

# AWS - Flask API

En este taller vas a poner en marcha una API Flask de Python dentro de un entorno Cloud como es el de Amazon. Desarrollarás tu API en local y la desplegarás en la nube, pudiendo acceder a la misma desde cualquier lugar.

Todos los recursos de AWS que se van a crear en este taller son gratuitos en la Free Tier de AWS durante los dos primeros meses. No obstante, se recomienda eliminarlos después del taller para evitar futuros cargos.

## Seguiremos los siguientes pasos:

1. Configurar un entorno virtual de Python en local, con los paquetes necesarios para desarrollar una API Flask.
2. Crearemos una API Flask
3. Implementación de la API en AWS mediante Elastic Beanstalk.
4. Eliminación de los recursos.

## Requisitos

Para el desarrollo de este taller necesitaremos:

- [Cuenta gratuita de AWS](#)
- Tener Python instalado en local

**NOTA:** Para el taller se recomienda crear una carpeta nueva, que llamaremos **eb-flask**, en la ruta que desees del ordenador.

## Crear un entorno virtual de Python

Cuando se realiza un despliegue en un entorno productivo no es necesario que vaya con todos los paquetes que has utilizado en tus analíticas, por lo que habrá que crear un entorno de Python desde 0, con los paquetes necesarios. Para ello, partiremos del intérprete de Python 3 que tengamos instalado, y añadiremos las librerías que consideremos.

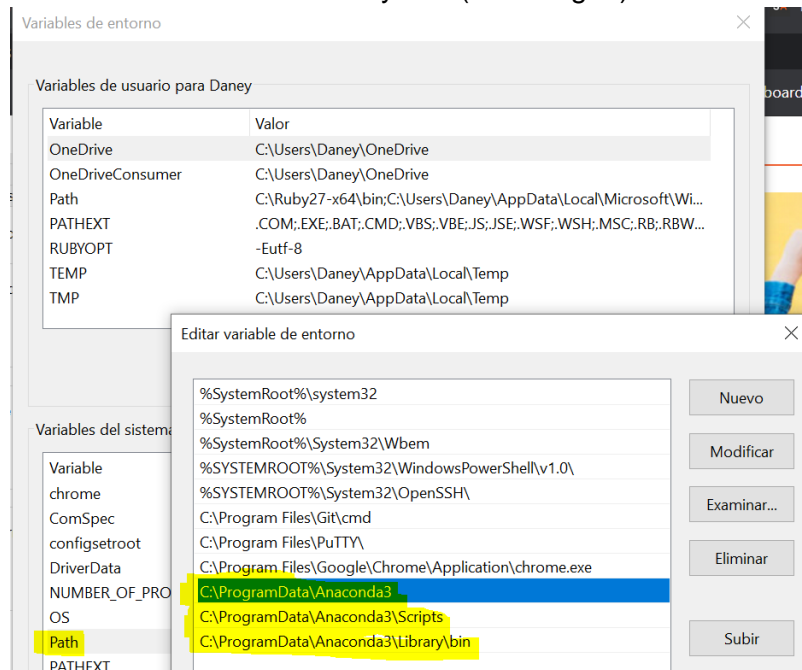
### Windows

Abre un terminal y utiliza el siguiente comando:

```
python -m venv C:\Users\usuario\eb-flask\virt
```

La ruta es dónde queremos que se cree el nuevo entorno virtual. En este caso “*virt*” sería su nombre. **Es necesario tener Python configurado en las variables de entorno de Windows**, de tal manera que solo haya que escribir “*python*”, y no la ruta entera del intérprete.

En caso de que el terminal no reconozca “python”, tendremos que actualizar las variables de entorno con las rutas de instalación de Python (Ver imagen)



Tras este paso, ya debería reconocer el comando “*python --version*”.

