Ciências ULisboa | Informática

Projeto de Tecnologias de Redes

Relatório de Situação do Projeto

(1ª 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

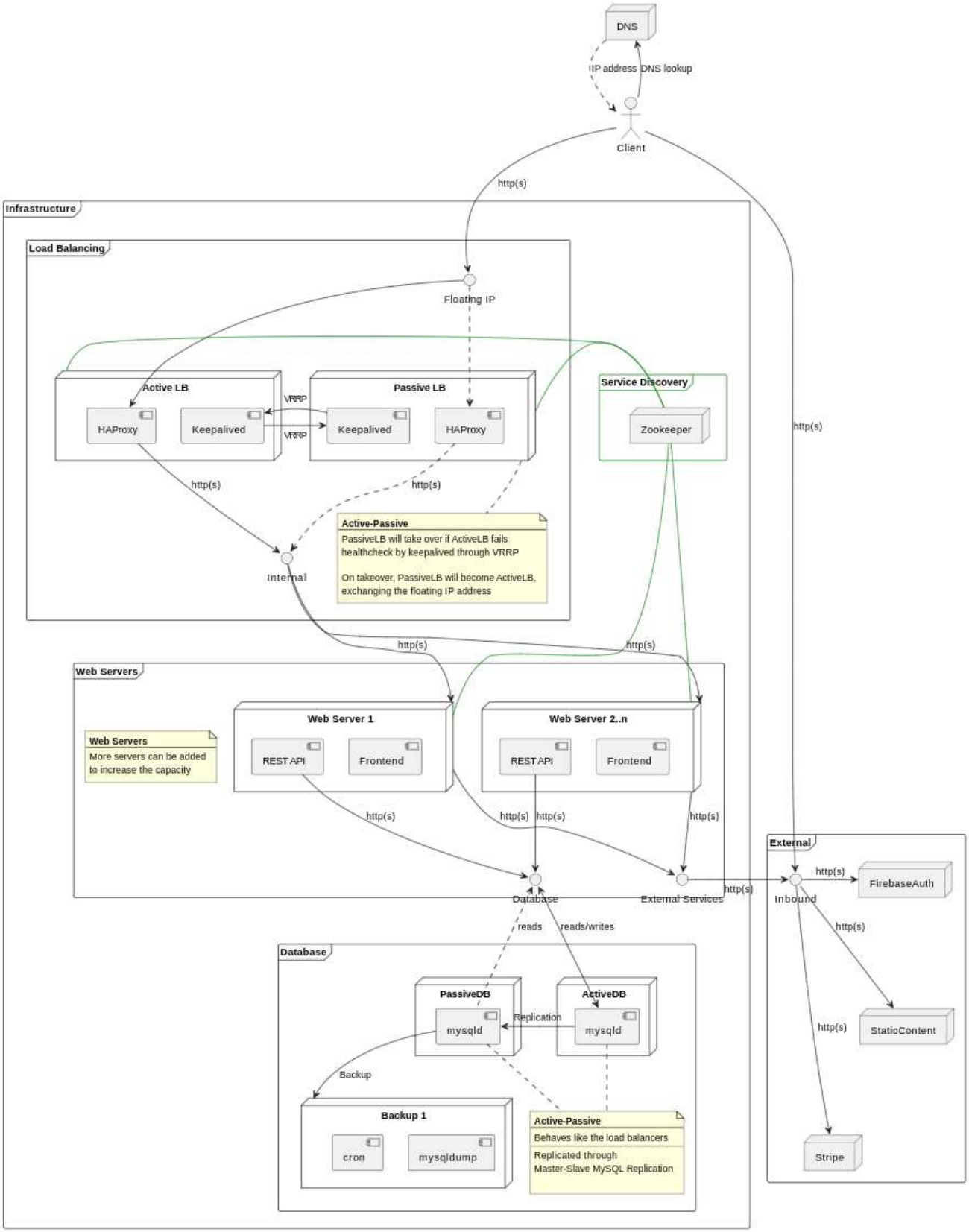
2 Arquitetura

O projeto está dividido em 3 grandes secções: o sistema, os clientes, e os serviços externos.

O primeiro está depois repartido em quatro componentes: balanceadores de carga, que distribuem a carga entre os servidores web, os servidores web, que são o núcleo da aplicação pois é onde está a maior parte da camada de negócio, a descoberta de serviços que permite aos balanceadores de carga identificarem os servidores web disponíveis, e por fim as bases de dados, onde estão persistidos os dados que suportam o funcionamento do sistema.

Quanto aos clientes, apenas se nota que desejam interagir com o sistema. Como o sistema faz uso de serviços externos, os clientes estão também impostos a fazer o mesmo.

