Universidad Politécnica de

Chiapas

Ingeniería en Software

C3 - A2 - Monitoreo de recursos

Materia: Sistemas Operativos
Profesor: MC. José Alonso Macías Montoya
Corte: Corte 3

Alumno: Gael Hueytlelt Villalobos
Matrícula: 233392

Tuxtla Gutiérrez, Chiapas, a 11 de noviembre de 2025

Índice

1. Introducción

El objetivo de esta actividad es diseñar e implementar un sistema de monitoreo de recursos (CPU, RAM y Almacenamiento) en una instancia EC2 con Ubuntu 24.04, utilizando un script BASH que notifique al usuario vía Telegram cuando se superen umbrales de uso predefinidos. Este ejercicio práctico busca consolidar conocimientos en la administración de sistemas Linux, programación en BASH y el uso de APIs para la gestión de alertas, simulando un entorno real de administración de infraestructura en la nube.

2. Requisitos y Metodología

La actividad se desarrolló siguiendo los requisitos provistos por el docente y se implementó una metodología de tres fases: Configuración del Entorno, Desarrollo del Script de Monitoreo y Pruebas Funcionales.

2.1. Requisitos Específicos

- La actividad fue desarrollada por el usuario **gael** (primer nombre del alumno), no el usuario por defecto (`ubuntu`).
- Se creó un script BASH para analizar `df` (disco), `free` (RAM) y la carga de núcleo.
- Los límites de alerta (umbral) se definieron mediante variables de entorno dentro del script.
- La alerta se envía a un bot de Telegram al superar el límite y **no se envía** si el uso es normal.

2.2. Recursos Utilizados

- **Instancia AWS:** EC2 t2.micro con Ubuntu Server 24.04.
- **Usuario:** `gael` (Creado y con permisos sudo).
- **Herramientas Linux:** `df`, `free`, `uptime`, `curl` (para API de Telegram) y `bc` (para manejar flotantes en carga de CPU).
- **Sistema de Alerta:** Bot de Telegram configurado con `BotFather`.

3. Desarrollo e Implementación del Script

3.1. Configuración del Acceso SSH y Telegram

Se configuró el acceso SSH para el nuevo usuario `gael`, asegurando que la clave privada `.pem` estuviera en el archivo `/home/gael/.ssh/authorized_keys`. Además, se obtuvieron el `TELEGRAM_BOT_TOKEN` y el `TELEGRAM_CHAT_ID` (Ver Figura 1).

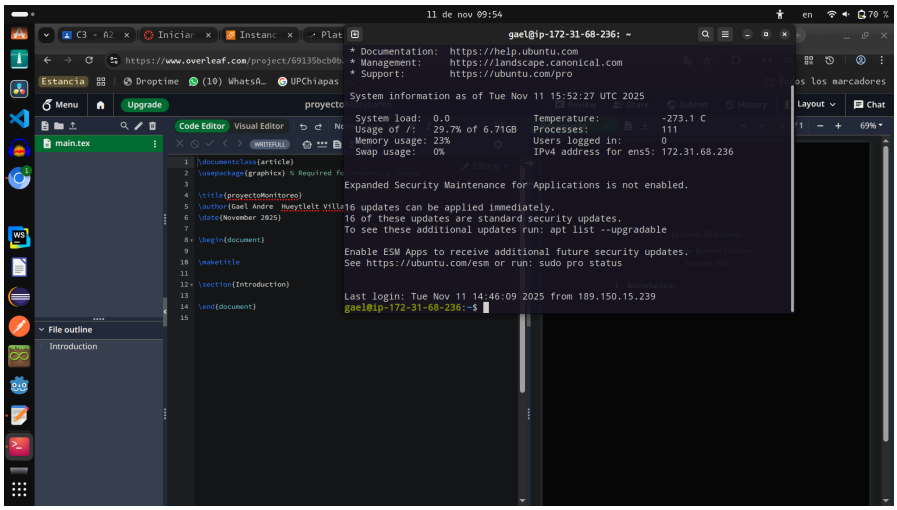

Figura 1: Conexión exitosa a la instancia EC2 con el usuario `gael`.

