WHITEPAPER

5 INSTRUMENTATION STRATEGIES FOR ARCHITECTING CONTAINERIZED APPLICATIONS (5 ESTRATEGIAS DE INSTRUMENTACIÓN PARA LA ARQUITECTURA DE APLICACIONES EN CONTENEDORES)

David Gordon

RESUMEN EJECUTIVO

Mientras las organizaciones en todo el mundo adoptan arquitecturas de aplicaciones basadas en contenedores, los desarrolladores deben adaptar las implementaciones de software existentes (originalmente implementadas en la infraestructura tradicional) para lograr un entorno de plataforma en contenedores. Este whitepaper presenta cinco temas de interés fundamentales para la implementación de aplicaciones en contenedores y las estrategias de instrumentación relacionadas que aprovechan las populares tecnologías de open source y nativas para contenedores.

CONFIGURACIÓN EXTERNALIZADA

En general, la entrega continua en entornos de contenedores se basa en el principio de que la imagen de un contenedor es un artefacto inmutable y que se puede certificar. Esto implica que la misma imagen probada en un entorno de QA (control de calidad), por ejemplo, se implementa sin cambios en el siguiente entorno, en la secuencia de canales de entrega e integración continuas (CI/CD). Sin embargo, algunas configuraciones de aplicaciones suelen estar destinadas a alternar entre entornos por un motivo justificado. Algunos ejemplos comunes de configuraciones de aplicaciones específicas del entorno incluyen cadenas de conexión de bases de datos externas, endpoints de interfaces de programación de aplicaciones (API) públicas e indicadores de activación y desactivación de funciones. A fin de preservar la inmutabilidad del contenedor, la configuración específica del entorno se debe cargar desde una fuente externa.

Las arquitecturas de software tradicionales suelen emplear una técnica en la que las aplicaciones consumen datos de la configuración específica del entorno ubicados en un sistema de archivos adjunto a una máquina que aloja las aplicaciones. Bajo este enfoque, se asume que una aplicación determinada se implementa en un host predecible o en un conjunto de ellos.

Las plataformas como Kubernetes y OpenShift usan un algoritmo de programación para evaluar la infraestructura disponible y luego asignan de forma dinámica los contenedores a los host en el clúster. Un contenedor puede asignarse a cualquier nodo de host que cumpla con los criterios del algoritmo de programación. En consecuencia, la información de configuración de las aplicaciones debe estar disponible en cualquier lugar del clúster.

Los enfoques modernos para proporcionar configuración de aplicaciones en un entorno de contenedores tienen un principio básico en común: proporcionar datos de configuración a través de un endpoint de servicios.

El servidor Spring Cloud Config es un conocido servicio de proveedor de configuración de aplicaciones de open source. De forma predeterminada, el servicio expone un endpoint RESTful que devuelve archivos de propiedad de configuración que se conservan en un repositorio Git. La comunidad Spring también proporciona un cliente del lado de las aplicaciones para recuperar información de configuración. Al usar estos componentes, se puede configurar una aplicación Spring para que, periódicamente, sondee el servicio de proveedor de configuración y recargue su contexto si se detectan cambios. Esta estrategia requiere una instancia en ejecución de servidor Spring Cloud Config y un almacén de datos de configuración como un repositorio Git alojado.

Un enfoque alternativo es la API OpenShift, que expone un objeto ConfigMap que puede representar un archivo, como un archivo de propiedad de aplicación, al que cualquier pod puede acceder sin autorización en el clúster. El cliente de Kubernetes de Spring Cloud Config permite que las aplicaciones Spring usen los datos de configuración directamente desde los ConfigMap en tiempo de ejecución. Por lo tanto, incluso sin un servidor Spring Cloud Config, una aplicación Spring implementada en OpenShift puede usar la configuración de forma dinámica. El cliente es compatible con Red Hat® JBoss® Fuse y su desarrollo fue upstream en la comunidad Fabric8.

ADMINISTRACIÓN DE REGISTROS

Por lo general, los registros de las aplicaciones se escriben en archivos en un disco y es posible suponer que el host de aplicaciones es predeterminado. Saber dónde se ejecutará un proceso de aplicaciones significa que los equipos de operaciones saben dónde se escribirán los registros. Dado que los contenedores se asignan de forma dinámica a los nodos cuando se implementan en una plataforma de contenedores, como Kubernetes, una aplicación diseñada para escribir registros en su sistema de archivos local no es eficiente. Los registros con varias convenciones de formato, nombre y ruta terminan dispersos por toda la infraestructura del clúster. Además, el sistema de archivos local de un contenedor suele seguir el ciclo de vida de este. Si se destruye un proceso del contenedor, todo lo que esté escrito en el sistema de archivos local también se destruirá.

