

# .:: Desplegar Flask App en Heroku ::.

</ By Surflaweb – Freddy Alcarazo >

<https://www.youtube.com/channel/UCy6ttBvsjtroucQocj3P7Q>

<https://github.com/alcarazolabs/surflaweb-scripts>

[creatorpart@gmail.com](mailto:creatorpart@gmail.com)

3 de abril del 2021

## 1. Requerimientos

- ✓ Primero debemos de tener instalado “Python” obviamente, en mi caso vengo usando el programa Anaconda.
- ✓ Tener instalado Heroku CLI y registrados en Heroku.
- ✓ Una App creada con Flask. En otra publicación veremos cómo desplegar una App heroku que utiliza una base de datos MySQL.

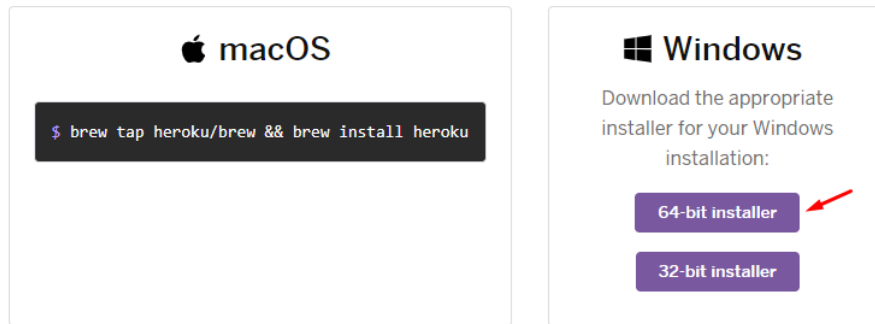
## 2. Instalando Heroku CLI.

Una vez registrados en Heroku procedemos a descargarnos la CLI de Heroku la cual nos permitirá subir nuestros apps al hosting a través de comandos heroku.

No dirigimos al sitio web:

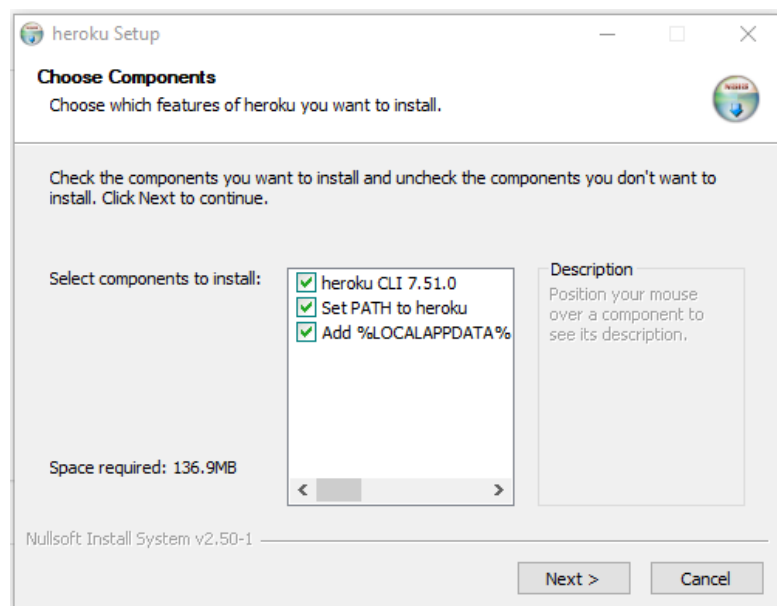
<https://devcenter.heroku.com/articles/heroku-cli>

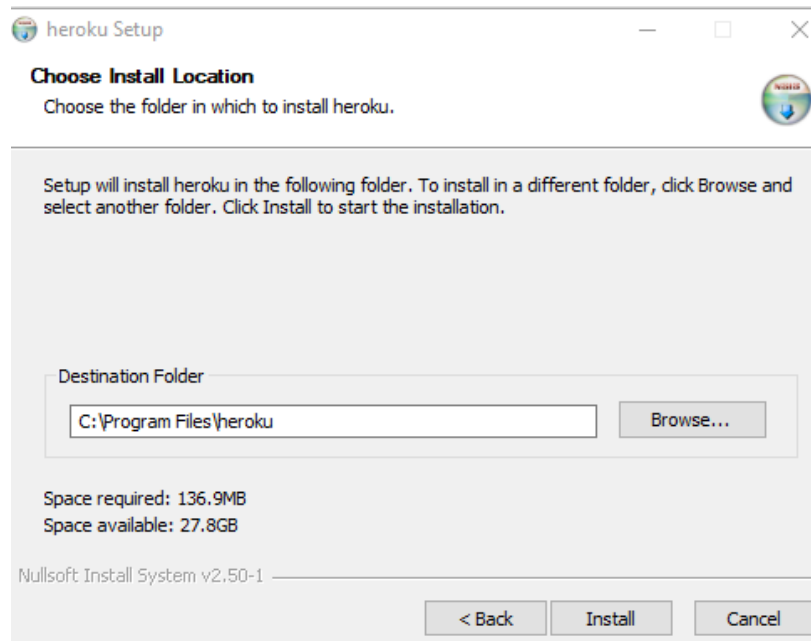
y descargamos la versión que se adecue a nuestro sistema operativo. En mi caso descargaré la versión para Windows.



La CLI de heroku solo pesa unos 22 Megabytes así que la descarga será rápida.

Una vez descargado le damos doble click:





Luego de haber terminado la instalación abrimos una terminal CMD en mi caso utilizo la anaconda prompt ya que ahí tengo corriendo Python y escribimos el siguiente comando:

```
heroku --version
```

```
(base) C:\Users\freddyalc>heroku --version
>> Warning: Our terms of service have changed: https://dashboar
heroku/7.51.0 win32-x64 node-v12.21.0
(base) C:\Users\freddyalc>
```

De esa manera conocemos la versión de Heroku que tenemos instalada.

