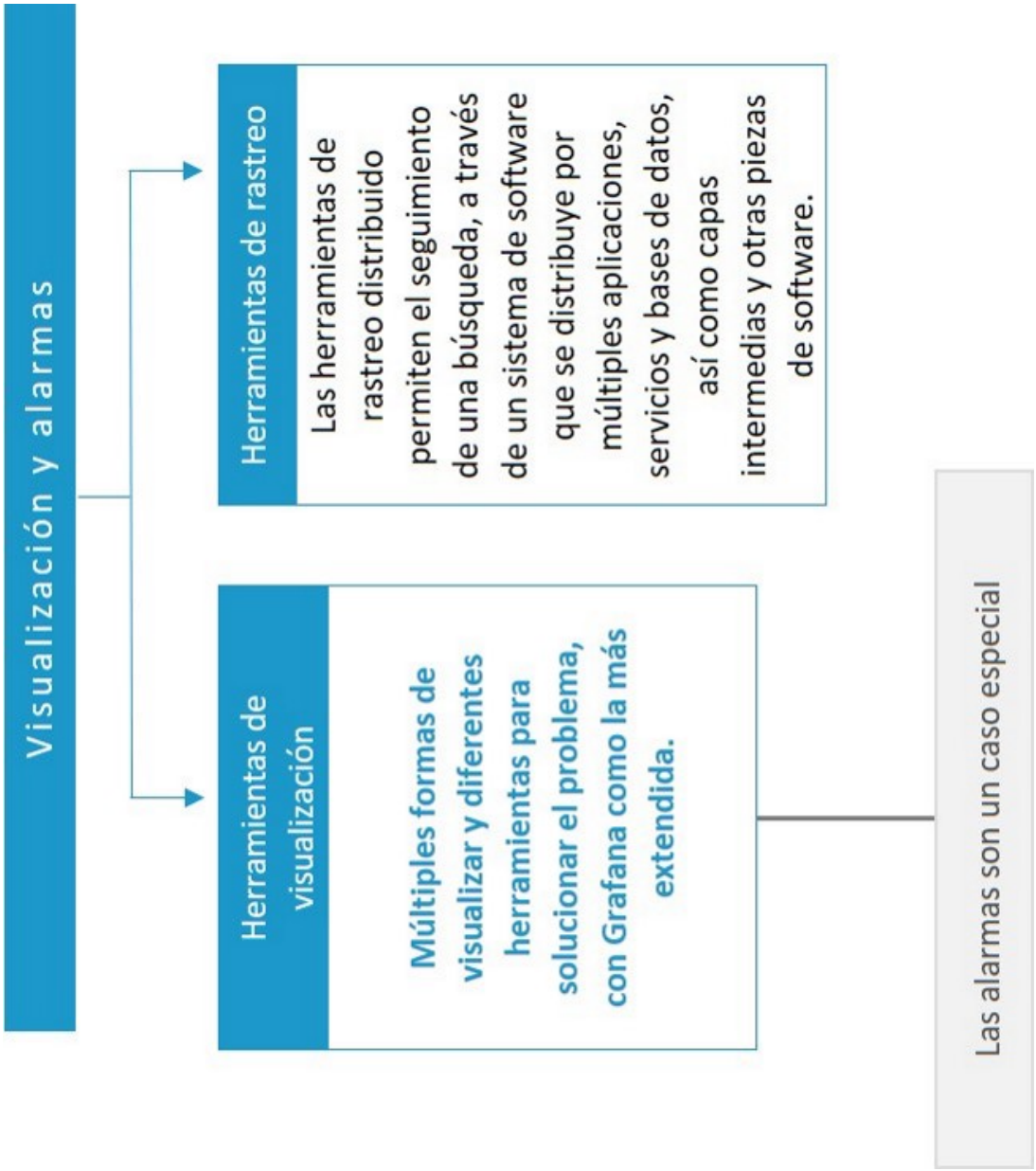
Cloud Computing, DevOps y DevOps Culture

Tema 10. Visualización y alarmas

Índice

Esquema



10.1. Introducción y objetivos

Los ingenieros DevOps tienen como parte de sus cometidos desarrollar y desplegar los entornos de producción, pero, además, asegurar su correcto funcionamiento. La monitorización es una pieza fundamental para garantizar el funcionamiento de un sistema, pero también lo son las alertas y que todos los datos asociados estén accesibles para ser utilizados. Por poner un ejemplo, el mejor sistema de *logs* no sirve de nada si nadie puede acceder a él para analizarlo.

