Universidad Nacional de IngenieríaFacultad de Ciencias
Escuela Profesional de Ciencia de la Computación

# Examen Final (Práctica)

## Administración de Redes - CC312

**Fecha: 20/12/2025** **Duración: 100 minutos** **Ciclo: 2025-II**

Apellidos: _____ Nombres: _____

**EJERCICIO I (5 puntos)**

Trabaja como ingeniero de automatización. Se ha detectado que una interfaz crítica en el router CSR1000v necesita ser auditado y re-etiquetado debido a un cambio en la topología de la red.

Su tarea consiste en crear un script en Python llamado examen_auto.py que realice una **operación híbrida**: leer el estado actual con un protocolo y modificarlo con el otro.

**Datos de Conexión**

*Asumiremos el uso del Sandbox Always-On de Cisco DevNet (IOS-XE).*

- **Host:** sandbox-iosxe-latest-1.cisco.com
- **Usuario:** developer
- **Contraseña:** C1sco12345
- **Puerto NETCONF:** 830
- **Puerto RESTCONF (HTTPS):** 443
- **Interfaz Objetivo:** GigabitEthernet2

**Requerimientos del Script**

**Parte 1: Auditoría con NETCONF (Lectura)**

Utilizando la librería ncclient:

1. Conéctate al dispositivo.
2. Utiliza un **Filtro XML** para obtener únicamente la configuración de la interfaz GigabitEthernet2 (usando el modelo YANG ietf-interfaces o Cisco-IOS-XE-native).
3. Imprime en consola el XML crudo o parseado de la configuración actual de esa interfaz (se debe ver la IP o el estado).

**Parte 2: Configuración con RESTCONF (Escritura)**

Utilizando la librería requests:

1. Construye la URL adecuada para apuntar a la interfaz GigabitEthernet2.
2. Envía una solicitud (PATCH o PUT) para cambiar la **descripción** de la interfaz.
3. **Nueva Descripción:** "ENLACE_CRITICO_EXAMEN_AUTO"
4. Imprime en consola el código de estado HTTP de la respuesta (debe ser 204 o 200).

**Parte 3: Validación (Manejo de Errores)**

1. El script debe manejar excepciones básicas (ej. si falla la conexión NETCONF o si RESTCONF devuelve un error 400/401/404).

**Criterios de Evaluación**

1. **Conexión NETCONF exitosa (1 pt):** Uso correcto de manager.connect.
2. **Filtro XML correcto (1.5 pts):** El script no descarga "toda" la configuración, solo la interfaz solicitada.
3. **Headers y URL RESTCONF (1.5 pts):** Headers correctos (Content-Type, Accept) y URL bien formada.
4. **Payload JSON correcto (0.5 pts):** Estructura JSON válida para el cambio.
5. **Limpieza y Ejecución (0.5 pts):** El código corre sin errores de sintaxis y muestra la salida solicitada.

**EJERCICIO 2 (5 puntos)**

Usted es el arquitecto de una Fintech. Debe desplegar un microservicio crítico de "Validación de transacciones". Por normativas de seguridad (PCI-DSS), el contenedor **no puede correr como root** y la aplicación debe ser capaz de **reiniciarse automáticamente** si se bloquea.

Su misión consta de 4 pasos:

**1. Dockerfile Seguro (1.5 pts)**

Cree un Dockerfile para la aplicación (requiere flask).

- **Restricción de Seguridad (Crucial):** El contenedor **NO** debe correr como usuario root. Debe crear un usuario específico en el Dockerfile y usarlo.
- **Optimización:** Use una imagen base slim o alpine.
- Construya la imagen con el tag: fintech-validator:v1.

**2. Secreto de Kubernetes (1 pt)**

No queme las credenciales en el YAML.

- Cree un objeto **Secret** en Kubernetes llamado app-secrets.
- Debe contener la clave api-key con el valor "SUPER_SECRET_TOKEN_123".

**3. Deployment de Alta Disponibilidad (1.5 pts)**

Cree un archivo deployment.yaml con las siguientes especificaciones:

- **Replicas:** 3 instancias.
- **Inyección de Configuración:** La variable de entorno TRANS_API_KEY del contenedor debe tomar su valor del Secret creado en el paso 2.
- **Self-Healing (Liveness Probe):** Configure una sonda que verifique el endpoint /health en el puerto 3000.
  - *InitialDelay:* 15 segundos (para dar tiempo al inicio).
  - *Period:* 5 segundos.

**4. Exposición del Servicio (1 pt)**

Exponga el Deployment mediante un **Service** de tipo NodePort o LoadBalancer que mapee el puerto 80 del servicio al puerto 3000 del contenedor.

**Entregables**

1. Contenido del Dockerfile.
2. Comando usado para crear el Secret.
3. Archivo deployment.yaml (puede incluir el Service o estar separado).

**Tip:**

Recuerde que puede generar la estructura base del YAML usando:

```
kubectl create deployment fintech --image=fintech-validator:v1 --replicas=3
--dry-run=client -o yaml > deploy.yaml
```