ISC	Infraestructuras y Servicios Cloud
25/26	Redes y Conectividad Avanzada y Contenedores
GIIA	**Enunciado de la práctica 4**

Práctica 4: Despliegue de Modelos de IA con Seguridad en Capas (Canary Deployment)

El Desafío de MLOps: El Motor de Recomendaciones

¡Bienvenidos a la práctica de sistemas de producción! En nuestro *e-commerce*, "RecomiendaMás", acabamos de terminar el desarrollo del **Modelo de Recomendaciones V2**. Este modelo es genial, pero **no podemos desplegarlo directamente** al 100% de nuestros clientes; sería catastrófico si fallara.

Nuestro reto es implementar un **Despliegue Canario (Canary Deployment)**, enviando solo el **10%** del tráfico al nuevo modelo, mientras el **90%** se mantiene seguro en el Modelo V1 de producción. Además, debemos construir la arquitectura con **seguridad por capas (Defensa en Profundidad)**, incluyendo GSS, NACLs, y WAF, para proteger nuestra VPC.

Objetivos de Aprendizaje

Al finalizar esta práctica, dominarás:

1. **Segmentación de Red:** Diferenciar y aislar recursos con **Subredes Públicas y Privadas**.
2. **Seguridad por Capas:** Implementar **Grupos de Seguridad (GSS)** y **NACLs** y entender cuándo usar cada uno.

3. **Ruteo Avanzado:** Configurar un **Balanceador de Carga L7** para el ruteo basado en peso (90/10).
4. **Seguridad de Datos:** Habilitar el acceso seguro a **S3** desde la red privada.
5. **Transición Arquitectónica:** Migrar el *Canary* V2 de una VM a un **Contenedor Docker**.

I. Código Base de la API

Ambos modelos son APIs de Flask que se ejecutan internamente en el puerto **8000**.

A. Modelo V1 (Producción - model_v1_prod.py)

Python

```python
from flask import Flask, jsonify, request
import time
app = Flask(__name__)
MODEL_ID = "V1 - Stable"

@app.route('/api/v1/recommendation', methods=['POST'])
def recommend_v1():
    start_time = time.time()
    data = request.get_json()
    user_id = data.get('user_id', 'N/A')
    recommendations = ["item_X_old_model", "item_Y_old_model"]
    latency = round((time.time() - start_time) * 1000)

    return jsonify({
        "event_log": f"User {user_id} purchase processed.",
        "recommendation": recommendations,
        "model_id": MODEL_ID,
        "latency_ms": latency
    }), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000)
```

B. Modelo V2 (Canary - model_v2_canary.py)

Python

```
from flask import Flask, jsonify, request
import time
app = Flask(__name__)
MODEL_ID = "V2 - DeepLearning - Canary"

@app.route('/api/v1/recommendation', methods=['POST'])
def recommend_v2():
    start_time = time.time()
    data = request.get_json()
    user_id = data.get('user_id', 'N/A')
    time.sleep(0.05) # Simula latencia mayor
    recommendations = ["item_A_new", "item_B_new", "item_C_new"]
    latency = round((time.time() - start_time) * 1000)
    print(f"[LOG] USER:{user_id}, MODEL:{MODEL_ID}, LATENCY:{latency}ms")

    return jsonify({
        "event_log": f"User {user_id} purchase processed. Metrics available.",
        "recommendation": recommendations,
        "model_id": MODEL_ID,
        "latency_ms": latency
    }), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000)
```

II. Fase 1: Redes y Seguridad de Perímetro

1. Configuración de Red (VPC Dashboard)

Acción	Configuración Exacta	Justificación
VPC y Subredes	Crear VPC (10.0.0.0/16). Subred **PÚBLICA**	**Segmentación:** La VPC es el límite de la red. La

Acción	Configuración Exacta	Justificación
	(10.0.10.0/24). Subred **PRIVADA** (10.0.20.0/24).	Subred PRIVADA aísla los modelos sensibles.
Enrutamiento Público	Crear **IGW**. En la Tabla de Ruteo PÚBLICA, añadir ruta 0.0.0.0/0 IGW.	Permite que el ALB reciba tráfico de Internet.
Enrutamiento Privado	Crear **NGW** en la Subred PÚBLICA (asignando EIP). En la Tabla de Ruteo PRIVADA, añadir ruta 0.0.0.0/0 NGW.	Permite que las VMs privadas accedan a internet (para instalación de software) sin recibir tráfico externo.