A partir del estudio de este tema, se pretenden conseguir los siguientes objetivos:

- Conocer la función que desempeñan las herramientas de visualización, las alarmas y las herramientas de rastreo para el mantenimiento de los sistemas.
- Reconocer cuáles son los casos de uso y las mejores prácticas de las alertas en el contexto de DevOps.
- Conocer cuáles son las herramientas más importantes y populares en el mercado actual para la visualización y rastreo.

Las herramientas de visualización también son llamadas herramientas de observabilidad. Estas últimas provienen de la teoría del control y hacen referencia a nuestra capacidad para entender un sistema a través de sus entradas y salidas. Podría considerarse a las herramientas de visualización y alerta como aquellas que proporcionan representaciones estructuradas de salidas del sistema**. Las alertas son, básicamente, una selección de salidas negativas del sistema y las visualizaciones son representaciones estructuradas desambiguadas, enfocadas en facilitar la comprensión al usuario.**

A la hora de hablar de las alertas, resulta importante destacar la importancia de poder filtrarlas y gestionarlas de forma correcta. Estas no deben enviarse si no son significativas para el usuario que las recibe, especialmente si dichas alertas van

dirigidas a varias personas y solo unos pocos pueden responder a situaciones anómalas del sistema que desencadena la alerta. La razón de peso para respetar esta práctica reside en el hecho de que **los usuarios que reciben una alerta, pasado algún tiempo, llegarán a ignorarlas y asumirán que no son importantes o que no van dirigidas a ellos.** Por ejemplo, si un operador está recibiendo cientos de correos electrónicos al día desde el sistema de alertas, ese operador en poco tiempo va a ignorar todos los correos electrónicos del sistema de alerta. El operador solo responderá a un incidente real cuando él o ella esté experimentando el problema en su propia piel, cuando se lo reporte su jefe o un cliente usuario del sistema.

Las alertas no deben usarse como un flujo constante de información o una actualización de estado. Están destinadas a notificar un determinado problema sobre el que el sistema no puede recuperarse de forma automática y se deben enviar solo a la persona que tenga la capacidad de poder recuperar el sistema. Aquello que quede fuera de esta definición no es una alerta y solo perjudicará a los empleados y a la cultura de la empresa.

Todos tenemos en mente un conjunto diferente de tipos de alerta, pero no deberían implicar niveles de prioridad (P1-P5) o utilizar palabras como «informativo», «advertencia» o «crítico». En su lugar, deberían describir las categorías genéricas que faciliten la respuesta a incidentes del sistema. Es posible que hayas notado que antes se ha mencionado un tipo de alertas «informativo», a pesar de que ahora se hace hincapié en que estas alertas no deberían estar incluidas. Esto sucede porque en la documentación de muchas herramientas y ejemplos se les llama incorrectamente (a mi juicio) alertas.

Alertas y herramientas de visualización de alertas

Los gráficos de barras y de líneas son los más habituales y frecuentes a la hora de representar este tipo de información: agregación de eventos y alarmas. Dependiendo

de la complejidad de la información que se desee mostrar y las dimensiones de los datos, también existen otras opciones que enriquecen la información y facilitan su análisis. A continuación, veremos algunas de ellas.

Mapas de calor

Este tipo de visualización resulta extremadamente útil para representar la información que proviene de histogramas. Es similar a un gráfico de barras, pero más enriquecido: puede mostrar gradientes dentro de las barras, que representan los diferentes percentiles respecto a la métrica general. Por ejemplo, **si necesitas buscar latencias de solicitud y comprender rápidamente la tendencia general, así como la distribución de todas las solicitudes, el mapa de calor resulta un gran aliado.** Además, permite usar colores para desambiguar la cantidad de cada sección de las barras.

Medidores

Los medidores se usan para mostrar una sola métrica de forma rápida, igual que un velocímetro muestra la velocidad, o un medidor de gas representa la cantidad de gas. **Al igual que ocurre con un medidor de gas, la mayoría de los medidores de monitorización indican claramente qué es bueno y qué no.** El código de colores es el indicador visual que permite al usuario interpretar el estado del sistema: el verde indica estabilidad, el naranja, riesgo inminente, y el rojo, estado crítico.