Accede al entorno virtual:

```
C:\Users\usuario\eb-flask\virt\Scripts\activate
```

Debería aparecer delante de la línea de comandos el nombre del entorno virtual. Después, mediante “*pip freeze*” veremos los paquetes que tenga instalados. De momento, ninguno.

```
C:\Users\Daney>C:\Users\Daney\Desktop\DEMOS\eb-flask\virt\Scripts\activate
(virt) C:\Users\Daney>pip freeze
(virt) C:\Users\Daney>
```

## MAC

IOS ya viene con un intérprete de Python instalado, pero no es ese el que queremos usar, sino el instalado a través de Anaconda. Ejecutamos:

```
which python
```

Que nos dará las rutas de todas las instalaciones de Python en el ordenador. A continuación ejecutamos:

```
c:/ruta/anaconda/python -m venv C:/ruta/eb-flask/virt
```

Donde la primera ruta se corresponde con el intérprete de Python que deseamos utilizar, y la segunda con dónde queremos que se cree el nuevo entorno virtual. “virt” es el nombre que le damos al entorno.

A continuación accedemos al entorno virtual mediante el comando:

```
source C:/ruta/nuevo/entorno/bin/activate
```

## Los siguientes pasos ya son comunes a Windows y MAC.

Ejecuta “*pip freeze*”. Esto devuelve todos los paquetes que hayas instalado. De momento ninguno.

Ahora instalamos flask

```
pip install flask
```

Comprueba de nuevo mediante “*pip freeze*” que flask está instalado. Ten en cuenta que flask tiene otras dependencias, por lo que no será el único paquete instalado.

Guarda la salida de los paquetes en un archivo llamado “*requirements.txt*”, este archivo le servirá a Elastic Beanstalk para instalar los paquetes en su entorno virtual. El entorno virtual que acabamos de crear no lo vamos a subir a la nube. Es el propio recurso de Amazon el que lo despliega a partir del “*requirements.txt*” que hemos obtenido.

```
pip freeze > C:\Users\usuario\eb-flask\requirements.txt
```

## Aplicación en Flask

En este punto del taller vamos a crear una APP sencilla, que primero probaremos en local para comprobar su correcto funcionamiento, y después desplegaremos en la nube. Simplemente crea un script de Python en la ruta de la app, y llámalo “*application.py*”

```
C:\Users\usuario\eb-flask\application.py
```

Dentro irá el siguiente código:

```
from flask import Flask

# print a nice greeting.
def say_hello(username = "World"):
    return '<p>Hello %s!</p>\n' % username

# some bits of text for the page.
header_text = '''
    <html>\n<head> <title>EB Flask Test</title> </head>\n<body>'''
instructions = '''
    <p><em>Hint</em>: This is a RESTful web service! Append a
username
```

```

    to the URL (for example: <code>/Thelonious</code>) to say hello
    to
    someone specific.</p>\n'''
home_link = '<p><a href="/">Back</a></p>\n'
footer_text = '</body>\n</html>'

application = Flask(__name__)

application.add_url_rule('/', 'index', (lambda: header_text +
    say_hello() + instructions + footer_text))

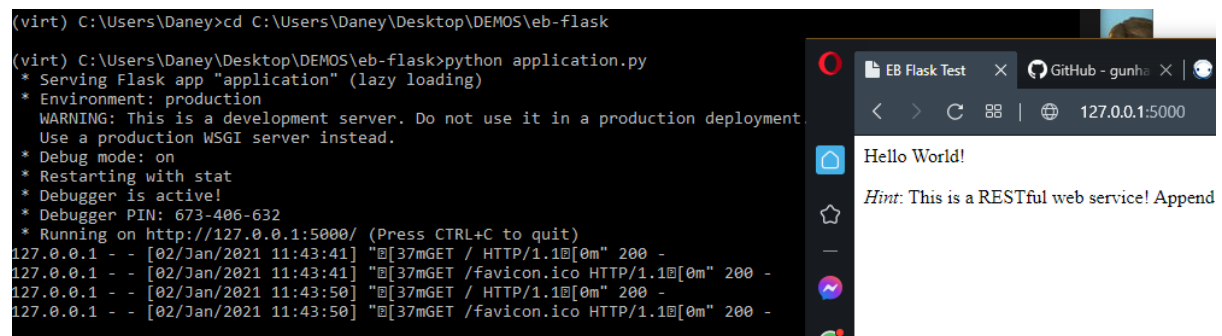
application.add_url_rule('/<username>', 'hello', (lambda username:
    header_text + say_hello(username) + home_link + footer_text))

# run the app.
if __name__ == "__main__":

    application.debug = True
    application.run()

