**Proyecto 1 Telemática - DFS Minimalista en AWS**
**Realizado por:** Samuel Alarcon, Alejandro Zapata y Andres Suarez

**Repositorio de Github:** https://github.com/AlarconG03/Proyecto_GridFS
**Usuario de AWS con el cual se ejecutó el proyecto:** asuarezr2 - Andres Suarez Rios
- Account ID: 471112753158
- Federated user: voclabs/user4023935=asuarezr2

**Link del video explicativo:**
**Autoevaluación:** 5.0

**1. Objetivo y marco teórico breve:**
Objetivo: Diseñar e implementar un sistema de archivos distribuido (DFS) minimalista que permite almacenar y recuperar archivos grandes, dividiéndolos en bloques distribuidos en múltiples nodos de datos, gestionados centralmente por un NameNode.

Marco teórico: Un DFS (Distributed File System) permite gestionar archivos que no residen en un solo servidor, sino que se dividen y almacenan en varios nodos.
- El NameNode actúa como controlador central que mantiene la metadata: qué archivos existen, en qué bloques están divididos y dónde se encuentran esos bloques.
- Los DataNodes son nodos de almacenamiento donde realmente se guardan los bloques.
- Los Clientes interactúan con el NameNode y los DataNodes para ejecutar operaciones de archivos (put, get, ls, rm).

El sistema implementado no usa replicación, lo que simplifica la arquitectura, pero implica que la caída de un DataNode conlleva pérdida parcial del archivo.

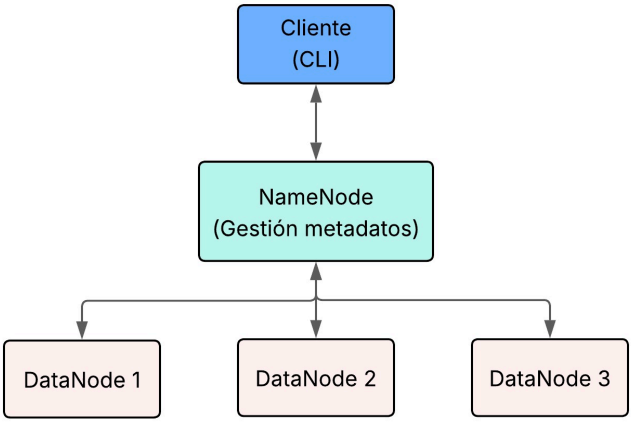
**2. Descripción del servicio y problema abordado:**
Descripción del servicio: El proyecto resuelve la necesidad de almacenar y acceder a archivos de forma distribuida en un entorno controlado (AWS con múltiples instancias EC2).

Problemas abordados:
- Cómo dividir un archivo en bloques para distribuirlo entre varios nodos.
- Cómo mantener una tabla central de metadatos accesible por múltiples clientes.
- Cómo permitir que distintos usuarios tengan su propio espacio de archivos (separación de sesiones).
- Cómo manejar fallos básicos cuando un DataNode no está disponible.

**3. Arquitectura del sistema y diagramas:**
Componentes:
- NameNode:
  - Mantiene los metadatos de archivos.
  - Provee autenticación de usuarios.
  - Registra operaciones put, ls, get, rm.
- DataNodes (3):
  - Reciben bloques de archivos desde los clientes.
  - Guardan los bloques como ficheros locales.
  - Devuelven bloques al cliente en operaciones get.

- Cliente:
    - Acceden al DFS a través de un CLI en Python.
    - Ejecutan operaciones sobre archivos (put, get, ls, rm).
    - Espacios de usuario independientes.

Arquitectura:




**4. Especificación de protocolos y APIs:**
Comunicación Cliente ↔ NameNode:

- Login: POST /login (JSON: username, password).
- Listar archivos: GET /ls?username=....
- Registrar archivo (put): POST /put (JSON con bloques).
- Obtener metadatos (get): GET /get?username=...&filename=....
- Eliminar archivo: DELETE /rm (JSON: username, filename).

Comunicación Cliente ↔ DataNode:

- Guardar bloque: POST /store_block (form-data con block_id, file).
- Obtener bloque: GET /get_block/<block_id>.

Todos los protocolos implementados sobre HTTP (REST) usando Flask en Python.

**5. Algoritmos de particionamiento y distribución:**
Particionamiento:

- Cada archivo se lee secuencialmente y se divide en bloques fijos de 5 bytes (configurable, ejemplo pequeño para pruebas).

Distribución:

- Los bloques se asignan a DataNodes mediante un algoritmo round-robin:
    - bloque0 → DataNode1
    - bloque1 → DataNode2
    - bloque2 → DataNode3
    - bloque3 → DataNode1
    - ...

Recuperación:

- El NameNode devuelve la lista de bloques y sus DataNodes.
- El cliente los descarga en orden y reconstruye el archivo.

**6. Descripción del entorno de ejecución:**

Infraestructura:

- AWS EC2, 5 instancias:
  - 1 NameNode (Ubuntu 24.04, t3.micro).
  - 3 DataNodes (Ubuntu 24.04, t3.micro).
  - 1 Cliente (Ubuntu 24.04, t3.micro).

Software:

- Python 3.12.
- Flask (servidores REST en NameNode y DataNodes).
- Requests (librería cliente para CLI).

Ejecución:

- Cada servicio corre de forma nativa en Linux (no se usó Docker).
- Entornos virtuales (venv) para aislar dependencias en cada instancia.

**7. Pruebas y análisis de resultados:**

Prueba 1: Almacenamiento y recuperación:

- Cliente Shakira sube prueba.txt con put.
- NameNode registra el archivo, DataNodes guardan bloques.
- Cliente ejecuta ls → lista prueba.txt.
- Cliente ejecuta get prueba.txt recuperar_prueba.txt → archivo reconstruido correctamente y visible en recuperar_prueba.txt.
  ✅ Resultado esperado.

Prueba 2: Sesiones separadas:

- Shakira sube prueba.txt.
- Petro hace ls → lista vacía.
  ✅ Archivos aislados por usuario.

Prueba 3: Fallo de un DataNode:

- Apagado de DataNode1.
- Cliente intenta get prueba.txt.
- Mensaje:
  - ⚠️ Bloque prueba.txt_block0 falló en DataNode-1
  - ❌ Archivo incompleto. Funcionaron: {DataNode-2, DataNode-3}, Fallaron: {DataNode-1}

  ✅ El sistema detecta fallo y reconstruye archivo incompleto con lo disponible.

Prueba 4: Eliminación de archivo:

- Cliente ejecuta rm prueba.txt.
- NameNode borra la metadata.
- ls ya no muestra el archivo.
  ✅ Funcionalidad correcta.