Taller 1: Evaluación Comparativa de Procesamiento Geoespacial

AUTHOR
Lina Quintero

PUBLISHED
2026-02-15

Metodología

En este ejercicio se procesó una banda de Sentinel-2A (~120 millones de píxeles) para comparar el rendimiento de diferentes motores geoespaciales bajo una operación aritmética y una reducción global (media).

Parte A: Resultados en JupyterLab

Motor Geoespacial	Lenguaje	Tiempo (s)	Media Global
R: terra	R (C++)	10.672	3766.625
R: stars	R	14.675	3766.625
Python: rasterio	Python	1.587	766.624630
Julia: Rasters.jl	Julia	7.10847472	3766.62463

Observaciones por Notebook

- **01_benchmark_terra.ipynb**: El motor `terra` es altamente eficiente ya que gestiona los archivos por referencia y realiza los cálculos en C++.
- **02_benchmark_stars.ipynb**: Este motor suele ser más lento en reducciones globales debido a la materialización de datos en la memoria de R.
- **03_benchmark_rasterio.ipynb**: Destaca por su velocidad al utilizar arreglos de NumPy optimizados con instrucciones de hardware.
- **04_benchmark_rasters_julia.ipynb**: Julia utiliza un modelo de pipelines composables que, tras la compilación inicial, permite un procesamiento nativo muy fluido.

parte B — Ejecución desde VSCode (terminal integrada)

Comparación de Rendimiento: Notebooks vs. Terminal

Motor de Procesamiento	Tiempo Notebook (Parte A)	Tiempo Terminal (Parte B)
R: terra	10.672 s	12.736 s

Motor de Procesamiento	Tiempo Notebook (Parte A)	Tiempo Terminal (Parte B)
R: stars	14.675 s	15.227 s
Python: rasterio	1.587 s	2.621 s
Julia: Rasters.jl	7.108 s	13.457 s

Parte C — Ejecución desde el Termina de Windows (PowerShell)

Resumen Comparativo de Tiempos de Procesamiento

Motor de Procesamiento	Parte A: JupyterLab (s)	Parte B: Terminal VS Code (s)	Parte C: PowerShell Directo (s)
R: terra	10.672	12.736	15.368
Python: rasterio	1.587	2.621	11.06
R: stars	14.675	2.621 s	2.477
Julia: Rasters.jl	7.108	13.457 s	18.36

Nota sobre la Parte C: Para la ejecución desde PowerShell se utilizó la **Opción 2 (Ejecución directa)** mediante el comando `docker exec`. Se observa que este método es el más eficiente para automatizar procesos de GeoAI, ya que consume menos recursos de interfaz gráfica de los 12GB de RAM asignados al contenedor.

Diferencias Notadas

- **Optimización de RAM:** Al prescindir de la interfaz de VS Code o Jupyter, se liberan recursos del sistema, permitiendo que los **12GB de RAM** se enfoquen exclusivamente en el procesamiento de datos.
- **Latencia de Inicialización:** La ejecución en PowerShell presenta una ligera demora inicial comparada con Jupyter, ya que debe cargar el intérprete y las librerías en cada llamada, mientras que en el Notebook el kernel ya está activo.
- **Estabilidad:** La terminal de Windows ofrece una ejecución más robusta ante bloqueos de la interfaz web, ideal para procesos que requieren alta intensidad de cómputo.
- **Compilación JIT:** En Julia, la terminal permite monitorear de forma más limpia el proceso de compilación inicial antes de la ejecución del benchmark.

Análisis y Conclusiones Finales

1. 📍 Entorno de ejecución

¿Notaron diferencias de tiempo entre JupyterLab, VSCode y PowerShell? Sí, se observaron variaciones. Los Notebooks (JupyterLab) suelen ser más rápidos en ejecuciones repetitivas porque el kernel ya está activo y las librerías cargadas en los 12GB de RAM.

Razón técnica: La ejecución en PowerShell o Terminal tiene un mayor overhead inicial, ya que cada vez que se corre un script, el sistema debe inicializar el intérprete (R, Python o Julia) y cargar todos los paquetes desde el disco al contenedor. En cambio, el kernel de Jupyter mantiene el entorno "caliente".

2. 🧱 Abstracción en la práctica

¿En qué motor creen que el costo de las abstracciones es más visible? En R (específicamente con stars).

Relación con los tiempos:

Al observar los tiempos de 14.675 s en Jupyter, se nota que, aunque la sintaxis es muy amigable (alta abstracción), el motor realiza múltiples conversiones internas y manejo de metadatos que lo hacen más lento que terra o rasterio. La abstracción facilita el código pero penaliza la velocidad de procesamiento puro.

3. 🔥 Julia y el costo de compilación (Warm-up)

¿El efecto del warm-up de Julia se notó más en algún entorno específico? Se notó mucho más en la Terminal (VS Code y PowerShell).

En la terminal, Julia debe realizar la compilación JIT (Just-In-Time) de las librerías Rasters y ArchGDAL en cada ejecución individual del script. En JupyterLab, una vez que la primera celda compila los paquetes, las ejecuciones siguientes son casi instantáneas porque el código ya está traducido a lenguaje de máquina en la memoria activa.

4. 🧠 Elección informada

¿Cambiarían su elección del "Titán" para una emergencia ambiental real? No, mantendría a Python (rasterio) como el titán por su velocidad consistente (cerca de 1.5s - 2.6s) en todos los entornos. En una emergencia, la capacidad de ejecutar scripts rápidos vía terminal (PowerShell) sin depender de una interfaz pesada es vital para procesar grandes volúmenes de datos Sentinel-2 de manera automática y confiable bajo la configuración de 12GB de RAM optimizada.