```

Lo siguiente que haremos será ejecutar el script para comprobar que funciona correctamente



The screenshot shows a terminal window with the following output:

```

(virt) C:\Users\Daney>cd C:\Users\Daney\Desktop\DEMOS\eb-flask
(virt) C:\Users\Daney\Desktop\DEMOS\eb-flask>python application.py
* Serving Flask app "application" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 673-406-632
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [02/Jan/2021 11:43:41] "[37mGET / HTTP/1.1[0m" 200 -
127.0.0.1 - - [02/Jan/2021 11:43:41] "[37mGET /favicon.ico HTTP/1.1[0m" 200 -
127.0.0.1 - - [02/Jan/2021 11:43:50] "[37mGET / HTTP/1.1[0m" 200 -
127.0.0.1 - - [02/Jan/2021 11:43:50] "[37mGET /favicon.ico HTTP/1.1[0m" 200 -

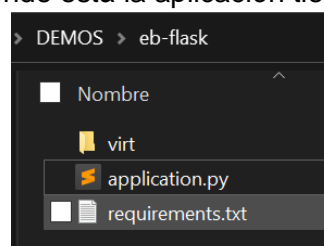
```

Next to the terminal is a web browser window titled "EB Flask Test" showing the URL "127.0.0.1:5000". The browser displays "Hello World!" and a hint: "Hint: This is a RESTful web service! Append".

**¿Qué tenemos hecho hasta ahora?** Hemos creado un nuevo entorno virtual únicamente con los paquetes necesarios para este proyecto, así como sus versiones. Esta información la hemos exportado a un .txt que será todo lo que necesitará Elastic Beanstalk para desplegar la aplicación.

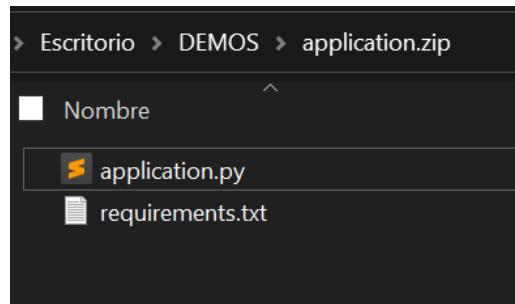
Además, hemos creado una sencilla app y desplegado en local con el nuevo intérprete, no con el de Anaconda en el que se incluyen todos los paquetes.

Asegúrate de que el directorio donde está la aplicación tiene la siguiente pinta:

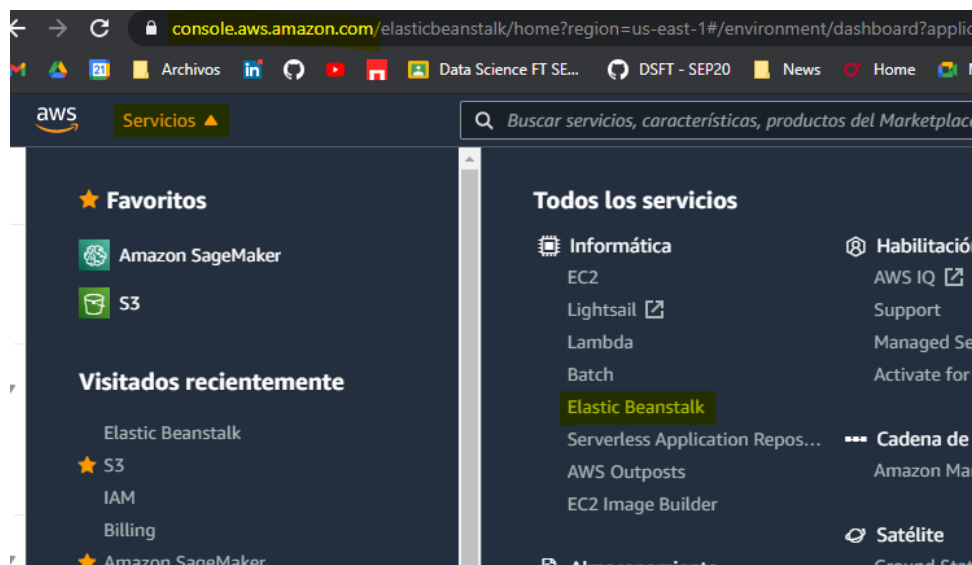


## Despliegue en Elastic Beanstalk

Ya tenemos todo lo necesario para realizar el despliegue. Lo primero que haremos será montar el paquete con el software. Simplemente hay que montar un zip con “*application.py*” y “*requirements.txt*”.



Ahora vamos a la consola de AWS -> Servicios -> Elastic Beanstalk. Desde aquí podremos desplegar, configurar y monitorizar cualquier despliegue web que hagamos. Ya que lo que queremos subir es una API que esté accesible desde la web, este es el sitio.



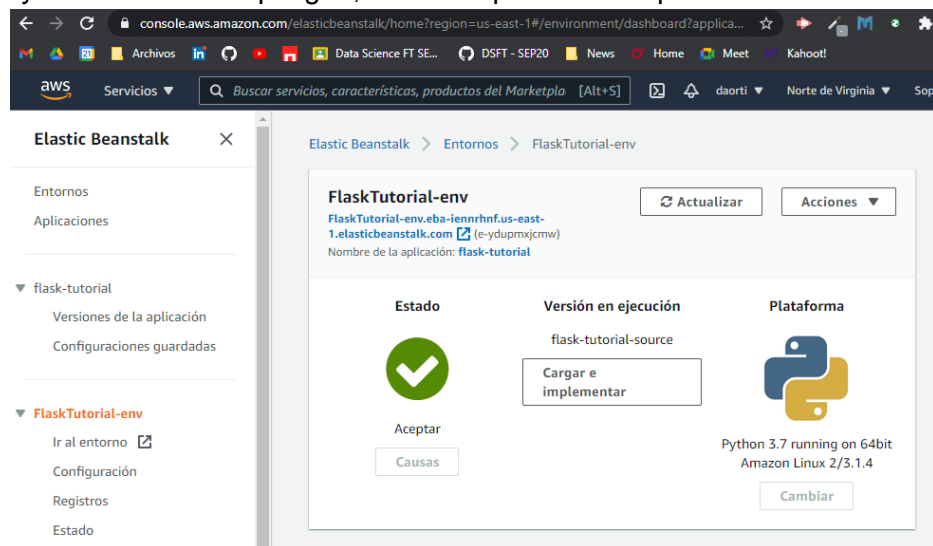
Seguiremos los siguientes pasos (todo lo que no se indique en estos pasos, déjalo por defecto):

1. Create Application
2. Introducir nombre. Cuidado con espacios y caracteres raros.
3. Etiquetas. No es necesario tocar nada
4. Plataforma -> Python
5. Ramificación de la plataforma -> La versión de Python tiene que coincidir con la del entorno virtual que hayamos creado. Por simplicidad en el desarrollo del taller, se van a usar versiones distintas (3.8 vs 3.7), que no es algo grave, pero para un despliegue productivo procura que las versiones coincidan. Selecciona la 3.7.
6. Código de aplicación -> Cargar el código -> Elegir archivo. Sube el zip creado con el código y los requirements.
7. Crear una aplicación

