UNIVERSIDAD DIEGO PORTALES

FACULTAD DE INGENIERÍA Y CIENCIAS

Tarea 3 sistemas distribuidos

Asignatura:
Sistemas Distribuidos
Profesor Catedra:
Nicolas Hidalgo
Estudiantes:
Paolo Nicolás Genta Valenzuela

Índice

1. Introducción

En el contexto de los sistemas distribuidos, la capacidad de recolectar, procesar y visualizar grandes volúmenes de datos en tiempo real representa un desafío clave para aplicaciones modernas. Esta tarea busca diseñar e implementar una solución distribuida capaz de simular, filtrar, procesar y analizar eventos de tráfico urbano, utilizando herramientas del ecosistema big data y tecnologías de visualización.
El objetivo central es construir una arquitectura funcional que permita transformar datos semiestructurados sobre eventos de tráfico —provenientes originalmente de fuentes como el mapa en vivo de Waze— en información estructurada útil, procesada y visualmente accesible. Para ello, se emplearon tecnologías distribuidas como Apache Pig para el análisis por lotes, Redis como sistema de caché de alta velocidad, Elasticsearch para el almacenamiento indexado de datos, y Kibana para su visualización interactiva.
Durante el desarrollo de la entrega 2, se detectaron limitaciones importantes en el proceso de recolección de datos reales mediante técnicas de scraping, tales como baja densidad de eventos capturados, dificultades de automatización estable y restricciones dinámicas del sitio. Por esta razón, se optó por generar un conjunto de datos sintéticos realistas, respetando las distribuciones típicas observadas en Waze y ajustados para representar patrones urbanos verosímiles de la Región Metropolitana.
El sistema completo se orquestó mediante contenedores Docker, lo que permitió una integración modular, reproducible y fácilmente desplegable entre los distintos componentes distribuidos.

2. Diseño del sistema distribuido

El sistema diseñado se basa en una arquitectura modular compuesta por componentes distribuidos, cada uno con una función específica dentro del flujo de procesamiento de datos. Esta modularidad permite escalar, depurar y reutilizar cada parte de manera independiente, facilitando además su despliegue mediante contenedores Docker.

2.1. Arquitectura general del sistema

El flujo general del sistema es el siguiente:

1. **Generación de datos sintéticos:** se simulan eventos de tráfico urbanos (como accidentes, congestión o presencia policial) utilizando un generador basado en Python. Este componente permite obtener un volumen de datos realista que imita los patrones de Waze, tanto en distribución geográfica como en frecuencia de tipos de evento.

2. **Preprocesamiento:** los datos generados son limpiados, normalizados y deduplicados. Este paso elimina eventos incompletos o mal estructurados, homogeneiza los tipos de evento (por ejemplo, distintas formas de referirse a "policía") y reduce eventos redundantes en un mismo lugar y tiempo cercano.

3. **Procesamiento por lotes con Pig:** los datos filtrados son procesados con Apache Pig sobre Hadoop local. Se generan agregaciones por tipo de evento, comuna y fecha, produciendo salidas estructuradas listas para análisis.

4. **Simulación de consultas a caché con Redis:** se utiliza Redis como sistema de caché en memoria, donde un script simula consultas frecuentes a combinaciones tipo–comuna, siguiendo distribuciones Zipf y uniforme. Esto permite evaluar el rendimiento del sistema en un escenario de acceso realista.

5. **Indexación en Elasticsearch:** los archivos resultantes del preprocesamiento, procesamiento por Pig y simulación de caché son cargados a Elasticsearch, creando índices separados para cada tipo de análisis.

6. **Visualización con Kibana:** finalmente, Kibana permite explorar y visualizar los datos procesados, construir dashboards y aplicar filtros interactivos para extraer información significativa

2.2. Justificacion de tecnologias utilizadas

Componente	Tecnología	Justificación técnica
Datos sintéticos	Python (CSV)	Permite generar volúmenes realistas con control sobre distribución y calidad.
Preprocesamiento	Pandas	Potente para limpieza y transformación rápida de datos tabulares.
Procesamiento	Apache Pig	Ideal para tareas de agregación en grandes volúmenes, compatible con Hadoop.
Caché	Redis	Caché en memoria muy eficiente; permite TTL y políticas LRU, ideal para simular tráfico.
Almacenamiento	Elasticsearch	Motor de búsqueda indexado, ideal para consultas estructuradas y escalables.
Visualización	Kibana	Se integra con Elasticsearch; permite dashboards interactivos y filtros dinámicos.
Orquestación	Docker	Facilita ejecución aislada y simultánea de todos los servicios.

