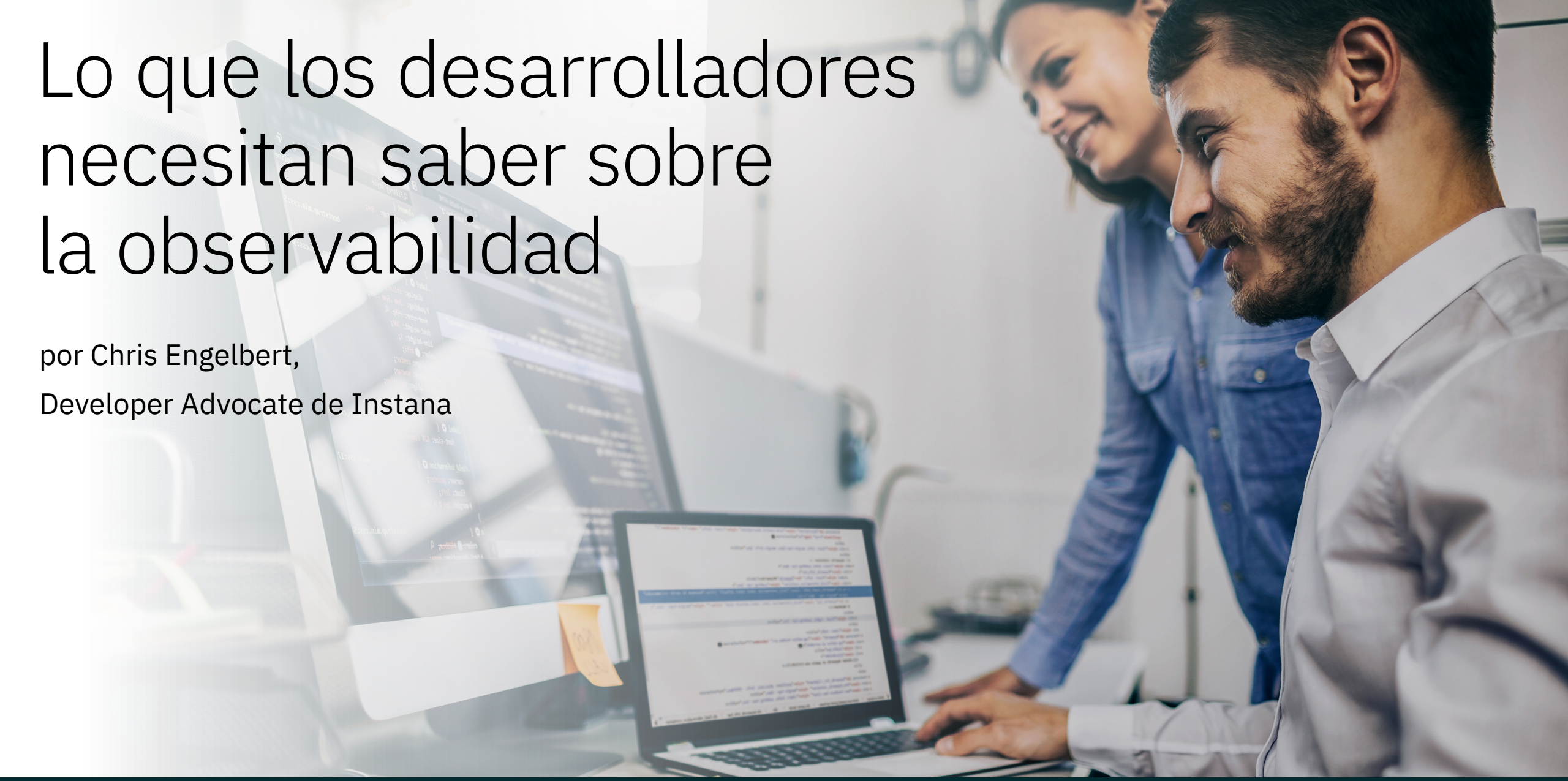
Lo que los desarrolladores necesitan saber sobre la observabilidad

por Chris Engelbert,

Developer Advocate de Instana



Contenido

01 Introducción

Como desarrolladores, podemos recordar la época en que Nagios era tecnología punta. Desde entonces, el término "supervisión" lleva asociadas algunas malas implicaciones. Examinar paneles de control cargados de diferentes gráficos y números no nos ayudaba a resolver los problemas y, aún así, se nos seguía pidiendo que los examináramos. Sin duda, esos paneles de control nos decían que algo iba mal, pero nuestro trabajo no consistía en que funcionara el sistema, ¿verdad? Al cabo de poco tiempo, terminamos fatigados con tantas métricas.

¿Por qué Nagios no funcionó mejor para los desarrolladores? Aparte de las métricas de rendimiento recopiladas directamente de los contadores integrados en el flujo de código de las aplicaciones, el trabajo era todo de supervisión de la caja negra únicamente, y consistía principalmente, de pings a hosts y conexiones a servicios, acompañados a menudo, de algún análisis de expresión regular de registro.

Por si fuera poco, antiguamente en los servicios de operaciones, el desarrollo y las operaciones eran dos mundos que estaban separados. Teníamos administradores de base de datos, lo que impedía a todos los demás ejecutar incluso una simple sentencia por su cuenta.

El equipo de operaciones, operando el sistema en producción, encargándose de los fallos del sistema e informando sobre los problemas a ingeniería, se pasaba el tiempo manteniendo a los desarrolladores alejados de los sistemas de producción. En el mejor de los casos, solo permitían el acceso a un número muy limitado de desarrolladores.

Por último, estaba el departamento de ingeniería. Nuestro trabajo consistía en crear características, que se "probaran" y eso era todo. Posteriormente nos involucramos en la corrección de errores. Para nosotros, el código fuente era un arte en sí mismo. Por supuesto, también manteníamos bien lejos a cualquier persona que no fuera desarrollador.

Hacer funcionar y controlar el sistema simplemente no formaba parte de nuestro trabajo. A excepción de una parte específica: agregar fragmentos de control a nuestro código fuente, algo parecido a colocar un revestimiento de aluminio en la Capilla Sixtina. Lo odiábamos por todas las razones válidas. Se extendía como un reguero de pólvora.

Las responsabilidades se dividieron y, durante mucho tiempo, los ingenieros de software escribían código y se iban a casa. A partir de ahí, Operaciones se hacía cargo y tenía que lidiar con todo lo que salía mal.

Pero, los tiempos han cambiado. Hoy en día, los sistemas tienen un aspecto diferente, las implementaciones funcionan de forma distinta y las fronteras entre equipos son difusas, si es que no se han eliminado por completo. Dicho esto, como desarrolladores, estamos más cerca de las operaciones que nunca y somos una parte indirecta del funcionamiento de los sistemas.

Con la complejidad de los sistemas modernos que ejecutan microservicios, un "control significativo" se ha convertido en una parte importante de nuestras vidas. Para impedir la sobrecarga de métricas del pasado, necesitamos examinar los beneficios de la observabilidad.

Este libro electrónico examina el nuevo mundo. Dejaremos atrás los sentimientos negativos sobre el control y daremos nuestros primeros pasos en el mundo de la observabilidad y su creciente importancia para los desarrolladores.

02 Terminología sobre observabilidad

En primer lugar, empecemos por la terminología básica sobre observabilidad. Para profundizar en el mundo de la observabilidad, es importante entender las diferencias entre control y observabilidad, así como la forma en que se representan los datos para el usuario.

