Guía Completa para Crear, Entrenar y Desplegar un Proyecto de Reconocimiento de Piedra, Papel o Tijeras

1. Resumen del Proyecto

Este documento proporciona una guía paso a paso sobre cómo crear, entrenar y desplegar un modelo de reconocimiento de gestos de piedra, papel o tijeras. El proyecto incluye la creación del modelo utilizando PyTorch, el desarrollo de una aplicación web con Flask, y el despliegue de la aplicación utilizando Docker.

2. Configuración del Entorno

Antes de empezar, necesitas configurar un entorno de Python con las dependencias necesarias. Asegúrate de tener Python 3.9 o posterior instalado. Las dependencias principales son Flask, PyTorch, torchvision, Pillow, y Docker.

Usa el siguiente comando para instalar las dependencias:
```
pip install Flask torch torchvision Pillow
```

3. Creación y Entrenamiento del Modelo

3.1 Definir la Arquitectura del Modelo

El modelo es una red neuronal convolucional (CNN) simple con tres capas convolucionales, seguido de capas de agrupamiento y totalmente conectadas para la clasificación.

```python
import torch
from torch import nn

class RockPaperScissorsModel(nn.Module):
    def __init__(self):
```

```
        super(RockPaperScissorsModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, 1)
        self.conv2 = nn.Conv2d(16, 32, 3, 1)
        self.conv3 = nn.Conv2d(32, 64, 3, 1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64*77*77, 128)
        self.fc2 = nn.Linear(128, 3)
    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = self.pool(torch.relu(self.conv3(x)))
        x = x.view(-1, 64*77*77)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

3.2 Preparación del Conjunto de Datos

El conjunto de datos debe incluir imágenes de manos mostrando piedra, papel y tijeras. Cada imagen debe tener una etiqueta correspondiente que indique el gesto.

3.3 Entrenar el Modelo

Entrena el modelo utilizando el conjunto de datos. Es importante dividir el conjunto de datos en conjuntos de entrenamiento y validación.

```python
# Código de ejemplo para entrenar el modelo
model = RockPaperScissorsModel()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
for epoch in range(10):
    # Bucle de entrenamiento aquí
    pass
```

3.4 Guardar el Modelo

Una vez que el modelo esté entrenado, guarda el estado del modelo para usarlo posteriormente en la aplicación Flask.

```python
torch.save(model.state_dict(), 'rock_paper_scissors_model.pt')
```

```

## 4. Construcción de la Aplicación Flask

### 4.1 Código Final de la Aplicación Flask (`app.py`)

El siguiente código representa la aplicación Flask completa, que incluye la carga del modelo, el manejo de subidas de imágenes y la predicción del gesto:

```python
from flask import Flask, request, jsonify, render_template
import torch
from torchvision import transforms
from PIL import Image
from model import RockPaperScissorsModel, predict_image
import random

app = Flask(__name__)
model = RockPaperScissorsModel()
model.load_state_dict(torch.load('rock_paper_scissors_model.pt'))
model.eval()

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        if 'file' not in request.files:
            return 'No se ha enviado ningún archivo'
        file = request.files['file']
        if file.filename == '':
            return 'No se ha seleccionado ningún archivo'
        if file:
            user_choice = predict_image(file)
            computer_choice = random.choice(['rock', 'paper', 'scissors'])
            result = determine_winner(user_choice, computer_choice)
            return render_template('index.html', user_choice=user_choice,
computer_choice=computer_choice, result=result)
    return render_template('index.html')

def determine_winner(user, computer):
    if user == computer:
        return 'Empate'
    elif (user == 'rock' and computer == 'scissors') or \
         (user == 'scissors' and computer == 'paper') or \
```
```

```
        (user == 'paper' and computer == 'rock'):
        return 'Ganaste'
    else:
        return 'Perdiste'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

4.2 Requisitos Finales para el Despliegue en Docker

Asegúrate de que tu archivo `requirements.txt` contenga las siguientes dependencias para un despliegue exitoso:

```text
Flask==2.1.0
torch==2.4.0+cpu
torchvision==0.19.0+cpu
Pillow==8.4.0
```