Cuadro 1: Tecnologías utilizadas y su justificación técnica

2.3. Diagrama de arquitectura del sistema distribuido

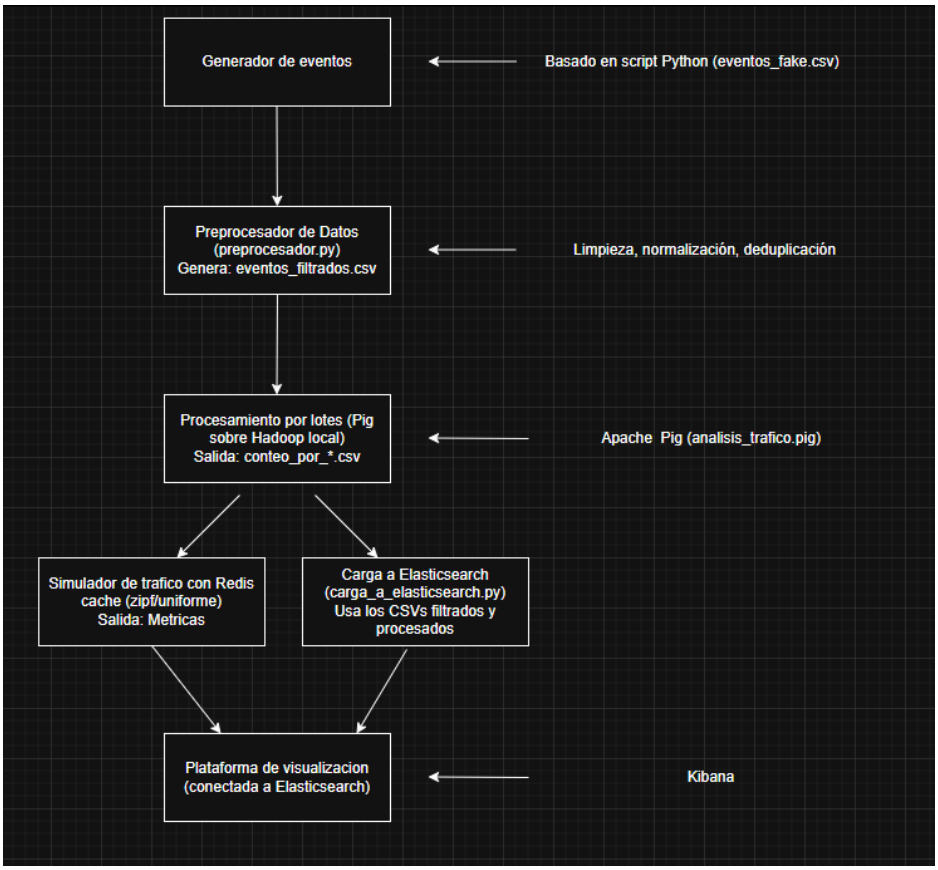

Figura 1: Diagrama de arquitectura del sistema distribuido

Flujo	Descripción
1	El generador crea eventos sintéticos realistas.
2	El preprocesador limpia, normaliza y deduplica los eventos.
3	Pig procesa los eventos filtrados y genera conteos por tipo, comuna, fecha.
4	Redis simula tráfico consultando eventos y entrega métricas de caché.
5	Elasticsearch indexa todos los CSVs (Pig y Redis).
6	Kibana visualiza y explora los datos indexados.

Cuadro 2: Flujo del sistema distribuido implementado

3. Generacion y preprocesamiento de datos

3.1. Generacion de eventos sinteticos

Debido a las limitaciones del scraping web en la tarea anterior —como la baja densidad de eventos disponibles en el mapa en vivo de Waze y restricciones dinámicas del sitio web— se optó por utilizar un generador de eventos sintéticos realistas, desarrollado en Python. Este generador simula eventos de tráfico urbanos en la Región Metropolitana, distribuidos en tiempo, tipo y comuna.

Se consideraron 10 tipos de evento comunes en Waze, asignando probabilidades de ocurrencia mediante pesos personalizados. Además, se definieron 15 comunas de alta circulación vehicular para distribuir los eventos.

Fragmento del generador (generador_fake_eventos.py

```
TIPOS_EVENTO = [
    "Policía", "Accidente", "Atasco", "Corte", "Bache", "Detención",
    "Obstáculo", "Clima", "Manifestación", "Choque múltiple"
]
PESOS = [0.25, 0.2, 0.2, 0.1, 0.05, 0.05, 0.05, 0.04, 0.03, 0.03]

COMUNAS = [
    "Santiago", "Maipú", "Ñuñoa", "Providencia", "Puente Alto",
    "La Florida", "Las Condes", "Macul", "San Bernardo", "Cerro Navia",
    "Pudahuel", "La Reina", "Recoleta", "Quilicura", "Independencia"
]
```

Figura 2: Fragmento de codigo

El generador produce eventos con formato estructurado (tipo, ubicación, timestamp, comuna), los cuales se guardan como eventos_fake.csv. Se generaron un total de 20.000 eventos sintéticos para esta tarea.

