Cloud Computing, DevOps y DevOps Culture

Tema 9. Monitorización

Índice

Esquema

Ideas clave

9.1. Introducción y objetivos

9.2. Herramientas de monitorización

9.3. Herramientas de agregación de logs

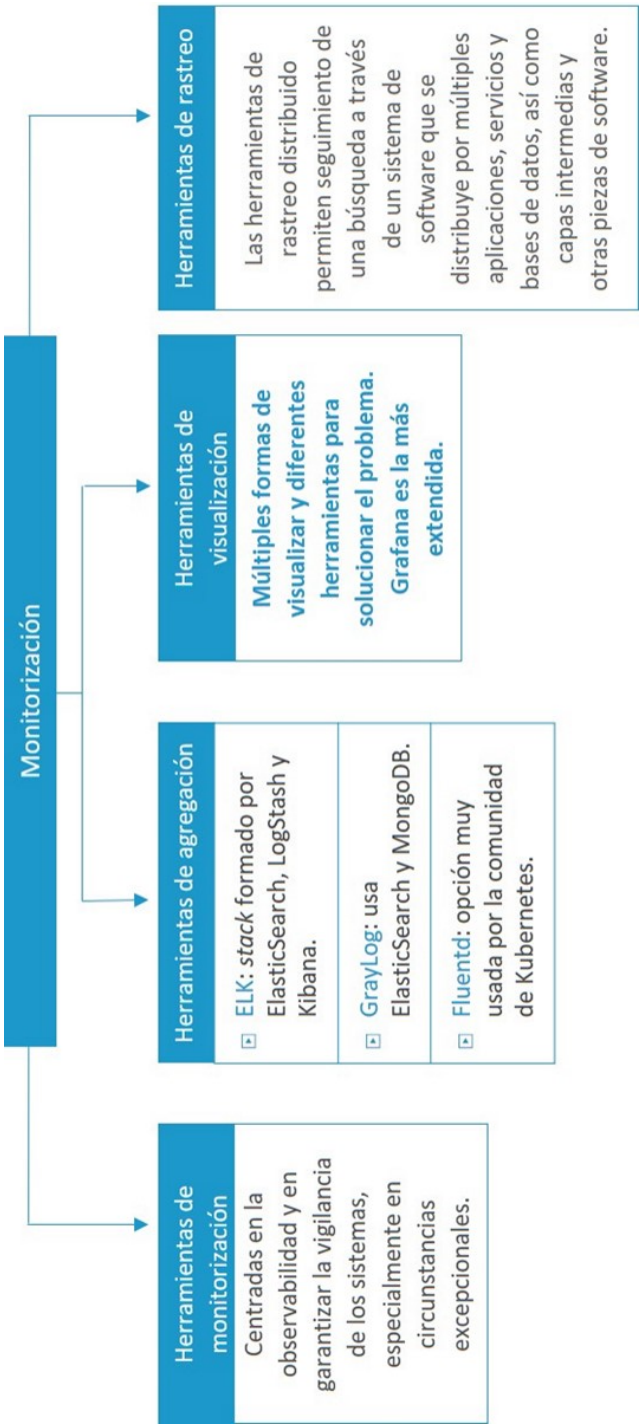
9.4. Referencias bibliográficas

A fondo

Tutorial de ELK

Test

Esquema



9.1. Introducción y objetivos

Los DevOps tienen, como parte de sus cometidos, no solo el desarrollar y desplegar los entornos de producción, sino el asegurar su correcto funcionamiento. Para hablar del correcto funcionamiento de un sistema, la monitorización es una pieza fundamental y necesaria. Empezaremos compartiendo algunos ejemplos del mundo real que nos ayudarán a ver la importancia de la monitorización para los DevOps.

Los objetivos que se pretenden conseguir a través del estudio de este tema son:

- Comprender la complejidad de establecer, gestionar y utilizar de forma eficiente un sistema de logs para aplicaciones complejas y multicapa.
- Entender la importancia de contar con herramientas de monitorización que permitan garantizar el mantenimiento de los sistemas durante veinticuatro horas al día y siete días a la semana.
- Descubrir herramientas que permitan abstraer la complejidad inherente de los logs mediante la agregación y el análisis automatizado.

Una historia sobre DevOps

Antes de comenzar, es importante mencionar que la historia aquí descrita es de la autoría de Dan Barker y puedes encontrarla en su lengua original en el artículo «The Open Source Guide to DevOps Monitoring Tools» (en la bibliografía podrás acceder al enlace).

«Érase una vez una aplicación con graves problemas de escalabilidad, carencia de recursos en varios órdenes de magnitud y muy poco tiempo para desarrollarse. No disponíamos de agregación de logs, ni agregación de métricas, ni información distribuida ni herramientas de visualización.Ante esta situación fue necesario trabajar directamente sobre los nodos reales de producción, utilizando herramientas como strace y grep, y accediendo fe forma manual a los registros de cada sistema. No pretendo decir que estas herramientas no son estupendas, pero no facilitan el análisis de registros distribuidos y, en consecuencia, no son las herramientas más apropiadas para estudiar el rendimiento de una aplicación moderna.Debido a ello se hizo necesario revisar una y otra vez los mismos registros múltiples veces. Creamos y probamos diferentes métodos, conectando y desconectando elementos de infraestructura y convirtiendo una tarea que debería ser sencilla en un verdadero trabajo detectivesco».

