2026/08/26

Previa

- Descargar este modelo
  - `wget -O gemma-3-1b-it-Q4_K_M.gguf "https://huggingface.co/ggml-org/gemma-3-1b-it-GGUF/resolve/main/gemma-3-1b-it-Q4_K_M.gguf"`

Script Python

- Monitorear errores del sistema:
  - monitor.py

```python
#!/usr/bin/env python3
import subprocess
import re
from datetime import datetime

def get_system_errors():
    """Extrae errores del sistema usando journalctl."""
    result = subprocess.run(
        ['journalctl', '-p', 'err', '-b', '--no-pager'],
        capture_output=True,
        text=True
    )
    lines = result.stdout.splitlines()
    # Filtrar líneas relevantes
    errors = [line for line in lines if 'error' in
line.lower()]
    return errors

def main():
    errors = get_system_errors()
    if not errors:
        print("□ No se encontraron errores en el sistema.")
        return

    print(f"⚠□ Se encontraron {len(errors)} errores.")
    # Mostrar los 5 primeros
    for err in errors[:5]:
        print(f"  - {err[:100]}...")

if __name__ == "__main__":
```

```
    main()
```

- Crear una carpeta

```
mkdir scripts/ && cd scripts/
```

  - Copiar el archivo monitor.py dentro de esta carpeta.
- Ejecutar:

  - ```
    python3 monitor.py
    ```

Script Python IA

- Descargar un archivo monitor-ia.py

  - monitor-ia.py

```
import subprocess
import json
import sys

def get_errors():
    """
    Obtiene los errores del sistema desde el journal (systemd).
    Retorna una lista de líneas de error (hasta 50).
    """
    try:
        # Opción 1: journalctl (mejor para sistemas con systemd)
        # -p 3 : solo errores (priority err)
        # -b   : desde el último arranque
        # -n 50: últimas 50 líneas
        result = subprocess.run(
            ['journalctl', '-p', '3', '-b', '-n', '50', '--no-pager'],
            capture_output=True,
            text=True,
            check=True
        )
        lines = result.stdout.strip().splitlines()
        # Filtra líneas vacías
        errors = [line for line in lines if line.strip()]
        if errors:
            return errors
        else:
            # Si no hay errores en journal, probar con dmesg
            return get_errors_dmesg()
```
```

```
    except (subprocess.CalledProcessError, FileNotFoundError):
        # Si journalctl no está disponible, usar dmesg
        return get_errors_dmesg()

def get_errors_dmesg():
    """Fallback: obtener errores del buffer de kernel con
dmesg."""
    try:
        result = subprocess.run(
            ['dmesg', '-T', '--level=err', '--color=never'],
            capture_output=True,
            text=True,
            check=True
        )
        lines = result.stdout.strip().splitlines()
        # Tomar últimas 30 líneas
        return lines[-30:] if lines else []
    except (subprocess.CalledProcessError, FileNotFoundError):
        # Si nada funciona, retornar lista vacía
        return []

def analyze_with_llama(errors):
    """Envía los errores a llama.cpp para análisis."""
    if not errors:
        return "No hay errores que analizar."

    # Limitamos a los primeros 10 para no exceder el contexto
    error_text = "\n".join(errors[:10])
    prompt = f"""
Eres un experto en administración de sistemas Linux.
Analiza los siguientes errores del sistema y sugiere
soluciones concretas:

{error_text}

Sugerencias (en español, concisas):
"""
    try:
        # Ajusta la ruta del modelo si es necesario
        result = subprocess.run(
            [
                'llama-cli',
                '-m', '/home/jared/llamacpp/gemma-3-1b-it-
Q4_K_M.gguf',
                '-p', prompt,
                '-n', '200',
                '--temp', '0.3'
            ],
            capture_output=True,
            text=True,
            check=True
```

```
        )
        return result.stdout
    except subprocess.CalledProcessError as e:
        return f"Error al ejecutar llama-cli: {e.stderr}"
    except FileNotFoundError:
        return "Error: No se encontró 'llama-cli'. Asegúrate
de que esté instalado y en el PATH."

def main():
    errors = get_errors()
    if not errors:
        print("✅ Sistema limpio. No se encontraron errores
recientes.")
        return

    print("🔍 Analizando errores con IA...")
    suggestions = analyze_with_llama(errors)
    print("\n💡 Sugerencias:")
    print(suggestions)

if __name__ == "__main__":
    main()
```

- Agregar el PATH con los binarios de llamacpp.
  - Editar el archivo .baschrc
  - Agregar al final la siguiente linea y guardar los cambios
    - `nano .bashrc`
    - `export PATH="$HOME/llamacpp/llama-b10615:$PATH"`
  - Recarga de los parámetros
    - `source .bashrc`

Portainer docker

1. Crear la carpeta portainer/ y dentro docker-compose.yml
   1. `mkdir ~/portainer && cd ~/portainer`
2. Archivo docker-compose.yml
   1. docker-compose.yml

```
services:
  portainer:
```

```
    image: portainer/portainer-ce:latest
    container_name: portainer
    restart: unless-stopped
    ports:
      - "9000:9000"   # Puerto HTTP (interfaz web)
      - "9443:9443"   # Puerto HTTPS (si usas SSL)
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock   # Acceso
al motor Docker
      - portainer_data:/data                        #
Persistencia de datos
    environment:
      - TZ=America/La_Paz   # Ajusta tu zona horaria
(opcional)

volumes:
  portainer_data:
```