

Desarrollo de un Microservicio con Índices Invertidos y Despliegue en DigitalOcean

Amir Rodríguez Mejía - 506222032
 Juan Esteban Pinzon Preciado - 506222727
 Hugo Alejandro Latorre Portela - 506222722
 Jose David Ramirez Beltrán - 506222723
 Fundación Universitaria Konrad Lorenz

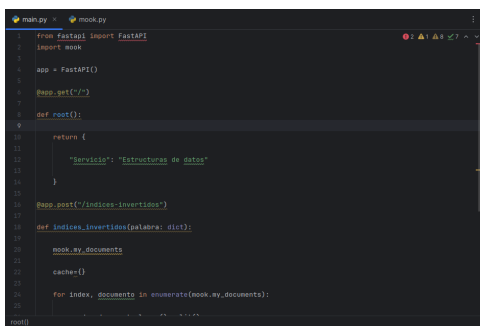
I. INTRODUCCIÓN

En el presente documento se estaría creando un algoritmo con el fin que sea convertido en un micro-servicio el cual tiene una tarea o función a desarrollar. Ahora, en este caso vamos hacer uso de varias herramientas como lo son los servidores en específico Digital Ocean el cual se encuentra configurado a base de las opciones y herramientas de Ubuntu o Nginx es un servidor web y servidor proxy ligero de código abierto que se utiliza comúnmente para servir sitios web estáticos y dinámico, APIS en específico el Framework de FASTAPI, Uvicorn como controlador operacional de salidas y entradas, el cual permite manejar múltiples conexiones simultáneamente de forma eficiente, entre otros.

II. DISEÑO DEL MICROSERVICIO

II-A. Código Completo del microservicio “Indices invertidos”

EN las siguientes imágenes se visualiza el algoritmo que será un microservicio incluyendo el retorno en la parte inferior, el cual devuelve la línea de texto donde se encuentre relacionada la palabra escrita de manera manual que se desee buscar.



```

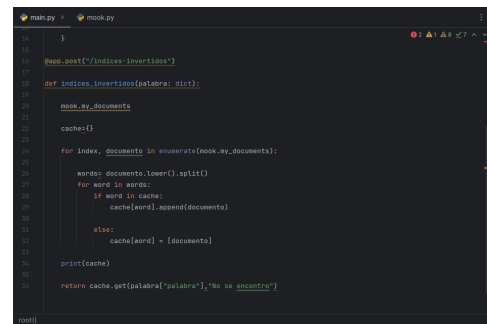
main.py  mock.py
from fastapi import FastAPI
import uvicorn
import mock

app = FastAPI()

@app.get("/")
def root():
    return {
        "Servicio": "Estructuras de datos"
    }

@app.post("/indices-invertidos")
def indices_invertidos(palabra: dict):
    mock.py_documents
    cache={}
    for index, documento in enumerate(mock.py_documents):
        words= documento.lower().split()
        for word in words:
            if word in cache:
                cache[word].append(documento)
            else:
                cache[word] = [documento]
    print(cache)
    return cache.get(palabra["palabra"],"No se encuentra")
  
```

Figura 1. Algoritmo indices invertidos



```

main.py  mock.py
def indices_invertidos(palabra: dict):
    mock.py_documents
    cache={}
    for index, documento in enumerate(mock.py_documents):
        words= documento.lower().split()
        for word in words:
            if word in cache:
                cache[word].append(documento)
            else:
                cache[word] = [documento]
    print(cache)
    return cache.get(palabra["palabra"],"No se encuentra")
  
```

Figura 2. Algoritmo indices invertidos

A continuación se relaciona el endpoint del microservicio, por definición se refiere a una URL específica en una API web o servicio web que se utiliza para realizar operaciones o acceder a recursos específicos. En el contexto de una aplicación web o API, un endpoint está asociado a una ruta y un método HTTP específico (GET, POST, PUT, DELETE, etc.).



