OC Pizza

Système de gestion informatique des pizzerias du groupe OC Pizza

Dossier d'exploitation

Version 1.0

Auteur
David Bouzerar
Développeur Backend

IT Consulting & Development

TABLE DES MATIERES

1 - VERSIONS

Auteur	Date	Description	Version
David Bouzerar	30/03/2021	Création du document	0.1
David Bouzerar	02/04/2021	Ajout des premiers chapitres	0.3
David Bouzerar	04/04/2021	Rédaction du document	0.6
David Bouzerar	07/04/2021	Finalisation du document	1.0

2 - INTRODUCTION

2.1 - Objet du document

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

L'objectif de ce document est d'apporter les informations nécessaires à la bonne exploitation du système et d'apporter un support si un problème technique intervient.

2.2 - Références

Pour de plus amples informations, se référer :

1. **DCF - 01** : Dossier de conception fonctionnelle
2. **DCT - 02** : Dossier de conception technique de l'application
3. **PV - 04** : PV de livraison

3 - PRE-REQUIS

3.1 - Système

3.1.1 - Serveur de Base de données

La base de données est hébergée sur un serveur virtuel dédié (souscrit chez **DigitalOcean**). Le **SGBDR** installé est **PostgreSQL 12**.

3.1.1.1 - Caractéristiques techniques

Le serveur dispose d'un OS **Linux** sans interface graphique. Dans un premier temps, nous utiliserons un seul CPU, 2 Go de Ram et 50 Go de stockage et ajusterons, si cela s'avère utile, en fonction des retours de l'interface de surveillance de **DigitalOcean**.

Ubuntu Server 20.04 (basée sur **Debian**) est la distribution installée. Il s'agit d'une version **LTS** dont la fin du support prendra fin en Avril 2025.

3.1.2 - Serveur d'application

L'application est hébergée sur un serveur virtuel dédié (souscrit chez **DigitalOcean**). **Python version 3.7** et **Django version 3.2** ainsi que les dépendances y sont installées. Nous y retrouvons également le serveur WEB **NGINX 1.18** ainsi que les **SDK Sentry** et client **Newrelic** nécessaires à la surveillance du serveur et de l'application.

3.1.2.1 - Caractéristiques techniques

Le serveur dispose d'un OS **Linux** sans interface graphique. Dans un premier temps, nous utiliserons un seul CPU, 2 Go de Ram et 50 Go de stockage. S'il s'avère que les ressources soient trop limitées, nous pourrons les ajuster en fonction des retours obtenus sur **NewRelic** et l'interface de surveillance de **DigitalOcean**.

Ubuntu Server 20.04 (basée sur **Debian**) est la distribution installée. Il s'agit d'une version **LTS** dont la fin du support prendra fin en Avril 2025.

3.2 - Bases de données

Le **modèle physique de données** est visible dans le dossier de conception technique de l'application, ainsi que dans la partie « ressources » du **repository**.

Les modèles sont créés à partir des informations fournies par ce document. Si des modifications sont effectuées sur la base, il est important de maintenir à jour ce document.

Il est important de noter que la base sera commune à toutes les pizzérias. Les « responsables » pourront consulter les données des différentes pizzerias et effectuer des exports dans le but de réaliser des statistiques.

3.3 - Web-services

3.3.1 - API Google Maps

L'**API Google Map** sera utilisée lors de la livraison des commandes. Elle permet notamment de géo-localiser le « Livreur » et de le diriger vers l'adresse du client.

L'**API** permet de calculer la distance et le temps restant à parcourir. Elle permet également de déterminer, dans le cas d'une livraison, que l'adresse saisie se situe bien dans le périmètre de livraison déterminé par les responsables du groupe.

4 - PROCEDURE DE DEPLOIEMENT

4.1 - Déploiement

4.1.1 - Paramétrer l'accès aux serveurs

Comme expliqué précédemment, l'application est hébergée chez **DigitalOcean**. Si, pour une raison quelconque, il est nécessaire de changer d'hébergeur, il faudra donc souscrire à une des offres proposées puis choisir une distribution **Linux** qui sera installée sur la machine virtuelle. Concernant le choix de l'offre il faudra l'adapter selon les ressources nécessaires, à savoir le nombre de **CPU**, la quantité de **RAM** ainsi que le stockage.

