How to Create Django Docker Images

section.io/engineering-education/django-docker

Docker is a containerization platform that makes it possible to build, ship, and run distributed applications in controlled environments with defined rules.

Prerequisites

1. Have Python 3.6 or newer installed on your computer.
2. Get Python package manager, pip installed on your computer.
3. You will need Docker installed on your computer.

For Docker installation, setup, and a quick start, visit Getting started with Docker.

Project setup

We will be creating a Docker image for the Todo application that we created in this tutorial. The `Todo` application that allows us to `create`, `read`, `update`, and `delete` Todo items via a REST API.

Ensure `virtualenv` is installed on your computer by executing the command below.

```
$ virtualenv --version
virtualenv 20.2.2 from /home/username/.local/lib/python3.8/site-
packages/virtualenv/__init__.py
```

If you get an error executing the command above, run the command below to install `virtualenv` on your computer.

```
$ pip install virtualenv
```

Create a working directory for the project by executing the command below.

```
$ mkdir todo
$ cd todo
```

Create a virtual environment for our project using the `virtualenv` module that we just installed. Execute the command below to create and activate the virtual environment.

```
$ virtualenv venv
$ source venv/bin/activate
```

In the working directory, clone the project from Github to your computer using the command below.

```
$ git clone https://github.com/paulodhiambo/django_todo.git
```

Move into the project directory by executing the command below.

```
$ cd django_todo
```

Execute the command below to install all the required dependencies for the project to run.

```
$ pip install -r requirements.txt
```

Run the application to verify that nothing is broken and the application runs without errors.

```
$ ./manage.py runserver
```

Below is the project structure.

```
└── django_todo         # < project root package
    ├── todo             # < todo app
    │   ├── admin.py
    │   ├── apps.py
    │   ├── migrations
    │   ├── models.py
    │   ├── serializers.py
    │   ├── urls.py
    │   └── views.py
    ├── manage.py
    ├── requirements.txt    # < Django dependencies list
    └── django_todo
        ├── settings.py # Django settings file
        ├── urls.py
        └── wsgi.py
```

Creating a Dockerfile

A Dockerfile is a text file that contains instructions on how the Docker image will be built. A Dockerfile contains the directives below.

- **FROM**: directive sets the base image from which the Docker container will be built.
- **WORKDIR**: directive sets the working directory in the image created.
- **RUN**: directive executes commands in the container.
- **COPY**: directive copies files from the file system into the container.
- **CMD**: directive sets the executable commands within the container.

In the root project directory, create a file with the name `Dockerfile` with no file extension. In the `Dockerfile` created above, add the code below.

```
# pull the official base image
FROM python:3.8.3-alpine

# set work directory
WORKDIR /usr/src/app

# set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# install dependencies
RUN pip install --upgrade pip
COPY ./requirements.txt /usr/src/app
RUN pip install -r requirements.txt

# copy project
COPY . /usr/src/app

EXPOSE 8000

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

- `FROM python:3.8.3-alpine` sets the base image from which the Docker container will be created.
- `WORKDIR /usr/src/app` sets the working directory inside the container to `/usr/src/app`.
- `ENV PYTHONDONTWRITEBYTECODE 1` prevents Python from copying pyc files to the container.
- `ENV PYTHONUNBUFFERED 1` ensures that Python output is logged to the terminal, making it possible to monitor Django logs in realtime.
- `RUN pip install --upgrade pip` installs and upgrades the pip version that is in the container.
- `COPY ./requirements.txt /usr/src/app` copies the `requirements.txt` file into the work directory in the container.
- `RUN pip install -r requirements.txt` installs all the required modules for the `django_todo` application to run in the container.
- `COPY . /usr/src/app` copies all the project source code to the working directory in the container.
- `EXPOSE 8000` exposes port 8000 for access from other applications.
- `CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]` sets the executable commands in the container.

Building the Docker image

To build the Docker image from the Dockerfile we created above, execute the command below.

```
$ docker build --tag django_todo:latest .
```

- `--tag` sets the tag for the image. For example, we are creating a Docker image from `python:3.8.3` that has the tag `alpine`. In our Docker image, `latest` is the tag set.

- The trailing `.` indicates that the `Dockerfile` is within the current working directory.

To list all the available images on your computer, execute the command below.

