ISC	Infraestructuras y Servicios Cloud
25/26	Redes Virtuales y Seguridad de Acceso
GIIA	Enunciado de la práctica 3

Versión 1: Conceptual y Tareas

Objetivos de la práctica

El objetivo de esta práctica es que los estudiantes de Ingeniería en Inteligencia Artificial comprendan y apliquen los conceptos de **redes seguras en la nube (VPC)** y **despliegue de aplicaciones multicapa** en Amazon Web Services (AWS).

Resultados de Aprendizaje

Al finalizar esta práctica, los estudiantes serán capaces de:

- **Diseñar y configurar** una arquitectura de red segura en AWS, utilizando subredes públicas y privadas, así como **Grupos de Seguridad** y **tablas de enrutamiento**.
- **Desplegar** un modelo de inteligencia artificial como un servicio accesible mediante una API REST.
- **Integrar** las diferentes capas de una aplicación (front-end y back-end) para que se comuniquen de forma segura en una red privada.
- **Demostrar** que los recursos críticos (el modelo de IA) están protegidos y solo son accesibles a través de una interfaz controlada (el front-end).

Descripción del proyecto

Se creará una aplicación de inferencia de IA en AWS con una arquitectura de dos capas. La aplicación consta de un **front-end web** y un **back-end** con un modelo de IA. El **back-end** y el modelo de IA se alojarán en una **subred privada** dentro de una VPC. El **front-end** se ubicará en una **subred pública**, actuando como el único punto de entrada para los usuarios.

La comunicación entre el front-end y el back-end se realizará a través de una llamada a una API interna, utilizando las **direcciones IP privadas** y las reglas del

Grupo de Seguridad. Para los datos del modelo, se utilizará un bucket de Amazon S3, al cual el back-end accederá de forma segura mediante un rol de IAM, sin necesidad de credenciales.

Tareas a realizar por los estudiantes

1. **Configurar la infraestructura de red:**
    - Crear una VPC con una subred pública y una privada.
    - Configurar un Internet Gateway para la subred pública y un NAT Gateway para la privada.
2. **Preparar el entorno de seguridad y almacenamiento:**
    - Crear un bucket de S3 con dos carpetas (`front-end` y `back-end`).
    - Crear dos grupos de seguridad, uno para el front-end (acceso público) y otro para el back-end (acceso restringido).
    - Crear un política de S3 que permita el acceso desde la red pública.
3. **Preparar los archivos de la aplicación y el modelo:**
    - Entrenar un modelo de IA simple y guardar el archivo `.pkl`.
    - Subir el modelo y los archivos de la aplicación a sus respectivas carpetas en S3.
4. **Desplegar los servicios de la aplicación:**
    - Lanzar una instancia EC2 con el Linux de AWS en la subred pública para el front-end.
    - Lanzar una instancia EC2 con el Linux de AWS en la subred privada para el back-end.
    - Usar scripts de **User Data** para automatizar la instalación de dependencias y la ejecución de los servidores.
5. **Verificar la seguridad y funcionalidad:**
    - Confirmar que el servidor del back-end no es accesible desde internet.
    - Verificar que la página web del front-end es accesible públicamente.
    - Probar la aplicación para confirmar que la comunicación interna entre el front-end y el back-end funciona correctamente.

Código y archivos necesarios

Los estudiantes necesitarán los siguientes archivos para completar la práctica.

1. Código para entrenar el modelo (Python)

Este script genera el archivo `modelo.pkl` que deben subir a S3.

Python

```
# Archivo: entrenar_modelo.py
import joblib
import numpy as np
from sklearn.linear_model import LinearRegression

# Crear un conjunto de datos de ejemplo
x_train = np.array([[1], [2], [3], [4], [5], [6], [7], [8], [9], [10]])
y_train = np.array([2.1, 4.2, 5.8, 8.1, 10.3, 11.9, 14.2, 16.0, 18.2, 20.5])

# Entrenar el modelo y guardarlo
model = LinearRegression()
model.fit(x_train, y_train)
joblib.dump(model, 'modelo.pkl')

print("Modelo entrenado y guardado como 'modelo.pkl'.")
```

---

2. Código del back-end (Python/Flask)

Este servidor se ejecuta en la instancia de la subred privada.

Python

```
# Archivo: app_backend.py
from flask import Flask, request, jsonify
import joblib
import os
import boto3

app = Flask(__name__)
S3_BUCKET = 'tu-nombre-de-bucket-unico' # ¡Cambia esto!
S3_MODEL_KEY = 'backend/modelo.pkl'
```

```
MODEL_LOCAL_PATH = 'modelo.pkl'

def download_model_from_s3():
    try:
        s3 = boto3.client('s3')
        s3.download_file(S3_BUCKET, S3_MODEL_KEY, MODEL_LOCAL_PATH)
        return True
    except Exception as e:
        print(f"Error descargando el modelo: {e}")
        return False

model_downloaded = download_model_from_s3()
model = None
if model_downloaded:
    model = joblib.load(MODEL_LOCAL_PATH)

@app.route('/predict', methods=['POST'])
def predict():
    if not model:
        return jsonify({"error": "Modelo no disponible"}), 500
    try:
        data = request.get_json(force=True)
        features = data['features']
        prediction = model.predict([features])
        return jsonify({"prediction": prediction.tolist()[0]})
    except Exception as e:
        return jsonify({"error": str(e)}), 400

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

---

3. Código del front-end (Node.js/Express)

Este servidor se ejecuta en la instancia de la subred pública.

JavaScript

```
// Archivo: app_front.js
const express = require('express');
```

```
const path = require('path');
const axios = require('axios');

const app = express();
const PORT = 80;

// ¡El estudiante debe reemplazar esto con la IP privada de su instancia de back-end!
const BACKEND_API_URL = 'http://IP_PRIVADA_DEL_BACKEND:5000/predict';

app.use(express.static(path.join(__dirname, 'public')));
app.use(express.json());

app.post('/api/predict', async (req, res) => {
    try {
        const response = await axios.post(BACKEND_API_URL, req.body);
        res.json(response.data);
    } catch (error) {
        console.error('Error al conectar con el backend:', error.message);
        res.status(500).json({ error: 'No se pudo conectar con el servicio de IA.' });
    }
});

app.listen(PORT, () => {
    console.log(`Servidor front-end escuchando en el puerto ${PORT}`);
});
```

---

4. Interfaz de usuario (HTML/JavaScript)

La página web que interactúa con el front-end.

HTML

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Inferencia de IA</title>
</head>
<body>
```

```
    <h1>Inferencia de modelo de IA segura</h1>
    <p>Ingresa una característica para que el modelo en la red privada haga una predicción:</p>
    <input type="number" id="featureInput" placeholder="Ingresa un número">
    <button onclick="predict()">Predecir</button>
    <h3>Resultado:</h3>
    <p id="result">El resultado aparecerá aquí.</p>

    <script>
        async function predict() {
            const feature = document.getElementById('featureInput').value;
            const resultElement = document.getElementById('result');
            resultElement.innerText = "Cargando...";

            try {
                const response = await fetch('/api/predict', {
                    method: 'POST',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify({ features: [parseFloat(feature)] })
                });
                const data = await response.json();
                resultElement.innerText = `Predicción: ${data.prediction.toFixed(2)}`;
            } catch (error) {
                resultElement.innerText = 'Error al conectar con la API del front-end.';
            }
        }
    </script>
</body>
</html>
```