Replicação de servidores Memcached com Repcached

Arquitetura master/slave em servidores memcached com libevent

Fabiano Araujo
Universidade La Salle
UNILASALLE
Canoas, Brasil

***Abstract*—TODO**
***Index Terms*—TODO**

I. Introdução

II. Referencial Teórico

A. Tolerância a falhas

B. Servidores cache

C. Arquitetura Master/Slave

III. Implementação

O experimento realizado para demonstração da ferramente se dispôs entre dois *containers* Docker [1], obtidos de uma imagem com todos as bibliotecas necessárias para execução de dois servidores Repcached [2] isolados.

A replicação fica por parte da definição cíclica dos IPs de servidor *master* e *slave*. Por vezes a definição da dependência ficou confusa e foi obtido o entendimento de que o a definição real entre *master* e *slave*, nesta ocasião, refere-se especificamente à escolha do desenvolvedor ou cliente de utilização de um IP.

Utilizando imagens pré definidas dos *containers*, algumas modificações foram realizadas para que pudesse ser exibida de forma coerente a comunicação entre os servidores nos *containers*. Isto porque os exemplos encontrados e fundamentados nas próprias imagens utilizadas como exemplo utilizam do próprio *host* de um *container* Docker com o próprio *container*, não deixando clara as limitações da comunicação entre dois *containers*.

Da imagem *yrobla/docker-repcached* [3], foram retirados os arquivos que criavam a possibilidade de logins com administradores, por não ser relevante ao experimento e, principalmente, foi postergada a inicialização do *daemon* Memcached. Mesmo que o projeto possua um nome diferente, Repcached, o mesmo é uma adaptação do Memcached adicionando apenas um parâmetro para identificação do IP do servidor cujo terá replicação dos dados.

A postergação da inicialização se fez necessária pelo contexto de utilizar dois *containers* Dockers. Em sua natureza, cada *container* possui um IP local cujo, por padrão, é definido por um dispositivo virtual de rede. Este dispositivo funciona como um DHCP entre *containers* atribuindo os IPs assim que o *container* for iniciado.

No código inicial no entanto era necessário definir o IP do servidor *slave* antes da inicialização. Porém como a inicialização é o evento que é atribuído um IP na rede virtual interna de *containers* Docker, não era possível definir a atribuição, gerando uma falha no procedimento pois o mesmo adotava o IP 127.0.0.1 caso não fosse especificado.

Exposta a porta 11211, padrão do servidor memcached, então, é inicializado o *container* e acessado o mesmo utilizando as diretivas *exec -it –entrypoint /bin/bash* para que se tenha acesso à um *shell* onde é possível iniciar o servidor memcached.

Iniciando os dois *containers* foi então verificado os IPs internos definidos pelo comando *docker network inspect bridge*. Tomando nota dos IPs atribuídos, os serviços memcached foram iniciados em ambos os *containers* apontando o parâmetro *-x* para o IP do *container* oposto. Por isto a atribuição de *master* e *slave* se tornou cíclica.

Tal comportamento permite que caso o *slave* caia e retorne em um momento posterior, o mesmo possua todos os dados de *master* e vice-versa. Diferente de estruturas como Redis que possui *slaves* como apenas leitura e propagação direta para *master*. Da forma como o Repmemcached é organizado ambos possuem a mesma atribuição

Omitir a definição do IP ou tornar a atribuição do IP pelo parâmetro *-x* para 127.0.0.1 inviabiliza o fluxo, sendo necessário reiniciar todo o processo em *master* caso o *slave* caia.

A estrutura utilizada no experimento pode ser visualizada na figura 1.

IV. Resultados

Foi possível realizar a replicação de ambientes Memcached com Repcached com sucesso entre *containers* Docker. Uma vez iniciados os *containers*, foi identificado os IPs dos mesmos e finalmente iniciados os serviços via *nohup*, uma vez que os parâmetros estavam melhor dispostos se explicitamente inseridos

Foram nomeados cache1 e cache2 respectivamente e tratada a conectividade de uma aplicação PHP com o cache2, sendo este ponto de entrada de obtenção e escrita da aplicação. Em termos de utilização, pode-se afirmar que a aplicação interagiu diretamente apenas com o cache2 e que devida

a funcionalidade de replicação, o cache1 foi alimentado de forma semelhante.

No momento do ensaio de uma queda, para que a aplicação pudesse continuar a consumir de um servidor Memcached, foram atribuídos dois IPs de servidores com pesos distintos. O peso do servidor, no contexto de aplicação PHP, segundo a sua documentação define a prioridade com que a conectividade irá acontecer, assim o servidor com peso maior terá prioridade e uma vez que não for possível conectar, tentara o segundo mais próximo [**?**]. Não é definida a forma de utilização caso os servidores tiverem o mesmo peso.

Em código PHP, é possível definir servidores de replicação dos dados, porém nada condiz com a replicação de informação integral que o Repcached se propõe. Das diretivas do PHP, o que se tem de replicação é a replicação imediata da chave salva pelo próprio script PHP, porém chaves criadas por terceiros não serão replicadas visto que não comunicação entre os servidores.

A replicação em código PHP também não permite que os dados armazenados sejam recuperados em caso de queda de um dos servidores, assim servindo apenas para a replicação em servidores com finalidades distintas ou que possuam replicação não conjunta, isto é, entre outros dois servidores que não se comunicam diretamente.

A aplicação contou com um endpoint em AJAX que demonstrava a informação armazenada diretamente por *telnet*, utilizando das diretivas, uma vez estabelecida conexão com Memcached de *get* e *set*.

Em outro experimento, foi realizada a conexão entre três Dockers, no intuito de aumenta a complexidade cíclica para três servidores. O experimento no entanto falhou pelo motivo de funcionalidade da biblioteca libevent. Dado o ambiente como demonstrado na figura 2, uma entrada no servidor *slave 1* não estaria armazenado do *slave 2*, mas apenas no *master*, pois o *slave 2* está tanto recebendo a informação como enviando para outro servidor que não se comunica diretamente com o ele, no caso, *master*.

Tal conclusão inviabiliza a utilização de Repmemcache em ambientes que demandam grande complexidade de tolerância a falhas ou que necessitem de um controle de alta disponibilidade visto que conforme seus autores, o Memcached não foi construído como um servidor de alta disponibilidade.

V. Conclusão

Por meios padrão, não é possível obter a replicação do ambiente Memcached, visto que o mesmo não possui instruções nem foi moldado para tal. Sua natureza de armazenamento de dados apenas em RAM também não prevê qualquer persistência de dados físicos que pudessem justificar a adaptação de outro serviço como leitura dos seus dados.

Assim o uso do Repcached permite a criação de um servidor *master* e um *slave* diretamente conectados, mas não permite a criação de *clusters* de informação. A replicação também não acontece em termos de processamento mas apenas de indicação de lugar na memória em outro endereço de rede além do servidor *master*, não configurando assim o comportamento de *cluster*.

A definição de que o Memcached não possui alta disponibilidade impacta diretamente no consumo atual da informação, dada a vasta capacidade tecnológica que outra ferramentas tem neste assunto, como Redis [4]. Cabe aos desenvolvedores e analistas identificarem a utilização do Memcached detalhadamente para evitar perigos de inviabilidade de comunicação por parte de falta de replicação.

Referências

[1] Docker. Docker. [Accessed: 15- Nov- 2018]. [Online]. Available: https://www.docker.com/

[2] K. Inc. Repcached. [Accessed: 15- Nov- 2018]. [Online]. Available: http://repcached.lab.klab.org/

[3] Yrobla. Base docker image to run a repcached server. [Accessed: 10- Nov- 2018]. [Online]. Available: https://github.com/yrobla/docker-repcached

[4] Redislabs. Redis. [Accessed: 17- Nov- 2018]. [Online]. Available: https://redis.io/