Ciências ULisboa | Informática

Projeto de Tecnologias de Redes

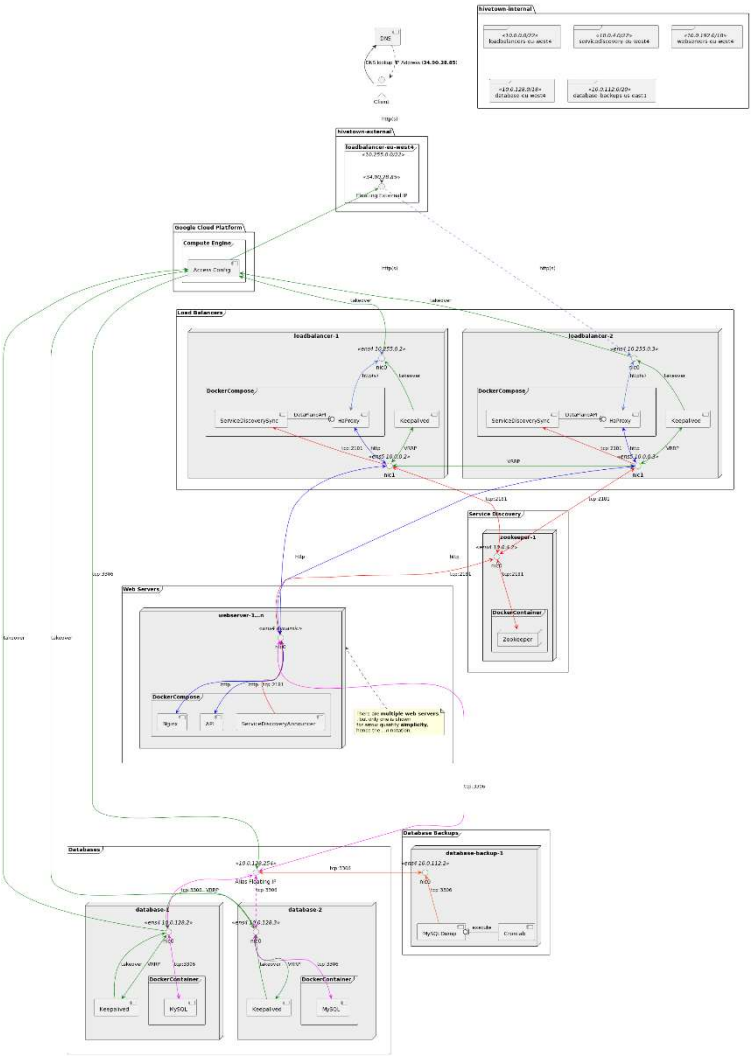
Relatório de Situação do Projeto
(2ª Avaliação Periódica)

---

1 Grupo 04

	Nome	Número
1	Bruno Gonzalez	56941
2	Lucas Pinto	56926
3	Madalena Rodrigues	55853
4	Matilde Silva	56895
5	Pedro Almeida	56897
6	Rómulo Nogueira	56935

2 Implementação da arquitetura distribuída



O ponto de entrada no serviço é dado pelo registo A do DNS, que aponta para o IP 34.90.28.85. Esse endereço é flutuante, ou seja, não é fixo, "saltando" entre instâncias. A instância que detém esse endereço é chamado balanceador de carga ativo, descrito em maior detalhe no ponto 2.1.
Ao chegarem pedidos ao balanceador de carga, este, pela sua lista de servidores *backend*, distribui-os com a estratégia *round robin*, ou seja, sequencialmente.
Para segmentar e proteger a rede interna, os balanceadores de carga possuem uma interface de rede na rede hivetown-external (rede que inclui endereços externos/da Internet) e uma interface interna na rede hivetown (rede interna, sem acesso do exterior). Assim, considera-se que a rede interna seja protegida, pois a única forma de entrada é através dos balanceadores de carga.
De modo a que os balanceadores conheçam os webservers disponíveis, estes possuem um *script* que sincroniza a configuração do *HaProxy* com a informação disponibilizada pelo servicediscovery (Zookeeper). O Zookeeper tem a função de centralizar as máquinas existentes que são balanceadas (os servidores web do frontend e da API REST). De forma a que o Zookeeper conheça os servidores web disponíveis, estes devem se fazer conhecer, ao criar um znode efémero com o IP e porta do serviço que oferecem, seja no /api-servers ou no /web-servers.
Os webservers desfrutam de uma base de dados ativa, exposta por um IP flutuante interno, descrito mais detalhadamente no ponto 2.3. Fazem uso também de serviços externos, para mapas, autenticação e pagamentos.