**Ahora tenemos que loguearnos a través de Heroku CLI** para poder subir nuestros proyectos. Hay dos formas, vemos a ver la primera forma la cual yo voy a usar:

Escribimos el comando:

```
heroku login -i
```

Para poner nuestras credenciales en la consola:

```
(base) C:\Users\freddyalc>heroku login -i
heroku: Enter your login credentials
Email: creatorpart@gmail.com
Password: *****
Logged in as creatorpart@gmail.com
(base) C:\Users\freddyalc>
```

Y listo como puede ver ya estoy logueado.

La otra manera de loguearse es poner el siguiente comando:

```
heroku login
```

Esto abrirá una ventana del navegador y nos pedirá que pongamos nuestro email y password luego automáticamente nos logueara.

### **3. Creando proyecto a través de Heroku CLI**

Para crear nuestro proyecto en Heroku hay dos formas, una desde su página web y la otra que vamos a usar es desde la CLI de Heroku que hemos instalado.

## Creando un proyecto.

### Primera forma:

```
heroku create
```

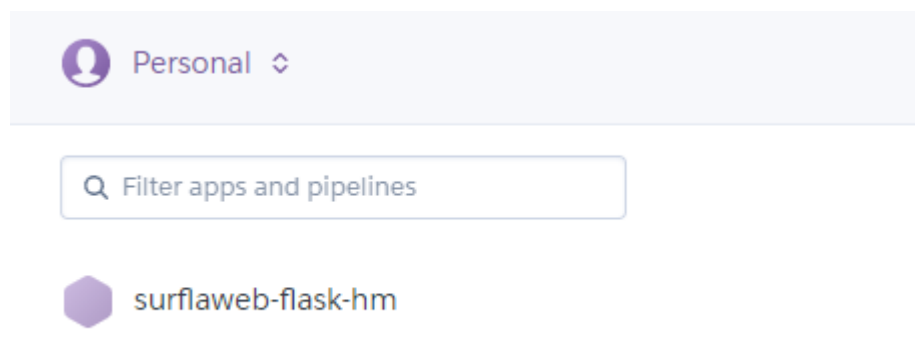
Al ejecutar el comando de arriba heroku nos creará un proyecto con cualquier nombre aleatorio. En nuestro caso creamos un proyecto con nuestro propio nombre, para eso ejecutamos:

```
heroku apps:create surflaweb-flask-hm
```

Nota que el nombre del proyecto aquí se llama **“surflaweb-flask-hm”** vamos a ejecutar eso:

```
(base) C:\Users\freddyalc>heroku apps:create surflaweb-flask-hm
Creating surflaweb-flask-hm... done
https://surflaweb-flask-hm.herokuapp.com/ | https://git.heroku.com/surflaweb-flask-hm.git
(base) C:\Users\freddyalc>
```

Y listo proyecto creado, verifico en la página web:



Ahora lo que vamos a hacer es renombrar el proyecto, le voy a quitar las dos últimas letras y el proyecto se llamará “**surflaweb-flask**”

Para eso ejecutamos el siguiente comando:

```
heroku apps:rename --app surflaweb-flask-hm surflaweb-flask
```

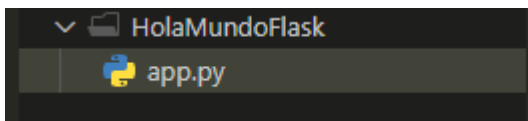
```
(base) C:\Users\freddyalc>heroku apps:rename --app surflaweb-flask-hm surflaweb-flask
Renaming surflaweb-flask-hm to surflaweb-flask... done
https://surflaweb-flask.herokuapp.com/ | https://git.heroku.com/surflaweb-flask.git
! Don't forget to update git remotes for all other local checkouts of the app.

(base) C:\Users\freddyalc>_
```

¡Y listo! Logramos cambiarle el nombre, este será el nombre del proyecto final.

#### 4. Proyecto Flask

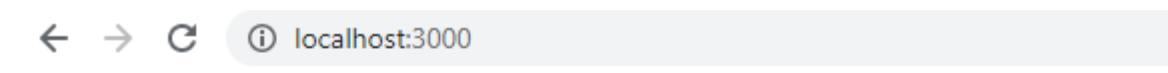
El proyecto creado con el micro-framework Flask solo consta de un archivo llamado app.py y este tiene todo el contenido del proyecto:



```
app.py x
HolaMundoFlask > app.py > ...
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def Index():
7
8      return "<h1>Hola mundo Flask en Heroku!!"
9
10 if __name__ == "__main__":
11     app.run(port=3000)
12
```

Como podemos ver solo estamos utilizando una librería. No estamos accediendo a bases de datos solo simplemente es un hola mundo. En otra publicación veremos cómo subir un proyecto a heroku que requiere por ejemplo de la librería flask\_mysqldb aquí tendremos que solicitar una base de datos MySQL a heroku pero para eso necesitamos de una tarjeta de crédito a la mano, lo bueno es que no cobra!

La siguiente imagen muestra este proyecto corriendo en localhost:3000



# Hola mundo Flask en Heroku!!

## 5. Crear virtualenv e instalar librerías

Virtualenv es una herramienta utilizada para crear un entorno Python aislado. Este entorno tiene sus propios directorios de instalación que no comparten bibliotecas con otros entornos virtualenv o las bibliotecas instaladas globalmente en el servidor.

<https://help.dreamhost.com/hc/es/articles/115000695551-Instalar-y-usar-virtualenv-con-Python->

Con esta herramienta se crea un entorno aislado dentro de nuestro proyecto y ahí instalaremos las librerías que el proyecto requiere.