Une fois cela fait, il est nécessaire de permettre la connexion **SSH** sur le serveur. Pour cela, ouvrir le port **TCP 22** nécessaire dans le cadre d'une connexion **SSH**. Il est également nécessaire d'autoriser le trafic sur les ports **TCP 80** (**HTTP**) et **TCP 443** (**HTTPS**).

A partir de là, se connecter au serveur grâce à la commande :

```
ssh root@(IP_DU_SERVEUR)
...
```

Puis ajouter un nouvel utilisateur pour autant que nécessaire :

```
root@(SERVER_NAME):~# adduser (USERNAME)
Enter new UNIX password :
Retype new UNIX password :
...
```

Autoriser ce nouvel utilisateur à exécuter la commande « sudo » :

```
root@(SERVER_NAME):~# gpasswd -a (USERNAME)
Adding user (USERNAME) to group sudo
...
```

Basculer sur l'utilisateur nouvellement créé :

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

(USERNAME)@(SERVER_NAME):~$
...
```

Créer un fichier « *authorized_keys* » dans le répertoire « *.ssh* » et y coller la clé publique de l'utilisateur de la session. Si le répertoire n'existe pas, le créer et modifier les permissions :

```
root@(SERVER_NAME):~# mkdir .ssh
root@(SERVER_NAME):~# chmod 700 .ssh
...
```

Pour des raisons de sécurité, il est recommandé, une fois les utilisateurs créés, d'empêcher la connexion **SSH** au serveur par l'intermédiaire du compte **Root**. Pour cela, éditer le fichier de configuration :

```
root@(SERVER_NAME):~# vi /etc/ssh/sshd_config
...
```

Puis modifier la valeur de « *PermitRootLogin* » sur « *no* » :

```
PermitRootLogin no
```

Et recharger le « *process* » **SSH** pour appliquer les modifications :

```
root@(SERVER_NAME):~# service ssh reload
...
```

Pour se reconnecter au serveur, il faudra maintenant saisir l'**UID** d'un utilisateur :

```
ssh (USERNAME)@(IP_DU_SERVEUR)
...
```

4.1.2 - Uploader l'application sur le serveur

4.1.2.1 - Installation des dépendances utiles

Avant toute chose, nous devons nous assurer que le serveur dispose des dernières mises à jour et le cas échéant, les installer :

```
(USERNAME)@(SERVER_NAME):~/$ sudo apt update && apt full-upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
...
```

Puis procéder à l'installation des dépendances :

```
(USERNAME)@(SERVER_NAME):~/$ sudo apt install python3-pip python-dev libpq-dev postgresql
postgresql-contrib

...
```

4.1.2.2 - Upload

Il suffit pour cela de cloner le **repository Gi**t de l'application :

```
(USERNAME)@(SERVER_NAME):~/$ git clone oc_pizza_project.git
Cloning into 'oc_pizza_project'...
remote: Counting objects: 740, done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 740 (delta 3), reused 11 (delta 1), pack-reused 724
Receiving objects: 100% (740/740), 1002.34 KiB | 0 bytes/s, done.
Resolving deltas: 100% (346/346), done.
Checking connectivity... done.
...
```

4.1.3 - Environnement virtuel

4.1.3.1 - Création de l'environnement

Mettre en place l'environnement virtuel, pour cela, installer « **pip** » et « **venv** »

```
(USERNAME)@(SERVER_NAME):~/$ sudo apt install python3-venv
...
```

Puis créer le nouvel environnement virtuel :

```
(USERNAME)@(SERVER_NAME):~/$ python3 -m venv project_env
...
```

Il ne reste plus, ensuite, qu'à l'activer :

```
(USERNAME)@(SERVER_NAME):~/$ source project_env/bin/activate
(project_env) (USERNAME)@(SERVER_NAME):~/$
...
```

4.1.3.2 - Installation des dépendances de l'application

