UTN
Facultad Regional Córdoba

TUP

Tecnicatura Universitaria en Programación

PROGRAMACIÓN IV
BACK END

Unidad Temática N°4:
Observability

Material de Estudio
2° Año – 4° Cuatrimestre

INTRODUCCIÓN A LA OBSERVABILIDAD (OBSERVABILITY)

La **observabilidad** en el contexto del desarrollo de software es un concepto esencial que ha ganado una importancia significativa en los últimos años. En un mundo cada vez más impulsado por la tecnología, donde las aplicaciones y sistemas son cada vez más complejos y distribuidos, la capacidad de observar y comprender el comportamiento de un sistema en tiempo real se ha convertido en una necesidad crítica para el éxito en el desarrollo y la operación de software.

La **observabilidad** se refiere a la capacidad de entender, analizar y rastrear el comportamiento de un sistema de software en tiempo real, permitiendo a los equipos de desarrollo y operaciones identificar problemas, diagnosticar fallos y optimizar el rendimiento de las aplicaciones.

Este concepto se ha vuelto especialmente relevante con la adopción generalizada de arquitecturas de microservicios y contenedores, donde los sistemas son altamente dinámicos y pueden involucrar numerosos componentes interconectados. La capacidad de rastrear transacciones a través de estos sistemas y comprender cómo se comportan en su conjunto es esencial para garantizar la fiabilidad y el rendimiento de las aplicaciones.

Evolución de la observabilidad

La evolución del concepto de **observabilidad** en el contexto del desarrollo de software está estrechamente relacionada con el crecimiento de sistemas cada vez más complejos y distribuidos.

El término "**observabilidad**" tiene sus raíces en la teoría de control, que se originó en la ingeniería y la física en el siglo XX. En el contexto de sistemas de control, la **observabilidad** se refiere a la capacidad de determinar el estado interno de un sistema a partir de las salidas observables. Esta idea se trasladó más tarde al ámbito de la informática.

Originalmente apareció lo que se conoce como "**Monitorización Tradicional**"; a medida que las aplicaciones y sistemas de software se volvieron más complejos y se trasladaron a entornos distribuidos, surgió la necesidad de supervisar su rendimiento y salud. Inicialmente, la monitorización se centraba en recopilar datos de alto nivel, como la utilización de CPU, la memoria y la conectividad de red. Si bien esto proporcionó información valiosa, no era suficiente para comprender completamente el comportamiento de sistemas altamente distribuidos y escalables.

La popularización de las arquitecturas de microservicios y contenedores en la década de 2010 marcó un punto de inflexión en la necesidad de observabilidad. Con la fragmentación de aplicaciones en múltiples microservicios y la rápida escalabilidad de contenedores, los enfoques de monitorización tradicionales se volvieron insuficientes. Los equipos de desarrollo y operaciones necesitaban una manera más profunda y detallada de comprender cómo funcionaban estos sistemas altamente distribuidos.

Frente a esa necesidad, **surgió la observabilidad como un enfoque que amplió la monitorización tradicional**. En lugar de limitarse a métricas de alto nivel, la observabilidad se centra en la recopilación de datos en tres áreas clave: **métricas, registros (logs) y trazas (traces)**. Esto proporciona una visión más rica y detallada del comportamiento de una aplicación. También se enfoca en la capacidad de búsqueda y análisis de estos datos, lo que permite a los equipos encontrar patrones, correlaciones y anomalías.

Con el crecimiento de la observabilidad, surgieron herramientas y prácticas especializadas. Herramientas como Prometheus, Grafana, Elasticsearch, Kibana, Jaeger y Zipkin se convirtieron en fundamentales para la implementación de estrategias de observabilidad. Además, prácticas como la instrumentación del código, **la configuración de alertas y la visualización de datos se volvieron comunes en los equipos de desarrollo** y operaciones. La observabilidad se ha convertido en un **componente esencial en el movimiento de DevOps** y las prácticas de Site Reliability Engineering (SRE). Ayuda a los equipos a comprender el impacto de las implementaciones y a mantener la confiabilidad de los sistemas en producción.

Observabilidad vs. Monitoreo

Podemos pensar en la observabilidad como una extensión de la monitorización tradicional, pero mientras que la monitorización se centra en recopilar datos sobre el estado de un sistema, la observabilidad va más allá al proporcionar una visión profunda y detallada del comportamiento interno de una aplicación, permitiendo a los equipos de desarrollo ver más allá de simples métricas y estadísticas. **La realidad es que la observabilidad no es lo mismo que el monitoreo**.

La observabilidad es un concepto basado en la teoría matemática de control, la cual define **la observabilidad como una medida de que tan bien se puede comprender y explicar cualquier estado interno de un sistema en función de sus salidas**. Este concepto es adaptado a la observabilidad del software, incluyendo

consideraciones que son específicas del ámbito de la ingeniería del software, por lo que para que una aplicación tenga observabilidad debemos ser capaces de:

1. Conocer el comportamiento interno de nuestra aplicación/sistema.
2. Comprender cualquier estado en el que la aplicación puede estar, incluso estados nunca vistos anteriormente y que no se podrían haber previsto.
3. Comprender el funcionamiento interno y el estado de la aplicación únicamente observando y preguntando con herramientas externas.
4. Comprender el estado interno sin generar nuevo código para manejar este estado.

El monitoreo tradicional y la observabilidad en el contexto del desarrollo de software son dos enfoques diferentes, y aunque ambos buscan proporcionar información sobre el estado y el rendimiento de un sistema, difieren en términos de alcance, profundidad y enfoque.

