Después de instalar el gestor de paquetes de Python y piepv comprobamos que todo está correctamente instalado:

```
vagrant@display:~/.local/bin$ pipenv --version
pipenv, version 2024.4.0
```

Creamos el directorio en el que almacenaremos nuestro proyecto y le cambiamos los permisos permisos:

```
vagrant@display:~$ sudo mkdir -p /var/www/app
vagrant@display:~$ sudo chown -R $USER:www-data /var/www/app
vagrant@display:~$ chmod -R 775 /var/www/app
vagrant@display:~$ cd /var/www/app/
vagrant@display:/var/www/app$ cd ..
vagrant@display:/var/www$ ls -l
total 8
drwxrwxr-x 2 vagrant www-data 4096 Jan 22 12:15 app
drwxr-xr-x 2 root    root     4096 Jan 22 08:02 html
```

Dentro del directorio de nuestra aplicación, creamos un archivo oculto `.env` que contendrá las variables de entorno necesarias.

```
vagrant@display:/var/www/app$ sudo nano .env
vagrant@display:/var/www/app$ ls
vagrant@display:/var/www/app$ ls -l
total 0
vagrant@display:/var/www/app$ ls -la
total 12
drwxrwxr-x 2 vagrant www-data 4096 Jan 22 12:17 .
drwxr-xr-x 4 root    root     4096 Jan 22 12:15 ..
-rw-r--r-- 1 root    root       39 Jan 22 12:17 .env
vagrant@display:/var/www/app$ sudo cat .env
FLASK_APP=wsgi.py
FLASK_ENV=production
```

Iniciamos ahora nuestro entorno virtual. *Pipenv* cargará las variables de entorno desde el fichero `.env` de forma automática:

```
vagrant@display:/var/www/app$ pipenv shell
Loading .env environment variables...
Creating a virtualenv for this project
Pipfile: /var/www/app/Pipfile
Using default python from /usr/bin/python3.9.2 to create virtualenv...
⠙ Creating virtual environment...created virtual environment CPython3.9.2.final.0-64 in 951ms
  creator CPython3Posix(dest=/home/vagrant/.local/share/virtualenvs/app-1lvW3LzD, clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=/home/vagrant/.local/share/virtualenv)
    added seed packages: pip==24.3.1, setuptools==75.8.0, wheel==0.45.1
  activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator

✓ Successfully created virtual environment!
Virtualenv location: /home/vagrant/.local/share/virtualenvs/app-1lvW3LzD
Creating a Pipfile for this project...
Launching subshell in virtual environment...
vagrant@display:/var/www/app$  source /home/vagrant/.local/share/virtualenvs/app-1lvW3LzD/bin/activate
(app) vagrant@display:/var/www/app$
```

Usamos `pipenv install flask gunicorn` para instalar las dependencias necesarias para nuestro proyecto:

```
(app) vagrant@display:/var/www/app$ pipenv install flask gunicorn
Loading .env environment variables...
Pipfile.lock not found, creating...
Locking [packages] dependencies...
Locking [dev-packages] dependencies...
Updated Pipfile.lock (a36a5392bb1e8bbc06bfaa0761e52593cf2d83b486696bf54667ba8da616c839)!
Installing flask...
✓ Installation Succeeded
Installing gunicorn...
✓ Installation Succeeded
Installing dependencies from Pipfile.lock (16c839)...
All dependencies are now up-to-date!
Upgrading flask, gunicorn in  dependencies.
Building requirements...
Resolving dependencies...
✓ Success!
Building requirements...
Resolving dependencies...
✓ Success!
Installing dependencies from Pipfile.lock (6f5432)...
All dependencies are now up-to-date!
Installing dependencies from Pipfile.lock (6f5432)...
(app) vagrant@display:/var/www/app$
```

Creamos una aplicación básica. El archivo que contendrá la aplicación propiamente dicha será `application.py` y `wsgi.py` se encargará únicamente de iniciarla y dejarla corriendo:
Tras crear todo con: touch application.py wsgi.py
Editamos los ficheros y ponemos:
`application.py`

Flask-en-Python > app > application.py

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    '''Index page route'''
    return '<h1>App desplegada</h1>'

```

`wsgi.py`

Flask-en-Python > app > wsgi.py

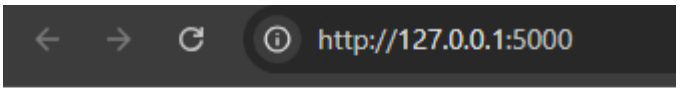
```python
from application import app

if __name__ == '__main__':
    app.run(debug=False)

```

Corramos ahora nuestra aplicación a modo de comprobación con el servidor web integrado de Flask. Si especificamos la dirección `0.0.0.0` lo que le estamos diciendo al servidor es que escuche en todas sus interfaces, si las tuviera:

```
(app) vagrant@display:/var/www/app$ flask run --host '0.0.0.0'
 * Tip: There are .env files present. 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 WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.0.2.15:5000
Press CTRL+C to quit
```



App desplegada

Tras la comprobación, paramos el servidor con `CTRL+C`. Comprobemos ahora que Gunicorn funciona correctamente también. Si os ha funcionado el servidor de desarrollo de Flask, podéis usar el comando `gunicorn --workers 4 --bind 0.0.0.0:5000 wsgi:app` para probar que la alicación funciona correctamente usando Gunicorn, accediendo con vuestro navegador de la misma forma que en el paso anterior:

```
(app) vagrant@display:/var/www/app$ gunicorn --workers 4 --bind 0.0.0.0:5000 wsgi:app
[2025-01-27 08:34:20 +0000] [12678] [INFO] Starting gunicorn 23.0.0
[2025-01-27 08:34:20 +0000] [12678] [INFO] Listening at: http://0.0.0.0:5000 (12678)
[2025-01-27 08:34:20 +0000] [12678] [INFO] Using worker: sync
[2025-01-27 08:34:20 +0000] [12765] [INFO] Booting worker with pid: 12765
[2025-01-27 08:34:20 +0000] [12786] [INFO] Booting worker with pid: 12786
[2025-01-27 08:34:20 +0000] [12790] [INFO] Booting worker with pid: 12790
[2025-01-27 08:34:20 +0000] [12794] [INFO] Booting worker with pid: 12794
```

Ya fuera de nuestro entorno virtual, crearemos un archivo para que `systemd` corra Gunicorn como un servicio del sistema más:

Flask-en-Python > flask_app.service

```
[Unit]
Description=flask app service - App con flask y Gunicorn
After=network.target
[Service]
User=vagrant
Group=www-data
Environment="PATH=/home/vagrant/.local/share/virtualenvs/app-1lvW3LzD/bin"
WorkingDirectory=/var/www/app
ExecStart=/home/vagrant/.local/share/virtualenvs/app-1lvW3LzD/bin/gunicorn --workers 3 --bind unix:/var/www/app/app.sock application:app

[Install]
WantedBy=multi-user.target
```

nformaremos a `systemd` que hay un nuevo servicio:

sudo systemctl daemon-reload

Ahora, como cada vez que se crea un servicio nuevo de `systemd`, se habilita y se inicia:

systemctl enable flask_app

systemctl start flask_app

Creamos un archivo con el nombre de nuestra aplicación y dentro estableceremos la configuración para ese sitio web. El archivo, como recordáis, debe estar en `/etc/nginx/sites-available/app.conf` *y tras ello lo editamos para que quede:*

Flask-en-Python > app.conf

```
server {
    listen 80;
    server_name app.izv www.app.izv;

    access_log /var/log/nginx/app.access.log;
    error_log /var/log/nginx/app.error.log;

    location / {
        include proxy_params;
        proxy_pass http://unix:/var/www/app/app.sock;
    }
}
```

Recordemos que ahora debemos crear un link simbólico del archivo de sitios webs disponibles al de sitios web activos y nos aseguramos de que se ha creado dicho link simbólico:

```
(app) vagrant@display:/etc/nginx/sites-available$ sudo ln -s /etc/nginx/sites-available/app.conf /etc/nginx/sites-enabled/
(app) vagrant@display:/etc/nginx/sites-available$ ls -l /etc/nginx/sites-enabled/ | grep app.conf
lrwxrwxrwx 1 root root 35 Jan 27 08:51 app.conf -> /etc/nginx/sites-available/app.conf
(app) vagrant@display:/etc/nginx/sites-available$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
(app) vagrant@display:/etc/nginx/sites-available$
```

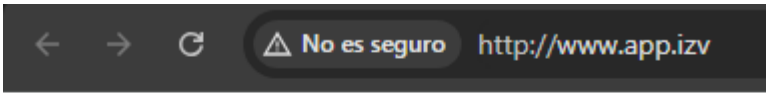
Ya no podremos acceder por IP a nuestra aplicación ya que ahora está siendo servida por Gunicorn y Nginx, necesitamos acceder por su `server_name`*. Puesto que aún no hemos tratado con el DNS, vamos a editar el archivo* `/etc/hosts` *de nuestra máquina anfitriona para que asocie la IP de la máquina virtual, a nuestro* `server_name`*.*

Este archivo, en Linux, está en `/etc/hosts` *y en Windows en* `C:\Windows\System32\drivers\etc\hosts`

Y deberemos añadirle la línea:

```
127.0.0.1   localhost
127.0.0.2   bullseye
::1     localhost ip6-localhost ip6-loopback
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters
192.168.50.20    app.izv www.app.izv
127.0.1.1 display display
```

El último paso es comprobar que todo el despliegue se ha realizado de forma correcta y está funcionando, para ello accedemos desde nuestra máquina anfitrión a http://app.izv/ o http://www.app.izv/



App desplegada

Tarea de ampliación

Tras hacer el git clone en /var/www saldrá esto:

```
(msdocs-python-flask-webapp-quickstart) vagrant@display:/var/www/msdocs-python-flask-webapp-quickstart$ ls
CHANGELOG.md  CONTRIBUTING.md  LICENSE.md  Pipfile  Pipfile.lock  README.md  app.py  requirements.txt  static  templates
(msdocs-python-flask-webapp-quickstart) vagrant@display:/var/www/msdocs-python-flask-webapp-quickstart$
```

Y un último detalle, Gunicorn debe iniciarse ahora así:

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

```
(msdocs-python-flask-webapp-quickstart) vagrant@display:/var/www/msdocs-python-flask-webapp-quickstart$ gunicorn --workers 4 --bind 0.0.0.0:5000 app:app
[2025-01-27 18:44:21 +0000] [3254] [INFO] Starting gunicorn 23.0.0
[2025-01-27 18:44:21 +0000] [3254] [INFO] Listening at: http://0.0.0.0:5000 (3254)
[2025-01-27 18:44:21 +0000] [3254] [INFO] Using worker: sync
[2025-01-27 18:44:21 +0000] [3255] [INFO] Booting worker with pid: 3255
[2025-01-27 18:44:21 +0000] [3256] [INFO] Booting worker with pid: 3256
[2025-01-27 18:44:21 +0000] [3257] [INFO] Booting worker with pid: 3257
[2025-01-27 18:44:21 +0000] [3259] [INFO] Booting worker with pid: 3259
Request for index page received
```

Y en el navegador a través de http://192.168.50.20:5000/ saldrá