Já os serviços externos auxiliam o funcionamento do sistema, como num regime de outsourcing, pois seria inviável e custoso implementar as componentes externas. Essas componentes dominam a autenticação, os ficheiros estáticos, e os pagamentos.


DNS
IP address
DNS lookup
Client
http(s)
Infrastructure
Load Balancing
Floating IP
Active LB
HAProxy
Keepalived
VRRP
VRRP
Passive LB
Keepalived
HAProxy
Service Discovery
Zookeeper
http(s)
http(s)
http(s)
Active-Passive
PassiveLB will take over if ActiveLB fails healthcheck by keepalived through VRRP
On takeover, PassiveLB will become ActiveLB, exchanging the floating IP address
Internal
http(s)
http(s)
Web Servers
Web Servers
More servers can be added to increase the capacity
Web Server 1
REST API
Frontend
Web Server 2..n
REST API
Frontend
http(s)
http(s)
http(s)
http(s)
External
Database
External Services
http(s)
Inbound
http(s)
FirebaseAuth
http(s)
StaticContent
http(s)
Stripe
reads
reads/writes
Database
PassiveDB
mysqld
Replication
ActiveDB
mysqld
Backup
Backup 1
cron
mysqldump
Active-Passive
Behaves like the load balancers
Replicated through
Master-Slave MySQL Replication

2.1 Design da arquitetura distribuída

O cliente começa por consultar servidores DNS, seguindo este protocolo largamente utilizado: começa pela sua cache local e percorre recursivamente os servidores até encontrar o domínio do serviço: hivetown.pt.

Obtendo o IP, pode então fazer as requisições que necessita, quer seja obter o cliente oficial, o website, quer pedidos à API REST.

Para que o sistema seja escalável, tolerante a faltas, e com desempenho razoável, decidiu-se escalar os servidores web, com 2 ou mais nós, seguindo uma arquitetura ativo-ativo. Para isto é necessário balancear a carga, o que é feito pelo *load balancer*.

O *load balancer* também não pode ser um ponto único de falha, sendo necessário pelo menos duas instâncias disponíveis para responder aos pedidos dos clientes. Seguiu-se, portanto, uma arquitetura ativo-passivo, em que o nó ativo responde a todos os pedidos e o(s) nó(s) passivo(s) servem de salvaguarda. No caso específico do projeto, apenas será implementado um nó ativo e um nó passivo.

De forma a que os balanceadores de carga estejam sincronizados entre si e com as suas tabelas de endereçamento atualizadas, centralizar-se-ão os dados através do *Zookeeper*.

Por fim, os *web servers* necessitam de persistir e obter os dados necessários para prestar o serviço, o que implica bases de dados. As bases de dados, assim como os *load balancers*, estão desenhadas seguindo a arquitetura ativo-passivo. Depois, manter-se-ão *snapshots* numa máquina própria para o efeito.

Já no detalhe da implementação, o sistema está dividido em componentes menores que comunicam entre si através da rede e que podem ser executados em diferentes *hosts*, físicos ou virtuais.

Como o sistema é homogéneo, todos os *hosts* têm as mesmas configurações de hardware. Isto facilita a gestão do sistema e garante que a aplicação pode ser executada de forma consistente em todos os *hosts*. Para que o código aplicacional seja ainda mais abstraído dos *hosts*, será utilizado o *Docker*.

O Docker é uma plataforma de conteinerização que permite aos desenvolvedores empacotar uma aplicação junto das suas dependências e configurações num container leve e portátil. Os *containers* do *Docker* podem ser executados em qualquer *host* com o *Docker* instalado, independentemente do hardware ou sistema operativo utilizado. Esta camada de abstração permite que a equipa se concentre no desenvolvimento e testes da aplicação, abstraindo-se da infraestrutura subjacente.

2.2 Especificar requisitos NF – escalabilidade

De forma a servir qualquer quantidade de utilizadores, é desejável que o sistema seja escalável, isto é, aumentar a capacidade das máquinas individuais (escalar de forma vertical), ou adicionar/remover nós ao sistema (escalar de forma horizontal).

No caso concreto do projeto, o requisito apenas requer escalar os servidores web (ver diagrama de implementação para mais detalhes). Para tal, serão desenvolvidos *scripts* especializados em:

- adicionar um nó ao sistema
    - criar a máquina virtual
    - configurar conforme necessário
    - anunciar ao *ZooKeeper*
    - aguardar e servir pedidos;
- remover um nó do sistema
    - Consiste na eliminação da máquina virtual.
    - Após a eliminação, a desconexão da máquina será detetada pelo *Zookeeper* que removerá a entrada da máquina da sua lista. Depois, os *watchers* nos balanceadores de carga são notificados de modo a remover a máquina desconectada das suas configurações.