```
(project_env) (USERNAME)@(SERVER_NAME):~/oc_pizza_project$ pip install -r requirements.txt
...
```

4.1.4 - Base de données

4.1.4.1 - Création de la base

Concernant la création de la base de données, il est nécessaire de préalablement faire quelques ajustements. Nous devons dans un premier temps créer la base, puis un utilisateur à qui l'on attribuera des droits sur la base.

Pour cela, il est nécessaire d'accéder à la console **PSQL** en procédant de la façon suivante :

```
(project_env) (USERNAME)@(SERVER_NAME):~/ sudo -u postgres psql
psql (10.16 (Ubuntu 10.16-0ubuntu0.18.04.1))
Type "help" for help.

postgres=#
...
```

Une fois connecté sur la console **PSQL**, nous pouvons créer la base de données, ainsi que le nouvel utilisateur :

```
postgres=# CREATE DATABASE ocpizza;
CREATE DATABASE

postgres=# CREATE USER (username) WITH PASSWORD '(password)';
CREATE ROLE
...
```

La base et l'utilisateur sont maintenant créés. Il est toutefois nécessaire, d'après la documentation officielle de Django, de modifier certains paramètres de la base de données, dans le but d'améliorer les performances lors du traitement des requêtes :

```
postgres=# ALTER ROLE (username) SET client_encoding TO 'utf8';
ALTER ROLE

postgres=# ALTER ROLE (username) SET default_transaction_isolation TO 'read committed';
ALTER ROLE

postgres=# ALTER ROLE (username) SET timezone TO 'Europe/Paris';
ALTER ROLE
...
```

Enfin, il ne reste plus qu'à attribuer tous les droits sur la base à notre nouvel utilisateur. Nous pourrons ainsi l'utiliser pour lire, écrire et supprimer des données.

```
postgres=# GRANT ALL PRIVILEGES ON DATABASE ocpizza TO (username);
GRANT
...
```

Il ne reste plus qu'à quitter la console **PSQL** :

```
postgres=# \q

(project_env) (USERNAME)@(SERVER_NAME):~/
...
```

4.1.4.2 - Fichier de configuration

Il est nécessaire de renseigner le fichier de configuration de l'application afin qu'elle puisse accéder à la base de données nouvellement créée. Pour cela, nous devons éditer le fichier « *production.py* » situé dans le dossier « *settings* » de l'application :

```
...
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': ocpizza,
        'USER': '(USERNAME)',
        'PASSWORD': '(PASSWORD)',
        'HOST': '(IP_DU_SERVEUR_DB)',
        'PORT': '5432',
    }
}
...
```

4.1.4.3 - Migrations

La base est maintenant créée et configurée pour que l'application puisse y accéder. Il est maintenant nécessaire faire les migrations **Django** :

```
(project_env) (USERNAME)@(SERVER_NAME):~/ ./oc_pizza_project/manage.py migrate
Operations to perform:
  Apply all migrations: admin, ...
Running migrations:
...
```

Cela aura pour effet de créer les tables, contraintes etc… de notre base de données. Il est possible d'importer directement des données compatibles, pour cela, partons du principe que nous possédons un fichier nommé « *store.json* » :

```
(project_env) (USERNAME)@(SERVER_NAME):~/ ./oc_pizza_project/manage.py loaddata
./dumps/store.json
Installed 476 object(s) from 1 fixture(s)
...
```

4.1.5 - Gestion des statiques

Pour générer les fichiers statiques, il est nécessaire d'indiquer à **Django** de le faire. Pour cela, le fichier de configuration contient une constante nommée « *STATIC_ROOT* » nécessaire à la collecte de ces fichiers statiques. Cette constante existe uniquement si la variable d'environnement « *ENV* » existe et qu'elle possède la valeur « *PRODUCTION* ».

Pour le moment cette variable n'existe pas, il est donc nécessaire de la créer :