Se requiere una convención en todo el clúster para el destino y el formato del registro a fin de lograr una solución de administración de registros nativos del contenedor. Esto se puede obtener al delegar la responsabilidad al motor del contenedor. Por ejemplo, de forma predeterminada, el docker captura la información de salida estándar y de errores estándar para el proceso principal de un contenedor, y la escribe en los archivos mediante un formato con base JSON estandarizado. Estos archivos se ubican en /etc/docker y tienen un nombre único con la id. del contenedor que produjo el registro. En este caso, el impacto en la implementación de la aplicación es sutil: dirige todos los registros de las aplicaciones a la salida estándar. Los marcos de registro, como Logback (el autoproclamado sucesor de log4j), pueden dirigir los registros a una salida estándar con una configuración sencilla. Diseñe aplicaciones basadas en contenedores para usar registradores de marcos que se pueden configurar, como Logback, para que el formato y el destino de salida se puedan administrar de forma eficiente. Los únicos destinos de registro deben der los de salida estándar (stdout) y error estándar (stderr).

Además de la organización del registro del contenedor, los equipos de operaciones necesitan una estrategia para recolectar y buscar los registros dispersos por toda la infraestructura de la plataforma del contenedor. EFK (ElasticSearch, Fluent.d, Kibana) es el nombre que se le da a una pila de tecnología conocida por proporcionar adición de registros, almacenamiento y presentación. Los registros de contenedores y componentes de plataforma se recopilan y se transmiten a un almacén de datos distribuidos. Los datos se presentan con la interfaz de usuario de Kibana que se puede configurar en gran medida, con capacidades que incluyen diseñar paneles e investigar el comportamiento individual del pod.

DETECCIÓN DISTRIBUIDA

La detección cuenta la historia de una transacción a medida que se propaga a través de un sistema distribuido. Por lo tanto, la implementación de una detección debe reconstituir la información sobre una transacción a partir de los datos recopilados de distintos componentes de un sistema.

Las aplicaciones basadas en contenedores suelen implementarse como muchos componentes que funcionan de forma conjunta, como un sistema. Las aplicaciones diseñadas como un conjunto de servicios modulares y que se pueden implementar de forma independiente siguen un popular estilo conocido como "microservicios". Si bien es totalmente opcional seguir los patrones estrictos de microservicios al desarrollar aplicaciones para implementarlas en una plataforma de contenedores, es importante tener en cuenta que las aplicaciones implementadas en plataformas de contenedores tienden a estar compuestas por múltiples contenedores. La detección de transacciones mediante una arquitectura altamente distribuida o desglosada es un gran desafío, ya que las fuentes de datos para detectar están dispersas a lo largo de un grupo de infraestructura. Actualmente, varias soluciones de open source para la detección en sistemas distribuidos gozan de un creciente protagonismo en el sector.

Zipkin es un sistema de detección de open source que se ha adoptado mucho en los últimos años y ha ganado popularidad como parte del ecosistema del marco Spring. Otras organizaciones, como Uber, han desarrollado nuevas implementaciones de herramientas de detección distribuida.

A fin de mitigar la preocupación de que la implementación de un marco de detección pueda resultar en un acoplamiento del sistema para esa implementación en particular, se fundó la iniciativa de OpenTracing para crear un estándar de detección independiente de los proveedores. Las especificaciones y convenciones semánticas descritas por la iniciativa de OpenTrancing fueron inspiradas en su mayor parte por Zipkin. Jaeger (de Uber), en el que se reemplazaron múltiples componentes de Zipkin, se adhiere a las convenciones de OpenTracing para mantener de forma efectiva la capacidad de cambiar entre implementaciones compatibles con OpenTracing.

Las soluciones que cumplen con los estándares de OpenTracing permiten la flexibilidad de la arquitectura a futuro. Algunas de las soluciones más efectivas y mejor adoptadas, como Zipkin y Jaeger, proporcionan opciones de disponibilidad inmediata.

MÉTRICAS

La recolección de métricas de aplicaciones efectivas en entornos de contenedores presenta muchos de los mismos cambios descritos anteriormente para registros y detección. Debido a la naturaleza efímera de los contenedores, los endpoints de las métricas no son estáticos; eventualmente, las instancias de contenedor terminan siendo reemplazadas por instancias actualizadas que se pueden ampliar en un nodo diferente en el clúster. Las plataformas como Kubernetes y OpenShift usan una abstracción de red llamada "servicio", que define un conjunto lógico de pods y una política por medio de la cual se puede acceder a ellos. Sin embargo, un servicio de Kubernetes proporciona un endpoint de supervisión que no es efectivo debido a que se necesitan estadísticas más detalladas sobre contenedores individuales, en lugar de un grupo de contenedores de forma colectiva.

Prometheus es un kit de herramientas de supervisión y alerta de open source que es especialmente popular en los contextos de microservicios. Prometheus incluye componentes para recolectar y mostrar métricas y un menú exhaustivo de bibliotecas de instrumentación, incluido el exportador Java Management Extension (JMX).

JMX es un estándar para la supervisión de aplicaciones Java™. Con la ayuda de los agentes de métricas como Jolokia y Prometheus JMX Exporter, un usuario de métricas puede observar una visión global de las métricas JMX desde un grupo de contenedores con la API de Kubernetes.

