Implementación de ID de Traza de Extremo a Extremo

Esta entrada del blog fue escrita con la asistencia de ChatGPT-4o.

Trabajé en una solución de ID de seguimiento de extremo a extremo para garantizar que cada solicitud y respuesta en nuestro sistema pueda rastrearse de manera consistente entre el frontend y el backend. Esta solución ayuda en la depuración, monitoreo y registro al asociar cada operación con un ID de seguimiento único. A continuación, se presenta una explicación detallada de cómo funciona la solución, junto con ejemplos de código.

Cómo Funciona

Frontend

La parte del frontend de esta solución implica generar un ID de seguimiento para cada solicitud y enviarlo junto con la información del cliente al backend. Este ID de seguimiento se utiliza para rastrear la solicitud a través de varias etapas de procesamiento en el backend.

1. Recopilación de Información del Cliente: Recopilamos información relevante del cliente, como las dimensiones de la pantalla, el tipo de red, la zona horaria y más. Esta información se envía junto con los encabezados de la solicitud.
2. Generación de ID de Traza: Se genera un ID de traza único para cada solicitud. Este ID de traza se incluye en los encabezados de la solicitud, lo que nos permite rastrear la solicitud a lo largo de su ciclo de vida.
3. API Fetch: La función `apiFetch` se utiliza para realizar llamadas a la API. Incluye el ID de seguimiento y la información del cliente en los encabezados de cada solicitud.

Backend

La parte del backend de la solución implica registrar el ID de traza con cada mensaje de registro e incluir el ID de traza en las respuestas. Esto nos permite rastrear las solicitudes a través del procesamiento del backend y hacer coincidir las respuestas con las solicitudes.

1. Manejo del Trace ID: El backend recibe el Trace ID desde los encabezados de la solicitud o genera uno nuevo si no se proporciona. El Trace ID se almacena en un objeto global de Flask para su uso durante todo el ciclo de vida de la solicitud.
2. Registro (Logging): Se utilizan formateadores de registros personalizados para incluir el ID de seguimiento en cada mensaje de registro. Esto garantiza que todos los mensajes de registro relacionados con una solicitud puedan correlacionarse utilizando el ID de seguimiento.
3. Manejo de la Respuesta: El ID de traza se incluye en los encabezados de la respuesta. Si ocurre un error, el ID de traza también se incluye en el cuerpo de la respuesta de error para ayudar en la depuración.

Kibana

Kibana es una interfaz de visualización de datos de código abierto diseñada para trabajar con Elasticsearch. Proporciona capacidades de visualización y exploración de datos, permitiendo a los usuarios crear gráficos, mapas y tablas interactivas a partir de los datos almacenados en Elasticsearch. Kibana es ampliamente utilizado en el análisis de registros (log analysis), monitoreo de aplicaciones y otras tareas relacionadas con la exploración de datos en tiempo real.

Características principales de Kibana:

1. **Visualización de datos**: Kibana permite crear gráficos, tablas y mapas interactivos para representar datos de manera clara y comprensible.
2. **Dashboards personalizables**: Los usuarios pueden crear paneles personalizados que combinan múltiples visualizaciones para monitorear y analizar datos en tiempo real.
3. **Exploración de datos**: Con la función “Discover”, los usuarios pueden explorar y filtrar datos almacenados en Elasticsearch de manera interactiva.
4. **Integración con Elasticsearch**: Kibana está diseñado para funcionar de manera nativa con Elasticsearch, lo que facilita la consulta y visualización de grandes volúmenes de datos.
5. **Alertas y monitoreo**: Kibana permite configurar alertas basadas en condiciones específicas de los datos, lo que es útil para el monitoreo proactivo de sistemas y aplicaciones.

6. **Machine Learning**: Kibana integra herramientas de machine learning para detectar anomalías y patrones en los datos.

Casos de uso comunes:

- **Análisis de registros (Log Analysis)**: Kibana es ampliamente utilizado para analizar y visualizar registros de sistemas y aplicaciones, lo que ayuda a identificar problemas y optimizar el rendimiento.
- **Monitoreo de infraestructura**: Con Kibana, los equipos de operaciones pueden monitorear el estado de servidores, redes y aplicaciones en tiempo real.
- **Análisis de métricas de negocio**: Las empresas pueden utilizar Kibana para visualizar y analizar métricas clave de negocio, como ventas, tráfico web y rendimiento de marketing.
- **Seguridad y cumplimiento**: Kibana se utiliza en la detección de amenazas y el análisis de eventos de seguridad, ayudando a las organizaciones a cumplir con normativas y proteger sus sistemas.