```
(project_env) (USERNAME)@(SERVER_NAME):~/ export ENV=PRODUCTION
...
```

Dans ce cas, la variable d'environnement n'existera que dans l'instance « *SSH* » en cours. Elle est volatile, mais cela suffira pour générer les statiques, nous verrons plus loin comment la rendre persistante.

La variable d'environnement est maintenant créée, il ne reste donc plus qu'à générer les fichiers statiques :

```
(project_env) (USERNAME)@(SERVER_NAME):~/ ./oc_pizza_project/manage.py collectstatic
# ...
93 static files copied to '/home/(USER)/oc_pizza/oc_project/staticfiles'
...
```

4.1.6 - Créez un administrateur

Il ne nous reste plus qu'à créer un utilisateur possédant les droits « *administrateur* » pour l'interface d'administration de l'application :

```
(project_env) (USERNAME)@(SERVER_NAME):~/ ./oc_pizza_project/manage.py createsuperuser
...
```

4.2 - Configuration du serveur HTTP

4.2.1 - Installation NGINX

L'installation du serveur **HTTP NGINX** se fait facilement mais nécessite une petite configuration. Débutons par le téléchargement / installation de l'application :

```
(project_env) (USERNAME)@(SERVER_NAME):~/ sudo apt-get install nginx
...
```

Si l'installation s'est bien déroulée, nous devrions tomber sur la page « **Welcome to nginx !** » en saisissant l'adresse IP publique du serveur sur un navigateur WEB.

4.2.2 - Configuration NGINX

4.2.2.1 - Création du fichier de configuration

Les liens vers les fichiers de configuration sont regroupés dans le dossier « *sites-enabled* ». Il est donc nécessaire d'y indiquer le chemin vers le fichier de configuration de l'application :

Il suffit donc de créer le fichier de configuration puis de créer un lien symbolique vers celui-ci :

```
(project_env) (USERNAME)@(SERVER_NAME):~/ sudo touch /etc/nginx/sites-available/oc_pizza
...
(project_env) (USERNAME)@(SERVER_NAME):~/ sudo ln –s /etc/nginx/sites-available/oc_pizza
/etc/nginx/sites-enabled
...
```

Une entrée vers le fichier de configuration doit apparaître maintenant dans « *sites-enabled* » :

```
(project_env) (USERNAME)@(SERVER_NAME):~/ vi /etc/nginx/sites-enabled



"===============================================================================
" Netrw Directory Listing                                        (netrw v156)
"   /etc/nginx/sites-enabled
"   Sorted by      name
"   Sort sequence:
[\/]$,\<core\%(\.\d\+\)\=\>,\.h$,\.c$,\.cpp$,\~\=\*$,*,\.o$,\.obj$,\.info$,\.swp$,\.bak$,\~$
"   Quick Help: <F1>:help  -:go up dir  D:delete  R:rename  s:sort-by  x:special
"===============================================================================
../
./
default@                          --> /etc/nginx/sites-available/default
oc_pizza@                         --> /etc/nginx/sites-available/oc_pizza
```

4.2.2.2 - Configuration

Il reste maintenant à configurer **NGINX** pour l'application *OC_Pizza*. **NGINX** permet s'occuper également de gérer les fichiers statiques.

Si un fichier statique est demandé, **NGINX** doit rediriger vers celui-ci sans passer par l'application, sinon, il redirigera le trafic vers l'application.

Ouvrons dans un premier temps le fichier de configuration nouvellement créé :

```
(project_env) (USERNAME)@(SERVER_NAME):~/ sudo vi /etc/nginx/sites-available/oc_pizza
...
```

Voici maintenant la configuration que doit contenir ce fichier :