3.2. Código Fuente del Script (`monitor.sh`)

El script BASH se diseñó para encapsular las variables de límite y la lógica de verificación en una sola ejecución. La función `send_alert` utiliza `curl` para interactuar con la API de Telegram.

Listado 1: Código del Script de Monitoreo (`monitor.sh`)

```
#!/bin/bash

# ================================================================
# 1. DEFINICI N DE VARIABLES DE ENTORNO Y L MITES
# ================================================================
TELEGRAM_BOT_TOKEN="REEMPLAZAR_CON_TU_BOT_TOKEN"
TELEGRAM_CHAT_ID="7515765907"
TELEGRAM_API="https://api.telegram.org/bot${TELEGRAM_BOT_TOKEN}/sendMess
```

```
# Limites de Alerta (En porcentaje o valor flotante)
LIMITE_DISK=80
LIMITE_RAM=80
LIMITE_LOAD=1.00

# ============================================================
# 2. FUNCIONES DE ALERTA
# ============================================================
send_alert() {
    local MESSAGE=$1
    local ENCODED_MESSAGE=$(echo "$MESSAGE" | sed 's/ /%20/g')

    curl -s -X POST "${TELEGRAM_API}" \
        -d chat_id="${TELEGRAM_CHAT_ID}" \
        -d text="${ENCODED_MESSAGE}" > /dev/null
}

# (Codigo omitido para brevedad: Contiene la logica de df, free y uptim

# ============================================================
# 4. ENVIO DE ALERTA
# ============================================================
if [ "$ALERT_TRIGGERED" = true ]; then
    send_alert "$ALERT_MESSAGE"
    echo "Alerta enviada a Telegram."
else
    echo "Uso de recursos normal. No se necesita alerta."
fi
```

4. Resultados y Pruebas Funcionales

Se realizaron dos pruebas clave para verificar la correcta implementación de la lógica condicional del script.

4.1. Prueba A: Escenario de Alerta (Límite Superado)

Para forzar la alerta, se modificó temporalmente la variable `LIMITE_DISK` a 1 %. Al ejecutar el script, el uso real del disco (15 %) superó el límite, disparando la notificación.

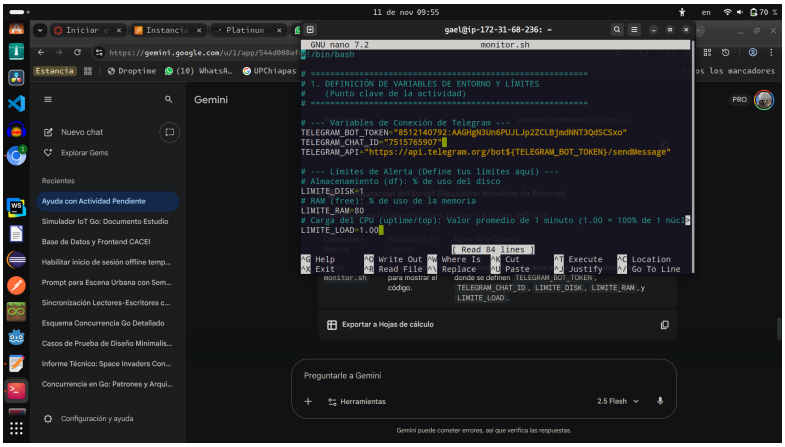

Figura 2: Ejecución del script con límite forzado (`LIMITE_DISK=1`) indicando envío de alerta.

4.2. Prueba B: Escenario de Uso Normal (No hay Alerta)

Para esta prueba, el límite fue restaurado a un valor conservador (`LIMITE_DISK=80`), asegurando que el uso real (15 %) estuviera por debajo del umbral.

