Reporte técnico: Proyecto final de Sistemas Operativos y Laboratorio

1. Información del Proyecto

- **Título del Proyecto:** Monitoreo de Recursos del Sistema en Tiempo Real con FastAPI, Prometheus y Grafana
- **Curso/Materia:** Sistemas Operativos
- **Integrantes:**
  - Jonathan Mazo González - jonathan.mazog@udea.edu.co
  - José David Henao Gallego - jose.henao1@udea.edu.co
  - Juan Esteban Aristizábal - jesteban.aristizabal@udea.edu.co
  - Sharid Samantha Madrid Ospina - sharid.madrid@udea.edu.co

- **Fecha de Entrega:** [Fecha]

2. Introducción

2.1. Objetivo del Proyecto

El objetivo es desarrollar una API REST que recolecte y exponga métricas del sistema operativo, como el uso de CPU, memoria RAM y operaciones de E/S, de forma sencilla y accesible. Con esto, queremos demostrar que es posible medir y visualizar el rendimiento del sistema usando herramientas modernas como FastAPI, Prometheus y Grafana.

2.2. Motivación y Justificación

Gracias a la experiencia laboral que hemos tenido en nuestras prácticas y trabajos, nos hemos dado cuenta de lo importante que es estar pendientes del rendimiento de los recursos de los sistemas que usamos a diario. Muchas veces, cuando algo va lento o falla, no sabemos qué está pasando por dentro ni qué procesos están consumiendo más CPU o memoria.
Por eso, decidimos desarrollar una herramienta que nos permitiera entender el trasfondo de todo esto, conociendo de forma práctica cómo se comporta el sistema y cómo se pueden visualizar estos datos de manera clara. Además, aprender a construir herramientas de monitoreo nos prepara para futuros proyectos donde sea importante optimizar el rendimiento de las aplicaciones que desarrollemos o gestionemos.

2.3. Alcance del Proyecto

Nuestro principal enfoque es realizar el monitoreo de los recursos del sistema y tener una interfaz donde podamos visualizar o exponer en tiempo real las métricas de rendimiento obtenidas de dicho monitoreo
En específico, buscamos monitorear:

- El uso de la CPU.
- Cuánta memoria RAM se está consumiendo.
- Las operaciones de entrada y salida (E/S) que realiza el sistema.

Estas métricas se exponen a través de una API creada con FastAPI y se integran con Prometheus y Grafana para poder visualizarlas de forma clara y comprensible en gráficos y paneles.
Fuera del alcance de este proyecto quedan aspectos como monitorear varios equipos al mismo tiempo o integrar alertas automáticas según los datos recolectados, ya que nuestro objetivo es mantener el enfoque en aprender y aplicar de forma práctica los conceptos de Sistemas Operativos a través de este proyecto.

3. Marco Teórico / Conceptos Fundamentales

Para desarrollar una API que **monitoree el rendimiento de un sistema en tiempo real**, es fundamental comprender ciertos aspectos clave de los sistemas operativos y las herramientas modernas de monitoreo.
Uno de estos pilares es la **planificación de procesos**, que incluye cómo el sistema operativo maneja la creación, ejecución y finalización de procesos. Comprender los estados por los que pasa un proceso (nuevo, listo, en ejecución, bloqueado y terminado) y cómo se decide el tiempo que cada proceso usa la CPU, permite interpretar mejor las métricas de uso de CPU y detectar posibles cuellos de botella en el sistema [1].
La **administración de memoria** también es un aspecto clave. Cada proceso necesita memoria para ejecutarse, y el sistema operativo se encarga de asignarla y liberarla de forma eficiente. Entender cómo funciona esta gestión, incluyendo conceptos como paginación y swapping, ayuda a analizar el consumo de RAM y a optimizar el uso de la memoria en las aplicaciones [2].
El manejo de las **operaciones de entrada y salida (E/S)** impacta directamente en el rendimiento del sistema. Las lecturas y escrituras en discos o en la red pueden convertirse en puntos críticos si no se manejan adecuadamente, generando demoras o bloqueos. Monitorear estas operaciones permite identificar y resolver problemas relacionados con la E/S antes de que afecten al sistema [3].
Para obtener información de estas métricas, usamos herramientas modernas como:

- **psutil**, una biblioteca de Python que permite acceder a información sobre el uso de CPU, memoria, discos y red de forma sencilla y en tiempo real [4].
- **py-spy**, un profiler para programas en Python que permite ver en qué partes del código se está gastando más tiempo sin necesidad de modificar el código fuente [5].

