OC Pizza

Projet 9

Dossier d'exploitation

Version 0.1

Auteur
POUTOT Simon
Étudiant

Table des matières

1 - VERSIONS

Auteur	Date	Description	Version
POUTOT Simon	01/05/2023	Création du document	0.1

2 - Introduction

2.1 - Objet du document

Le présent document constitue le dossier d'exploitation de l'application Part2Pizza.

Objectif du document est de décrire précisément l'ensemble des configurations techniques liés aux serveurs du système informatique.

2.2 - Références

Pour de plus amples informations, se référer :

1. **PDF - 01** : POUTOT_Simon_1_dossier_conception_fonctionnelle
2. **PDF - 02** : POUTOT_Simon_2_dossier_conception_technique

3 - PRÉ-REQUIS

3.1 - Système

3.1.1 - Hébergement sur le cloud

Afin de déployer l'application sur un serveur, il faut déjà l'acheter ! Pour l'instant vous n'avez pas besoin de la puissance d'un ordinateur entier, vous pouvez donc opter pour un hébergement mutualisé. Concrètement, cela signifie que vous réservez une petite partie d'un serveur.

Digital Ocean est une entreprise de *Cloud Computing* qui offre de nombreux services. Simple d'utilisation, elle propose des serveurs à des prix très raisonnables. Pour les besoins de cette, réserverez un serveur à 5$ par mois.

3.1.1.1 - Caractéristiques techniques

Datacenter : London – Datacenter 1 – LON1

Image chosen : Ubuntu

Version : 22.10 x64

Size chosen / Droplet Type : Basic

CPU Option : Regular = Disk type : SSD

Authentication Method choosen : SSH Key

Enfin, Digital Ocean vous demande la clé publique. Si vous en avez déjà une, regardez dans le répertoire Users/vous/.ssh et copiez la clé présente dans le document id_rsa.pub.

```
$ cd /Users/celinems/.ssh/
$ cat id_rsa.pub
```

3.1.1.2 - Activer le pare-feu (Firewall)

Il est très simple d'activer un pare-feu sur Digital Ocean ! Dans l'interface d'administration, cliquez sur Networking > Firewalls > Create firewall.

Dans *Inbound Rules*, sélectionnez SSH (port 22) et HTTP (port 80).

Ajoutez le nom de votre *droplet* dans "Apply to Droplets" et cliquez sur "Create Firewall".

3.1.1.3 - Connexion en SSH – Via un nouvel utilisateur

Connectez-vous en ssh en tapant la commande root@IPDUSERVEUR.

```
$ ssh root@178.62.117.192
```

Créez un utilisateur ayant des droits restreints en tapant la commande adduser puis en renseignant les différentes valeurs demandées.

Ajouter l'utilisateur au groupe des super utilisateurs (sudo). Cela lui permettra d'utiliser la commande sudo.

```
root@disquaire:~# gpasswd -a celinems sudo
Adding user celinems to group sudo
```

(Pour toute la documentation, remplacer 'celinems' par votre nom d'utilisateur ainsi que 'disquaire' par le nom de votre droplet)

Puis associer votre clé publique à cet utilisateur.

Commencez par déclarer que vous êtes le nouvel utilisateur :

```
root@disquaire:~# su - celinems
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
celinems@disquaire:~$
```

Créez le dossier .ssh dans le répertoire de votre utilisateur et changez les permissions d'accès.

```
celinems@disquaire:~$ mkdir .ssh
celinems@disquaire:~$ chmod 700 .ssh
```

Créez ensuite un nouveau document pour héberger votre clé publique : authorized_keys.

```
celinems@disquaire:~$ vi .ssh/authorized_keys
```

Collez-y la clé publique copiée précédemment.

Tapez sur la touche esc (échappe) de votre clavier, puis :wq (*write and quit*) pour enregistrer les changements et quitter l'éditeur.