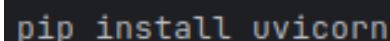
```

@app.get("/")
def root():
    return {
        "Servicio": "Estructuras de datos"
    }

@app.post("/indices-invertidos")
def indices_invertidos(palabra: dict):
  
```

Figura 3. Endpoint del microservicio

Para que el servicio sea creado en un servidor de manera local se hace necesario la instalación del framework Fast API y el servidor Uvicorn el cual se utiliza para iniciar y gestionar el servidor que sirve la aplicación web.



```

pip install uvicorn
  
```

Figura 4. Instalación del framework FastAPI

```
main.py x mock.py
1 from flask import Flask
2 import gunicorn
3
4 app = Flask(__name__)
5
6 if __name__ == '__main__':
7     gunicorn.run(main=app, host='0.0.0.0', port=8080)
```

Figura 5. Instalación del gestor de servidor web

Siendo así que para iniciar una aplicación web se debe realizar el siguiente comando.

```
(venv) amir28@PC:~/PycharmProjects/TallerCinco$ uvicorn main:app --port 8080
INFO: Started server process [18172]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://127.0.0.1:8080 (Press CTRL+C to quit)
```

Figura 6. Comando para inicializar la aplicación web en el servidor local

III. PROCEDIMIENTO PARA EL DESPLIEGUE DEL SERVIDOR EN DIGITAL OCEAN

1- Debemos ingresar a digital ocean y realizar la creación del droplet el cual es una vps, recordemos que la configuración es la terminal de ubuntu para hacer uso de los comandos propios del sistema operativo, de igual manera debemos elegir el servidor, en este caso hicimos uso del servidor ubicado en San Francisco – Estados Unidos.

2- Una vez que Digital Ocean nos indique el IP el cual se puede acceder con contraseña, nos dirigimos a la interfaz del entorno “Pycharm” para poder ingresar o tener acceso al servidor de la siguiente manera:

```
Terminal Local x + v
(venv) amir28@PC:~/PycharmProjects/TallerCinco$ ssh root@24.199.112.172
The authenticity of host '24.199.112.172 (24.199.112.172)' can't be established
```

Figura 7. Ingreso al servidor

3- De esta manera ingresamos al servidor y se puede visualizar de la siguiente manera:

```
Last login: Sat Oct 21 21:36:42 2023 from 181.237.93.79
root@Buscador:~#
```

Figura 8. Ingreso al servidor

4- Con el siguiente comando actualizamos los paquetes registrados en ubuntu.

```
root@Buscador:~# apt update
0% [Working]
```

Figura 9. Comando de actualización

5- Posteriormente procedemos a realizar las instalaciones como nginx, uvicorn, python3, entre otros.

5.1 A continuación se visualiza el comando para instalar el servidor http nginx, es decir nuestro servidor web:

```
0 upgraded, 0 newly installed, 0 to remove and 0 not installed.
root@Buscador:~# apt install nginx
```

Figura 10. Instalación Nginx

5.2 Posteriormente validamos si se encuentra instalado git, con el fin de realizar el proceso en el cual se clona el microservicio en el servidor:

```
nginx: [emerg] invalid number of arguments in 'map' directive
root@Buscador:~# git
```

Figura 11. Instalación git

6- Para realizar el proceso de clonación, tomamos la URL de clonación, con el fin de realizar el comando para subir el microservicio al servidor, el proceso se visualiza a continuación:

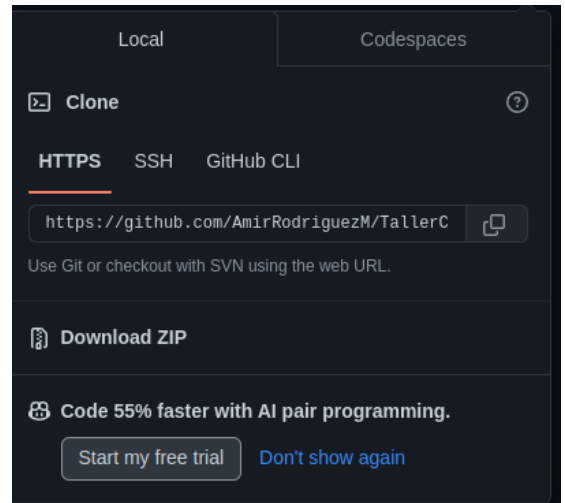


Figura 12. Url a clonar

7- No sin antes haber ingresado a la carpeta home que tiene el servidor y luego escribimos ls, allí es donde se realiza el proceso de clonación, el proceso se visualiza a continuación:

```
Terminal Local x + v
root@Buscador:~# cd /home/
root@Buscador:/home# ls
Taller5
root@Buscador:/home#
```

Figura 13. Ubicación del espacio para clonar el microservicio

8- Se procede a realizar el proceso de clonación:

```
root@Buscador:/home# git clone https://github.com/AmirRodriguezM/TallerCinco.git
```

Figura 14. Comando para clonar

9- Una vez allí para ingresar al microservicio realizamos el siguiente proceso:

```

root@Buscador:/home# ls
Taller5 TallerCinco
root@Buscador:/home# cd TallerCinco
-bash: cd: TallerCinco: No such file or directory
root@Buscador:/home# cd TallerCinco
root@Buscador:/home/TallerCinco#

```

Figura 15. Ingreso al microservicio

10- Posteriormente ingresamos a la versión de python 3 y luego escribimos exit():

```

root@Buscador:/home/TallerCinco# python3
Python 3.11.4 (main, Jun 9 2023, 07:59:55) [GCC 12.3.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>>

```

Figura 16. Establecimiento de configuración python3

11- Luego instalamos Uvicorn pero versión python 3 con el siguiente comando:

```

root@Buscador:/home/TallerCinco# pip3 install uvicorn

```

Figura 17. Instalación Uvicorn versión python 3

12- Luego instalamos el paquete pip de python3:

```

root@Buscador:/home/TallerCinco# apt install python3-pip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done

```

Figura 18. Instalación Uvicorn versión python 3

13- Posteriormente instalamos uvicorn en:

```

0 upgraded, 0 newly installed, 0 to remove and 32 not upgraded
root@Buscador:/home/TallerCinco# pip3 install uvicorn

```

Figura 19. Instalación Uvicorn versión python 3

14- De la siguiente manera iniciamos nuestra aplicación:

```

root@Buscador:/home/TallerCinco#
root@Buscador:/home/TallerCinco# uvicorn main:app --port 8001
INFO: Started server process [442766]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://127.0.0.1:8001 (Press CTRL+C to quit)

```

Figura 20. Microservicio en funcionamiento de manera local

15- En caso que el fastapi no se encuentre instalado, simplemente realizamos el siguiente comando:

```

root@Buscador:/home/TallerCinco# pip3 install fastapi

```

Figura 21. Instalación fastapi

16- En este momento ingresamos a la URL que nos arroja uvicorn de manera local, solo que en este caso va arrojar un

error debido a que se está dirigiendo o se está apuntado hacia si misma.

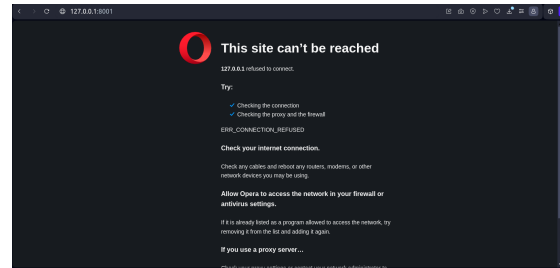


Figura 22. Servidor caído debido a la terminal cerrado

Para solucionar el auto apuntamiento, se crea un -host con el fin que se apunte directamente al IP de nuestro servidor.

```

root@Buscador:/home/TallerCinco# uvicorn main:app --port 8001 --host 24.199.112.172
INFO: Started server process [442878]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://24.199.112.172:8001 (Press CTRL+C to quit)