- **cProfile**, una herramienta de Python que proporciona estadísticas detalladas sobre la ejecución de programas para identificar funciones que consumen más tiempo y recursos [6].
- **perf**, una herramienta de Linux para analizar el rendimiento del sistema utilizando contadores de eventos y visualizando el uso de CPU y memoria a nivel de kernel [7].

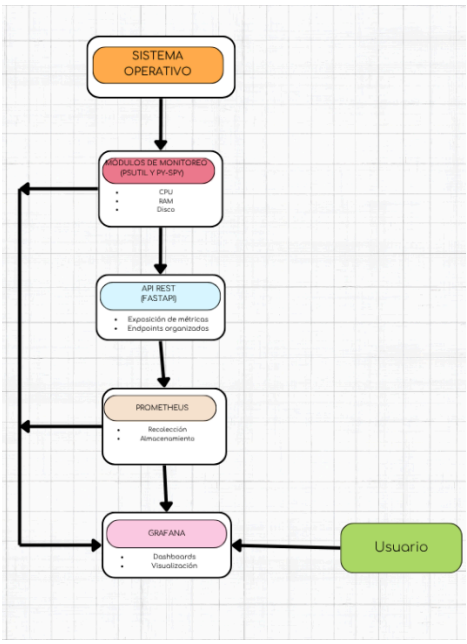
Para visualizar y analizar las métricas recolectadas, utilizamos:

- **Prometheus**, un sistema de monitoreo de código abierto que recopila y almacena métricas en forma de series temporales, permitiendo consultas eficientes [8].
- **Grafana**, una herramienta que se integra con Prometheus para crear dashboards interactivos que facilitan la interpretación visual de las métricas recolectadas [9].

4. Diseño e Implementación

4.1. Diseño de la Solución

Para el diseño de este proyecto, planteamos una arquitectura modular que nos permitiera recolectar métricas de rendimiento de forma clara y exponerlas a través de una API accesible.
La solución se compone de los siguientes módulos principales:
Módulos de monitoreo, encargados de recolectar métricas del sistema como uso de CPU, memoria y operaciones de E/S en tiempo real.
API REST, que expone estas métricas mediante endpoints organizados y fáciles de consumir.
Integración con Prometheus, que se encarga de recolectar estas métricas de la API.
Visualización en Grafana, para analizar las métricas a través de dashboards interactivos.
La decisión de utilizar una arquitectura modular se tomó para facilitar el mantenimiento y futuras expansiones, como agregar nuevas métricas o cambiar tecnologías sin afectar todo el sistema.
En el diagrama se muestra el flujo del proyecto:



El Sistema Operativo Linux genera métricas que son capturadas por los módulos de monitoreo (psutil y py-spy). Estos datos se exponen a través de una API REST creada con FastAPI, la cual es consultada por Prometheus para recolectar y almacenar las métricas. Luego, Grafana se conecta a Prometheus para visualizar estos datos en dashboards, permitiendo al usuario consultar el rendimiento del sistema en tiempo real de forma clara y organizada.

4.2. Tecnologías y Herramientas

Para la implementación del proyecto, utilizamos las siguientes tecnologías:

- Python: Lenguaje de programación principal por su facilidad de uso y rapidez de desarrollo.
- FastAPI: Framework para construir la API REST de forma eficiente.
- psutil: Librería de Python para recolectar métricas del sistema.
- Prometheus: Herramienta de monitoreo para recolección y almacenamiento de métricas.
- Grafana: Herramienta para visualizar métricas de manera gráfica e interactiva.
- GitHub: Para control de versiones y trabajo colaborativo.
- Visual Studio Code: Editor de código utilizado durante el desarrollo.
- Sistemas basados en Linux: Entorno de pruebas para facilitar el acceso a las métricas del sistema.

4.3. Detalles de Implementación

Para implementar las funcionalidades clave, organizamos el proyecto en carpetas:

- app/: Contiene el núcleo de la API, con módulos para rutas, servicios de monitoreo y utilidades.
- routes/: Contiene los endpoints para exponer las métricas de CPU, memoria y E/S.
- services/: Contiene los scripts que usan psutil para capturar las métricas de rendimiento.
- exporters/: Configura el exportador de Prometheus para exponer las métricas recolectadas.
- grafana/ y prometheus/: Contienen archivos de configuración para dashboards y scraping de métricas.
- tests/: Contiene pruebas automatizadas para garantizar el correcto funcionamiento de la API.

Los archivos de configuración de Prometheus (prometheus.yml) indican la URL de nuestra API para el scraping de las métricas en /metrics. En Grafana, configuramos dashboards para visualizar el uso de CPU, memoria y operaciones de E/S de manera clara.
Para capturar las métricas, utilizamos las funciones de psutil, por ejemplo:

- psutil.cpu_percent() para obtener el porcentaje de uso de CPU.
- psutil.virtual_memory() para obtener el uso de memoria RAM.
- psutil.disk_io_counters() para obtener operaciones de Disco.

Cada uno de estos datos se organiza en estructuras tipo JSON y se exponen a través de FastAPI, permitiendo que Prometheus pueda consultarlas de forma periódica y almacenar las series temporales para su análisis en Grafana.
Con este diseño e implementación, logramos que el sistema sea entendible, funcional y práctico, manteniendo la modularidad para que podamos agregar nuevas métricas o integraciones futuras de forma sencilla.

5. Pruebas y Evaluación

5.1. Metodología de Pruebas

Para probar el proyecto utilizamos pruebas funcionales manuales, verificando que cada endpoint de la API respondiera correctamente y entregara las métricas esperadas. Además, realizamos pruebas de integración con Prometheus y Grafana, observando que las métricas se recolectaran y visualizaran en tiempo real. Creamos casos de prueba específicos para CPU, memoria y E/S bajo distintos niveles de carga.

5.2. Casos de Prueba y Resultados

ID Caso de prueba	**Descripción**	**Resultado esperado**	**Resultado obtenido**	**Éxito/Fallo**
CP-001	Consultar endpoint `/metrics`	Respuesta con métricas JSON	Respuesta con métricas JSON	Éxito

CP-002	Visualizar métricas en Grafana	Dashboard muestra CPU, RAM y Disco	Dashboard muestra CPU, RAM y Disco	Éxito
CP-003	Alta carga en CPU (prueba de estrés)	Reflejar incremento en métricas de CPU	Métricas de CPU aumentan en tiempo real	Éxito
CP-004	Simular alto uso de memoria	Mostrar incremento en uso de RAM	Incremento en gráficas de uso de RAM	Éxito
CP-005	Verificar estabilidad en recolección de datos	Prometheus recolectar métricas sin fallas	Recolección estable sin fallas	Éxito
CP-006	Verificar el endpoint /metrics/system	Devuelve conteo de procesos e hilos en JSON	Devuelve conteo correcto en JSON	Éxito
CP-007	Validar panel en Grafana para procesos e hilos	Dashboard muestra valores actualizados	Valores correctos en tiempo real en Grafana	Éxito
CP-008	Verificar el endpoint /metrics/system/network	Devuelve bytes enviados y recibidos	Respuesta JSON con métricas de red	Éxito
CP-009	Validar panel en Grafana para tráfico de red	Dashboard muestra bytes enviados/recibidos	Paneles muestran datos acumulados en Grafana	Éxito
CP-010	Verificar endpoint /metrics/system para uptime	Devuelve uptime en segundos	Respuesta correcta en JSON	Éxito
CP-011	Validar panel en Grafana para uptime	Panel muestra uptime en segundos	Panel correcto y actualizado en Grafana	Éxito

5.3. Evaluación del Rendimiento (si aplica)

Realizamos pruebas generando carga en el sistema (procesos de cálculo y lectura/escritura en disco) y observamos en Grafana:

- Incremento en el uso de CPU de 15% a 85% durante cargas altas.
- Uso de memoria reflejado correctamente al abrir múltiples programas.
- Operaciones de E/S mostraron picos al copiar archivos grandes.

5.4. Problemas Encontrados y Soluciones

Problema: Error al exponer métricas en el endpoint /metrics debido a conflicto de puertos.
Solución: Cambiamos el puerto de exposición de la API en el archivo de configuración.
Problema: Grafana no mostraba datos actualizados.
Solución: Verificamos la configuración del scraping de Prometheus y ajustamos el intervalo a 5 segundos.
Problema: Lecturas de E/S consistentes en pruebas iniciales.
Solución: Ajustamos el uso de psutil.disk_io_counters() con parámetros correctos.

6. Conclusiones

Lo más valioso de este proyecto fue conocer a cabalidad el trasfondo del funcionamiento de los sistemas que usamos todos los días y darnos cuenta de cómo algo que aprendimos en clase podía llevarse a la realidad con un proyecto práctico.
Con esta API, logramos cumplir el objetivo de monitorear el rendimiento del sistema en tiempo real y visualizar de forma clara métricas de CPU, RAM y operaciones de E/S. Esto nos permitió aplicar de forma práctica conceptos como planificación de procesos, administración de memoria y manejo de E/S, entendiendo cómo estos impactan en el rendimiento de un sistema.
Además de fortalecer nuestro aprendizaje en sistemas operativos, este proyecto nos mostró que con herramientas accesibles y trabajo en equipo es posible construir soluciones que sean útiles y nos motivan a seguir aprendiendo.