Desta forma, com máquinas menos capazes, mas com um sistema homogéneo, o seviço pode ser disponibilizado a um elevado número de utilizadores, facilitando a manutenção do mesmo.

O serviço subjacente usado para o sistema será a *Google Cloud Platform (GCP)*, que permite a criação de máquinas virtuais partindo de imagens doutras máquinas virtuais. Dessa forma, permitirá a configuração de uma máquina virtual de cada tipo (servidor web, balanceador de carga, etc) para que posteriormente seja simples a adição de nós ao sistema, algo que se pretende tornar automatizado.

2.2.1 Docker

Como referido anteriormente, os nós do sistema serão homogéneos, visto que as máquinas virtuais na *GCP* partilham as mesmas características. Para além disso, cada nó irá executar *Docker Containers*, abstraindo mais profundamente aspetos que difiram entre máquinas, garantindo que o código aplicacional será executado da mesma forma independentemente do *host*, cujas características podem vir a mudar no futuro.

A disponibilização das imagens do projeto será efetuada através do *Docker Hub*, integração abordada em mais detalhe na secção “Github - uso com branches individuais e releases”.

Especificando as *Docker Images*, para os balanceadores de carga, será executado um *Docker Container* a partir da imagem desenvolvida especificamente para este componente. Esta é uma extensão da *instantlinux/haproxy-keepalived*, configurada para as necessidades do projeto.

Os *Web Servers (WS)* consistem em dois *Docker Containers*:

1. o primeiro para servir o *front-end*, separada por 2 *build stages*:
    a. Uma *build stage,* apelidada de *builder*, que parte da imagem *node-alpine* para instalar dependências e fazer *transpile* dos ficheiros do projeto (Vue, TS, CSS) em código base da *Web* (HTML, CSS, JS, minificados)
    b. Outra *build stage,* apelidada de *runner*, para realmente disponibilizar os ficheiros, com uso da imagem *nginx-alpine*, aplicando também configurações conforme as necessidades do projeto.
2. o segundo para o *back-end*, também para minimizar o tamanho da imagem final, será separado em 2 *build stages*:
    a. Uma *build stage* também para instalar e transpilar, partindo da imagem *node-alpine*
    b. Outra *build stage* para executar a API *REST*.

Para executar ambos na mesma máquina será utilizado *Docker Compose*. Como o *nodejs* é *single threaded*, o backend pode até ser executado em duas instâncias por máquina para aproveitar melhor o CPU. Isto, contudo, depende das características do *host*.

2.3 Especificar requisitos NF – segurança

No que respeita às *firewalls*, permitirão expor e limitar entradas apenas às estritamente necessárias para o funcionamento do sistema. Por exemplo, as bases de dados não serão expostas à Internet, apenas à rede interna e às máquinas que necessitam acesso. Estas *firewalls* são implementadas pelo *iptables* em cada *Docker Container*, que por defeito possui uma configuração cuidada, mas que pode ser mais bem especificada para se tornar mais restrita. Ainda assim, a *GCP* permite configurar as redes e máquinas para tornar a exposição mais controlada.

No fundo, cada tipo de componente tem a sua subrede. Uma subrede para os balanceadores de carga, uma para a descoberta de serviços, uma para os servidores web, e uma para as bases de dados. Entre estas a comunicação será limitada e controlada, apenas sendo permitida a estritamente necessária. As subredes serão implementadas numa *VPC*, que permite comunicação entre regiões.

Os pedidos ao serviço chegam da *Internet* até ao *Floating IP* que redireciona, como um *NAT*, para o *Load Balancer* ativo. Por isso, os balanceadores estão expostos à Internet, mas sempre por trás de *firewalls* de rede e das próprias máquinas.

O seguinte passo na comunicação é entre os balanceadores e os servidores web, isto numa rede já interna e segura, que apenas aceita comunicação dos balanceadores e apenas comunica com esses, com a descoberta de serviços, com as bases de dados, e com serviços externos. Neste último ponto deve-se frisar que as saídas só são permitidas para os endereços específicos dos serviços externos, e cuja entrada na rede apenas é aceite de conexões já estabelecidas.

Nas bases de dados acontece algo parecido no que concerne o *backup*, porém é possível tirar proveito de funcionalidades da *VPC* para simplificar e ao mesmo tempo tornar a rede mais segura.

Relativamente à comunicação, esta mais abstraída dos detalhes de rede, será obrigatório comunicar por *HTTPS* (*TLS/SSL*). Como referido anteriormente, os balanceadores de carga são de camada 7, o que implica que terão de decifrar os pedidos.