10.2. Herramientas de alerta

Bosun

Seguramente conoces Stack Overflow, que es muy popular entre los desarrolladores y también lo es entre los profesionales de operaciones. Stack Exchange es una red similar a Stack Oveflow que se ha encargado de crear, implementar y mantener a Bosun durante varios años. A finales de 2019, Skyscanner ha anunciado que asumiría las tareas y responsabilidades hasta entonces desarrolladas por Stack Exchange.

Al igual que Prometheus, Bosun está escrito en Go y tiene más funcionalidad que Prometheus, ya que **puede interactuar con muchos sistemas en contextos más complejos que la agregación de métricas** (es compatible con Graphite, InfluxDB, OpenTSDB y Elasticsearch) y, adicionalmente, **también puede consumir datos de sistemas de agregación de registros y eventos.** La arquitectura de Bosun consiste en: un único servidor, un backend como OpenTSDB, Redis y sus agentes recolectores. Estos últimos detectan automáticamente los servicios en un *host* e informan sobre las métricas para esos procesos y otras métricas adicionales que pueda originar el sistema. El servidor de Bosun consulta los *backends* para determinar si existe alguna situación que requiera el disparar una alerta.

Bosun también **puede ser utilizado por herramientas como Grafana para facilitar las consultas a los backends subyacentes a través de una única interfaz.** Otro caso de uso común es el uso de Redis para almacenar estados y metadatos de Bosun.

Por último, agregaremos que una característica realmente interesante de esta herramienta es **que permite validar las reglas de alertas contra datos históricos.** Esto es algo que se echa de menos en Prometheus y que, probablemente, incluyan en algún momento.

Cabot

Cabot fue creado por una compañía llamada Arachnys con el objetivo de poder monitorizar servicios y no solo las máquinas. Parece un concepto simple, pero es de una importancia crucial, ya que en los entornos actuales de nada nos sirve saber que una máquina está funcionando si los contenedores, servicios web o microservicios que aloja, han dejado de responder de forma correcta.

Cabot permite monitorizar servicios lógicos, que se ejecutan en una máquina o 100, como unidades lógicas. Es decir, no se preocupa de monitorizar las características físicas de un clúster (cosas como disco, memoria, etc.), sino que su trabajo se centra en recibir alertas cuando el tiempo de respuesta de su API deja de ser aceptable, por ejemplo.

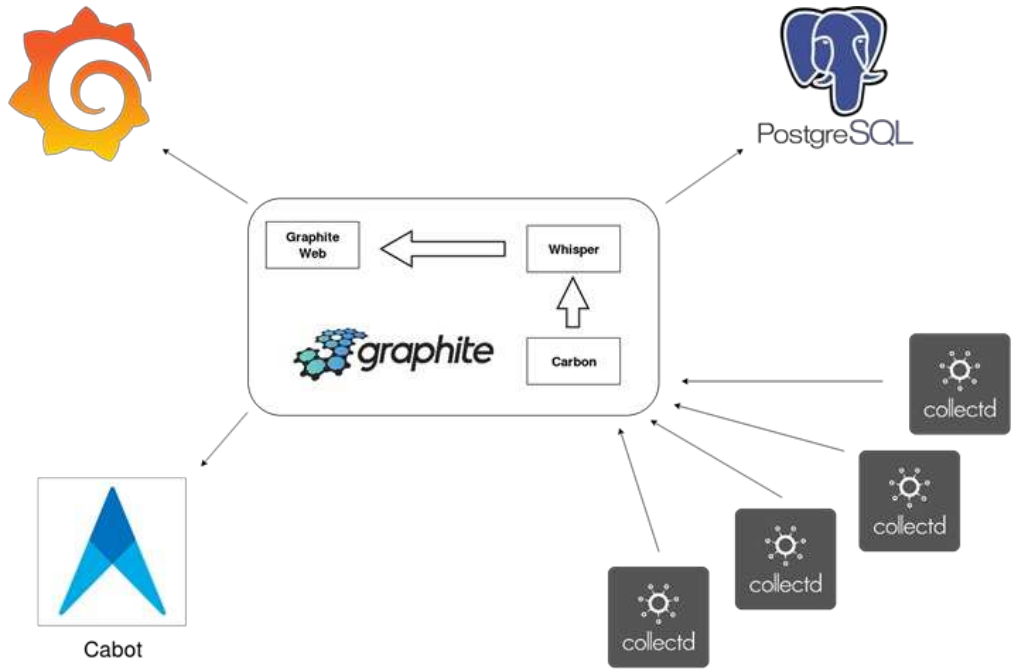


Figura 1. Sistema de alertas con Cabot y PostgreSQL. Fuente: Hackerearth, s.f.

Flame Graphs

Esta herramienta presenta una interfaz muy novedosa: gráficos de llamas. Si lo que buscas es realizar un análisis visual muy rápido y poco exhaustivo, esta no es tu

herramienta. En cambio, si lo que es deseas conocer en detalle las causas de un fallo y evaluar problemas específicos, puedes estar seguro de que sacarás partido de ella.

Creada en 2011, esta herramienta se centra en la CPU y memoria y las tramas de datos asociadas. El eje X enumera segmentos en orden, y el eje Y muestra la profundidad de la pila. Cada rectángulo es un marco de pila e incluye la función que lo llama. Cuanto más ancho es el rectángulo, más aparece en la pila. Este método es invaluable cuando se trata de diagnosticar el rendimiento del sistema a nivel de la aplicación.

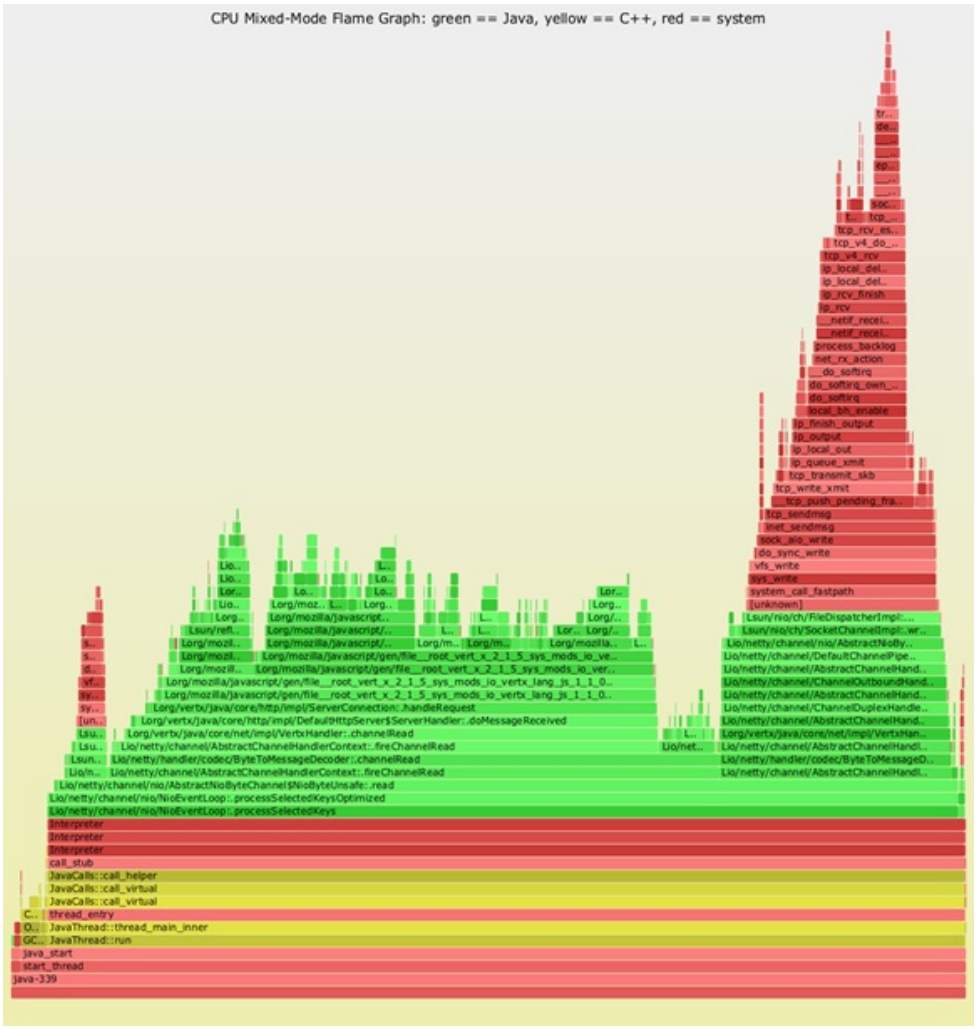


Figura 2. Ejemplo de FlameGraph. Fuente: The Netflix Tech Blog, 2015.