Puis changez les permissions d'accès au fichier authorized_keys et quittez l'utilisateur en tapant exit :

```
celinems@disquaire:~$ chmod 600 .ssh/authorized_keys
celinems@disquaire:~/.ssh$ exit
logout
root@disquaire:~#
```

Supprimez la méthode de connexion Root en modifiant la configuration générale du programme SSH. Elle se trouve dans le document sshd_config qui regroupe les réglages du programme.

```
root@disquaire:~# vi /etc/ssh/sshd_config
```

Puis trouvez la ligne PermitRootLogin yes. Elle spécifie que la connexion par l'utilisateur Root est possible. Remplacez yes par no.

Il faut recharger le programme afin que les changements soient pris en compte. Tapez la commande suivante :

```
root@disquaire:~# service ssh reload
```

Connectez-vous en tapant la commande ssh nomutilisateur@IP.

4 - Telechargez l'application sur le serveur distant

4.1 - Librairies utiles

Le projet est écrit en Python et utilise le système de gestion de bases de données PostgreSQL.

Installez ces programmes à l'aide de cette commande :

```
celinems@disquaire:~$ sudo apt-get update
celinems@disquaire:~$ sudo apt-get install python3-pip python3-dev libpq-dev postgresql postgresql-contrib
```

La commande apt-get update met à jour les programmes actuels du système. Il est important de le faire avant d'installer d'autres logiciels.

4.1.1 - Téléchargez l'application

Téléchargez l'application via Git dans le répertoire de l'utilisateur.

```
celinems@disquaire:~/disquaire$ git clone votredepot.git
```

4.1.2 - Environnement virtuel et dépendances

Installez Pip, Virtualenv et créez un nouvel environnement virtuel.

```
celinems@disquaire:~/$ sudo apt install virtualenv
celinems@disquaire:~/$ virtualenv env -p python3
celinems@disquaire:~/$ source env/bin/activate
(env) celinems@disquaire:~/$
```

Puis installez les dépendances du projet en tapant la commande suivante :

```
(env) celinems@disquaire:~/$ pip install -r disquaire/requirements.txt
```

4.2 - Migrer les données

4.2.1 - Création de la base de données

Connectez-vous en tant qu'administrateur à la console PostgreSQL en tapant la commande sudo -u postgres psql. Puis créez une nouvelle base de données.

Afin de créer l'utilisateur avec la commande : CREATE USER vous WITH PASSWORD 'motdepasse';.

Une dernière modification est nécessaire avant de poursuivre.

Si la base contient déjà la configuration demandée, l'ORM n'aura pas à les indiquer pour toute nouvelle connexion.

```
postgres=# ALTER ROLE celinems SET client_encoding TO 'utf8';
ALTER ROLE
postgres=# ALTER ROLE celinems SET default_transaction_isolation TO 'read committed';
ALTER ROLE
postgres=# ALTER ROLE celinems SET timezone TO 'Europe/Paris';
ALTER ROLE
```

Enfin, indiquez à PostgreSQL que l'utilisateur a tous les droits sur la base. Il pourra ainsi lire, écrire et supprimer des données !

```
postgres=# GRANT ALL PRIVILEGES ON DATABASE disquaire TO celinems;
GRANT
```

Tapez \q pour quitter la console de PostgreSQL.

4.2.2 - Modification du fichier de configuration

Modifiez le fichier settings.py pour y faire figurer les informations de connexion :

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'disquaire',
        'USER': 'celinems',
        'PASSWORD': '0+0=LaTeteàT0t0',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

4.2.3 - Génération des fichiers statiques

Générez-les en exécutant les commandes :

```
(env) celinems@disquaire~$ export ENV=PRODUCTION
(env) celinems@disquaire~$ ./disquaire/manage.py collectstatic
# ...
93 static files copied to '/home/celinems/disquaire/disquaire_project/staticfiles'.
```

