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**.

Recursos y Código Base de la API

El código es una API sencilla de Flask que simula la recomendación y se ejecutará internamente en el puerto **8000**.

1. Modelo V1 (Producción - `model_v1_prod.py`)

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():
    # Simula la lógica de recomendación
    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)
```

2. Modelo V2 (Canary - `model_v2_canary.py`)

(Nota: Este código simula la escritura en S3 y tiene una latencia ligeramente mayor).

Python

```
from flask import Flask, jsonify, request
import time, os

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)

    # *** Simulación: Escribir métricas en S3 (Requiere VPC Endpoint/IAM)
***
    log_data = f"[LOG_S3] USER:{user_id}, LATENCY:{latency}ms"
    print(log_data)

    return jsonify({
        "event_log": f"User {user_id} purchase processed. Metrics logged to
S3.",
        "recommendation": recommendations,
        "model_id": MODEL_ID,
        "latency_ms": latency
    }), 200

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

Fase 1: Redes, AMI y Seguridad NACL

Concepto Clave: Aislamiento. Nuestros servidores de modelos (Back-end) nunca deben ser accesibles directamente. Usaremos **NACLs** como la primera línea de defensa a nivel de subred.

Paso	Acción en la Consola AWS (VPC/EC2)	Explicación Conceptual
1.1 Infraestructura	Crear VPC (10.0.0.0/16), Subred PÚBLICA (10.0.10.0/24), Subred PRIVADA (10.0.20.0/24), e IGW.	La Subred PÚBLICA tiene la ruta al IGW. La PRIVADA no la tiene, aislando el Back-end.
1.2 NACLs (Seguridad Sin Estado)	Crear NACL-PÚBLICA y NACL-PRIVADA.	Actúan como un *firewall de subred*. Regla Clave: La NACL PRIVADA solo permite tráfico entrante en el puerto 8000 desde el CIDR de la Subred PÚBLICA (10.0.10.0/24).
1.3 AMI y Despliegue	Crear la AMI base. Lanzar 4 VMs (3 V1, 1 V2) con esta AMI.	La AMI estandariza el *runtime* (Python/Flask). Todas las VMs van a la Subred PRIVADA con IP Pública DESHABILITADA.
1.4 GSS (Seguridad con Estado)	Crear SG-ALB (Puerto 80 desde 0.0.0.0/0) y SG-BACKEND.	El SG-BACKEND permite la entrada en el puerto 8000 solo desde el SG-ALB. Esto complementa a la NACL, actuando como un *firewall con estado* a nivel de instancia.

Fase 2: Ruteo Canary (L7), WAF y S3 Seguro

Concepto Clave: Control. El ALB controla el tráfico y el WAF lo filtra, mientras que el VPC Endpoint garantiza que el *logging* de V2 sea seguro.

Paso	Acción en la Consola AWS	Explicación Conceptual
2.1 Implementación del ALB	Crear **ALB** en la **Subred PÚBLICA**, asociado al **SG-ALB**. Crear *Listener* en el puerto **80 (HTTP)**.	El ALB es el único componente público. Usamos HTTP para simplificar el *lab*.
2.2 Ruteo 90/10	Crear `TG-V1` y `TG-V2` (Puerto 8000). Establecer el **Ruteo Basado en Peso**: **90% a TG-V1 / 10% a TG-V2**.	El Balanceador L7 es el único que puede aplicar lógica de negocio (porcentaje) al tráfico.
2.3 WAF (Rate Limiting)	Crear **Web ACL**, asociarla al ALB. Implementar una regla de **Rate Limiting** estricta.	El **WAF** es el *firewall de la capa de aplicación*. Protege al ALB y al *Canary* V2 de ataques de *scripts* o sobrecargas de peticiones.
2.4 VPC Endpoint S3	Crear un **VPC Endpoint de Gateway** para S3. Asociarlo a la **Tabla de Ruteo PRIVADA**.	Permite que el código V2 (en la Subred Privada) escriba métricas a S3 **sin necesidad de pasar por el IGW**, manteniendo el aislamiento de la red.

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

Paso	Acción en la Consola AWS	Explicación Conceptual
		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***. Modificar el **SG-BACKEND** para permitir el Puerto **80** desde el **SG-ALB**.	El ALB ahora habla en el puerto **80** con la nueva VM. La regla 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?