Problemas (aparentemente) aleatorios

«En otra ocasión y escenario diferente, estábamos teniendo problemas con una aplicación en producción que sufría problemas de memoria. Los departamentos de Operaciones e Ingeniería colaboraban estrechamente para buscar una solución.El problema no ocurría de manera consistente y era casi imposible relacionar los fallos con el tiempo de ejecución. Por ello, no podíamos predecir el fallo. Para empeorar más el panorama, el problema estaba extendido de forma sistemática en cientos de servidores y afectaba a otras aplicaciones.Afortunadamente, contábamos con herramientas de agregación de registros (logs), herramientas de rastreo distribuido, históricos, trazas y métricas. Todas estas herramientas estaban

conectadas a otras herramientas adicionales que facilitan las tareas de visualización.Gracias a las herramientas de visualización nos fue posible detectar una tendencia y un claro incremento en el uso de memoria. Allí buscábamos un pico que nos permitiese tener una alerta, a fin de diagnosticar el problema en tiempo real cuando este ocurriese.Al detectar la alerta, el sistema de registros agregados permitió correlacionar otros eventos en el sistema con el pico de memoria. A su vez, esto nos permitió buscar las llamadas relacionadas y detectar una en concreto que parecía estar funcionando incorrectamente, generando una transacción que intentaba cargar un archivo enorme. La carga nunca llegaba a producirse de forma exitosa y generaba el fallo de memoria. Una vez detectado el lugar donde se producía el error, nos resultó muy fácil analizarlo en detalle y crear una solución.Podría pensarse que la segunda situación ocurrió tiempo después de la primera y que hemos mejorado con el tiempo. O tal vez que se trata simplemente de diferentes empresas y trabajos, pero en realidad, la segunda situación tuvo lugar antes que la primera.De hecho, cronológicamente se trata de un primer trabajo en una empresa con una cultura de automatización bastante avanzada y que hace buen uso de las herramientas e integración, mientras que en el segundo trabajo (el primero en mi historia) no se contaba con herramientas que permitieran “observabilidad o visibilidad” de sus sistemas».

Observabilidad

La observabilidad no es solo un término comercial, sino que es un componente de control. Básicamente, **la observabilidad significa que se puede estimar un estado particular de un sistema basado en una salida.** En términos más generales, el estado de un sistema debería ser determinista de sus entradas. En un sistema

suficientemente complejo, puede ser en extremo dificultoso implementar la plena observabilidad. Sin embargo, debemos esforzarnos por tener observabilidad como mecanismo de control. El imperativo de esta necesidad es mucho más evidente cuando el sistema falla.

Tipos de herramientas de observabilidad

Comenzaremos por mencionar a la **agregación de métricas.** Este tipo de herramientas son, habitualmente, las que primero deberíamos incluir, ya que son fáciles de integrar en, prácticamente, cualquier aplicación escrita en un lenguaje moderno. La siguiente prioridad podría ser, por ejemplo, **monitorizar la sesión** porque, aunque probablemente requiera pequeñas modificaciones en las aplicaciones, proporciona un tremendo valor. El tercer nivel son las **alertas y sistemas de visualización** que, para su correcta implementación, se requerirá que las dos primeras herramientas (métricas y monitorización de la sesión) estén bien implementadas.

- **Agregación de métricas.** Este tipo de herramienta consiste un **conjunto de series de datos temporales.** Estas series son datos ordenados en el tiempo y tomados normalmente con un intervalo interno consistente. Su consistencia es la que permite aplicar cálculos avanzados a las series y proporcionar un análisis predictivo utilizando simples regresiones o también algoritmos más avanzados.

- **Agregación de registros.** Estas herramientas **interactúan con tipos de datos que están más relacionados con eventos** que con una serie de datos consistentes. Esta salida, a menudo, se emite cuando un sistema entra en un estado no deseado. Pensemos en ellos como una forma de componer información relevante.

- **Alertas/visualizaciones.** Puede parecer que esto no encaja con los otros tipos de herramientas, ya que es realmente posterior y más avanzado que las demás, pero proporciona una **forma eficiente de consumo para los otros tipos** e incluso puede producir su propia salida. Este tipo de herramienta hace que el sistema sea más comprensible y también ayuda a crear sistemas más interactivos a través de proactividad y reacción ante estados negativos del sistema.

- **Rastreo distribuido.** Al igual que el rastreo dentro de una sola aplicación, **el rastreo distribuido nos permite seguir una sola transacción, a través de una transacción que ocurre sobre el sistema completo.** Esto permite concentrarnos en transacciones específicas solo cuando estas podrían estar experimentando problemas. Por supuesto, es imposible trazar todo y, por lo tanto, (debido al rendimiento) lo más habitual y aconsejable es aplicar un algoritmo de muestreo.