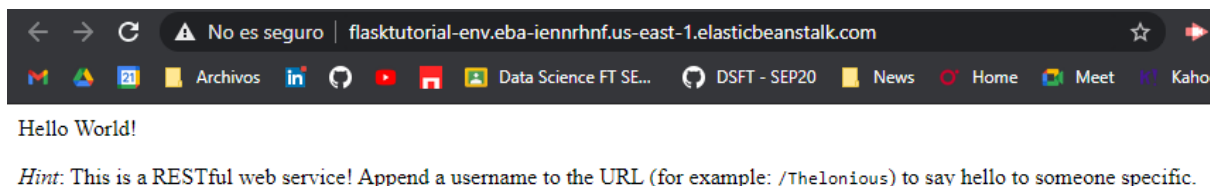
Automáticamente va a realizar el despliegue de la APP, con todos los recursos necesarios. **Esto puede tardar unos minutos.** Realizarlo manualmente es algo tedioso, ya que necesitamos una máquina, un grupo de seguridad, un balanceador de carga, entre otros recursos... Por suerte EC tiene esto bastante automatizado y nos ahorra mucho tiempo.

**NOTA:** Si se desea, se puede configurar todo el despliegue por línea de comandos mediante el CLI de EB. La diferencia es que no hace falta acceder al portal de AWS, sino que desde un terminal podemos realizar todo el despliegue y configuración de la APP, así como de su entorno.

Una vez haya acabado el despliegue, debería aparecer una pantalla similar a la siguiente:



Haz click en “Ir al entorno” en el menú de la izquierda para acceder al servicio web, ya desplegado en la nube.



## Actualización del servicio

Vamos a realizar una actualización del servicio web que tenemos desplegado. Añadiremos algunos datos con un enrutado para acceder a los mismos. Crea una nueva aplicación (application.py) y añade:

```
from flask import Flask,jsonify
from data.datos_dummy import books
```

```
application = Flask(__name__)
```

```

@application.route('/')
def home():
    return "<h1>Distant Reading Archive</h1><p>This site is a prototype API for distant
reading of science fiction novels.</p>"

@application.route('/api/v1/resources/books/all', methods=['GET'])
def get_all():
    # Para pasar a un json, que es lo que devuelve la API
    return jsonify(books)

# run the app.
if __name__ == "__main__":
    # Setting debug to True enables debug output. This line should be
    # removed before deploying a production app.
    application.debug = True
    application.run()

```

---

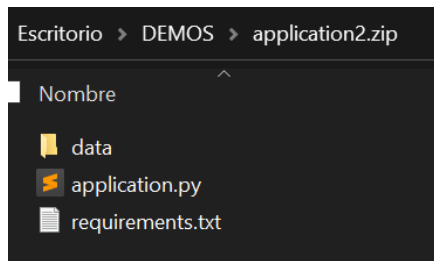
Fíjate que en el código importa datos de un script llamado *"data.datos\_dummy"*. Por tanto:

1. Crea una nueva carpeta llamada "data"
2. Dentro un script nuevo llamado "datos\_dummy.py"
3. Añade en el script la siguiente lista de libros:

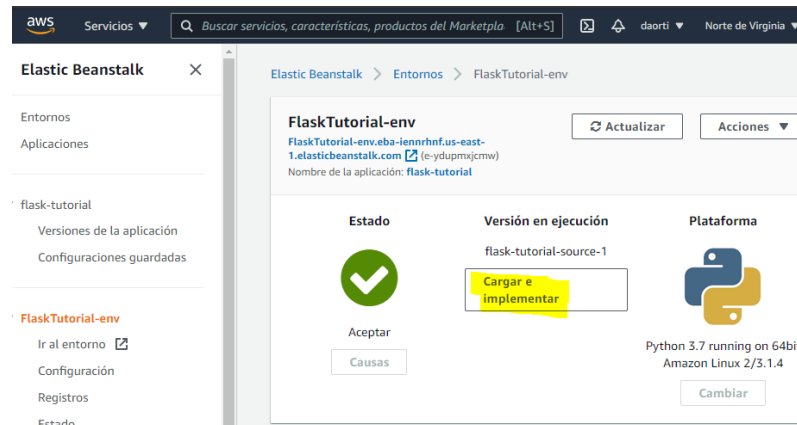
```

books = [
    {'id': 0,
     'title': 'A Fire Upon the Deep',
     'author': 'Vernor Vinge',
     'first_sentence': 'The coldsleep itself was dreamless.',
     'year_published': '1992'},
    {'id': 1,
     'title': 'The Ones Who Walk Away From Omelas',
     'author': 'Ursula K. Le Guin',
     'first_sentence': 'With a clamor of bells that set the swallows soaring, the Festival
of Summer came to the city Omelas, bright-towered by the sea.',
     'published': '1973'},
    {'id': 2,
     'title': 'Dhalgren',
     'author': 'Samuel R. Delany',
     'first_sentence': 'to wound the autumnal city.',
     'published': '1975'}
]

