ADR - Solución de logs y observabilidad

ADR-BACKEND-TRACE-LOGS-TELEMETRY

Estado	VIGENTE			
TAGS				
Tomada el	PRE-RELEASE: 1 jun 2024 / RELEASE: 1 jul 2024 / RETIRO: - Version: 4 / Date: 05-07-02024 / Author: LUIS MARTINEZ FONTIVEROS / Comment:			
Responsable	LUIS MARTINEZ FONTIVEROS ALBERTO FUENTES GOMEZ			
Documentación Asociada				
Solución Afectada	Todas las nuevas soluciones software a partir de la fecha de decisión.			
	Responsable	Aprobado por	Consultados	Informados
	DESARROLLO SOFTWARE RRHH (AYESA) ALBERTO FUENTES GOMEZ	LUIS MARTINEZ FONTIVEROS	DESARROLLO SOFTWARE RRHH (AYESA) CARLOS GONZALEZ SANTIAGO	DESARROLLO SOFTWARE RRHH (AYESA) DESARROLLO SOFTWARE ECONOMICO-FINANCIERO (NTTDATA) DESARROLLO SOFTWARE ASISTENCIAL (NTTDATA) DESARROLLO SOFTWARE RRHH (AYESA) AREA GOBIERNO TECNOLOGICO AREA SISTEMAS

Asunto a decidir

Se Requiere una decisión respecto a la estrategia a seguir con respecto a la gestión de logs, en primera instancia, y a toda observabilidad en el escenario objetivo.

Contexto

Analisis de contexto y posiblidades previas

Hay que tener en cuenta algunos elementos de alto impacto en la solución que se plantee:

1. Debemos registrar los logs generados por *la solución entregada.*
2. Asumamos *la solución entregada* debería estar como máximo entre un 70 y 80% de su capacidad en los picos de carga con el objeto de tener margen de crecimiento.
3. Los logs deberían conservarse al menos 7 días (no tendría sentido periodos inferiores ya que podemos perder información muy importante). Nos podemos hacer una idea bastante clara de la cantidad de Gb de información que supone esto.

Ahora vamos a aquellas **cosas que NO podemos hacer por poner en riesgo la estabilidad del sistema**:

- **Registrar los logs en los en el sistema** se archivos de *la solución entregada*. Si observamos el punto 3 nos quedaremos rápidamente sin sitio y tumbaremos el sistema; eso está descartado, se considera mala praxis en contenedores.
- **Registrar los logs en la BBDD de *la solución entregada***. De nuevo el punto 3 aparece como una condena ¿cuanto tardaremos que dejar sin espacio al namespace de la BBDD?
- ***La solución entregada* registra la información en un volumen persistente**. Esto implica que que una herramienta que está al 70-80% de su capacidad debe administrar también los logs y los ficheros asociados.

Estos puntos descartan el uso de los recursos de *la solución entregada* para la tarea de gestión de logs. Dicho de otra forma, tenemos entonces estas restricciones:

- La información debe salir de los pods
- Cuanta mas carga de trabajo en *la solución entregada* mas logs se generan y si los administra la propia *solución entregada* mas carga que debe asumir, por lo que no debe realizar esa tarea
- En el caso de una solución ya finalizada o un producto el control sobre el código y por lo tanto de la instrumentación es limitado.

Teniendo claro que debemos sacar la información de *la solución entregada* mediante otro aplicativo tenemos las siguientes cuestiones:

1. ¿Qué aplicativo usamos?
2. ¿Dónde registramos la información generada?

Para el punto 1 tenemos las siguientes opciones como alternativas viables:

- **OpenTelemetry**:
  - La aplicación ya existe: Otel- javaagent
  - Es estable.
  - Forma de trabajar:
    - Tiene que enviar la información a un Collector de OpenTelemetry.
    - El collector es un pod externo a *la solución entregada* y no impactará en su rendimiento
    - Desde el Collector se puede exportar a donde se desee ya que posee numerosos exportadores ya desarrollados.
  - Consecuencia:
    - Nos acerca al objetivo final por lo que aprovechamos todo lo que se haga
- **kafka producer**:
  - La aplicación no existe
  - La aplicación debería leer de la consola los logs y enviarlos a un Topic kafka.
  - Forma de trabajar:
    - El nuevo desarrollo básicamente debe funcionar como Opentelemetry ya que no queremos ni mas carga ni bloqueos por la gestión de ficheros, es decir como un aplicativo ejecutándose en el mismo pod que *la solución* y leyendo de la consola.
    - La información leída se debe mandar a kafka con una garantía de at-least-once y debemos tener un kafka con un topic optimizado para asumir la carga de trabajo (particiones/replicas/ políticas de retención etc).
  - Consecuencias:
    - Hablamos de un nuevo desarrollo que es complejo.
    - Igualmente necesitamos de una infraestructura para dar soporte, en este caso kafka
    - Todo lo que hagamos se va a desechar en cuanto se levante la observabilidad con opentelemetry
    - En cuanto tengamos la observabilidad de CTI deberemos implementar igualmente la opción 1.- Opentelemetry