```

Figura 23. Microservicio en funcionamiento de manera pública

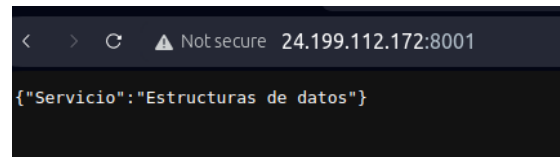


Figura 24. Vista desde el navegador

Hasta este momento el servidor se cae o falla si cerramos la consola, por esa razón es necesario realizar el proceso en el cual se configura el microservicio para que sea escuchado por medio del puerto 80.

17- Procedemos a realizar la modificación en default, ya que ahí se realiza el cambio en el proxy pass para dirigir cualquier puerto al 80.

```

root@Buscador:/home/TallerCinco# ls
__pycache__ main.py mook.py requirements.txt
root@Buscador:/home/TallerCinco# cd /etc/nginx/sites-available/
root@Buscador:/etc/nginx/sites-available# ls
default
root@Buscador:/etc/nginx/sites-available# vi default

```

Figura 25. Comando para configurar el proxy

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    location / {
        proxy_pass http://24.199.112.172
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

Figura 26. Vista interna del default

Recordemos que para guardar la información debemos escribir :wp. Ahora, de la siguiente manera comprobamos que el proxy este funcionando de manera correcta:

```
root@Buscador:/etc/nginx/sites-available# service nginx restart
```

Figura 27. Restablecimiento del servidor

18- Posteriormente se instala el pm2:

```
root@Buscador:/home/TallerCinco# npm install -pm2 -g
```

Figura 28. Instalación pm2

19- El siguiente comando funciona para hacer uso de herramientas con el fin que el servidor permanezca funcionando una vez se cerrada la consola.

Básicamente creo un proceso que es llamado como se desee, está corriendo en el root e inicie la aplicación en el servidor.

```
root@Buscador:/home/TallerCinco# pm2 start uvicorn --name "Mi Buscador" -- --k uvicorn.workers.UvicornWorker - "24.199.112.172:8002" "main.py"
```

Figura 29. Comando para iniciar aplicación sin necesidad de consola

El comando funciona de la siguiente mane, pm2 start (inicializar un proceso administrado por pm2), en el caso de uvicorn (Nos indica el nombre a la aplicación y se visualice en la ventana).

20- Posteriormente se crea un bash para ejecutar un comando y nos permita iniciar el servidor:

```
root@Buscador:/home/TallerCinco# vi prueba.sh
```

Figura 30. Comando para configurar el ejecutable ssh

Por último se coloca las siguientes líneas de código

```
#!/bin/bash
uvicorn main:app --host http://24.199.112.172 --port 8002
```

Figura 31. Código en el archivo ssh

Resumidamente se indica el siguiente comando para inicializar el servidor:

pm2 start prueba.sh

id	name	namespace	version	mode	pid	uptime	v	status	cpu	mem	user	watching
1	Mi Buscador	default	N/A	start	0	0	15	errored	0%	0%	root	01:00:10
0	Mi Buscador	default	N/A	start	0	0	15	errored	0%	0%	root	01:00:10
3	Mi Buscador	default	N/A	start	0	0	15	errored	0%	0%	root	01:00:10
4	prueba	default	N/A	start	463779	0%	0	online	0%	3.3m	root	01:00:10
2	run	default	N/A	start	12961	310	0	online	0%	2.3m	root	01:00:10

Figura 32. Bash online (prueba.sh)

21- Por último verificamos que el servidor se encuentra funcionando sin necesidad de la terminal en pycharm.

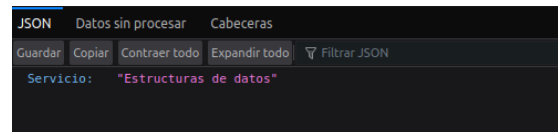


Figura 33. Microservicio en funcionamiento sin necesidad de la consola