10.3. Herramientas de visualización

Grafana

Casi todo el mundo ha oído hablar de Grafana y, tal vez, muchos de ustedes incluso lo están utilizando en la actualidad. Esta herramienta fue creada por Torkel Ödegaard. En sus comienzos era un sistema de visualización para Kibana, al que Torkel modificó y posteriormente convirtió en Grafana.

El objetivo de Grafana es representar los datos de monitorización de una forma más usable y agradable. Una de sus ventajas más notables es que puede recopilar datos nativos de Graphite, Elasticsearch, OpenTSDB, Prometheus y InfluxDB. También existe una versión Enterprise que incluye complementos para que soporte más fuentes de datos, pero no hay razón por la que estos complementos no puedan ser también open *source*, o para que cualquiera pueda escribir nuevos complementos.

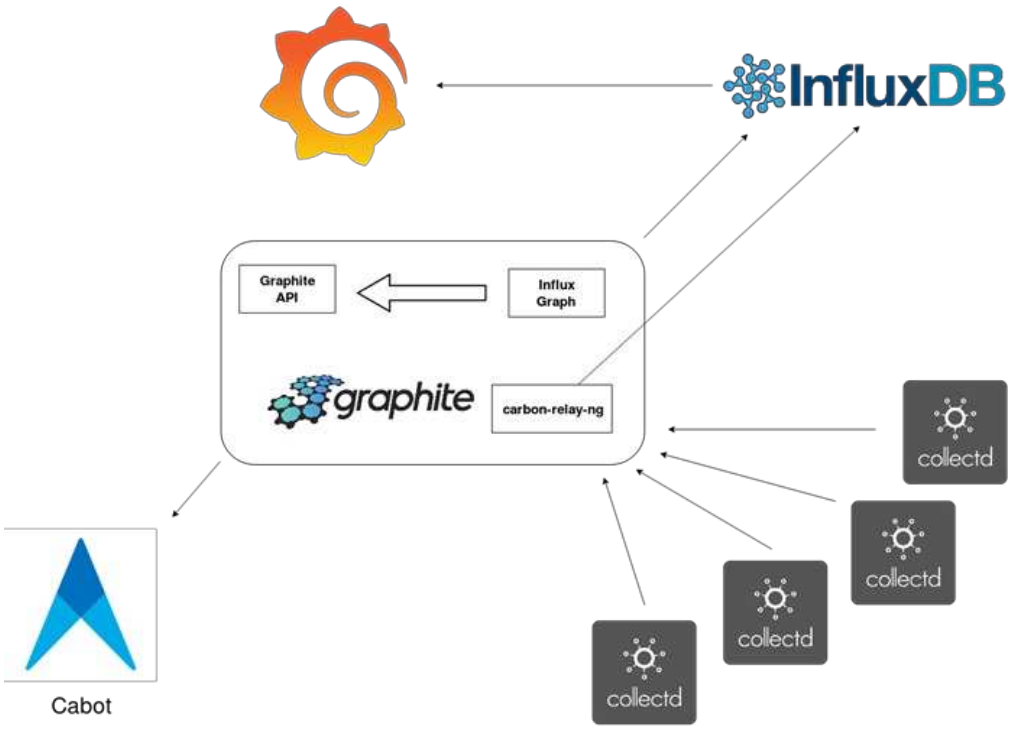


Figura 3. Sistema de alertas con Cabot e Influx DB. Fuente: Hackerearth, s.f.

Grafana está basado en la web, así que cualquiera puede acceder a la información, aunque también tiene sistemas de autenticación. Tiene la capacidad de proporcionar información relevante en apenas un vistazo porque integra muchos tipos diferentes de visualizaciones. Adicionalmente, Grafana ha empezado a incluir herramientas adicionales: ahora permite configurar alertas visualmente. Eso significa que puedes mirar un gráfico (que muestre dónde se debería haber activado una alerta debido a alguna degradación del sistema), hacer «clic» en el gráfico donde quieres que se active la alerta y luego indicarle a Grafana dónde debe enviar la notificación. Esto es tremendamente práctico y, aunque no necesariamente reemplaza a una plataforma de alertas, ciertamente ayuda a mejorar el servicio.

Figura 4. Ejemplo de Panel de Control con Grafana. Fuente: Wilson, 2021.

Grafana también ha introducido más funciones de colaboración con el tiempo: los usuarios pueden compartir paneles (*dashboards*), una característica muy popular entre las herramientas colaborativas de Software.

StatsAgg

StatsAgg es una plataforma de alertas (y también una plataforma de agregación de métricas) creada por Pearson, que más allá de ser una editorial, también tiene presencia en la web y una sólida relación con O'Reilly Media.

Su aportación más importante está relacionada con la colaboración y son, nada más y nada menos, que las anotaciones. Estas permiten a un usuario agregar contexto, a parte de un gráfico, y los demás usuarios pueden usar este contexto para comprender mejor el sistema. Esta es una característica de gran valor cuando un equipo está atravesando un incidente y la comunicación es crítica, ya que el contexto ayuda a que el conocimiento se comparta de forma rápida con todo el equipo.

Vizceral

Netflix creó Vizceral para **comprender más a fondo sus patrones de tráfico y, en concreto, mejorar los problemas que genera la conmutación de tráfico con errores.** A diferencia de Grafana, que es una herramienta más general, Vizceral es muy específica para los casos de uso de Netflix y si bien no es probable que sea la mejor elección para todo el mundo, sí es un buen ejemplo de las ventajas que proporcionan las herramientas de visualización.

10.4. Herramientas de rastreo distribuido

Las herramientas de rastreo distribuido permiten dar seguimiento a una búsqueda en los logs a través de un sistema de software distribuido en múltiples aplicaciones, servicios y bases de datos, así como capas intermedias y otras piezas de software. Esto permite profundizar en detalle sobre todo lo que sucede dentro de un sistema software.

Estas herramientas ofrecen representaciones gráficas que muestran cuánto tiempo ha tardado una solicitud en un paso determinado. De esta forma, el usuario puede determinar dónde está el problema que hace que el sistema experimente latencias o bloqueos. Es decir, en lugar de considerar al sistema como un árbol de búsqueda con muchas conexiones, en el momento en que las solicitudes comienzan a fallar, los desarrolladores y el personal de operaciones pueden ver exactamente dónde están los problemas (desde el principio). Esta información también puede revelar dónde podrían originarse los cambios de rendimiento tras un despliegue. Siempre es una buena estrategia tener herramientas que permitan identificar regresiones, automáticamente alertando de forma temprana sobre aquellos comportamientos anómalos antes de que impacten a los clientes.

¿Cómo funciona el rastreo?

El rastreo se basa en un identificador único que se asigna a cada solicitud y que, en general, se inyecta en los encabezados. Esta identificación registra de manera única, la cual se denomina «rastro». Este es la representación abstracta general de toda la transacción y se compone de tramos. Estos tramos representan al trabajo real que se realiza, como cuando se llama a un servicio o se hace una solicitud a la base de datos y poseen un identificador único. A su vez, estos tramos pueden crear tramos posteriores llamados tramos secundarios, y los tramos hijos pueden tener múltiples padres.

Ideas clave

Una vez que una transacción (o rastreo) ha seguido su curso, se puede buscar en cada capa y ver de forma global toda la ejecución. Hay múltiples herramientas que pretenden ofrecer esta funcionalidad y hablaremos de algunas de ellas a continuación.

API OpenTracing

OpenTracing es una especificación que surgió de Zipkin con el **objetivo de proveer compatibilidad multiplataforma.** Ofrece un proveedor neutro a nivel de API para agregar rastreo a las aplicaciones e integrar esos datos en sistemas de rastreo distribuido. Zipkin, Jaeger y AppDash son ejemplos de herramientas de código abierto que han adoptado esta especificación, pero incluso herramientas propietarias como Datadog e Instana también lo están haciendo. Se prevé que su expansión continúe hasta que sea un verdadero estándar de facto.

Herramientas disponibles

Zipkin

Desarrollado por Twitter, Zipkin está escrito en Java y puede usar Cassandra o Elasticsearch como *backends* escalables, lo que proporciona opciones válidas para, prácticamente, cualquier tipo de empresa. **El sistema consta de clientes, recolectores, un servicio de consulta y una interfaz de usuario web.** Los datos recopilados por cada reportero (*reporter*, cliente) se transmiten de forma síncrona a los recolectores. Los recolectores los almacenan en la base de datos y la interfaz de usuario web presenta los datos al usuario final en un formato consumible. La entrega de datos a los recolectores se puede realizar en tres formas diferentes: HTTP, Kafka y Scribe.

La comunidad Zipkin también ha creado Brave, una implementación de cliente Java compatible con Zipkin sin dependencias. Sin embargo, hay muchas otras implementaciones y Zipkin es compatible con el estándar OpenTracing.