```
$ docker image ls
REPOSITORY    TAG            IMAGE ID       CREATED         SIZE
django_todo   latest         6e06c89267f1   3 hours ago     147MB
todo          latest         b10c177c6d58   4 hours ago     162MB
<none>        <none>         2f418d359923   4 hours ago     162MB
centos        latest         300e315adb2f   3 weeks ago     209MB
python        3.8.3-alpine   8ecf5a48c789   6 months ago    78.9MB
hello-world   latest         bf756fb1ae65   12 months ago   13.3kB
```

From the above list, we see the `django_todo` image that we have created.

Creating and running the Docker Container

To build and run a Docker container from the Docker image we created above, run the command below.

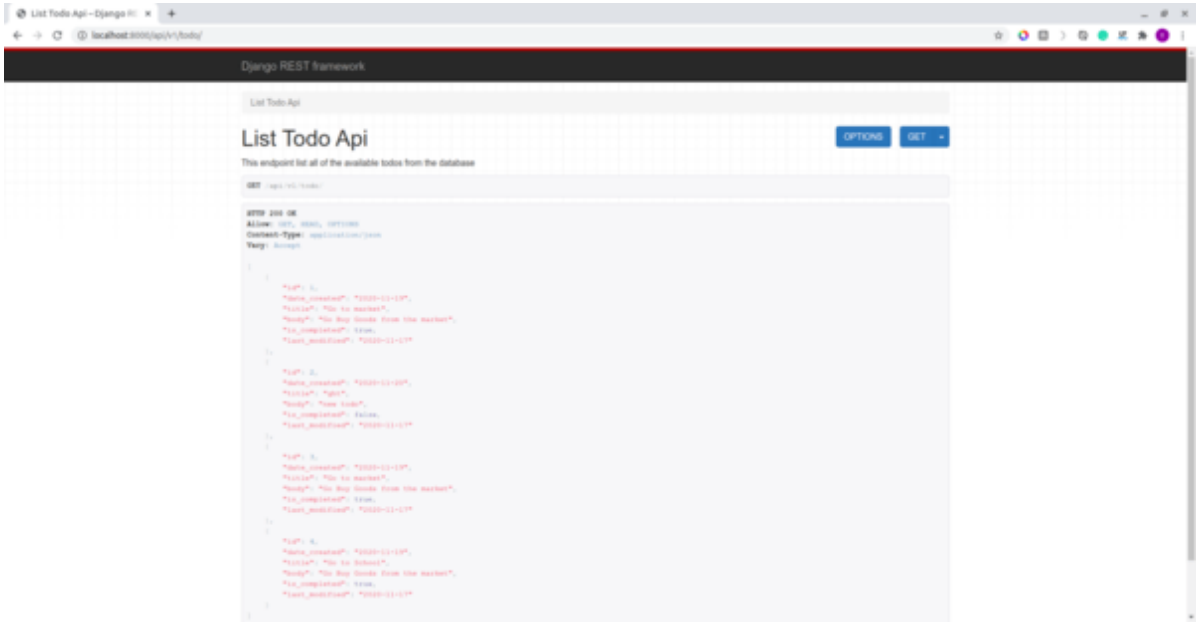
```
$ docker run --name django_todo -d -p 8000:8000 django_todo:latest
```

- `--name` sets the name of the Docker container.
- `-d` makes the image run in detached mode. The image is capable of running in the background.
- `-p 8000:8000` maps port 8000 in the Docker container to port 8000 in localhost.
- `django_todo: latest` specifies which image is used to build the Docker container.

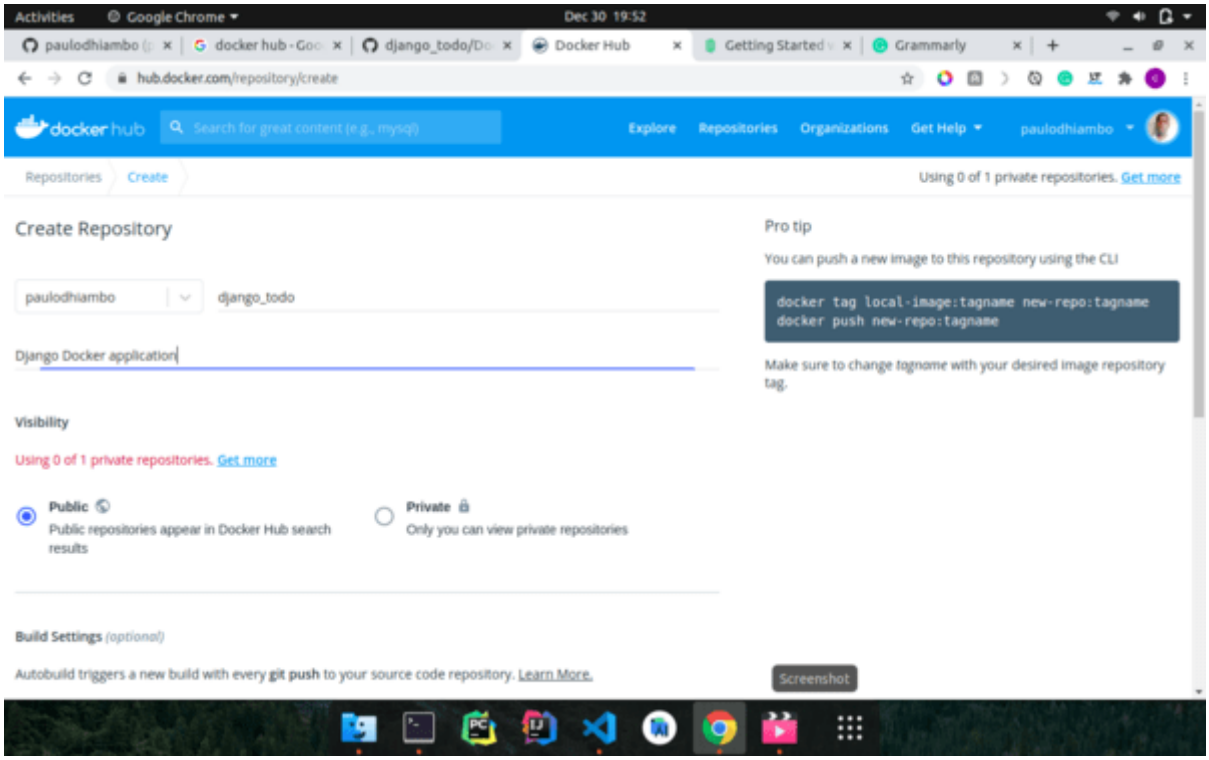
To list all the running Docker containers, execute the command below.