Se recomienda exponer las métricas de las aplicaciones para todos y cada uno de los componentes de las aplicaciones en un entorno de contenedores. Considere el uso de bibliotecas de instrumentación populares para publicar los endpoints de las métricas que cumplen con los formatos de exposición de Prometheus. Al igual que el dominio de detección distribuida, pareciera que las métricas y la supervisión de las aplicaciones tienden hacia una especificación comúnmente aceptada, y Prometheus cuenta con la popularidad para postularse como el estándar por excelencia.

COMPROBACIÓN DE ESTADO

La instrumentación de las aplicaciones en contenedores con endpoints de comprobación de estado es fundamental para las arquitecturas de autocomprobación. Las aplicaciones implementadas en la infraestructura tradicional tienden a tener direcciones de red estáticas, y la supervisión tradicional de las aplicaciones suele aprovechar la previsibilidad de un servicio o un endpoint de host. Debido a que los contenedores están programados para ejecutarse en nodos de forma dinámica, los servicios de comprobación de estado deben rastrear todas las instancias del contenedor.

Kubernetes proporciona una función que supervisa automáticamente los endpoints de comprobación de estado y responde a los contenedores que no están en un estado adecuado. Todos los contenedores en un clúster se supervisan, y Kubernetes tiene la capacidad de responder a las comprobaciones de estado mediante la implementación, la eliminación o el reinicio de los pods. La aplicación debe exponer una API para observar su estado, por lo que requiere una instrumentación menor.

La instrumentación de la comprobación de estado proporciona varios niveles de efectividad. Por ejemplo, una aplicación puede exponer un controlador aislado que devuelve un código de respuesta de 200 HTTP cuando se invoca. Esta comprobación de estado es útil en muchos casos, pero solo revela algunos tipos de problemas. Si una aplicación no tiene una conexión adecuada a una base de datos, es probable que una comprobación de estado superficial en el endpoint no detecte el problema.

Las comprobaciones de estado más efectivas realizan un inventario exhaustivo del estado de todos los componentes y las conexiones que son fundamentales para la aplicación. Spring Boot Actuator es una popular biblioteca de instrumentación de comprobación de estado que escanea un contexto de aplicaciones Spring e interroga el estado de tiempo de ejecución de cada componente que encuentra. El tipo de comprobación de estado detallado con el que es compatible Spring Boot Actuator es altamente recomendable para las aplicaciones en contenedor.

RESUMEN

Mientras la adopción de contenedores es cada vez mayor, la migración de las aplicaciones tradicionales intentará mantener este ritmo. Contar con un gran suministro de herramientas para abordar las preocupaciones comunes sobre contenedores optimizará tanto el desarrollo nuevo como la migración de aplicaciones heredadas para una implementación efectiva y con capacidad de adaptación.

En muchos casos, las implementaciones de aplicaciones pueden ser efectivas sin cambios, ya sean implementadas en una plataforma de contenedores o en una infraestructura tradicional. Sin embargo, administrar de forma eficiente las aplicaciones que están compuestas por varios contenedores y están implementadas en un grupo de infraestructura es posible mediante tecnologías que exponen la información sobre los tiempos de ejecución de las aplicaciones y permiten que las aplicaciones observen y respondan a las condiciones del entorno.

Estos recursos proporcionan más información sobre las tecnologías de instrumentación descritas en el whitepaper a continuación:

- **Spring Cloud Config Kubernetes** for externalized application configuration (https://github.com/spring-cloud-incubator/spring-cloud-kubernetes)
- **Logback** for application log management (https://logback.qos.ch)
- **Zipkin** (https://zipkin.io) and **Jaeger** (http://jaegertracing.io) for distributed tracing using the **Open Tracing** standard (http://opentracing.io)
- **Prometheus** (https://prometheus.io) as a metrics toolkit including **Prometheus JMX Exporter** (https://github.com/prometheus/jmx_exporter) for metrics publishing
- **Spring Boot Actuator** (https://spring.io/guides/gs/actuator-service) for exposing deep health check endpoints

ARGENTINA
Ingeniero Butty 240, 14º piso
Ciudad de Buenos Aires
Argentina
+54 11 4329 7300

CHILE
Avda. Apoquindo Nº 2827
oficina 701, Piso 7
Los Condes, Santiago, Chile
+562 2597 7000

COLOMBIA
Red Hat Colombia S.A.S
Cra 9 No. 115-06 Piso 19 Of 1906
Edificio Tierra Firme Bogota,
Colombia
+571 5088631
+52 55 8851 6400

MÉXICO
Calle Río Lerma 232
Cuauhtémoc
06500 Ciudad de México
Mexico
+52 55 8851 6400

ESPAÑA
Torre de Cristal
Paseo de la Castellana 259C
Piso 17 Norte
28046 Madrid
+34 914148800

facebook.com/redhatinc
@RedHatIberia
Red Hat EMEA

es.redhat.com
#f11943_0418