

Docker-ready Spirit/Django Application

Created by Mert Okumuş ([merto-dvp](#))

Project Information:

This application uses:

- **Django**
- **nitely/Spirit**, a Python based forum built using the Django framework. For further information: <https://spirit.readthedocs.io/en/latest/>
- **Docker**, a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. For further information: <https://docs.docker.com/>
- **NGINX**, a web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache. For further information: <https://docs.nginx.com/>
- **gunicorn**, a Python WSGI HTTP Server for UNIX. For further information: <https://docs.gunicorn.org/en/stable/>
- **Bash**, a Unix shell and command language. For further information: <https://devdocs.io/bash/>

Project Folder:

```
— spirit-docker-v2 (Main GitHub Repository / https://github.com/merto-dvp/spirit-docker-v2 )
  — README.md (Readme & Tutorial)
  — app
    | — config
    |   — nginx
    |   — Dockerfile (Docker config for nginx)
    |   — nginx.conf (nginx server config, Server Ports)
    | — src
    |   — static (Static files for Django/Spirit)
    |   — admin
    |   — spirit
    | — mysite (Default app folder for Spirit, You can change it)
    |   — settings
    |     — base.py
    |     — dev.py (Main settings for application, DATABASE, DEBUG, SECRET_KEY, ALLOWED_HOSTS)
    |     — prod.py
    |     — test.py
    |   — celery.py
    |   — urls.py
    |   — wsgi.py
    | — after_settings_script.sh ( Script for saving changes made in settings folder.)
    | — cleanup_script.sh      ( Script for cleaning folder, making it ready for new app)
    | — docker-compose.yml    ( Settings for Docker-compose & containers, DJANGO, NGINX,DATABASE(POSTGRE))
    | — Dockerfile            ( Settings for Docker)
    | — env.dev                ( File for easier changes on DEBUG, SECRET_KEY, ALLOWED_HOSTS (dev.py))
    | — install_script.sh     ( Start point /Main installation script, Installs required packages for Docker, Python etc)
    | — manual_install_script.sh (Script for manual installation of Spirit/ Read Manual Installation Guide for details)
    | — requirements.txt      (Contains package names for required resources)
    | — spirit_install_app_script.sh (Script for installing default (boilerplate) project from GitHub repo/
    |                               Asks for permission / https://github.com/merto-dvp/django-spirit-ready-app )
```

Installation & Run

Make sure internet connection is available.

Works on Ubuntu 20.04 LTS, (Tested on Microsoft Azure & Amazon EC2 virtual machines)

For Automatic Installation, you can run:

```
git clone https://github.com/merto-dvp/spirit-docker-v2 && cd spirit-docker-v2/app && bash auto_install_script.sh
```

Now we continue with semi-auto script powered installation:

First of all Run `sudo apt update`

Make sure Git works. If not , run `sudo apt install git`

Clone the main repository from GitHub:

Run:

```
git clone https://github.com/merto-dvp/spirit-docker-v2
```

Navigate to folder spirit-docker-v2/app:

Run:

```
cd spirit-docker-v2/app
```

Run `install_script.sh` to start:

```
sudo bash install_script.sh
```

Wait until script loads everything. If docker hello-world app works, App folder & Docker are ready to serve.

This will run another script `spirit_install_app_script.sh` .

This script asks for your permission to install default project hosted on GitHub repo.

```
docker-compose version 1.17.1, build unknown
Done.
Synchronizing state of docker.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable docker
groupadd: group 'docker' already exists

Installed Docker and requirements successfully.

Installed Docker and requirements successfully.
Do you want to install a ready to serve project for Spirit?
It is ready to serve with Docker and will be downloaded from GitHub.
This will set Django(Spirit) app name as <mysite>.
You can cancel this for building app manually.
[Y/n] ? y
```

If you don't want to install default project, you can skip this by (N n) or Ctrl-C.

This project is made with default installation guide: <https://github.com/nitely/Spirit>

```
pip install django-spirit
spirit startproject mysite
cd mysite
python manage.py spiritinstall
python manage.py createsuperuser
python manage.py runserver
```

It is also hosted on GitHub: <https://github.com/merto-dvp/django-spirit-ready-app>

Project name will be set to <mysite>. App folder is configured to run mysite project as default.

Run `sudo docker-compose run web python3 manage.py createsuperuser` for creating an admin account.