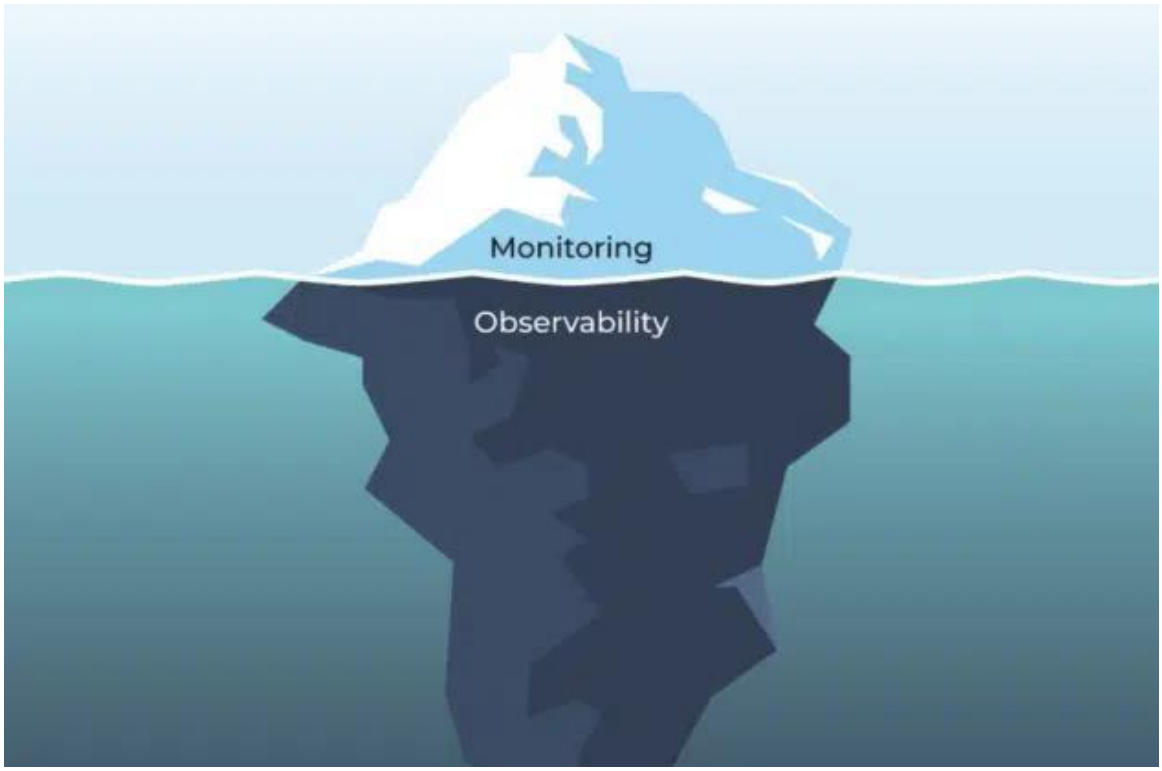

Imagen 1: Extraída de internet.

Monitoreo Tradicional

- **Métricas de Alto Nivel**: El monitoreo tradicional se enfoca en la recopilación de métricas de alto nivel, como la utilización de CPU, la memoria, el tráfico de red y la latencia de las solicitudes. Estas métricas proporcionan una visión general del rendimiento de un sistema, pero carecen de detalles sobre lo que está sucediendo dentro de la aplicación.
- **Visión Limitada**: El monitoreo tradicional tiende a ofrecer una visión superficial de la salud del sistema. Por ejemplo, puede indicar que la CPU de un servidor está al 90%, pero no proporciona información sobre qué proceso o parte de la aplicación está generando esa carga.
- **Reactividad**: El enfoque del monitoreo tradicional es principalmente reactivo. Se utiliza para detectar problemas obvios, como caídas del sistema o errores críticos, pero no siempre es efectivo para identificar problemas sutiles o intermitentes.
- **Escalabilidad Limitada**: A medida que las aplicaciones se vuelven más complejas y distribuidas, el monitoreo tradicional puede tener dificultades para mantenerse al día. Los sistemas modernos, como las arquitecturas de microservicios, pueden generar una gran cantidad de datos que son difíciles de gestionar con enfoques tradicionales.

Observabilidad

- **Datos Detallados y Abundantes**: La observabilidad se centra en la recopilación de datos detallados y abundantes sobre el funcionamiento interno de una aplicación. Esto incluye métricas, registros (logs) y trazas (traces) que proporcionan información contextual sobre cada aspecto de la aplicación.
- **Búsqueda y Análisis Flexibles**: La observabilidad no se limita a la recopilación de datos, sino que también se enfoca en la capacidad de buscar y analizar estos datos de manera eficiente. Esto permite a los equipos de desarrollo identificar patrones, correlaciones y anomalías, así como rastrear el comportamiento de una solicitud a través de múltiples componentes.
- **Enfoque Proactivo**: La observabilidad fomenta un enfoque proactivo para la gestión de sistemas. Permite a los equipos detectar problemas potenciales antes de que se conviertan en fallos críticos. Al configurar

alertas y automatización, es posible tomar medidas preventivas para mantener el sistema en un estado óptimo.

- **Adaptabilidad a Sistemas Modernos**: La observabilidad está diseñada para adaptarse a entornos modernos, como arquitecturas de microservicios y contenedores. Puede manejar la complejidad de sistemas altamente distribuidos y escalables, lo que es crucial en la informática actual.

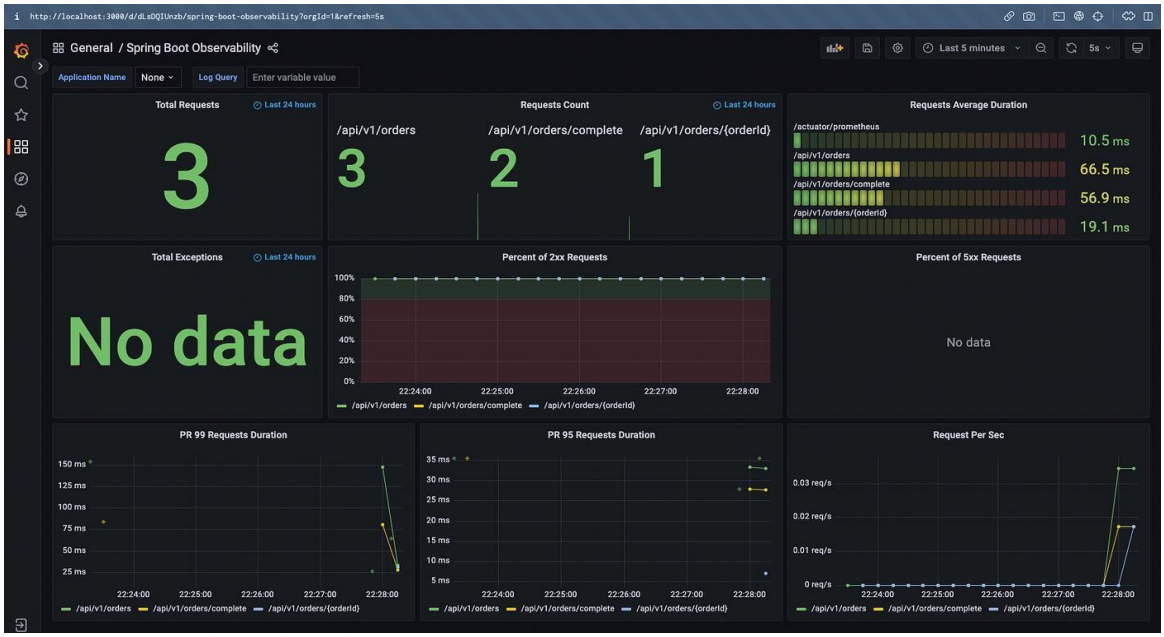

Imagen 2: Extraída de internet.

The Four Golden Signals

Las "***Cuatro Señales Doradas***" (The Four Golden Signals) es un concepto fundamental en el ámbito de la observabilidad de sistemas y aplicaciones. (***If you can only measure four metrics of your user-facing system, focus on these four***). Estas cuatro señales son consideradas esenciales porque proporcionan una visión holística del rendimiento y la salud de una aplicación en producción. Cuando se combinan y se monitorean de manera efectiva, ayudan a los equipos de desarrollo y operaciones a identificar y diagnosticar problemas de manera proactiva, así como a optimizar el rendimiento de la aplicación.

La noción de las Cuatro Señales Doradas se originó en el libro "Site Reliability Engineering" (SRE) de Google, donde se describe la importancia de estas métricas para la gestión de sistemas a escala. Estas señales proporcionan una base sólida

para la observabilidad y son un punto de partida para implementar prácticas de SRE y garantizar la confiabilidad y el rendimiento de las aplicaciones en producción.