Con ayuda de anaconda y debido a que tiene Python instalado, instalamos virtualenv:

```
pip install virtualenv
```

```
(base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>pip install virtualenv
Collecting virtualenv
  Downloading virtualenv-20.4.3-py2.py3-none-any.whl (7.2 MB)
    |-----| 7.2 MB 91 kB/s
Collecting distlib<1,>=0.3.1
  Downloading distlib-0.3.1-py2.py3-none-any.whl (335 kB)
    |-----| 335 kB 75 kB/s
Requirement already satisfied: filelock<4,>=3.0.0 in c:\users\freddyalc\anaconda3\lib\site-packages (from virtualenv) (3.0.0)
Requirement already satisfied: six<2,>=1.9.0 in c:\users\freddyalc\anaconda3\lib\site-packages (from virtualenv) (1.15.0)
Collecting appdirs<2,>=1.4.3
  Downloading appdirs-1.4.4-py2.py3-none-any.whl (9.6 kB)
Installing collected packages: distlib, appdirs, virtualenv
Successfully installed appdirs-1.4.4 distlib-0.3.1 virtualenv-20.4.3

(base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>
```

Como vemos ya se instaló el “**virtualenv**” ahora tenemos que crear un ambiente virtual dentro del proyecto para eso debemos de acceder dentro de la carpeta del proyecto, a continuación ustedes podrán ver que estoy dentro de la carpeta del proyecto y solo existe el archivo app.py:

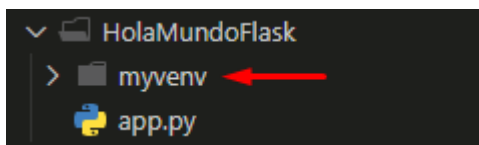
```
Directorio de C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask
03/04/2021  16:54    <DIR>          .
03/04/2021  16:54    <DIR>          ..
03/04/2021  16:59             190 app.py
                1 archivos             190 bytes
                2 dirs 29,302,697,984 bytes libres

(base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>
```

Ahora creamos un virtual env para el proyecto llamado “**myenv**”

```
python -m venv myenv
```

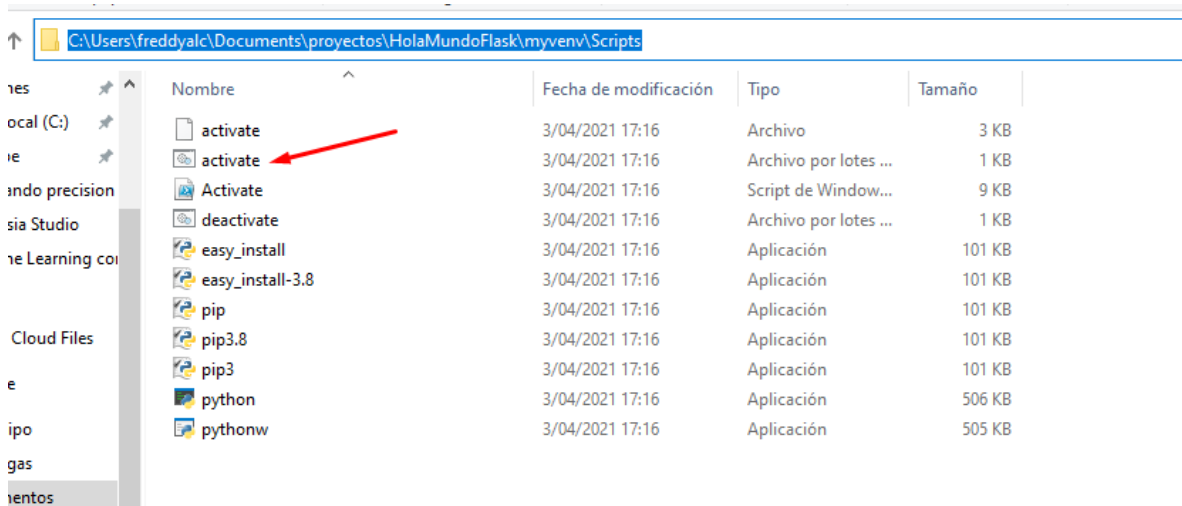
y listo ahora veremos dentro del proyecto que se creó la carpeta “myenv”



Ahora activaremos ese virtual env llamado “myenv” para instalarle las librerías que requiere el proyecto:



En Windows vamos a hacer lo siguiente, copiamos la ruta de la carpeta **“Scripts”** de nuestro virtual env creado y al final le agregamos el nombre del archivo batch active.



Lo ejecutamos desde la consola de la siguiente manera:

```
(base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask\myenv\Scripts\activate
(myenv) (base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>
```

Y listo ya estamos dentro del ambiente creado con “virtualenv” de Python.

Si eres usuario de Linux solo escribe:

```
source myenv/bin/activate
```

y eso activaría el ambiente.

Ahora ejecutamos **pip freeze** para ver si existen algunas librerías por defecto instaladas dentro de ese enviroment:

```
pip freeze
```

```
(myvenv) (base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>pip freeze  
(myvenv) (base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>_
```

Vemos que no existe nada.

Ahora ahí instalaremos las librerías que requiere el proyecto en este caso solo “Flask”:

```
pip install flask
```

```
(myvenv) (base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>pip install flask  
Collecting flask  
  Using cached https://files.pythonhosted.org/packages/f2/28/2a03252dfb9ebf377f40fba6a7841b47083260bf8bd8e73...  
Collecting click>=5.1 (from flask)  
  Using cached https://files.pythonhosted.org/packages/d2/3d/fa76db83bf75c4f8d338c2fd15c8d33fdd7ad23a9b5e57e...  
Collecting Werkzeug>=0.15 (from flask)  
  Using cached https://files.pythonhosted.org/packages/cc/94/5f7079a0e0bd6863ef8f1da638721e9da21e5bacee5975...  
Collecting itsdangerous>=0.24 (from flask)  
  Using cached https://files.pythonhosted.org/packages/76/ae/44b03b253d6fade317f32c24d100b3b35c2239807046a4d...  
Collecting Jinja2>=2.10.1 (from flask)  
  Using cached https://files.pythonhosted.org/packages/7e/c2/1eece8c95ddbc9b1aeb64f5783a9e07a286de42191b7204...  
Collecting MarkupSafe>=0.23 (from Jinja2>=2.10.1->flask)  
  Downloading https://files.pythonhosted.org/packages/4f/8b/da8a2ae5780d38271ac6e691756fc938cf4df8f225eb8aad...  
Installing collected packages: click, Werkzeug, itsdangerous, MarkupSafe, Jinja2, flask  
Successfully installed Jinja2-2.11.3 MarkupSafe-1.1.1 Werkzeug-1.0.1 click-7.1.2 flask-1.1.2 itsdangerous-1.1.2  
WARNING: You are using pip version 19.2.3, however version 21.0.1 is available.  
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

Ahora verificamos que librerías tenemos instaladas en el enviroment creado como “**pip freeze**”:

```
(myvenv) (base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>pip freeze  
click==7.1.2  
Flask==1.1.2  
itsdangerous==1.1.0  
Jinja2==2.11.3  
MarkupSafe==1.1.1  
Werkzeug==1.0.1  
  
(myvenv) (base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>
```

## Lo siguiente que vamos a instalar en una librería llamada “gunicorn”

Gunicorn es un servidor HTTP para sistemas Unix que cumple la especificación WSGI. Nos permite servir nuestra aplicación Flask con múltiples *workers* para incrementar el rendimiento de nuestra aplicación.

Es recomendable para entornos de producción no usar el servidor web integrado en Flask, que tiene como objetivo un entorno de desarrollo.

Esta se encargará de levantar un servidor web en producción y ejecutará el archivo `app.py`, luego lo definiremos creando un archivo `Procfile`.

```
pip install gunicorn
```

```
(myvenv) (base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>pip install gunicorn
Collecting gunicorn
  Downloading https://files.pythonhosted.org/packages/28/5b/0d1f0296485a6af03366604142ea8f19f6...
    | 378kB 467kB/s
Requirement already satisfied: setuptools>=3.0 in c:\users\freddyalc\documents\proyectos\holamundo\venv\lib\site-packages (from gunicorn)
Installing collected packages: gunicorn
  Running setup.py install for gunicorn ... done
Successfully installed gunicorn-20.1.0
WARNING: You are using pip version 19.2.3, however version 21.0.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
(myvenv) (base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>
```

Ahora otra vez verificamos que librerías tenemos instaladas en el enviroment creado como “**pip freeze**”:

```
(myvenv) (base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>pip freeze
click==7.1.2
Flask==1.1.2
gunicorn==20.1.0
itsdangerous==1.1.0
Jinja2==2.11.3
MarkupSafe==1.1.1
Werkzeug==1.0.1
```

Y listo como podemos ver ahí están todas nuestras librerías que el proyecto va a usar también vemos que esta Jinja2 que es un motor de plantillas usado en Flask.

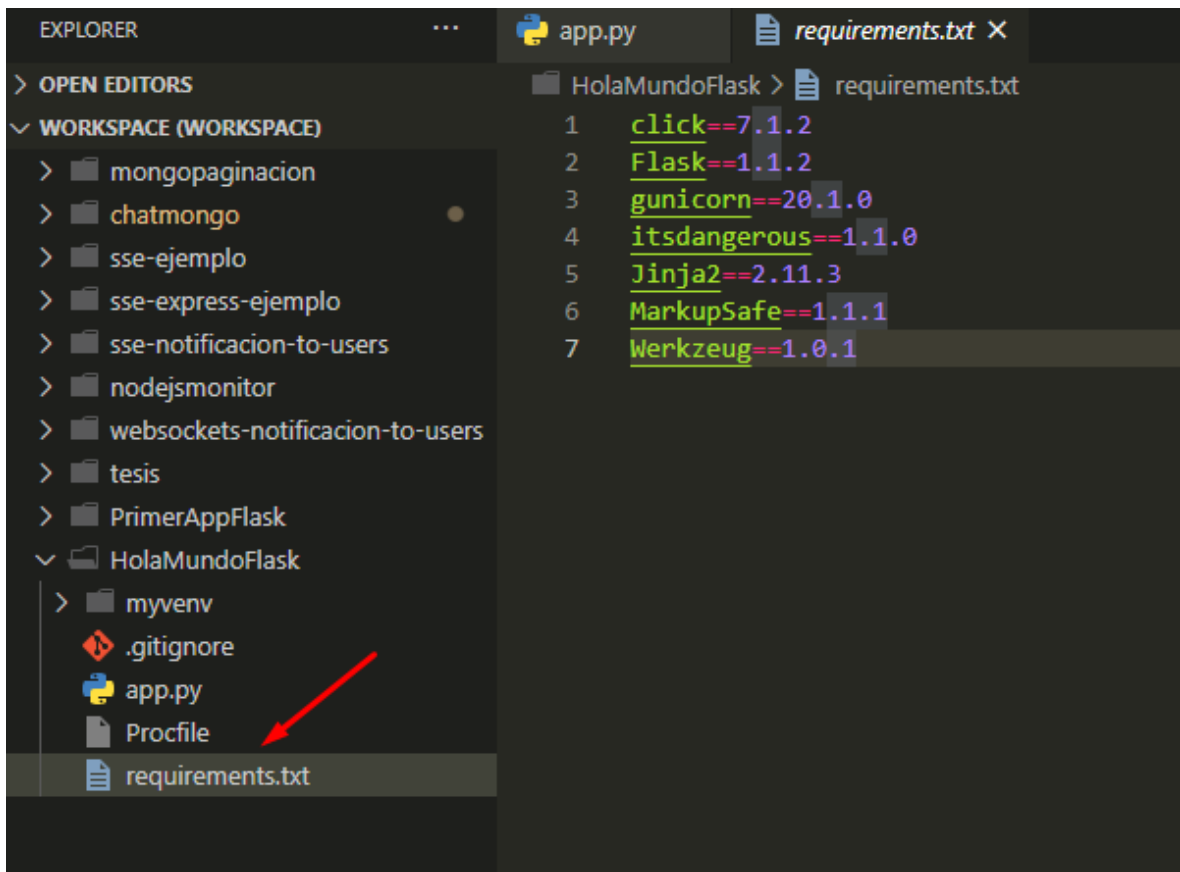
**Ahora tenemos que crear un archivo llamado “requirements.txt” para poner ahí todos los nombres de la librerías y versiones que se instalaran en Heroku:**