Características comunes para DevOps

Existen numerosas características que deben estar presentes en cualquier tipo de herramienta de observabilidad. Mencionaremos algunas de ellas ahora y profundizaremos, también, más adelante:

- **OpenAPI:** esta especificación se llamaba anteriormente Swagger, pero se renombró cuando fue adoptado por la iniciativa OpenAPI dentro de la Fundación Linux. La especificación OpenAPI es una **herramienta independiente del lenguaje, que puede generar, automáticamente, documentación de métodos, parámetros y modelos.** Swagger se utiliza para generar interfaces RESTful en HTTP, pero es independiente del protocolo. Un usuario puede crear un cliente en casi cualquier lenguaje muy fácilmente, si todavía no existe uno. Cualquier herramienta que seleccionemos debería tener este tipo de *Application Programming Interface* (en siglas, API). Si una herramienta aún no la tiene, es posible que debamos buscar otras opciones. Las herramientas que no han implementado esta especificación, o no lo tienen en su hoja de ruta, probablemente tengan otras deficiencias al adoptar estándares de código abierto y especificaciones modernas.

Ideas clave

- **Open source:** hay muchas herramientas buenas que no son de código abierto y que pueden ser una opción adecuada para las empresas. Si eliges una de esas soluciones, asegúrate de que su documentación y las herramientas accesorias sean de código abierto. **La observación del código abierto de las herramientas puede proporcionar información valiosa sobre cómo están funcionando,** además de los otros beneficios derivados del open source.

- **Estándares abiertos:** independientemente de si una herramienta es de código abierto o no, siempre que sea posible, debe usar estándares abiertos. Ya hemos comentado uno de estos estándares, OpenAPI, pero hay muchos más. En lo sucesivo, profundizaremos un poco más en este tema para asegurarnos de saber que existen y dónde se usan.

- **Acceso fluido:** cuando mencionamos a la observabilidad y la transparencia, nos referimos a **que todos los usuarios implicados puedan ver los datos.** Las herramientas que elijas deben estar abiertas de forma predeterminada. Es posible que desees restringir algunas áreas, pero querrás facilitar el acceso y limitarlo, solo si es absolutamente necesario. Lo último que querrás son las barreras de acceso a la hora de intentar resolver un problema.

- **Federación:** es similar a la estrategia de acceso, pero permite **proporcionar información a todas las partes implicadas y controlar sus propias áreas de forma local.** Muchos sistemas heredados están diseñados de una manera que requiere que todos los datos fluyan a través de un sistema central, independientemente de la necesidad. Consecuentemente, esto también centraliza el control alrededor de esos datos. La federación habilita la agregación local en el sistema, el procesamiento y el control, mientras que la organización central recopila los mismos datos o datos resumidos. Este modelo aumenta la agilidad, flexibilidad y usabilidad.

Agregación de métricas

Analizaremos, a continuación, las características de la agregación de métricas y las **características de los datos de series temporales.**

Contadores

Un contador es **una métrica que representa un valor numérico que solo tenderá a aumentar** (en otras palabras, un contador nunca debería disminuir). Los contadores acumulan valores y presentan el total actual cuando el usuario lo solicita. Se utilizan para contabilizar: el número total de solicitudes web, el número de errores, número de visitantes, entre otros. Esto es análogo a una persona con un dispositivo contador en la entrada de un evento que contabiliza a todas las personas que ingresan a un establecimiento. Generalmente, no existe la opción de disminuir el contador sin tener que reiniciarlo.

Medidores

Un medidor es similar a un contador que representa un valor numérico, pero a diferencia del contador, puede también disminuir. Es una **representación de un valor en un momento determinado.** Un termómetro es un buen ejemplo de un medidor, porque se mueve hacia arriba y hacia abajo al medir la temperatura y ofrece una lectura de un punto en el tiempo. Otros usos incluyen el mostrar la carga de la UCP (CPU), uso de memoria, red y número de hilos.

Cuantiles

No son un tipo de métrica, pero están relacionados con las siguientes dos secciones: histogramas y resúmenes. Vamos a aclarar nuestra comprensión de los cuantiles con un ejemplo: el percentil. Este es un tipo de cuantil. Los percentiles son algo que vemos regularmente y deberían ayudarnos a comprender este concepto. Sabemos que un percentil tiene 100 valores posibles. A menudo, los vemos relacionados con pruebas médicas o resultados de rendimiento y, generalmente, afirmando la posición

de alguien dentro del percentil, por ejemplo 85 o algún otro valor. Esto significa que la persona que califica dentro de ese percentil tiene un valor real que cae dentro del 85 y 86 por ciento. Esta persona también obtuvo un puntaje en el 15 % superior de todos los estudiantes.

Histogramas

Es **una muestra de observaciones con varios contadores.** Estos cuentan todas las observaciones y se comportan como medidores que suman los valores de las observaciones y utilizan «cubos» (cada uno de los elementos del histograma) o agrupaciones para segmentar los valores y para vincular los conjuntos de datos de una manera productiva. Esto se ve comúnmente con cuantiles relacionados con el cumplimiento de acuerdos de nivel de servicio (en siglas, SLA). Vamos a suponer que queremos asegurarnos de que el 95 % de nuestras solicitudes son respondidas en un plazo de tiempo menor a 500 milisegundos. Podríamos usar un «cubo» con un límite superior de 0,5 segundos para recopilar todos los valores que caen por debajo de los 500 milisegundos. Entonces, podríamos determinar cuántas de las solicitudes totales han caído en ese cubo (sección del histograma). También podemos determinar qué tan lejos estamos de lo establecido en el *Service Level Agreement* (en siglas, SLA), pero esto puede ser difícil de hacer con precisión. Los histogramas son métricas agregadas que se acumulan desde múltiples instancias hasta un servidor central. Esto proporciona una oportunidad para entender el sistema como un todo, en lugar de nodo por nodo.

Resúmenes

Los resúmenes son similares a los histogramas en el sentido de que **unifican muchas observaciones, pero la agregación ocurre en el servidor.** Un resumen también usa una ventana de tiempo deslizante, por lo que sirve para casos ligeramente diferentes que un histograma, pero se usa para los mismos tipos de métricas. En la práctica, lo más usual es que se utilice un histograma, a menos que necesiten medidas más precisas.

Pull-push

No se puede escribir ningún texto sobre las herramientas de agregación de métricas sin abordar el debate *push vs. pull*. ¿Qué es? El debate se centra en si es mejor tener un sistema de agregación para que se envíen datos o tener un sistema de agregación de métricas que reúna datos preguntando a un API.

Hay muchas herramientas disponibles, tanto de código abierto como comerciales. Nos centraremos en herramientas de código abierto, pero algunas tienen un modelo de núcleo abierto con algunos componentes de pago. Además, algunas de estas cuentan con componentes adicionales de observabilidad, principalmente centrados en las alertas y la visualización.

9.2. Herramientas de monitorización

Prometheus

Esta es la **herramienta de monitorización de series temporales** más reconocida (y popular) y la solución ideal para aplicaciones nativas de la nube. Está alojada dentro del Cloud Native Computing Foundation (en siglas, CNCF), pero fue creado por Matt Proud y Julius Volz, y patrocinado por SoundCloud. Existe mucha documentación sobre Prometheus que te ayudará a entender cómo funciona, por ejemplo, visitando su página web.

Prometheus es un sistema basado en extracción (*pull*) que utiliza una configuración local para definir cómo recolectar los datos. Esta información se recopila y guarda en un motor de almacenamiento altamente eficiente, dentro del disco local. El sistema de almacenamiento utiliza un archivo único agregado por métrica. Este almacenamiento no es con pérdida (nos estamos refiriendo a que, en este caso, almacenamos todos los datos, no solo resúmenes o los datos más significativos), lo que significa que la fidelidad de los datos de hace un año es tan alta como la de los datos que se están recopilando en la actualidad. Sin embargo, es posible que un usuario no desee conservar tantos datos locales. Afortunadamente, hay una opción de almacenamiento remoto para retención y análisis de datos a largo plazo.

Figura 1. Ejemplo de datos de Prometheus con Grafana. Fuente: Admin Magazine, s.f.

Prometheus incluye un lenguaje avanzado para seleccionar y presentar datos llamados PromQL. La información se puede mostrar de forma gráfica, tabular o ser expuesta externamente a través de una API REST. El lenguaje permite a un usuario crear regresiones, analizar datos en tiempo real, o datos históricos de tendencias. Las etiquetas también son una gran herramienta para intercalar y consultar datos, y se pueden asociar con nombres de métricas, por ejemplo.

Prometheus también ofrece un modelo de federación, que favorece un control más localizado al permitir que los equipos tengan su propio Prometheus, mientras que los equipos centrales también pueden tener uno propio. Los sistemas centrales podrían atacar los mismos *end-points* que los Prometheus locales, pero también pueden atacar el local (del servidor central) para obtener los datos agregados de forma local.

Otro dato que resulta muy interesante sobre Prometheus es que dispone de un gestor de alertas: Alertmanager. Este sistema habilita la agregación de alertas, así como flujos más complejos para limitar cuando se envía una alerta. Por ejemplo, supongamos que diez nodos generan la misma alarma al mismo tiempo. Probablemente, no se necesite enviar una alerta por cada uno de los diez nodos,

sino que parece más eficiente enviar una única alerta que contenga toda la información. También, es posible enviar un correo electrónico al equipo de sistemas para que sepan que esos nodos están inactivos.

Graphite

Graphite existe desde hace mucho tiempo y se ha vuelto omnipresente en la industria. Es un sistema de inserción de logs que recibe datos desde las aplicaciones. El funcionamiento es relativamente simple, cada aplicación inyecta (*push*) los datos hacia el componente de Graphite. Carbon, por su parte, almacena estos datos en la base de datos Whisper, y esa base de datos junto con Carbon se leen desde el componente web Graphite, que actúa como interfaz de usuario para representar gráficamente sus datos en un navegador. En la versión de código abierto, todo se ejecuta en un host único: no hay datos o configuración almacenados en sistemas externos, por lo que es bastante fácil de administrar, pero es menos robusto que la versión comercial.

Figura 2. Ejemplo de salida producida por Graphite. Fuente: Montilla, 2017.

OpenTSDB

OpenTSDB es una **base de datos de series de tiempo de código abierto,** como su nombre lo indica (Open Time Series Data Base u OpenTSDB). Esta herramienta almacena sus métricas en Hadoop y esto significa que es inherentemente escalable. Si en tu entorno de trabajo tienes un clúster Hadoop, esta podría ser una buena opción para almacenar métricas a largo plazo. En el caso contrario, si no cuentas con un clúster Hadoop, la sobrecarga operativa podría ser demasiado grande. Sin embargo, OpenTSDB es compatible con Google Bigtable como *backend*, que es un servicio en la nube que no requiere mantenimiento.

Esta herramienta utiliza un sistema de emparejamiento clave-valor al que llama «etiquetas» para identificar métricas y agregar dimensionalidad. Tiene un lenguaje de consulta, pero es más limitado que Prometheus ('PromQL), aunque, sin embargo, tiene varias funciones integradas (muy interesantes) con el aprendizaje automático. La API es el principal punto de entrada para las consultas, similar a InfluxDB.

Por último, añadiremos que OpenTSDB no ofrece una capacidad de alerta, lo que dificulta la integración con la respuesta a incidentes que estén en proceso. Este tipo de sistema puede ser excelente a largo plazo. A diferencia de Prometheus, aquí se trata de garantizar el almacenamiento de datos para realizar un análisis histórico y revelar problemas sistémicos, en lugar de utilizarse para la identificación y respuesta rápida a problemas puntuales.

Estándar OpenMetrics

OpenMetrics está formado por un grupo de ingenieros que busca establecer un **formato de publicación estándar para datos de métricas.** Si esta iniciativa es exitosa, en el corto o mediano plazo, tendríamos una abstracción en toda la industria que nos permitiría utilizar herramientas y proveedores con facilidad, cambiando frecuentemente de unas a otras sin ninguna fricción. Empresas líderes como Datadog ya han comenzado a ofrecer herramientas que pueden consumir el formato de publicación de Prometheus.

InfluxDB

InfluxDB es un producto relativamente nuevo, más reciente que Prometheus, y que **utiliza un modelo de núcleo abierto (solo el producto básico es open source), pero con posible coste adicional, si se desea escalado y agregación.** InfluxDB es parte de la pila (*stack*) Telegraf, InfluxDB, Chronograf y Kapacitor (en siglas, TICK), por lo que incluiremos todas las características de esos componentes. InfluxDB utiliza un sistema de pares clave-valor llamado «etiquetas» para agregar dimensionalidad a las métricas, similar a Prometheus y Graphite. Los datos métricos pueden ser de tipo float64, int64, *bool* y *string* con resolución de nanosegundos, lo que es un rango más amplio que la mayoría de las otras herramientas en este apartado. De hecho, la pila TICK es más una plataforma de agregación de eventos que un sistema de agregación de series de tiempo nativa. InfluxDB utiliza un registro de escritura anticipada y una colección de archivos de datos de solo lectura. La arquitectura de esta pila es diferente dependiendo si se trata de la versión de código abierto o la comercial. El proyecto incluye Google e InfluxData (entre otros), lo que significa que InfluxDB, eventualmente, adoptará el estándar de OpenMetrics.

Tipos de agregación

Los sistemas de agregación de registros no pueden realizar las mismas tareas o funciones que los sistemas de agregación de métricas. Muchos vendedores presentan sus sistemas de agregación de registros como la solución a todos los problemas de observabilidad. A este respecto, es necesario tener en cuenta que, si bien la agregación de registros es una herramienta valiosa, no es una buena herramienta para obtener datos de series temporales. Algunas características valiosas en una serie temporal de métricas son la existencia de un intervalo regular y el sistema de almacenamiento personalizado, específicamente diseñado para series de datos temporales. Si se utiliza un sistema de agregación de registros en un intervalo regular, puede funcionar de forma similar a las métricas. Sin embargo, el sistema de almacenamiento no está optimizado para las consultas que son típicas en un sistema de agregación de métricas. Por tanto, es probable que un sistema de agregación no sea apto para series temporales de datos. Entonces te preguntarás: ¿para qué sirve?

Un sistema de agregación de registros sirve, principalmente, para coleccionar datos de eventos, es decir, actividades irregulares que son significativas. Un ejemplo podrían ser los registros de acceso para un servidor web, que resultan relevantes porque queremos saber qué o quién está accediendo a nuestros sistemas y cuándo. Otro ejemplo sería un error de aplicación, porque no es una condición de funcionamiento normal.

Recomendaciones para guardar registros de eventos:

- Incluye una marca de tiempo.
- Usa un formato orientado a documentos (JSON).
- No registres eventos insignificantes.
- Registra todos los errores de la aplicación.

- Incluye *warnings*.
- Incluye un registro de «encendido».
- Escribe mensajes que resulten legibles (por seres humanos).
- No registres datos informativos de producción.
- No registres nada que un humano no pueda leer o ante lo que no pueda reaccionar.

Costes de agregación en la nube

Al investigar las herramientas de agregación de registros, la nube podría resultarnos una opción atractiva. Sin embargo, **hay que tener en cuenta que puede traer acarreados costes significativos.** Los registros pueden representar una gran cantidad de datos cuando se agregan datos de cientos o miles de hosts y aplicaciones. La ingestión, almacenamiento y recuperación de esos datos puede llegar a ser muy costosa en sistemas basados la nube.

Como punto de referencia de un sistema real, una colección de alrededor de 500 nodos con unos cientos de aplicaciones resulta en 200 GB de datos de registro por día. Probablemente, hay margen de mejora para este sistema, pero incluso reduciéndolo a la mitad costará casi 10 000 dólares por mes en muchos softwares como servicio (en siglas, SaaS) especializados. Por supuesto, también hay que tener en cuenta que, en general, los costes tienden a bajar, que redunda en nuestro beneficio, pero al mismo tiempo nuestra necesidad de almacenamiento crecerá, con lo cual dicho beneficio se verá compensado.

9.3. Herramientas de agregación de logs

El listado de posibles herramientas no es muy largo, sobre todo si lo que pretendemos es que se puedan ver datos de las tendencias de más de un año atrás. Veremos, a continuación, algunas herramientas de código abierto y otras de pago pensadas para el despliegue en un entorno privado y otras que son ofrecidas como servicio.

ELK

ELK es la sigla correspondiente a Elasticsearch, Logstash y Kibana, y es **la herramienta de agregación de registros de código abierto más popular del mercado.** Netflix, Facebook, Microsoft, LinkedIn y Cisco la han incorporado a su conjunto de herramientas. Elástic es el responsable de desarrollar y mantener todos los componentes. Elasticsearch es esencialmente una base de datos No SQL Lucene con una implementación de motores de búsqueda. Logstash es un sistema de tuberías de registros que ingiere datos, los transforma y los carga en un almacenamiento como Elasticsearch. Kibana, por su parte, es una capa de visualización sobre Elasticsearch.

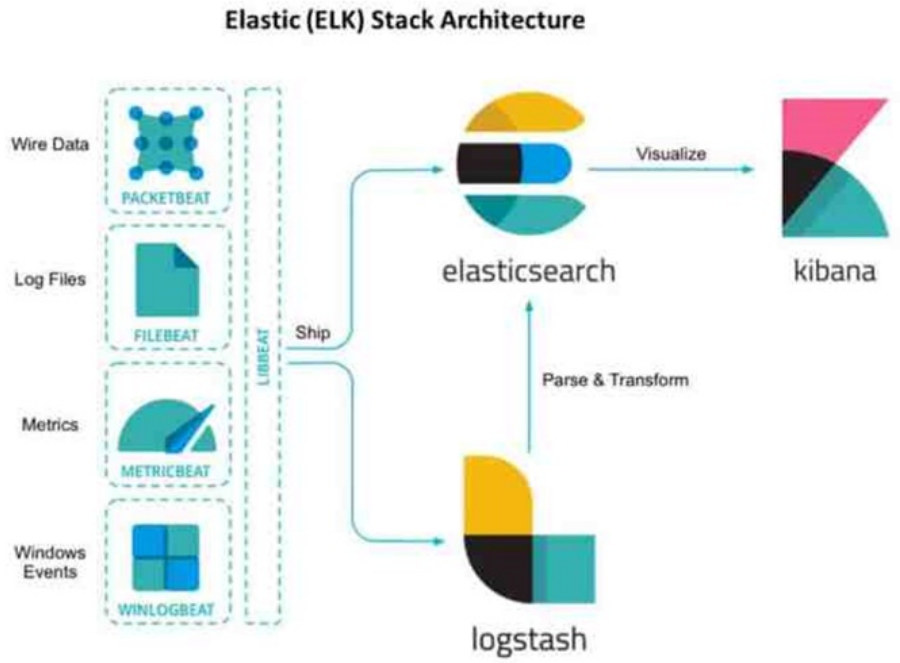


Figura 3. Arquitectura ELK. Fuente: SysAdminXpert, 2020.

Hace unos años se presentaron los «Beats». Estos son colecciones de datos que simplifican el proceso de envío de datos a Logstash. En lugar de necesitar comprender la sintaxis más apropiada para cada tipo de registro, un usuario puede instalar un Beat que exportará registros de NGINX o registros proxy de Envoy correctamente para que puedan ser usados efectivamente dentro de Elasticsearch.

Al instalar una pila ELK para dar servicio a un entorno de producción, a menudo se incluyen otras piezas como Kafka, Redis, y NGINX. Además, es común reemplazar Logstash con Fluentd, del que hablaremos más adelante. Este sistema puede ser complejo a la hora de operar y, de hecho, en sus primeros tiempos fueron muy frecuentes las quejas de los usuarios. Afortunadamente, muchos de los *issues* y fallos se han solucionado, pero sigue siendo un sistema complejo, por lo que es posible que no desees probar suerte si vas a implementarlo en una organización pequeña.

Dicho esto, **hay servicios disponibles que hacen que el usuario no tenga que preocuparse por su ejecución:** Logz.io es uno de ellos. De todos modos, como comentamos antes, para el caso de ciertos servicios en la nube, el coste para una gran cantidad de datos es relativamente elevado. En caso de que no pudieses permitirte Logz.io, una buena opción podría ser Amazon Elasticsearch Service que es un servicio de Amazon Web Services (en siglas, AWS) integrado con Lambda y S3. Esta es una opción, *a priori*, mucho más barata, pero se requiere algo de administración y tiene ciertas limitaciones.

Puedes obtener más información acerca del servicio Elasticsearch en Amazon Web Services a través de la siguiente dirección web:

https://aws.amazon.com/es/elasticsearch-service/

Elastic, la empresa matriz de ELK, ofrece un producto más robusto que utiliza el modelo de núcleo abierto (el módulo principal es *open source*, pero se lo pueden añadir componentes privativos) al que se le pueden añadir módulos comerciales con funcionalidad extra. También, se puede alojar en las plataformas Google Cloud o AWS. Esta podría ser la mejor opción, ya que esta combinación de herramientas y plataformas de alojamiento ofrece un precio más económico.

La pila ELK también ofrece excelentes herramientas de visualización a través de Kibana, pero carece de una función de alertas. Elastic proporciona funcionalidad de alertas dentro del complemento comercial X-Pack, pero no hay ninguna solución para la versión de código abierto. Yelp ha creado una solución a este problema, llamada ElastAlert.

Graylog

Graylog ha aumentado recientemente en popularidad, pero fue creado por Lennart Koopmann en 2010. A pesar de que su uso es cada vez mayor, todavía va muy por detrás de ELK. La razón principal es que tiene menos comunidad, soporte y

características, aunque puede usar los mismos Beats que la pila ELK. Graylog ha ganado elogios en la comunidad Go con la introducción del Graylog Collector escrito en Go.

En su *backend*, Graylog utiliza Elasticsearch y MongoDB. Esto lo hace tan complejo de ejecutar como la pila de ELK o, incluso, un poco más. Sin embargo, Graylog viene con alertas integradas en la versión de código abierto, así como otras características notables como geolocalización.

Uno de los mayores problemas con los sistemas de agregación de registros es la latencia, los *streams* eliminan ese problema en Graylog. Tan pronto como el registro entra, se puede enrutar a otros sistemas a través de un *stream* sin ser procesado totalmente.

La característica más sobresaliente de Graylog es la capacidad de geolocalización, que admite el trazado de direcciones IP en un mapa. Esta es una característica bastante común y también está disponible en Kibana, pero agrega mucho valor, sobre todo al estar disponible en la versión de código abierto. Esta compañía ofrece soporte a los usuarios por un coste adicional. También ofrece un modelo abierto (*open source* con funcionalidad básica) y la posibilidad de ampliar a una versión *Enterprise* que ofrece registros de auditoría y soporte adicional. No hay muchas otras opciones de soporte o alojamiento, fuera de las oficiales.

Fluentd

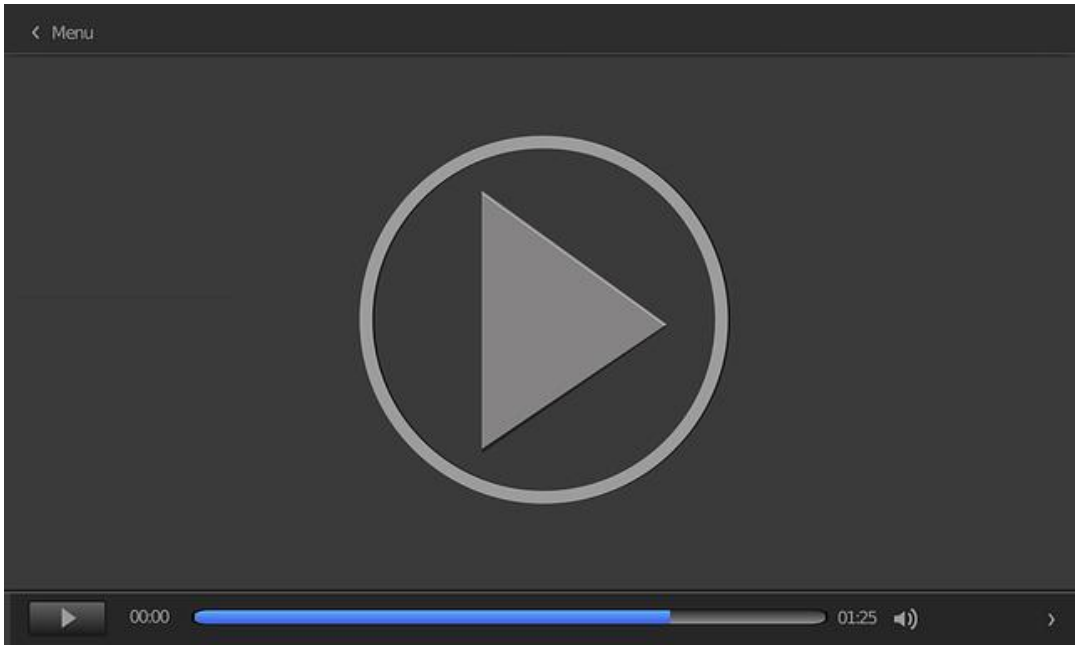
Fluentd fue desarrollado por Treasure Data. Está escrito en C y es una de las herramientas de referencia recomendadas por AWS y Google Cloud. Para muchas empresas, se ha convertido en un posible reemplazo para Logstash. **Actúa como un agregador local que recopila todos los registros de nodos y los envía a los sistemas de almacenamiento centralizados.** Utiliza un robusto sistema de complementos para proporcionar variadas integraciones que lo hacen compatible con numerosas fuentes de datos y formatos de salidas. Hay más de 500 complementos

diferentes disponibles, así que los casos de uso más comunes están totalmente resueltos. Y, de hecho, en el caso excepcional de que no exista, el usuario puede optar por contribuir y construir uno.

Fluentd es una opción muy común en el entorno de Kubernetes, debido a sus bajos requisitos de memoria (solo decenas de megabytes) y su alto rendimiento. En un sistema basado en Kubernetes, donde cada pod tiene auto escalabilidad, el consumo de memoria aumentará linealmente con cada nuevo pod creado. Su utilización Fluentd reducirá drásticamente el uso de recursos del sistema, a diferencia de lo que ocurre con algunas aplicaciones java que no han sido diseñadas con esta capacidad.

Ideas clave

Los microservicios son una arquitectura en auge en la actualidad y están siendo adoptados por múltiples equipos en todo el globo. Sin embargo, una incorrecta estrategia sobre la aplicación de los microservicios puede tener un impacto negativo de gran escala. En el vídeo *Microservicios* analizaremos los errores más comunes que todo desarrollador debe evitar.



Accede al vídeo:https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=9db6635d-5b11-49fc-8962-ad78013d2a25

9.4. Referencias bibliográficas

Admin Magazine. (s.f.). Efficiently planning and expanding the capacities of a cloud. *Admin Magazine*. https://www.admin-magazine.com/Archive/2018/44/Efficiently-planning-and-expanding-the-capacities-of-a-cloud/(offset)/6

Montilla, I. F. (2017, febrero 11). Scaling django: measuring your django app's performance. Dev Community. https://dev.to/iferminm/scaling-django-2-profiling-your-django-app-4in6

SysAdminXpert. (2020, junio 12). *ELK Stack Architecture Elasticsearch Logstash and Kibana*. https://sysadminxpert.com/elk-stack-architecture-elasticsearch-logstash-and-kibana/

Tutorial de ELK

Paradigma Digital. (2019, abril 5). *Curso Elastic Search – Capítulo I* [Vídeo]. YouTube. https://www.youtube.com/watch?v=UIN2NeMb7xc

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, ELK es una de las herramientas líderes en el mercado, y este recurso te permitirá saber más sobre eso.

Test

1. Selecciona la definición correcta de histograma:

   A. Un histograma muestra los datos acumulados de tipos de datos.

   B. Es un tipo de gráficos.

   C. Facilidad de encontrar la información de forma rápida.

   D. Ninguna de las anteriores.

2. Selecciona la definición correcta de observabilidad:

   A. Característica que permite estimar un estado particular de un sistema basado en una salida.

   B. Capacidad de facilitar la visión de los datos.

   C. Facilidad de encontrar la información en la herramienta.

   D. Ninguna de las anteriores.

3. Selecciona la definición correcta de agregación de métricas:

   A. Es una herramienta que generalmente se utiliza para analizar series de datos temporales.

   B. Capacidad de facilitar la visión de los datos agregados.

   C. Facilidad de encontrar la información en la herramienta.

   D. Ninguna de las anteriores.

4. Selecciona la afirmación que se mejor describa el rastreo distribuido:

   A. El rastreo distribuido es similar al rastreo estándar pero la carga se reparte.

   B. El rastreo distribuido te permite seguir una sola transacción a través de una transacción que ocurre sobre el sistema completo.

   C. Las respuestas A y B son correctas.

   D. Ninguna es correcta.

5. Selecciona la afirmación que mejor describe Prometheus.

A. Esta es la herramienta de monitorización de series temporales más reconocida.

B. Una herramienta de monitorización.

C. Las respuestas A y B son correctas.

D. Ninguna es correcta.

6. Selecciona la afirmación que mejor describe la federación desde el punto de vista de la monitorización.

A. La federación permite tener control y datos globales, pero manteniendo el control local.

B. El modelo de federación tiene que ver con la gobernanza de IT y, por lo tanto, no está relacionado con DevOps.

. Las respuestas A y B son correctas.

D. Ninguna es correcta.

7. Selecciona la afirmación que mejor describe las implicaciones del Open Source.

A. El software es gratis y puedes hacer lo que quieras.

B. El desarrollador del software lo pone a disposiciones de los usuarios bajo una licencia concreta donde, entre otros derechos, existe el acceso al código fuente.

C. El software está desarrollado siguiendo estándares que garantizan su compatibilidad.

D. Ninguna es correcta.

8. Selecciona la afirmación que mejor describe a la Open API.

A. Es un API que está abierto, es decir, sin securizar.

B. Es un API que no contempla federación.

C. OpenAPI es un estándar abierto para las API.

D. A y B son correctas.

9. Selecciona la afirmación que mejor describe la herramienta ELK.

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

B. Es un protocolo de transmisión de logs.

C. A y D son correctas.

D. ELK es el stack conformado por ElasticSearch, LogStash y Kibana.

10. Selecciona la afirmación que mejor describe el término pull-push.

A. Es una de las herramientas de monitorización más usadas junto con ELK.

B. Es un protocolo de recepción de logs.

C. Se refiere al dilema sobre las ventajas e inconvenientes de construir sistemas que llaman a otro para solicitar información o por el contrario esperan a ser informados.

D. Ninguna de las anteriores.