After running this script, it is going to run Docker configurations and **serve the application at 0.0.0.0:8000**.

Manual Installation

If you don't want to install default project, you can run this script. But you have to configure the project by yourself. Use sudo if you get permission denied or related errors.

Script:

```
sudo bash manual_install_script.sh
```

This script will clean up & config the app folder. Then will install default Spirit/Django app with SQLite3 database and default settings. It will run default installation guide instructions to build up project.

<https://github.com/nitely/Spirit>.

After running script it will ask you to choose a project folder name for creating app.

```
Choose a project name for Spirit/Django app:
App:mysite

Creating Spirit app as: mysite
Creating spirit project...
ok
Operations to perform:
```

After successful creation and setting folders,

```
Created Spirit project as mysite
This is the default project for nitely/Spirit
For using it with Docker, you have to configure database and docker-compose.yml
Also you have to configure NGINX and gunicorn settings.
azireuser@ubuntu2:~/newtest2/spirit-docker-v2/app$ ls
Dockerfile      cleanup_script.sh  django.log         install.sh         manual_install_script.sh  spirit-install-ready-app.sh  spirit_install_app_script.sh
__init__.py     config             docker-compose.yml  install_script.sh  mysite                 spirit_after_db_script.sh   src
after_settings_script.sh  db.sqlite3        env.dev            manage.py          requirements.txt         spirit_init.sh              startup_script.sh
```

Now you have to configure Docker & Spirit app.

For changing database and using advantages of app/env.dev file
(ALLOWED_HOSTS,DEBUG,SECRET_KEY),change settings in projects settings folder.

For changing it, go to your (Django/Spirit) project folder (<mysite> for default)

<project_folder>/settings/dev.py

And use a text editor to edit dev.py. (Use sudo if required)

```
@ubuntu2:~/newtest2/spirit-docker-v2/app/mysite$ cd ..
@ubuntu2:~/newtest2/spirit-docker-v2/app$ cd mysite/settings
@ubuntu2:~/newtest2/spirit-docker-v2/app/mysite/settings$ ls
init      .py  __pycache__  base.py  dev.py  prod.py  test.py
@ubuntu2:~/newtest2/spirit-docker-v2/app/mysite/settings$ sudo nano dev.py
```

For using env.dev, import os package by adding

```
import os
```

to dev.py.

After that,

Overwrite DEBUG=True to :

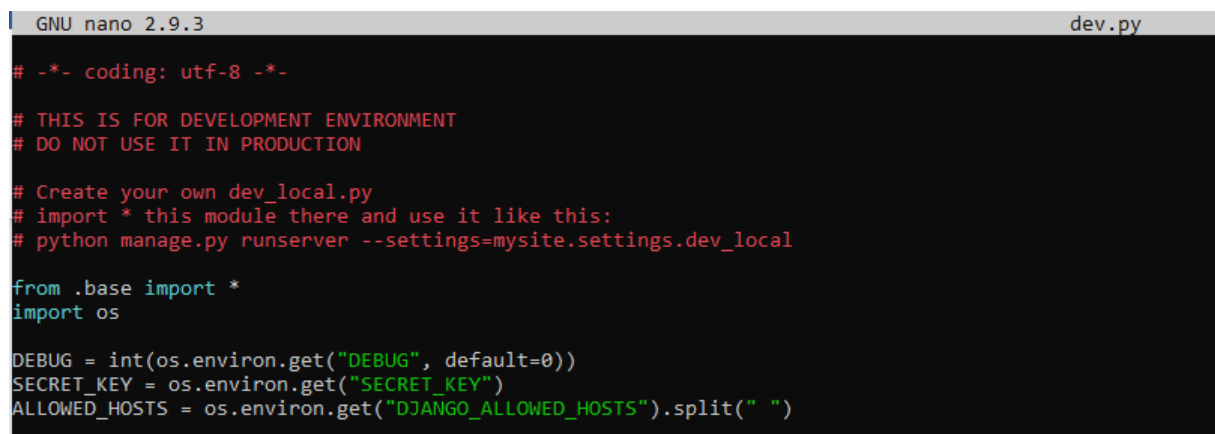
```
DEBUG = int(os.environ.get("DEBUG", default=0))
```

Overwrite SECRET_KEY = "DEV" to :

```
SECRET_KEY = os.environ.get("SECRET_KEY")
```