2. Integración del VPC Endpoint (Acceso a S3)

Concepto Clave: Se asume que el VPC Endpoint de tipo Gateway para S3 ya existe en el Lab. Esta ruta privada anula el NAT Gateway para el tráfico a S3, que es la mejor práctica de seguridad y eficiencia.

- **Acción de Verificación:** Ir a **VPC Dashboard** **Endpoints**. Obtener el **ID del VPC Endpoint** (vpce-xxxxxxxxxx).
- **Acción de Enrutamiento: Verificar** que la Tabla de Ruteo PRIVADA ya tiene una ruta específica para los prefijos de S3 apuntando al ID de este Endpoint.

3. Seguridad por Capas (NACLs y GSS)

Componente	Reglas de Entrada Clave	Justificación
NACL-PÚBLICA	**Entrada:** HTTP (80) desde 0.0.0.0/0. **Salida:** Puertos Efímeros (1024-65535) a 0.0.0.0/0.	**Primera capa (Stateless).** Filtra el tráfico de perímetro y permite el tráfico de respuesta para la navegación web.
NACL-PRIVADA	**Entrada:** TCP (8000) desde CIDR de la Subred PÚBLICA (10.0.10.0/24). **Salida:** Puertos Efímeros (1024-65535) al CIDR de la Subred PÚBLICA.	**Segunda capa (Stateless).** Solo permite el tráfico que viene del ALB y permite que la respuesta de la API regrese.
SG-ALB	HTTP (80) desde 0.0.0.0/0. SSH (22) desde 0.0.0.0/0.	**Firewall con estado (Stateful) público.**
SG-BACKEND	Custom TCP (8000) desde **SG-ALB**.	**Firewall con estado (Stateful) de instancia.** Solo permite el tráfico desde el ALB.

III. Despliegue de Servicios y Ruteo Canary

1. Scripts de Inicialización (User Data para todas las VMs Ubuntu)

Instrucción: Reemplaza los valores de las claves y la región en el script.

Bash

```
#!/bin/bash
# Script de User Data para Ubuntu (Instalación Completa de Runtimes y Descarga de S3)

# 1. Instalación de Herramientas de Sistema
sudo apt-get update -y
```

```
sudo apt-get install -y python3-pip python3-venv git nodejs npm
pip3 install flask joblib
pip3 install awscli

# 2. Configurar Credenciales AWS CLI (Obligatorio para autenticar la descarga)
# Reemplaza TUS_CLAVES y tu-region
sudo aws configure set aws_access_key_id TU_CLAVE_DE_ACCESO_ID
sudo aws configure set aws_secret_access_key TU_CLAVE_SECRETA_COMPLETA
sudo aws configure set default.region tu-region
sudo aws configure set default.output json

# 3. Descargar Artefactos de S3 (Se accede por el VPC Endpoint)
mkdir -p /home/ubuntu/app
cd /home/ubuntu/app

# Usamos SUDO HOME=/home/ubuntu para que la CLI encuentre las credenciales
sudo HOME=/home/ubuntu aws s3 cp s3://tu-nombre-de-bucket-
unico/backend/model_v1_prod.py .
sudo HOME=/home/ubuntu aws s3 cp s3://tu-nombre-de-bucket-
unico/backend/model_v2_canary.py .
sudo HOME=/home/ubuntu aws s3 cp s3://tu-nombre-de-bucket-unico/backend/modelo.pkl .

# 4. Asignar propiedad a los archivos
sudo chown -R ubuntu:ubuntu /home/ubuntu/app

# 5. Ejecución (Requiere conexión SSH posterior del estudiante)
# Los estudiantes deben conectarse por SSH y ejecutar el modelo correcto:
# Para V1: python3 model_v1_prod.py > /dev/null 2>&1 &
# Para V2: python3 model_v2_canary.py > /dev/null 2>&1 &
```