7. Trabajo Futuro (Opcional)

Para el futuro, se podrían realizar las siguientes mejoras y extensiones al proyecto:

- Monitoreo en múltiples equipos de forma centralizada, para observar métricas de diferentes sistemas en un solo dashboard.
- Agregar alertas automáticas en Grafana, que notifiquen cuando el uso de CPU o memoria supere ciertos límites.
- Desplegar el sistema en contenedores Docker, facilitando su instalación en diferentes entornos.

8. Referencias

[1] IBM. “What is a process?” IBM Documentation.
https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-what-is-process

[2] Red Hat. “Understanding Linux Memory Management.” https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/ch-memory

[3] GeeksforGeeks. “I/O Management in Operating System.” https://www.geeksforgeeks.org/i-o-management-in-operating-system/

[4] Rodola, G. “psutil – process and system utilities.” https://psutil.readthedocs.io/

[5] Ben Frederickson. “py-spy: Sampling profiler for Python programs.” https://github.com/benfred/py-spy

[6] Python Software Foundation. “The Python Profilers.” https://docs.python.org/3/library/profile.html

[7] Brendan Gregg. “perf: Linux Pofiling with Performance Counters.” https://perf.wiki.kernel.org/index.php/Main_Page

[8] Prometheus Authors. “Introduction to Prometheus.” https://prometheus.io/docs/introduction/overview/

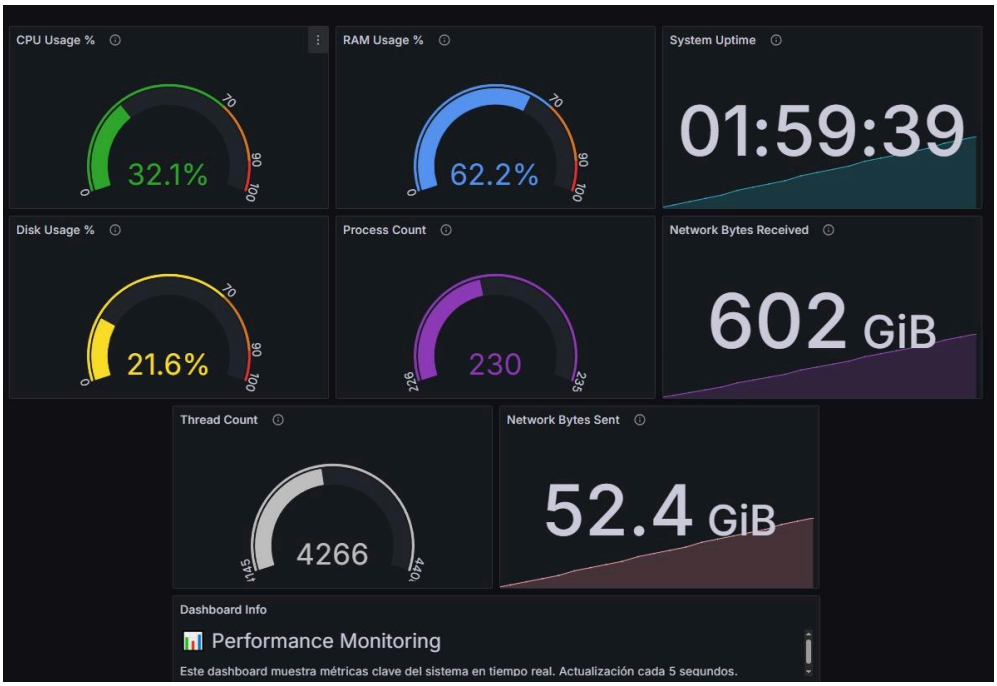
[9] Grafana Labs. “Prometheus Data Source Configuration.” https://grafana.com/docs/grafana/latest/datasources/prometheus/

Enumere todas las fuentes que consultaste (libros, artículos, sitios web, documentación de herramientas, etc.) utilizando un formato de citación consistente.

9. Anexos

A continuación, se incluye material adicional que complementa el reporte:

Captura del Dashboard en Grafana:



La imagen muestra el dashboard de Performance Monitoring en Grafana, donde se visualizan métricas clave como uso de CPU, RAM, disco, número de procesos, bytes de red enviados y recibidos, y tiempo de actividad del sistema en tiempo real:

Enlace al repositorio del proyecto:

- https://github.com/Performance-API-SO-2025/PerformanceAPI

Video de presentación del proyecto:

- https://www.youtube.com/watch?v=czDRhuk4LYw

Instaladores:

- https://drive.google.com/drive/folders/14CrkRh-kaXsW-5RJXbu6AM-cJ3KBorgO?usp=sharing