Ejemplo básico de uso:

```
# Iniciar Kibana (asumiendo que Elasticsearch ya está en ejecución)
bin/kibana
```

Una vez que Kibana esté en funcionamiento, puedes acceder a la interfaz web a través de `http://localhost:5601`. Desde allí, puedes comenzar a crear visualizaciones y paneles personalizados.

Conclusión: Kibana es una herramienta poderosa para la visualización y exploración de datos, especialmente cuando se utiliza junto con Elasticsearch. Su facilidad de uso y su capacidad para manejar grandes volúmenes de datos lo convierten en una opción popular para equipos de operaciones, desarrolladores y analistas de datos.

Kibana es una herramienta potente para visualizar y buscar datos de registros almacenados en Elasticsearch. Con nuestra solución de Trace ID, puedes rastrear y depurar solicitudes fácilmente utilizando Kibana. El ID de traza, que se incluye en cada entrada de registro, se puede utilizar para filtrar y buscar registros específicos.

Para buscar registros con un ID de traza específico, puedes utilizar el Lenguaje de Consulta de Kibana (KQL). Por ejemplo, puedes buscar todos los registros relacionados con un ID de traza en particular con la siguiente consulta:

```
trace_id:"Lc6t"
```

Esta consulta devolverá todas las entradas de registro que contengan el ID de traza “Lc6t”, lo que te permitirá rastrear la ruta de la solicitud a través del sistema. Además, puedes combinar esta consulta con otros criterios para reducir los resultados de búsqueda, como filtrar por nivel de registro, marca de tiempo o palabras clave específicas dentro de los mensajes de registro.

Al aprovechar las capacidades de visualización de Kibana, también puedes crear paneles que muestren métricas y tendencias basadas en los IDs de traza. Por ejemplo, puedes visualizar el número de solicitudes procesadas, los tiempos de respuesta promedio y las tasas de error, todo correlacionado con sus respectivos IDs de traza. Esto ayuda a identificar patrones y posibles problemas en el rendimiento y la confiabilidad de tu aplicación.

El uso de Kibana en conjunto con nuestra solución de Trace ID ofrece un enfoque integral para monitorear, depurar y analizar el comportamiento de tu sistema, asegurando que cada solicitud pueda ser rastreada e investigada de manera efectiva.

Frontend

api.js

```
const BASE_URL = process.env.REACT_APP_BASE_URL;
```

// Función para obtener información del cliente const getClientInfo = () => { const { language, platform, cookieEnabled, doNotTrack, onLine } = navigator; const { width, height } = window.screen; const connection = navigator.connection || navigator.mozConnection || navigator.webkitConnection; const networkType = connection ? connection.effectiveType : ‘unknown’; const timeZone = Intl.DateTimeFormat().resolvedOptions().timeZone; const referrer = document.referrer; const viewportWidth = window.innerWidth; const viewportHeight = window.innerHeight;

```
return {
    screenWidth: width,
    screenHeight: height,
```

```
    networkType,
    timeZone,
    language,
    platform,
    cookieEnabled,
    doNotTrack,
    onLine,
    referrer,
    viewportWidth,
    viewportHeight
};

};
```

// Función para generar un ID de traza único export const generateTraceId = (length = 4) => { const characters = ‘ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789’; let traceId = ’ ’; for (let i = 0; i < length; i++) { const randomIndex = Math.floor(Math.random() * characters.length); traceId += characters.charAt(randomIndex); } return traceId; };

```
export const apiFetch = async (endpoint, options = {}) => {
    const url = `${BASE_URL}${endpoint}`;
    const clientInfo = getClientInfo();

const traceId = options.traceId || generateTraceId();

    const headers = {
        'Content-Type': 'application/json',
        'X-Client-Info': JSON.stringify(clientInfo),
        'X-Trace-Id': traceId,
        ...(options.headers || {})
    };

    const response = await fetch(url, {
        ...options,
        headers
    });
```

```
    return response;
};
```