Lancez les migrations :

```
(env) celinems@disquaire:~$ ./disquaire/manage.py migrate
```

Enfin, créez un administrateur :

```
(env) celinems@disquaire:~$ ./disquaire/manage.py createsuperuser
```

4.3 - Serveur web

Nginx (que l'on prononce ènjyne-x) est un logiciel libre de serveur web, écrit par Igor Sysoev, dont le développement a débuté en 2002 pour les besoins d'un site russe à très fort trafic (Rambler). Il est rapidement devenu très populaire grâce à sa syntaxe facile à prendre en mains. En septembre 2017, Nginx est utilisé par 20,75% des sites actifs dans le monde selon Netcraft.

Téléchargez-le sur le serveur en tapant la commande suivante :

```
sudo apt-get install nginx
```

4.3.1 - Configuration par défaut

La configuration par défaut de Nginx est située dans etc/nginx/sites-enabled/.

4.3.2 - Mise en application

Par convention, les fichiers de configuration sont regroupés dans le dossier sites-enabled.

Vous devez donc :

- créer un nouveau fichier dans sites-available ;
- ajouter un lien symbolique dans sites-enabled grâce à la commande ln.

```
celinems@disquaire:/etc/nginx/$ sudo touch sites-available/disquaire
[sudo] password for celinems:
celinems@disquaire:/etc/nginx/$ sudo ln -s /etc/nginx/sites-available/disquaire /etc/nginx/sites-enabled
```

4.3.2.1 - Servir l'application Django

Ouvrez le document etc/nginx/sites-available/disquaire que vous venez de créer avec cette commande :

`sudo vi sites-available/disquaire`

Puis modifiez le fichier comme ci-dessous avec vos propres informations :

```
server {

    listen 80; server_name 178.62.117.192;
    root /home/celinems/disquaire/;

    location / {
        proxy_set_header Host $http_host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_redirect off;
        proxy_pass http://127.0.0.1:8000;
    }

}
```

Rechargez Nginx en exécutant la commande reload.

```
celinems@disquaire:/etc/nginx$ sudo service nginx reload
```

Autorisez le nom de domaine en modifiant le fichier settings.py.

```
# ...
ALLOWED_HOSTS = ['178.62.117.192']
# ...
```

Supprimez les lignes suivantes du fichier de configuration de l'application Django :

```
MIDDLEWARE = [
    # ...
    # 'whitenoise.middleware.WhiteNoiseMiddleware', => delete this line
]

# ...

if os.environ.get('ENV') == 'PRODUCTION':
    # ...
    # Simplified static file serving.
    # https://warehouse.python.org/project/whitenoise/
    # STATICFILES_STORAGE = 'whitenoise.storage.CompressedManifestStaticFilesStorage' => delete this line

    # ...
```

Puis configurez Nginx.

etc/nginx/sites-available/disquaire

```
server {
    # ...

    location /static {
        alias /home/celinems/disquaire/disquaire_project/staticfiles/;
    }

    location / {
        proxy_set_header Host $http_host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_redirect off;
        if (!-f $request_filename) {
            proxy_pass http://127.0.0.1:8000;
            break;
        }
    }
}
```

Rechargez Nginx en exécutant la commande reload.

4.4 - Gunicorn

Gunicorn est le diminutif de *Green Unicorn*. Il s'agit d'un serveur HTTP Python pour Unix qui utilise les spécifications WSGI *(Web Server Gateway Interface)*. Particulièrement puissant, il est également rapide et facile à configurer.

Pour lancer le serveur Gunicorn, rien de plus simple : utilisez la commande gunicorn et passez en paramètre le module de configuration.

Exécutez donc la commande suivante :

```
(env) celinems@disquaire:~/disquaire$ gunicorn disquaire_project.wsgi:application
```

4.5 - Supervisor

Supervisor est un système qui permet de contrôler un ensemble de processus vivant dans un environnement UNIX. Pour faire simple, Supervisor lance des services et les redémarre s'ils échouent.