```
$ docker container ps
CONTAINER ID   IMAGE                COMMAND                  CREATED       STATUS
PORTS                    NAMES
d73306a9fb04   django_todo:latest   "python manage.py ru…"   3 hours ago   Up 3
hours   0.0.0.0:8000->8000/tcp   django_todo
```

1. On your browser visit `localhost` on port 8000 to confirm if the `django_todo` application is running in the container.



Publishing the Docker image to Docker Hub



Docker Hub is a repository of container images that can be used to create Docker containers.

Visit Docker Hub and create an account if you don't have one.

Once you have created an account and logged in to Docker Hub, create a repository with the name `django_todo` and the description `Django Docker Image`.

Now that we have created a repository in Docker Hub, to push the Docker image to the repository we have created we execute the command below.

```
$ docker login

$ docker tag django_todo:latest <Docker Hub username>/django_todo:latest

$ docker push <Docker Hub username>/django_todo:latest
```

- `docker login` command logs you into your Docker Hub account on the command line, making it possible to push Docker images to Docker Hub.
- `docker tag django_todo: latest <Docker Hub username>/django_todo: latest` command modifies the tag of our local Docker image to include the Docker Hub username.
- `docker push <Docker Hub username>/django_todo: latest` pushes the image to Docker Hub repository we created earlier.

Note: Replace `<Docker Hub username>` with your actual Docker Hub username.

The full source code for the application is in this GitHub repository.

Conclusion

Docker is a great tool for packaging applications with the required dependencies to run them both locally and in production. Docker playground is an online playground that you can use to test various Docker commands without installing Docker locally on your computer.

Happy Coding!

Peer Review Contributions by: Geoffrey Mungai