Overwrite ALLOWED_HOSTS = ['127.0.0.1',] to :

```
ALLOWED_HOSTS = os.environ.get("DJANGO_ALLOWED_HOSTS").split(" ")
```



```
GNU nano 2.9.3 dev.py
# -*- coding: utf-8 -*-
# THIS IS FOR DEVELOPMENT ENVIRONMENT
# DO NOT USE IT IN PRODUCTION
# Create your own dev_local.py
# import * this module there and use it like this:
# python manage.py runserver --settings=mysite.settings.dev_local
from .base import *
import os
DEBUG = int(os.environ.get("DEBUG", default=0))
SECRET_KEY = os.environ.get("SECRET_KEY")
ALLOWED_HOSTS = os.environ.get("DJANGO_ALLOWED_HOSTS").split(" ")
```

This is also required for avoiding DisallowedHost message. But you have to make sure env.dev file is ready.

Overwrite default DATABASE settings :

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

to:

```
DATABASES = {
    'default_old': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    },
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'postgres',
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': 'db',
        'PORT': 5432,
    }
}
```

```
# Database
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases
DATABASES = {
    'default_old': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    },
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'postgres',
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': 'db',
        'PORT': 5432,
    }
}
```

Now you can use PostgreSQL.

After overwriting all, save base.py and go back to project root folder. (cd ..)

Edit env.dev file with a text editor.

You can change

```
DEBUG=1 ( For enabling debug )
SECRET_KEY=j_o)qybin6wd60ph#z=iuuzhb7h45w8+sf6_$3iaz22^obifr1
DJANGO_ALLOWED_HOSTS=localhost 127.0.0.1 [::1] * 0.0.0.0
```

settings easily in this file.

```
GNU nano 2.9.3 env.dev
DEBUG=0
SECRET_KEY=j_o)qybin6wd60ph#z=iuuzhb7h45w8+sf6_$3iaz22^obifr1
DJANGO_ALLOWED_HOSTS=localhost 127.0.0.1 [::1] * 0.0.0.0
```

After changing settings and using with Docker, you also need to change NGINX and gunicorn settings. Because for default, they're set to work with <mysite> project folder name.

For NGINX settings:

```
cd config/nginx/
sudo nano nginx.conf
```

For gunicorn settings:

For running app you need to configure gunicorn settings. Edit `app/docker-compose.yml` with a text editor:

Change & overwrite gunicorn `mysite.wsgi` to:

```
gunicorn <project_name>.wsgi
```

command: `bash -c "python manage.py makemigrations && python manage.py migrate && gunicorn <project_name>.wsgi -b 0.0.0.0:8000"`

```

GNU nano 2.9.3                                     docker-compose.yml
version: "2"
services:
  nginx:
    image: nginx:latest
    container_name: NGINX-CONTAINER
    ports:
      - "8000:8000"
    volumes:
      - ./src:/src
      - ./config/nginx:/etc/nginx/conf.d
      - /static:/static <--- HERE
    depends_on:
      - web
  web:
    build: .
    container_name: DJANGO-CONTAINER
    command: bash -c "python manage.py makemigrations && python manage.py migrate && gunicorn .wsgi -b 0.0.0.0:8000"
    depends_on:
      - db
    volumes:
      - ./src:/src
      - /static:/static <--- HERE
    expose:
      - "8000"
    env_file:
      - ./env.dev
  db:
    image: postgres:latest
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=postgres
    container_name: PSQ-CONTAINER

```

your project name

You can configure Docker-compose settings here.

After all, simply run after_settings_script.sh :

```
bash after_settings_script.sh
```

This will configure Docker containers again and will serve the application.

Also you can run the <docker-compose up> command to serve application at any time.

Congratulations, your app works at <http://<IP address>:8000> !

```

** System restart required **
Last login: Sun Jan 3 13:20:52 2021 from 81. .164
@buntuv2:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
23933be3daa   nginx:latest   "/docker-entrypoint..." 7 minutes ago  Up 21 seconds  80/tcp, 0.0.0.0:8000->8000/tcp      NGINX-CONTAINER
0aeb335147c   app_web        "bash -c 'python man..." 7 minutes ago  Up 21 seconds  8000/tcp                           DJANGO-CONTAINER
afb507cda07   postgres:late  "docker-entrypoint.s..." 7 minutes ago  Up 22 seconds  5432/tcp                           PSQ-CONTAINER
@buntuv2:~$

```