Installez-le avec cette commande :

```
sudo apt-get install supervisor
```

Supervisor lit un fichier de configuration situé dans etc/supervisor. Chaque fichier finissant en .conf et situé dans le chemin /etc/supervisor/conf.d représente un **processus** qui est surveillé par Supervisor. Créez-en un nouveau :

```
$ sudo vi /etc/supervisor/conf.d/disquaire-gunicorn.conf
```

À l'intérieur, ajoutez la commande à exécuter pour démarrer Gunicorn :

disquaire-gunicorn.conf

```
command=/home/celinems/env/bin/gunicorn disquaire_project.wsgi:application
```

Ajoutez plusieurs options telles que le nom du processus ou l'utilisateur Unix qui exécute la commande (root ou tout autre utilisateur).

```
[program:disquaire-gunicorn]
command = /home/celinems/env/bin/gunicorn disquaire_project.wsgi:application
user = celinems
directory = /home/celinems/disquaire
autostart = true
autorestart = true
```

Vous pouvez également passer des variables d'environnement ! Profitez-en et ajoutez-les :

```
environment = ENV="PRODUCTION",SECRET_KEY="unenouvelleclésecrete"
```

Il est temps de dire à Supervisor de lancer les processus !

```
$ sudo supervisorctl reread
disquaire-gunicorn: available
$ sudo supervisorctl update
disquaire-gunicorn: added process group
```

Vérifiez que le processus a bien été ajouté en tapant la commande status :

```
$ sudo supervisorctl status
disquaire-gunicorn                RUNNING   pid 16309, uptime 0:00:09
```

4.6 - Séparez les environnements

Il est d'usage de séparer les fichiers de configuration en fonction de l'environnement. La manière la plus simple d'effectuer cela est de créer un module contenant un fichier par environnement.

4.6.1 - Environnement local

Créez un nouveau module :

```
$ mkdir disquaire_project/settings
$ touch disquaire_project/settings/__init__.py
```

Par commodité, les réglages par défaut seront ceux de développement. Vous pourrez ensuite ajouter un fichier par environnement pour écraser les réglages par défaut.

Copiez l'intégralité de settings.py et collez-le dans __init__.py. Quand c'est fait, vous pouvez supprimer settings.py.
Puis parcourez le fichier __init__.py pour ne garder que les réglages spécifiques à votre environnement de développement local. Vous pouvez par conséquent supprimer les structures conditionnelles if os.environ.get('ENV') == 'PRODUCTION'

4.6.2 - Environnement de production

Le fichier concernant les réglages de production est particulier car il contient des informations sensibles : les identifiants de connexion à la base de données, la clé secrète…

Ce fichier de configuration ne doit exister **que sur le serveur** et nulle part ailleurs. Vous pouvez donc :

- créer un fichier .gitignore pour indiquer à Git de ne pas tracker production.py ;
- envoyer sur GitHub les modifications que vous venez juste d'apporter ;
- puller sur le serveur ces mêmes modifications ;
- créer un fichier production.py sur le serveur et y ajouter les informations.

Commencez par :

```
$ touch .gitignore
```

.gitignore

```
disquaire_project/settings/production.py
__pycache__
disquaire_project/staticfiles/
```

```
$ git add .gitignore disquaire_project/settings
$ git commit -m "new settings configuration"
$ git push origin master
```

Puis rendez-vous sur le serveur :

```
celinems@disquaire:~/disquaire$ git pull origin master
```

À présent, ajoutez-y un nouveau document qui regroupera les informations de production :

```
$ celinems@disquaire:~/disquaire$ touch disquaire_project/settings/production.py
```

settings/production.py

```python
from . import *

SECRET_KEY = '-~aO;| F;rE[??/w^zcumh(9'
DEBUG = False
ALLOWED_HOSTS = ['178.62.117.192']

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql', # on utilise l'adaptateur postgresql
        'NAME': 'disquaire', # le nom de notre base de données créée précédemment
        'USER': 'celinems', # attention : remplacez par votre nom d'utilisateur !!
        'PASSWORD': '0+0=LaTeteàT0t0',
        'HOST': '',
        'PORT': '5432',
    }
}

```

Indiquez à Django que vous voulez utiliser ce fichier de configuration et non *settings.py*.