Distribución por tipo (resumen de resultados):

Tipo de evento	Cantidad
Policía	4.050
Atasco	3.389
Accidente	3.377
Corte	1.792
Obstáculo	1.008
Detención	933
Bache	909
Clima	776
Choque múltiple	575
Manifestación	569

Cuadro 3: Cantidad de eventos por tipo

Este resultado valida que la distribución fue respetada y coherente con lo esperado en contextos urbanos reales.

Distribución temporal (eventos por fecha):

Fecha	Cantidad
2025-06-24	987
2025-06-25	3.509
2025-06-26	3.473
2025-06-27	3.489
2025-06-28	3.455
2025-06-29	2.465

Cuadro 4: Cantidad de eventos generados por día

Los eventos se generaron con una dispersión temporal uniforme en los últimos 5 días.

Distribucion por comuna

Comuna	Cantidad
Providencia	1.190
La Florida	1.187
Recoleta	1.179
Santiago	1.174
Puente Alto	1.169
Quilicura	1.168
Las Condes	1.182
San Bernardo	1.181
Independencia	1.149
Cerro Navia	1.146
Macul	1.142
Maipú	1.170
Ñuñoa	1.104
La Reina	1.113
Pudahuel	1.124

Cuadro 5: Cantidad de eventos por comuna

Esta distribución muestra un reparto equitativo entre comunas, simulando una cobertura territorial completa de eventos urbanos.

3.2. Preprocesamiento de datos

Una vez generados los eventos, se aplicó un proceso de limpieza, normalización y deduplicación utilizando Pandas en el script preprocesador.py.

Principales pasos:

- Eliminación de filas nulas: se eliminaron registros sin valores en campos clave (tipo, ubicacion, timestamp, comuna).
- Normalización de tipos: se unificaron variantes textuales (e.g., “vehículo detenido” → “Detención”).
- Conversión de timestamp: se estandarizó el campo a formato datetime.
- Deduplicación por ventana temporal: se eliminaron eventos con el mismo tipo y comuna ocurridos en un rango de 5 minutos para evitar redundancia artificial.

Resultados del filtrado

Archivo	Eventos
`eventos_fake.csv`	20.000
`eventos_filtrados.csv`	(ej. ~17.000)*

Cuadro 6: Cantidad de eventos por archivo

4. Procesamiento distribuido con Apache Pig

Una vez preprocesados los eventos sintéticos, se utilizó Apache Pig para realizar un procesamiento por lotes y generar agregaciones útiles para el análisis posterior. Pig permite ejecutar tareas de transformación de datos de manera declarativa sobre el framework Hadoop, ideal para ambientes distribuidos.

4.1. Objetivo del procesamiento

El propósito de esta etapa fue generar archivos de salida con conteos agrupados por:

- Tipo de evento
- Comuna
- Fecha (extraida del campo timestamp)

Esto permite posteriormente crear visualizaciones claras y realizar análisis comparativos de patrones urbanos.

4.2. Script utilizado (analisis_trafico.pig)

processing > analisis_trafico.pig

```
-- Cargar el archivo CSV limpio
eventos = LOAD '/opt/pigjob/data/eventos_filtrados.csv'
    USING PigStorage(',')
    AS (tipo:chararray, ubicacion:chararray, timestamp:chararray, comuna:chararray);

-- Contar cantidad de eventos por tipo
por_tipo = GROUP eventos BY tipo;
conteo_por_tipo = FOREACH por_tipo GENERATE group AS tipo, COUNT(eventos) AS cantidad;

-- Contar cantidad de eventos por comuna
por_comuna = GROUP eventos BY comuna;
conteo_por_comuna = FOREACH por_comuna GENERATE group AS comuna, COUNT(eventos) AS cantidad;

-- Contar eventos por día (extraemos solo la fecha del timestamp)
eventos_con_fecha = FOREACH eventos GENERATE tipo, comuna, SUBSTRING(timestamp, 0, 10) AS fecha;
por_fecha = GROUP eventos_con_fecha BY fecha;
conteo_por_fecha = FOREACH por_fecha GENERATE group AS fecha, COUNT(eventos_con_fecha) AS cantidad;

-- Guardar los resultados
STORE conteo_por_tipo INTO '/opt/pigjob/pig/output/conteo_por_tipo' USING PigStorage(',');
STORE conteo_por_comuna INTO '/opt/pigjob/pig/output/conteo_por_comuna' USING PigStorage(',');
STORE conteo_por_fecha INTO '/opt/pigjob/pig/output/conteo_por_fecha' USING PigStorage(',');

