Curs 2023 - 2024

Desplegament d'una aplicació Flask

Guillermo Vidal Frasquet

Desplegament d'Aplicacions Web

Pràctica

IES Jaume II El Just
Tavernes de la Valldigna

Continguts

1 Desplegament d'una aplicació Flask (Python)

1.1 Prerequisits

Servidor Debian amb els següents paquets instal·lats:

- Nginx
- Gunicorn
- Pipenv

1.2 Introducció

1.2.1 Què és un framework?

Actualment en el desenvolupament modern d'aplicacions web s'utilitzen diferents **Frameworks** que són eines que ens donen un esquema de treball i una sèrie d'utilitats i funcions que ens facilita i ens abstrau de la construcció de pàgines web dinàmiques.

En general els **Frameworks** estan associat a llenguatges de programació (**Ruby on Rails** (Ruby), **Symphony** (PHP)), en el món de Python el més conegut és **Django** però **Flask** és una opció que potser no té una corba d'aprenentatge tan elevada però ens possibilita la creació d'aplicacions web igual de complexes de les quals es poden crear en Django.

1.2.2 Flask

En l'actualitat existeixen moltes opcions per a crear pàgines web i molts llenguatges (PHP, JAVA), i en este cas **Flask** ens permet crear d'una manera molt senzilla aplicacions web amb Python.

Flask és un "micro" Framework escrit en Python i concebut per a facilitar el desenvolupament d'Aplicacions Web sota el patró **MVC**.

La paraula "micro" no vol dir que siga un projecte xicotet o que ens permeta fer pàgines web xicotetes sinó que en instal·lar **Flask** tenim les eines necessàries per a crear una aplicació web funcional però si es necessita en algun moment una nova funcionalitat hi ha un conjunt molt gran extensions (plugins) que es poden instal·lar amb Flask que van dotant-lo de funcionalitat.

De principi en la instal·lació no es tenen totes les funcionalitats que es poden necessitar però d'una manera molt senzilla es poden estendre el projecte amb noves funcionalitats per mitjà de `plugins`.

El patró **MVC** és una manera de treballar que permet diferenciar i separar el que és el model de dades (les dades que tindrà l'App que normalment estan guardades en BBDD), la vista (pàgina HTML) i el controlador (on es gestiona les peticions de l'app web).

1.2.3 Gunicorn

Quan s'implementa una aplicació web basada en `Python`, normalment es tenen estes tres peces:

- Servidor web (`Nginx`, `Apache`)
- Servidor d'aplicacions WSGI (`Gunicorn`, `uWSGI`, `mod_wsgi`, `Waitress`)
- Aplicació web (`Django`, `Flask`, `Pyramid`, `FastAPI`)

Els servidors web processen i distribueixen les sol·licituds dels navegadors i altres clients i envien respostes a estos.

WSGI (Web Server Gateway Interface) proporciona un conjunt de regles per a estandarditzar el comportament i la comunicació entre servidors web i aplicacions web. Mitjançant l'ús de servidors i aplicacions web compatibles amb WSGI, els desenvolupadors poden concentrar el seu temps i energia en el desenvolupament d'aplicacions web en lloc d'administrar la comunicació entre l'aplicació i el servidor web.

Finalment, `Gunicorn`, que és l'abreviatura de **Green Unicorn**, és un servidor d'aplicacions WSGI que es troba entre el servidor web i la seua aplicació web, gestionant la comunicació entre els dos. Accepta sol·licituds del servidor i les tradueix (a través de WSGI) en alguna cosa que l'aplicació web pot entendre abans de passar-la a l'aplicació web real. Envia respostes des de l'aplicació web al servidor. També s'encarrega d'executar diverses instàncies de l'aplicació web, reiniciant-les segons siga necessari i distribuint sol·licituds a instàncies saludables.

1.2.4 Gestor de paquets `pip`

pip és el comando per a instal·lar paquets de `Python` integrats en les fonts des de la versió 3.4.

Este comando automatitza la connexió al lloc https://pypi.org/, la descàrrega, la instal·lació i fins i tot la compilació del mòdul sol·licitat.

A més, s'ocupa de les dependències de cada paquet.

1.2.5 Entorns virtuals en `Python`

Un entorn virtual és una manera de tindre múltiples instàncies paral·leles de l'intèrpret de `Python`, cadascuna amb diferents conjunts de paquets i diferents configuracions. Cada entorn virtual conté una còpia independent de l'intèrpret de `Python`, incloent-hi còpies de les seues utilitats de suport.

Els paquets instal·lats en cada entorn virtual només es veuen en eixe entorn virtual i en cap altre. Fins i tot els paquets grans i complexos amb binaris dependents de la plataforma poden ser acorralats entre si en entorns virtuals.

D'esta manera, tindrem entorns independents entre si, semblança a com ocorria amb els directoris dels projectes de `Node.js`. D'aquesta manera, els entorns virtuals de `Python` ens permeten instal·lar un paquet de `Python` en una ubicació aïllada en lloc d'instal·lar-ho de manera global.

1.2.6 Pipenv

`Pipenv` és una eina que apunta a portar tot el millor del món d'empaquetat (`bundler`, `composer`, `npm`, `cargo`, `yarn`, etc.) al món de `Python`.

Automàticament crea i maneja un entorn virtual per als teus projectes, també permet agregar/eliminar paquets des del teu `Pipfile` així com com instal·lar/desinstal·lar paquets. També genera el més important, l'arxiu `Pipfile.lock`, que és usat per a produir determinat `build`.

1.3 Procediment complet per al desplegament

1. Instal·lem el gestor de paquets de Python `pip`:

   ```
   sudo apt-get update
   sudo apt-get install python3-pip
   ```

2. Instalamos el paquete `pipenv` para gestionar los entornos virtuales:

   ```
   pip3 install pipenv --break-system-packages
   ```

3. I comprovem que està instal·lat correctament mostrant la seua versió:

   ```
   PATH=$PATH:/home/guillermo/.local/bin
   pipenv --version
   ```

4. Creem el directori en el qual emmagatzemarem el nostre projecte:

   ```
   sudo mkdir /var/www/nom_meua_aplicacio
   ```

5. En crear-ho amb `sudo`, els permisos pertanyen a `root`:

   ```
   guillermo@debian-daw:~$ sudo mkdir /var/www/aplicacio_flask
   guillermo@debian-daw:~$ ls /var/www/aplicacio_flask/ -la
   total 8
   drwxr-xr-x 2 root root 4096 nov 17 16:46 .
   drwxr-xr-x 6 root root 4096 nov 17 16:46 ..
   guillermo@debian-daw:~$
   ```

6. Cal canviar-ho perquè el propietari siga el nostre usuari (`guillermo` en el meu cas) i pertanya al grup `www-data`, l'usuari usat per defecte pel servidor web per a córrer:

   ```
   sudo chown -R $USER:www-data /var/www/nom_meua_aplicacio
   ```

7. Establim els permisos adequats a este directori, perquè puga ser llegit per tot el món:

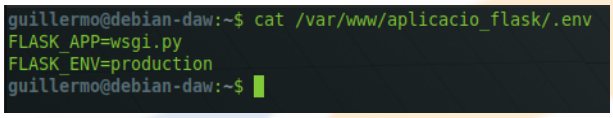
   ```
   chmod -R 775 /var/www/nom_meua_aplicacio
   ```

És **indispensable** assignar estos permisos, d'una altra forma obtindríem un error en accedir a l'aplicació quan posem en marxa **Nginx**.

8. Dins del directori de la nostra aplicació, creem un arxiu ocult `.env` que contindrà les variables d'entorn necessàries:

   ```
   touch .env
   ```

9. Editem l'arxiu i afegim les variables, indicant quin és l'arxiu `.py` de l'aplicació i l'entorn, que en el nostre cas serà `production`:

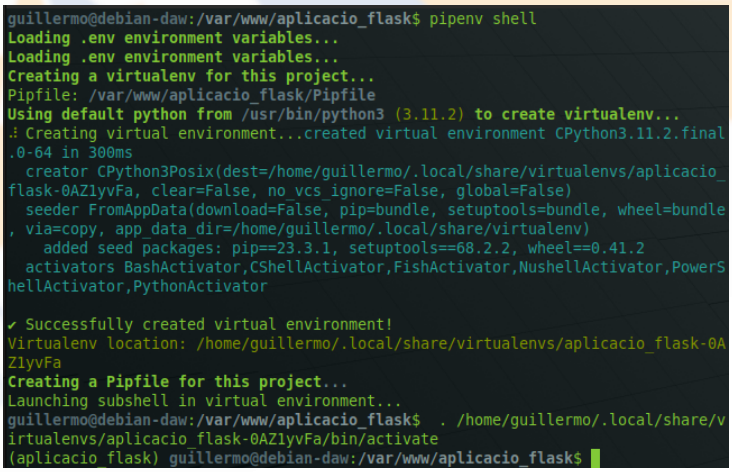


> En el món laboral real, se suposa que l'aplicació prèviament ha passat pels entorns de `dev`, `test` i `preprod` per al desenvolupament i prova d'esta, abans de passar-la a producció.

10. Iniciem ara el nostre entorn virtual. `pipenv` carregarà les variables d'entorn des del fitxer `.env` de manera automàtica:

    ```
    pipenv shell
    ```

    Veurem que se'ns inicia l'entorn virtual, cosa que comprovem perquè apareix el seu nom a l'inici del prompt del `shell`:



11. Usem `pipenv` per a instal·lar les dependències necessàries per al nostre projecte:

    ```
    pipenv install flask gunicorn
    ```

12. Anem ara a crear l'aplicació Flask més simple possible, a mode de **PoC** (proof of concept o prova de concepte). L'arxiu que contindrà l'aplicació pròpiament dita serà `application.py` i `wsgi.py` s'encarregarà únicament d'iniciar-la i deixar-la corrent:

    ```
    touch application.py wsgi.py
    ```

    I després de crear els arxius, els editem per a deixar-los així:

    ```
    (aplicacio_flask) guillermo@debian-daw:/var/www/aplicacio_flask$ cat application.py
    from flask import Flask

    app = Flask(__name__)

    @app.route('/')
    def index():
            '''Index page route'''

            return '<h1>Aplicació desplegada</h1>
    (aplicacio_flask) guillermo@debian-daw:/var/www/aplicacio_flask$
    ```

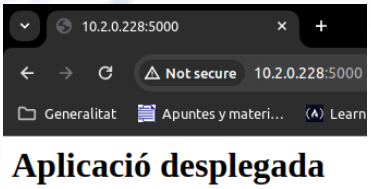
    ```
    (aplicacio_flask) guillermo@debian-daw:/var/www/aplicacio_flask$ cat wsgi.py
    from application import app

    if __name__ == '__main__':
            app.run(debug=False)
    (aplicacio_flask) guillermo@debian-daw:/var/www/aplicacio_flask$
    ```

13. Correguem ara la nostra aplicació a mode de comprovació amb el servidor web integrat de Flask. Si especifiquem la direcció `0.0.0.0` el que li estem dient al servidor és que escolte en totes les seues interfícies, si les tinguera:

    ```
    (aplicacio_flask) guillermo@debian-daw:/var/www/aplicacio_flask$ flask run --host '0.0.0.0'
     * Tip: There are .env or .flaskenv files present. Do "pip install python-dotenv" to use them.
     * Serving Flask app 'wsgi.py'
     * Debug mode: off
    WARNING: This is a development server. Do not use it in a production deployment. Use a production W
    SGI server instead.
     * Running on all addresses (0.0.0.0)
     * Running on http://127.0.0.1:5000
     * Running on http://10.2.0.228:5000
    Press CTRL+C to quit
    ```

14. Ara podrem accedir a l'aplicació des del nostre ordinador, la nostra màquina amfitrió, introduint en un navegador web: `http://IP-maq-virtual:5000`, després de la comprovació, parem el servidor amb CTRL+C:

15. Comprovem ara que `Gunicorn` funciona correctament també. Si us ha funcionat el servidor de desenvolupament de `Flask`, podeu usar el següent comando per a provar que l'aplicació

funciona correctament usant `Gunicorn`, accedint amb el vostre navegador de la mateixa forma que en el pas anterior:

```
gunicorn --workers 4 --bind 0.0.0.0:5000 wsgi:app
```

On:

- `--workers N` estableix el número de *workers* o fils que volem utilitzar, com ocorria amb `Node Express`. Dependrà del número de `cores` que li hàgem donat a la CPU de la nostra màquina virtual.
- `--bind 0.0.0.0:5000` fa que el servidor escolte peticions per totes les seues interfícies de xarxa i en el port 5000.
- `wsgi:app` és el nom de l'arxiu amb extensió `.py` i app és la instància de l'aplicació Flask dins de l'arxiu.

16. Encara dins del nostre entorn virtual, hem de prendre nota de qual és el `path` o ruta des de la qual s'executa `gunicorn` per a poder configurar més endavant un servei del sistema. Podem esbrinar-lo així:

```
(aplicacio_flask) guillermo@debian-daw:/var/www/aplicacio_flask$ which gunicorn
/home/guillermo/.local/share/virtualenvs/aplicacio_flask-0AZ1yvFa/bin/gunicorn
(aplicacio_flask) guillermo@debian-daw:/var/www/aplicacio_flask$
```

I després d'això hem d'eixir del nostre entorn virtual amb el senzill comando `deactivate`.

