WHITEPAPER

# CINCO ESTRATÉGIAS DE INSTRUMENTAÇÃO PARA ARQUITETAR APLICAÇÕES EM CONTAINERS

David Gordon

## RESUMO EXECUTIVO

Organizações no mundo todo estão adotando arquiteturas de aplicação baseadas em containers. Como resultado, os desenvolvedores precisam ajustar as implementações de software existentes, que passam da infraestrutura tradicional para o ambiente da plataforma de containers. Neste whitepaper, abordamos cinco fatores essenciais na containerização de aplicações. Além disso, apresentamos as estratégias de instrumentação correspondentes que tiram proveito de tecnologias de container open source e nativas em alta no momento.

### CONFIGURAÇÕES EXTERIORIZADAS

Em geral, a entrega contínua nos ambientes de container parte do princípio de que a imagem do container é um elemento imutável e garantido. Isso significa considerar que a mesma imagem testada em um ambiente de controle de qualidade (QA) será implantada sem alterações no próximo ambiente da sequência pipeline de integração e entrega contínuas (CI/CD). No entanto, algumas configurações de aplicação variam entre os ambientes, e há uma boa razão para isso. Alguns exemplos comuns de configurações de aplicação específicas do ambiente são as strings de conexão do banco de dados externo, os endpoints da interface de programação de aplicações (API) e os sinalizadores de alternância de recursos. Para preservar a imutabilidade do container, é importante carregar a configuração específica do ambiente a partir de uma fonte externa.

Muitas vezes, as arquiteturas de software tradicionais usam uma técnica em que as aplicações consomem os dados de configuração específicos para o ambiente, localizados em um sistema de arquivos vinculado à máquina host. Essa abordagem parte do pressuposto de que uma determinada aplicação é implantada em um host ou conjunto de hosts previsíveis.

Plataformas como Kubernetes e OpenShift usam um algoritmo de programação para avaliar a infraestrutura disponível e atribuir de forma dinâmica os containers aos hosts no cluster. É possível atribuir um container a qualquer nó do host que atenda aos critérios do algoritmo de programação. Portanto, as informações de configuração da aplicação precisam estar disponíveis em todos os pontos no cluster.

As abordagens modernas para o fornecimento de configuração de aplicações em um ambiente de container compartilham do mesmo princípio básico: fornecer dados de configuração por meio de um endpoint de serviço.

Um serviço de provedor de configuração para aplicações open source muito usado é o Spring Cloud Config Server. Por padrão, esse serviço expõe um endpoint RESTful que retorna arquivos de propriedade de configuração mantidos em um repositório Git. A comunidade do Spring também fornece um cliente do lado da aplicação para recuperar informações de configuração. Ao usar esses componentes, é possível definir uma aplicação Spring para consultar periodicamente o serviço do provedor de configurações e recarregar o contexto se modificações forem detectadas. Essa estratégia requer uma instância em execução do Spring Cloud Config Server e um armazenamento de dados de configuração, como um repositório Git hospedado.

Uma abordagem alternativa é oferecida pela API do OpenShift: ela expõe um objeto ConfigMap, que pode representar um arquivo, como por exemplo um arquivo de propriedade da aplicação, acessível por qualquer pod autorizado no cluster. Com o cliente do Spring Cloud Config Kubernetes, as aplicações Spring podem usar dados de configuração diretamente do ConfigMaps no ambiente de execução. Portanto, mesmo sem um Spring Cloud Config Server em execução, a aplicação Spring implantada no OpenShift usa a configuração de forma dinâmica. O cliente é compatível com o Red Hat® JBoss® Fuse e foi desenvolvido de maneira upstream na comunidade do Fabric8.

## GERENCIAMENTO DE LOGS

Os logs da aplicação costumam ser gravados em arquivos em um disco. Novamente, o pressuposto é de que o host da aplicação é predeterminado. Ao conhecer qual é o local de execução do processo da aplicação, as equipes de operações sabem onde os registros serão gravados. Como os containers são atribuídos dinamicamente aos nós ao serem implantados em plataformas como Kubernetes, uma aplicação projetada para gravar logs no sistema de arquivos local não será eficiente. Os registros com várias convenções de caminho, nome e formatação acabam ficando espalhados pela infraestrutura do cluster. Além disso, o sistema de arquivos local do container geralmente segue o ciclo de vida do próprio container. Quando o processo é eliminado, tudo o que está gravado no sistema de arquivos local também é.