```
server {
    listen 80;
    server_name (SERVER_DOMAIN_NAME);
    root (APPLICATION_ABSOLUTE_PATH);

    location /static {
        alias (APPLICATION_ABSOLUTE_PATH)/oc_pizza_app/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;
        }
    }
}
```

Voici à quoi doit ressembler le fichier de configuration **NGINX** pour l'application **OC_PIZZA**. La partie « *SERVER_DOMAIN_NAME* » doit être remplacée par le nom de domaine préalablement souscrit. Si nous en sommes à un stade où nous n'avons pas encore souscrit à un nom de domaine, il est possible de le remplacer par l'adresse IP publique du serveur.

La partie « *APPLICATION_ABSOLUTE_PATH* » doit être remplacée par le chemin absolu de l'application sur le serveur.

Pour le reste, nous pouvons voir que le bloc « *location /static* » contient le chemin vers les fichiers statiques de l'application. **NGINX** s'appuiera donc là-dessus et sur le chemin de l'application pour la redirection.

Il ne reste plus qu'à charger la nouvelle configuration, il faut pour cela relancer le serveur **NGINX** :

```
(project_env) (USERNAME)@(SERVER_NAME):~/ sudo service nginx reload
...
```

Il est également nécessaire de mettre à jour la partie « *ALLOWED_HOSTS* » du fichier de settings « *production.py* » de l'application :

```
...
ALLOWED_HOSTS = ['(SERVER_DOMAIN_NAME)']
...
```

Il faut donc saisir le nom de domaine de l'application ou l'IP publique si nous n'en avons pas pour le moment.

4.2.3 - Configuration de Supervisor & Gunicorn

4.2.3.1 - Installation de Supervisor

Comme expliqué précédemment, **Supervisor** est un gestionnaire de services permettant de démarrer des services et les surveiller. Dans le cas présent, il nous permet de lancer « **Gunicorn** » au démarrage du serveur puis le surveiller.

L'installation se fait simplement par le gestionnaire de package **Ubuntu** :

```
(project_env) (USERNAME)@(SERVER_NAME):~/ sudo apt-get install supervisor
...
```

4.2.3.2 - Configuration de Supervisor

Supervisor s'appuie sur un fichier de configuration situé dans « */etc/supervisor/conf.d* ». Chaque fichier « *.conf* » représente un processus dont **Supervisor** doit s'occuper.

Il faut donc en créer un pour l'application :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo touch /etc/supervisor/conf.d/oc_pizza_gunicorn.conf
...
```

Puis, ensuite, l'éditer :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo vi /etc/supervisor/conf.d/oc_pizza_gunicorn.conf
...
```

```
[program:oc_pizza]
command = (APPLICATION_ABSOLUTE_PATH)/bin/newrelic-admin run-program
(APPLICATION_ABSOLUTE_PATH)/bin/gunicorn oc_pizza_app.wsgi:application
user = (USERNAME)
directory = (APPLICATION_ABSOLUTE_PATH)/oc_pizza_app
autostart = true
autorestart = true
```

```
environment = DJANGO_SETTINGS_MODULE='oc_pizza_app.settings.production',SECRET_KEY="(SECRET_KEY)
",NEW_RELIC_CONFIG_FILE=(APPLICATION_ABSOLUTE_PATH)/oc_pizza_app/newrelic.ini
```

La configuration proposée ci-dessus contient également les informations concernant **NewRelic**. Se reporter à la rubrique concernée pour le mettre en place.

Pour le reste, voici les champs à modifier :

- *APPLICATION_ABSOLUTE_PATH* : Chemin absolu de l'application sur le serveur
- *USERNAME* : Le nom de l'utilisateur chargé du lancement de l'application
- *SECRET_KEY* : Clé secrète de l'application utilisée par **Django**

A noter que nos variables d'environnements sont renseignées dans ce fichier de configuration.

NGINX étant absolument primordial, ajoutons également une surveillance sur celui-ci :

```
[program:nginx]
command = /usr/sbin/nginx -g "daemon off;"
autostart = true
autorestart = true
numprocs = 1
startsecs = 0
process_name=%(program_name)s_%(process_num)02d
stderr_logfile=/var/log/supervisor/%(program_name)s_stderr.log
stderr_logfile_maxbytes=10MB
stdout_logfile=/var/log/supervisor/%(program_name)s_stdout.log
stdout_logfile_maxbytes=10MB
```

Il ne reste plus qu'à demander à **Supervisor** de prendre en compte les nouveaux processus à surveiller :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo supervisorctl reread
nginx: available
oc_pizza: available
...
```