2.4 Especificar requisitos NF - tolerância a faltas

Optou-se por atingir tolerância a faltas através de redundância – disponibilizar mais de um nó para cada elemento do sistema distribuído.

Para as bases de dados, serão configuradas duas máquinas virtuais no modo ativo-passivo em que o servidor ativo é o principal e o passivo é o servidor de salvaguarda em caso de falha do primeiro.

O servidor ativo é responsável por executar todas as operações de leitura e escrita na base de dados. A replicação deste é realizada através da cópia dos dados do servidor ativo para o servidor passivo em tempo real ou próximo, de forma a garantir que os dados estejam sempre atualizados no nó passivo.

Quando o ativo falha, o nó passivo toma controlo (invertendo-se as suas funções) e assume a responsabilidade de responder aos pedidos. Isto garante que todas as operações que envolvam a base de dados estão sempre disponíveis, mesmo em caso de falha num nó. Uma vez restaurado, o servidor ativo volta a assumir o controlo e o servidor passivo assume novamente o modo de espera.

Também, para salvaguardar dados de múltiplos momentos, são efetuados snapshots por uma máquina dedicada para o efeito. Em caso de perda de dados, necessidade de rever o histórico, ou por qualquer outro motivo, será possível efetuar restauro de dados a partir destes snapshots.

É importante evidenciar que será utilizado uma única máquina de *backup*, uma vez que se considera que a informação atual estará salvaguardada a qualquer momento por estar presente em três máquinas. Tomou-se esta decisão porque a adição de uma outra máquina implicaria um custo superior, sobretudo pelas suas características de elevado tamanho de disco. O backup utilizado estará localizado noutra região, mais barata e com o objetivo de obter resiliência geográfica. Apesar de ter uma latência superior à de um backup na rede local, não é crítica para o funcionamento do sistema.

3 Interoperabilidade

Tendo em conta o requisito de interoperabilidade, foi desenhada a especificação da *API REST* com o intuito de se tornar o principal meio de comunicação entre os clientes oficiais, *browsers* no *website* https://hivetown.pt, e outros que pretendam incorporar o sistema nas suas aplicações. Assim, todas as funcionalidades do negócio presentes no *website* oficial são expostas através desta *API REST*. A especificação pode ser encontrada no repositório do Moodle disponibilizado pelos docentes, e usa o padrão *OpenAPI v3*, podendo ser visualizada através do *Swagger*.

4 Ambiente de desenvolvimento

Todos os artefactos desenvolvidos estarão guardados sob forma de repositórios no *GitHub*, na organização Hivetown.

Quanto aos repositórios, o projeto encontra-se dividido em dois principais: hivetown/backend e hivetown/frontend. Para além destes existem outros para planeamento (diagramas) e configurações dos *load balancers* e bases de dados.

Em todos é adotada uma abordagem que define o ramo *main* como *read-only*, o que obriga os colaboradores a criar *pull requests* para contribuir para o repositório. É requerido que outro colaborador faça *peer review* e aceite o *pull request* caso tudo esteja conforme.

Contudo, visto que os repositórios são privados e fazem parte de uma organização, estão apenas disponíveis aos membros do projeto, e para redução de custos, não se optou por fazer upgrade à conta da organização. Isto torna-se um inconveniente, já que impede a aplicação de regras aos ramos: as regras apenas aparecem como aviso, pelo que ninguém é obrigado pelo sistema a segui-las. Assim, o ramo *main* pode ser modificado por qualquer colaborador com permissão, a não ser que a conta da organização seja na versão *Team* ($4/utilizador/mês) ou *Enterprise* ($21/utilizador/mês).

No que toca às mensagens de *commits*, os membros devem seguir a especificação das *commits* convencionais, algo que é obrigado por um *hook* do *Husky* quando se efetua *commit*. Isto permite que se utilizem automações para criação de *changelogs*, *version bumping* e é uma boa prática, que se traduz num bom entendimento para os restantes membros da equipa. Nas versões será adotado o versionamento semântico *semver*, aplicado em simultâneo na versão da *release* e na versão do *package.json*.

Serão efetuadas *releases* pelas *Github Actions* sempre que um *pull request* for aceite, e por consequência fundido no ramo principal.

Ainda nas automações, é implementada *integração contínua (CI)* através das *Github Actions*, que executa os testes unitários e efetua *linting* ao projeto para garantir que está de acordo com o padrão definido. Para além disso, está também definida a implementação contínua (CD), que publicará as *Docker Images* no *Docker Hub*. Usando, por exemplo, a imagem containrrr/watchtower, é possível atualizar automaticamente os nós do sistema.