```

Figura 3: Codigo analisis_trafico.pig

4.3. Resultados obtenidos

El procesamiento genero tres archivos clave:

- conteo_por_tipo.csv

- conteo_por_comuna.csv
- conteo_por_fecha.csv

Estos fueron posteriormente indexados en Elasticsearch para su visualización.

A continuación, se muestran las visualizaciones obtenidas a partir de los resultados:

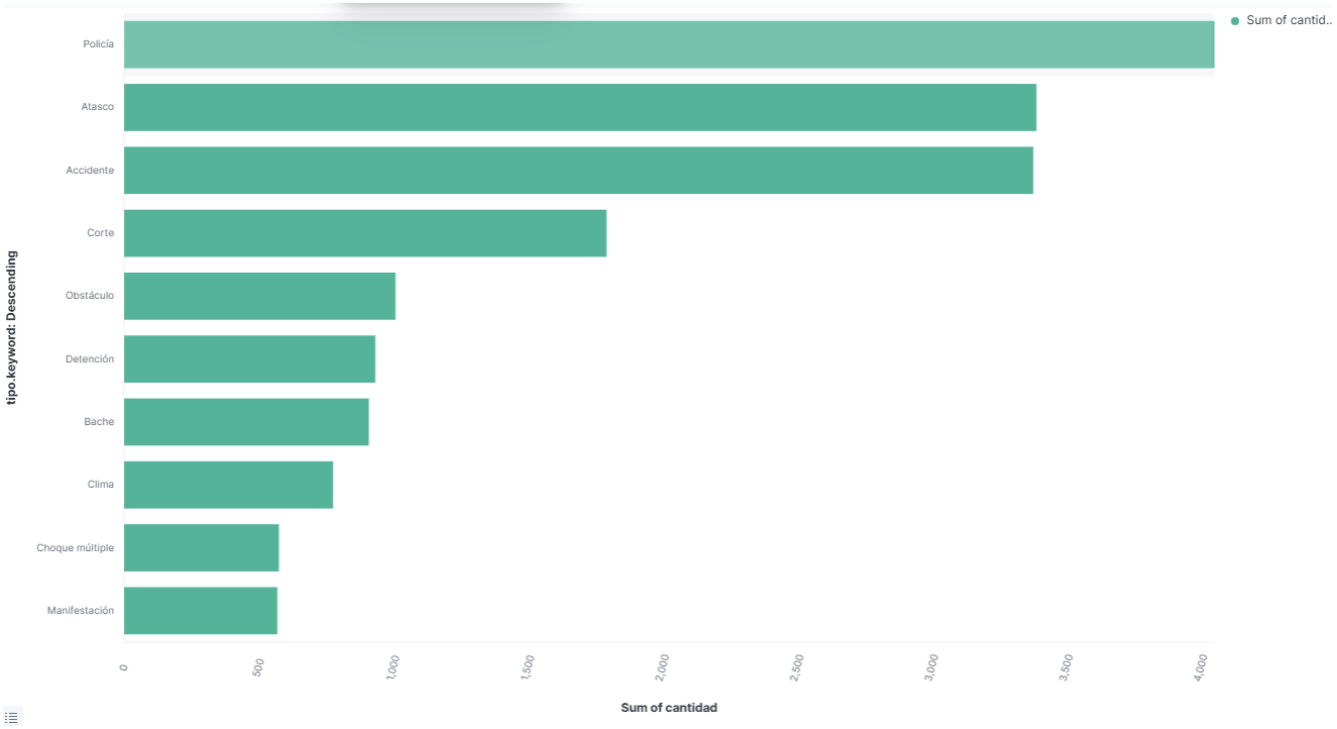

Figura 4: Eventos por tipo

Se observa que los tipos más frecuentes son “Policía”, “Atasco” y “Accidente”, representando en conjunto más del 60 % del total de eventos. Este resultado es coherente con las distribuciones configuradas en el generador sintético.