Django cherche la variable d'environnement DJANGO_SETTINGS_MODULE pour connaître le module qui contient les configurations. Par défaut, il s'agit de votreprojet.settings.

Le changer en utilisant Supervisor.

```
celinems@disquaire:~/disquaire$ sudo vi /etc/supervisor/conf.d/disquaire-gunicorn.conf
```

Puis modifiez la ligne :

```
environment = DJANGO_SETTINGS_MODULE='disquaire_project.settings.production'
```

Indiquez à Supervisor que le fichier a été modifié :

```
(env) celinems@disquaire:~/disquaire$ sudo supervisorctl reread
disquaire-gunicorn: changed
(env) celinems@disquaire:~/disquaire$ sudo supervisorctl update
disquaire-gunicorn: stopped
disquaire-gunicorn: updated process group
(env) celinems@disquaire:~/disquaire$ sudo supervisorctl status
disquaire-gunicorn                RUNNING   pid 21764, uptime 0:00:09
```

5 - Intégrer des changements automatiquement

5.1 - Intégration continue (CI)

L'intégration continue *(Continuous Integration)* est une des pratiques les plus répandues des méthodologies de projet Agile. Elle permet de **diminuer le risque d'erreur** en production et d'**améliorer le temps consacré à l'intégration**.

Il est assez simple d'utiliser un service d'intégration continue qui va lancer des tests automatisés lorsqu'une *pull request* est acceptée. Voici comment cela se déroule :

- Une pull request est créée.
- Le service d'intégration continue lance l'application sur un serveur de test.
- Si les tests échouent, le service indique que la fonctionnalité n'est pas prête à être intégrée.
- Si les tests passent, le service affiche un voyant vert.
- Sonia peut alors relire sereinement la pull request. Si elle la valide, le code est intégré.

5.1.1 - Travis

Travis est un service d'automatisation qui s'interface parfaitement avec GitHub. Il permet de créer un environnement de développement rapidement, d'y faire tourner une application et d'y lancer des tests.

voici les différentes étapes à mettre en place pour utiliser Travis :

- indiquer à Travis de surveiller un dépôt GitHub ;
- intégrer un fichier de configuration *.travis.yml* dans le projet GitHub.

Voici, par exemple, un fichier *.travis.yml* :

```
language: python
python:
  - "3.5"
# command to install dependencies
install:
  - pip install -r requirements.txt
# command to run tests
script:
  - pytest # or py.test for Python versions 3.5 and below
```

Ce script va installer les dépendances listées dans le fichier *requirements.txt* puis exécuter la commande pytest.

5.1.1.1 - Activer l'intégration continue avec Travis CI

Rendez-vous à l'adresse https://www.travis-ci.com/ puis cliquez sur *"Sign in with GitHub"*. Connectez-vous en utilisant votre identifiant et votre mot de passe GitHub.

Vous êtes redirigé vers le tableau de bord de Travis. Il regroupe les différents projets pour lesquels vous avez activé l'intégration continue. Pour l'instant vous n'en avez aucun. Cliquez sur la croix à droite de l'onglet *"My repositories"* pour en ajouter un.

Travis affiche alors tous les dépôts publics de votre compte GitHub. Trouvez celui que vous avez forké tout à l'heure et activez-le en cliquant sur la croix grise.