También necesitarás un archivo `Dockerfile` como este:

```Dockerfile
FROM python:3.9-slim
WORKDIR /app
COPY . /app
RUN pip install --no-cache-dir -r requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```

5. Despliegue de la Aplicación

5.1 Configuración de un Entorno en la Nube

Puedes desplegar tu aplicación Flask Dockerizada en una plataforma en la nube, como Ploomber.io.

5.2 Configuración del Despliegue

Asegúrate de que todas las variables de entorno y configuraciones estén correctamente configuradas para el despliegue.

6. Orden de ejecucion de Scripts

6.1 Limpieza de dataset

Ejecutando el script clean_dataset.py se eliminann todas las imagenes que no correspondan a piedra, papel o tiejra y las que aparece mas de una persona ya que el modelo pretende reconocer una sola persona,

6.2 torch_model.py

Este script entrena el modelo con las imagenes y labels del dataset.

6.3 compress_model.py

Este script reduce el tamanio del modelo disminuyendo su precision y obteniendo un modelo mas liviano que facilita su despliegue en plataformas de nube. Despues de ejecutar este script se obtiene el modelo llamado compressed_model.pt

6.4 flask_api/app.py

Como se describe anteriormente, este es el script que detona la aplciacion flask.

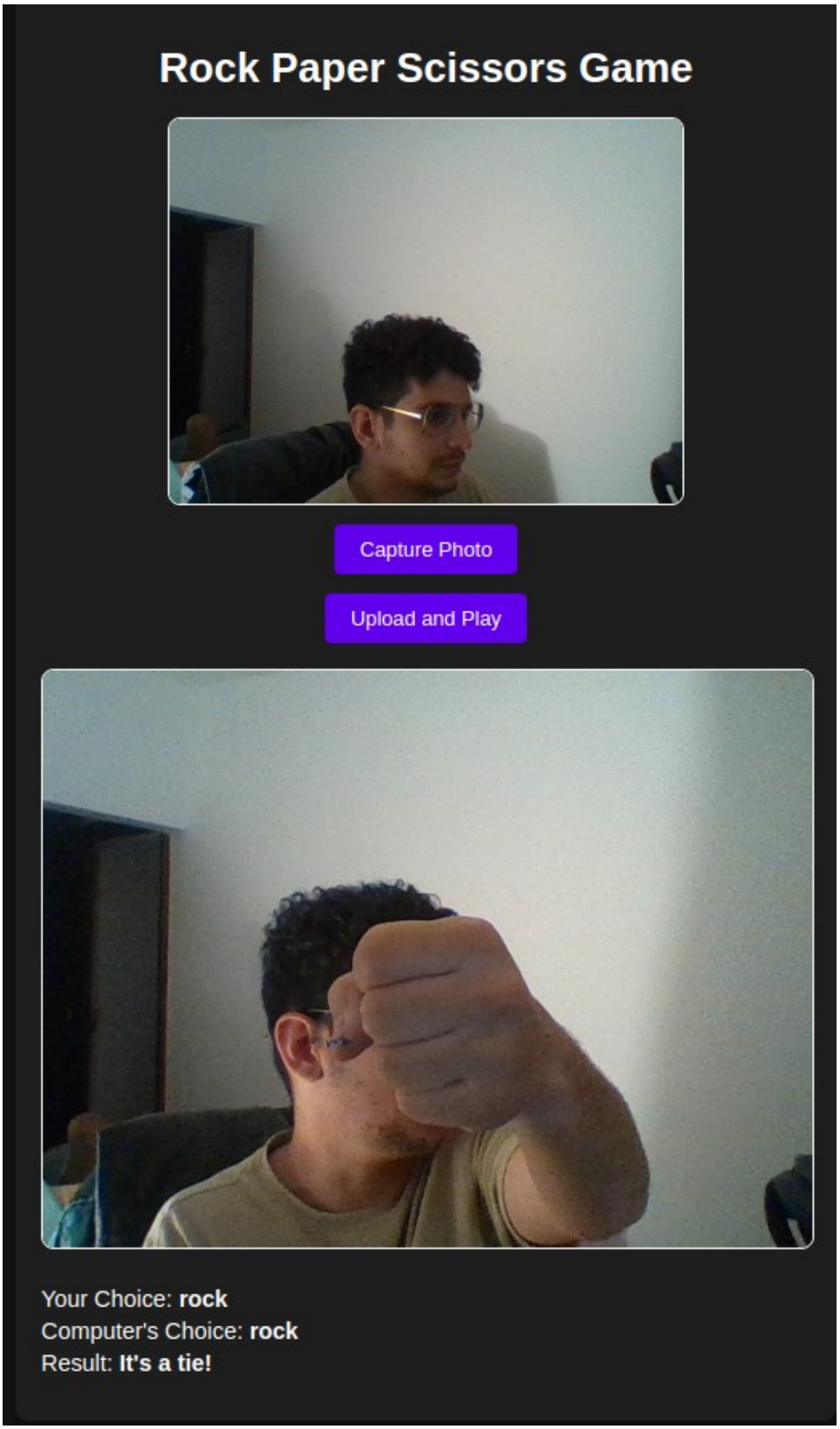
6.5 flask_api/model.py

Este script hace parte de la aplicacionm flask, se encarga de cargar el modelo y ejecutar la funcion mediante la que se predice la imagen capturada.

6.6 flask_api/templates/index.html

Este documento consiste en la estructura del frontend que se carga mediante la aplicacion flask, este documento contiene toda la logica para obtener imagenes en vivo de la camara web y realizar los llamados al endpoint de cargar imagen en donde se carga, se procesa y se determina si el usuario gana o pierde la partida contra la maquina.