Estas señales son métricas clave que ofrecen información esencial sobre el rendimiento y la salud de un sistema, y son ampliamente utilizadas para evaluar el comportamiento de aplicaciones en producción. Estas cuatro señales son:

Latencia (Latency): La latencia se refiere al tiempo que tarda un sistema en responder a una solicitud. Mide la duración entre el momento en que se inicia una solicitud y el momento en que se completa. La latencia es un indicador crítico del rendimiento de una aplicación y puede afectar la experiencia del usuario. Un aumento inesperado de la latencia puede indicar problemas en la infraestructura, la red, o el código de la aplicación.

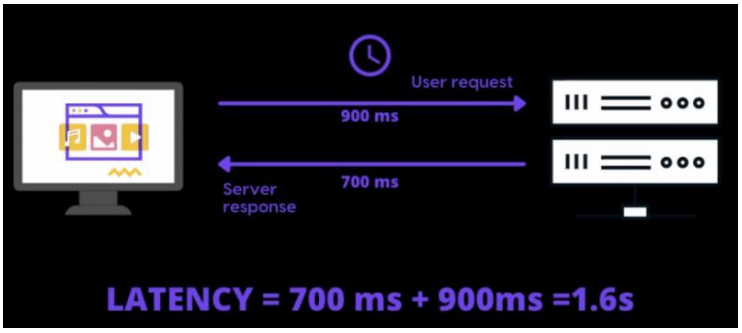

Imagen 3: Extraída de internet.

Tasa de Errores (Errors): La tasa de errores se refiere a la frecuencia con la que se producen errores o excepciones en una aplicación. Estos errores pueden variar desde pequeñas excepciones de manejo de errores hasta fallos críticos. La tasa de errores es un indicador importante de la fiabilidad y la estabilidad de una aplicación. Un aumento repentino en la tasa de errores puede indicar problemas en el código, la configuración o la infraestructura.

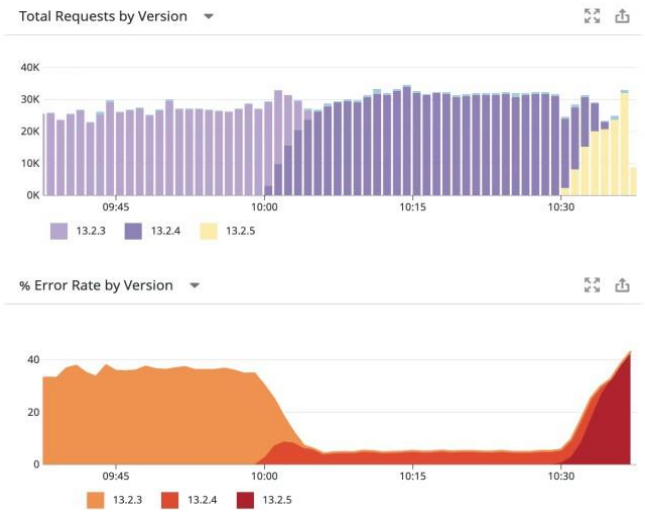

Imagen 4: Extraída de internet.

Tráfico (Traffic): El tráfico se refiere a la cantidad de solicitudes que recibe una aplicación en un período de tiempo determinado. El volumen de tráfico puede variar en función de la hora del día, las campañas promocionales u otros factores. La capacidad de una aplicación para manejar el tráfico entrante es un factor crítico en la gestión de la capacidad y la escalabilidad.

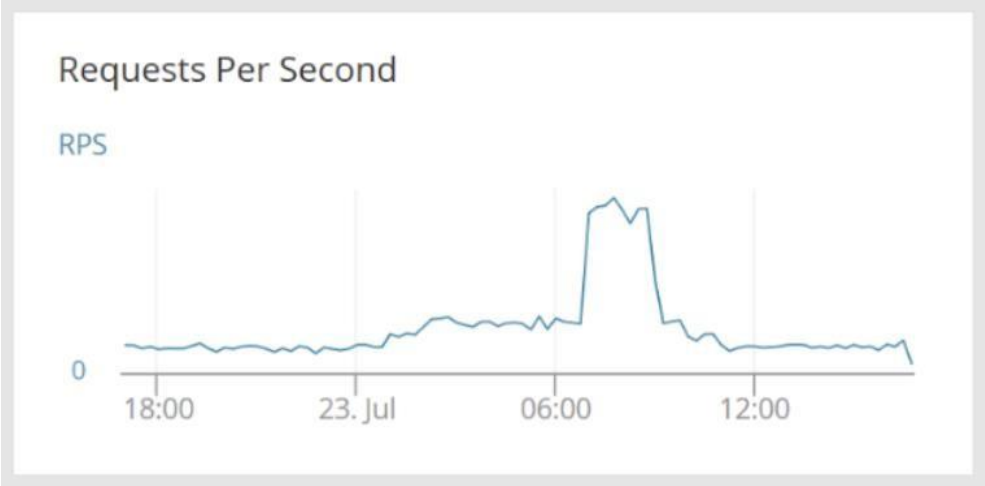

Imagen 5: Extraída de internet.

Saturación (Saturation): La saturación se relaciona con la utilización de los recursos del sistema, como la CPU, la memoria, el espacio en disco y la capacidad de red. Mide cuán cerca está un recurso de alcanzar su capacidad máxima. La saturación es un indicador clave para la planificación de capacidad y el rendimiento del sistema. Si un recurso se satura, puede provocar un aumento en la latencia y los errores.

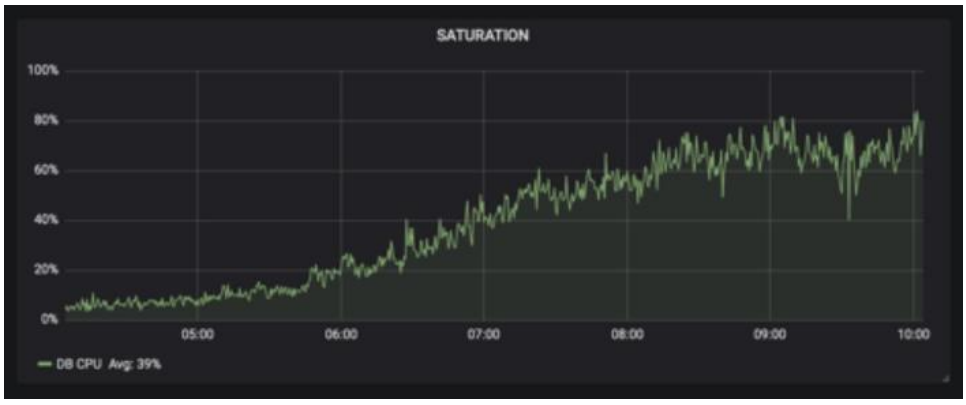

Imagen 6: Extraída de internet.

LOGS (Registros)

Los registros, comúnmente referidos como **"logs"**, son una parte fundamental en el mundo del desarrollo de software y la gestión de sistemas. Estos registros se utilizan para registrar información relevante sobre el comportamiento de una aplicación o sistema en ejecución. Los registros desempeñan un papel esencial en la solución de problemas, el diagnóstico de errores, la auditoría, la seguridad y la observabilidad, lo que los convierte en una herramienta crucial para los desarrolladores, ingenieros de sistemas y administradores de infraestructura.

¿Qué son los Logs?

En su forma más básica, **un log es un registro cronológico de eventos o actividades que ocurren en una aplicación o sistema**. Estos eventos pueden variar desde simples mensajes informativos hasta mensajes de error o advertencia, y pueden incluir información sobre transacciones, solicitudes de usuarios, interacciones con bases de datos y mucho más.