App.js

```
try {
  const response = await apiFetch('api', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(content),
    traceId: traceId
  });

if (response.ok) {
    const data = await response.json();
    //...
} else {
    const errorData = await response.json();
    const errorMessage = errorData.message || 'Ocurrió un error desconocido';
    let errorToastMessage = errorMessage;
    errorToastMessage += ` (Trace ID: ${traceId})`;
    toast.error(errorToastMessage, {
      autoClose: 8000
    });
    setError(errorToastMessage);
}
} catch (error) {
  let errorString = error instanceof Error ? error.message : JSON.stringify(error);
```

const duration = (Date.now() - startTime) / 1000;

if (error.response) { // La solicitud fue realizada y el servidor respondió con un código de estado que no está en el rango de 2xx errorString += `(HTTP ${error.response.status}: ${error.response.statusText})`; console.error('Datos del error en la respuesta:', error.response.data); } else if (error.request) { // La solicitud fue realizada pero no se recibió respuesta errorString

+= ’ (No se recibió respuesta)‘; console.error(’Datos del error en la solicitud:’, error.request); } else { // Algo ocurrió al configurar la solicitud que provocó un Error errorString += `(Error al configurar la solicitud: ${error.message}); }

```
errorString += ` (ID de seguimiento: ${traceId})`;
```

if (error instanceof Error) { errorString += `\nStack: ${error.stack}; }

```
errorString += JSON.stringify(error);

errorString += ` (Duración: ${duration} segundos)`;

toast.error(`Error: ${errorString}`, {
    autoClose: 8000
  });
  setError(errorString);
} finally {
  toast.dismiss(toastId);
}
```

Backend

__init__.py

```
# -*- encoding: utf-8 -*-

import os
import json
import time
import uuid
import string
import random

from flask import Flask, request, Response, g, has_request_context
from flask_cors import CORS

from .routes import initialize_routes
from .models import db, insert_default_config
```

```
import logging
from logging.handlers import RotatingFileHandler
from prometheus_client import Counter, generate_latest, Gauge
from flask_migrate import Migrate
from logstash_formatter import LogstashFormatterV1
```

Traducción al español:

```
from .routes import initialize_routes
from .models import db, insert_default_config
import logging
from logging.handlers import RotatingFileHandler
from prometheus_client import Counter, generate_latest, Gauge
from flask_migrate import Migrate
from logstash_formatter import LogstashFormatterV1
```

Nota: Los nombres de las bibliotecas y funciones no se traducen, ya que son específicos del código y deben mantenerse en inglés para su correcto funcionamiento.

```
app = Flask(__name__)
```

app.config.from_object('api.config.BaseConfig')

db.init_app(app) initialize_routes(app)

CORS(app)

```
migrate = Migrate(app, db)

class RequestFormatter(logging.Formatter):
    def format(self, record):
        if has_request_context():
            record.trace_id = getattr(g, 'trace_id', 'unknown')
        else:
            record.trace_id = 'unknown'
        return super().format(record)

class CustomLogstashFormatter(LogstashFormatterV1):
    def format(self, record):
```

```
        if has_request_context():
            record.trace_id = getattr(g, 'trace_id', 'unknown')
        else:
            record.trace_id = 'unknown'
        return super().format(record)

def setup_loggers():
    logstash_handler = RotatingFileHandler(
        'app.log', maxBytes=100000000, backupCount=1)
    logstash_handler.setLevel(logging.DEBUG)
    logstash_formatter = CustomLogstashFormatter()
    logstash_handler.setFormatter(logstash_formatter)

txt_handler = RotatingFileHandler(
    'plain.log', maxBytes=100000000, backupCount=1)
txt_handler.setLevel(logging.DEBUG)
txt_formatter = RequestFormatter(
    '%(asctime)s %(levelname)s: %(message)s [en %(pathname)s:%(lineno)d] [trace_id: %(trace_id)s]')
txt_handler.setFormatter(txt_formatter)

root_logger = logging.getLogger()
root_logger.setLevel(logging.DEBUG)
root_logger.addHandler(logstash_handler)
root_logger.addHandler(txt_handler)

app.logger.addHandler(logstash_handler)
app.logger.addHandler(txt_handler)

werkzeug_logger = logging.getLogger('werkzeug')
werkzeug_logger.setLevel(logging.DEBUG)
werkzeug_logger.addHandler(logstash_handler)
werkzeug_logger.addHandler(txt_handler)

setup_loggers()

def generar_trace_id(longitud=4):
    caracteres = string.ascii_letters + string.digits
    return ''.join(random.choice(caracteres) for _ in range(longitud))
```

```python
@app.before_request
def before_request():
    request.start_time = time.time()
    trace_id = request.headers.get('X-Trace-Id', generate_trace_id())
    g.trace_id = trace_id

    client_info = request.headers.get('X-Client-Info')
    if client_info:
        try:
            client_info_json = json.loads(client_info)
            logging.info(f"Información del Cliente: {client_info_json}")
        except json.JSONDecodeError:
            logging.warning("Formato JSON inválido para el encabezado X-Client-Info")

@app.after_request
def after_request(response):
    response.headers['X-Trace-Id'] = g.trace_id

if response.status_code != 200:
    logging.error(f'Código de estado de la respuesta: {response.status_code}')
    logging.error(f'Cuerpo de la respuesta: {response.get_data(as_text=True)}')

    if response.content_type == 'application/json':
        try:
            response_json = response.get_json()
            response_json['trace_id'] = g.trace_id
            response.set_data(json.dumps(response_json))
        except Exception as e:
            logging.error(f"Error al agregar trace_id a la respuesta: {e}")

    return response
```