```
4. Monta el nuevo zip con los siguientes archivos:



Ya tienes el nuevo paquete listo para desplegar. Ve a la consola de AWS y carga la nueva versión de la app:

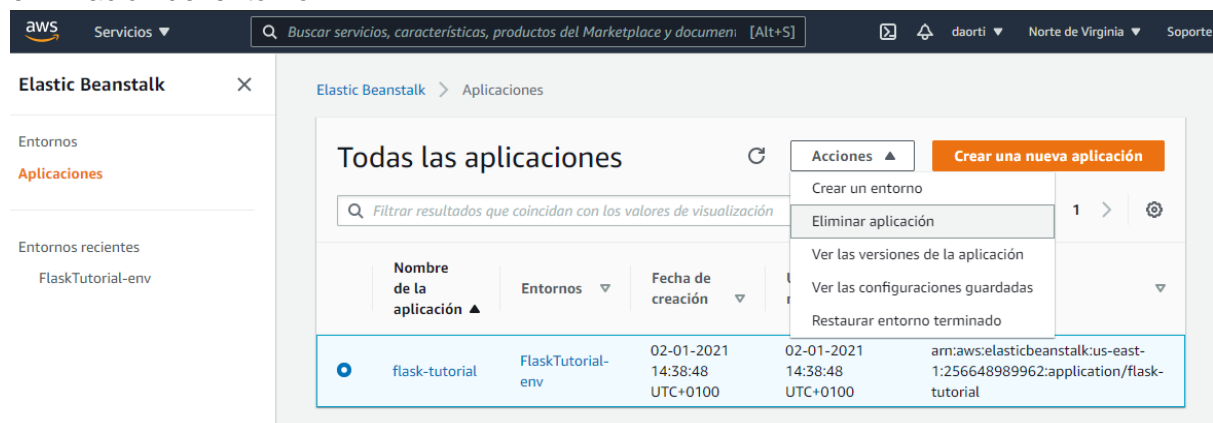


Accede ahora a los datos añadidos en la APP.

## Eliminación

El acceso a estos recursos no es ilimitado, por lo que deberías eliminar todo lo que hayas creado en este taller para evitar futuros cobros en AWS.

Elimina la aplicación. Suele tardar unos minutos en eliminarse. También se producirá la eliminación del entorno.



Por último, elimina el bucket de S3 que crea EC. Ve a Servicios -> Almacenamiento -> S3 -> Buckets, y elimínalo.



aws

Servicios

Buscar servicios, características, productos del Marketplace y documentos [Alt+S]

Amazon S3

Buckets

Puntos de acceso

Operaciones por lotes

Analizador de acceso para S3

Configuración de la cuenta para el bloqueo de acceso público

Storage Lens

Paneles

Configuración de AWS Organizations

Características destacadas

Seguimos mejorando la consola de S3 para que sea más rápida y sencilla de utilizar. Si tiene comentarios sobre la experiencia actualizada, elija **Proporcionar comentarios**.

Amazon S3

Buckets (1)

Los buckets son contenedores de datos almacenados en S3. [Más información](#)


Copiar ARN

Vaciar

Eliminar

Crear bucket

Buscar buckets por nombre

	Nombre	Región	Acceso
	elasticbeanstalk-us-east-1-256648989962	EE. UU. Este (Norte de Virginia) us-east-1	Los objetos pueden ser públicos