Puis lui demander de démarrer la supervision des nouveaux processus :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo supervisorctl update
nginx: added process group
oc_pizza: added process group
...
```

On s'assure enfin que **Supervisor** est actif et surveille bien les processus :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo supervisorctl status
nginx                            RUNNING   pid 18429, uptime 0:01:47
oc_pizza                         RUNNING   pid 16309, uptime 0:01:47
...
```

5 - SUPERVISION/MONITORING

5.1 - Sentry

L'installation de **Sentry**, qui pour rappel va nous permettre d'avoir un retour sur les erreurs et les exceptions rencontrées éventuellement rencontrées par l'application. Il nous servira également de gestionnaire des **logs** sur certains éléments suivis de l'application.

5.1.1 - Installation

SDK Sentry est un package **Python**. Il est donc normalement installé en même temps que les « *requirements* » de l'application. Si ce n'est pas le cas, l'installation se fait simplement par la commande suivante :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ pip install --upgrade sentry-sdk
...
```

5.1.2 - Configuration

La configuration de **Sentry** se fait par l'intermédiaire du fichier « *production.py* » dans les « *settings* » de l'application :

```
import logging
import sentry_sdk

from sentry_sdk.integrations.django import DjangoIntegration
from sentry_sdk.integrations.logging import LoggingIntegration

...

sentry_logging = LoggingIntegration(
    level=logging.INFO,
    event_level=logging.INFO
)

sentry_sdk.init(
    dsn="(LINK_WITH_SENTRY_ACCOUNT)",
    integrations=[DjangoIntegration(), sentry_logging],
    traces_sample_rate=1.0,
    send_default_pii=True
)

...
```

Les erreurs et exceptions seront maintenant automatiquement remontées vers le « *Dashboard* » **Sentry** de l'application. A noter que les **logs** affichés par la librairie « *logging* » seront également remontés et porteront le « *tag* » **INFO**.

5.2 - New Relic

Comme indiqué précédemment, **NewRelic** permet de surveiller, entre autres, les performances de l'application.

5.2.1 - Installation

A la manière de **Sentry**, **NewRelic** possède également un package **Python** pouvant être installé par l'intermédiaire de **pip** :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ pip install newrelic
...
```

5.2.2 - Configuration

Une fois installé, il est nécessaire de générer un fichier de configuration à la racine du projet. **NewRelic** permet de le générer automatiquement, il suffit de saisir la commande suivante :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ newrelic-admin generate-config (LICENCE_KEY)
newrelic.ini
...
```

Il est important de saisir la « *LICENCE_KEY* » donnée lors de la création de l'entrée dans le *Dashboard* de **NewRelic**. La configuration est alors terminée et les données remonteront automatiquement. Le service est exécuté automatiquement puisque nous l'avons déjà intégré dans notre fichier de configuration **NGINX**, lors du lancement de **GUNICORN**.

6 - PROCEDURE DE DEMARRAGE

6.1 - Base de données

Si pour une raison ou une autre il est nécessaire de démarrer, manuellement, le service lié à **PostgreSQL**, saisir la commande suivante :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo service postgresql start
...
```

6.2 - Application

Le démarrage de l'application se fait par l'intermédiaire du fichier de configuration **Supervisor** de celle-ci. Il n'est donc pas nécessaire de la démarrer manuellement. A savoir que l'application de **Monitoring NewRelic** est également gérée par **Supervisor**. Les variables d'environnements sont également déclarées dans le fichier de configuration **Supervisor**.