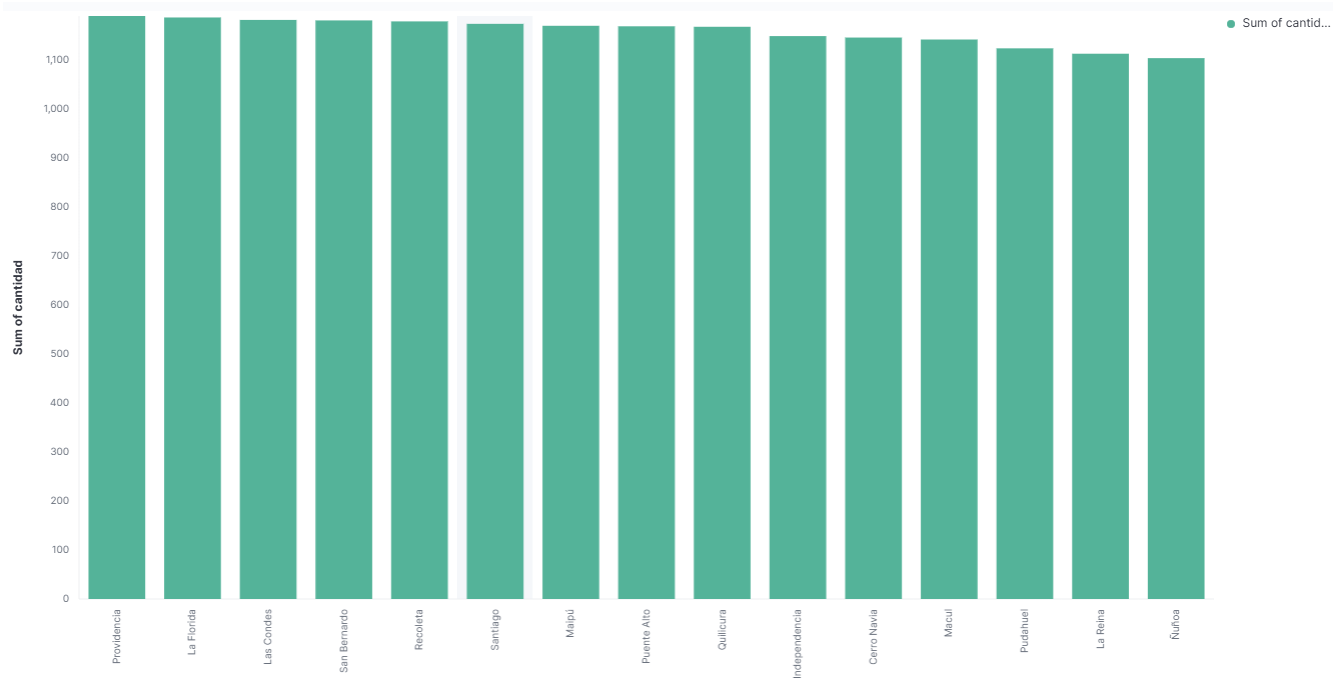

Figura 5: Eventos por comuna

La distribución entre comunas es relativamente homogénea, ya que el generador asigna eventos aleatoriamente a 15 comunas seleccionadas. Esto permite realizar comparaciones equitativas entre zonas urbanas.

