P10 – Déployez votre application sur un serveur comme un pro !

Liens :
Application : http://purbeurre.jm-hayons74.fr
Github : https://github.com/jmlm74/P10-Deployez
Trello : https://trello.com/b/BA49JFvW/p10-d%C3%A9ployez

Objectifs et contraintes

Le principal objectif est de déployer l'application "PurBeurre" du P8 sur un serveur type de production hébergé. Ce serveur étant un serveur de production, les nouvelles versions doivent avoir été testées et validées avant d'être mises en production. Il faut donc utiliser un outil d'Intégration Continue pour cela. De plus, les éventuelles erreurs ou bugs doivent être automatiquement remontés et le serveur doit être monitoré afin de servir au mieux les clients. Enfin, il s'agit d'un serveur directement accessible depuis Internet, un niveau de sécurité minimum est nécessaire.

Principes et mise en œuvre

Le serveur proprement dit

Pour l'hébergeur, j'ai opté pour Digital Ocean. Outre les tarifs attractifs, Digital Ocean propose une gamme complète de serveurs VPS (RAM – CPU – DISK) qui bénéficie d'un monitoring complet avec seuils d'alerte et alertes par mail mais aussi d'un Firewall très facilement paramétrable. Pour ce qui est de l'OS j'ai choisi de mettre une Debian 10 (directement proposé par Digital Ocean). J'ai entre-autres rajouté à l'OS les modules IPTABLES et FAIL2BAN afin de parer les attaques SSH dites « brut force » ainsi que certaines attaques sur le HTTP/HTTPS (recherche pages admin, overflows…). Les mises à jour du serveur sont gérées par un module supplémentaire : UNATTENDED-UPGRADES. Ce module met automatique à jour l'OS et envoi un mail en cas de nécessité de redémarrage → Aucun redémarrage intempestif n'est effectué.

Le serveur « applicatif »

Par serveur applicatif, j'entends ici la partie applicative du serveur (Application et base de données).
Le serveur web est un serveur APACHE 2.4 avec les modules WSGI pour Django et SSL pour la sécurité des échanges clients/serveur (cryptage des données – avec une clé auto-signée pour le cas présent). Le serveur apache permet aussi de gérer les fichiers statiques.
Le serveur de base de données est un serveur PostgreSQL sans particularité quant à sa configuration.

J'ai choisi de mettre ce serveur applicatif sous DOCKER pour plusieurs raisons : Déploiement et mises à jour plus faciles, stabilité des conteneurs, déplacement d'un serveur à l'autre très rapide, démarrage rapide…J'ai donc utilisé 2 conteneurs qui sont gérés via Docker-Compose. Docker-Compose permet d'ordonner le démarrage des conteneurs (base en 1 et applcation en 2). Une chose avec Docker qu'il ne faut jamais perdre de vue est que les données ne sont JAMAIS conservées entre 2 démarrages. Docker permet de « monter » des fichiers ou répertoires depuis le serveur ce qui permet donc de conserver les données directement sur le serveur et non plus dans le conteneur. Elles ne sont donc plus réinitialisées à chaque démarrage du conteneur. Il y a donc 2 conteneurs :
- 1 conteneur Postgres directement fourni par Docker-Hub venant de Postgres.org directement. Les data sont stockées sur le serveur ainsi que le fichier de configuration et les logs.
- 1 conteneur Application contenant le serveur WEB et l'application proprement dite. Seuls les fichiers de configuration, les

logs et les certificats SSL sont conservés sur le serveur.
J'utilise un 3e conteneur qui est en fait un serveur WEB minimaliste (HTTP et HTTPS) qui va afficher une page d'attente pendant la mise à jour. Ce serveur peut aussi être démarré manuellement s'il devait y avoir une maintenance plus longue. Ces conteneurs sont hébergés sur le site https://hub.docker.com/. Un lien entre le conteneur applicatif (purbeurre) et le repo github a été paramétré. Cela permet une régénération automatique du conteneur sur le site à chaque nouveau push effectué sur la branche master du projet Github. Ensuite, un crontab qui se lance tous les mardi matin à 04:00 va récupérer le nouveau conteneur et le lancer, ou il est aussi possible de lancer la commande en étant connecté sous l'utilisateur root. A noter que la procédure de migration effectue le collectstatic et les migrations Django avant de rendre la main. Cette procédure dure moins de 5 minutes et ne demande aucune intervention.

Le Continuous Integration (CI)

Pour l'intégration continue, le choix s'est porté sur TRAVIS-CI. Il a donc fallu créer et paramétrer un compte sur le site de TRAVIS et lié un nouveau repository au repository existant sur Github. J'ai opté, pour tester en automatique, de créer une branche Github (« CI ») et de paramétrer TRAVIS pour qu'il lance les tests à chaque push sur cette branche. Les tests ont tout de suite bien fonctionné sauf l'utilisation du module Chrome avec Selenium alors que le module Gecko n'a quant à lui posé aucun problème. Les tests étant bons, il suffisait alors de « merger » CI sur master puis de « pusher » master pour que la régénération du conteneur Docker se fasse automatiquement. La nouvelle version est alors prête à être récupérée.

Le monitoring

2 outils ont été utilisés pour le monitoring :

Monitoring applicatif :
Le monitoring applicatif est en fait une remontée automatique des bugs et incidents (erreurs) de l'applicatif. Cette remontée a été faite grâce à l'outil Sentry. L'utilisation de cet outil nécessite outre le fait de se créer un compte et un paramétrage sur leur site de modifier le settings.py mais aussi le code source pour pouvoir remonter certaines erreurs. La seule modification du settings.py de production permet toutefois de récupérer un bon nombre d'erreurs de façon automatique. A noter que le fait de mettre Sentry dans le settings.py de développement a faussé les tests. La ligne `self.assertEquals(resolve(path).func, index)` qui valide que le path de urls.py renvoi bien sur la fonction du views.py retournait systématiquement une erreur. J'ai donc mis Sentry que sur le settings.py de production. D'un autre côté, le fait de remonter sur Sentry les erreurs lors de la phase de développement n'est certainement pas la chose à faire !

Monitoring serveur :
Pour le monitoring côté serveur, j'ai utilisé le monitoring proposé en standard par Digital Ocean. En effet, celui-ci propose un monitoring assez complet permettant de surveiller l'activité CPU, le load average, l'espace disque, le réseau… Par contre, il ne monitore pas les services. Le fait que ce monitoring soit de base et simple mais suffisant m'a conduit à utiliser un simple script bash lancé toutes les 10mn par crontab pour monitorer les services (SSH, HTTP/HTTPS, POSTGRES). Cela évite la mise en place d'un ou plusieurs démons supplémentaires (type Newrelic, Nagios ou autre) pour seulement 4 services.

Divers

Le nom de domaine utilisé est un nom de domaine que je possède depuis plusieurs années maintenant. Il est hébergé par OVH.
Le certificat SSL utilisé est un certificat auto-signé. Il faut donc l'accepter dans le navigateur. Cela permet d'utiliser le site en HTTPS.
L'initialisation de la base de données se fait à travers une commande doker exec et est une commande manage.py. Elle est exécutée tous les lundis matin après la mise à jour automatique de l'application (cf crontab).