Aplicándolo a un ejemplo de la vida cotidiana es:

- **Negocio local:** "08:15 - Se abrió la panadería; caja inicial $20.000; ingresó primer cliente 08:22." Sirve para auditar horarios, apertura de caja y primeros movimientos del día.
- **Consultorio médico:** *"10:40 - Paciente 'Sosa, M.' llegó 10 minutos tarde; motivo: control; salió 11:05." Ayuda a ver puntualidad, tiempos de atención y posibles cuellos de botella en la sala de espera.*

Importancia de los Logs:

Los registros desempeñan varios roles vitales en el desarrollo de software y la gestión de sistemas:

- **Solución de Problemas y Depuración**: Los registros proporcionan pistas fundamentales para identificar y solucionar problemas. Cuando una aplicación se comporta de manera inesperada o arroja errores, los registros ayudan a los desarrolladores a rastrear la causa subyacente y corregirla.
- **Diagnóstico de Errores**: Los registros detallan los errores y excepciones que se producen durante la ejecución de una aplicación. Estos mensajes de error son valiosos para comprender por qué falló una operación y para tomar medidas correctivas.

- **Auditoría y Cumplimiento**: Los registros son esenciales para rastrear las acciones realizadas en un sistema. Esto es fundamental para fines de auditoría y para garantizar el cumplimiento de regulaciones y políticas, especialmente en entornos sensibles como las transacciones financieras o la atención médica.
- **Seguridad**: Los registros también desempeñan un papel clave en la detección de actividades sospechosas o intentos de intrusión. Los eventos de seguridad y las infracciones pueden ser identificados a través de la inspección de registros.
- **Observabilidad**: Los registros son un componente crucial de la observabilidad en sistemas distribuidos y de alto rendimiento. Proporcionan una visión detallada del comportamiento de una aplicación y de cómo interactúa con otros sistemas.

Tipos de Logs:

Los registros pueden ser de diferentes tipos, y cada tipo se utiliza para un propósito específico:

- **Logs de Aplicación**: Estos registros contienen información sobre el funcionamiento interno de la aplicación. Pueden incluir mensajes informativos, errores y advertencias generados por el código de la aplicación.
- **Logs de Acceso**: Registran las solicitudes de los usuarios y sus interacciones con la aplicación, incluyendo datos como las direcciones IP, los nombres de usuario y las URLs solicitadas.
- **Logs de Seguridad**: Estos registros se centran en eventos de seguridad, como intentos de inicio de sesión fallidos, cambios de contraseña y otros eventos relacionados con la seguridad.
- **Logs de Sistema**: Contienen información sobre el funcionamiento del sistema operativo y la infraestructura, como reinicios, cambios en la configuración del hardware y errores del sistema.
- **Logs de Servidor Web**: Específicos de los servidores web, estos registros rastrean las solicitudes HTTP, los códigos de respuesta y los tiempos de respuesta.

- **Logs de Base de Datos**: Contienen detalles sobre operaciones en una base de datos, como consultas SQL, transacciones y errores relacionados con la base de datos.

Herramientas y Prácticas de Logging:

La gestión efectiva de logs involucra herramientas y prácticas especializadas. Las herramientas de registro (logging) permiten la recolección, el almacenamiento y la visualización de logs, mientras que las mejores prácticas incluyen la estructuración de mensajes de registro, la configuración de **niveles de gravedad** (como INFO, WARNING, ERROR), y la rotación de logs para gestionar el crecimiento de los archivos de registro. Seguridad, cumplir con regulaciones y optimizar el rendimiento de las aplicaciones y sistemas.

Niveles de log

Los niveles de log, a veces también llamados **niveles de gravedad**, son una parte fundamental en la gestión de logs (registros) en el desarrollo de software y la administración de sistemas. Los niveles de log se utilizan para clasificar los mensajes de registro según su importancia y gravedad, lo que **facilita la identificación y el enfoque en eventos críticos**. Los niveles de log más comunes, en orden descendente de gravedad, son:

- **TRACE (ALL):** Se usa para mostrar un detalle mayor que el de DEBUG y está en el top de la jerarquía.
- **DEBUG**: Este nivel se utiliza para mensajes informativos detallados que son útiles para el diagnóstico de problemas y la depuración. Los mensajes de nivel DEBUG suelen contener información técnica y detallada sobre el funcionamiento de la aplicación.
- **INFO**: Los mensajes de nivel INFO proporcionan información general y estados importantes del sistema o de la aplicación. Se utilizan para registrar eventos significativos que no son necesariamente errores, como el inicio de la aplicación o la finalización de una operación importante.
- **WARNING**: Los mensajes de nivel WARNING se utilizan para indicar eventos o condiciones que podrían llevar a un problema en el futuro si no se abordan. Aunque no son errores críticos, los mensajes de advertencia señalan situaciones potencialmente problemáticas.

- **ERROR**: Los mensajes de nivel ERROR indican que se ha producido un error, pero la aplicación todavía puede continuar funcionando. Estos eventos representan problemas que deben abordarse, pero la aplicación no necesariamente se detiene por completo.
- **CRITICAL**: Los mensajes de nivel CRITICAL son los más graves y señalan errores o problemas críticos que pueden causar la detención o el fallo de la aplicación. Estos eventos requieren atención inmediata y acción para restaurar la integridad del sistema.
- **FATAL**: El nivel FATAL indica eventos extremadamente críticos que generalmente resultan en el cierre o la terminación forzosa de la aplicación. Estos eventos representan fallos graves que amenazan la integridad del sistema.

Estos niveles de log permiten a los desarrolladores y administradores clasificar y filtrar los mensajes de registro de acuerdo con su importancia. Por ejemplo, durante la operación normal de una aplicación, es común registrar mensajes de nivel INFO para proporcionar información sobre su funcionamiento, mientras que los mensajes de nivel ERROR y CRITICAL se utilizan para identificar y abordar problemas críticos.

Logs (registros) estructurados y desestructurados

Los registros estructurados y desestructurados se refieren a dos enfoques diferentes para la generación y el almacenamiento de mensajes de registro (logs) en aplicaciones y sistemas. Cada enfoque tiene sus propias ventajas y desventajas, y la elección entre registros estructurados y desestructurados depende de los requisitos específicos de la aplicación y de la facilidad de análisis de los registros.

Registros Estructurados

Los registros estructurados son mensajes de registro que **siguen un formato predeterminado y organizado. Cada mensaje se divide en campos con nombres y valores**, lo que facilita su análisis y consulta. Estos registros son particularmente útiles cuando se requiere una búsqueda y filtrado avanzados de los datos de registro. Ejemplo de un registro estructurado en formato JSON:

```
{
  "timestamp": "2023-10-16T14:30:00",
  "level": "ERROR",
  "message": "Error al procesar la solicitud HTTP",
  "user_id": 12345,
  "service": "API",
  "http_status": 500
}
```

Imagen 7: Extraída de internet.

- "**timestamp**" es la marca de tiempo del registro.
- "**level**" indica el nivel de gravedad del registro.
- "**message**" proporciona una descripción del evento.
- "**user_id**" registra el ID del usuario relacionado con la operación.
- "**service**" identifica el servicio o componente que generó el registro.
- "**http_status**" indica el estado HTTP en una solicitud, en este caso, un error 500.