7. Capturas de pantalla



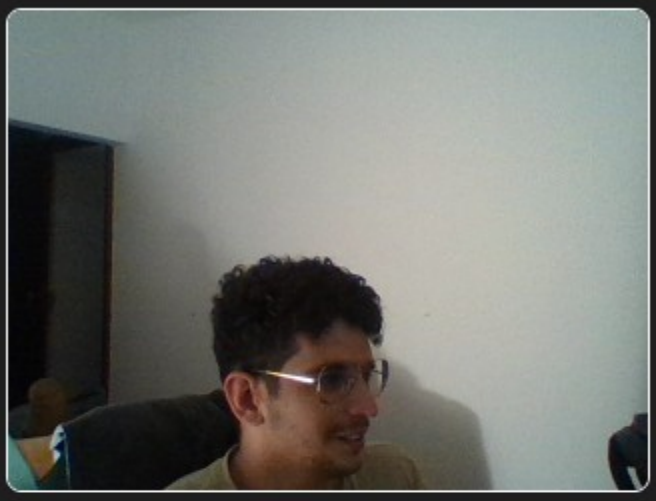
Rock Paper Scissors Game

Capture Photo

Upload and Play

Your Choice: **scissors**
Computer's Choice: **scissors**
Result: **It's a tie!**

Rock Paper Scissors Game



Capture Photo

Upload and Play

Your Choice: **paper**
Computer's Choice: **rock**
Result: **You win!**

8.Despliegue

Para el despliegue de la aplicación en la nube se utiliza la plataforma https://www.platform.ploomber.io la cual permite desplegar la aplicación flask e incluso una imagen docker sin necesidad de ningún pago.

8.1 comprimir archivos

El despliegue en ploomber requiere que todos los archivos a desplegar esten en una carpeta zip la cual se sube directamente a la plata forma.