Em uma solução de gerenciamento de logs nativos do container, é necessária uma convenção de destino e formatação que atue em todo o cluster. Isso pode ser alcançado ao delegar a tarefa para a plataforma do container. Por exemplo: por padrão, o Docker coleta informações de erro e de saída relacionadas a um processo principal do container e grava essas informações nos arquivos usando um formato padronizado baseado em JSON. Esses arquivos estão localizados em /etc/docker e têm nomes exclusivos que incluem a ID do container que produziu o log. Nesse caso, o impacto na implementação da aplicação é sutil: todos os logs da aplicação são direcionados para a saída padrão. Estruturas como o Logback, o autoproclamado sucessor do log4j, podem direcionar o log para uma saída padrão com uma configuração simples. Desenvolva aplicações baseadas em container para que usem agentes de estruturas configuráveis, como o Logback. Assim, é possível gerenciar com eficiência o destino de saída e a formatação. A saída padrão (stdout) e o erro padrão (stderr) devem ser os únicos destinos da geração de logs.

Além da organização dos logs do container, as equipes de operações precisam de uma estratégia para pesquisar e coletar os logs em toda a infraestrutura da plataforma. Um stack de tecnologia popular que fornece visualização, armazenamento e agregação de logs é o EFK (ElasticSearch, Fluent.d e Kibana). Logs de container, assim como logs de componentes da plataforma, são coletados e transmitidos para um armazenamento de dados distribuído. Os dados são apresentados com o Kibana, que tem uma interface de usuário altamente configurável, com recursos que incluem a criação de painéis e a investigação de comportamentos individuais do pod.

## RASTREAMENTO DISTRIBUÍDO

Com o rastreamento, é possível acompanhar uma transação à medida que ela se propaga em um sistema distribuído. Assim, uma implementação de rastreamento deve ser capaz de unir as informações sobre a transação, usando dados coletados a partir de diversos componentes do sistema.

Muitas vezes, as aplicações baseadas em container são implantadas como vários componentes que funcionam juntos como um sistema. Já as aplicações que usam um conjunto de serviços modulares e implantáveis separadamente na arquitetura seguem uma tendência chamada de microsserviços. Seguir os rígidos padrões dos microsserviços ao desenvolver aplicações para implantação em uma plataforma de container é totalmente opcional. No entanto, é importante lembrar que essas aplicações costumam ser formadas por vários containers. O rastreamento de transações por meio de uma arquitetura decomposta e/ou altamente distribuída é desafiador. Isso acontece porque as fontes dos dados de rastreamento estão espalhadas em todo o pool de infraestrutura. Para resolver essa questão, há diversas soluções open source para o rastreamento de sistemas distribuídos que estão em alta no setor.

O Zipkin, por exemplo, é um sistema de rastreamento open source que foi muito adotado nos últimos anos e tornou-se conhecido por ser parte do ecossistema de estrutura do Spring. Outras organizações, como a Uber, desenvolveram novas implementações de ferramentas de rastreamento distribuído.

A OpenTracing Initiative foi fundada para criar um padrão de rastreamento independente de fornecedor. O objetivo era resolver a questão de uma implementação de estrutura de rastreamento que poderia resultar no acoplamento do sistema. As especificações e convenções semânticas descritas pela OpenTracing Initiative são altamente inspiradas no Zipkin. Usado pela Uber, o Jaeger substitui vários componentes do Zipkin e adere às convenções OpenTracing. Isso preserva a habilidade de alternar entre implementações em conformidade com a OpenTracing de forma eficiente.

As soluções que estão em conformidade com os padrões OpenTracing viabilizam a flexibilidade de arquitetura no futuro. Algumas das soluções mais eficazes e adotadas, como Zipkin e Jaeger, são opções com disponibilidade imediata.

## MÉTRICAS

Na coleta efetiva de métricas de aplicação em ambientes de container, há muitos dos mesmos desafios descritos nas seções de registro e rastreamento. Devido à natureza temporária dos containers, os endpoints de métricas não são estáticos. As instâncias de container acabam sendo substituídas por outras mais modernas que podem ser inicializadas em um nó diferente no cluster. Plataformas como Kubernetes e OpenShift usam uma abstração de rede chamada de serviço que define um conjunto lógico de pods e uma política para acessá-los. No entanto, um serviço Kubernetes não fornece um endpoint de monitoramento eficaz, porque são necessárias estatísticas mais granulares sobre containers separados, em vez de agrupados.