2. Configuración del Balanceador de Carga (ALB)

Paso	Acción en la Consola AWS	Concepto Clave y Justificación
4.1 Crear TGs	Crear **TG-V1** y **TG-V2** (Puerto 8000). Registrar	**Target Groups:** Agrupan los destinos para que el

Paso	Acción en la Consola AWS	Concepto Clave y Justificación
	las 3 VMs V1 y la 1 VM V2 con sus **IPs privadas**.	Balanceador de Carga pueda dirigirse a ellos.
4.2 Implementación del Ruteo 90/10	**Listener puerto 80**: Configurar la regla para **Ruteo Basado en Peso**: **90% a TG-V1** y **10% a TG-V2**.	**Canary Deployment (L7):** El Balanceador de Carga aplica la lógica de negocio del porcentaje al tráfico entrante.

V. Fase 3: Transición a Contenedores

Concepto Clave: Evolución. Introducimos Docker como una mejora del *Canary* para demostrar la portabilidad y la gestión de la red de contenedores.

Paso	Acción en la Consola AWS	Explicación Conceptual
3.1 Preparar Docker Host	Lanzar una nueva **VM *Host*** en la **Subred PRIVADA**, instalar Docker.	Este es el servidor que alojará el contenedor V2. Necesita tener una **ruta interna** para ser accesible desde el ALB.
3.2 Ejecutar el Contenedor	Ejecutar el contenedor V2 **mapeando el puerto 80 del *host***: `docker run -d -p 80:8000 model_v2_canary_image`.	El *host* de la VM ahora escucha en el puerto 80. La red de Docker (con su componente **iptables**) se encarga de dirigir el tráfico al puerto 8000 del contenedor.
3.3 Ajuste de GSS y Re-Ruteo	Crear `TG-V2-Docker` (Puerto **80**) y registrar la **IP privada del *Host***.	El ALB ahora habla en el puerto **80** con la nueva VM. La regla

Paso	Acción en la Consola AWS	Explicación Conceptual
	Modificar el **SG-BACKEND** para permitir el Puerto **80** desde el **SG-ALB**.	del ALB se ajusta: **10% del tráfico** al `TG-V2-Docker`.

Preguntas de Reflexión para el Informe

- **Seguridad y NACLs:** Si el **SG-BACKEND** ya bloquea el tráfico no deseado al puerto 8000, ¿qué valor de **seguridad adicional** aporta la **NACL**? ¿Podría el atacante interceptar la respuesta de la API usando el NACL? ¿Por qué?
- **WAF vs. NACL:** Usted usó ambos. ¿Qué servicio detendría un ataque de **Inyección de Código SQL** (WAF o NACL) y por qué?
- **VPC Endpoints:** Explique cómo el **VPC Endpoint** permite que el tráfico de *logging* del **Modelo V2** nunca pase por el **Internet Gateway (IGW)** de su VPC.
- **Redes de Contenedores:** ¿Qué **IP y Puerto** de la **VM *Host*** debe usar el **Balanceador de Carga** para dirigir el tráfico al Modelo V2 dentro del contenedor? ¿Cómo se llama el **componente de red de Docker** que realiza la traducción del puerto (80 al 8000)?
- ***Rollback* Rápido:** En el contexto del *Canary Deployment*, ¿por qué el uso del **Contenedor Docker** permite una **reversión** a la versión V1 mucho más rápida y segura que si el V2 se hubiera quedado en una VM tradicional?