17. Com que ja hem de tindre instal·lat Nginx en el nostre sistema, l'iniciem i comprovem que el seu estat siga actiu:

```
sudo systemctl start nginx
```

```
sudo systemctl status nginx
```

18. Ja fora del nostre entorn virtual, crearem un arxiu perquè `systemd` córrega `gunicorn` com un servei més del sistema:

```
guillermo@debian-daw:~$ cat /etc/systemd/system/flask_app.service
[Unit]
Description=flask_app.service - Una aplicació flask d'exemple amb Gunicorn
After=network.target

[Service]
User=guillermo
Group=www-data
Environment="PATH=/home/guillermo/.local/share/virtualenvs/aplicacio_flask-0AZ1yvFa/bin"
WorkingDirectory=/var/www/aplicacio_flask/
ExecStart=/home/guillermo/.local/share/virtualenvs/aplicacio_flask-0AZ1yvFa/bin/gunicorn --workers 3
--bind unix://var/www/aplicacio_flask/aplicacio_flask.sock wsgi:app

[Install]
WantedBy=multi-user.target
guillermo@debian-daw:~$
```

On:

- `User`: Estableix l'usuari que té permisos sobre el directori del projecte (el ficareu en el pas 5).
- `Group`: Estableix el grup que té permisos sobre el directori del projecte (el ficareu en el pas 5).
- `Environment`: Estableix el directori `bin` (on es guarden els binaris executables) dins de l'entorn virtual (el que vegereu en el pas 16).
- `WorkingDirectory`: Estableix el directori base on resideix el nostre projecte.
- `ExecStart`: Estableix el `path` on es troba l'executable de `gunicorn` dins de l'entorn virtual, així com les opcions i comandos amb els quals s'iniciarà.

> Heu de canviar els valors perquè coincidisquen amb els del vostre cas particular.

19. Ara, com cada vegada que es crea un servei nou de `systemd`, s'habilita i s'inicia:

    ```
    systemctl enable nom_meu_servei
    ```

    ```
    systemctl start nom_meu_servei
    ```

    Recordeu que el nom del servei és el nom de l'arxiu que heu creat en el pas anterior.

    Passem ara a configurar **Nginx**, que és una cosa que ja hauríem de tindre dominat de capítols anteriors.

20. Creem un arxiu amb el nom de la nostra aplicació i dins establirem la configuració per a eixe lloc web. L'arxiu, com recordeu, ha d'estar en `/etc/nginx/sites-available/nom_aplicacio` i després d'això ho editem perquè quede:

```
server {
    listen 80;
    server_name mi_aplicacion www.mi_aplicacion;
    # Nom del domini, ja veurem més endavant com el DNS resoldrà
       este nom per a accedir a la nostra aplicació.

    access_log /var/log/nginx/mi_aplicacion.access.log; # On
       estaran situats els logs d'accés i d'errors.
    error_log /var/log/nginx/mi_aplicacion.error.log;

    location / {
        include proxy_params;
        proxy_pass http://unix:/var/www/nombre_aplicacion/
           nombre_aplicacion.sock;
        # Bloc on se li indica a Nginx que faça de proxy invers
           cap al socket creat en la nostra pròpia màquina per
           gunicorn per a accedir a la nostra aplicació Flask.
    }
}
```

21. Recordem que ara hem de crear un link simbòlic de l'arxiu de llocs webs disponibles al de llocs web actius:

    ```
    sudo ln -s /etc/nginx/sites-available/nom_aplicacio /etc/nginx/sites-enabled/
    ```

    I ens assegurem que s'ha creat aquest link simbòlic:

    ```
    ls -l /etc/nginx/sites-enabled/ | grep nom_aplicacio
    ```

22. Ens assegurem que la configuració de Nginx no conté errors, reiniciem Nginx i comprovem que està actiu:

    ```
    nginx -t
    ```

    ```
    sudo systemctl restart nginx
    ```

    ```
    sudo systemctl status nginx
    ```

23. Ja no podrem accedir per IP a la nostra aplicació ja que ara està sent servida per `Gunicorn` i Nginx, necessitem accedir pel seu `server_name`. Com que encara no hem tractat amb el DNS, editarem l'arxiu `/etc/hosts` de la nostra màquina amfitriona perquè associe la IP de la màquina virtual, al nostre `server_name`.

    I haurem d'afegir-li la línia:

    ```
    192.168.X.X myproject www.myproject
    ```

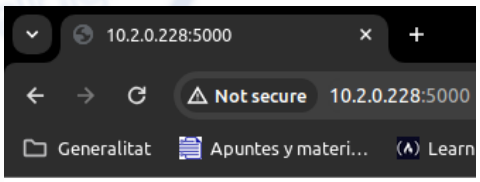
24. L'últim pas és comprovar que tot el desplegament s'ha realitzat de manera correcta i està funcionant, per a això accedim des de la nostra màquina amfitrió a:

    ```
    http://nom_aplicacio
    ```

    O:

    ```
    http://www.nom_aplicacio
    ```

    I hauria de mostrar-vos la mateixa pàgina que en el pas 14:



Aplicació desplegada