DjangoRestFramework Microservice

Release 1.0rc1

Alain Ivars

Aug 03, 2022

CONTENTS:

CHAPTER

ONE

WHAT IS DRF-MICROSERVICE

drf-microservice is a ready-to-use API skeleton:

- Cookiescutter-drf-microservice generated it,

And you:

- add your unittest and endpoints,

And it will help you to:

- generate the documentation with Swagger-Coreapi,
- test it with Tox,
- package it Docker,
- deploy it (docker-compose), for Terraform or Ansible have a look at https://github.com/alainivars/utils2devops

Something disturb you in the code? Don't hesitate to open an issue and contribute.

Online documentation is here on Readthedoc Online source code available on Github

1.1 Usage

- **Now we just jump in the new directory and run tox to ::**
  - be sure that everything as worked fine
  - generate the documentation
  - generate an virtualenv

run tests:

```
cd drf-microservice
tox
```

- An virtualenv is already ready for you at

  ```
  tox -l
  py38-django31
  ```

- or you can create your

  ```
  python3 -m venv /pass/to/venv
  ```

- Activate it

```
source .tox/py38-django31/bin/activate
```

- Then

```
SECRET_KEY=my_secret_key python manage.py makemigrations
SECRET_KEY=my_secret_key python manage.py migrate
SECRET_KEY=my_secret_key python manage.py createsuperuser
```

- if you want enable the debug mode

```
DJANGO_ENABLE_DEBUG=1
```

nut if you don't you need to deploy the static files:

```
python manage.py collectstatic
```

- then run it

```
SECRET_KEY=my_secret_key python manage.py runserver
```

- the existing endpoints in production are:

```
/swagger/openapi(?P<format>\.json|\.yaml)
/swagger/openapi/
/swagger/redoc/
/admin/
/api-auth/
/api-auth-token/
/docs/
/icinga/
/icinga2/
/api/v1/file/
/media/(?P<path>.*)
```

- added endpoints for tests are:

```
/400/
/403/
/404/
/500/
```

1.2 Functionalities

- support basic auth
- support token auth
- endpoint json file POST,GET
- endpoint login/logout
- endpoint get tocken
- endpoint get status and running version

- postgreSQL support
- **doc Swagger-OpenApi v2:**

  this exposes 4 new endpoints:
  A JSON view of your API specification at /swagger/openapi.json A YAML view of your API specification at /swagger/openapi.yaml A swagger-ui view of your API specification at /swagger/openapi/ A ReDoc view of your API specification at /swagger/redoc/

  - **`API validation badge`_**
  - **`Code generation`_** with **`Swagger codeGen`_**

  And soo much more, take a look at: https://github.com/axnsan12/drf-yasg

1.2.1 Todo

- AWS ssm secret
- endpoint json file DELETE,PUT?
- **create different version:**
  - Aws S3 support (in progress)
  - Aws RDS support
  - Aws Elastisearch support
  - Redis support
  - Aerospike support
  - …

1.3 DevOps tools

- the dockerfile configuration file
- the docker-compose configuration file
- endpoint get status Nagios/Icinga2

1.3.1 Todo

- the dockerfile multi-stage configuration (in progress)
- the dockerfile TraefiK configuration (in progress)
- the Packer configuration file (in progress)
- the Terraform configuration file AWS (in progress)
- the Terraform configuration file GCD
- the Terraform configuration file Azure
- add getSentry support
- add Aws Cloudwatch support
- the Ansible configuration file AWS

- the Ansible configuration file GCD
- the Ansible configuration file Azure
- the Juju configuration file AWS
- the Juju configuration file GCD
- the Juju configuration file Azure
- Make static doc more modular & less duplicated

1.4 Interact with the API

To see the documentation for the API In development mode, login at

```
curl --request POST \
  --url http://127.0.0.1:8000/api-auth/login/ \
  --header 'content-type: application/json' \
  --data '{
    "username": "admin",
    "password": "admin"
    }'
```