```
pip freeze > requirements.txt
```

Sin embargo, aquí hay un problema. El sistema no me deja escribir el archivo por que no ejecute anaconda prompt como administrador:

```
(myvenv) (base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>pip freeze > requeriments.txt
Acceso denegado.
(myvenv) (base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>
```

Como no me deja crear el archivo automáticamente con “pip freeze” y además haber probado abriendo el anaconda prompt como administrador, lo que vamos a hacer es crear el archivo manualmente dentro del proyecto y pegaremos ahí el resultado que nos arroje el comando “**pip freeze**”. Así copiaremos toda esa lista de librerías desde la consola y las pegaremos dentro del archivo creado.



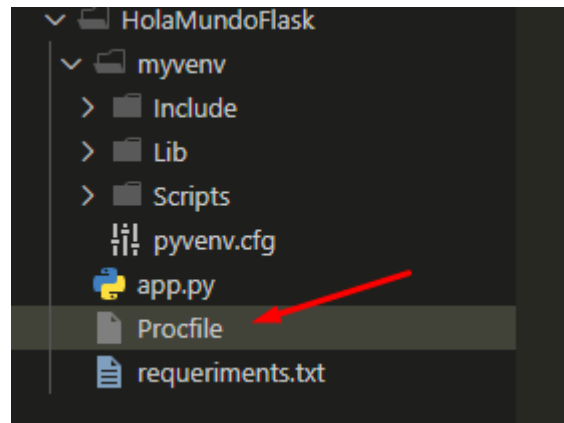
Verificar que el nombre del archivo este bien escrito o tendremos problemas.

## 6. Crear archivo Procfile

Un Procfile es un mecanismo para declarar qué comandos ejecutan los dynos de su aplicación en la plataforma Heroku. ... Otros procesos, como los workers en segundo plano, pueden tener cualquier nombre, y puede usar el cinturón de herramientas de Heroku para iniciar o detener esos procesos haciendo referencia a su nombre.

- ✓ **El archivo Procfile no tiene extensión y debe estar en el directorio raíz del proyecto.**

Creamos el archivo:



Luego a este archivo le agregaremos el siguiente contenido:

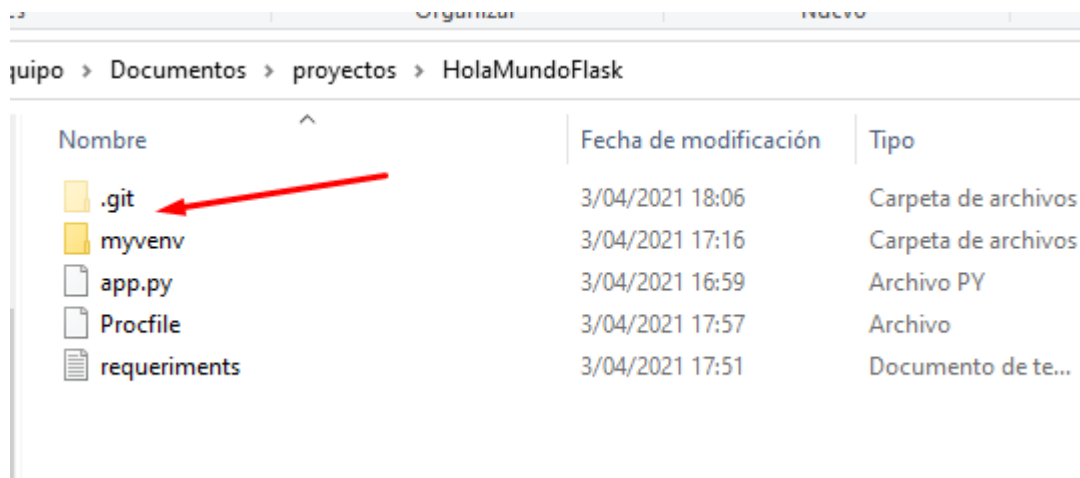
```
web: gunicorn app:app
```

Vemos que estamos usando la librería gunicorn, esta creara un servidor HTTP en producción ejecutando nuestro **app.py**. En este caso ya no usamos “Python app.py” si no que **gunicorn** se encargará de crear el servidor.

## 7. Crear repositorio del proyecto.

Con git ya instalado inicializamos un repositorio con el siguiente comando:

```
git init
```



Equipo > Documentos > proyectos > HolaMundoFlask		
Nombre	Fecha de modificación	Tipo
.git	3/04/2021 18:06	Carpeta de archivos
myvenv	3/04/2021 17:16	Carpeta de archivos
app.py	3/04/2021 16:59	Archivo PY
Procfile	3/04/2021 17:57	Archivo
requirements	3/04/2021 17:51	Documento de te...

Agregamos los archivos al repositorio:

```
git add app.py Procfile requirements.txt
```

```
s\HolaMundoFlask>git add app.py requirements.txt Procfile  
s\HolaMundoFlask>
```

No agregamos la carpeta “**myvenv**” por qué no es necesaria para el proyecto, esta solo la utilizamos para instalar las librerías y obtener los requerimientos.

Luego hacemos “**commit**”

```
git commit -m “primer commit”
```

```
(base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>git commit -m "primer commit"  
[master (root-commit) 888a63f] primer commit  
3 files changed, 19 insertions(+)  
create mode 100644 Procfile  
create mode 100644 app.py  
create mode 100644 requirements.txt
```

Fijémonos que dice que se agregamos esos cambios a la rama “master” los repositorios de github ahora usan una rama “main” como principal.

**Como he cerrado la terminal y la he vuelto abrir me voy a loguear con otra vez a travez de la Heroku CLI:**

```
heroku login -i
```

Al parecer las credenciales ya se quedan guardadas y no es necesario volver a loguearse.

**Ahora vamos a inicializar el repositorio remoto en nuestro proyecto:**