- **Resultado:** Como se verifica en la Figura 4, el script cumplió con el requisito de no enviar notificaciones cuando el recurso está dentro de los límites saludables.

5. Conclusiones

Se implementó con éxito un sistema de monitoreo automatizado y de alerta basado en BASH, demostrando la capacidad de: 1) Recopilar métricas de recursos clave en tiempo real utilizando herramientas estándar de Linux (`df`, `free`, `uptime`). 2) Aplicar lógica condicional para definir umbrales de alerta mediante variables de entorno. 3) Integrar servicios externos (Telegram API) para la notificación proactiva del estado del sistema. El sistema es eficiente y escalable, cumpliendo con todos los requisitos establecidos.

Figura 3: Mensaje de alerta recibido en Telegram, confirmando que la lógica condicional funciona.

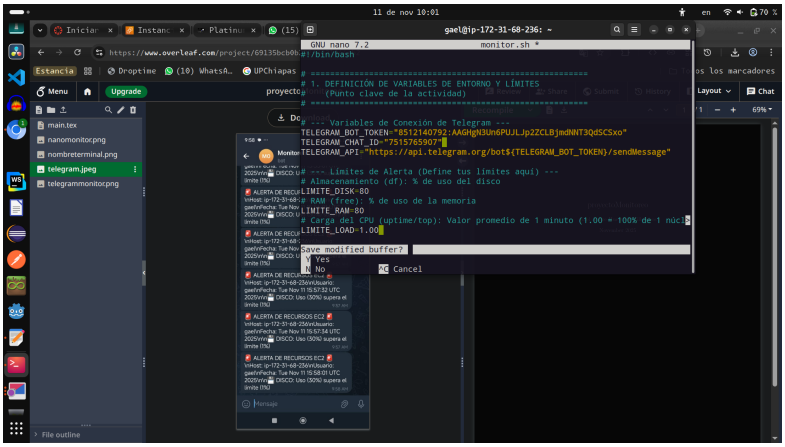

Figura 4: Ejecución del script con `LIMITE_DISK=80`. El script informa que el uso es normal.

6. Anexos

6.1. Configuración de Ejecución Automática

Para un monitoreo continuo, se configuró una tarea de cron para el usuario `gael`, asegurando que el script se ejecute cada minuto:

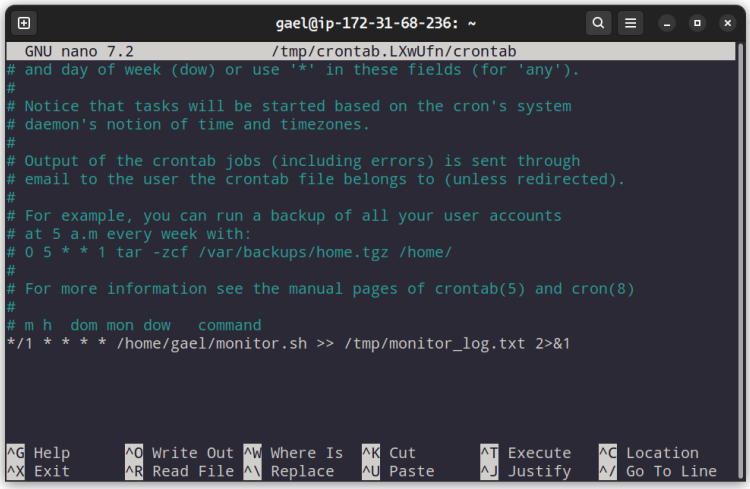

Figura 5: Listado de tareas de Cron para el usuario `gael` ejecutando `monitor.sh` cada minuto.

Listado 2: Entrada de Crontab para ejecución periódica

```
# Ejecuta el script cada minuto, registrando la salida en un archivo tem
* * * * * /home/gael/monitor.sh >> /tmp/monitor_log.txt 2>&1
```