Nessa área, o Prometheus é um kit de ferramentas open source para alerta e monitoramento muito usado no contexto dos microsserviços. Ele inclui componentes para coletar e exibir métricas, além de várias opções de bibliotecas de instrumentação, incluindo o exportador Java Management Extension (JMX),

que é o padrão para monitoramento de aplicações Java™. Com agentes de métricas como Jolokia e Prometheus JMX Exporter, o usuário tem uma visualização agregada das métricas JMX de um conjunto de containers por meio da API do Kubernetes.

É recomendável expor as métricas de todos os componentes da aplicação em um ambiente de container. Use bibliotecas de instrumentação famosas para publicar endpoints de métricas que estejam em conformidade com os formatos de exibição do Prometheus. Assim como no caso do domínio de rastreamento distribuído, a tendência no monitoramento e nas métricas de aplicação parece estar se voltando para adoção de uma especificação muito aceita, sendo o Prometheus o padrão líder do setor no momento.

## VERIFICAÇÕES DE INTEGRIDADE

A instrumentação de aplicações em containers com endpoints de verificação de integridade é crítica nas arquiteturas de autocorreção. As aplicações implantadas na infraestrutura tradicional costumam ter endereços de rede estáticos. Muitas vezes, o monitoramento tradicional de aplicações aproveita a previsibilidade de um endpoint de host ou de serviço. Como os containers são programados para execução dinâmica nos nós, os serviços de monitoramento de integridade precisam acompanhar todas as instâncias deles.

O Kubernetes fornece um recurso que monitora automaticamente os endpoints de verificação de integridade e que lida com os containers instáveis. Todos os containers no cluster são monitorados, e o Kubernetes responde às verificações de integridade implantando, excluindo e reiniciando os pods. A aplicação precisa expor uma API que monitore sua integridade, o que requer pouca instrumentação.

A instrumentação da verificação de integridade proporciona diversos níveis de eficácia. Por exemplo, uma aplicação pode expor um controlador isolado que retorna um código de resposta HTTP 200 ao ser chamado. Essa verificação de integridade é útil em muitos casos, mas é capaz de detectar apenas determinados tipos de problemas. Se a conexão de uma aplicação com o banco de dados não for íntegra, uma verificação superficial no endpoint provavelmente não detectará esse problema.

As verificações de integridade mais eficazes fazem o inventário detalhado do status de todos os componentes e conexões críticos para a aplicação. Nesse campo, o Spring Boot Actuator é uma biblioteca de verificação de integridade popular que analisa os contextos das aplicações Spring e apura o status do ambiente de execução de cada componente encontrado. Esse tipo de verificação de integridade profunda viabilizada pelo Spring Boot Actuator é altamente recomendado para as aplicações em containers.

## RESUMO

A migração das aplicações tradicionais tentará acompanhar o ritmo cada vez mais acelerado da adoção de containers. Ter em mãos um conjunto de ferramentas para solucionar as dificuldades mais comuns da containerização significa otimizar a migração de aplicações legadas e o desenvolvimento inovador, visando uma implementação adaptável e eficaz.

Em muitos casos, as implementações de aplicações são eficazes sem mudanças, com a implantação em uma plataforma de container ou infraestrutura tradicional. No entanto, o que possibilita o gerenciamento eficiente de aplicações formadas por muitos containers e implantadas em um pool de infraestruturas são as tecnologias que expõem informações sobre ambientes de execução e proporcionam a visualização e resposta a condições do ambiente.

Mais informações sobre as tecnologias de instrumentação descritas neste whitepaper estão disponíveis através dos recursos a seguir:

- **Spring Cloud Config Kubernetes** para a configuração exteriorizada de aplicações (https://github.com/spring-cloud-incubator/spring-cloud-kubernetes)
- **Logback** para o gerenciamento de logs de aplicações (https://logback.qos.ch)
- **Zipkin** (https://zipkin.io) e **Jaeger** (http://jaegertracing.io) para rastreamento distribuído usando o padrão **OpenTracing** (http://opentracing.io)
- **Prometheus** (https://prometheus.io) como kit de ferramentas de métricas, incluindo o **Prometheus JMX Exporter** (https://github.com/prometheus/jmx_exporter) para publicação
- **Spring Boot Actuator** (https://spring.io/guides/gs/actuator-service) para expor endpoints de verificação de integridade profunda



AMÉRICA LATINA
+54 11 4329 7300
latammktg@redhat.com

BRASIL
+55 11 3629 6000
marketing-br@redhat.com

**facebook.com/redhatinc**
**@redhat**
**linkedin.com/company/red-hat**

**br.redhat.com**
**#f11943_0418**