```
heroku git:remote -a surflaweb-flask
```

**Notar que le estamos pasando el nombre del proyecto Heroku creado anteriormente.**

```
(myvenv) (base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>heroku git:remote -a surflaweb-flask
set git remote heroku to https://git.heroku.com/surflaweb-flask.git
(myvenv) (base) C:\Users\freddyalc\Documents\proyectos\HolaMundoFlask>
```

## **8. Subir código del proyecto a Heroku**

Envíar el código al repositorio en Heroku usando Git

Solo nos resta hacer push con git y todo quedaría subido:

```
git push heroku master
```

y listo!



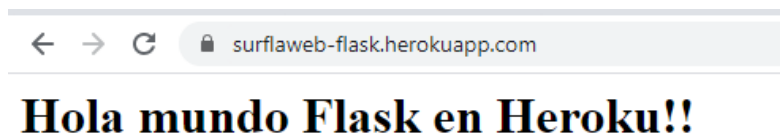
```

remote: -----> Building on the Heroku-20 stack
remote: -----> Determining which buildpack to use for this app
remote: -----> Python app detected
remote: -----> Requirements file has been changed, clearing cached dependencies
remote: -----> Installing python-3.6.13
remote: -----> Installing pip 20.1.1, setuptools 47.1.1 and wheel 0.34.2
remote: -----> Installing SQLite3
remote: -----> Installing requirements with pip
remote: Collecting click==7.1.2
remote:   Downloading click-7.1.2-py2.py3-none-any.whl (82 kB)
remote: Collecting Flask==1.1.2
remote:   Downloading Flask-1.1.2-py2.py3-none-any.whl (94 kB)
remote: Collecting gunicorn==20.1.0
remote:   Downloading gunicorn-20.1.0.tar.gz (370 kB)
remote: Collecting itsdangerous==1.1.0
remote:   Downloading itsdangerous-1.1.0-py2.py3-none-any.whl (16 kB)
remote: Collecting Jinja2==2.11.3
remote:   Downloading Jinja2-2.11.3-py2.py3-none-any.whl (125 kB)
remote: Collecting MarkupSafe==1.1.1
remote:   Downloading MarkupSafe-1.1.1-cp36-cp36m-manylinux2010_x86_64.whl (32 kB)
remote: Collecting Werkzeug==1.0.1
remote:   Downloading Werkzeug-1.0.1-py2.py3-none-any.whl (298 kB)
remote: Building wheels for collected packages: gunicorn
remote: Building wheel for gunicorn (setup.py): started
remote: Building wheel for gunicorn (setup.py): finished with status 'done'
remote: Created wheel for gunicorn: filename=gunicorn-20.1.0-py3-none-any.whl size=78918 sha256=cb
08b
remote:   Stored in directory: /tmp/pip-ephem-wheel-cache-x23gea8m/wheels/9a/86/37/cad4bc71746b420e1
remote: Successfully built gunicorn
remote: Installing collected packages: click, itsdangerous, MarkupSafe, Jinja2, Werkzeug, Flask, gun
remote: Successfully installed Flask-1.1.2 Jinja2-2.11.3 MarkupSafe-1.1.1 Werkzeug-1.0.1 click-7.1.2
remote: -----> Discovering process types
remote: Procfile declares types -> web
remote:
remote: -----> Compressing...
remote: Done: 48.4M
remote: -----> Launching...
remote: Released v5
remote: https://surflaweb-flask.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/surflaweb-flask.git
* [new branch] master -> master

```

Por defecto instalo la app con **Python 3.6**, también podemos crear un archivo llamado “runtime.txt” en el directorio raíz del proyecto especificando la versión de Python que deseamos por ejemplo “:

