, let's install Django and Create a Django application.

```
$ sudo pip install django==1.9
$ django-admin startproject djangoapp
```

Requirements File

Create a requirements file inside the djangoapp directory and define the dependencies required by the application.

```
$ cd djangoapp
$ nano requirements.txt
```

Add the following dependencies.

```
#requirements.txt

Django==1.9
gunicorn==19.6.0
```

Create Docker file

Docker has the ability to build images automatically by reading instructions from a Dockerfile. A docker file contains all the commands and instructions that Docker uses to build images.

Let's define some of the basic commands used in a Dockerfile.

- **FROM** - initializes a new build stage and sets the Base Image for subsequent instructions. As such, a valid Dockerfile must start with a FROM instruction.
- **RUN** - runs the command specified.
- **ADD** - Copy a file(s) into the container.
- **EXPOSE** - informs Docker that the container listens on the specified network ports at runtime.
- **CMD** - provide defaults for an executing container.

Now let's create a file named Dockerfile.

`$ nano Dockerfile`

Let's begin by defining all the properties required in a Dockerfile. Define the base image and maintainer name.

```
# base image
FROM python:2.7

# File Author / Maintainer
MAINTAINER Esther
```

Next, copy the application folder inside the container and define the directory where CMD will execute.

```
# Copy the application folder inside the container
ADD . /usr/src/app

# set the default directory where CMD will execute
WORKDIR /usr/src/app
```

Finally, set the default command to execute.

`CMD exec gunicorn djangoapp.wsgi:application --bind 0.0.0.0:8000 --workers 3`

Your final Dockerfile should now look like this.

```
# set the base image
FROM python:2.7

# File Author / Maintainer
MAINTAINER Esther

#add project files to the usr/src/app folder
ADD . /usr/src/app

#set directoty where CMD will execute
WORKDIR /usr/src/app

COPY requirements.txt ./

# Get pip to download and install requirements:
RUN pip install --no-cache-dir -r requirements.txt

# Expose ports
EXPOSE 8000

# default command to execute
CMD exec gunicorn djangoapp.wsgi:application --bind 0.0.0.0:8000 --workers 3
```

Build the Docker Image

Run the following command to build the docker image.

```
$ sudo docker build -t django_application_image .

Sending build context to Docker daemon   12.8kB
Step 1/7 : FROM python:2.7
 ---&gt; 2863c80c418c
Step 2/7 : ADD . /usr/src/app
 ---&gt; 09b03ff8466e
Step 3/7 : WORKDIR /usr/src/app
Removing intermediate container a71a3bf6af90
 ---&gt; 3186c92adc85
Step 4/7 : COPY requirements.txt ./
 ---&gt; 701c0be5e039
Step 5/7 : RUN pip install --no-cache-dir -r requirements.txt
 ---&gt; Running in ed034f98db74
Collecting Django==1.9 (from -r requirements.txt (line 1))
  Downloading Django-1.9-py2.py3-none-any.whl (6.6MB)
Collecting gunicorn==19.6.0 (from -r requirements.txt (line 2))
  Downloading gunicorn-19.6.0-py2.py3-none-any.whl (114kB)
Installing collected packages: Django, gunicorn
Successfully installed Django-1.9 gunicorn-19.6.0
Removing intermediate container ed034f98db74
 ---&gt; 1ffd08204a07
Step 6/7 : EXPOSE 8000
 ---&gt; Running in 987b48e1a4ef
Removing intermediate container 987b48e1a4ef
 ---&gt; ef889d6e8fcb
Step 7/7 : CMD exec gunicorn djangoapp.wsgi:application --bind 0.0.0.0:8000 --workers 3
 ---&gt; Running in 4d929e361d0f
Removing intermediate container 4d929e361d0f
 ---&gt; c6baca437c64
Successfully built c6baca437c64
Successfully tagged django_application_image:latest
```

Your built image is now in your machine's local Docker image registry. You can check your image by running `$ sudo docker images`.

```
REPOSITORY                 TAG        IMAGE ID        CREATED          SIZE
django_application_image   latest     c6baca437c64    34 minutes ago   702MB
```

Run the App

```
$ sudo docker run -p 8000:8000 -i -t django_application_image

[2018-03-25 12:29:08 +0000] [1] [INFO] Starting gunicorn 19.6.0
[2018-03-25 12:29:08 +0000] [1] [INFO] Listening at: http://0.0.0.0:8000
[2018-03-25 12:29:08 +0000] [1] [INFO] Using worker: sync
[2018-03-25 12:29:08 +0000] [8] [INFO] Booting worker with pid: 8
[2018-03-25 12:29:08 +0000] [9] [INFO] Booting worker with pid: 9
[2018-03-25 12:29:08 +0000] [10] [INFO] Booting worker with pid: 10
```

You should see a message that **gunicorn** is serving your app at **http://0.0.0.0:8000**. Navigate to your servers IP (ip_address:8000) and you should see the Django welcome page.

To see running containers:

```
$ sudo docker ps -a
CONTAINER ID     IMAGE                      COMMAND                  CREATED          STATUS
100695b41a0a     django_application_image   "/bin/sh -c 'exec gu…"   13 seconds ago   Exited (0) 4 seconds a
```

Conclusion

Occasionally you might face some challenges when using Docker. The first thing to do when you experience an error is to check Docker logs files as they provide some information on what might have gone wrong.

Docker and other containers are a powerful alternative to traditional virtual machines for application development. To learn more about running containers on Alibaba Cloud, visit the Container Service page.

Topics: WEB DEV, DJANGO, DOCKER, ALIBABA CLOUD