Jaeger

Jaeger es un proyecto más reciente, creado por Uber Technologies. Está escrito en Go, por lo que no tiene problemas de dependencias específicas que tienen que ser instaladas como requisito previo en un servidor, ni sufre la sobrecarga de requerir un intérprete o una máquina virtual. Al igual que Zipkin, Jaeger también es compatible con Cassandra y Elasticsearch como almacenamiento. A su vez, también es totalmente compatible con el estándar OpenTracing (como ocurre con Zipkin).

La arquitectura de Jaeger es similar a la de Zipkin, con **clientes, recolectores, un servicio de consultas y una interfaz de usuario web, pero también tiene un agente en cada host que agrega localmente los datos.** El agente recibe los datos a través de una conexión UDP, los agrupa y los envía a un recolector. Este recibe los datos de forma resumida, lo que, en cierta forma, implica un ahorro. El servicio de consulta puede acceder al almacén de datos directamente y proporcionar esa información a la interfaz de usuario web.

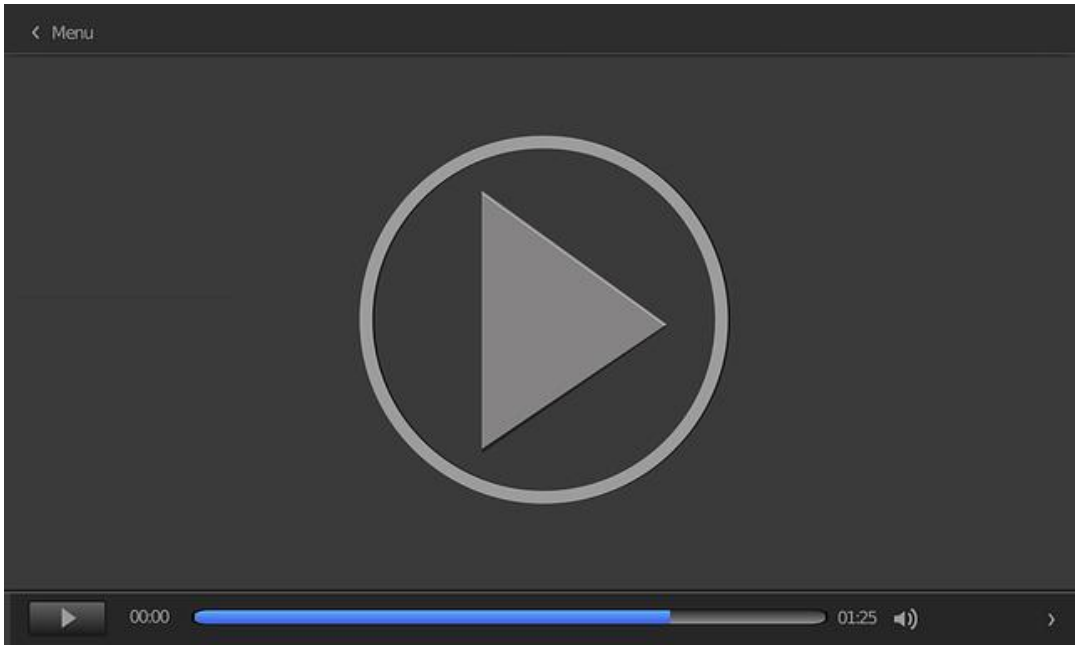
Por defecto, un usuario no obtendrá todos los rastros del Jaeger. El sistema muestrea 0,1 % (uno en 1000) de las trazas que pasan por cada cliente (el hecho de tener que mantener y transmitir todos los rastros sería un poco abrumador para la mayoría de los sistemas). Sin embargo, este porcentaje se puede aumentar o disminuir a través de la configuración de los agentes. **Jaeger utiliza el método de muestreo probabilístico, que trata de adivinar si un nuevo rastro debe ser muestreado o no.** El muestreo adaptativo está en su hoja de ruta, pero podrás ayudar a mejorar el algoritmo de muestreo agregando contexto adicional para que tome las decisiones más adecuadas.

AppDash

AppDash es un sistema de rastreo distribuido escrito en Go, como Jaeger. Fue creado por Sourcegraph y está basado en Dapper de Google y Zipkin de Twitter. Al igual que Jaeger y Zipkin, AppDash es compatible con el estándar OpenTracing, pero

requiere un componente adicional (lo que añade riesgo y complejidad). **A alto nivel, la arquitectura de AppDash consiste, principalmente, en tres componentes: un cliente, un recolector local y un control remoto recolector.** AppDash proporciona implementaciones en Python, Golang y Ruby.