Registro

Puedes buscar todos los registros relacionados con un ID de traza específico utilizando la siguiente consulta:

trace_id:"Lc6t"


```
{
  "_index": "flask-logs-2024.07.05",
  "_type": "_doc",
  "_id": "Ae9zgZABqOMSOpxCZC5X",
  "_version": 1,
  "_score": 1,
  "_source": {
    "tags": [
      "_grokparsefailure"
    ],
    "filename": "generate.py",
    "funcName": "post",
    "message": "Solicitud procesada exitosamente",
    "@version": 1,
    "name": "root",
    "host": "ip-172-31-35-xxx.ec2.internal",
    "relativeCreated": 685817.8744316101,
    "levelname": "INFO",
    "created": 1720158740.894831,
    "thread": 139715118360128,
    "threadName": "Thread-5",
    "levelno": 20,
    "pathname": "/home/project/project-name/api/routes/generate.py",
    "msecs": 894.8309421539307,
    "processName": "MainProcess",
    "lineno": 287,
    "path": "/home/project/project-name/app.log",
    "args": [],
    "source_host": "ip-172-31-35-xxx.ec2.internal",
    "module": "generate",
    "trace_id": "Lc6t",
    "stack_info": null,
    "process": 107613,
    "@timestamp": "2024-07-05T05:52:20.894Z"
  },
```

```
"fields": {
  "levelname.keyword": [
    "INFO"
  ],
  "tags.keyword": [
    "_grokparsefailure"
  ],
  "relativeCreated": [
    685817.9
  ],
  "processName.keyword": [
    "MainProcess"
  ],
  "filename.keyword": [
    "generate.py"
  ],
  "funcName": [
    "post"
  ],
  "path": [
    "/home/project/project-name/app.log"
  ],
  "processName": [
    "MainProcess"
  ],
  "@version": [
    1
  ],
  "host": [
    "ip-172-31-35-xxx.ec2.internal"
  ],
  "msecs": [
    894.83093
  ],
  "source_host.keyword": [
    "ip-172-31-35-xxx.ec2.internal"
```

```
],
"host.keyword": [
  "ip-172-31-35-xxx.ec2.internal"
],
"levelname": [
  "INFO"
],
"process": [
  107613
],
"threadName.keyword": [
  "Thread-5"
],
"trace_id": [
  "Lc6t"
],
"source_host": [
  "ip-172-31-35-xxx.ec2.internal"
],
"created": [
  1720158700
],
"module": [
  "generate"
],
"module.keyword": [
  "generate"
],
"name.keyword": [
  "root"
],
"thread": [
  139715118360128
],
"message": [
  "Solicitud procesada exitosamente"
```

```
],
"levelno": [
  20
],
"trace_id.keyword": [
  "Lc6t"
],
"threadName": [
  "Thread-5"
],
"pathname": [
  "/home/project/project-name/api/routes/generate.py"
],
"tags": [
  "_grokparsefailure"
],
"pathname.keyword": [
  "/home/project/project-name/api/routes/generate.py"
],
"@timestamp": [
  "2024-07-05T05:52:20.894Z"
],
"filename": [
  "generate.py"
],
"lineno": [
  287
],
"message.keyword": [
  "Solicitud procesada exitosamente"
],
"name": [
  "root"
],
"funcName.keyword": [
  "post"
```

```
    ],
    "path.keyword": [
      "/home/project/project-name/app.log"
    ]
  }
}
```

Como se muestra arriba, puedes ver el ID de traza en el registro.