5.1.1.2 - Créer un fichier de configuration

Comme pour les réglages de production, créez un nouveau fichier `travis.py` dans le dossier `settings`.

```python
from . import *

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': '',
        'USER': 'postgres',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
    },
}
```

Créez un nouveau document, *.travis.yml* au même niveau que *manage.py*.

```yaml
language: python
python:
  - '3.5'

# safelist
branches:
  only:
    - staging

before_script:
  - pip install -r requirements.txt

services:
  - postgresql

env: DJANGO_SETTINGS_MODULE=disquaire_project.settings.travis

script:
  - ./manage.py test
```

Envoyez les fichiers sur GitHub.

```
$ git add .travis.yml disquaire_project/settings/travis.py
$ git commit -m "update travis config"
$ git push origin staging
```

6 - Suveillez l'activité d'un serveur

6.1 - Activer le monitoring de Digital Ocean

Installez l'agent de Digital Ocean sur votre serveur :

```
$ curl -sSL https://agent.digitalocean.com/install.sh | sh
```

Puis allez dans l'interface d'administration de Digital Ocean. Dans votre *droplet* vous avez déjà accès à bien des informations.

6.2 - Configurer Sentry

Sentry est un tableau de bord qui vous permet de visualiser ce qui se passe dans l'application Django.

Créez-vous un compte.

Commencez par choisir Django et créez un nouveau projet. Ensuite, revenez dans la console et installez la librairie Raven.

```
$ pip install raven
```

Enfin, mettez à jour le fichier de configuration de production :

settings/production.py

```
import raven

INSTALLED_APPS += [
    'raven.contrib.django.raven_compat',
]


RAVEN_CONFIG = {
    'dsn': 'https://somethingverylong@sentry.io/216272', # caution replace by your own!!
    # If you are using git, you can also automatically configure the
    # release based on the git info.
    'release': raven.fetch_git_sha(os.path.dirname(os.pardir)),
}

LOGGING = {
    'version': 1,
    'disable_existing_loggers': True,
    'root': {
        'level': 'INFO', # WARNING by default. Change this to capture more than warnings.
        'handlers': ['sentry'],
    },
    'formatters': {
        'verbose': {
            'format': '%(levelname)s %(asctime)s %(module)s '
                      '%(process)d %(thread)d %(message)s'
        },
    },
```

```
    'handlers': {
        'sentry': {
            'level': 'INFO', # To capture more than ERROR, change to WARNING, INFO, etc.
            'class': 'raven.contrib.django.raven_compat.handlers.SentryHandler',
            'tags': {'custom-tag': 'x'},
        },
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
            'formatter': 'verbose'
        }
    },
    'loggers': {
        'django.db.backends': {
            'level': 'ERROR',
            'handlers': ['console'],
            'propagate': False,
        },
        'raven': {
            'level': 'DEBUG',
            'handlers': ['console'],
            'propagate': False,
        },
        'sentry.errors': {
            'level': 'DEBUG',
            'handlers': ['console'],
            'propagate': False,
        },
    },
}
```

Sauvez et relancez le serveur via Supervisor :

```
$ sudo supervisorctl restart disquaire-gunicorn
```

Regardez dans Sentry si un bug a fait son apparition.

7 - PROCÉDURE DE MISE À JOUR

7.1 - Tâche Cron

Cron est un programme qui permet aux utilisateurs des systèmes Unix d'exécuter automatiquement des scripts, des commandes ou des logiciels à une date et une heure spécifiée à l'avance, ou selon un cycle défini à l'avance.

7.1.1 - Planifier sa tâche cron

Créez un fichier crontab avec l'utilisateur actuel.

Tapez la commande suivante dans le terminal :

```
crontab -e
```

Y insérer cette commande :

```
0 0 * * 0 /home/simon/P8_01_site/env/bin/python /home/simon/P8_01_site/part2pizza/manage.py populate_bdd 10 500
```

Sauvegardez et quittez.

Pour vérifier qu'elle est bien opérationnelle, tapez cette commande :

```
crontab -l
```

Elle devrait s'afficher.