Escalabilidade de Cluster Docker Baseado em Recursos do Sistema

David J M Cavalcanti and Delano H Oliveira

Centro de Informática – CIn, Universidade Federal de Pernambuco (UFPE)
Pernambuco, PE 50740-560 Brasil
{djmc, dho}@cin.ufpe.br

1 Introdução

O Docker[1] é uma ferramenta open source que provê uma maneira rápida para construir, implantar e executar aplicações utilizando containers. Container é o ambiente que permite aos desenvolvedores virtualizar suas aplicações junto com todas as partes que precisa, como bibliotecas e outras dependências, e fornecer tudo como um único pacote. Ao fazer isso, é possível executar as aplicações em qualquer máquina que tenha o docker instalado.

Além das características mencionadas, o Docker tem outro recurso interessante, que é a capacidade de reunir um grupo de máquinas e transformá-las em um cluster docker. Quando em um cluster, cada máquina docker recebe o nome de nó. Assim, as aplicacões não precisam ser carregadas e executadas em uma única máquina, mas também em vários de nós que trabalham de forma conjunta e coordenada. Desta forma, o cluster docker provê recursos de balanceamento de carga e escalabilidade para as aplicacões entre os nós.

Balanceamento de carga é a capacidade do cluster docker de dividir a quantidade de trabalho para executar uma aplicação entre os nós que compõe o cluster. Enquanto que, escalabilidade é a habilidade do cluster de modificar sua capacidade de recursos dinamicamente, em resposta a mudancas de aumento/diminuicão de carga de trabalho em seu ambiente de execucão. Geralmente, a escalabilidade é feita em tempo de execucão e, é baseada no uso de recursos computacionais, tais como, uso de processador e memória. Por exemplo, um servico pode está demandando muito recurso de processamento em um container, em resposta o cluster escalar criando e adicionando um novo container para o servico. Além disso, a escalabilidade é feita através de ferramentas que auxiliam no gerenciamento de cluster, como o Rancher[2]. No entanto, a escalabilidade é ativada de forma não automática, ou seja, através da acão humana, ou seja, utilizador do cluster, que através de algum servico de monitoramento, observar que os recursos estão se esgotando, executa a escalabilidade, a partir de um uma ferramenta de gerenciamento de cluster.

Neste contexto, este trabalho propõe um mecanismo de escalabilidade dinâmico e automático, baseado em recursos computacionais. Dinâmico significa que o

[1] https://www.docker.com/
[2] https://rancher.com/

escalonamento é feito em tempo de execução, sem a necessidade de parar o cluster e automático significa que ausência de intervenção humana durante o escalonamento. Embora, o utilizador precise definir a métrica que determina a escalabilidade, todo o processo de escalonamento é feito pelo mecanismo em conjunto com a ferramenta de gerenciamento de container.

2 Gerência de Configuração e Ambiente

Antes de apresentar o mecanismo proposto, esta secão introduz alguns conceitos básicos para um melhor entendimento do trabalho.

2.1 Gerenciamento de Cluster Docker

Rancher - A definição oficial é que o Rancher é um software para entrega de Kubernetes como um serviço . Mas na prática esse software é uma plataforma que permite o gerenciamento de infraestruturas de cluster em diversos ambientes e com diferentes orquestradores, tais como: Cattle, Kubernetes e Docker Swarm. O Rancher oferece um serviço web que permite todo gerenciamento através da interface gráfica. Ainda possui um conjunto de templates de serviços para fazer o deploy com poucas configurações.

2.2 Monitoramento de Recursos Computacionais

Como dito anteriormente, o mecanismo escala o cluster baseado no consumo de recursos computacionais. Para isso, o utilizador determina valores básicos para as métricas, que caso esses valores sejam satisfeitos, o escalonador deve agir. Por exemplo, o utilizador pode informar que se o consumo do processador do container for acima de 50% então deve-se escalonar. Neste contexto, faz-se necessário o monitoramento contínuo dos recursos computacionais do cluter. Para implementar o monitoramento, foi utilizado um conjunto de ferramentas, semelhante ao apresentado por [1]. A seguir apresentamos as ferramentas utilizadas:

cAdvisor - para obter as informações do uso de recursos computacionais dos containers que estão em execução no cluster foi utilizado o cAdvisor[3]. Ele é um programa que coleta, processa e exporta as informações do uso de recursos e das características de desempenho dos containers. Ele mantém os parâmetros de recursos, histórico de uso e estatísticas. Esses dados são exportados para cada container e para toda a máquina.

InfluxDB - O Influxdb[4] é um data series database que permite o armazenamento de dados em ordem cronológica, muito usado para armazenar dados de monitoramento.

[3] https://github.com/google/cadvisor
[4] https://www.influxdata.com/time-series-platform/influxdb/

Grafana - O Grafana[5] é uma plataforma aberta para análise e monitoramento. O seu diferencial é a facilidade na criação de gráficos e dashboards usando uma interface gráfica fora a possibilidade de usar diferentes backends para disponibilizar os dados em tempo real. Também é possível criar alertas para múltiplas situações permitindo inclusive realizar alguma tomada de decisão com o uso de webhooks.

3 Características da aplicação

Neste trabalho foi desenvolvido um mecanismo que realiza a ligação da ferramenta de monitoramento com o software de gerenciamento do cluster de forma automática simplificando o gerenciamento de escalabilidade dos serviços. O projeto foi chamado de autoscaling-rancher-grafana[6], escrito em Node.js[7] por permitir agilidade no desenvolvimento de serviços web.

O autoscaling-rancher-grafana possui três funcionalidades básicas:

1. Descobrir serviços para escalabilidade
2. Configurar ferramentas
3. Promover escalabilidade

3.1 Descobrir serviços para escalabilidade

A primeira tarefa é filtrar os serviços que foram marcados para serem escaláveis. Para fazer essa marcação é necessário adicionar um texto em formato JSON[8] com chaves pré-definidas. Esse método foi escolhido pois a descrição é o único campo que pode ser editado a qualquer momento no Rancher. No listing 1.1 apresentamos um exemplo de uma descrição em JSON com as chaves que são consideradas.

Listing 1.1. Exemplo de marcação de serviço com descrição em JSON

```
{scale.up: <instancias>, scale.up.cpu: <threshould>}
{scale.up: 1, scale.up.cpu: 5000}
```

A chave scape.up define a quantidade de instâncias que serão criadas ao atingir o critério de escalabilidade. E a chave scale.up.cpu define o threshould da média de utilização de CPU em hertz no último minuto. Essa periodicidade pode ser alterada mudando valores do template do Dashboard do Grafana que é o arquivo PanelTemplate.json. Deve ser alterado os valores das chaves alert.conditions[0].query.params[1], alert.frequency e targets[0].select[0][2].params[0].

[5] https://grafana.com/
[6] https://github.com/delanohelio/autoscaling-rancher-grafana
[7] https://nodejs.org/en/
[8] https://www.json.org/

3.2 Configurar ferramentas

Esta funcionalidade verifica e cria os componentes necessário em cada ferramenta para realizar a escalabilidade. Nesse caso são criados Alerts no Grafana e Webhooks no Rancher. Mas antes da criação desses componentes são criados dois componentes que dão suporte ao Alert que são o Dashboard e o Alert Channel, permitindo assim a criação de Alerts no Dashboard usando um mesmo Alert Channel, esse último apontando para um endpoint desse serviço ao qual será abordado no póximo tópico.

Com o Grafana preparado, para cada serviço que é sinalizado para a escalabilidade é verificado se há um Alert para esse serviço, se não houver ele é criado. Do mesmo modo é feito para o Webhook no Rancher. Ao fazer isso, essas informações são armazenadas em variáveis para acesso posterior.

3.3 Promover escalabilidade

Uma vez os Alerts criados e acionados, ele faz uma requisição para o Webhook configurado no Alert Channel. Isso indica que teríamos que ter um Alert Channel para cada Alert pois cada um desses Alert se refere a um serviço. Diante disso, criamos um redirecionador de Alerts, a ideia é ter um único Webhook que faz a requisição para os demais.

Assim foi criado um endpoint que recebe todas as requisições dos Alerts, identifica o serviço que enviou a requisição, recupera a informação do Webhook do Rancher ao qual se refere o serviço e finalmente faz uma requisição para esse Webhook. A escalabilidade de fato é feita pelo Rancher ao receber uma requisição nos Webhooks.

3.4 Visão de Uso

Para usar o serviço você precisa acessar os endpoints:

- Retorna as variáveis salvas no serviço

```
/
```

- Acessa o Rancher e o Grafana e extrai os serviços com descrição no formato definido, alertas cadastrados e webhooks cadastrados. Além disso ele faz o setup de Dashboard, Alerts e Webhooks quando não existe. Faz a atualização de todos as variáveis.

```
/update
```

- Retorna os serviços extraidos e salvos no serviço

```
/service
```

- Extrai e atualiza os serviços marcados

```
/service/update
```

– Retorna os webhooks extraidos e salvos no serviço

```
GET /webhook
```

– Recebe as requisições dos alertas do Grafana

```
POST /webhook
```

– Extrai e atualiza os webhooks

```
/webhook/update
```

4 Arquitetura

A Figura 1 apresenta a arquitetura do mecanismo.

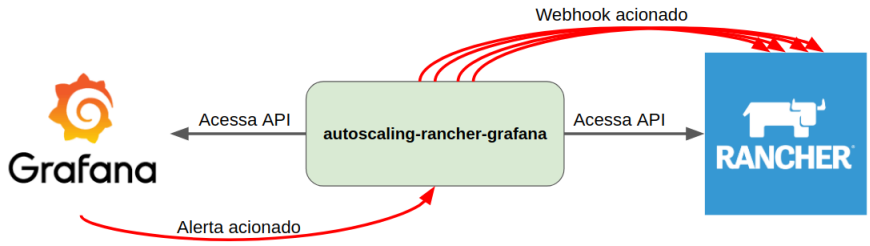

Fig. 1. Arquitetura do Mecanismo de Escalabilidade

O autoscaling-racher-grafana usa as APIs das ferramentas Grafana e Rancher para recuperar informações e criar componentes. Além disso, o Grafana é configurado para enviar as requisições dos alerta para o autoscaling-rancher-grafana que por sua vez faz uma requisição para o Webhook apropiado.

5 Visão de Implantação

Para o funcionamento desse mecanismo de escalabilidade é necessário realizar um passo-a-passo para a instalação de todas as ferramentas e serviços necessários.

1. Instalar o Rancher (você pode usar o tutorial oficial do site)
2. Instalar Prometheus + Grafana + Influxdb (disponibilizamos um tutorial no Github)
3. Criar Keys de acesso no Rancher
4. Editar o arquivo index.js e atualizar os valores de URL_API_RANCHER, CONFIG, APIKEYS. A primeira com o domínio de acesso do Rancher, a segunda com o domínio de acesso do autoscaling-rancher-grafana e a última com os dados de acesso da API do Rancher

5. Criar TOKEN de acesso no Grafana
6. Editar o arquivo grafana.js e atualizar os valores de URL_API_GRAFANA e TOKEN. O primeiro com o domínio de acesso do Grafana e o segundo com o TOKEN de acesso da API do Grafana
7. Marcar os seviços que monitorar e escalar, conforme explicado na seção 3.
8. Executar o autoscaling-rancher-grafana e acessar o endpoint /update

6 Revisão do Projeto

O projeto foi elaborado para realizar entregas parciais semanais e dividido em duas grandes partes: Monitoramento e Scalabilidade de Serviços. A cada semana era atribuído de uma a duas tarefas de cada parte do sistema a um componente do projeto.

Os principais desafios encontrados foram desconhecimento em Docker, Monitoramento de Cluster, limitações das ferramentas ao usar em máquinas virtuais, configuração das ferramentas. Decidimos usar o Rancher para simplificar a criação e gerenciamento do cluster mas mesmo assim tivemos dificuldades na instalação das ferramentas de monitoramento. Decidimos estudar as diversas formas para configurar as ferramentas de monitoramento e consolidamos um tutorial para configuração do Prometheus, Grafana e Influxdb monitorando o Rancher.

7 Revisão Individual

7.1 David Cavalcanti

O desenvolvimento deste projeto foi bem desafiador, uma vez que apresenta tecnologias novas e com poucas documentações, o que dificulta a sua execução. Contudo, promove uma ótima aprendizagem uma vez que lhe incentiva e obriga a estudar conceitos, que são atualmente temas férteis de pesquisa. A lição aprendida é a necessidade de sempre escutar e acompanhar o desenvolvimento destas novas tecnologias, para que, no surgimento de desafios semelhantes, estes possam ser superados mais facilmente.

7.2 Delano Oliveira

O desafio de desevolver um projeto fora da sua zona de experiências dificulta o gerenciamento de riscos uma vez que você não tem históricos para se basear. Acredito que a principal lição é fazer uma consultoria com alguém que tenha tido a experiência para melhor gestão do escopo do projeto.

References

1. DIEDRICH, Cristiano. Coleta de Métricas Docker Swarm. 2017. Disponível em: <https://www.mundodocker.com.br/coleta-metricas-no-docker-swarm/>. Acesso em: 14 dez. 2018.