Lo que los desarrolladores deben saber sobre la observabilidad

por Chris Engelbert, Experto de Instana Developer

INSTANA
an IBM Company

Empiece aquí

Contenido

01

Introducción

02

Terminología de la observabilidad

03

Rastreo distribuido

04

Ingeniería de alto rendimiento

05

Desarrollo basado en la observabilidad

06

Ser o no ser... código abierto

07

Simplificar nuestra vidas

01 Introducción

Como desarrolladores, podemos recordar la época en que Nagios era una tecnología de vanguardia. Desde entonces, el término "supervisión" ha tenido algunas malas implicaciones asociadas a TI. Mirar los tableros cargados de diferentes gráficos y números no nos ayudó a resolver los problemas, y sin embargo se nos pidió que los miráramos. Efectivamente, esos tableros nos decían que algo andaba mal, pero nuestro trabajo no estaba ejecutando el sistema, ¿no es cierto? Acabamos con la fatiga de las métricas después de un corto período de tiempo.

¿Por qué Nagios no funcionó mejor para nosotros los desarrolladores? Aparte de las métricas de rendimiento recogidas directamente de los contadores incorporados en el flujo de código de las aplicaciones, todo era una supervisión de caja negra, que consistía sobre todo en pings a hosts y conexiones a servicios, a menudo acompañados de algún análisis de expresiones regulares (regex) de registros.

Por si fuera poco, en los viejos tiempos de las operaciones de servicio, el mundo del desarrollo y el de las operaciones estaban estrictamente separados. Teníamos administradores de bases de datos, lo que impedía a todos los demás ejecutar incluso una declaración selectiva por su cuenta.

El equipo de operaciones -que maneja el sistema en producción, se ocupa de los fallos del sistema y transmite la información sobre los problemas a ingeniería- hizo una carrera, a partir de mantener a los desarrolladores alejados de los sistemas de producción. En el mejor de los casos, solo permiten el acceso a un número muy limitado de desarrolladores.

Por último, estaba el departamento de ingeniería. Fue nuestro trabajo construir funcionalidades y probarlas, y eso fue lo que hicimos. La siguiente vez que nos involucramos fue en la corrección de errores. Para nosotros, el código de origen era un arte en sí mismo. Por supuesto, también mantuvimos alejados de nuestras cosas a los que no eran desarrolladores.

Operar y supervisar el sistema simplemente no era parte de nuestro trabajo. Salvo una parte específica, añadir supervisión a nuestro hermoso código de origen, era algo parecido a poner revestimiento de aluminio en la Capilla Sixtina. Lo odiamos por todas las razones válidas. Se extendió como un reguero de pólvora.

Las responsabilidades se dividieron y durante mucho tiempo, los ingenieros de software escribieron código y se fueron a casa. A partir de ahí, el departamento de operaciones se encargaba de todo lo que salía mal.

Sin embargo, los tiempos han cambiado. Hoy en día, los sistemas tienen un aspecto diferente, las implementaciones funcionan de forma distinta y las fronteras entre los equipos se difuminan, si no desaparecen por completo. Dicho esto, como desarrolladores, estamos más cerca de operaciones que nunca y tenemos un papel indirecto en la explotación de los sistemas.

Con la complejidad de los sistemas modernos que ejecutan microservicios, la "supervisión significativa" se convierte en un papel importante de nuestras vidas. Para evitar la sobrecarga de métricas del pasado, tenemos que estudiar las ventajas de la observabilidad.

Este libro electrónico es un examen del nuevo mundo. Dejaremos atrás todos los malos sentimientos acerca de la supervisión y daremos nuestros primeros pasos en el mundo de la observabilidad y su importancia cada vez mayor para los desarrolladores.

02 Terminología de la observabilidad

En primer lugar, vamos a dejar a un lado los fundamentos de la terminología de la observabilidad. Para conocer más en el mundo de la observabilidad, es importante entender las diferencias entre supervisión y observabilidad junto con la forma de representar los datos al usuario.

Supervisión
Tradicionalmente, la supervisión se ha centrado en las métricas de las series temporales. El proceso era siempre el mismo: recopilar un montón de métricas, poner esas métricas en gráficos en paneles de control, averiguar para qué métricas establecer alertas y elegir algunos límites para las alertas. Este abordaje, aunque es mejor que nada, dista mucho de ser ideal, ya que da lugar a demasiadas o pocas alertas, a una falsa sensación de seguridad y a un tiempo considerable dedicado al proceso de resolución de problemas.

Incluso si ampliamos la supervisión para incluir la salud y el rendimiento de los servicios, la supervisión tradicional adopta un abordaje de alerta basado en los 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, hay que centrarse en la recopilación de datos relevantes que tengan un contexto integrado.

Observabilidad
Aunque la observabilidad no es nada nuevo, es el complemento perfecto para la supervisión. Puede considerarse como un superconjunto.

El término observabilidad implica que observamos algo. Lo importante aquí es observar a un nivel más granular que la supervisión. Sin dejar de incluir todos los números y gráficos de la supervisión, la observabilidad añade el conocimiento de lo que es significativo supervisar a todos los equipos diferentes, antes 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 recolectado los datos necesarios para ayudar a resolver un problema desconcertante.

Además de esta cuestión, la observabilidad añade un rastreo distribuido, básicamente un rastreo del paquete de microservicios. Discutiremos lo que significa. Y no lo olvide: un registro analítico significativo tiene un alcance mucho mayor que la simple búsqueda de errores basada en una expresión regular (regex).

02 Terminología de la observabilidad

Los dos últimos elementos están concebidos para llevar a cabo la supervisión desde el interior de las aplicaciones y los servicios. A diferencia de la simple supervisión, recolectamos información directamente del interior del servicio y la unimos a todo lo que sabemos sobre el sistema.

La observabilidad está diseñada con el conocimiento de los dominios de fallo identificados de nuestro sistema, por ejemplo, un servicio que se conecta a otro servicio a través de HTTP puede fallar en la conexión por varias razones, eso es un dominio de fallo identificado. Los dominios de fallo pueden depender de la forma en que la gente piensa en el servicio o sistema en su conjunto. Por esta razón, es importante tener en cuenta diferentes perspectivas a la hora de diseñar los dominios de fallo y el contexto asociado, por ejemplo, para facilitar la depuración, dependiendo de la audiencia.

Al examinar la observabilidad, vemos tres pilares para aportar conocimiento y comprensión a nuestro problema: las métricas de salud y rendimiento, las huellas distribuidas y los registros.

Observabilidad empresarial
En la empresa, la observabilidad debe ser altamente escalable para manejar las complejidades y la escalabilidad de los sistemas transitorios que ejecutan microservicios de corta duración o aplicaciones sin servidor. Los sistemas diseñados para la observabilidad de la empresa se mantienen automáticamente al día con los cambios constantes de la infraestructura o el panorama de los servicios. Parte de este proceso consiste en descubrir instancias o servicios nuevos o apagados sin intervención manual. Normalmente proporcionan bibliotecas envolventes para realizar cambios mínimos y no intrusivos en el código de origen, a veces incluso llegando a la instrumentación de código totalmente automática para añadir mediciones y sondeos de estado en una aplicación ya en funcionamiento.

La instrumentación del código proporciona una forma de añadir los puntos de inicio y finalización de las operaciones necesarios en la base de código en ejecución sin añadirlos manualmente en el código de origen. Dicho esto, los desarrolladores se ven liberados en su mayoría del tedioso trabajo de añadir elementos de supervisión y rastreo a su código base, lo que les deja más tiempo para trabajar realmente en los casos de uso de negocios.

Además, la elaboración de perfiles de código de los sistemas de desarrollo, de ensayo y de producción, siempre activa pero con poca sobrecarga, permite conocer mejor el rendimiento de los servicios en condiciones reales, algo que a los desarrolladores les faltó 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 mostrarlo a la antigua usanza con números, tablas y diagramas. El panorama general es que el contexto de los datos es lo importante para resolver los 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, que facilite la comprensión y genere información procesable. También suelen ayudar en el análisis de la causa raíz al proporcionar el contexto necesario en torno a los eventos e información correlacionados.

03 Rastreo distribuido

Las huellas distribuidas son para los sistemas distribuidos lo que las huellas del paquete son para las aplicaciones y las excepciones o pánicos. Es una técnica para capturar y cronometrar los gestores de servicios y las llamadas internas mientras una solicitud pasa al escenario de sistemas para generar su respuesta. Por esa razón, a veces también se denomina rastreo de solicitudes distribuidas.

El rastreo distribuido ayuda a los desarrolladores a analizar los flujos de solicitudes y a identificar la causa raíz de los problemas o los cuellos de botella en el rendimiento.

Imagínese que un usuario llama a un servicio de atención al usuario solicitando los datos de la cuenta propia. El propio servicio llama a varios servicios en el paquete para obtener diferentes tipos de información. La Figura 1 muestra el flujo básico de la llamada. Aunque la información, como quién llama a quién, es visible, en el diagrama faltan datos importantes como la información sobre el tiempo.

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 tiempo y como una vista de árbol del paquete.

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

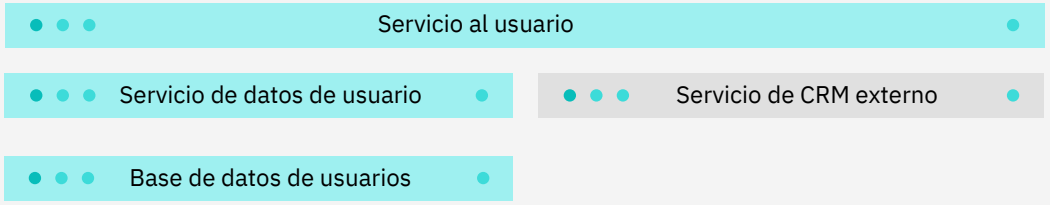


Figura 2. El flujo de llamadas como diagrama de tiempos y jerarquías

Figura 3. Diagrama de jerarquía del flujo de código

03 Rastreo distribuido

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

Para representar esa información, la huella se desglosa en elementos, como períodos y llamadas.

Un período, a veces llamado período de tiempo, representa el tiempo de ejecución de una sola operación dentro del flujo de la huella distribuida. Una operación en este contexto es una ejecución de código con una hora de inicio y una hora de finalización, así como posibles tramos padre e hijo.

Además, un período contiene metadatos que proporcionan un contexto en torno al propio período. Para conectar varios períodos, se utilizan identificadores (span-id y trace-id) para construir las jerarquías padre/hijo.

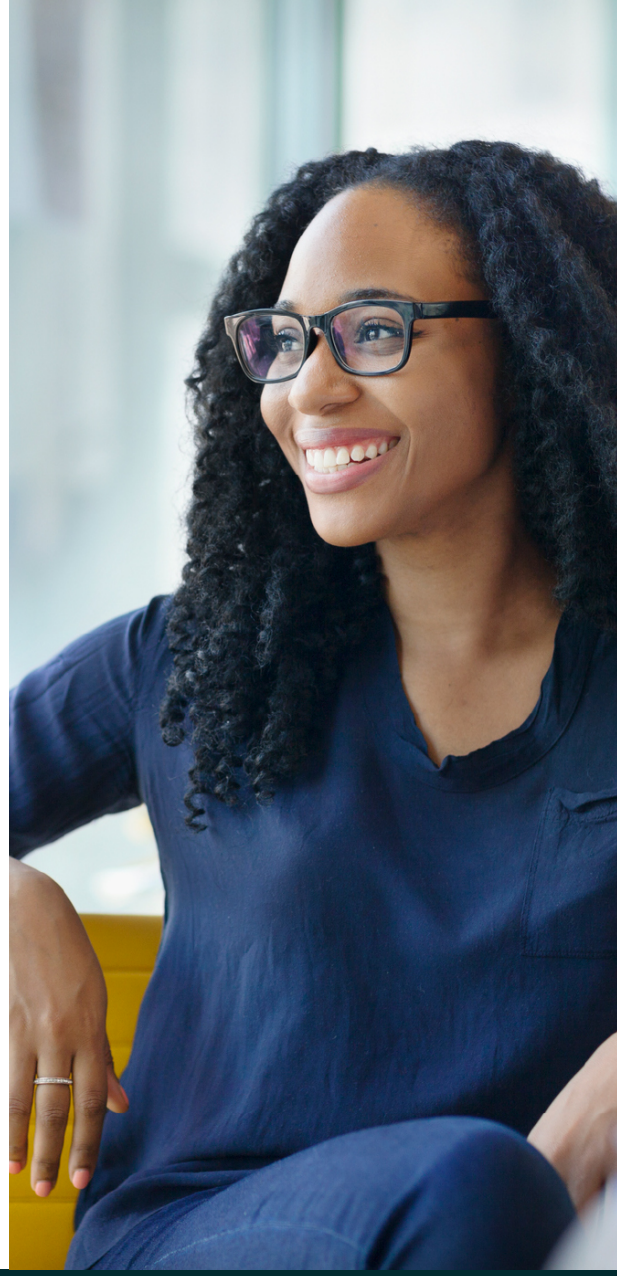
La comunicación entre dos períodos se representa mediante llamadas. Las llamadas contienen información sobre qué tipo de conexión se ha utilizado, información de cabecera y datos de respuesta, como los códigos de estado HTTP si una llamada fue errónea. También se puede añadir información contextual personalizada.

La importancia de la observabilidad
Cuando se trabaja con sistemas distribuidos, los retos son muy diferentes a los de las aplicaciones monolíticas que construíamos 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 sub-llamadas a otros servicios o bases de datos. La comprensión de 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 métricos en los tableros porque regresamos al estado en el que estábamos hace años. Lo que buscamos es una información relevante que nos lleve a la causa de un problema lo antes posible. Necesitamos que el sistema nos diga qué métricas son importantes para el dominio del fallo de cada servicio específico. Es importante centrar nuestra atención solo en lo que importa.

Y ese enfoque tiene sentido porque no hay garantía de que acabemos con un sistema sin fallos, sin importar cuántas métricas adicionemos. Es más bien al revés; los fallos en nuestro sistema no se pueden evitar. . Escribí sobre este tema en una publicación de blog titulada Construcción de aplicaciones resilientes - Adopción del fallo, que entra en detalle sobre ese tema. Lo mejor que podemos hacer es prepararnos para este caso comprendiendo el mencionado dominio del fracaso.



03 Rastreo distribuido

Al final, el elemento importante para nosotros como desarrolladores es volver a nuestra creatividad y motivación. Queremos trabajar en lo que hace avanzar a la empresa, nuestros casos de uso de negocio. No queremos abarrotar nuestro código de origen con millones de métricas o puntos de rastreo. Y lo más importante es que no queremos la constante actividad de "apagar fuegos" ni la corrección de errores.

Sin embargo, en el último punto, un sistema de observabilidad limpio, útil y que proporcione información puede ayudar en todas esas tareas. Proporcionando bibliotecas de "cobertura delgada" o, mejor aún, instrumentación automática, podemos mantener nuestra base de fuentes limpia, fiel al caso de uso del negocio. También nos devuelve mucho tiempo: el que necesitábamos para analizar problemas complejos en sistemas distribuidos, el que empleábamos para añadir puntos de métrica, el que dedicábamos a entender la interacción y la comunicación entre sistemas y, por último, el que perdíamos en encontrar cuellos de botella. Hoy en día, una huella distribuida puede ayudarnos a comprender las deficiencias y la evolución del sistema actual antes de que nos encontremos con cuellos de botella o problemas de escalabilidad.

Tiempo de resolución
Cuando una implementación falla o nuestra nueva versión se comporta de forma errática, tenemos que ser rápidos. O bien se retrocede a una versión anterior o se analiza el problema, se encuentra una solución, se aplica y se sigue adelante.

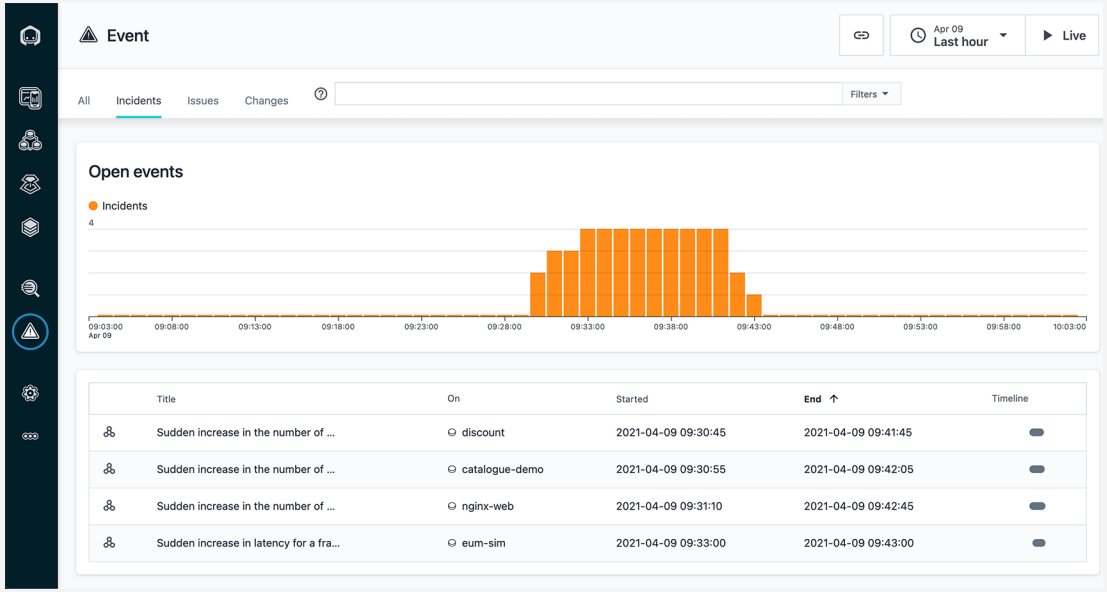


Figura 4. Una visión rápida del estado de un servicio con marcadores de liberación. Fuente: Instana.

03
Rastreo distribuido

A veces, el problema es solo un pequeño descuido, reproducible localmente y rápidamente solucionable.
Sin embargo 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 forma de prever la complejidad de los problemas que surgirán en nuestros entornos de producción. Debido a esa falta de previsibilidad, necesitamos recopilar los datos necesarios de la producción a medida que esta se va ejecutando.

Al proporcionar una forma sencilla de ver el flujo de la solicitud utilizando la huella distribuida, es fácil reunir rápidamente información sobre dónde falló la solicitud. También podemos ver si el error subió por la pila o se gestionó en algún punto intermedio, y si un usuario se vio afectado.

Las huellas distribuidas también proporcionan información sobre los llamadores y los llamados junto con sus respectivas cabeceras e información de tiempo o reintentos.

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

Este enfoque de análisis descendente no solo es mucho más rápido, sino que también ofrece la posibilidad de encontrar rápidamente la causa real del problema. Obtenemos una visión general de todos los aspectos del problema e identificamos los que tenemos que incluir en la investigación en función de las dependencias reales ascendentes o descendentes.

En resumen, cuando se busca el problema, la observabilidad proporciona la información que necesitamos para actuar rápidamente y, en el mejor de los casos, con una dependencia mínima de otros equipos o socios externos, como las empresas de hosting. Esta última puede ser especialmente lenta en responder cuando se investiga un problema de tiempo crítico.

La implementación ha salido mal
Históricamente, como desarrollador, yo podría obviar esta sección. Las implementaciones no eran mi responsabilidad.

03 Rastreo distribuido

Hoy en día, sin embargo, los desarrolladores escriben configuraciones Dockerfile para construir imágenes Docker y a menudo también proporcionan configuraciones de implementación en forma de descriptores YAML de Kubernetes (K8s), describiendo los contenedores, servicios y más, o al menos partes de ellos, por buenas razones.

No es una responsabilidad exclusiva de DevOps proporcionar esos descriptores, ya que una Implementación completa es más comúnmente una combinación de descriptores de implementación K8 de ingeniería y DevOps.

De todos modos, nuestra última Implementación falló, y no hay ningún problema obvio con el proceso de Implementación en sí, pero el servicio acaba inmediatamente después de la Implementación. Ahora es nuestro momento de brillar.

El primer paso consiste en averiguar. ¿Fuimos realmente nosotros? La observación de una huella distribuida puede responder rápidamente a esta pregunta. ¿Nuestro servicio provoca un error, por ejemplo, el estado de HTTP 500, o el problema viene de más abajo en el paquete? ¿La dependencia del servicio descendente devuelve el fallo? Quizá estemos enviando los datos equivocados. De nuevo, un vistazo rápido puede responder a esa pregunta también.

Las soluciones de observabilidad proporcionan las pruebas necesarias, comunicadas originalmente por los sistemas y basadas en nuestro análisis del dominio de los fallos, como observaciones precisas y contextuales. Las huellas distribuidas son el principal testigo cuando se busca una forma de obtener rápidamente una comprensión de situaciones aún desconocidas. El tiempo promedio de reparación y el tiempo promedio de restablecimiento del servicio son las métricas clave aquí.

¿Recuerda la regla de "no implementar el viernes"? Yo también, y apuesto a que todos los demás también. Hoy en día, con una buena capacidad de observación y la posibilidad de comprobar inmediatamente los pequeños cambios en el comportamiento, la latencia o la tasa de error, podemos y debemos implementar los viernes, aunque quizás a primera hora de la mañana. En este caso, nos quedan bastantes horas para que se solucione un problema o para retroceder si ocurre algo.

Si no podemos arreglar un problema inmediatamente porque nos aventuramos demasiado hacia lo desconocido, nuestra única opción es retroceder a una construcción más antigua. Con un flujo de implementación automatizada, este proceso es bastante fácil. Ponemos en marcha nuestro canal de integración continua y entrega continua (CI/CD) con una etiqueta o versión de lanzamiento diferente y listo. Acabamos de ganar mucho tiempo para investigar más a fondo, arreglar con más cuidado y eliminar el estrés adicional para ser lo más rápidos posible.

04
Ingeniería de alto rendimiento

Después de tanto hablar de acelerar la ingeniería y de dedicar el tiempo a lo que aporta más valor, tenemos que hablar de la propia ingeniería y de cómo cambia o ha cambiado ya en años anteriores.

Página 18 del informe Aceleración del estado de DevOps 2019 sacó a relucir 4 puntos necesarios para un equipo de ingeniería de alto rendimiento en los próximos días. Las viñetas apuntan al hecho de que los administradores de ingeniería, operaciones y base de datos ya no están en esos ámbitos totalmente separados.

Pero, ¿qué significa realmente esta afirmación? ¿Vamos a ser operaciones? Desde luego que no. Nunca me consideraría una persona de operaciones, aunque trabajo bastante a diario. La diferencia, sin embargo, sigue existiendo. No soy en absoluto un experto en ingeniería de implementación, aunque otros sí lo son. Y esta afirmación es válida para cualquier otra materia, incluso en ingeniería. No todo el mundo es ingeniero de rendimiento. Sigue siendo necesario contar con esas personas altamente especializadas, pero obtener una visión global es importante para todos, más importante que nunca.

04
Ingeniería de alto rendimiento

Las 4 preguntas que queremos responder son:

- ¿Cuál es nuestra frecuencia de implementación?
- ¿Cuál es nuestro tiempo de oportunidad de negocios para los cambios?
- ¿Cuál es nuestro tiempo para restaurar el servicio?
- ¿Cuál es nuestra tasa de fallas de cambio?

Las respuestas a estas preguntas nos indican la rapidez con la que podemos iterar. Nuestras respuestas son una buena guía, pero sea sincero. Lo mismo ocurre con nuestra base de código; solo podemos optimizar si entendemos 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 una visión mejor y más rápida de cómo se comportan las nuevas versiones.

¿Cuánto tiempo tarda el código en salir en vivo?
¿Cuánto tiempo necesitamos realmente para una implementación? ¿Es una operación importante, posiblemente un tren de liberación entre todos los equipos? Tal vez podamos implementar independientemente. O podemos implementar automáticamente después de que todas las pruebas estén en verde.

¿Con qué rapidez podemos recuperarnos de una interrupción?
Si algo va mal, ¿quién puede analizar el problema? Mi pregunta favorita es: "¿quién puede resolver el problema de mi servicio si estoy de vacaciones, sin teléfono ni conexión a Internet, y cuánto tardará la persona en entender el problema en primer lugar?" ¿Cuántas implantaciones fallan?

Por último, pero no menos importante, la pregunta más común, la que la mayoría de la gente probablemente puede responder sin pensar demasiado: ¿fracasan las implantaciones y, en caso afirmativo, cuántas y por qué?

Esta lista es un excelente punto de partida para encontrar y comprender la velocidad de las iteraciones en nuestra empresa, equipo y servicio o aplicación.

Sin embargo, yo iría un paso más allá y diría que falta una pregunta importante: preguntar sobre las diferencias entre las versiones. Es genial tener un número bajo de implementaciones fallidas, y ya podemos implementar varias veces al día. Sin embargo, todavía hay una pregunta sin hacer: ¿cuál es la diferencia de errores y rendimiento entre las versiones?

Las mejores implementaciones y la iteración más rápida no valen nada si cada versión se desempeña peor que la anterior o aumenta la tasa de error. Al final, no queremos regresar 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 de 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 todavía están en la fase de adopción, pero es casi seguro que los nuevos sistemas ya no se construyen de forma monolítica. La escalabilidad se ha convertido en algo demasiado importante, y escalar verticalmente es demasiado caro y, en algunos casos, inalcanzable.

Los microservicios, aunque no son la solución milagrosa que muchos proclaman, están definitivamente aquí para quedarse. Dividir grandes porciones de trabajo en pedazos pequeños e implementarlos 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 es que esta escalabilidad tiene un costo al añadir muchas operaciones de red entre los servicios. Las operaciones pueden agotarse, fallar o devolver basura. Estamos luchando contra un nuevo tipo de enemigo.

Además, recuerdo la época en que los datos se almacenaban en bases de datos relacionales. No se hicieron preguntas; no hubo 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 de gráficos o de series temporales, hasta almacenes de columnas para agregaciones extremadamente rápidas.

04
Ingeniería de alto rendimiento

Todos esos sistemas son diferentes y se comportan de forma distinta. Nadie puede ser experto en todos ellos. Sin embargo, lo bueno es que como desarrolladores tenemos la oportunidad de elegir la mejor herramienta para el trabajo mientras aprendemos cosas nuevas.

Por último, pero no por ello menos importante, hoy implementamos los sistemas y servicios de forma diferente a como lo hacíamos hace unos años. La implementación en sistemas dedicados se ha convertido en la minoría de las nuevas instalaciones. Ahora casi todo el mundo implementa en máquinas virtuales. Muchos ya implementan en la nube o en entornos auto-hospedados con Kubernetes o Cloud Foundry. Docker y otros tiempos de ejecución de contenedores, como CRI-O, son habituales en las máquinas de los desarrolladores.

Todas esas tecnologías son maravillosas al acercar el desarrollo en mi máquina al entorno en el que finalmente se ejecutarán. Siempre me disgustó la frase "funciona en mi máquina", aunque fuera cierto. La razón por la que no me gustó es simple. Significaba que algo estaba pasando, algo que no podía entender o explicar inmediatamente.

Canales de desarrollo
Si queremos iterar más rápido, tenemos que tener un buen soporte. Aunque las pruebas de unidad no son, afortunadamente, nada nuevo y las pruebas de integración básica se emplean habitualmente, todo lo que viene después de ellas se sigue despreciando por ser demasiado complejo, demasiado caro o no lo suficientemente relacionado con un desarrollador.

Sin embargo, con los ciclos de iteración cortos y los sistemas distribuidos, las pruebas de integración son más importantes que nunca. La integración continua en un sistema más cercano al montaje y a la producción es imprescindible para conseguir ciclos de retroalimentación rápidos durante el desarrollo. También son una buena etapa para recopilar información sobre el primer rendimiento y la tasa de error. La detección de posibles problemas en una fase temprana del desarrollo de una función puede evitar largos ciclos de reconstrucción 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 regulares siempre que sea posible. Las razones son las mismas que antes: ciclos de retroalimentación cortos y 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 todos los elementos mencionados anteriormente implementados. También requiere el valor de exponerse a fracasar. Con el conocimiento de que la comprensión de los fallos, el rápido análisis de la causa raíz y la implementación de correcciones de errores no son splo posibles, sino que cuentan con el soporte de todos los miembros del equipo y las herramientas a lo largo del proceso de producción, el valor es mucho mayor, automáticamente. Nos sentimos mucho más seguros de cara al futuro.

Dicho esto, hay que reconocer rápidamente las situaciones de fracaso. Las herramientas de observabilidad enriquecidas por el contexto tienen que apoyar el proceso de análisis del problema y ayudar a encontrar la causa raíz en el menor tiempo posible. Además, todos los servicios deben ser tan resistentes a los fallos de las dependencias como sea posible.

Una vez que se ha producido y confirmado la corrección, la cadena de producción se pone en marcha y construye, prueba e implementa la nueva versión en producción. Los servicios pequeños e independientes son más fáciles de manejar con este proceso que los grandes monolitos.

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

Y estamos en vivo
Justo después de poner en vivo una implementación, la pregunta más importante más allá de “¿funciona?” era “¿cómo se desempeña?”

Como ya se ha dicho, un ciclo de entrega rápido solo tiene sentido cuando las nuevas versiones no son generalmente peores en rendimiento que las antiguas.

04
Ingeniería de alto rendimiento

Existen excepciones a esta regla, pero solo debería ser un éxito de rendimiento conocido y esperado por adelantado, algo ya calculado en el posible escalamiento de la infraestructura.

Las métricas clave importantes que hay que vigilar inmediatamente después de la implantación son cualquier tipo de cambio inesperado en la latencia promedio, la tasa de error o las llamadas descendentes, por ejemplo, el número de llamadas a la base de datos o similares. Las magníficas herramientas de observabilidad ofrecen comparaciones directas del antes y el después de la implementación para facilitar la accesibilidad.

De todos modos, a nuestra implementación le ha ido bien, los números inmediatos parecen buenos y el sistema se comporta dentro de los rangos esperados. ¿Es este el final de la historia? ¿Nuestra implementación vive feliz para siempre?

Desgraciadamente, sabemos que este resultado no suele darse en los entornos de producción. Algo está a punto de suceder todo el tiempo. Prepárese. Aunque los tiempos de "apagar fuegos" se han reducido en gran medida, nunca desaparecerán del todo. Recuerde que construir un sistema infalible es simplemente imposible.

Es cierto que en esta etapa del proceso, hay menos participación directa de la ingeniería cuando algo

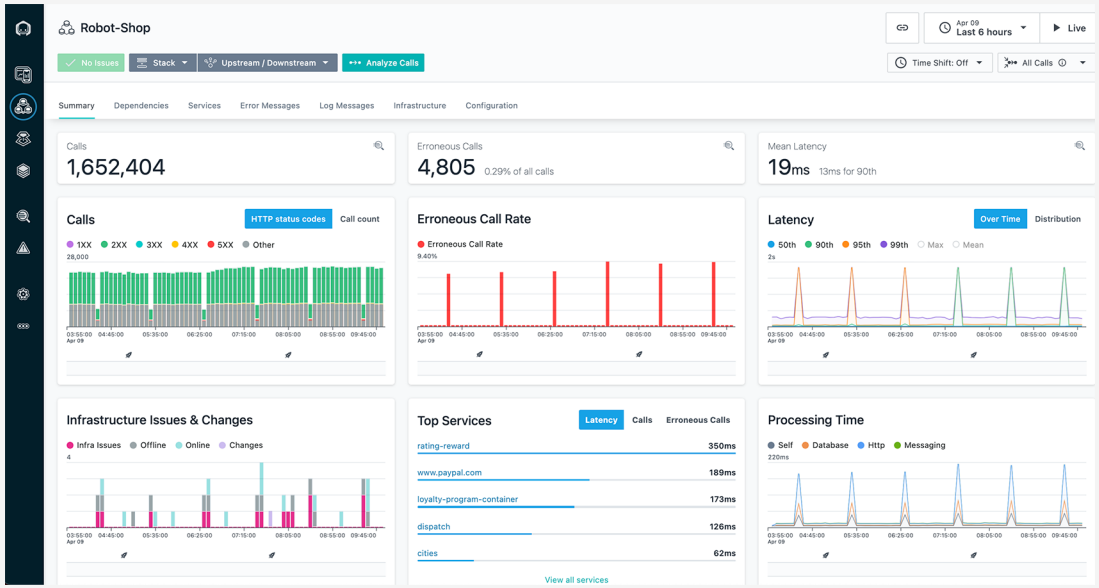


Figura 5. Marcadores de lanzamiento en el panel de control de los servicios para una visión inmediata después y entre el lanzamiento, fuente: Instana

04
Ingeniería de alto rendimiento

va mal. Sin embargo, cuando se nos llama y se nos pide que intervengamos, una visión general inteligente de todos los componentes, redes, máquinas, aplicaciones y su interconexión nos ayuda a profundizar rápidamente. Es una tarea que solo se complica con cada porción de abstracción, o complejidad arquitectónica, que añadimos al sistema. Lo que nos simplifica y facilita la vida durante el desarrollo, aumenta la complejidad cuando se analizan parcialmente sistemas desconocidos.

Solo hay una medida sobre la que debemos juzgarnos a nosotros mismos, especialmente para las situaciones de guardia: el MTTGBTB-Tiempo promedio para regresar a la cama. Gracias a Karthik Kumar por esta métrica tan importante.

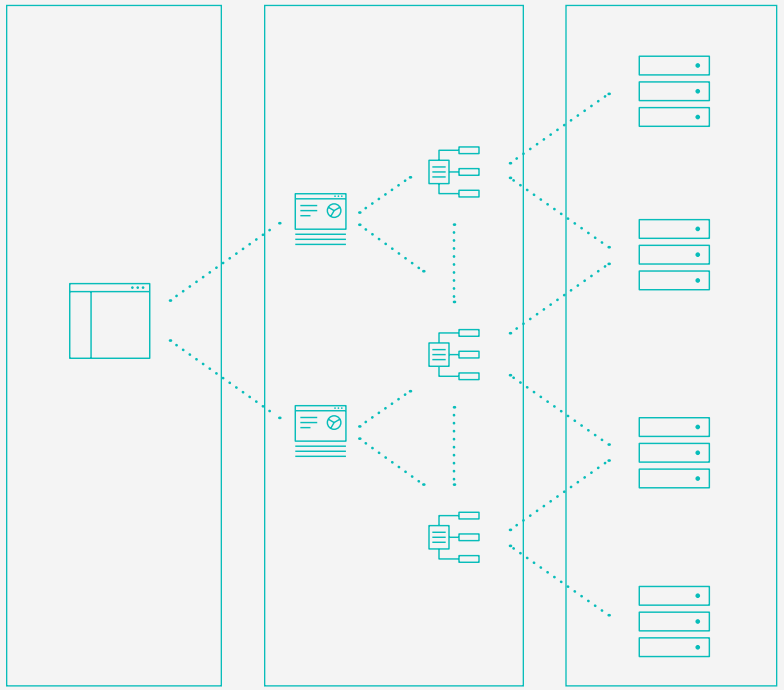


Figura 6. La complejidad arquitectónica impide a veces un razonamiento rápido

05 Desarrollo basado en la observabilidad

Ahora que entendemos la importancia de la observabilidad, veamos el valor del desarrollo impulsado por la observabilidad (ODD). 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 aplicarse desde el principio y ejecutarse a lo largo del ciclo de desarrollo. Estas pruebas permiten conocer el rendimiento antes, durante y después del desarrollo de la función. Ayudan a comparar las primeras expectativas con la realidad y proporcionan información si el sistema de producción requiere ser escalado antes de la implementación final de las funciones.

Construir el ciclo de desarrollo en torno a la idea de ciclos cortos de retroalimentación. La verdadera clave es hacer de la observabilidad una parte central del proceso de desarrollo. Haga que sea un pensamiento proactivo y trabaje en conjunto con los otros equipos. No construya silos.

Durante esas interacciones, proponga estados métricos y comience a aplicarlos al principio del proceso de implementación de la función. No olvide tener en cuenta el número de métricas. La selección óptima incluye 3 - 5, como máximo 10 métricas.

Si la solución de observabilidad requiere una integración manual mediante envolturas, añádalas también desde el principio. Si podemos hacer que nuestro servicio sea instrumentado automáticamente, mejor aún. Nuestro código -nuestro"arte"- permanece intacto.

Para las optimizaciones, no hay que adivinar. Para seleccionar los puntos de optimización, hay que tener en cuenta todos los datos proporcionados por la solución de observabilidad, las métricas, las huellas distribuidas y la información de la infraestructura. Encuentre los factores que más contribuyen a la latencia y a los índices de error y arréglelos primero. La fruta que cuelga bajo brinda victorias rápidas y motivación.

Además, eche un vistazo a los datos de perfilado de código. Un perfilador de producción siempre activo aumentará nuestra comprensión a un nivel que antes era casi imposible de obtener. Hasta hace poco, nunca había tenido la oportunidad de tener un perfil en mi computadora que se acercara a lo que se veía en la producción. Pero por favor recuerde, no utilice cualquier perfilador. Los perfiladores listos para la producción están hechos a medida para su respectivo caso de uso, con una sobrecarga extremadamente baja.

El último consejo probablemente dolerá tan solo con leerlo. Pruebe... espere a verlo... en producción. Sí, ha leído bien. Pruebe en producción, pero con marcadores de funciones. Habilite el nuevo comportamiento, la nueva función, con base en un conjunto de identificaciones de usuario y parámetros especiales dados, y sea creativo.

El razonamiento que subyace a este paso, y el alejamiento total de la antigua regla de no probar nunca en producción, es la singularidad de cada entorno. El estado de la infraestructura, las dependencias, la fecha y la hora, la implementación, el entorno en sí, y la fase lunar... todo puede afectar a nuestro código.

Después de toda la discusión sobre el desarrollo y de por qué la observabilidad facilita la vida del desarrollador, ¿cómo podemos aplicarlo? En primer lugar, pregunte al equipo de DevOps si ya tiene una solución que utilice otro equipo.

06
Ser o no ser... código abierto

Como en casi todo, la comunidad de código abierto ha hecho un magnífico trabajo al ofrecer al mundo varias opciones. Las opciones son buenas, ¿no?

Sí y no. Uno de los principales obstáculos de la mayoría de las soluciones es que o bien realizan una supervisión de las métricas o un 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 tableros independientes , silos de datos independientes y, por lo general, no tienen una forma de saltar entre métricas, huellas e infraestructura para una única solicitud.

Creación manual de la observabilidad
Para crear observabilidad, es común utilizar dos herramientas separadas, como se mencionó, una para el rol de métricas y supervisión, generalmente Prometheus, y otra para el rastreo distribuido, generalmente Jaeger o Zipkin.

Estas soluciones de código abierto están disponibles "de forma gratuita", explico la razón de las comillas, y pueden integrarse con un servicio de aplicación. La integración en este caso significa que tenemos que añadir el rastreador y el recolector de métricas a nuestro código de origen. Existen envoltorios para las bibliotecas más comunes en muchos lenguajes de programación y pueden utilizarse directamente.

Un ejemplo sencillo para una métrica de Prometheus, midiendo el número de solicitudes por segundo, sería similar al siguiente fragmento, utilizando una instancia de contador para contar el número de solicitudes. Ver Figura 2.

No está mal para el uso directo. Afortunadamente, muchos marcos de trabajo proporcionan este tipo de métricas con Prometheus de forma inmediata.

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

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

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

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

Figura 8 Ejemplo de Hibernate con Prometheus

06
Ser o no ser... código abierto

Vemos que todavía no es realmente complicado, pero de alguna manera se siente como si estuviéramos en el nivel de "regreso a los viejos tiempos". Lo peor es que abarrotamos nuestro código con probablemente más ceremonias de supervisón y rastreo que de lógica de negocio real.

Dar sentido a los datos
Junto con los datos de registro almacenados en algo como Logstash o Splunk, ahora hay un montón de datos a los cuales recurrir cuando ocurre un problema. Sin embargo, hay otro problema: la propia pila de observabilidad.

Con 3 mil sistemas independientes que no comparten datos o correlaciones de datos, dar sentido a los diferentes orígenes de datos es complicado y requiere mucho tiempo. El usuario es el que tiene que hacer coincidir las marcas de tiempo, encontrar las correlaciones, establecer la conexión y, finalmente, construir el contexto para resolver el enigma en cuestión.

Hay que tener en cuenta que dar sentido a todo esto es el mayor 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);

  Configuración.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 rastreo de Java con Jaeger

06

Ser o no ser... código abierto

El costo real del código abierto
¿Recuerda que antes mencioné "de forma gratuita" cuando hablamos de ello un poco? Ahora es el momento.

En contra de la creencia común, el código abierto no es gratuito. Es cierto que no hay ningún costo de licencia para poder utilizarlo. El mayor costo de la observabilidad del OSS se mide en tiempo, ya que las empresas pagan a las personas para que entiendan cómo operar el paquete de observabilidad, cómo utilizarlo y cómo dar sentido a los datos. En esta ocasión, a menudo lo que tenemos, se aleja de la entrega de funcionalidad del negocio.

También puede ser necesario pagar a ingenieros adicionales para que ajusten las soluciones de observabilidad del OSS. No olvidemos tampoco el tiempo necesario para integrar la supervisión y el rastreo distribuido. Todas estas tareas implican costos adicionales que se suman al tiempo necesario para implementar la observabilidad del OSS en la base de origen.

Mantener esas integraciones en línea con nuestros objetivos previamente establecidos puede fácilmente añadir hasta 5 % - 10 % del tiempo de desarrollo, tiempo que en realidad tratamos de evitar cuando dejamos la era de Zabbix y Nagios.

Cuando se observan los equipos de DevOps y Operaciones, este número solo aumenta porque ahora estamos operando varios sistemas, que es otro caso. Solo estamos observando nuestro caso de desarrollo.

Por último, pero no menos importante, también hay un costo por el funcionamiento del paquete, el espacio de almacenamiento necesario para guardar todos los datos, y el tiempo de cálculo de intentar reimplementar incluso las correlaciones automáticas más básicas.

De repente, estas herramientas"gratuitas" de código abierto se han vuelto muy caras y siguen presentando brechas de observabilidad y contexto.

La solución del proveedor
Las soluciones comerciales, en cambio, ofrecen muchas, si no todas, las funciones anteriores en una única solución. Si ciertas partes no se proporcionan directamente, integran servicios externos hasta un punto que parece que se almacena internamente. Se incluye la correlación de datos de los sistemas internos y externos.

Sin embargo, la correlación de datos entre todas las partes de nuestros sistemas es el elemento más importante. La infraestructura, las aplicaciones, los servicios, la red, el registro... todos los datos se preprocesan de forma que la información necesaria sea fácil de recopilar y tenga sentido.

Algunos proveedores van un paso más allá y correlacionan los problemas encontrados en diferentes partes del sistema para crear alertas más accionables y específicas. La correlación de todas las incidencias o eventos que van juntos proporciona una visión más rápida sobre el alcance y la causa de una incidencia, los servicios relacionados y si hay impacto para el usuario.

Además, algunos proveedores ofrecen una visión inmediata de los cambios antes y después de los lanzamientos, como se ve en la Figura 5. Para lograr esa visión, se requiere la integración con canales CI/CD comunes, servicios DevOps y herramientas de desarrollo.

06

Ser o no ser... código abierto

Más allá del código abierto
Sin embargo, como desarrolladores, buscamos el trabajo de desarrollo . En lugar de configurar todo manualmente, algunas soluciones comerciales ofrecen una instrumentación del código totalmente automática.

¿Recuerda nuestro ejemplo de código de antes y se pregunta cómo sería si se utilizara una instrumentación de código totalmente automatizada?

Eso es correcto, no se necesitan cambios de código para la monitorización de la adición, el rastreo distribuido o la recopilación de rendimiento. Automation recoge los servicios, sin importar si se iniciaron directamente, en un contenedor Docker o dentro de Kubernetes, Red Hat® OpenShift® Platform o entornos similares. Tras el descubrimiento automático, el servicio se instrumenta sobre la marcha, se generan tableros basados en las mejores prácticas identificadas para el servicio o la plataforma en cuestión, y estamos listos para empezar. No es necesaria ninguna intervención manual. ¿Qué más podemos pedir?

La ventaja de la correlación automática
Como ya se ha mencionado, la principal ventaja de una solución de observabilidad empresarial es la capacidad de correlacionar las métricas y las huellas de la máquina, la infraestructura y las aplicaciones y servicios. Las huellas distribuidas entregan la comprensión del 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 varios tableros para diferentes servicios es que es casi imposible hacer coincidir los tiempos y recoger el contexto general del problema.

La mayor ventaja de la correlación automática, como se ha descrito anteriormente, es el insight inmediato. Al examinar un problema o incidente, la solución del proveedor ya ha hecho todo el trabajo de detective para proporcionar las piezas importantes de información como evidencia contextual y llevarnos directamente al punto 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 nuestra vidas

El costo de las soluciones del proveedor
A diferencia de las herramientas de código abierto, las soluciones de los proveedores no pretenden ser gratuitas.

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

Con la eliminación del costo de integración de la supervisión y el seguimiento en los servicios y la infraestructura, las soluciones de los proveedores, al final, suelen ser más rentables, con menos costos operativos y de desarrollo. Y no olvidemos el costo de almacenamiento necesario para guardar toda la información y los datos.

Tomar una decisión informada
Elegir entre un código abierto o una solución comercial es preguntar cuánto tiempo queremos dedicar al desarrollo de funcionalidades no relacionadas con el negocio, no es una cuestión de costo, ya que ninguna de las dos soluciones es gratuita.

Hasta ahora, nos hemos centrado en los desarrolladores a los que no les gusta añadir la supervisión y el rastreo a su código, y ciertamente, ese es el mayor grupo. Sin embargo, hay desarrolladores a los que les gusta esforzarse. Yo, por mi parte, me considero alguien con el gen del esfuerzo. Aun así, prefiero jugar con las nuevas tecnologías, no con una métrica de rendimiento.

Aparte de eso, mi experiencia me dice que nos olvidamos de las recopilaciones de datos más importantes cuando las hacemos manualmente, y no solo una vez. O volvemos a caer en el viejo hábito de añadir todo lo posible en lugar de centrarnos en nuestros dominios de fracaso previamente definidos.

Las soluciones de los proveedores ofrecen un conjunto cada vez mayor de mejores prácticas para la recopilación automática de métricas y comprenden lenguajes de programación, marcos de trabajo e integraciones de servicios. Estas buenas prácticas eliminan la mayor parte del trabajo a la hora de definir los posibles dominios de fallo para sistemas nuevos o desconocidos. Con las soluciones de los proveedores, es raro que volvamos a perder información importante.

07
Simplificar nuestra vidas

La generación de software actual difiere en gran medida de los métodos tradicionales que hemos utilizado durante las décadas anteriores. Las aplicaciones se están dividiendo en pequeños microservicios, mientras que las implementaciones se construyen sobre Docker, Kubernetes y canales automáticos de CI/CD.

Seguir todos esos cambios durante las operaciones y analizar los problemas se torna cada vez más complicado. Las nuevas formas de reunir insights de lo que está ocurriendo son importantes, sobre todo cuando se trata de una llamada y se intenta entender un problema en un componente desconocido .

La observabilidad de la empresa es la clave. Elija la lucha correcta; no pelee con su solución de observabilidad.

© Copyright 2021 Instana, una compañía de IBM

IBM Argentina
Pje. Ing. Enrique Butty 275
C.A.B.A – Argentina

Producido en los Estados Unidos de América
Abril de 2021

IBM, el logotipo de IBM e ibm.com son marcas registradas de International Business Machines Corporation, registradas en muchas jurisdicciones de todo el mundo. Otros nombres de productos y servicios pueden ser marcas registradas de IBM o de otras compañías. Una lista actualizada de las marcas registradas de IBM puede ser consultada en ibm.com/trademark.

Instana es una marca registrada o una marca comercial registrada de Instana, Inc., una Compañía de IBM.

Red Hat y OpenShift son marcas registradas de Red Hat, Inc. o sus filiales en los Estados Unidos y otros países.

Este documento está actualizado conforme a la fecha inicial de la publicación y puede ser modificado por IBM en cualquier momento. No todas las ofertas están disponibles en todos los países donde opera IBM.

LA INFORMACIÓN DE ESTE DOCUMENTO SE PROPORCIONA "TAL CUAL", SIN NINGUNA GARANTÍA, EXPLÍCITA O IMPLÍCITA, NO INCLUYE NINGUNA GARANTÍA DE COMERCIALIZACIÓN E IDONEIDAD PARA UNA FINALIDAD CONCRETA NI CUALQUIER GARANTÍA O CONDICIÓN DE NO INFRACCIÓN. Los productos de IBM están garantizados de acuerdo con los términos y condiciones de los acuerdos bajo los cuales se proporcionan.