Actually the default mode is “development” (same to the state of this project) in that mode a default login is the the db with username=’admin’ password=’admin’ you will get back in return your token:

```
{"key":"400a4e55c729ec899c9f6ac07818f2f21e3b4143"}
```

Then open to see the full auto-generated documentation of you API:

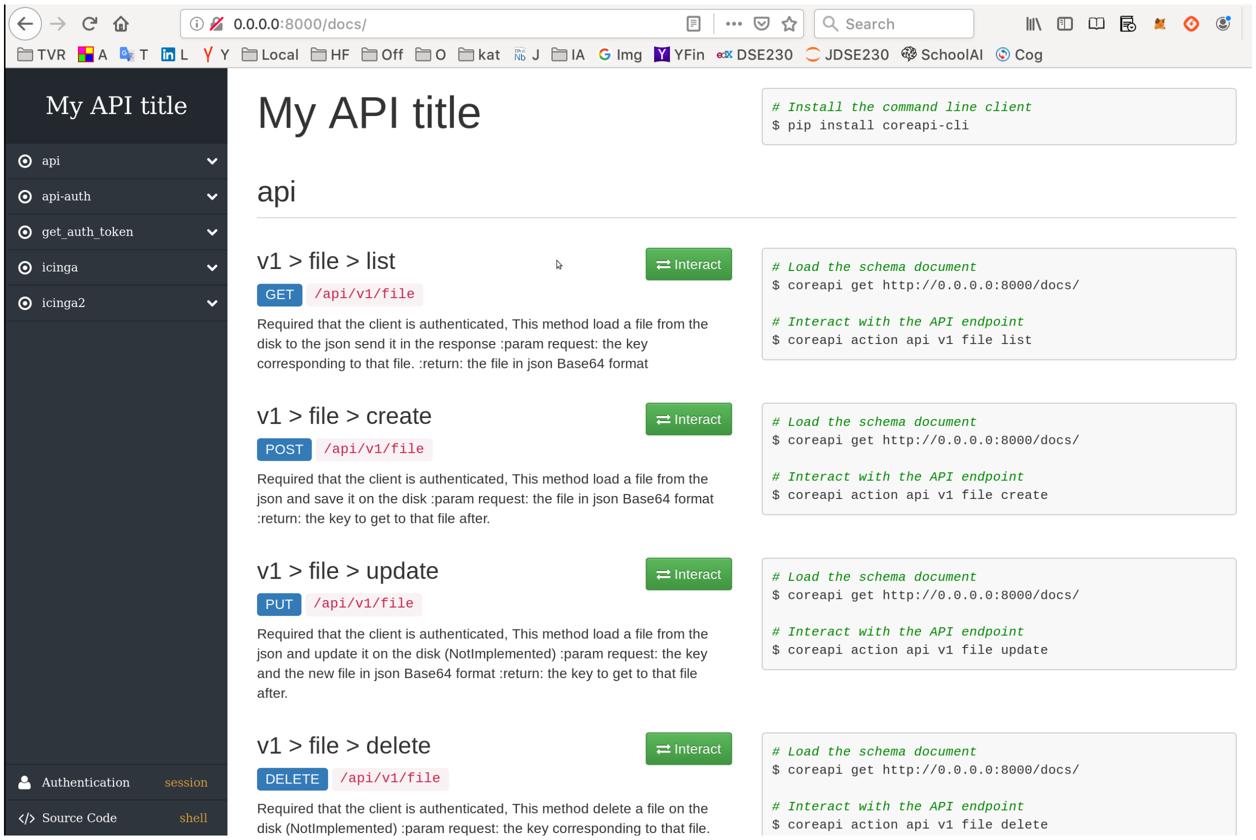
```
curl --request GET \
  --url http://127.0.0.1:8000/docs/ \
  --header 'authorization: Basic YWRtaW46YWRtaW4='
```

or by if BasicAuthentication is disabled and that wil be normally the case in prod and QA we use the Token:

```
curl --request GET \
  --url http://127.0.0.1:8000/docs/ \
  --header 'authorization: Token 400a4e55c729ec899c9f6ac07818f2f21e3b4143'
```