Estas dos alternativas envían la información a un elemento de infraestructura mediante un protocolo optimizado (habitualmente grpc) **lo que nos daría el rendimiento que buscamos**

Evaluemos ahora qué hacer con la información registrada:

- **Collector OpenTelemetry**
  - **Ya dispone de exportadores**: esto significa que podemos elegir qué hacer, desde enviarla a consola, a un fichero, enviarla otro collector, un elastic, kafka... . Tenemos una amplia variedad de exportadores que ya están disponibles.
- **Kafka**
  - **Es necesario crear consumidores** para poder sacar la información
  - También podemos optar por **dejar la información en kafka** para lo cual **es necesario aprovisionar espacio suficiente en Kafka** para asumir la información de logs que va a llegar
  - Si dejamos la información en kafka **hay que asumir el problema de consultarla**, lo que de nuevo nos lleva a la necesidad de **un consumidor a desarrollar**

Antes de entrar en detalle hay que contar con que **la observabilidad en el escenario objetivo se plantea basada en el estándar Opentelemetry.**

> **OpenTelemetry**
>
> OpenTelemetry es una colección de herramientas, API y SDK que ayudan a desarrollar y exportar datos de observabilidad, como trazas, métricas y registros, de manera estandarizada y compatible con múltiples sistemas de monitoreo y análisis. Diseñado para ser un estándar abierto

Se proponen (2) opciones para no saturar el sistema de gestión de *la solución entregada*:

- implementar Kafka,
- implementar el OpenTelemetry de CTI

Alternativas consideradas

Las alternativas se consideran en tres aspectos principales además de los aspectos secundarios que se consideren oportunos:

1. Debe ser una alternativa viable
2. Se debe minimizar el impacto sobre la funcionalidad propia de *la solución entregada*

3. Debe acercarnos en la medida de lo posible al escenario objetivo

Kafka

Kafka es un sistema distribuido altamente resiliente que permite tanto el registro de información siguiendo un patrón CQRS como una comunicación asíncrona mediante PUB/SUB.

En este caso de uso se debe disponer de un productor que permite la ingesta de los logs generados por la aplicación sobre la que se aplica la observación, publicando los registros en un topic de kafka para que sean consumidos en el sistema que deseemos utilizar para almacenar, indexar y/o consultar la información. Esto implica un productor que debe desplegarse en el Pod junto a la aplicación que se está monitorizando o que tenga acceso a los logs de dicha aplicación

Ventajas

- Garantía de entrega
- Resiliencia (información replicada entre brokers)
- Cercano al tiempo real
- Desacoplamiento entre productor y consumidor de los logs

Desventajas

- No nos acerca al escenario objetivo
- Requiere un desarrollo específico (productor, consumidor y cualquiera otra pieza necesaria)
- Requerirá retrabajos para llegar a la arquitectura objetivo

OpenTelemetry

Opentelemetry dispone de integración con distintos lenguajes mediante agentes. También está disponible la integración de Opentelemetry con Quarkus, en este caso sólo dispone de soporte a traza por lo que no cubre ni el MVP ni la plataforma objetivo.

Opentelemetry contempla dos opciones para la instrumentalización: **Code-Based**, aplicable a desarrollos donde se tiene el control total sobre el código, y **Zero-Coded** donde las funcionalidades desarrolladas para opentelemetry extraen la información de librerías y entornos de ejecución. Actualmente la opción Zero-coded está disponible para las tecnologías basadas en los siguientes lenguajes de programación:

- .NET
- Go

- Java
- JavaScript
- PHP
- Python

Para establecer la observabilidad de herramientas desarrolladas en alguna de estas tecnologías sin impactar su implementación es necesario ir a la solución Zero-Coded.

Opentelemetry Zero-coded

Ventajas

- Está totalmente alineado con la plataforma de observabilidad objetivo
- Nos permite extraer toda la observabilidad, tanto los logs como métricas y trazas
- Evitaremos retrabajos futuros
- No usa los recursos propios de *la solución entregada* por lo que no incrementa su carga de trabajo

Desventajas

- Es necesario crear una imagen que además de *la solución entregada* incluya el agente de OpenTelemetry
- Requerimos de los collectores de CTI, de no existir sería necesario aprovisionar un collector que exporte al sistema destino que se considere oportuno

Decisión tomada

En base al análisis de las opciones planteada **se concluye** que la mejor alternativa es **el uso de OpenTelemetry mediante la instrumentación con opentelemetry-javaagent**

Todas las opciones analizadas diferentes de Opentelémetry o bien amenazan la estabilidad del sistema o bien implican sobrecostes y desarrollos no reutilizables, por lo que los sobrecostes no se recuperarán.