8.2 requerimientos

Dentro de la carpeta .zip hay un archivo llamado requirements.txt, es el mismo archivo que se requiere al desplegar una imagen docker lo que evidencia que ploomber realmetne esta levantando una imagen docker con nuestra aplicación. Este archivo contiene las librerias que se deben isntalar de Python para que nuestra aplicación funcione. Se adjunta el contenido de dicho archivo.

Flask==2.1.0

Werkzeug==2.0.3

torch==2.4.0+cpu

torchvision==0.19.0+cpu

Pillow

numpy

8.3 Despleigue

El despliegue en ploomber requiere que todos los archivos a desplegar estén en una carpeta zip la cual se sube directamente a la plata forma, una vez allí, empieza a desplegarse la aplicación y después de desplegada se puede acceder a la sección de aplicaciones de ploomber donde podremos ver el link de acceso a la aplicación recién desplegada.

8.4 Capturas de pantalla del proceso de despliegue

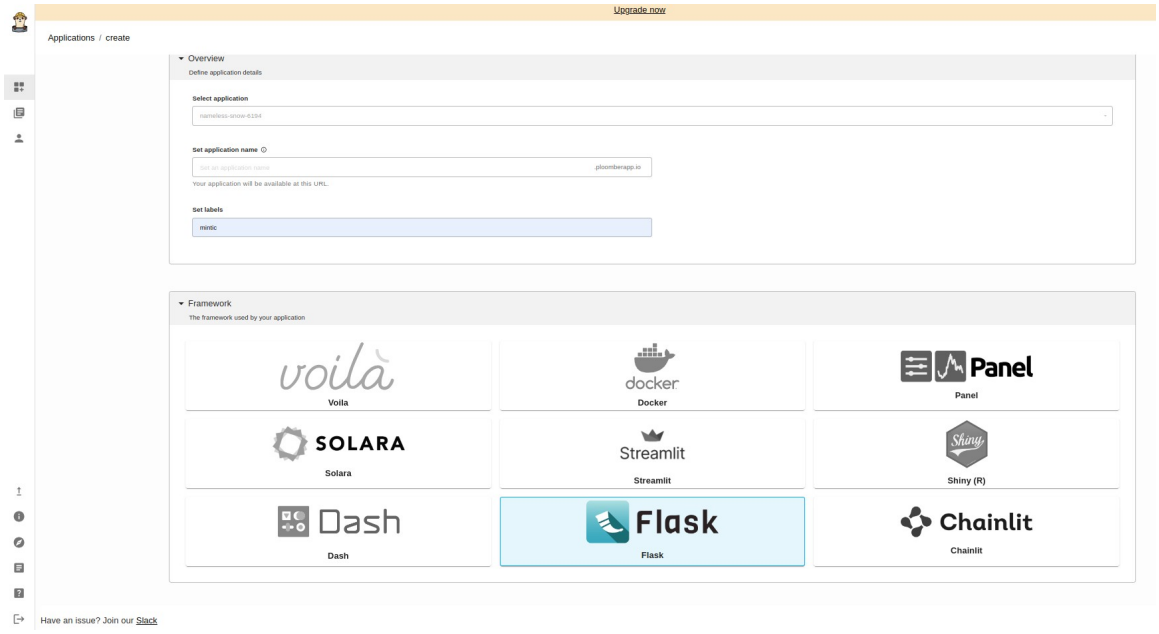


Imagen 1.

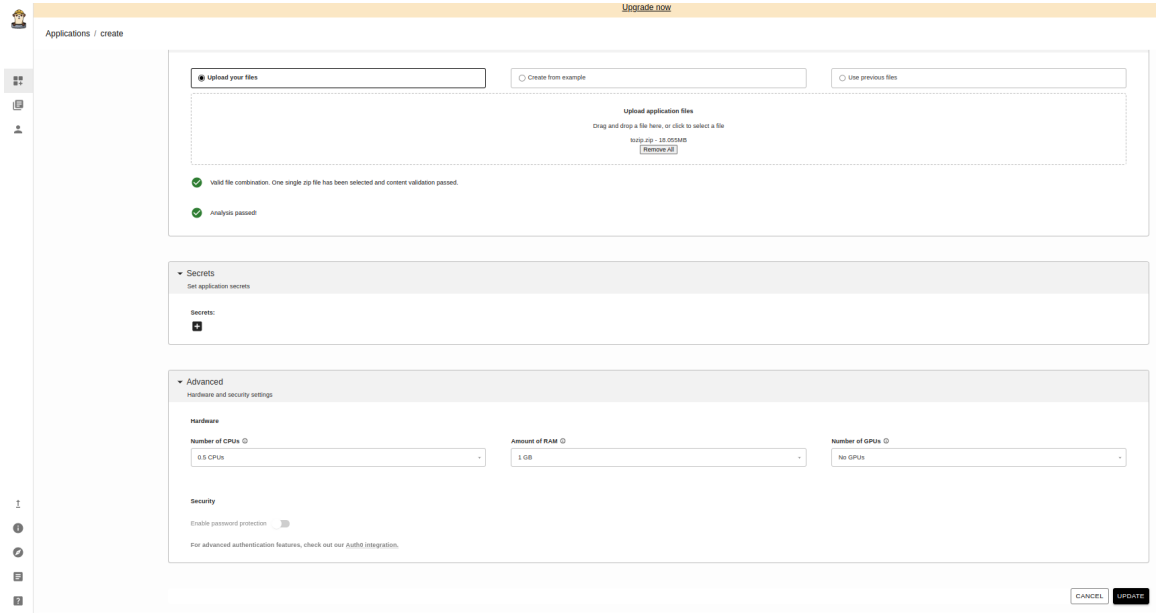


Imagen 2.

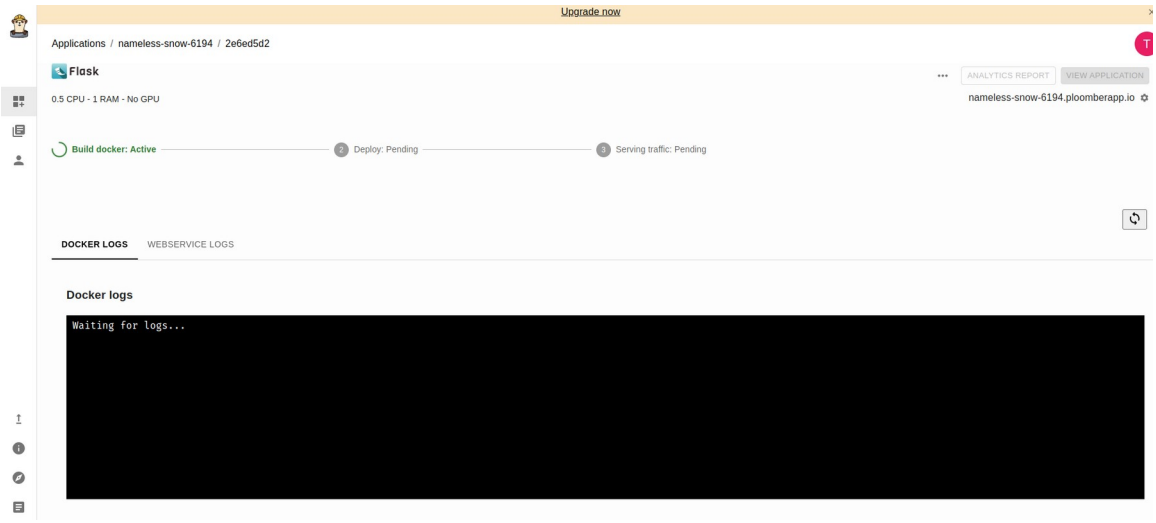


Imagen 3.

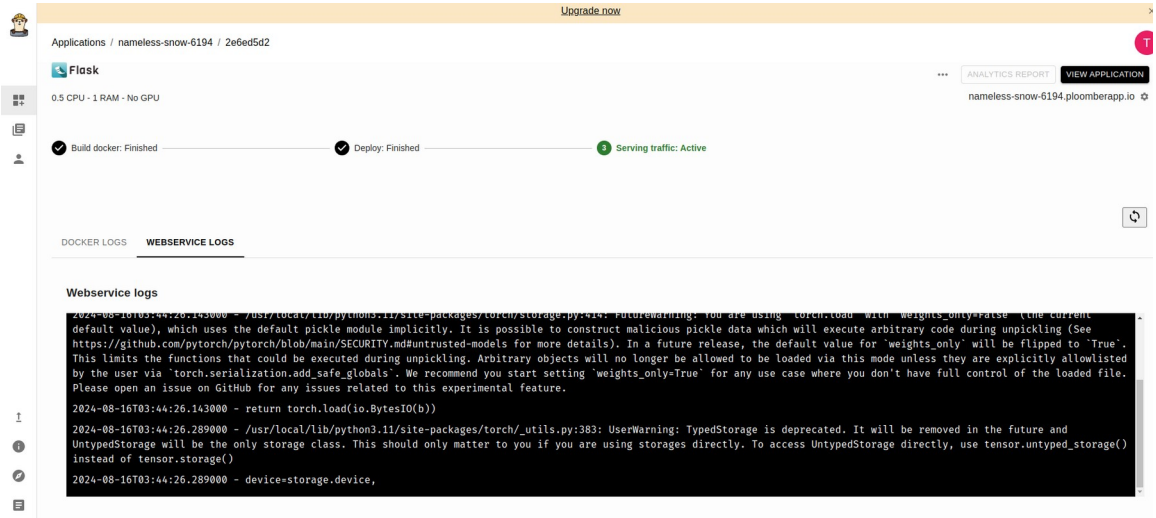


Imagen 4.

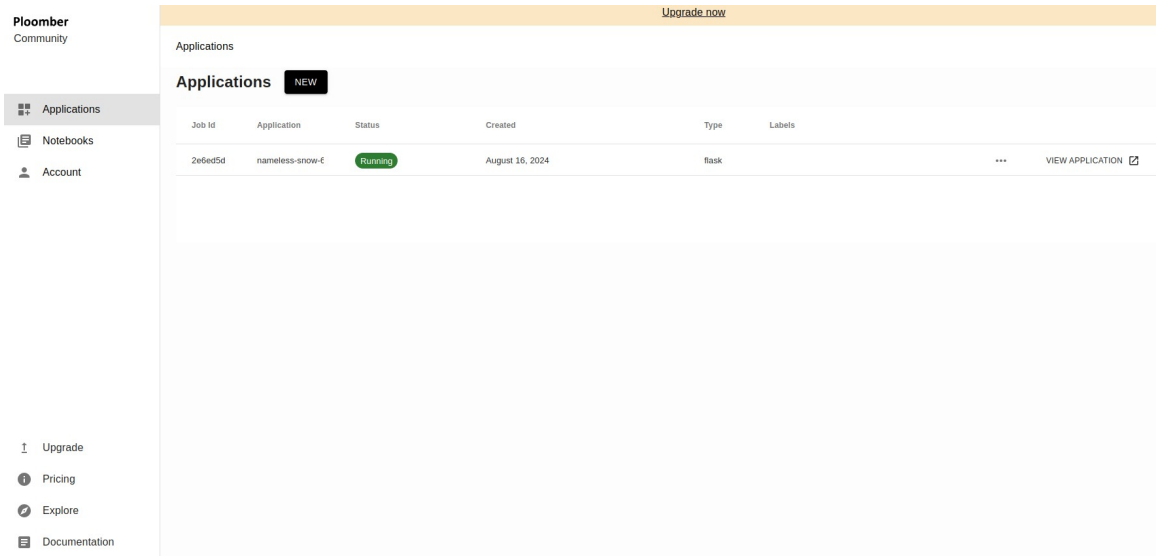


Imagen 5.

La imágenes 1 y 2 corresponden a una misma pantalla en donde se realiza la carga de los archivos de la aplicación y se especifica que es una aplicación flask, se especifican los recursos de la instancia (se especifica el mínimo de recursos para evitar cualquier restricción).

La imagen 3 corresponde al inicio del despliegue y la imagen 4 corresponde al despliegue finalizado. Entre la imagen 3 y 4 no se realiza ninguna acción.

En la imagen 5 se observa el panel de aplicaciones donde se observa la aplicación desplegada y el link donde se puede acceder.