Then open

```
http://127.0.0.1:8000/docs/
```



My API title

```
# Install the command line client
$ pip install coreapi-cli
```

api

v1 > file > list

GET `/api/v1/file`

Required that the client is authenticated, This method load a file from the disk to the json send it in the response :param request: the key corresponding to that file. :return: the file in json Base64 format

```
# Load the schema document
$ coreapi get http://0.0.0.0:8000/docs/

# Interact with the API endpoint
$ coreapi action api v1 file list
```

v1 > file > create

POST `/api/v1/file`

Required that the client is authenticated, This method load a file from the json and save it on the disk :param request: the file in json Base64 format :return: the key to get to that file after.

```
# Load the schema document
$ coreapi get http://0.0.0.0:8000/docs/

# Interact with the API endpoint
$ coreapi action api v1 file create
```

v1 > file > update

PUT `/api/v1/file`

Required that the client is authenticated, This method load a file from the json and update it on the disk (NotImplemented) :param request: the key and the new file in json Base64 format :return: the key to get to that file after.

```
# Load the schema document
$ coreapi get http://0.0.0.0:8000/docs/

# Interact with the API endpoint
$ coreapi action api v1 file update
```

v1 > file > delete

DELETE `/api/v1/file`

Required that the client is authenticated, This method delete a file on the disk (NotImplemented) :param request: the key corresponding to that file.

```
# Load the schema document
$ coreapi get http://0.0.0.0:8000/docs/

# Interact with the API endpoint
$ coreapi action api v1 file delete
```

1.5 Testing

You can run the tests by

```
SECRET_KEY=my_secret_key python manage.py test
```

or by

```
python setup.py test
```

or by

```
DJANGO_SETTINGS_MODULE={{cookiecutter.app_name}}.config.local SECRET_KEY=my_secret_key␣
↪pytest
```

1.6 Security check

Before dockerization for deployment to production, don't forget to check if by

```
SECRET_KEY=my_secret_key python manage.py check --deploy
```

1.7 Build and run the image with Docker 2 different way

In both case, after the run, open a console to the docker container to run:

```
docker images | grep drf-ms-sqlite
Get the container id and with it:
docker exec -it be64cad1af93 bash
And inside the remote console:
export DJANGO_SECRET_KEY=local; export DJANGO_SETTINGS_MODULE=my_api.settings.local;␣
↪python manage.py createsuperuser
```

1.7.1 Build and run the image, all with docker-compose

You will need docker-compose compatible 3.8+ format, version 1.25.5+ https://github.com/docker/compose/releases/

Build and run with docker-compose:

```
docker-compose -f docker-compose.drf-ms-sqlite.yml up
```

Delete the container, the network and the image:

```
docker-compose -f docker-compose.drf-ms-sqlite.yml rm -f
docker network rm drf-microservice_default
docker rmi drf-ms-sqlite:latest
```

1.7.2 Build and run the image with Docker

Build the Docker image:

```
docker build -t drf-ms-sqlite:0.8.1wodc --label drf-ms-sqlite.wodc -f Dockerfile.drf-ms-
↪sqlite.wodc .
```

Run the container as service:

```
docker run -d -v "/home/a/repositories/dj/drf-microservice:/drf-microservice" -p␣
↪8000:8000 --name drf-ms-sqlite.wodc drf-ms-sqlite:0.8.1wodc
```

Stop, Delete the container and the image:

```
docker stop drf-ms-sqlite.wodc
docker rm drf-ms-sqlite.wodc
docker rmi drf-ms-sqlite:0.8.1wodc
```

