Web Application Deployment

Environment

In order to separate deployment to development settings, a *settings* module is defined where the default settings for development are placed into the ***init***.*py*. A *production.py* can contain all the development settings and redefine some constant values.
The private CONSTANT values (secret key, database credential, static files configuration, etc.) are defined in a *envar.py* that is not tracked by *git*.
By moving the setttings files, the application directory path is also redefine:
`BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))`

Continuous integration

Connected to the GitHub account, Travis is configured to build tests on the specified repository branch. The *.travis.yml* define the Django settings module to the Travis settings file and the chome driver to use for tests: `- google-chrome-stable --headless --disable-gpu --remote-debugging-port=9222 https://www.chromestatus.com &`

set production setting / separate environment / env. variable

Modification du fichier de configuration Set config file for prod database

Deployment

Host

From DigitalOcean, an Infrastructure as a Service (IAAS), a virtual/cloud server is rented and used to host the web application. The aim is to be able to install every required software tool according to the system administrator's needs.

Server Configuration

A server space, also called ***Droplet*** by DigitalOcean, is created by defining the operating system distribution, the memory and computer ressources: Linux Ubuntu 20.04 (LTS) x64, 1 shared CPU, 1 GB RAM, 25 GB SSD Disk, 1000 GB bandwidth for data transfer.
To be nearest to the end-users' localization, London is selected as the server location.

To remote control the server, a previously generated SSH public key is also defined.

Finally, a firewall is added and defining InBound rules allowing ssh, http and https incoming traffic

Host server remote control

Using MS Windows, the SSH connection is performed with **PuTTY**. The same operations are also available using **OpenSSH** with a Linux distribution.

SSH keys generation

Putty allows to generate private and public keys using *Puttygen*. A **passphrase** is defined with private key for more security.
Besides, the format used by *PuTTYGen* for the public key is incompatible with *OpenSSH* on Linux servers, so a fix is described int a this article.

Authentication

With *PuTTY*, authentication and connection to the remote host server is performed by first defining the related private key location on the administration machine and the login username and IP address of the host server. Then, as the public key is defined on the host server we connect to this last one as ***root*** by running `root@ipaddress` command and specifying the *passphrase*.

In second, in order to preserve host server integrity, we define a *user* with restricted action compared with *root* and assign the ssh public key to this user. The login username is also set in *PuTTY* session configuration to perform connection by `user@ipaddress` command.

Web application installation

First, the libraries for *python* and *PostgreSQL* (database management) are installed.

- `sudo apt-get install python3-pip python3-dev libpq-dev virtualenv postgresql postgresql-contrib`

The application is downloaded from repository hosted on Github, then the virtual environnement is installed with the application requirements into the application directory.

Being logged in as administrator to the database management system, the application database and an assigned user are created according to the defined *Django* production settings. The database can be defined by Django with a *migration* command: `python manage.py migrate` and initialized if necessary by a *load data* command `python manage.py loaddata application_dump.json`.
Finally a *super user* is created by `python manage.py createsuperuser`.

The static files need to be generated for the production settings:

```
$ export ENV=PRODUCTION
$ python manage.py collectstatic
```

Web server

Nginx is used as HTTP server to provide the static files and redirect requests to the Django web application. The configuration and installation is described in this article.

Web application server: Gunicorn

supervisor

Gunicorn is define as wsgi application service and *supervisor* is the service to reload gunicorn if necessary.

Monitoring

Server ressources and performances

Once the application is online, it is interesting to monitore the host server state. *DigitalOcean* providing monitoring service, some thresholds can be defined to alert by sending an e-mail in case of insufficient performance or ressources.

Logging

Sentry is used as a dashbord to display all errors and logs of the Django web appication. After having created a Sentry project, the related token defined into the production settings file as ***Sentry DSN*** value. The level of logs is also set to `INFO` in order to collect by *Sentry* some events as request for a product that is not already strored in the database.

```
sentry_logging = LoggingIntegration(
    level=logging.DEBUG,        # Capture info and above as breadcrumbs
    event_level=logging.INFO  # Send errors as events
)

sentry_sdk.init(
    dsn=SENTRY_DSN,
    integrations=[DjangoIntegration(), sentry_logging],

    # If you wish to associate users to errors (assuming you are using
    # django.contrib.auth) you may enable sending PII data.
    send_default_pii=True
)
```

Automation

With the Linux **crontab** aplication, a cron job is defined to run every week an update of the product database from Openfoodfact:
`0 2 * * 2 sh /path/to/app/zcronjob.sh`
The script defines `PRODUCTION` as environment variable and the product update command to execute:
`/path/to/app/venv/bin/python /path/to/app/manage.py initializedatabase 0 >> /tmp/djg_opnfct_cron_sh.log`

Domain Name

IP address: 134.209.185.7