```
python-3.9.2
```



## Deploy a Heroku con Github


Lo que vamos hacer queridos amigos es vincular nuestra cuenta de github a heroku :D así nos evitaremos varios problema en caso de que no funcione la CLI de Heroku con Git.

Vamos a crear un repositorio primero en **github** y subiremos todos los archivos excepto la carpeta “myvenv”.


**Por lo tanto, creamos un repositorio, en mi caso será privado porque no quiero que otros vean el código de mi proyecto privado por lo que solo estará disponible con heroku previa vinculación de la cuenta.**

Owner \*

Repository name \*

 alcarazolabs ▾

/


flask-holamundo-heroku 

Great repository names are short and memorable. Need inspiration? How about [studious-potato?](#)

Description (optional)


Proyecto hola mundo para heroku.

☐

 **Public**

Anyone on the internet can see this repository. You choose who can commit.

☒

 **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐

**Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

☐

**Add .gitignore**

Choose which files not to track from a list of templates. [Learn more.](#)

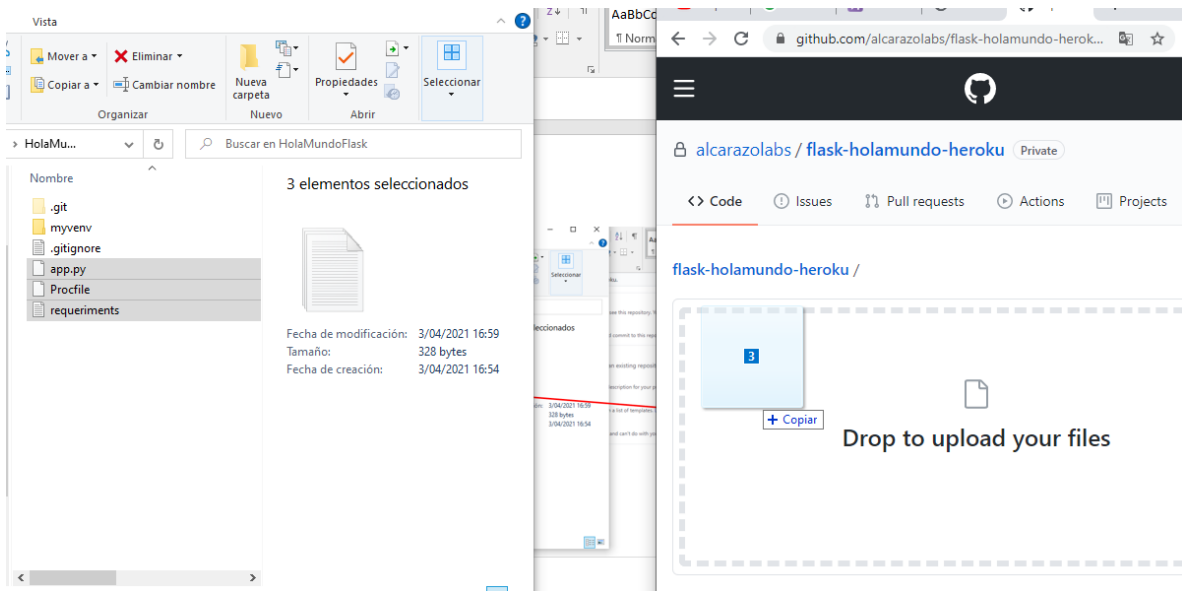
☐

**Choose a license**

A license tells others what they can and can't do with your code. [Learn more.](#)

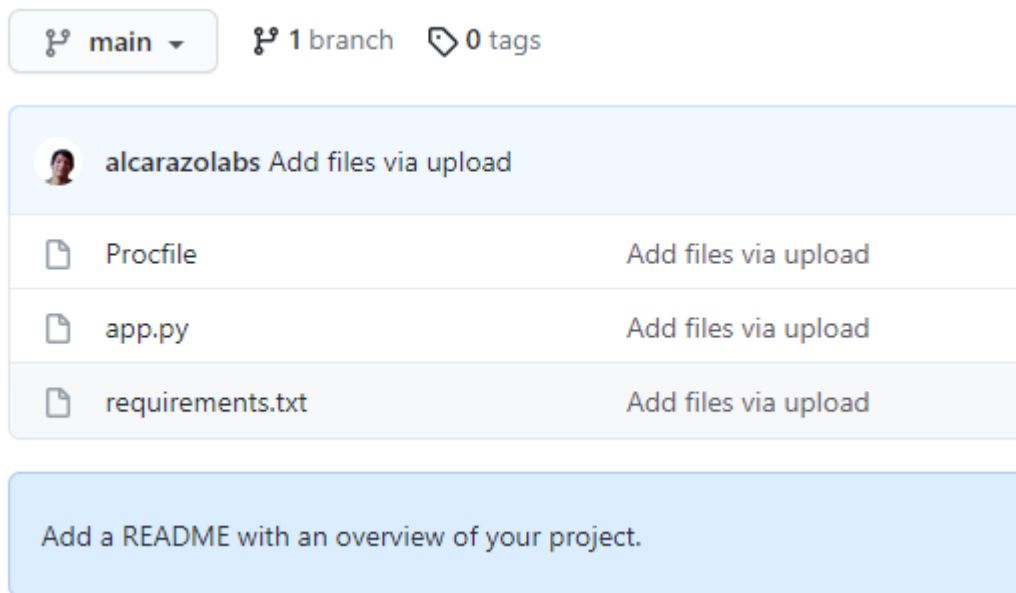
Create repository

Creado el repositorio arrastramos o subimos como deseamos los archivos al repositorio, en mi caso solo arrastre los archivos al repo:

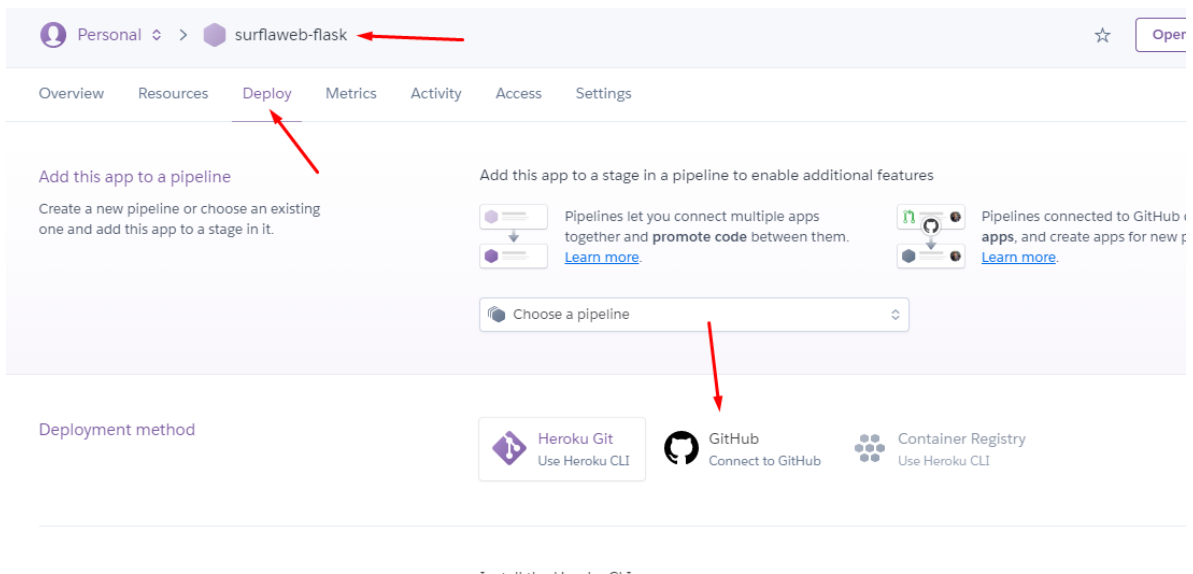


Ojo! En esta imagen hay un error, que corregí luego miren el nombre del archivo “requirements.txt” esta como “requeriments.txt”

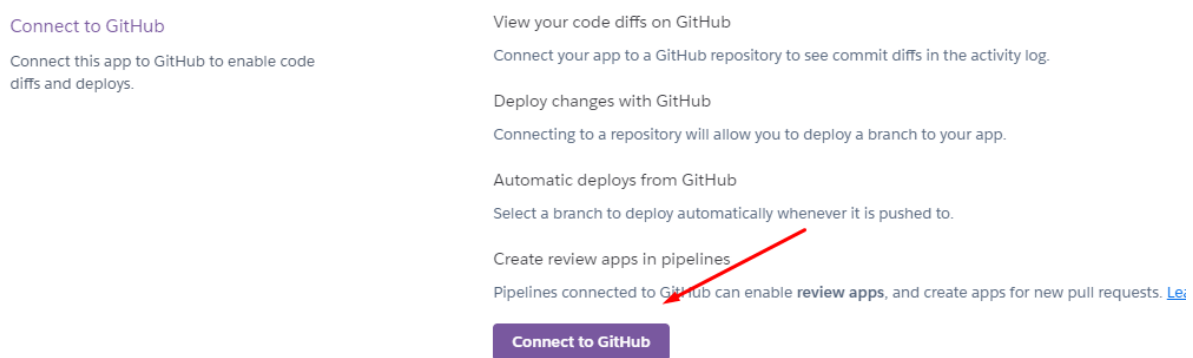