1.8 Build and run the image with Docker

Pre-condition, set the required credential and secret:

```
# postgres
export POSTGRES_HOST=trust
export POSTGRES_HOST_AUTH_METHOD=trust
export POSTGRES_DB=postgres
export POSTGRES_USER=postgres
export POSTGRES_PASSWORD=postgres
# django postgres dev and test
export DB_ENGINE=django.db.backends.postgresql
export DB_HOST=127.0.0.1
export DB_PORT=5432
export DB_NAME=drfms_db
export DB_USER=drfms_user
export DB_PASS=drfms_pass
# django dev and test
export DJANGO_ENABLE_DEBUG=1
export DJANGO_SETTINGS_MODULE=my_api.settings.local
export DJANGO_SECRET_KEY=MyVerySecretKey
export DJANGO_SUPERUSER_USERNAME=superuser
export DJANGO_SUPERUSER_PASSWORD=password
export DJANGO_SUPERUSER_EMAIL=superuser@domain.local
```

Build and run with docker-compose:

```
docker-compose -f docker-compose.drf_ms_pg.yml up --force-recreate
docker-compose -f docker-compose.drf_ms_pg.yml exec -u postgres db1_pg psql -c 'CREATE␣
↪DATABASE drfms_db;'
docker-compose -f docker-compose.drf_ms_pg.yml exec -T -u postgres db1_pg psql drfms_db
↪< db_pg_initiate.sql
docker-compose -f docker-compose.drf_ms_pg.yml exec drf_ms_pg python manage.py migrate --
↪noinput
docker-compose -f docker-compose.drf_ms_pg.yml exec drf_ms_pg python manage.py␣
↪collectstatic --noinput
docker-compose -f docker-compose.drf_ms_pg.yml exec drf_ms_pg python manage.py␣
↪createsuperuser --username $DJANGO_SUPERUSER_USERNAME --email $DJANGO_SUPERUSER_EMAIL -
↪-noinput
docker-compose -f docker-compose.drf_ms_pg.yml down
```

Now everything is fine, you can run it as service:

```
docker-compose -f docker-compose.drf_ms_pg.yml up -d
```

Or Close and clean, remove All of the compose file, with permanents volume also:

```
docker-compose -f docker-compose.drf_ms_pg.yml down -v --rmi local --remove-orphans
or with also downloaded images
docker-compose -f docker-compose.drf_ms_pg.yml down -v --rmi all --remove-orphans
```

Or Close and clean, remove All of the compose file, except permanents volume (to save your data for future use):

```
docker-compose -f docker-compose.drf_ms_pg.yml down --rmi local --remove-orphans
or with also downloaded images
docker-compose -f docker-compose.drf_ms_pg.yml down --rmi all --remove-orphans
```

usefull cmds:

```
docker exec -it drf-microservice_db1_pg_1 psql -d drfms_db -U drfms_user
```

1.9 If you Use Aws

Aws secret required ???:: WORK IN PROGESS

APPNAME_USERNAME_PASSWD => a client API password SECRET_KEY => the secret key

Aws Env required:

```
AWS_REGION_NAME => default="eu-east-1"
AWS_APPNAME_SECRET_NAME =>The name of the secret bucket
```

1.10 Releases Notes

- 1.0rc1: Release candidate for version 1.0, update dependencies
- 0.8.4: Update documentation and fix docker-compose file for postgres 12.4
- 0.8.3: Update documentation
- 0.8.2: fix docker config file and docker-compose file for postgres 12.4
- 0.8.1: fix docker config file and docker-compose file for sqlite
- 0.8.0: Update dependencies
- 0.7.2: Add Swagger to CoreApi, compliance with OpenApi V2, Api code generator
- 0.7.1: doc modular & less duplicated, the docker config file
- 0.7.0: Cookiescutter-drf-microservice got it own separate repository
- 0.6.1: Update dependencies
- 0.6.0: total refactoring for add cookiecutter
- 0.5.2: fix dependencies security alert
- 0.5.1: fix some document presentation on github and pypi
- 0.5.0: Initial publication version

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search