Observabilidad, Métricas y KPIs

De la monitorización básica a la inteligencia de negocio

1. ¿Qué es la Observabilidad? (Más allá de la Monitorización)

En la ingeniería moderna, distinguimos dos conceptos clave que a menudo se confunden:

- **Monitorización:** Te dice **cuándo** algo está mal basándose en lo que ya sabes que debes buscar (ej. "CPU al 90%"). Es reactiva.
- **Observabilidad:** Te permite entender **por qué** algo está mal interrogando al sistema, incluso ante problemas que nunca habías visto ("desconocidos desconocidos"). Se basa en explorar patrones no definidos de antemano.

> **La meta:** No solo ver que la web va lenta, sino descubrir que la latencia se debe a una consulta específica a la base de datos de usuarios premium en la región eu-west-1.

2. Los 4 Pilares de la Telemetría

Para lograr observabilidad, necesitamos recolectar cuatro tipos de señales (Telemetría):
s

1. **Métricas (Metrics):** Datos numéricos medidos en el tiempo. Son baratas de almacenar y rápidas de consultar.
    - *Ejemplo:* Uso de RAM, peticiones por segundo, latencia media.
    - *Nos dicen:* "¿Hay un problema?".
2. **Registros (Logs):** Eventos discretos e inmutables con marca de tiempo.
    - *Ejemplo:* "Error 500: Database connection failed".
    - *Nos dicen:* "¿Por qué ocurrió el problema?".
3. **Trazas (Traces):** El recorrido de una petición a través de múltiples microservicios.
    - *Ejemplo:* El usuario hace click -> API Gateway -> Lambda -> DynamoDB.
    - *Nos dicen:* "¿Dónde está el cuello de botella?".
4. **Perfiles (Profiles):** Análisis del uso de recursos a nivel de código (qué función consume más CPU). Es el pilar más reciente.

3. Métricas Técnicas vs. KPIs de Negocio

Es vital diferenciar entre salud del sistema y salud del negocio.

Métricas Técnicas (System Health)

Miden el rendimiento de la infraestructura y el software.

- **Golden Signals (SRE):** Latencia, Tráfico, Errores y Saturación.
- **Uso:** Para los ingenieros (DevOps/SRE). Si la CPU está al 100%, escalamos máquinas.

KPIs (Key Performance Indicators)

Miden el éxito de la organización o del producto basándose en objetivos estratégicos.

- **Ejemplos:** Tasa de conversión de carritos, Ingresos por hora, Usuarios activos diarios (DAU), Coste de adquisición (CAC).
- **Uso:** Para Product Owners y Negocio. Si los ingresos caen, es una emergencia, aunque la CPU esté perfecta.

4. Métricas de Rendimiento de Equipo (DORA)

Además de medir el software, medimos cómo de bien trabajamos nosotros. Las métricas **DORA** (DevOps Research and Assessment) son el estándar de la industria:

1. **Frecuencia de Despliegue (Deployment Frequency):** ¿Con qué frecuencia llevamos código a producción? (Busca: Varias veces al día).
2. **Tiempo de Entrega (Lead Time for Changes):** Tiempo desde el "commit" hasta que corre en producción.
3. **Tasa de Fallos en Cambios (Change Failure Rate):** ¿Qué porcentaje de despliegues rompen algo?
4. **Tiempo Medio de Recuperación (MTTR):** Cuando algo se rompe, ¿cuánto tardamos en arreglarlo?

> **Objetivo:** Velocidad y Estabilidad no son opuestas; los equipos de alto rendimiento tienen ambas.

5. Arquitectura de un Sistema de Observabilidad

Para montar esto en la nube, necesitamos un **Pipeline de Telemetría** que desacople la generación de datos de su almacenamiento.

1. **Instrumentación (El Origen):** El código emite datos.
    - *Estándar actual:* **OpenTelemetry (OTel)**. Es agnóstico al proveedor. Instrumentas una vez y envías a donde quieras.
2. **Recolección y Transporte:** Agentes que recogen los datos.
    - *Herramientas:* OTel Collector, Fluent Bit, Vector. Actúan como intermediarios para no saturar la app.
3. **Almacenamiento y Análisis (Backend):**
    - *Métricas:* Prometheus, CloudWatch Metrics.
    - *Logs:* Loki (barato, indexa solo etiquetas), Elasticsearch (potente búsqueda texto completo), CloudWatch Logs.
    - *Trazas:* Jaeger, Tempo, X-Ray.
4. **Visualización:**
    - **Grafana:** El estándar de facto para dashboards unificados.

6. Estrategia de Negocio: Correlación de Datos

El verdadero valor surge al cruzar datos.

- **Ejemplo de Dashboard "North Star":**
  - Un panel muestra: **Ingresos en tiempo real** (Métrica de Negocio).
  - Otro panel muestra: **Latencia del Checkout** (Métrica Técnica).
  - *Insight:* ¿La caída de ingresos de las 10:00 AM coincide con el despliegue de la versión 2.1?.

Cómo implementarlo:
Usando **OpenTelemetry**, añadimos "Atributos personalizados" a las trazas o métricas:

```
# Ejemplo conceptual
span.set_attribute("user.type", "premium")
span.set_attribute("cart.value", 150.00)
```

Esto permite filtrar en Grafana: *"Muéstrame los errores 500, pero solo de usuarios que tenían más de 100€ en el carrito"*.

7. Ecosistema de Herramientas: Clasificación

Para construir un sistema robusto, debemos seleccionar las piezas adecuadas para cada etapa del ciclo de vida del dato. A continuación, las herramientas más destacadas en la industria actual.

A. Instrumentación (Generación de datos)

- **OpenTelemetry (OTel) SDKs:** El estándar actual. Permite instrumentar la aplicación una sola vez y enviar los datos a cualquier backend. Soporta trazas, métricas y logs de forma unificada.
- **Prometheus Client Libraries:** Específicas para métricas. Exponen un endpoint `/metrics` que es "raspado" (scraped) periódicamente.
- **Telegraf:** Un agente basado en plugins muy versátil para recopilar métricas de bases de datos, sistemas y sensores IoT.
- **Elastic Beats:** Agentes ligeros (Filebeat, Metricbeat) para enviar datos específicamente al ecosistema Elastic.

B. Recolección y Transporte (La Tubería)

Estas herramientas actúan como intermediarios, procesando y enviando datos para no sobrecargar la aplicación principal.

- **OpenTelemetry Collector:** Un proxy neutral que recibe, procesa (filtra/transforma) y exporta telemetría. Es la pieza clave para desacoplar el origen del destino.
- **Fluent Bit:** Procesador de logs extremadamente ligero y rápido (escrito en C). Es el estándar para entornos de contenedores y Kubernetes debido a su bajo consumo de memoria.
- **Vector:** Herramienta de alto rendimiento escrita en Rust. Destaca por su seguridad de memoria y eficiencia en el manejo de logs y métricas.
- **Apache Kafka:** Utilizado en arquitecturas masivas como búfer intermedio para desacoplar la recolección del almacenamiento y evitar pérdida de datos en picos de tráfico.

C. Almacenamiento y Análisis (El Backend)

- **Métricas (Series Temporales):**
  - **Prometheus:** El estándar de facto para métricas en Kubernetes.
  - **Mimir / Thanos:** Soluciones para escalar Prometheus a largo plazo y alta disponibilidad.
  - **TimescaleDB / InfluxDB:** Bases de datos especializadas en series temporales de alto rendimiento.
- **Logs:**
  - **Grafana Loki:** Indexa solo metadatos (etiquetas), lo que lo hace muy barato y eficiente (similar a Prometheus pero para logs).
  - **Elasticsearch / OpenSearch:** Motores de búsqueda de texto completo. Ideales para búsquedas complejas y análisis forense profundo.
- **Trazas:**
  - **Jaeger:** Sistema de trazado distribuido clásico.
  - **Grafana Tempo:** Backend de trazas de alta escala y bajo coste, diseñado para no necesitar muestreo agresivo.

D. Visualización (El Panel de Control)

- **Grafana:** La plataforma líder de código abierto. Permite visualizar datos de múltiples fuentes (Prometheus, Loki, Elastic, SQL) en un solo dashboard unificado.
- **Kibana:** La interfaz nativa de Elastic Stack. Potente para la exploración profunda de logs y visualización de datos de seguridad.

8. Recomendación de Stacks (Arquitecturas)

Elegir el stack correcto depende de la madurez del equipo, el presupuesto y la escala. Aquí presento tres niveles de complejidad.

Nivel 0: El Stack Nativo (AWS CloudWatch)

Si ya estamos en AWS y buscamos la **mínima fricción operativa**, la respuesta es usar las herramientas nativas de la plataforma.

- **Filosofía:** Integración inmediata. No hay que instalar servidores de bases de datos ni gestionar clústeres; es un servicio totalmente gestionado
- **Ideal para:** Proyectos con tiempos ajustados, equipos pequeños o arquitecturas 100% Serverless donde no queremos mantener infraestructura de monitoreo

Arquitectura del Stack CloudWatch

- **Instrumentación (Recolectores):**
  - **Automática:** Servicios como EC2, Lambda o RDS envían métricas básicas (CPU, Disco) automáticamente.
  - **CloudWatch Agent:** Se instala en las instancias para enviar métricas personalizadas (ej. uso de memoria RAM) y logs de aplicación
- **Almacenamiento:** CloudWatch Logs y Metrics.
- **Visualización:**
  - **Nativa:** CloudWatch Dashboards (simple y directo).
  - **Híbrida:** Grafana conectado a la API de CloudWatch (para visualizaciones más potentes)

Nivel 1: El Stack "Starter Cloud-Native" (Simple y Eficiente)

Ideal para empezar en Kubernetes o sistemas medianos donde el coste es prioridad.

- **Composición:**
  - *Instrumentación:* Prometheus Libs + Logs a `stdout`.
  - *Recolección:* Prometheus (Scraping) + Promtail (Logs).
  - *Almacenamiento:* Prometheus + Loki.
  - *Visualización:* Grafana.
- **Pros:** Muy fácil de desplegar. Loki es muy barato al no indexar el texto completo. Integración nativa perfecta en Grafana.
- **Contras:** Búsquedas de texto libre en logs son más lentas que en Elastic. Retención de datos limitada si no se configura almacenamiento en objetos (S3).

Nivel 2: El Stack "Modern Enterprise" (Estándar OTel)

Para empresas que buscan evitar el bloqueo de proveedores y necesitan trazabilidad completa.

- **Composición:**
  - *Instrumentación:* **OpenTelemetry SDKs**.
  - *Recolección:* **OTel Collector** (Gateway central).
  - *Almacenamiento:* Mimir (Métricas) + Loki (Logs) + Tempo (Trazas) (Stack LGTM).
  - *Visualización:* Grafana.
- **Pros: Agnóstico al proveedor**: Si cambias de backend, tu código no se toca. Visibilidad total (Logs, Métricas y Trazas correlacionados).
- **Contras:** Curva de aprendizaje del OTel Collector. Gestionar Mimir/Tempo requiere conocimientos de operaciones.

Nivel 3: El Stack "Log-Heavy / Compliance" (Robusto)

Para organizaciones con requisitos masivos de logs, auditoría o seguridad (SIEM).

- **Composición:**
  - *Instrumentación:* Beats / Fluentd.
  - *Transporte:* Fluent Bit -> **Kafka** -> Logstash.
  - *Almacenamiento:* **Elasticsearch / OpenSearch**.
  - *Visualización:* Kibana.
- **Pros: Kafka** garantiza cero pérdida de datos ante picos. Elasticsearch permite búsquedas de texto completo instantáneas y complejas, vital para seguridad.
- **Contras: Costoso y pesado**. Elasticsearch consume mucha RAM/CPU. Operar un clúster de Kafka añade complejidad significativa.

Resumen de Selección

Necesidad	Stack Recomendado	Por qué
Solución Básica	Cloudwatch	Integración nativa, sin mantenimiento.
Inicio Rápido / K8s	Prometheus + Loki + Grafana	Bajo coste, despliegue sencillo, estándar en K8s.
Flexibilidad Futura	OpenTelemetry + Grafana Stack	Estandarización, evita vendor lock-in, correlación total.
Busqueda Logs Compleja	ELK Stack (con Kafka)	Potencia de búsqueda de texto, robustez ante fallos.
SaaS (Sin Ops)	Datadog / New Relic / Dynatrace	"Funciona solo", pero alto coste a escala.