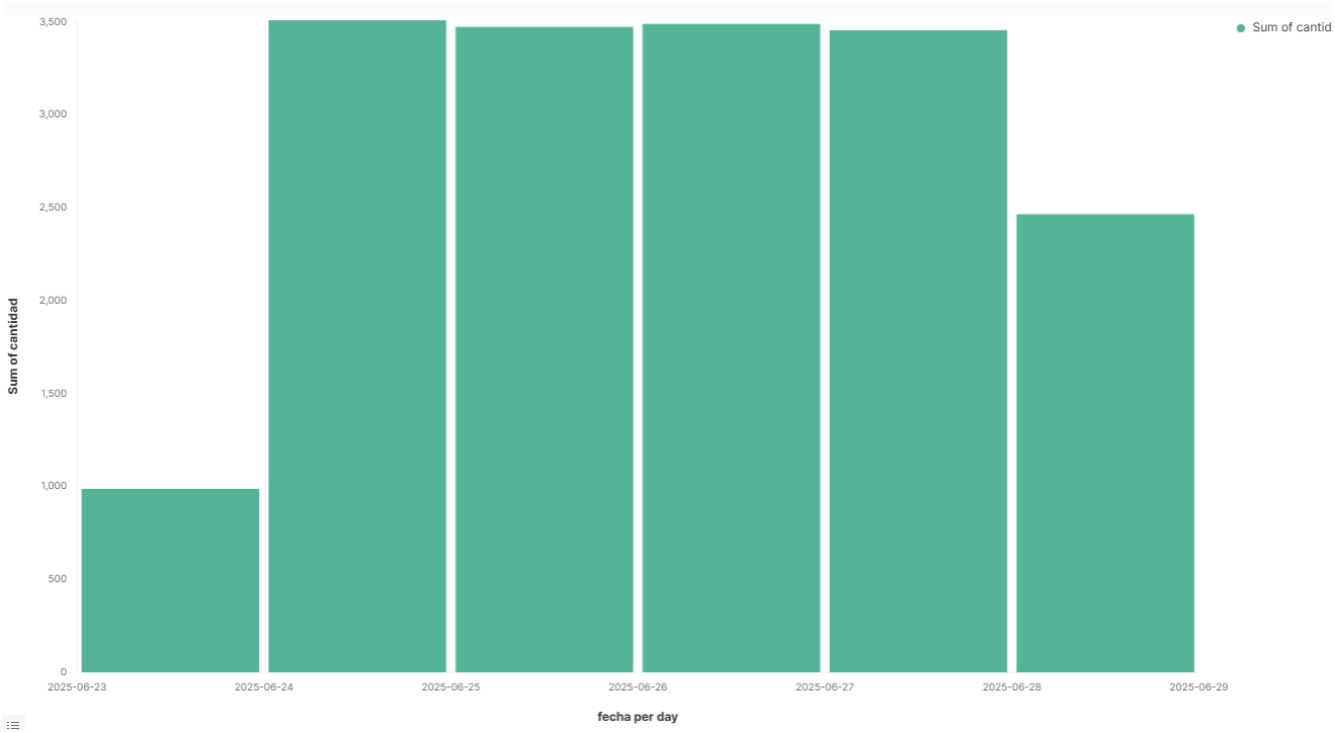

Figura 6: Eventos por fecha

El volumen diario de eventos es estable, con una ligera disminución hacia el último día. Esto refleja una dispersión temporal balanceada y útil para evaluar tendencias por fecha.

5. Simulacion de cache con Redis

Uno de los aspectos clave en un sistema distribuido orientado a consulta eficiente de datos es el uso de caché en memoria. Para simular este componente, se utilizó Redis, una base de datos clave-valor de alto rendimiento. Se desarrolló un script en Python que simula tráfico de consultas sobre los eventos procesados, permitiendo medir el comportamiento del caché bajo distintas condiciones de acceso.

5.1. Descripcion del funcionamiento

El script generador_cache_redis.py simula una carga de 14.000 consultas a Redis, accediendo a combinaciones tipo–comuna. Soporta dos distribuciones de acceso:

- Uniforme: todas las combinaciones tienen la misma probabilidad
- Zipf: simula un patrón realista donde pocas claves concentran la mayoría de los accesos (ley de potencias)

```
46          if r.exists(clave):
47              _ = json.loads(r.get(clave))
48              hits += 1
49          else:
50              resultados = df[(df["tipo"] == tipo) & (df["comuna"] == comuna)]
51              data = resultados.to_dict(orient="records")
52              r.set(clave, json.dumps(data))
53              r.expire(clave, TTL_SEGUNDOS)
54              misses += 1
```

Figura 7: Fragmento del codigo relevante

Cada clave es almacenada con un TTL de 60 segundos, y Redis está configurado con una política allkeys-lru, permitiendo que las claves menos usadas sean descartadas primero cuando se alcanza el límite de memoria.

5.2. Metricas Obtenidas

El resultado de la simulación queda almacenado en el archivo resultados_cache.csv, que incluye las siguientes métricas clave:

Métrica	**Uniforme**	**Zipf**
Consultas totales	14.000	14.000
Hits	13.850	13.887
Misses	150	113
Hit rate (%)	98.93 %	99.19 %
Tiempo promedio (s)	0.0012	0.0012

Cuadro 7: Comparación de rendimiento entre distribuciones Uniforme y Zipf

Ambas distribuciones muestran un desempeño muy alto, con tiempos de respuesta consistentes y tasas de aciertos superiores al 98 %.

5.3. Distribucion de frecuencias claves

Además, se generó el archivo frecuencia_claves.csv que contiene las combinaciones tipo–comuna más consultadas durante la simulación. Este archivo es útil para detectar qué zonas urbanas presentan mayor demanda en el sistema.

