Documentación de Orquestación: Multi-Pipeline (pipelines.yml)

Descripción General

El archivo pipelines.yml actúa como el **controlador maestro** de la instancia de Logstash. Su función es permitir la ejecución simultánea de múltiples flujos de datos (pipelines) dentro de un mismo proceso de Logstash, garantizando que operen de manera aislada y con recursos dedicados.

En este proyecto, se utiliza para separar la ingesta de datos de negocio (Juegos de Steam) de la ingesta de datos técnicos (Logs del Scraper).

Configuración Implementada

A continuación se detalla la configuración activa en /home/g6/reto/logstash-9.2.1/config/pipelines.yml:

```
# Pipeline 1: Los Juegos (Datos de negocio + IA)
- pipeline.id: steam_ingestion
  path.config: "/home/g6/reto/logstash-9.2.1/config/Datos.conf"
  pipeline.workers: 2  # Mayor potencia para procesamiento pesado

# Pipeline 2: Los Logs (Monitorización + Grok)
- pipeline.id: grok-logs
  path.config: "/home/g6/reto/logstash-9.2.1/config/Logs.conf"
  pipeline.workers: 1  # Un solo hilo para mantener el orden secuencial
```

Análisis de Pipelines

1. Pipeline de Negocio: steam_ingestion

Este pipeline es el "núcleo" del reto, encargado de procesar la información de los videojuegos.

- **pipeline.id: steam_ingestion**: Identificador único para monitoreo. Si hay un error en Kibana/Monitoring, aparecerá bajo este nombre.
- **path.config**: Apunta al archivo Datos.conf.
- **pipeline.workers: 2**:
  - **Estrategia:** Paralelismo.
  - **Justificación:** El procesamiento de estos datos es computacionalmente costoso.

Implica parsear objetos JSON grandes y realizar conversiones de tipos (vectores, precios). Al asignar 2 workers, Logstash puede procesar dos eventos simultáneamente, duplicando teóricamente la velocidad de ingesta.

2. Pipeline de Observabilidad: grok-logs

Este pipeline es secundario y sirve para monitorear la salud del scraper.

- **pipeline.id: grok-logs**: Identificador para los logs técnicos.
- **path.config**: Apunta al archivo Logs.conf.
- **pipeline.workers: 1**:
  - **Estrategia:** Secuencialidad.
  - **Justificación:**
    1. **Orden:** En los logs de errores, el orden de llegada es vital (saber qué error ocurrió antes que otro). Al usar un solo worker, se garantiza mejor el orden secuencial de escritura en Elasticsearch.
    2. **Eficiencia:** El parseo de texto plano es ligero. No es necesario desperdiciar múltiples núcleos de CPU en esta tarea.

Ventajas de esta Arquitectura

El uso de pipelines.yml frente a un único archivo de configuración gigante ofrece tres ventajas críticas para el reto:

1. Aislamiento de Fallos (Reliability):
   Si el archivo Logs.conf tiene un error de sintaxis o el pipeline de logs se bloquea por un mensaje mal formado, el pipeline de steam_ingestion NO se detiene. Los juegos siguen ingestándose sin interrupción.
2. Gestión de Recursos (Tuning):
   Permite asignar recursos de CPU de forma granular. Podemos dar más potencia a la tarea crítica (Steam) y limitar la tarea secundaria (Logs) para que no compitan por el procesador.
3. Limpieza de Datos:
   Evita la "contaminación cruzada". Los eventos del log nunca pasarán por los filtros de los juegos y viceversa, ya que viven en tuberías totalmente separadas desde su nacimiento.