Los registros estructurados son fáciles de analizar y consultar utilizando herramientas de búsqueda y agregación. Por ejemplo, es posible buscar todos los registros con nivel "ERROR" o filtrar eventos relacionados con un usuario específico.

Registros Desestructurados

Los registros desestructurados son mensajes de registro en formato libre, sin una estructura predeterminada. Suelen contener texto sin formato y pueden ser más difíciles de analizar y consultar, ya que no siguen una estructura coherente. Un registro desestructurado sería así.

```
2023-10-16 14:30:00 - ERROR: Error al procesar la solicitud HTTP. Usuario ID:
12345. Servicio: API. Estado HTTP: 500
```

En este ejemplo, todos los detalles del evento se presentan en un solo mensaje de texto sin formato. No hay una estructura definida con campos clave y valores asociados. Esto puede dificultar la búsqueda y el análisis automatizados de eventos específicos en registros desestructurados.

En definitiva, lo importante de estos tipos de registro es que, **los registros estructurados, son ideales cuando se requiere un análisis detallado y filtrado de los datos de registro**. Facilitan la búsqueda de eventos específicos y la creación de métricas personalizadas. Sin embargo, pueden ocupar más espacio debido a la estructura definida. **Los registros desestructurados, son más simples de generar y pueden ser adecuados para aplicaciones de menor escala**. Son menos adecuados para un análisis automatizado y requieren un mayor esfuerzo de parsing para extraer información relevante.

Log en Open Telemetry

"A log is a timestamped text record, either structured (recommended) or unstructured, with metadata. Of all telemetry signals, logs have the biggest legacy. Most programming languages have built-in logging capabilities or well-known, widely used logging libraries. Although logs are an independent data source, they may also be attached to spans. In OpenTelemetry, any data that is not part of a distributed trace or a metric is a log. For example, events are a specific type of log. Logs often contain detailed debugging/diagnostic info, such as inputs to an operation, the result of the operation, and any supporting metadata for that operation."

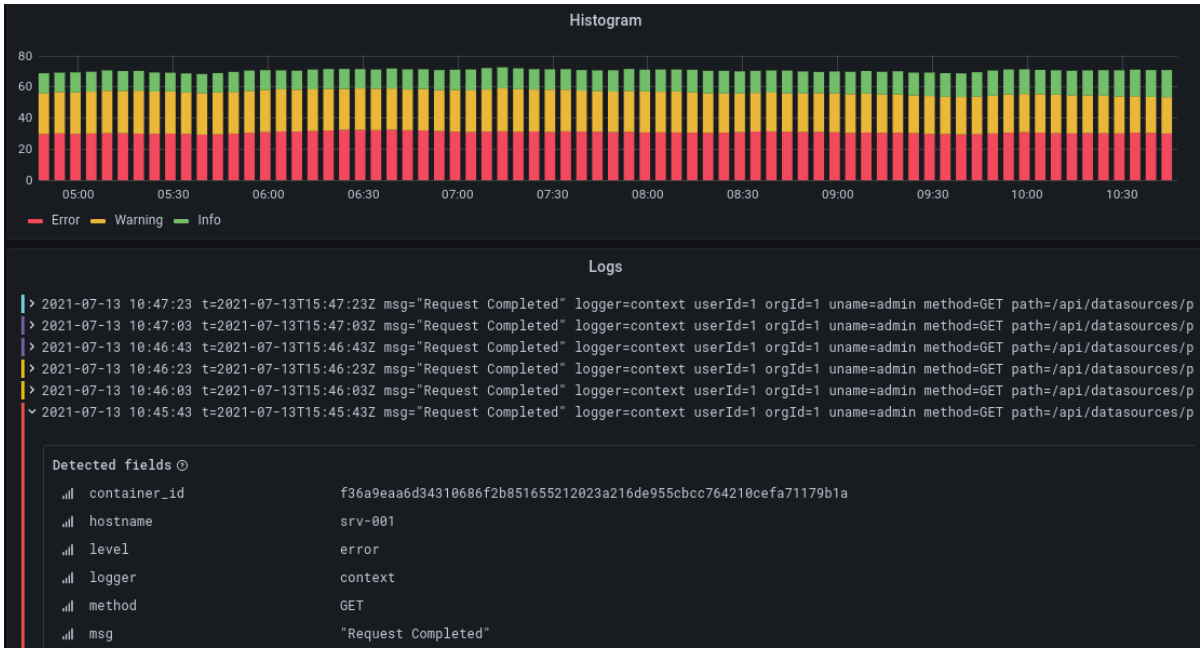

Imagen 8: Extraída de internet.

METRICAS

En el mundo del desarrollo de software, la toma de decisiones informadas es esencial para el éxito de los proyectos. Las métricas, como parte integral de este proceso, desempeñan un papel crucial al proporcionar datos cuantitativos y cualitativos que permiten a los equipos de desarrollo y las organizaciones evaluar, analizar y mejorar sus procesos y resultados.

¿Qué son las Métricas?

En el contexto del desarrollo de software, **una métrica es una medida cuantitativa o cualitativa que se utiliza para evaluar un aspecto específico de un proceso, proyecto o producto de software**. Estas medidas pueden abordar una amplia gama de áreas, desde la calidad del código y el rendimiento de una aplicación hasta la eficiencia de los equipos de desarrollo y la satisfacción del cliente. Las métricas se obtienen a través de la recopilación y el análisis de datos relevantes, y se utilizan para evaluar el estado actual, medir el progreso y tomar decisiones informadas. Las **métricas cuantitativas son medidas expresadas en números y valores numéricos**. Estas métricas proporcionan datos objetivos y medibles que se utilizan para cuantificar y evaluar aspectos específicos (Por ejemplo, velocidad de carga de una página web, porcentaje de uso de CPU o número de errores de software por día). **Las métricas cualitativas se utilizan para evaluar características subjetivas y cualidades que no se pueden medir fácilmente en términos numéricos.** Estas métricas se basan en juicios de valor y descripciones subjetivas (Por ejemplo, satisfacción del cliente, usabilidad de una aplicación o calidad del diseño de una interfaz de usuario).

Importancia de las Métricas en el Desarrollo de Software

- La importancia de las métricas en el desarrollo de software es innegable.
- Permiten a los equipos y organizaciones:
- Evaluar la eficiencia y la efectividad de sus procesos.
- Identificar áreas de mejora y abordar problemas de manera proactiva.
- Establecer metas y medir el progreso hacia el logro de esas metas.
- Tomar decisiones basadas en datos objetivos en lugar de suposiciones.
- Demostrar el valor y la calidad del software entregado a partes interesadas y clientes.
- Evaluar y gestionar el riesgo en proyectos de desarrollo de software.

¿Para Qué Sirven las Métricas en el Desarrollo de Software?

Las métricas desempeñan una variedad de roles fundamentales en el desarrollo de software:

- **Evaluación de la Calidad del Software**: Las métricas pueden ayudar a evaluar la calidad del código, identificar problemas potenciales y definir estándares de calidad. Ejemplos incluyen métricas de complejidad ciclomática, cobertura de pruebas y hallazgos de revisión de código.
- **Gestión de Proyectos**: En la gestión de proyectos, las métricas ayudan a evaluar el progreso, estimar los plazos y los costos, y tomar decisiones basadas en datos sólidos. Por ejemplo, el seguimiento del progreso mediante métricas de velocidad en Scrum.
- **Mejora Continua**: Las métricas proporcionan información para la mejora continua de los procesos de desarrollo de software. Ayudan a identificar áreas que necesitan ajustes y a medir el impacto de las mejoras implementadas.
- **Rendimiento y Eficiencia**: Las métricas de rendimiento ayudan a evaluar el desempeño de una aplicación en términos de velocidad, utilización de recursos y escalabilidad. Estos datos son esenciales para la optimización.
- **Satisfacción del Cliente**: Las métricas de satisfacción del cliente, como encuestas de retroalimentación y tasas de retención, ayudan a comprender cómo los usuarios perciben y utilizan un producto.
- **Toma de Decisiones Estratégicas**: Las métricas aportan información objetiva que respalda la toma de decisiones estratégicas en el desarrollo de software. Esto incluye decisiones sobre nuevas características, inversiones en infraestructura y planificación a largo plazo.

Tipos de métricas

En **OpenTelemetry** las mediciones son capturadas por instrumentos métricos.

Un instrumento métrico se define por:

- Nombre
- Tipo (Kind)
- Unidad (opcional)
- Descripción (opcional)

El nombre, la unidad y la descripción los elige el desarrollador o se definen mediante convenciones semánticas para los más comunes, como métricas de solicitudes y procesos.

En este contexto, se recopilan **tres tipos** (kinds) principales de métricas: **métricas de rastreo** (tracing metrics), **métricas de contadores** (counter metrics) y **métricas de histogramas** (histogram metrics). A continuación, se explica cada uno de estos tipos de métricas en detalle

El tipo de instrumento es uno de los siguientes.

Counter

Son una categoría de métricas utilizadas para realizar un seguimiento de eventos discretos, como ocurrencias de un evento particular. A diferencia de otras métricas que registran valores continuos o distribuciones, **los contadores son métricas simples que aumentan o disminuyen en respuesta a eventos específicos en el sistema que se está monitorizando**. Estos contadores proporcionan información sobre la cantidad de veces que un evento ocurre, lo que es fundamental para evaluar el uso y el comportamiento de una aplicación o sistema.

Un valor que se acumula con el tiempo; puede pensar en esto como el odómetro de un automóvil; solo sube.

- **Cafetería:** “Cafés vendidos hoy: 73” Cada venta incrementa el contador; nunca baja.
- **Cine:** “Entradas emitidas para la función de las 20:00: 152”. Refleja el total de tickets vendidos; es acumulativo.

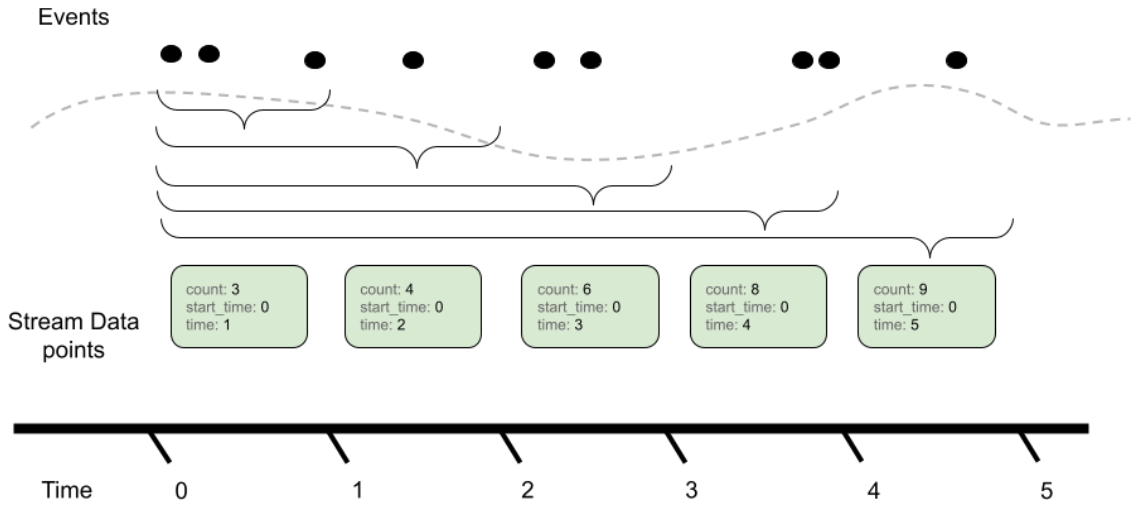

Imagen 9: Extraída de internet.

Asynchronous Counter

Igual que el Contador, pero se recopila una vez por cada exportación. Podría usarse si no tiene acceso a los incrementos continuos, sino sólo al valor agregado.

UpDownCounter

Un valor que se acumula con el tiempo, pero que también puede volver a bajar. Un ejemplo podría ser la longitud de una cola, que aumentará y disminuirá con la cantidad de elementos de trabajo en la cola.

- **Supermercado:** “Personas dentro del local: 34”. Sube cuando entra alguien, baja cuando sale.
- **Estacionamiento:** “Autos actualmente estacionados: 87”. Aumenta con cada ingreso y disminuye con cada egreso.

Asynchronous UpDownCounter

Igual que UpDownCounter, pero se recopila una vez por cada exportación. Podría usarse si no tiene acceso a los cambios continuos, sino solo al valor agregado (por ejemplo, el tamaño de la cola actual).

Gauge

Para medir valores que pueden aumentar o disminuir de forma arbitraria a lo largo del tiempo. A diferencia de otros tipos de métricas que se suman o promedian, como los contadores y los histogramas, las métricas de tipo Gauge **registran un valor puntual en un momento específico y no almacenan información sobre los cambios acumulativos**.

Las métricas de tipo Gauge son especialmente útiles cuando se desea medir estados o valores instantáneos que no se acumulan, como el uso de la CPU, la memoria disponible o el número actual de usuarios conectados a una aplicación. Estas mediciones son asincrónicas. Más detalles en el siguiente link.

- **Hogar:** “Temperatura del living ahora: 22,5 °C”. Es el valor en este instante; no acumula.
- **Auto:** “Nivel de combustible actual: 3/8 de tanque”. Un estado momentáneo que cambia con el uso o la recarga.

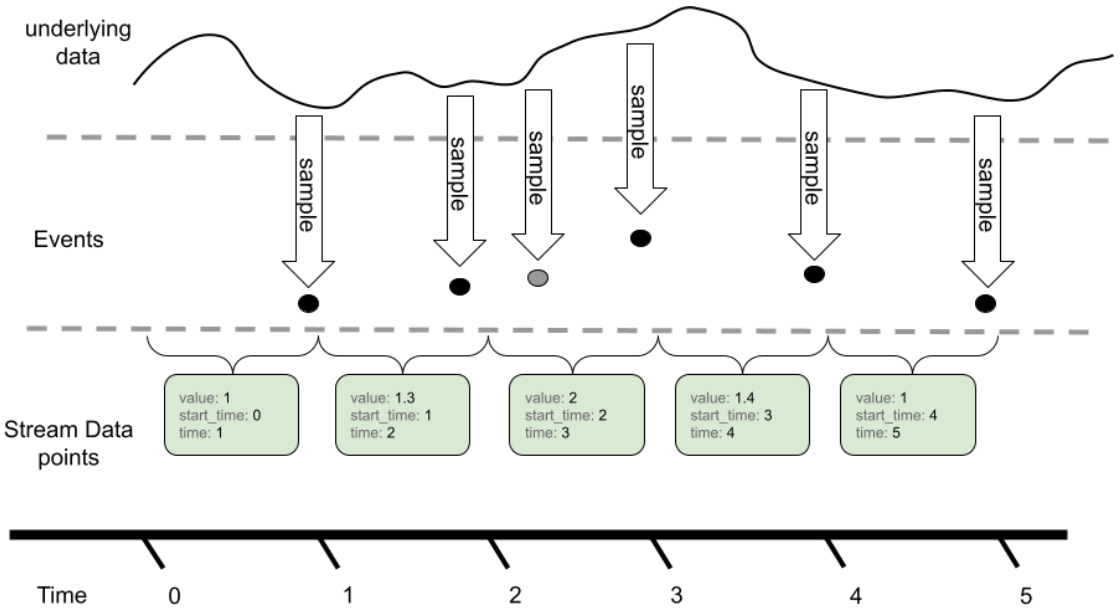

Imagen 10: Extraída de internet.

Histogram

Una agregación de valores del lado del cliente, como latencias de solicitud. Un histograma es una buena opción si está interesado en estadísticas de valor. Por ejemplo: ¿Cuántas solicitudes tardan menos de 1 segundo?

A diferencia de otras métricas que se centran en valores puntuales o contadores, los histogramas **registran la frecuencia con la que ocurren distintos rangos de valores en un conjunto de datos**. Estas métricas son particularmente útiles para evaluar la variabilidad y la dispersión de datos, lo que permite un análisis más detallado y profundo del comportamiento de una aplicación o sistema. Más detalles en el siguiente link.

- **Farmacia (tiempos de espera):**

  0–5 min: 18 personas · 5–10 min: 25 · 10–15 min: 9 · >15 min: 3. Permite ver en qué rango se concentran la mayoría de las esperas.

- **Patio de comidas (precio del almuerzo):**

  ≤ $3.000: 12 tickets · $3.001–$5.000: 40 · $5.001–$7.000: 22 · >$7.000: 6 Muestra cómo se distribuyen los precios que paga la gente.

Metrica en Open Telemetry

"A metric is a measurement of a service captured at runtime. The moment of capturing a measurements is known as a metric event, which consists not only of the measurement itself, but also the time at which it was captured and associated metadata.

Application and request metrics are important indicators of availability and performance. Custom metrics can provide insights into how availability indicators impact user experience or the business. Collected data can be used to alert of an outage or trigger scheduling decisions to scale up a deployment automatically upon high demand."

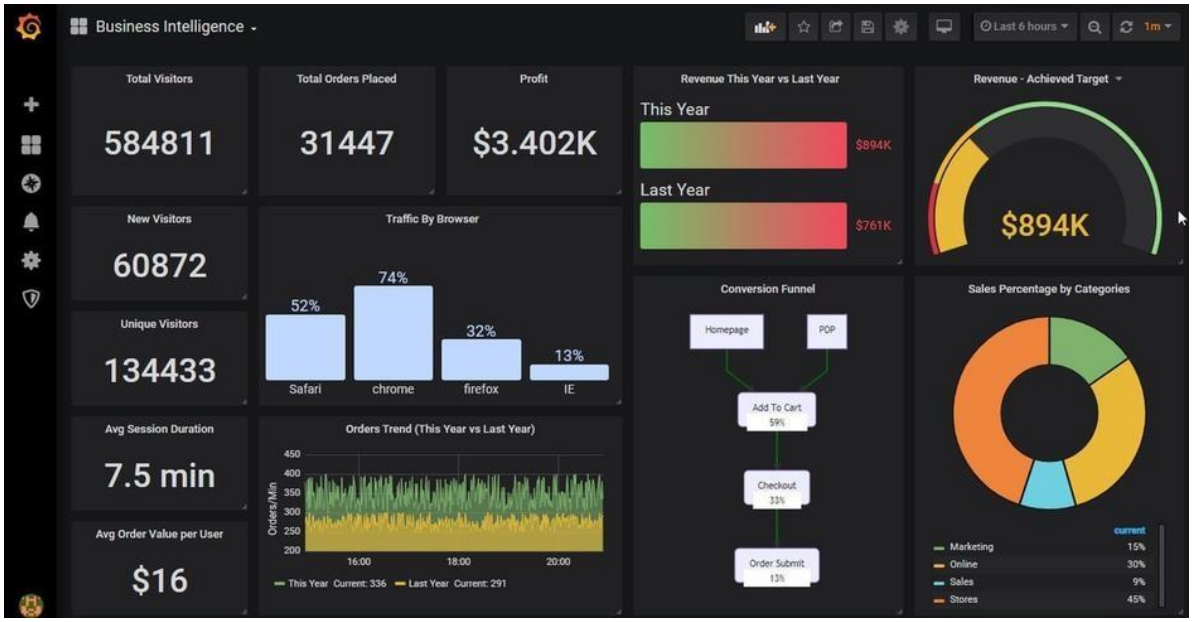

Imagen 12: Extraída de internet.

TRAZAS

Como ya vimos, la observabilidad en sistemas distribuidos se refiere a la capacidad de comprender y analizar el comportamiento de un sistema en tiempo real. La observabilidad es esencial para garantizar el rendimiento, la confiabilidad y la solución eficiente de problemas en entornos de software complejos y distribuidos. En este contexto, las "trazas" desempeñan un papel fundamental. **Las trazas son registros detallados de la ejecución de una aplicación o sistema a lo largo del tiempo, y su importancia radica en su capacidad para proporcionar visibilidad y contexto en entornos distribuidos.**

¿Qué son las Trazas?

Las trazas, también conocidas como "logs de trazas" o "eventos de trazabilidad", **son registros detallados de las actividades y eventos que ocurren dentro de una aplicación o sistema en un momento específico**. Estas actividades pueden incluir solicitudes de usuarios, procesamiento de datos, interacciones con bases de datos, eventos del sistema, comunicaciones de red y más. Las trazas capturan información como marcas de tiempo, identificadores de transacciones, detalles de solicitudes, resultados de operaciones y otros datos relevantes.

En definitiva, **las trazas son registros de seguimiento que muestran el flujo de ejecución de un sistema**. Las trazas se utilizan para rastrear la secuencia de eventos que ocurren en el software o arquitectura. Las trazas muestran la ejecución de las diferentes partes del código y cómo interactúan entre sí. Estas suelen incluir información sobre algunas llamadas a funciones, identificadores únicos, parámetros o valores de retorno, todo esto dependiendo de las necesidades.

El objetivo principal de las trazas es básicamente identificar el tiempo de ejecución y el camino que sigue una solicitud o transacción a través del sistema. Las trazas son fundamentales para darle seguimiento, factor muy importante en sistemas distribuidos.

Tracing

La definición de **tracing** pasa por la definición de request. Una request es una transacción en nuestro sistema que tiene principio y fin (lifecycle). El tracing se encarga de estudiar el comportamiento de las requests y de su lifecycle. **En los sistemas modernos distribuidos es posible realizar tracing siguiendo una request a lo largo de su procesamiento en sistemas distintos**, cada salto de un

sistema a otro se conoce como span. Esta técnica se utiliza normalmente para estudiar problemas de rendimiento y hacer debugging de errores puntuales.