En el vídeo *Open source* se define qué es Open source y por qué es un magnífico ejemplo de colaboración.



Accede al vídeo:https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=376b7d8d-52af-4567-a68a-ad78013d29c1

10.5. Referencias bibliográficas

Hackerearth. (s.f.). *Monitoring and alert system using Graphite and Cabot*. http://engineering.hackerearth.com/2017/03/21/monitoring-and-alert-system-using-graphite-and-cabot/

The Netflix Tech Blog (2015, junio 24). Java in Flames. *The Netflix Tech Blog*. http://engineering.hackerearth.com/2017/03/21/monitoring-and-alert-system-using-graphite-and-cabot/

Wilson, B. (2021, julio 2). 16 Best Container Orchestration Tools and Services. *Devopscube*. https://devopscube.com/docker-container-clustering-tools/

Tutorial de Grafana

Albert Coronado. (2018, julio 23). *Analítica y monitorización con la plataforma Grafana* [Vídeo]. YouTube. https://www.youtube.com/watch?v=g8l9i1dKqYI

Si bien este tema solo pretende dar una visión general de las herramientas disponibles sin entrar en la instalación u operación de las herramientas concretas, Grafana es una de las herramientas líderes en el mercado y este recurso te permitirá saber más.

Test

1. Selecciona la afirmación que mejor describe a AppDash.

   A. Es una forma de visualizar de forma rápida cualquier métrica.

   B. AppDash es un sistema de rastreo distribuido escrito en Go.

   C. A y D son incorrectas.

   D. Es un tipo de dashboard.

2. Selecciona la afirmación que mejor describe a Zipkin.

   A. Es una forma de visualizar de forma rápida cualquier métrica.

   B. Fue uno de los primeros sistemas de rastreo distribuido y fue desarrollado por Twitter.

   C. A y D son incorrectas.

   D. Es una herramienta de compresión de logs.

3. Selecciona la afirmación que mejor describe a los Flame Graphs.

   A. Son una herramienta de visualización de múltiples métricas que recibe ese nombre por su aspecto gráfico.

   B. Son una forma de visualizar de forma rápida cualquier métrica.

   C. A y D son incorrectas.

   D. Son un tipo de mapa de calor.

4. Selecciona la afirmación que mejor describe Cabot.

   A. Es una herramienta de visualización de métricas.

   B. Es una forma de visualizar de forma rápida cualquier métrica.

   C. A y D son correctas.

   D. No es un concepto pertinente para este tema.

5. Selecciona la afirmación que mejor describe Bosun.

   A. Es una herramienta de agregación de métricas, muy similar a Prometheus.

   B. Es una forma de visualizar de forma rápida una métrica concreta.

   C. A y D son falsas.

   D. No es un concepto pertinente para este tema.

6. Selecciona la afirmación que mejor describe el concepto de «medidor».

   A. Es una herramienta.

   B. Es una forma de visualizar de forma rápida una métrica concreta.

   C. A y D son correctas.

   D. No es un concepto pertinente para este tema.

7. Selecciona la afirmación que mejor describe un mapa de calor.

   A. Es una herramienta que permite saber la temperatura de los servidores.

   B. Es extremadamente útil en el caso de querer representar información preveniente de histogramas.

   C. A y D son correctas.

   D. El mapa de calor está definido sobre OpenTracing.

8. Selecciona la afirmación que mejor describe OpenTracing.

   A. Es una herramienta.

   B. Es un estándar de las API.

   C. A y D son correctas.

   D. ELK definió OpenTracing.

9. Selecciona la afirmación que mejor describe a la herramienta FlameGraf.

A. Es una de las herramientas de visualización desarrollada por Netflix.

B. Se trata de la detección de situaciones críticas en los gráficos de visualización.

C. A y D son correctas.

D. Es un tipo de gráfico que permite exponer información muy detallada normalmente exponiendo información de CPU y memoria.

10. Selecciona la afirmación que mejor describe Grafana.

A. Es una de las herramientas de visualización más usadas.

B. Es una metodología para mostrar logs y alarmas.

C. Es una herramienta que permite reunir y representar en un mismo lugar información de diferentes fuentes.

D. A y C son correctas.