2.1 Implementação dos balanceadores de carga

Os balanceadores de carga estão implementados seguindo uma arquitetura ativa-passiva, onde uma instância responde a todos os pedidos e, quando falta, a instância passiva toma controlo (faz *takeover*) e passa a ser a ativa. Quando a instância ativa original retorna de boa saúde, esta tem maior prioridade, portanto assume novamente o papel de ativa e a outra retorna a passiva.
A implementação desta arquitetura assenta no keepalived, que anuncia mensagens VRRP, sendo que quando a instância passiva deteta uma falta na receção dessas mensagens, toma controlo da operação. Essa tomada de posse está definida num *script* que remove o IP Externo flutuante da outra instância e o atribui a si própria. Como esta já estava pronta a ser usada e apenas aguardava entrar em ação, a tomada de posse é simples, já que apenas troca de IP.
Como referido na introdução da implementação da arquitetura, existe um script em cada balanceador para assegurar sincronismo entre o sistema de descoberta de serviços (servicediscovery – zookeeper) e a configuração do balanceador. Este script foi desenvolvido em *python* e faz *watch* aos *znodes api-servers* e *web-servers*. Quando há alguma alteração, esta é processada e altera-se a configuração do HaProxy através da sua Data Plane API.

2.2 Implementação dos web servers

Os servidores web possuem também um *script* relativo à descoberta de serviços, com a finalidade de anunciarem a sua presença e disponibilidade de resposta a pedidos web ou api. Neste *script*, é criado o referido *znode* efémero, e o *script* é mantido a executar durante a vida inteira do servidor. Isso deve-se à remoção dos nós efémeros do Zookeeper quando a inatividade é superior a 30s, o que garante que quando o webserver sofre uma falta, o Zookeeper irá perceber e removê-lo da sua lista, que como consequência notifica os *scripts* de sincronização dos balanceadores com a informação de que esse webserver já não responde a pedidos.
A implementação nuclear do servidor web assenta, para além da descoberta de serviços, em dois *docker containers*, que servem o servidor web do frontend e o servidor web da API.
O primeiro é um servidor que disponibiliza os ficheiros compilados (HTML/JS/CSS) da interface do cliente, portanto o seu propósito é que esses ficheiros sejam transferidos pelo cliente e aqui termina a sua interação.
O último é executado do lado do servidor e interage com a base de dados ativa e com serviços externos.

2.3 Implementação das bases de dados

Foram criadas duas instâncias no GCP para as bases de dados com os nomes vm-database-1 e vm-database-2, e ainda uma que funciona como servidor de backups, vm-database-backup.
Primeiramente, foi instalado o Docker nas 3 máquinas com o objetivo de uniformizar a configuração dos 3 servidores.
Tanto a vm-database-1 como a vm-database-2 podem desempenhar o papel de "master" ou de "slave" e por isso a estrutura de ficheiros criada foi idêntica em ambas:

- Uma pasta master: que contém os ficheiros necessários à criação e configuração de um Docker container que acolhe um servidor MySQL com funções de Master.
- Uma pasta slave: que contém os ficheiros necessários à criação e configuração de um Docker container que acolhe um servidor MySQL com funções de Slave (que realiza a replicação dos dados da Base de Dados Master)
- Uma pasta keepalived: com as configurações para lidar com a tolerância a faltas (explicada na secção 4)
- Um conjunto de Bash scripts:
    - backup.sh: realiza um backup local;
    - init_master.sh: apenas presente da vm-database-1 com o propósito de iniciar o servidor MySQL como master;
    - initSlave.sh: apenas presente na vm-database-2 com o propósito iniciar o servidor MySQL como slave;
    - newMaster.sh: realiza a passagem de um servidor MySQL slave para um master
    - newSlave.sh: realiza a passagem do antigo servidor MySQL Master (quando este volta, depois de cair) para slave.
    - verify.sh: detalhado na secção da tolerância a faltas

A replicação é feita no modo ativo-passivo, em que o servidor designado como Master é responsável pela gravação dos dados (inicialmente a vm-database-1) enquanto o servidor Slave (inicialmente vm-database-2) mantém cópias sincronizadas do servidor Master.

O servidor de backups está configurado para fazer backups todas as noites às 02h da manhã, hora provável de reduzida atividade no servidor.
Esta automatização foi feita através do crontab que executa o script backup.sh.