Trazas en Open Telemetry

"Traces give us the big picture of what happens when a request is made to an application. Whether your application is a monolith with a single database or a sophisticated mesh of services, traces are essential to understanding the full "path" a request takes in your application".

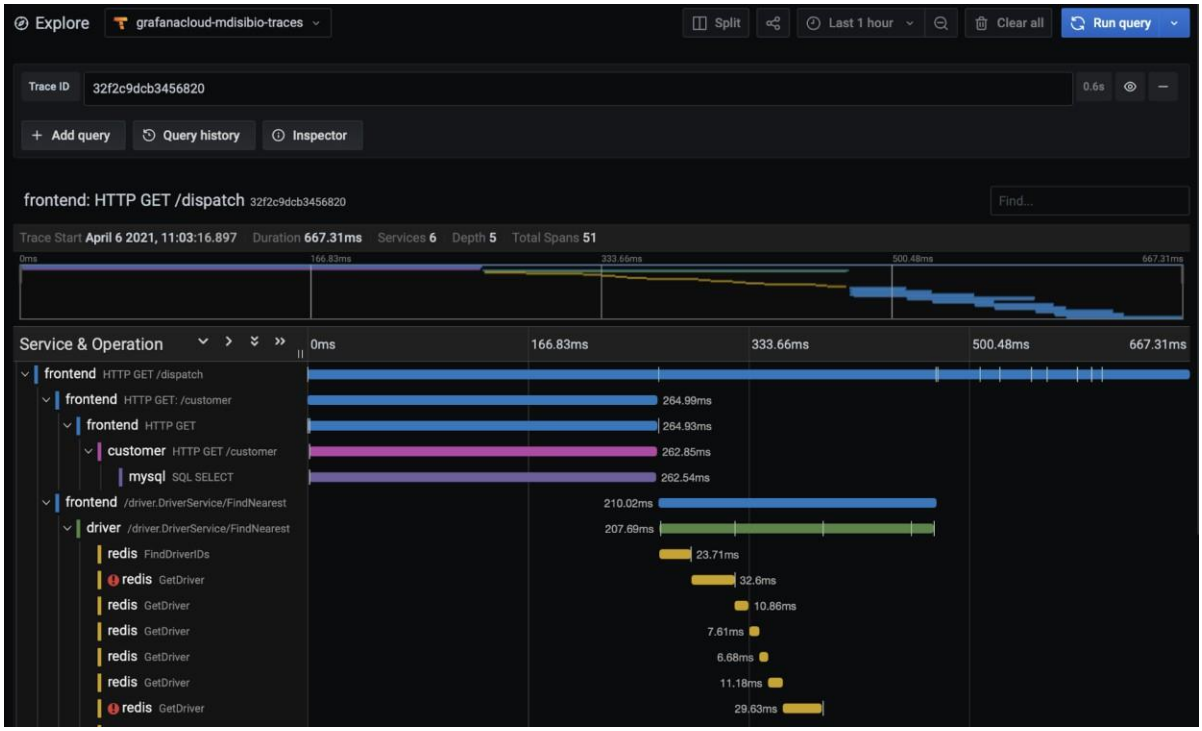

Imagen 13: Extraída de internet.

OPEN TELEMETRY (OTel)

Definición del sitio OpenTelemetry: *"**OpenTelemetry is an Observability framework and toolkit designed to create and manage telemetry data such as traces, metrics, and logs.** Crucially, OpenTelemetry is vendor- and tool-agnostic, meaning that it can be used with a broad variety of Observability backends, including open source tools like Jaeger and Prometheus, as well as commercial offerings. OpenTelemetry is a Cloud Native Computing Foundation (CNCF) project."*

En pocas palabras, **OpenTelemetry es una iniciativa de código abierto que proporciona una plataforma unificada para la observabilidad en aplicaciones y sistemas distribuidos**. Este framework nació con el propósito de resolver desafíos fundamentales en la observabilidad y el diagnóstico de sistemas modernos, que son cada vez más complejos y distribuidos.

Origen de OpenTelemetry

OpenTelemetry es el resultado de la fusión de dos proyectos de código abierto ampliamente utilizados en el ámbito de la observabilidad: OpenTracing y OpenCensus. OpenTracing se centró en proporcionar una API común para instrumentar aplicaciones y rastrear solicitudes a través de sistemas distribuidos, mientras que OpenCensus se centró en la recopilación de datos de telemetría, como métricas y eventos de rastreo. **En 2019, estos dos proyectos se unieron bajo el nombre de OpenTelemetry para crear una solución unificada y más poderosa**.

¿Qué es OpenTelemetry?

OpenTelemetry es una plataforma de observabilidad que permite a los desarrolladores y operadores recopilar datos críticos para entender el rendimiento y el comportamiento de aplicaciones y sistemas distribuidos. **Proporciona una API común para instrumentar aplicaciones y recopilar métricas, trazas y registros (logs)**. Las características clave de OpenTelemetry son:

- **Instrumentación Consistente**: OpenTelemetry ofrece una API coherente y unificada que los desarrolladores pueden utilizar para agregar instrumentación a sus aplicaciones. Esto facilita la captura de datos de telemetría sin importar el lenguaje de programación o el marco utilizado.
- **Recopilación de Datos de Telemetría**: OpenTelemetry admite la recopilación de datos de telemetría, que incluye métricas, trazas y logs,

para proporcionar una imagen completa del rendimiento y el comportamiento de las aplicaciones.

- **Integración con Plataformas de Observabilidad**: OpenTelemetry se integra con una variedad de herramientas y plataformas de observabilidad, como sistemas de trazado (por ejemplo, Jaeger, Zipkin), sistemas de métricas (por ejemplo, Prometheus), almacenamiento de registros y visualización de datos.
- **Flexibilidad y Extensibilidad**: OpenTelemetry permite la personalización y la extensibilidad a través de plug-ins y adaptadores para adaptarse a los requisitos específicos de las aplicaciones y sistemas.

Componentes principales de Open Telemetry (OTel)

- A specification for all components
- A standard protocol that defines the shape of telemetry data
- Semantic conventions that define a standard naming scheme for common telemetry data types
- APIs that define how to generate telemetry data
- A library ecosystem that implements instrumentation for common libraries and frameworks
- Automatic instrumentation components that generate telemetry data without requiring code changes
- Language SDKs that implement the specification, APIs, and export of telemetry data
- The OpenTelemetry Collector, a proxy that receives, processes, and exports telemetry data
- Various other tools, such as the OpenTelemetry Operator for Kubernetes, OpenTelemetry Helm Charts, and community assets for FaaS

OpenTelemetry is compatible with a wide variety of ecosystem integrations.

OpenTelemetry is supported by 40+ vendors, many of whom provide commercial support for OpenTelemetry and contribute to the project directly.

BIBLIOGRAFÍA

- The Utilization Saturation and Errors (USE) Method - Brendan Gregg (https://brendangregg.com/usemethod.html)
- The Site Reliability Engineering (https://sre.google/sre-book/table-of-contents/)
- opentelemetry.io (https://opentelemetry.io/docs/concepts/observability-primer/)