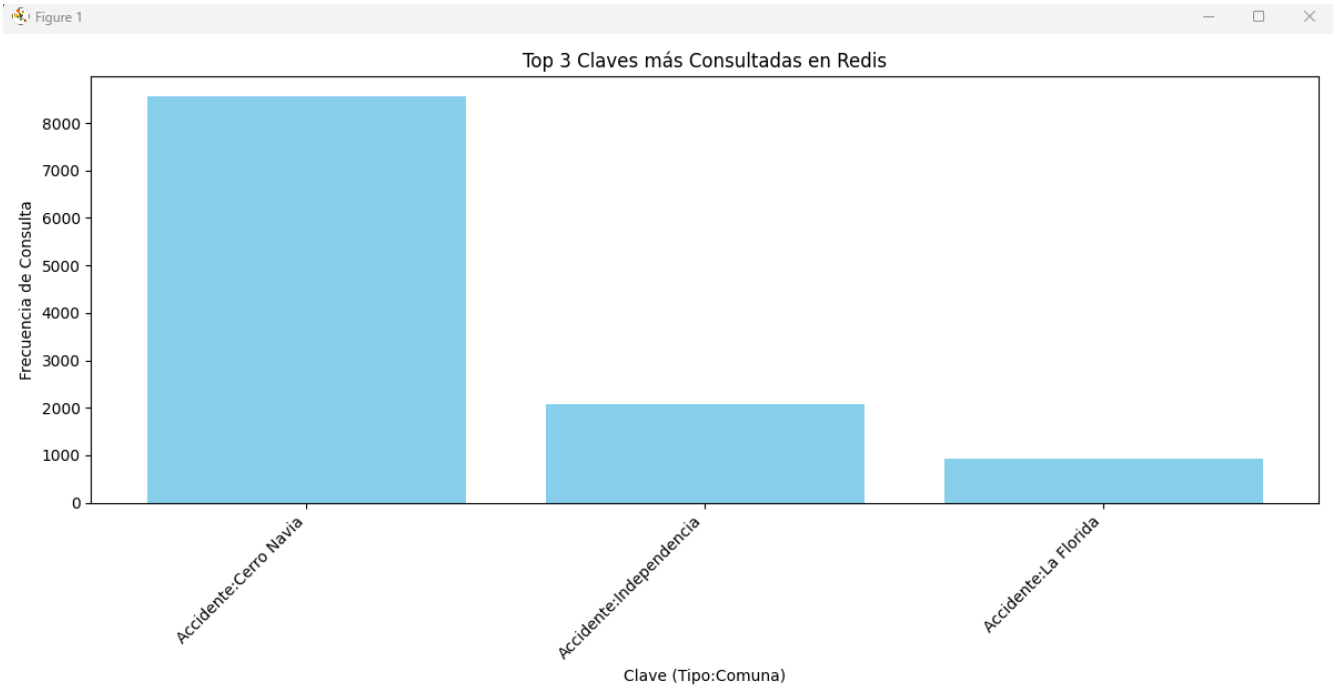

Figura 8: Claves más consultadas en Redis (frecuencia acumulada por combinación tipo–comuna)

6. Visualizacion con Kibana

Una vez obtenidos los resultados del procesamiento con Pig y la simulación con Redis, se procedió a cargar los datos a Elasticsearch, un motor de búsqueda y análisis distribuido que permite realizar consultas estructuradas de forma rápida. La visualización de estos datos se realizó mediante Kibana, una herramienta que se integra nativamente con Elasticsearch y permite crear dashboards interactivos.

6.1. Carga de datos a Elasticsearch

Para indexar los archivos .csv generados (conteo_por_tipo.csv, conteo_por_comuna.csv, conteo_por_fecha.csv, resultados_cache.csv, etc.), se utilizó el script carga_a_elasticsearch.py, el cual emplea la API bulk de Elasticsearch para insertar grandes volúmenes de datos eficientemente.

```
if "cantidad" in doc:
    try:
        doc["cantidad"] = int(doc["cantidad"])
```

Figura 9: Fragmento código cargar a Elasticsearch

Esta conversión forzada permite que Kibana reconozca el campo cantidad como tipo numérico y lo utilice en agregaciones (sum, avg, etc.). De igual forma, el campo fecha se formatea correctamente (YYYY-MM-DD) para permitir histogramas temporales

6.2. Creacion de dashboards en Kibana

Una vez cargados los índices, se crearon Index Patterns en Kibana para los siguientes conjuntos de datos:

Índice	Campo de tiempo	Campos clave
`conteo_por_tipo`	ninguno	`tipo`, `cantidad`
`conteo_por_comuna`	ninguno	`comuna`, `cantidad`
`conteo_por_fecha`	`fecha`	`fecha`, `cantidad`

Cuadro 8: Índices generados en Elasticsearch y sus campos clave

Con estos patrones, se construyeron tres visualizaciones principales:

- Grafico de barras: eventos por tipo
- Grafico de barras: eventos por comuna
- Grafico de barras: eventos por fecha

6.3. Dashboard final

Todas las visualizaciones fueron agrupadas en un único dashboard interactivo, que permite aplicar filtros cruzados por comuna, tipo de evento y fecha. Este dashboard permite al usuario realizar un análisis visual completo y dinámico del comportamiento urbano simulado.

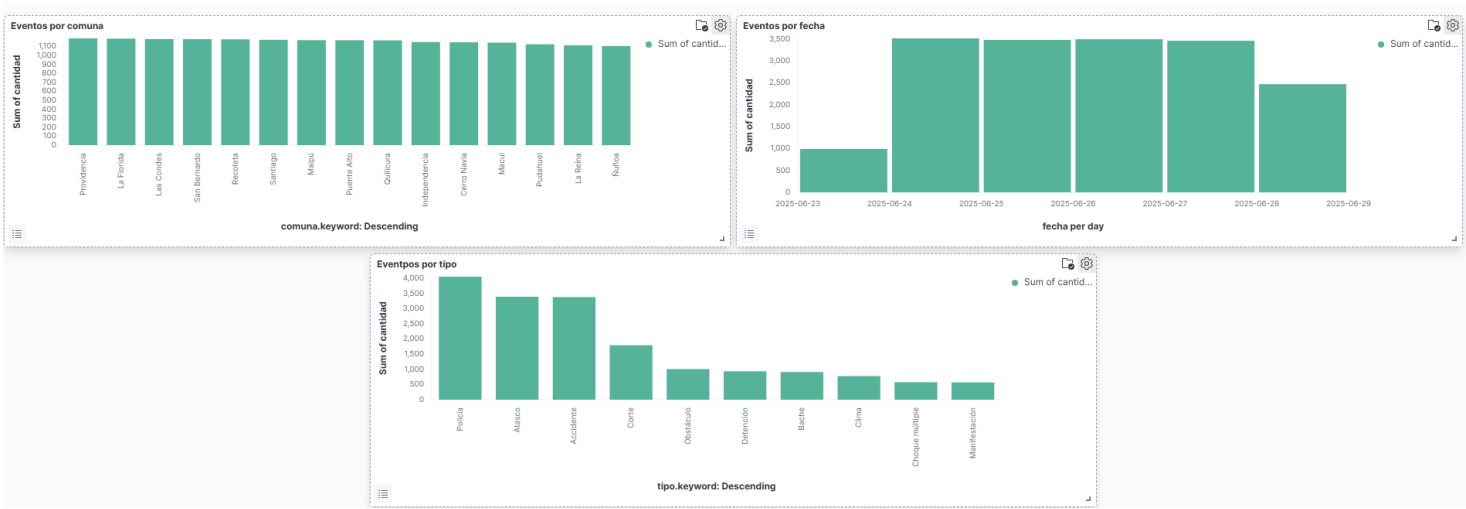

Figura 10: Dashboard interactivo en Kibana consolidando eventos por tipo, comuna y fecha.

7. Conclusión

El desarrollo de esta tarea permitió diseñar, implementar y evaluar un sistema distribuido completo, enfocado en la recolección, procesamiento, análisis y visualización de eventos urbanos simulados. A pesar de las limitaciones presentadas en la obtención de datos reales, el uso de eventos sintéticos realistas —validados con distribuciones y visualizaciones coherentes— permitió simular condiciones comparables a las de un sistema en producción.

El uso de herramientas como Apache Pig, Redis, Elasticsearch y Kibana permitió cubrir distintas etapas de un flujo típico en sistemas distribuidos de datos: desde el procesamiento por lotes hasta la visualización interactiva, pasando por optimización en memoria con caché y despliegue mediante contenedores. La integración modular a través de Docker facilitó la construcción de una solución flexible, extensible y reproducible.

Se evidenció el valor de aplicar mecanismos de caché en consultas repetitivas, logrando tasas de aciertos superiores al 98 %, y se visualizaron patrones de comportamiento espacial y temporal mediante dashboards interactivos en Kibana.

Como trabajo futuro, se sugiere profundizar en el análisis en tiempo real mediante tecnologías como Apache Kafka o Spark Streaming, y avanzar hacia el uso de datos reales mediante scrapers más robustos o acceso a APIs especializadas.

8. Anexos

Repositorio GitHub en donde se encuentran todos los códigos utilizados

Aquí está el repositorio: https://github.com/PaolooG/Tarea03_SD_final.git

Enlace a video por drive

Enlace: https://drive.google.com/file/d/1WGXiO-1L-LxEGQ33aUab0zQHrimjI0qP/view?usp=drive_link