**Y listo:**



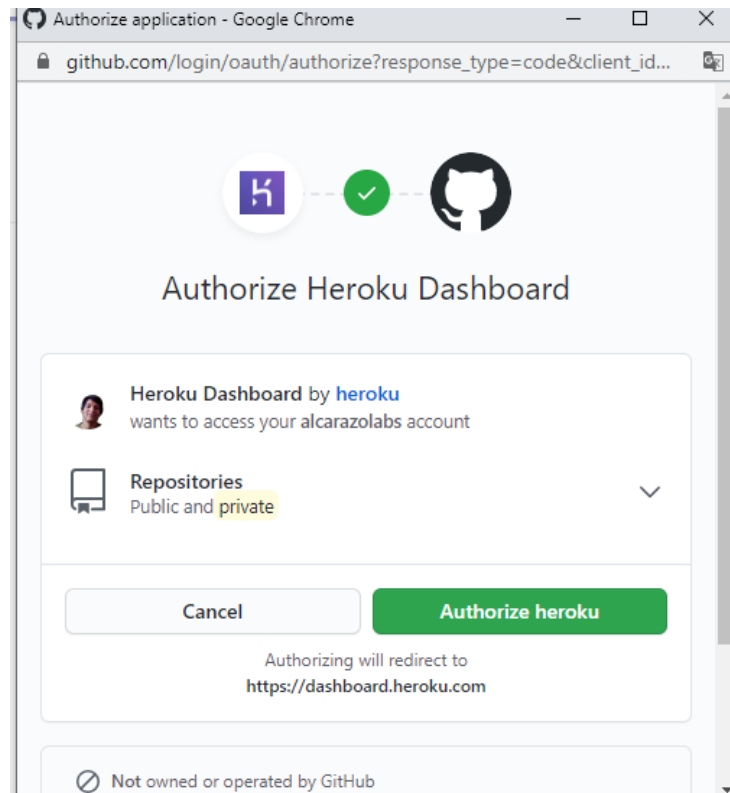
Ahora nos vamos a la página de Heroku en la opción “deploy” del proyecto creado:



Hacemos click en “Github”

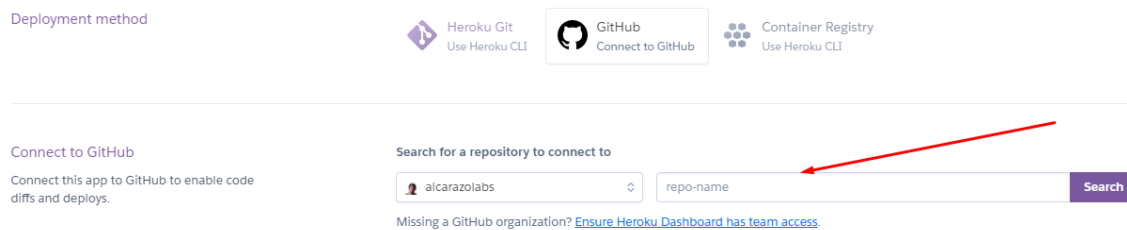


Ahora nos dirá que necesitará de acceder a nuestros repositorios:

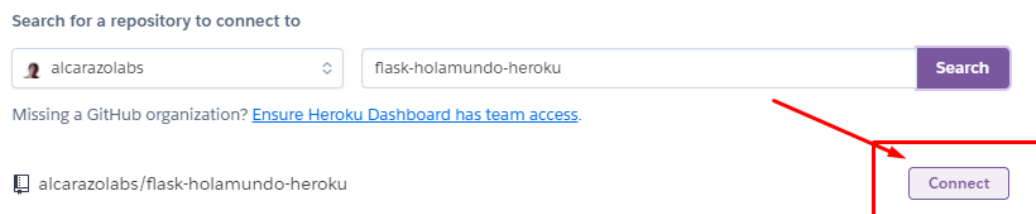


Hace click en "Authorize heroku"


Y listo! Ahora introducimos el nombre del repositorio y lo buscamos:



Hacemos click en "Connect"




Luego click en “Enable automatic deploys”

 You can now change your main deploy branch from “master” to “main” for both manual and automatic deploys, please follow the instructions [here](#).

Enable automatic deploys from GitHub

Every push to the branch you specify here will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch is always in a deployable state and any tests have passed before you push. [Learn more](#).

Choose a branch to deploy

 main

☐ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

**Enable Automatic Deploys**


Esto es que cada cambio que suceda en el repo de github también se reflejará en Heroku, elegimos la rama “main” como es sabido github renombro su rama “master” a “main” por motivos sociales.

Por ultimo hacemos click en el botón “Deploy branch”

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. [Learn more](#).

Choose a branch to deploy

 main

**Deploy Branch**

¡Y listo!

Receive code from GitHub

✓

Build main 2a637534

✓

Release phase

✓

Deploy to Heroku

✓

Your app was successfully deployed.

View

← → ↻

surflaweb-flask.herokuapp.com

**Hola mundo Flask en Heroku, código de Github!!**

Listo!