Control

Tradicionalmente, el control se ha centrado en las métricas de series temporales. El proceso era siempre el mismo: recopilar un montón de métricas, colocarlas en gráficos en paneles de control, averiguar para qué métricas se deben establecer alertas y elegir algunos umbrales para las alertas. Este método, si bien era mejor que nada, distaba mucho de ser ideal, ya que se generaban demasiadas o muy pocas alertas, había una falsa sensación de seguridad y se dedicaba demasiado tiempo al proceso de resolución de problemas.

Aunque ampliemos el control para incluir el estado y el rendimiento de los servicios, el control tradicional emplea un método de alerta basado en síntomas. Los problemas son los síntomas a los que reacciona. Si un servicio externo se vuelve inalcanzable, es un síntoma. En lugar de responder a los síntomas, debería hacerse hincapié en la recopilación de datos relevantes que tengan un contexto integrado en todas partes.

Observabilidad

Aunque la observabilidad no es nada nuevo, es el complemento perfecto del control. Puede considerarse como un superconjunto.

El término observabilidad implica que observamos algo. La parte importante aquí es observar a un nivel más pormenorizado que un mero control. Sin dejar de incluir todos los números y gráficos del control, la observabilidad añade el conocimiento de lo que es importante supervisar en todos los diferentes equipos previamente separados.

Nadie necesita cientos de gráficos y tablas, sobre todo si nunca ayudan a resolver los problemas. Por otro lado, es posible que ni siquiera se hayan recopilado los datos necesarios para ayudar a resolver un problema.

Además de este problema, la observabilidad añade rastreo distribuido, básicamente, rastreo de una pila de microservicios. Explicaremos lo que eso significa. Y no lo olvide: los análisis de registros significativos van mucho más allá de una simple búsqueda basada en expresiones regulares (regex).

02 Terminología sobre observabilidad

Los dos últimos elementos se han concebido para llevar el control al interior de las aplicaciones y los servicios. A diferencia del simple control, recopilamos información directamente desde el interior del servicio y la juntamos con todo lo que sabemos sobre el sistema.

La observabilidad está diseñada con el conocimiento de los dominios de error conocidos de nuestro sistema, por ejemplo, un servicio que se conecta a otro servicio a través de HTTP puede fallar al conectarse por varias razones, eso, es un dominio de error conocido. Los dominios de error pueden depender de la forma en que las personas piensan sobre el servicio o el sistema en conjunto. Por esa razón, es importante tener en cuenta diferentes perspectivas a la hora de diseñar los dominios de error y el contexto asociado a ellos, por ejemplo, para facilitar la depuración, dependiendo de a quién vaya destinada.

En cuanto a la observabilidad, tenemos tres pilares que ofrecen una visión y comprensión de nuestro problema: las métricas de estado y rendimiento, los rastreos y registros distribuidos.

Observabilidad en la empresa
En la empresa, la observabilidad debe ser altamente escalable para gestionar las complejidades y la escalabilidad de los sistemas transitorios. Los sistemas diseñados para la observabilidad en la empresa se adaptan automáticamente a los constantes cambios en el entorno de servicio o la infraestructura. Parte de este proceso consiste en descubrir nuevas instancias o servicios o bien cerrar instancias o servicios sin intervención manual. Normalmente proporcionan bibliotecas de envoltorios para realizar cambios mínimos y no intrusivos en el código fuente, llegando incluso hasta la instrumentación de código totalmente automática para añadir mediciones y sondeos de estado a una aplicación que ya está en ejecución.

La instrumentación de código proporciona una forma de añadir los puntos necesarios iniciales y finales de las operaciones en el código base en ejecución sin añadirlos manualmente al código fuente. Dicho esto, los desarrolladores se ven en gran parte liberados del tedioso trabajo que supone añadir elementos de control y rastreo a su código base, lo que les deja más tiempo para trabajar realmente en los casos prácticos de la empresa.

Además, la creación de perfiles de código con poca carga, pero siempre activos, de los sistemas de desarrollo, transferencia y producción, siempre ofrece una mayor perspectiva del rendimiento de los servicios en condiciones reales, algo que los desarrolladores han echado de menos durante mucho tiempo. Los sistemas siempre se han comportado de forma diferente a nuestras expectativas en los entornos de producción.

Sin embargo, después de recopilar todos esos datos, es importante no limitarse a visualizarlos a la antigua usanza con números, tablas y diagramas. La visión más general es que el contexto de los datos es lo importante para resolver problemas, especialmente para los desarrolladores.

Para que las cantidades masivas de datos sean útiles, las soluciones de observabilidad de la empresa deben proporcionar una correlación automática, lo cual facilita la comprensión y crea información útil. También suelen contribuir al análisis de la causa raíz proporcionando el contexto necesario en torno a los eventos y la información correlacionados.

03 Rastreo distribuido

Los rastreos distribuidos son para los sistemas distribuidos los que las pilas de rastreo son para las aplicaciones y las excepciones o situaciones de pánico. Es una técnica para capturar y cronometrar los controladores de servicio y las llamadas internas mientras una solicitud se abre camino por un entorno de sistemas para generar la respuesta. Por esa razón, a veces también recibe el nombre de rastreo de solicitud distribuido.

El rastreo distribuido ayuda a los desarrolladores a analizar los flujos de solicitud y a precisar la causa raíz de los problemas o los cuellos de botella de rendimiento.

Imagine que un usuario llama a un servicio de atención al usuario para solicitar los datos de su cuenta. El propio servicio llama a unos cuantos servicios de la pila para recuperar diferentes tipos de información. En la figura 1 se muestra el flujo básico de la llamada. Aunque la información, como, por ejemplo, quién llama a quién, es visible, en el diagrama faltan datos importantes como la información de tiempos.

Este escenario muestra dónde entra en juego el rastreo distribuido. Las figuras 2 y 3 muestran el flujo de llamadas de dos maneras diferentes, como una barra de temporización y como una vista de árbol similar a un rastreo de pila.

Estas dos vistas divulgan la información necesaria para que los desarrolladores encuentren llamadas lentas, operaciones de larga duración, servicios que fallan y dónde se producen esos fallos.

Figura 1. Una solicitud a un servicio al usuario que se mueve a lo largo de los servicios internos

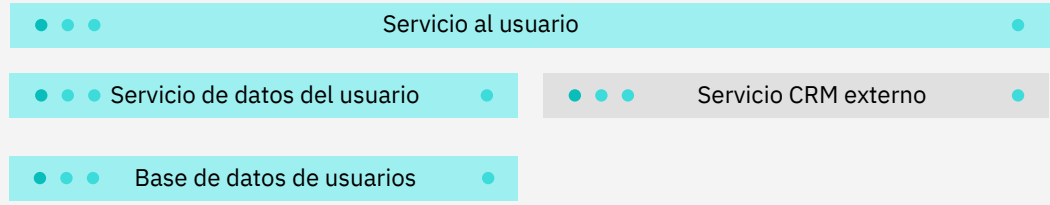


Figura 2. El flujo de llamadas como diagrama de temporización y jerarquía

Figura 3. Diagrama de jerarquía del flujo de código

03 Rastreo distribuido

Para representar esa información, el rastreo se subdivide en elementos, como los intervalos y las llamadas.

Un intervalo, a veces llamado intervalo de tiempo, representa el tiempo de ejecución de una única operación dentro del flujo del rastreo distribuido. En este contexto, una operación es una ejecución de código con una hora de inicio y una hora de finalización así como posibles intervalos padre e hijo.

Un intervalo contiene además metadatos para proporcionar contexto alrededor del propio intervalo real. Para conectar varios intervalos se utilizan identificadores (span-id y trace-id) para construir las jerarquías padre/hijo.

La comunicación entre dos intervalos se representa mediante llamadas. Las llamadas contienen información acerca de qué tipo de conexión se ha utilizado, información de cabecera y datos de respuesta, como, por ejemplo, los códigos de estado HTTP si una llamada no se ha realizado correctamente. También se puede añadir información de contexto personalizada.

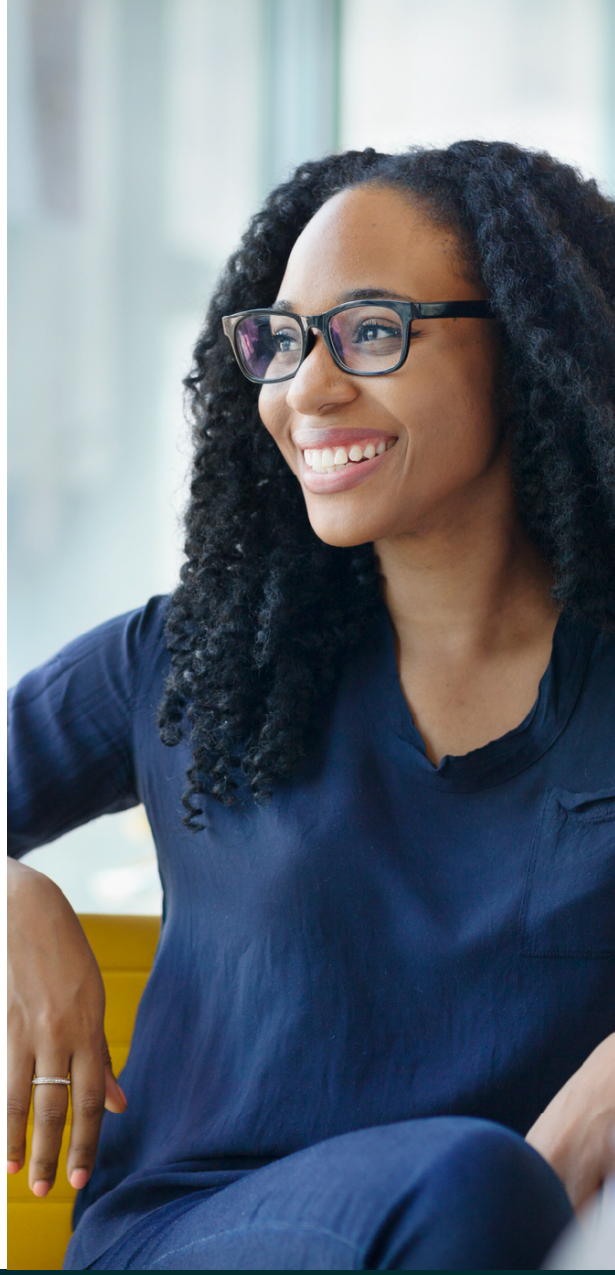
Importancia de la observabilidad

Cuando se trabaja con sistemas distribuidos, los desafíos son muy diferentes a los de las aplicaciones monolíticas que creábamos en el pasado. Aunque nos gusta pensar en nuestro mundo en términos sencillos, la realidad es muy diferente, a veces sin que lo sepamos.

Muchos sistemas y servicios diferentes funcionan de forma independiente y simultánea. La mayoría de los servicios siguen teniendo una o más subllamadas a otros servicios o bases de datos. Comprender la interacción entre esos servicios es uno de los elementos más importantes para un desarrollador cuando trata de arreglar un error o mitigar un problema de rendimiento.

También es importante no tener demasiados gráficos de métricas en los paneles de control porque sino retrocederemos al estado en el que estábamos hace años. Lo que buscamos es información relevante que nos lleve a la causa de un problema lo más rápidamente posible. Necesitamos que el sistema nos diga qué métricas son importantes para el dominio de error de cada servicio específico. Lo importante aquí es centrar nuestra atención únicamente en lo que importa.

Y esa atención tiene sentido porque no hay ninguna garantía de que vayamos a terminar con un sistema libre de fallos, independientemente de cuántas métricas añadamos. Sucede más bien al revés; los fallos en nuestro sistema son inevitables. Escribí sobre esta cuestión en una publicación de blog titulada Building Resilient Applications-Embrace the Failure (Creación de aplicaciones resistentes: aceptar el fracaso), que describe exhaustivamente este tema. Lo mejor que podemos hacer es prepararnos para este caso comprendiendo el dominio de error que he mencionado antes.



03
Rastreo distribuido

Al final, el elemento importante para nosotros en tanto que desarrolladores es volver a nuestra creatividad y motivación. Queremos trabajar en los elementos que hacen avanzar la empresa, nuestros casos prácticos empresariales. No queremos saturar nuestro código fuente con millones de métricas o puntos de rastreo. Y lo más importante; no queremos estar constantemente luchando por apagar fuegos o corregir errores.

En el último punto, sin embargo, un sistema de observabilidad limpio, útil y que proporcione información puede ayudar con todas esas tareas. Proporcionando librerías de envoltorios delgados o, mejor aún instrumentación automática, podemos mantener nuestro código fuente base limpio, fiel al caso práctico empresarial. También nos permite recuperar mucho tiempo: el tiempo que necesitábamos para analizar problemas complejos en sistemas distribuidos, el tiempo que utilizábamos para añadir puntos de métricas, el tiempo que hemos empleado en comprender la interacción y comunicación entre sistemas, y por último, el tiempo perdido en buscar cuellos de botella. Hoy en día, un rastreo distribuido puede ayudarnos a comprender las deficiencias y evoluciones del sistema actual antes de que nos encontremos con cuellos de botella o problemas de escalabilidad.

Tiempo de resolución

Cuando un despliegue falla o nuestra nueva versión se comporta de forma errática, tenemos que ser rápidos. Se trata de retroceder a una versión más antigua o analizar el problema, encontrar un arreglo, implementarlo y avanzar.

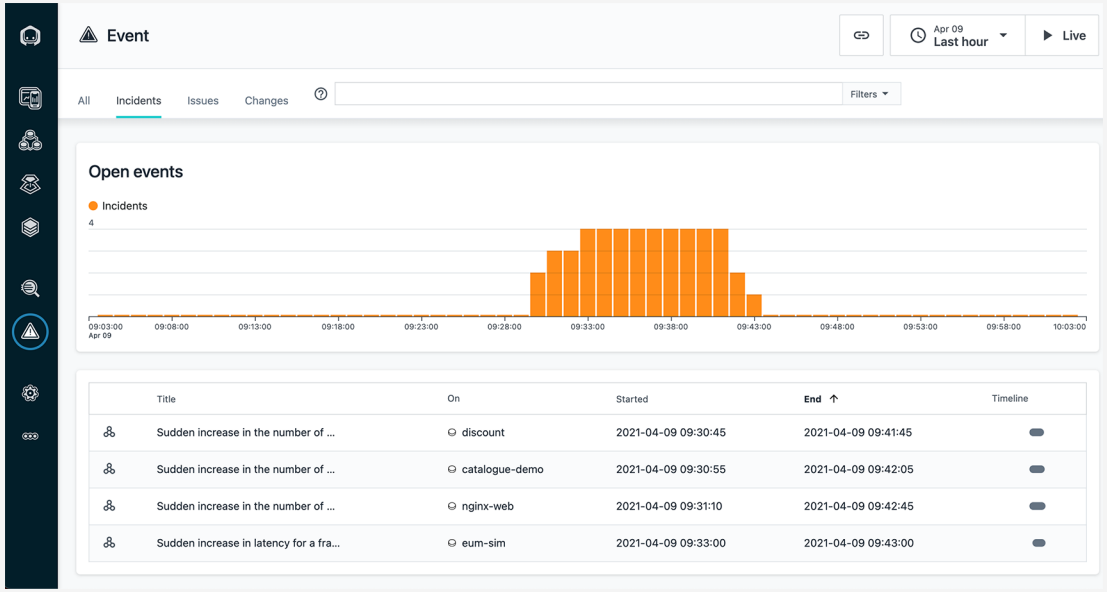


Figura 4. Visión rápida del estado de un servicio con marcadores. Fuente: Instana.

03 Rastreo distribuido

A veces, el problema es tan solo un pequeño descuido, reproducible localmente y rápidamente solucionable. Pero se vuelve más complejo cuando no podemos reproducirlo directamente a nivel local. En este caso, las herramientas de observabilidad pueden ser de gran ayuda. La realidad es que no hay manera de prever la complejidad de los problemas que van a surgir en nuestros entornos de producción. Debido a esa falta de previsibilidad, tenemos que recopilar los datos necesarios de producción a medida que se van ejecutando.

Gracias a una forma sencilla de mirar el flujo de solicitudes utilizando el rastreo distribuido, es fácil de recopilar rápidamente información sobre dónde ha fallado la solicitud. También podemos ver si el error surgió en la pila o se gestionó en algún punto intermedio, y si un usuario se vio afectado.

Los rastreos distribuidos también proporcionan información sobre los autores y los destinatarios de la llamada junto con sus respectivas cabeceras e información de temporización o reintento.

Esta información ayuda a dar un primer paso rápido en el análisis de la causa raíz y a conseguir potencialmente que los compañeros del área responsable ayuden en el problema. Todo ello puede lograrse sin señalar con el dedo al azar, sino con conclusiones basadas en datos.

Este método de análisis descendente no solo es mucho más rápido, sino que también ofrece la posibilidad de averiguar rápidamente la verdadera causa del problema. Obtenemos una visión general de todos los aspectos del problema e identificamos aquellos que necesitamos incluir en la investigación en base a las dependencias reales en sentido ascendente o en sentido descendente.

Para resumir, a la hora de averiguar el problema, la observabilidad suministra la información que necesitamos para actuar con rapidez y, en el mejor de los casos, con una mínima dependencia de otros equipos o socios externos como las empresas de alojamiento. Estos últimos pueden ser especialmente lenta en responder cuando se investiga un problema en el que el tiempo es crucial.

La implementación ha fallado
Históricamente, como desarrollador, podría saltarme esta sección. Las implementaciones no eran mi responsabilidad.

03 Rastreo distribuido

Hoy en día, sin embargo, los desarrolladores escriben configuraciones Dockerfile para compilar imágenes Docker y a menudo también proporcionan configuraciones de implementación en forma de Kubernetes (K8s), describiendo los contenedores, los servicios, etc., o al menos, parte de ellos, por una buena razón.

Proporcionar esos descriptores no es responsabilidad exclusiva de DevOps, ya que una implementación suele ser una combinación de descriptores de despliegue de ingeniería K8s y DevOps.

De todos modos, nuestra última implementación ha fallado, y no hay ningún problema obvio en el proceso de implantación en sí, pero el servicio se interrumpe justo después de la implementación. Ha llegado el momento de lucirnos.

El primer paso es averiguar qué sucede. ¿Fuimos realmente nosotros? A menudo, examinar un rastreo distribuido puede responder rápidamente a esa pregunta. ¿Es nuestro servicio el que da error, por ejemplo, HTTP estado 500 o bien el problema viene de más abajo en la pila? ¿La dependencia del servicio en sentido descendente devuelve el error? Tal vez estemos enviando los datos equivocados. De nuevo, una ojeada rápida puede responder a esa pregunta, también.

Las soluciones de observabilidad proporcionan las pruebas necesarias, notificadas originalmente por los sistemas y basadas en nuestro análisis de dominio de error, como observaciones precisas y contextuales. Los rastreos distribuidos son el principal testigo cuando se busca una manera de comprender rápidamente situaciones aún desconocidas. El tiempo medio para reparar y el tiempo medio para restablecer el servicio son las métricas clave aquí.

¿Recuerda la regla de "No implementar el viernes"? Yo también, y seguro que todo el mundo. Hoy en día, yo diría, que con una buena observabilidad y la posibilidad de ver inmediatamente pequeños cambios en el comportamiento, la latencia o el índice de errores, podemos y debemos implementar los viernes, aunque quizá a primera hora de la mañana. En este caso, nos quedan unas cuantas horas hasta conseguir arreglar un problema o revertirlo si sucede alguna cosa.

Si no podemos arreglar un problema de inmediato porque nos hemos metido en un agujero, nuestra única opción es retroceder a una compilación más antigua. Con un flujo de despliegue automatizado, este proceso es bastante fácil. Empecemos nuestra ejecución secuencial de integración continua y entrega continua (CI/CD) con una etiqueta diferente o una versión diferente y allá vamos. Acabamos de ganar mucho tiempo para investigar más a fondo, arreglar con más cuidado y eliminar la tensión adicional para ser lo más rápidos posible.

04
Ingeniería de alto rendimiento

Después de tanto hablar de acelerar la ingeniería y dedicar tiempo a lo que aporta más valor, es importante hablar de la propia ingeniería y cómo cambia o ya ha cambiado respecto a los años anteriores.

En la página 18 del informe Accelerate State of DevOps 2019 se plantean 4 cuestiones necesarias para un equipo de ingeniería de alto rendimiento en los próximos días. Apuntan al hecho de que los administradores de ingeniería, operaciones y base de datos ya no están en esos espacios totalmente separados.

Pero, ¿esto qué quiere decir exactamente? ¿Vamos a ser de operaciones? En absoluto. Nunca me consideraría un profesional de operaciones, aunque hago bastante ese trabajo a diario. La diferencia, sin embargo, sigue existiendo. No soy en absoluto un experto en ingeniería de despliegue, aunque otros sí que lo son. Y esa frase es válida para cualquier otro asunto, incluso en ingeniería. No todo el mundo es ingeniero de rendimiento. Sigue siendo necesario contar con esas personas altamente especializadas, pero conseguir la gran visión de conjunto importante para todo el mundo, más importante que nunca.

04 Ingeniería de alto rendimiento

Las 4 preguntas a las que queremos responder son:

- ¿Cuál es nuestra frecuencia de implementación?
- ¿Cuál es nuestro plazo de entrega para los cambios?
- ¿Cuál es nuestro tiempo para restaurar el servicio?
- ¿Cuál es nuestro porcentaje de errores?

Las respuestas a estas preguntas nos dicen lo rápido que podemos iterar. Nuestras respuestas ofrecen una buena orientación, pero seamos sinceros. Lo mismo ocurre con nuestro código base; solo podemos optimizar si comprendemos el problema.

¿Con qué frecuencia implementamos?

¿Implementamos una vez al año, una vez al mes, una vez al día o varias veces al día? Implementar más a menudo significa que necesitamos tener un mejor conocimiento acerca de cómo se comportan las nuevas versiones.

¿Cuánto tiempo tarda el código en entrar en funcionamiento?

¿Cuánto tiempo necesitamos realmente para una implementación? ¿Es una operación de envergadura, posiblemente una nueva versión entre todos los equipos? Tal vez podamos implementar de forma independiente. O podemos implementar automáticamente después de que pasen todas las pruebas.

¿En cuánto tiempo podemos recuperarnos de un corte del servicio?

Si algo va mal, ¿quién puede analizar el problema? Mi pregunta favorita es: "¿Quién puede resolver el problema en mi servicio si estoy de vacaciones, sin teléfono ni conexión a Internet, y cuánto tardará la persona en comprender el problema desde el principio?" ¿Cuántas implementaciones fallan?

Por último, pero no por ello menos importante, la pregunta más habitual, la que probablemente la mayoría de las personas puedan responder sin pensárselo demasiado: ¿fracasan las implementaciones?; en caso afirmativo, ¿cuántas y por qué?

Esta lista es un excelente punto de partida para buscar y entender la velocidad de las iteraciones en nuestra compañía, nuestro equipo y servicio o aplicación.

Pero, yo iría un paso más allá, y diría que falta una pregunta importante: una pregunta sobre las diferencias entre versiones. Es genial tener un número bajo de despliegues anómalos, y a lo mejor, podemos implementar varias veces al día. Sin embargo, aún queda una pregunta por responder: ¿Cuál es la diferencia de errores y rendimiento entre versiones?

Los mejores despliegues y la iteración más rápida no valen nada si cada versión funciona peor que la anterior o aumenta la frecuencia de errores. Al final, no queremos volver a apagar fuegos, ¿verdad?

El desarrollo está cambiando

Es evidente que la forma de desarrollar aplicaciones está cambiando. En los últimos años ha surgido la necesidad de ampliar los sistemas más que nunca. Dejando a un lado las soluciones de Internet de las cosas (IoT), el número de usuarios, clientes o personas que queremos ver felizmente comprometidos con lo que hemos construido es cada vez mayor.

Muchas empresas están todavía en fase de adopción, pero es casi seguro que los nuevos sistemas ya no se construyan de forma monolítica. La escalabilidad se ha vuelto demasiado importante, y el escalado verticalmente es demasiado caro y, en algunos casos, inalcanzable.

Los microservicios, aunque no son la fórmula milagrosa que muchos proclaman, están definitivamente aquí para quedarse. Trocear el trabajo en pequeñas partes y desplegarlas de forma independiente suena muy bien y es increíble para la escalabilidad. La sensación de poder escalar partes del sistema de forma independiente según las necesidades es increíble. Pero en el lado negativo, esta escalabilidad tiene un coste al añadir muchas operaciones de red entre servicios. Las operaciones pueden agotarse, fallar o devolver desperdicios. Estamos luchando contra un nuevo tipo de enemigo.

04
Ingeniería de alto rendimiento

Además, recuerdo los años en que los datos se almacenaban en bases de datos relacionales. No se hacían preguntas; no habían alternativas. Aunque las bases de datos relacionales están aquí para quedarse, hoy en día tenemos muchos sistemas diferentes entre los que elegir, desde simples almacenes de valores clave o de documentos, pasando por bases de datos gráficas o de series temporales, hasta los almacenes de columnas para agregaciones extremadamente rápidas.

Todos esos sistemas son diferentes, y todos se comportan de forma diferente. No hay nadie que sea un experto en todos ellos. Sin embargo, lo maravilloso como desarrollador es que tenemos la posibilidad de elegir la mejor herramienta para trabajar al tiempo que aprendemos cosas nuevas.

Por último, pero no menos importante, hoy en día implementamos sistemas y servicios de forma diferente que hace unos años. La implementación de sistemas dedicados se ha convertido en minoritaria en las nuevas instalaciones. Ahora, casi todo el mundo despliega en máquinas virtuales. Muchos ya despliegan en la nube o en entornos autoalojados con Kubernetes o Cloud Foundry. Docker y otros tiempos de ejecución de contenedor como CRI-O son habituales en las máquinas de los desarrolladores

Todas esas tecnologías son maravillosas y acercan el desarrollo en mi máquina al entorno en el que finalmente se van a ejecutar. Nunca me gustó la sentencia "Funciona en mi máquina", aunque fuera cierta. La razón por la que no me gustaba es simple. Significaba que pasaba algo, algo que no podía comprender o explicar de modo inmediato.

Ejecuciones secuenciales de desarrollo

Si queremos iterar más rápido, necesitamos tener instalado un buen soporte. Aunque afortunadamente las pruebas unitarias no sean nada nuevo y las pruebas de integración básica se empleen habitualmente, todo lo que viene después se sigue despreciando por ser demasiado complejo, demasiado caro o que no merece la pena para un desarrollador.

Sin embargo, con ciclos de iteración cortos y sistemas distribuidos, las pruebas de integración se han vuelto más importantes que nunca. La integración continua en un sistema más cerca de la transferencia y la producción es imprescindible para lograr ciclos de retroalimentación rápidos durante el desarrollo. También son un buen momento para recopilar información sobre el primer rendimiento y sobre el porcentaje de errores. Descubrir los posibles problemas en una fase temprana del desarrollo de un dispositivo puede impedir largos ciclos de recompilación y aportar una rápida validación de las expectativas, o demostrar que son erróneas.

Otra pieza del rompecabezas es que es mejor automatizar las pruebas de carga periódicas siempre que sea posible. Las razones son las mismas que antes: bucles de retroalimentación cortos y una validación temprana de modelos, expectativas y cuestiones relacionadas con el rendimiento.

El último paso en la optimización del equipo de ingeniería de alto rendimiento es la entrega continua, que requiere que todos los elementos mencionados anteriormente estén en su sitio. También requiere el valor de fracasar. Sabiendo que la comprensión de los errores, el rápido análisis de la causa raíz y la implementación de la corrección de errores no son posibles, pero cuentan con el soporte de los miembros de todo el equipo y de las herramientas durante todo el proceso de producción, la valentía es mucho mayor, automáticamente. Nos sentimos mucho más seguros de cara al futuro.

Dicho esto, hay que reconocer rápidamente las situaciones de error. Las herramientas de observabilidad enriquecidas por el contexto tienen que dar soporte al proceso de análisis de problemas y ayudar a encontrar la causa raíz en el menor tiempo posible. Además, todos los servicios deben ser lo más resistentes a dependencias erróneas que sea posible.

Una vez producido y confirmado el arreglo, la ejecución secuencial se pone en marcha y compila, prueba y despliega la nueva versión en producción. Los servicios pequeños e independientes son más fáciles de gestionar con un proceso de este tipo que los grandes monolitos.

Por último, solo para subrayarlo de nuevo, todos los pasos deben estar totalmente automatizados para minimizar el impacto de una implementación anómala o una versión que no funciona.

04
Ingeniería de alto rendimiento

Y estamos en marcha
Justo después de que una implementación entre en funcionamiento, la pregunta más importante más allá de "¿Funciona?" era "¿Qué tal funciona?"

Como he mencionado antes, un ciclo de inversión rápido solo tiene sentido cuando las nuevas versiones no funcionan peor por lo general que las antiguas. Existen excepciones a esta regla, pero solo debería tratarse de un impacto inicial en el rendimiento conocido y previsto, algo ya calculado en la posible escala de la infraestructura.

Las métricas clave importantes que hay que supervisar inmediatamente después de la implementación son cualquier tipo de cambios imprevistos en la latencia de promedio, el porcentaje de errores o las llamadas en sentido descendente, por ejemplo, el número de llamadas a la base de datos o similares. Las buenas herramientas de observabilidad ofrecen comparaciones directas antes y después del despliegue para una fácil accesibilidad.

De todos modos, nuestra implementación fue bien, los números inmediatos son buenos y el sistema se comporta dentro de los rangos previstos. ¿Es este el final de la historia? ¿Nuestra implementación está feliz para siempre?

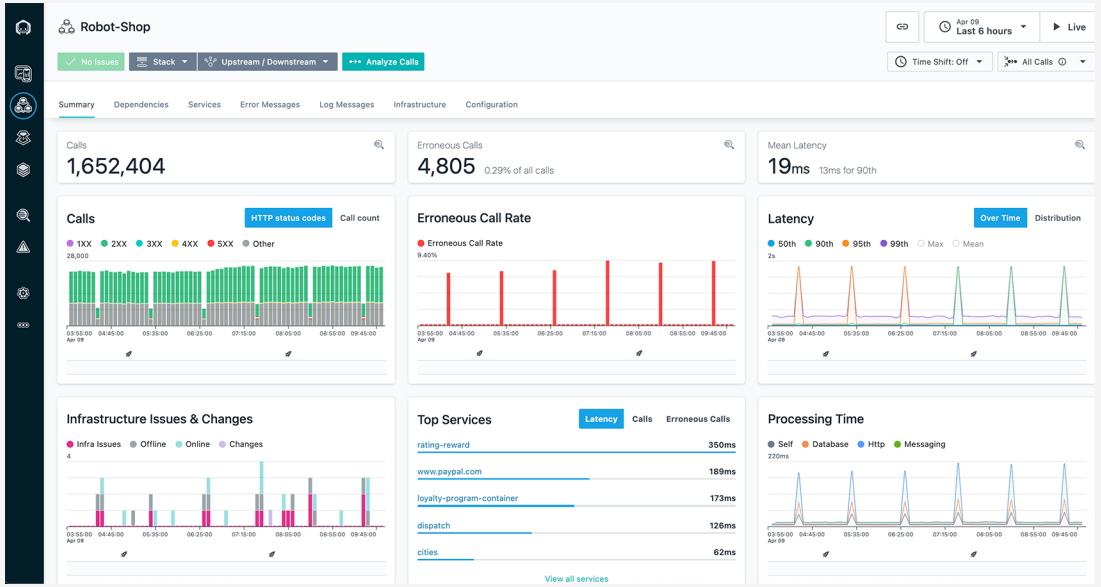


Figura 5. Marcadores de release en el panel de control para tener una visión inmediata después y entre releases, fuente: Instana

04
Ingeniería de alto rendimiento

Desafortunadamente, sabemos que este resultado no suele ser el caso en los entornos de producción. Algo está a punto de suceder todo el tiempo. Preparémonos. Aunque los tiempos de apagar fuegos se han reducido en un orden de magnitud, nunca desaparecerán del todo. Recuerda, construir un sistema infalible es simplemente imposible.

Es cierto que en esta fase del proceso, hay menos participación directa de la ingeniería cuando algo va mal. Para recalcarlo de nuevo, cuando estamos de guardia y se nos pide que intervengamos, una visión general inteligente de todos los componentes, redes, máquinas, aplicaciones que participan y su interconexión nos ayuda a profundizar rápidamente. Es una tarea que se complica cada vez que añadimos un poco de abstracción o complejidad arquitectónica al sistema. Lo que nos simplifica y facilita la vida durante el desarrollo, aumenta en complejidad cuando se analizan parcialmente sistemas desconocidos.

Sólo hay una medida sobre la que debemos juzgar nosotros mismos, especialmente para las situaciones de guardia: —Tiempo medio para volver a la cama (MTTGBTB, por sus siglas en inglés). Gracias a Karthik Kumar por esta medida tan importante.

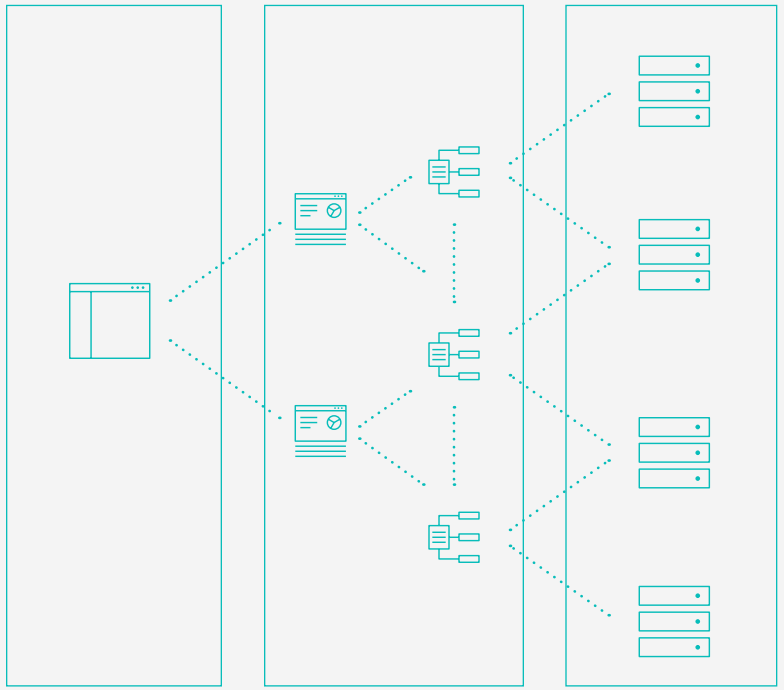


Figura 6. La complejidad arquitectónica impide a veces un razonamiento rápido

05 Desarrollo basado en la observabilidad

Ahora que comprendemos la importancia de la observabilidad, veamos el valor del desarrollo basado en la observabilidad (ODD, por sus siglas en inglés). Resumamos rápidamente los elementos más importantes que se describen como ODD.

El ODD es una extensión directa del desarrollo basado en el comportamiento. Amplía las pruebas para incluir el comportamiento de un componente con expectativas en torno al rendimiento y el estado. Las pruebas deben realizarse temprano y ejecutarse a lo largo del ciclo de desarrollo. Estas pruebas ofrecen una visión del rendimiento antes, durante y después del desarrollo de las funciones. Ayudan a comparar las expectativas iniciales con la realidad y suministran información si el sistema de producción requiere escalarse antes del despliegue final de características.

Cree el ciclo de desarrollo en torno a la idea de los bucles cortos de retroalimentación. Lo fundamental es hacer que la observabilidad sea una parte central del proceso de desarrollo. Hacer que sea un pensamiento proactivo y trabajar junto con otros equipos. No construir silos.

Durante esas interacciones, elabore métricas de estado y comience a implementarlas en las primeras etapas del proceso de implementación de la función. Recuerda que debes tener presente el número de métricas. La selección óptima es entre 3 y 5 métricas, como máximo 10.

Si la solución de observabilidad requiere una integración manual mediante envoltorios, añádalos también desde el inicio. Si podemos tener nuestro servicio instrumentado automáticamente, mucho mejor. Nuestro código, nuestro "arte" se mantiene intacto.

Para las optimizaciones, mejor no hacer suposiciones. Para seleccionar los puntos de optimización, hay que tener en cuenta todos los datos que proporciona la solución de observabilidad, las métricas, los rastreos distribuidos y la información de la infraestructura. Encuentre los factores que más contribuyen a la latencia y a los porcentajes de errores y arréglelos primero. El fruto al alcance de todos da victorias rápidas y motivación.

Además, eche una ojeada a los datos de creación de perfiles de código. Un perfilador de producción siempre activo aumentará nuestra comprensión a un nivel que antes era casi imposible de recopilar. Hasta hace poco, nunca tuve la oportunidad de obtener un perfil en mi sistema que se acercara a lo que veía en producción. Pero recuerde, no utilice cualquier perfilador. Los perfiladores listos para la producción están hechos a medida para su respectivo caso práctico, con una sobrecarga extremadamente baja.

La última sugerencia probablemente le ofenderá con solo leerla. Pruebe ... espere ... en producción. Sí, lo ha leído bien. Pruebe en producción, pero con distintivos por funciones. Habilite el nuevo comportamiento, la nueva funcionalidad, basándose en un conjunto de ID de usuario, parámetros especiales especificados y sea creativo.

El razonamiento que subyace a ese paso, y el rechazo total de la antigua regla de nunca probar en producción, es la singularidad de cada entorno. El estado de la infraestructura, las dependencias, la fecha y la hora, el despliegue, el entorno en sí, y la fase lunar... todo puede afectar a nuestro código.

Después de todo lo debatido en torno el desarrollo y por qué la observabilidad facilita la vida del desarrollador, ¿cómo podemos aplicarla? En primer lugar, pregunte al equipo de DevOps si ya tienen una solución utilizada por otro equipo.

06
Ser o no ser... código abierto

Como con casi todo, la comunidad de código abierto ha hecho un gran trabajo suministrando al mundo varias opciones. Las opciones son buenas, ¿no es así?

Sí y no. El principal obstáculo que hay en la mayoría de las soluciones es que o bien llevan a cabo la supervisión de métricas o el rastreo distribuido, pero no ambos. El problema básico es que perdemos la correlación entre la información. Las herramientas de software de código abierto (OSS) tienen paneles de control, silos de datos independientes y, por lo general, no tienen una forma de saltar entre métricas, rastreos e infraestructura para una única solicitud.

Creación manual de la observabilidad

Para crear la observabilidad, es habitual utilizar dos herramientas separadas, tal como hemos mencionado, una para la parte de métrica y supervisión, a menudo Prometheus, y otra para el rastreo distribuido, a menudo Jaeger o Zipkin.

Estas soluciones de código abierto están disponibles "gratis" — explicaré la razón de las comillas— y se pueden integrar con un servicio de aplicaciones. La integración en este caso significa que tenemos que añadir el rastreador y el recopilador de métricas a nuestro código fuente. Existen envoltorios para las librerías más comunes disponibles en muchos lenguajes de programación y se pueden utilizar directamente.

Un ejemplo sencillo para una métrica de Prometheus, midiendo el número de solicitudes por segundo, tendría un aspecto similar al siguiente fragmento de código, utilizando una instancia de contador para recuento del número de solicitudes. Véase la Figura 7.

No está mal para el uso directo. Afortunadamente, muchos marcos de trabajo proporcionan ese tipo de métricas utilizando Prometheus al momento.

La otra parte es utilizar Prometheus para medir el uso de recursos como las conexiones de la base de datos o consultar los tiempos de ejecución. Aquí, tenemos un montón de integraciones de la comunidad Prometheus. En el mundo Java, la famosa solución de correlación relacional de objetos (ORM) de Hibernate se puede utilizar con tan solo un poco de código. Véase la Figura 8.

La integración de Prometheus se encargará automáticamente de todos los detalles sucios de la implementación.

En el lado del rastreo distribuido, tenemos a Zipkin o Jaeger. Con ambas soluciones, podemos añadir rastreo distribuido a nuestra aplicación y, de nuevo, la integración no es demasiado complicada, como muestra el siguiente fragmento de código, basado en nuestro FooHandler existente. Véase la figura 9.

```
public class FooHandler {
  Counter counter = Counter
      .build()
      .namespace("my-app")
      .name("foo-handler")
      .help("number of requests")
      .register();

public ResponseEntity handler(Request request) {
  counter.inc(1);
  // do some business thing here
  }
}
```

Figura 7. Ejemplo de una medida de Prometheus

```
new HibernateStatisticsCollector()
   .add(sessionFactory, "my-app")
   .register();
```

Figura 8. Ejemplo de Hibernate con Prometheus

06
Ser o no ser... código abierto

Vemos que aún no es realmente complicado, pero de alguna manera tenemos la sensación de que hemos vuelto a "los viejos tiempos". Lo peor es que hemos abarrotado probablemente nuestro código con más formalismos de supervisión y rastreo que con verdadera lógica empresarial.

Dar sentido a los datos

Junto con los datos de registro almacenados en algo parecido a Logstash o Splunk, ahora hay una gran cantidad de datos en los que escarbar cuando ocurre un problema. Sin embargo, existe otra problema: la propia observabilidad de la pila.

Con 3 sistemas independientes que no comparten ningún dato o ninguna correlación de datos, dar sentido a las diferentes fuentes de datos es complicado y requiere mucho tiempo. El usuario es quien tiene que hacer coincidir las indicaciones de fecha y hora, establecer la conexión y finalmente, construir el contexto para resolver el enigma en cuestión.

Tenga presente que darle sentido a todo es el el principal problema de las herramientas de observabilidad de código abierto. Ahora tenemos más sistemas individuales y más datos sin el necesario contexto agregado propio.

```
public class FooHandler {
  Counter counter = Counter
        .build()
        .namespace("my-app")
        .name("foo-handler")
        .help("number of requests")
        .register();

private JaegerTracer tracer;

public FooHandler() {
  Configuration.SamplerConfiguration samplerConfig =
   Configuration.SamplerConfiguration
    .fromEnv()
    .withType("const")
    .withParam(1);

  Configuration.ReporterConfiguration reporterConfig =
   Configuration.ReporterConfiguration
    .fromEnv()
    .withLogSpans(true);

  Configuration config = new Configuration()
   .withSampler(samplerConfig)
   .withReporter(reporterConfig);

   this.tracer = config.getTracer();
  }

public ResponseEntity handler(Request request) {
   Span span = tracer.buildSpan("foo handler").start()
   try {
    counter.inc(1);
     // do some business thing here
   } catch (Exception e) {
     span.setTag("http.status_code", 500);
   } finally {
     span.finish();
   }
  }
}
```

Figura 9. Fragmento de código de rastreo Java con Jaeger

06

Ser o no ser... código abierto

Coste real del código abierto
¿Recuerda cuando mencioné antes lo de "gratis" y dije que hablaríamos de ello? Ha llegado el momento.

Al contrario de lo que suele creerse, el código abierto no es gratis. Es cierto que no hay ningún coste de licencia para poderlo utilizar. El coste principal de observabilidad del software de código abierto (OSS) se mide en tiempo, tiempo en el que las empresas pagan para que las personas entiendan cómo hacer funcionar la pila de observabilidad, cómo utilizarla y cómo dotar de sentido a los datos. Este tiempo, a menudo lo que menos tenemos, se pierde en entregar la funcionalidad de la empresa.

También puede que sea necesario pagar a más ingenieros para ajustar la observabilidad del software de código abierto (OSS). Tampoco olvidemos el tiempo necesario en integrar la supervisión y el rastreo distribuido. Estas tareas suponen todas ellas un coste adicional además del tiempo que se tarda en implementar la observabilidad OSS en la base del código fuente.

Mantener esas integraciones en consonancia con nuestros objetivos previamente establecidos puede añadir fácilmente hasta un 5%—10% del tiempo de desarrollo que tratamos realmente de evitar cuando dejamos la época de Zabbix y Nagios.

Si examinamos los equipos de DevOps y de operaciones, este número solo aumenta porque ahora estamos operando varios sistemas, y esto es otra historia. Sólo miramos nuestra historia de desarrollo.

Por último, pero no por ello menos importante, también existe un coste de funcionamiento de la pila, el espacio de almacenamiento necesario para guardar todos los datos, y el tiempo de cálculo por intentar reimplementar incluso las correlaciones automáticas más básicas.

De repente, este conjunto de herramientas de código abierto "gratis" se ha vuelto muy caro y sigue presentando carencias de observabilidad y contexto.

Solución del proveedor
Las soluciones comerciales, en cambio, ofrecen muchas, si no todas, las prestaciones anteriores en una única solución. Si determinadas partes no se proporcionan directamente, integran servicios externos hasta un punto en que parece que esté almacenado internamente. La correlación de datos de los sistemas internos y externos está incluida.

Pero, la correlación de datos entre todas las partes de nuestros sistemas es el elemento más importante. La infraestructura, aplicaciones, servicios, red, registro... todos los datos están previamente procesados para que la información necesaria sea fácil de recopilar y tenga sentido.

Algunos proveedores dan un paso más y correlacionan los problemas encontrados en diferentes partes del sistema para crear alertas más viables y específicas. Correlacionar todos los problemas que van juntos ofrece una visión aún más rápida del alcance y la causa de una incidencia, los servicios relacionados y si hay impacto para el usuario.

Además, algunos proveedores proporcionan una información inmediata de los cambios antes y después de los releases, tal como se ve en la Figura 5. Para lograr esta información, es necesaria la integración con las ejecuciones secuenciales CI/CD, servicios DevOps y herramientas de desarrollo habituales.

06

Ser o no ser... código abierto

Más allá del código abierto
Sin embargo, como desarrollador, lo que buscamos es trabajo de desarrollo. En lugar de configurarlo todo manualmente, algunas soluciones comerciales ofrecen una instrumentación de código totalmente automática.

¿Recuerda nuestro código de ejemplo de antes y se pregunta qué aspecto tendría al utilizar la instrumentación de código totalmente automatizada?

Así es, no son necesarios cambios en el código para añadir supervisión, rastreo distribuido o recopilación de rendimiento. La automatización recopila los servicios, independientemente de si se inician directamente, en un contenedor Docker o dentro de Kubernetes, Red Hat® OpenShift® Platform u otros entornos similares. Después de descubrirlo automáticamente, el servicio se instrumenta sobre la marcha, se generan paneles de control basados en los procedimientos recomendados conocidos para el servicio o el entorno especificados y ya estamos listos. No es necesaria ninguna intervención manual. ¿Qué más podemos pedir?

Ventaja de la correlación automática
Como he mencionado antes, la principal ventaja de una solución de observabilidad de la empresa es la posibilidad de correlacionar las métricas y los rastreos de máquina, infraestructura y de aplicación y servicios. Los rastreos distribuidos permiten comprender el flujo de la solicitud, mientras que las métricas proporcionan los puntos de rendimiento necesarios.

Sin embargo, correlacionar manualmente es bastante engorroso. La principal razón por la que a las personas no les gusta tener múltiples paneles de control para diferentes servicios es que es casi imposible que coincidan los intervalos de tiempo y recopilar el contexto general del problema.

La mayor ventaja de la correlación automática, como se ha descrito anteriormente, es la visión inmediata. Al examinar un problema o un incidente, la solución del proveedor ya ha realizado todo el trabajo de investigación para proporcionar las piezas de información importantes como prueba contextual y llevarnos justo al lugar interesante.

```
public class FooHandler {
   public ResponseEntity handler(Request
request) {
      // do some business thing here
   }
}
```

Figura 10. Código con instrumentación totalmente automatizada con Instana AutoTrace

07

Simplificar nuestras vidas

Coste de las soluciones de proveedores
A diferencia de las herramientas de código abierto, las soluciones de los proveedores no pretenden ser gratis.

Con el código abierto, tenemos que operar el sistema y pagar por el almacenamiento de datos y la potencia computacional. Por el contrario, las soluciones de los proveedores ofrecen todas las herramientas necesarias para suministrar una completa observabilidad de la empresa como un entorno de software como servicio (SaaS) alojado o en las instalaciones en los casos en que sea un requisito.

Con el coste eliminado de la integración de la supervisión y el rastreo en los servicios y la infraestructura, las soluciones de los proveedores, al final, suelen ser más rentables, con menos costes operativos y de desarrollo. Y no olvidemos el coste de almacenamiento necesario para almacenar toda la información y los datos.

Tomar una decisión con conocimiento de causa
La elección entre una solución de código abierto o una solución comercial es una cuestión de cuánto tiempo queremos dedicar al desarrollo de funcionalidades no comerciales; no es un tema de costes, ya que ninguna de las dos soluciones es gratuita.

Hasta ahora, nos hemos centrado en los desarrolladores que no desean añadir supervisión y rastreo a su código, y sin duda, es el grupo más grande. Sin embargo, hay desarrolladores a los que les gusta jugar. Yo, por mi parte, me considero alguien que lleva el juego en el ADN. Aun así, prefiero jugar con las nuevas tecnologías, no con una medida de rendimiento.

Aparte de eso, mi experiencia me dice que olvidamos las recopilaciones de datos más importantes cuando los hacemos manualmente, y no solo una vez. O bien caemos de nuevo en la vieja costumbre de añadir todo lo que es posible en lugar de centrarnos en nuestros dominios de error definidos.

Las soluciones de los proveedores proporcionan un conjunto cada vez mayor de procedimientos recomendados para recopilar automáticamente las métricas y comprender los lenguajes de programación, los marcos de trabajo y las integraciones de servicios. Estos procedimientos recomendados eliminan la mayor parte del trabajo a la hora de definir los posibles dominios de error para sistemas nuevos o desconocidos. Con las soluciones de los proveedores, es raro que volvamos a encontrarnos que falta de nuevo información importante.

07

Simplificar nuestras vidas

La creación de software hoy en día difiere en gran medida de los métodos tradicionales que hemos empleado durante las décadas anteriores. Las aplicaciones se dividen en pequeños microservicios, mientras que las implementaciones se construyen sobre Docker, Kubernetes y ejecuciones secuenciales automáticas CI/CD.

Estar al tanto de todos esos cambios durante las operaciones y analizar los problemas es cada vez más complicado. Es importante contar con nuevas formas de obtener información sobre que está ocurriendo, especialmente cuando se está de guardia y se intenta comprender un problema en un componente desconocido.

La observabilidad de la empresa es clave. Elija la lucha correcta; no luche contra la solución de observabilidad.

IBM España, S.A
Tel.: +34-91-397-6611
Santa Hortensia, 26-28
28002 Madrid
Spain

Producido en los Estados Unidos de América
Abril de 2021