3 Escalabilidade

Por escalar o núcleo do serviço, os webservers foram implementados seguindo uma arquitetura ativa-ativa que escala de forma horizontal (quantidade de máquinas versus vertical (qualidade de máquinas)), criando a necessidade para os balanceadores de carga.
Para escalar para cima, ou seja adicionar mais nós, são iniciadas ou criadas máquinas partindo duma imagem de máquina do webserver esqueleto, permitindo que seja somente necessária a execução de uma linha de comandos no terminal.
O mesmo se aplica ao escalar para baixo, removendo nós, em que basta parar ou eliminar uma máquina dos webservers, também com apenas uma linha de comandos no terminal.
Em grande parte, isto é possível pela descoberta de serviços mencionada anteriormente: quando um webserver se torna disponível, este anuncia-se ao zookeeper que notifica os balanceadores de carga. Pelo contrário, quando é escalado para baixo ou sofre uma falta, o mecanismo de saúde do HaProxy nota que o servidor já não responde e deixa de lhe enviar pedidos, mas aguarda ainda que o *script* de sincronização com o zookeeper tome a ação de alterar a configuração, removendo assim o webserver de forma definitiva. A última parte é importante, visto que a cada 1s, o HaProxy envia um pedido de saúde a cada webserver na sua configuração. Com poucos servidores não existe qualquer problema, mas após algum tempo, com o aparecimento e desaparecimento de centenas ou até milhares de servidores, torna-se um desperdício de recursos e pode ainda afetar o serviço. Em suma, quando o servidor deixa de servir o seu propósito, é removido. Se no futuro contribuir para o serviço da mesma forma, será novamente adicionado.

4 Tolerância a Faltas

Fundamentalmente, o sistema atinge tolerância a faltas por redundância.
Assim, nenhum componente está sozinho e tem sempre pelo menos uma instância que o pode substituir em caso de falta.
Nos balanceadores de carga, por uma arquitetura ativa-passiva, o nó ativo é responsável por toda a operação desse tipo, enquanto o passivo aguarda que o primeiro falte para o substituir.
Nos webservers, por uma arquitetura ativa-ativa, todos os nós são iguais. É por isto que existe a necessidade do balanceamento de carga. Havendo pelo menos duas instâncias, com a possibilidade de escalar para cima, mas também para baixo, o componente é redundante e por consequência tolerante a faltas.
Nas bases de dados, assim como nos balanceadores de carga, (mas com diferenças críticas) quando o ativo sofre uma falta, o passivo entra em ação.
Relativamente à descoberta de serviços, o Zookeeper não é tolerante a faltas nem redundante neste momento, pois é considerado uma instância protegida. No futuro, se existir a possibilidade devido a tempo disponível não requirido por outras tarefas do projeto, este componente será tornado redundante e assim tolerante a faltas. Para tal, seriam necessárias mais duas instâncias *follower* da instância líder.

Bases de Dados

Para além dos ficheiros descritos na secção 2.3, ambas as instâncias têm dois ficheiros que são executados para lidar com falhas.
Quando o master falha o slave fica a saber pela comunicação VRRP estabelecida através do keepalived e executa um script takeover.sh, responsável por fazer a troca do docker container para master de forma a possibilitar escritas, backups e eventualmente a conexão de outro servidor para replicação.
Quando o "antigo master" voltar a ficar ativo, é executado ao iniciar o script verify.sh, que tem a função de verificar se o servidor foi abaixo quando era master ou slave. Caso tenha acontecido quando era master, este executa a mudança para slave e fica automaticamente atualizado com o "novo master" e pronto a replicar. Caso tenha acontecido quando era slave, apenas reinicia o docker container.
Estas trocas de master para slave também implicam a troca de um alias de IP interno, para possibilitar que os webserveres se conectem à base de dados master (quer a execução decorra na vm-database-1 ou na vm-database-2).

Exemplo:
Contexto: vm-database-1 iniciada como master e vm-database-2 iniciada como slave.
Alias IP do master: 10.0.128.10

T0:

A instância vm-database-1 cai
A instância vm-database 2 recebe a informação e executa o takeover.sh que:

- Retira o alias IP da vm-database-1;
- Coloca o alias IP em si mesmo;
- Altera as configurações do keepalived para Master;
- Faz a troca do docker container mysql de slave para master;
- Fica pronta para receber pedidos.

Tx:

A instância vm-database-1 volta a estar operacional e executa o script verify.sh assim que é ligada:

- Altera as configurações do keepalived para Backup;
- Inícia o docker container como slave;
- Pronta a replicar.