Documentación del Pipeline de Monitorización: Logs del Scraper (Logs.conf)

Este documento detalla la configuración del pipeline encargado de procesar los **logs técnicos** generados por el script de scraping. A diferencia de los datos de negocio (juegos), estos datos son métricas operacionales (latencia, errores, códigos de estado HTTP) críticas para la observabilidad del sistema.

1. Fuente de Datos y Etiquetado (Input)

La entrada lee un archivo de texto plano donde el scraper escribe sus eventos línea por línea.

- **Archivo fuente:** /home/g6/reto/datos/scraper_metrics.log
- **Estrategia de Etiquetado:**
  - tags => ["scraper_metrics"]
  - **¿Por qué?** En entornos donde Logstash corre múltiples pipelines o inputs, es vital etiquetar el origen. Esto permite que en la sección filter usemos un condicional if "scraper_metrics" in [tags] para asegurar que **solo** aplicamos estas transformaciones a los logs, y no intentamos parsear los juegos JSON con estas reglas.

2. Lógica de Parseo en Cascada (Filter)

El filtrado sigue una estrategia de "embudo" o cascada: primero se estructura lo general y luego se especifican los detalles según el nivel del log.

2.1. Nivel 1: Separación de Cabecera (Grok General)

Todos los logs, sean errores o información, comparten un formato común al inicio (Timestamp y Nivel).

- **Patrón:** ^%{TIMESTAMP_ISO8601:log_timestamp} - %{LOGLEVEL:log_level} - %{GREEDYDATA:raw_message}$
- **Resultado:**
  - log_timestamp: La fecha y hora escrita en el log.
  - log_level: INFO o ERROR.
  - raw_message: El resto del texto que contiene los datos específicos que procesaremos en el paso 2.

2.2. Nivel 2: Extracción Condicional (Grok Específico)

Dependiendo del log_level extraído en el paso anterior, se aplica una lógica diferente:

A. Logs de tipo INFO (Métricas de Éxito)

Se extraen métricas de rendimiento para visualizar la salud del scraper.

- **Datos extraídos:**
  - request_url: La URL atacada.
  - http_status: El código de respuesta (ej. 200, 404). Se tipa como **int**.
  - api_latency: Tiempo de respuesta. Se tipa como **float** para calcular promedios de velocidad.
  - search_offset: Paginación actual.

B. Logs de tipo ERROR (Diagnóstico de Fallos)

Se usa un patrón Grok avanzado con **operador OR** (?:...)|(?:...) para manejar dos formatos de error posibles:

1. **Errores de conexión:** Captura error_type, failed_url y failed_offset.
2. **Errores genéricos:** Captura error_type, failed_url y error_details (stack trace o mensaje).

2.3. Gestión Temporal (Date)

- **Match:** yyyy-MM-dd HH:mm:ss
- **Timezone:** Europe/Madrid
- **Importancia:** A diferencia de los datos de Steam (que suelen venir en UTC), los logs de aplicaciones locales suelen estar en la hora del servidor. Especificar la zona horaria correcta es vital para que Kibana muestre los logs en el momento exacto que ocurrieron.

2.4. Limpieza (Mutate)

Se eliminan los campos intermedios raw_message (el texto sucio restante) y message (la línea original completa) para no duplicar información en Elasticsearch.

3. Salida y Almacenamiento (Output)

3.1. Enrutamiento Condicional

El bloque output también está envuelto en if "scraper_metrics" in [tags]. Esto previene que los logs técnicos terminen mezclados en el índice de juegos de Steam si ambos pipelines corrieran juntos.

3.2. Indexación de Series Temporales

- **Índice:** scraper_logs-%{+yyyy.MM.dd}
- **Uso:** Al igual que con los datos, se crea un índice diario. Esto permite establecer políticas de retención (ej. "borrar logs técnicos de más de 7 días") sin afectar a los datos de

negocio.