6.2.1 - Installation des dépendances

L'installation des dépendances de l'application se fait par l'intermédiaire du gestionnaire de paquets **pip**. Elles sont contenues dans le fichier « *requirements.txt* » à la racine du projet, pour les installer, simplement saisir la commande :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ pip install -r requirements.txt
...
```

6.2.2 - Génération des fichiers statiques

Nous en avons déjà parlé dans un chapitre précédent, mais pour rappel, voici la commande à saisir pour indiquer à **Django** de générer les statiques :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ python manage.py collectstatic
...
```

6.2.3 - Migrations

```
(project_env)(USERNAME)@(SERVER_NAME):~/ python manage.py makemigrations
...
(project_env)(USERNAME)@(SERVER_NAME):~/ python manage.py migrate
...
```

6.2.4 - Supervisor

```
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo service supervisor start
...
(project_env)(USERNAME)@(SERVER_NAME):~/ python manage.py migrate
...
```

7 - PROCEDURE DE MISE A JOUR

7.1 - Base de données

S'il s'avère nécessaire de redémarrer manuellement **PostgreSQL** (dans le cas d'une modification de la configuration par exemple), saisir la commande suivante :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo service postgresql restart
...
```

7.2 - Application

Concernant l'application, si une mise à jour est effectuée et dans la mesure ou son lancement est géré par **Supervisor**, il faudra préalablement stopper **Supervisor** avant de le redémarrer une fois la mise à jour effectuée.

Par contre, si une mise à jour est effectuée côté **Supervisor**, il sera nécessaire d'exécuter les commandes suivantes :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo supervisorctl reread
...
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo supervisorctl update
...
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo supervisorctl restart
...
```

Cela aura pour effet de prendre en charge les modifications apportées puis de redémarrer **Supervisor**.

8 - PROCEDURE D'ARRET

8.1 - Base de données

Pour stopper l'activité du serveur **PostgreSQL**, saisir la commande :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo service postgresql stop
...
```

8.2 - Application

Pour stopper l'application, il est nécessaire d'arrêter le service **Supervisor** :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo service supervisor stop
...
```

8.3 - Redémarrage du serveur Linux

Pour redémarrer le système d'exploitation :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo /sbin/reboot
...
```

Pour arrêter le serveur immédiatement :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo shutdown -h now
...
```

Ou de façon différée :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ sudo shutdown -h 30
...
```

Pour éteindre le serveur 30 minutes plus tard

9 - SAUVEGARDE ET RESTAURATION

Concernant la sauvegarde de la base de données, une tâche **CRON** tourne toutes les 4h sur le serveur. Nous pouvons l'éditer de la façon suivante :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ crontab -e
...
```

```
...
*/4 * * * * backup_db.sh >> /var/log/oc_pizza/backup_db.log 2>&1
...
```

Il est possible de consulter un fichier **log** de la sauvegarde dans « */var/log/oc_pizza* »

Concernant la restauration de la base en cas de problème quelconque, il faudra saisir la commande suivante :

```
(project_env)(USERNAME)@(SERVER_NAME):~/ python manage.py loaddate (DATABASE_BACKUP)
...
```

10 - INTEGRATION CONTINUE

Comme expliqué dans les autres documents, nous utilisons un service d'intégration continue nommée **Travis CI**. Ce chapitre se trouve à la fin du document car il n'est pas lié au serveur de production lui-même mais plutôt au développement de l'application, ou dans le cadre du développement d'une nouvelle fonctionnalité par exemple.

Il est lié au « *repository* » **Git** et se charge de surveiller les **pull-requests** sur la branche « *dev* » de l'application. Lors d'un push sur la branche, Travis met en place un environnement permettant d'exécuter les différents tests écrits pour l'application.

Pour sa mise en place, il faut créer un simple fichier nommé « *.travis.yml* » à la racine du projet. Ce fichier doit contenir les éléments nécessaires à **Travis** pour mettre en place l'environnement et exécuter les tests. Un fichier de settings **Django** minimaliste est également créé concernant la configuration de la base de données. Voici à quoi cela ressemble :

.travis.yml

```
language: python
python:
      - '3.7'

before_script:
      - pip install -r requirements.txt

branches:
    only:
        - dev

env:
      - DJANGO_SETTINGS_MODULE=oc_pizza_app.settings.travis

services:
      - postgresql

script:
      - python manage.py test
```

settings.travis.py

```
from . import *

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