Tabla de opciones/riesgos

	Alternativas	Riesgos	N. riesgo	Observaciones	Agravantes /dependencias	Mitigaciones	Comentarios
A	Registro en kafka	• Requiere de Infraestructura • Necesidad de desarrollos para generar los eventos • Necesidad de desarrollos para la consulta de eventos	Medio /Bajo	Existe un kafka en la organización. Es necesario abordar los desarrollos necesarios e integrarlo en keycloak	• Requiere Kafka • Requiere nuevos desarrollos		Requiere de desarrollos no reutilizables
B	Opentelemetry	Requiere de Infraestructura	Bajo	Es necesario incorporar la opción que aplique Zero-coded a *la solución entregada* y exportar a un colector de opentelemetry	• Requiere de un Collector OTEL	• Desplegar un Collector propio	Es necesario plantearse qué hacer con la salida. existen exportadores

Riesgo	**Tipo** **Riesgo Técnico | Riesgo Operativo | Riesgo Financiero | Riesgo de seguridad**	**Probabilidad baja | media |alta**	**Impacto baja | media |alta**	**Plan de Acción Aceptar | Mitigar**	**Propuestas Disparadoras**				**Propuestas Mitigadoras**			
Riesgo por dependencia de infraestructura de la que no se dispone	Riesgo Técnico	alta	alto	Mitigar	E				E*			
Riesgo por trabajos no aprovechables	Riesgo Financiero	alta	medio	Aceptar	D							
Riesgo por retrabajos	Riesgo Financiero	alta	medio	Aceptar	D							

E*: Despliegue temporal de un collector de Opentelemetry propio y exporta la salida a un sistema por definir (file, ELK, EFK, Kafka...) puede ser necesario algún desarrollo si el sistema de explotación no dispone de exportador nativo. (los exportadores existentes están en la url https://github.com/open-telemetry/opentelemetry-collector-contrib/exporter). La complejidad de despliegue y configuración de un collector es baja

Con Kafka

- Es necesario que desarrollar el extractor.
- Es necesario optimizar la producción de eventos (productor, topic...)
- Kafka es apto para producción
- Es necesario desarrollar la consulta de la información registrada
- Sólo aplicable a los Logs, ni trazas ni métricas
- Desarrollo no reutilizable

Con OpenTelemetry:

- No es necesario que desarrollar el extractor (Zero-Coded).
- No es necesario cuestionarse si está maduro para producción, ya es apto para producción.
- No es necesario que desarrollar el Collector; ya existe y está en versión apta para producción.
- No es necesario desarrollar los exportadores.
- Activar o desactivar Métricas, Trazas, Logs se hace con variables de entorno por lo que es trivial.
- Apuntar de un Collector a otro es cambiar una URL, por lo que el impacto se puede considerar como muy bajo

---

Consecuencias

La instrumentación mediante OpenTelemery se basa en tres partes bien diferenciadas:

1. `instrumentation`: Code-based o Zero-Code . Esta pieza se ocupa de recopilar logs, trazas y métricas y exportarlas a un collector
2. Collector: Es el responsable de recibir todas las señales que se le envían, aplicar las transformaciones que se le indiquen (si es que se le indica alguna transformación) y exportar a otro collector o sistema tercero (kafka, grafana jeager, etc.)
3. Herramientas de consulta y/o control: aquellas herramientas utilizadas para el acceso a la información generada y su uso.

Teniendo estos tres bloques claros Debemos tener claro que **el primer bloque aplica a *la solución entregada*** y los puntos 2 y 3 pertenecen CTI.

Todo lo indicado hasta ahora en este apartado implica que para desplegar *la solución entregada*:

- **Es necesario generar Dockerfile que permita crear una imagen de *la solución entregada* que incluya la opción aplicable Zero-Code y que será utilizada en los siguientes puntos**
- **Es necesario adaptar el helm chart de despliegue de *la solución entregada* para que incluya la configuración que sea necesaria para la opción aplicable Zero-Code**
- **Es necesario adaptar el helm chart de despliegue de *la solución entregada* para que incluya la ejecución de la opción aplicable Zero-Code en el arranque**
- **La configuración de la opción aplicable Zero-Code debe apuntar a la URL de un Collector proporcionado por CTI**

Dependencias externas

La puesta en operación de la solución requiere de una provisión por parte de sistemas de mecanismos adecuados en el escenario ideal, si dicha provisión no es posible, este ADR podrá verse afectado por cambios.

El diseño final podría considerarse de la siguiente forma, donde la salida de los colectores es la que se decida aplicar ya que en este punto no está amenazada la estabilidad del sistema y es posible escalar para asumir el rendimiento necesario. Importante, en el punto anterior se indica el ámbito de responsabilidad de cada actor, esta imagen permite tener una perspectiva del objetivo: