



Relatório de Situação do Projeto
(4ª Reunião de Avaliação)

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

Todo o código, ficheiros de configuração e outros ficheiros diversos estão disponibilizados no Github da organização [HiveTown](#).

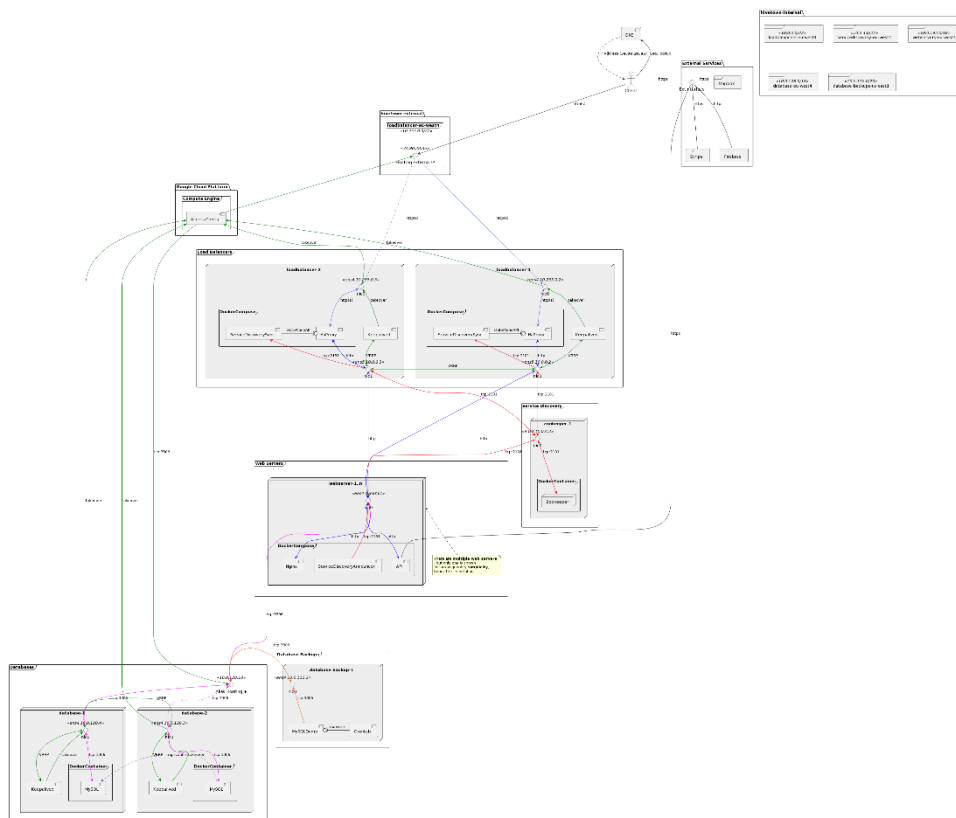
Relativamente à infraestrutura, estarão subdivididos nos repositório [hivetown/infrastructure](#) e [hivetown/planning](#), sendo que este último apenas possui ficheiros de diagramas e especificações (planeamento).

O backend (API REST) está disponibilizado em [hivetown/backend](#), e o frontend (interface) está disponível em [hivetown/frontend](#). Os ficheiros relativos à infraestrutura presentes nestes dois últimos repositórios são os Dockerfiles e a este associados.

O infrastructure está documentado, pelo que pode ser consultado durante a leitura do relatório para o compreender de melhor forma, o que evita tornar este documento longo e exaustivo.

2 Requisitos não-funcionais

2.1 Design da Arquitetura distribuída – (10.0%)



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, esta disponibilizada em hivetown.pt/api.

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 pelos balanceadores *haproxy*.

Os balanceadores também não podem ser um ponto único de falha, sendo necessárias 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, em máquinas separadas. O *healthcheck* é feito pelo *keepalived* através do protocolo *VRRP*.

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](#). Cada servidor web ou da api rest cria um znode efémero em /web-servers ou /api-servers, respetivamente, com o seu ip e porta. Para os balanceadores atualizarem a sua lista de endereçamento, fazem “watch” aos znodes /web-servers e /api-servers através dum *script python* que é executado em paralelo com o *haproxy*, num ambiente *Docker Compose*. Esse script altera a configuração do *haproxy* através da *dataplaneapi*, que apenas é exposta na rede local/privada a *containers* do *Docker Compose*.

Por fim, os servidores da API 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 balanceadores, estão

desenhadas seguindo a arquitetura ativo-passivo. Depois, manter-se-ão *snapshots* (*backups*) 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*. Apesar disso, e para que o código aplicacional seja ainda mais abstraído dos *hosts*, foi utilizado [*Docker*](#).

2.2 Especificação API – (10.0%)

A especificação da API pode ser encontrada em <https://docs.hivetown.pt>, e o ficheiro yml está no repositório [hivetown/planning](#) e em anexo, no moodle.

2.3 Especificação dos Requisitos NF - escalabilidade – (1.0%)

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, apenas foram escalados os servidores web (ver diagrama de implementação para mais detalhes). Para tal, foram desenvolvidos *scripts* especializados em:

- adicionar um nó ao sistema
 - criar a máquina virtual com base numa imagem de máquina
- remover um nó do sistema
 - Consiste na eliminação da máquina virtual.
 - Após a eliminação, a desconexão da máquina será detetada pelo *Zookeeper* (porque o *znode* é efémero) que removerá a entrada da máquina da sua lista. Depois, os *watchers* nos balanceadores de carga são notificados e removem a máquina desconectada das suas listas de endereçamento.

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

O serviço subjacente usado para a infraestrutura do sistema será a [Google Cloud Platform](#) (*GCP*), que permite a criação de máquinas virtuais novas, mas também 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.

2.3.1 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 é efetuada através do [Docker Hub](#): [hivetown/frontend](#) e [hivetown/backend](#), 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 em *Docker Compose* uma imagem do *haproxy* ([haproxytech/haproxy-alpine:2.6.10](#)) com configurações aplicadas conforme as necessidades do projeto, e uma imagem *python* ([python:3.11.3-alpine3.16](#)) para a interface entre *haproxy* e *zookeeper*.

Os *Web Servers (WS)* consistem em quatro ***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](#), 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:18--alpine](#)
 - b. Outra *build stage* para executar a API *REST*.
3. O terceiro para gerir a interface entre webserver e *zookeeper*, com um *script python* semelhante ao dos balanceadores, mas com o propósito inverso – notificar o *zookeeper* através de *znodes* efémeros
4. O quarto e último atualiza os containers quando existe uma nova imagem publicada. A imagem deste é a [containrrr/watchtower](#).

2.4 Especificação dos Requisitos NF - segurança – (1.0%)

No que respeita às *firewalls*, permitirão expor e limitar entradas/saídas 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](#) pelo *Docker* em cada *Docker Container*, que por defeito possui uma configuração cuidada, mas que pode ser mais bem especificada para se tornar mais restrita. Para além disso, é necessário explicitamente definir que portas são expostas ao *host*, portanto o funcionamento das aplicações fica restrito aos *containers* e apenas é exposto ao próprio *host* o mínimo. Ainda assim, a *GCP* permite configurar as redes e máquinas para tornar a exposição mais controlada, sendo esta a principal firewall (network based) utilizada no projeto.

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.

<input type="checkbox"/>	Name	Type	Targets	Filters	Protocols / ports	Action	Priority	Network ↑
<input type="checkbox"/>	hivetown-allow-zookeeper-client	Egress	zookeeper-clie	IP ranges: 10.0.4.0/22	tcp:2181	Allow	1000	hivetown
<input type="checkbox"/>	hivetown-allow-http-loadbalancer-to-webservers	Ingress	Apply to all	IP ranges: 10.0.0.0/22	tcp:8080, 8081	Allow	1000	hivetown
<input type="checkbox"/>	hivetown-allow-mysql	Ingress	mysql	IP ranges: 10.0.192.0/18, 10.0.	tcp:3306	Allow	1000	hivetown
<input type="checkbox"/>	hivetown-allow-rrp-database	Ingress	rrp-database	Tags: rrp-database	112	Allow	1000	hivetown
<input type="checkbox"/>	hivetown-allow-rrp-loadbalancers	Ingress	rrp-loadbalan	Tags: rrp-loadbalancer	112	Allow	1000	hivetown
<input type="checkbox"/>	hivetown-allow-zookeeper-server	Ingress	zookeeper-ser	Tags: zookeeper-client	tcp:2181	Allow	1000	hivetown
<input type="checkbox"/>	ssh-backups	Ingress	Apply to all	IP ranges: 10.0.128.4, 10.0.128	tcp:22	Allow	1000	hivetown
<input type="checkbox"/>	hivetown-allow-icmp	Ingress	Apply to all	IP ranges: 0.0.0.0/0	icmp	Allow	65534	hivetown
<input type="checkbox"/>	hivetown-allow-ssh	Ingress	ssh	IP ranges: 35.235.240.0/20	tcp:22	Allow	65534	hivetown
<input type="checkbox"/>	hivetown-external-allow-http	Ingress	http-server	IP ranges: 0.0.0.0/0	tcp:80, 443 icmp	Allow	65534	hivetown-external
<input type="checkbox"/>	hivetown-external-allow-ssh	Ingress	ssh	IP ranges: 35.235.240.0/20	tcp:22	Allow	65534	hivetown-external

Nesta imagem é possível visualizar as regras de firewall implementadas. Não obstante, podem ser consultadas em maior detalhe no repositório infrastructure, em cada componente.

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.

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 obrigatória a comunicação por *HTTPS (TLS/SSL)*. Os pedidos HTTP são redirecionados para HTTPS para manter compatibilidade com quem utilize HTTP. Como referido anteriormente, os balanceadores de carga são de camada 7, o que implica que terão de decifrar os pedidos (SSL Termination).

2.5 Especificação dos Requisitos NF - tolerância a faltas – (1.0%)

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

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 – torna o estado do *mysql* como *master* e aponta o *IP Alias* (funciona como *IP Flutuante*) para si) 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 original torna-se o passivo corrente enquanto que o passivo original mantém-se o ativo até ter uma falta.

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 está localizado noutra região (us-east1), 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.

2.6 GitHub com branches individuais e releases – (4.0%)

A estratégia de *branching* utilizada é por *feature* (ou se for feito algum *fix*) e diz respeito apenas aos artefactos que compõem essa alteração.

Quanto às releases, é necessário notar primeiramente que as *commits* feitas no *backend* foram, de início, seguindo a especificação das [commits convencionais](#). Apenas após algum tempo é que o mesmo processo foi adotado no *frontend* e restantes repositórios. Também no *backend*, usando o [husky](#), foi possível obrigar a que certos *scripts* executem com sucesso – no âmbito do projeto, foi apenas definido um *script* que verifica se a mensagem de *commit* segue o padrão referido anteriormente. Isto está disponível na diretoria “.husky”,

A utilização deste padrão é crítica para uma automação de versionamento, pois dessa forma os *commits* são legíveis por humanos e máquinas. Este padrão segue o versionamento semântico (semver), que estabelece *major*, *minor*, e *patch*, separados por pontos.

Portanto, usando a Github Action [release-please](#) da Google é possível aproveitar o potencial dos pontos referidos acima: quando há alguma alteração ao *main* (ie: um *PR* é aceite), a *action* executa e é feito um novo *PR* que inclui alterações ao *package.json* (como se trata dum projeto *node* este ficheiro inclui algumas configurações e, mais importante neste caso, a versão) e ao CHANGELOG. Quando esse *PR* é aceite, a *action* cria uma *release* com a nova versão e alterações feitas, tal como aparece no CHANGELOG.

2.7 Implementação da arquitetura distribuída – (12.0%)

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 webserver 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 webserver desfrutam de uma base de dados ativa, exposta por um IP flutuante interno, descrito mais detalhadamente no ponto 2.7.3. Fazem uso também de serviços externos, para mapas, autenticação e pagamentos.

2.7.1 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.

2.7.2 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.

2.7.3 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 na vm-database-1 com o propósito de iniciar o servidor MySQL como master;
 - initSlave.sh: apenas presente na vm-database-2 com o propósito de 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.

2.8 Balanceador de carga (e.g., HAProxy, Nginx) e escalabilidade – (10.0%)

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 (isto é, deixa de as receber), 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*.

A escalabilidade é manual, portanto necessita de um administrador de sistemas. A GCP disponibiliza a *gcloud cli* que permite, pela linha de comandos, interagir com a *Compute Engine*, onde estão alocados os servidores web. Assim, pela linha de comandos e usando o *gcloud cli*, é possível especificar a zona onde se deve criar o servidor web.

O restante é automático. Durante o *startup* dum servidor web é executado um script (em `infrastructure/web-servers/entrypoint.sh`) para que o `MACHINE_IP` do ficheiro de configurações (`.env`) obtenha o valor do IP atual da máquina. Depois disso, é reiniciado o Docker Compose e o servidor está pronto a receber pedidos.

Por escalar o núcleo do serviço, os webserver 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 webserver, 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.

2.9 Tolerância a faltas e verificação de saúde – (5.0%)

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 webserver, 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, estas que já foram mencionadas anteriormente) 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 e por se tratar de uma funcionalidade extra. No futuro, se existir a possibilidade devido a tempo disponível não requerido 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. A escolha de mais duas instâncias é porque com um total de 3 é possível haver 1 falha, e está relacionado com o quórum, algoritmo de eleição de líder usado pelo zookeeper.

2.10 Implementação de mecanismo de Autenticação e Autorização (Auth0) – (5.0%)

2.10.1 Autenticação

Primeiramente, para que um utilizador possa iniciar sessão, necessita de se registar na aplicação através da página disponibilizada para o efeito. Essa página é depois responsável pela criação da seguinte transação:

1. Criar utilizador no serviço externo de autenticação (Firebase)
2. Criar o consumidor/produtor na aplicação

Desta forma, a aplicação nunca acede nem salvaguarda qualquer dado ou rasto das passwords, delegando esse serviço para o Firebase. Como é uma transação, se um dos passos falha, tudo falha, voltando atrás. As informações guardadas pela aplicação são o email, a morada, o número de telemóvel, NIF, entre outros.

Ao registar-se, o utilizador está autenticado. Abaixo descreve-se o processo de autenticação quando o utilizador inicia sessão.

Com um utilizador registado em ambos os sistemas, é possível que este inicie sessão (se autentique perante o sistema). Para tal, numa página específica para o login, são solicitados o email e a senha, que são enviados ao Firebase. Apenas em caso de resposta positiva, o token JWT que identifica o utilizador é persistido no browser por uma cookie e guardado em memória na *store Vuex*. Para confirmar uma autenticação bem-sucedida, a mesma página executa um pedido **GET /auth** à REST API que retorna o consumidor/produtor autenticado ou um erro. O valor retornado é também guardado em memória na *store Vuex*.

No frontend, os acessos a componentes e ações são controlados para evitar erros, mas por si só este método não garante qualquer segurança.

Para tal, a API REST verifica em cada rota se é necessária autenticação e autorização, respondendo com erro caso o utilizador não se autentique corretamente ou não tenha permissões suficientes para aceder ao recurso.

2.10.2 Autorização

A autorização tem por base um controlo de acesso baseado em cargos (RBAC), em que um utilizador pode possuir um cargo e este é que detém as permissões. Os cargos existentes atualmente são os seguintes, por ordem de grau de permissão decrescente:

1. Root
 - ALL – acesso a todas as rotas e métodos da API REST
2. AccountManager
 - ALL_CONSUMER – acesso a todas as rotas e métodos (inclui eliminar consumidores) em /consumers da API REST
 - ALL_PRODUCER – acesso a todas as rotas e métodos (inclui eliminar fornecedores) em /producers da API REST
3. AccountEditor
 - READ_OTHER_CONSUMER – acesso a operações de leitura em /consumers da API REST (que não do próprio)
 - READ_OTHER_PRODUCER – acesso a operações de leitura em /producers da API REST (que não do próprio)
 - WRITE_OTHER_CONSUMER – acesso a operações de leitura em /consumers e algumas operações de eliminação de relações do consumidor (mas nunca do consumidor em si) da API REST (que não do próprio)

- **WRITE_OTHER_PRODUCER** – acesso a operações de leitura em `/producers` e algumas operações de eliminação de relações do fornecedor (mas nunca do fornecedor em si) da API REST (que não do próprio)
4. **ContentManager**
 - **ALL_CATEGORY** – acesso a operações de escrita e eliminação em `/categories`
 - **ALL_PRODUCT** – acesso a operações de escrita e eliminação em `/categories`
 5. **ContentEditor**
 - **WRITE_CATEGORY** – acesso a operações de escrita em `/categories`
 - **WRITE_PRODUCT** – acesso a operações de escrita em `/categories`

No fundo, os *Managers* podem eliminar entidades, enquanto os *Editors* apenas podem criar e editar entidades. A exceção de eliminação deve-se a “editar” propriedades de relações das entidades – por exemplo: eliminar uma morada de um consumidor.

As permissões são implementadas com recurso a *bitmasks* a inteiros de 32 bits, mas para além das permissões por RBAC, é possível que uma rota defina as suas permissões próprias, sendo que **ou** o utilizador tem a permissão do RBAC **ou** tem as permissões (todas – e entre si) da rota – por exemplo, para aceder a **GET /consumers/:idConsumer/addresses**, ou um utilizador possui a permissão **READ_OTHER_CONSUMER** ou o seu id é igual ao id do consumidor solicitado.

2.11 Canais seguros, DNS e configuração de firewall – (10.0%)

2.11.1 Uso de TLS com certificado assinado por uma Autoridade Certificadora – (4.0%)

Para que seja garantida confidencialidade na comunicação assim como autenticação do servidor, é necessário que o ponto de início do sistema esteja equipado com certificados assinados por uma Autoridade Certificadora.

A AC escolhida foi a Let’s Encrypt, que os fornece gratuitamente. O balanceador de carga 1 é o responsável pela geração e renovação dos certificados, pelo que os restantes (o balanceador de carga 2 no projeto) terão de ir buscar o certificado ao gestor.

A implementação necessita que a gestão de certificados seja feita de forma manual. Isto é, a criação do primeiro certificado (neste passo é necessário desativar a porta 443 do HaProxy do balanceador assim como o redirecionamento quando o pedido não é HTTPS) tem de ser feito pelo administrador de sistemas. Para auxiliar, existe um script que gera o certificado.

Esse script usa a imagem `certbot/certbot` do docker hub para gerar o certificado, e para responder ao `acme-challenge`, expõem-se localmente na porta 8888. O balanceador foi configurado para redirecionar tráfego com path começando por `/.well-known/acme-challenges/` para essa porta, ou seja, para o `certbot`.

A transferência dos certificados do balanceador gestor para os restantes é uma operação manual que cabe ao administrador de sistemas fazer. Isto, no futuro, deverá ser automatizado pois este processo deve ocorrer a cada 60 dias (o certificado expira em 90 dias, mas assim permite alguma margem).

Da mesma forma, a renovação de certificados cabe também ao administrador.

No futuro, ao automatizar estas duas operações, seria permitido ao sistema gerir-se a si próprio, renovando os certificados “eternamente”.

Pela arquitetura, nota-se que é feita uma terminação SSL/TLS nos balanceadores. Isto torna a gestão de certificados imensamente mais simples pois apenas é necessário fazê-la em duas máquinas, ao invés de n máquinas (os webserver). A principal razão para esta escolha de arquitetura é que, quem tem acesso à rede interna tem também acesso às bases de dados, onde a informação não está cifrada. Logo, cifrar a comunicação na rede interna não é uma mais-valia, e acrescenta carga aos servidores de forma desnecessária.

De seguida, é possível visualizar o ficheiro template (abordado em 3.4 *Gestão de IPs e Credenciais*) da configuração do HaProxy. As alterações efetuadas para esta entrega são o bind na porta 443 para receber tráfego HTTPS, redirecionamento de acme challenges para o certbot, e a definição do backend do certbot. *host.docker.internal* é um nome que aponta para o ip da máquina host.

```
global
log 127.0.0.1 local0
log 127.0.0.1 local1 debug
maxconn 45000
daemon
stats socket /var/run/haproxy.sock mode 600 level admin
stats timeout 2m

defaults hvt defaults
log global
mode http
retries 3
timeout connect 4s
timeout server 30s
timeout client 30s
timeout check 3s

frontend hvt-frontend from hvt-defaults
bind *:80
bind *:443 ssl crt /etc/letsencrypt/live/hivetown.pt/hivetown.pt.pem ssl-min-ver TLSv1.2
# Test URI to see if its a letsencrypt request
acl letsencrypt-acl path_beg /.well-known/acme-challenge/
use_backend letsencrypt-backend if letsencrypt-acl
# Redirect to https if using http
http-request redirect scheme https unless { ssl_fc }

stats enable
stats uri /haproxy
stats realm Haproxy Statistics
stats auth $(HAProxy_STATS_USERNAME):$(HAProxy_STATS_PASSWORD)

acl is_api url_beg /api
use_backend hvt-api if is_api

default_backend hvt-web

# Proxy to letsencrypt backend
backend letsencrypt-backend from hvt-defaults
server certbot host.docker.internal:8080

backend hvt-web from hvt-defaults
default-server check
balance roundrobin
option httpchk GET /

backend hvt-api from hvt-defaults
default-server check
balance roundrobin
option httpchk GET /

# Rewrite the path to remove the /api prefix with replace path
http-request replace-path /api(/)?(.*) /2

userlist dataplane_users
user $(HAProxy_DATAPLANEAPI_USERNAME) insecure-password $(HAProxy_DATAPLANEAPI_PASSWORD)

program api
command /usr/bin/dataplaneapi --scheme http --host 0.0.0.0 --port 4444 --haproxy-bin
/usr/local/sbin/haproxy --config-file /usr/local/etc/haproxy/haproxy.cfg --reload-cmd 'kill -SIGUSR2 1' -
-reload-delay 5 --restart-cmd 'kill -SIGUSR2 1' --userlist dataplane_users --write-timeout=120s --log-
-foutdout --log-level=trace
no option start-on-reload
```

2.11.2 Nome de domínio registado e associado ao IP estático – (2.0%)

WHOIS: <https://www.pt.pt/ferramentas/whois/detalhes/?site=hivetown&tld=.pt>

O domínio *hivetown.pt* foi registado a 7 de dezembro de 2022, após escolha do nome pelos membros do grupo.

O registo foi feito em <https://dominios.pt> por este oferecer domínios no TLD .PT gratuitamente.

Após o registo, os *nameservers* foram alterados para que o servidor de DNS utilizado fosse o da Cloudflare, ao invés do servidor de DNS do *dominios.pt*, permitindo assim que o tráfego seja *proxied* pela Cloudflare quando solicitado, o que torna o serviço protegido pela vasta rede da Cloudflare relativamente a ataques de disponibilidade, sobretudo ataques distribuídos de negação de serviço (DDoS).

De momento, apenas se utiliza a Cloudflare para a gestão de DNS (estando o *proxy* desativo), mas este poderia ser ativo prontamente, podendo ainda ser ativada a opção “Under Attack” que limita mais profundamente o acesso por diversas estratégias da Cloudflare (por exemplo – captchas).

Relativamente aos registos DNS existentes, de momento, são apenas dois:

1. A record em **hivetown.pt** que aponta para **34.90.28.85** (o endereço flutuante dos balanceadores – IP público do serviço)
2. CNAME record em **www.hivetown.pt** que aponta para **hivetown.pt**

2.11.3 Configurações de Firewall – (4.0%)

Para informação mais detalhada de implementação e comandos ver READMEs do repositório [infrastructure](#).

2.11.3.1 Rede externa

Nesta rede apenas são permitidas ligações TCP nos portos 80 e 443, partindo de qualquer IP (0.0.0.0/0). Ainda, para que as máquinas possam ser acedidas pelos administradores, foi feita a

exceção de permitir SSH para as máquinas com tag de rede ssh partindo da subrede 35.235.240.0/20. Esta subrede é da Google, usada para a Google Compute Console. A gestão de chaves é feita pela Google e é garantido que apenas quem tem permissão de acesso ao projeto pode aceder às máquinas.

2.11.3.2 Rede interna

A mesma regra de ssh foi permitida nesta rede VPC.

É permitido tráfego ICMP dentro da subrede interna

2.11.3.2.1 Zookeeper

Para os clientes do zookeeper (balanceadores e webservers), que tenham a tag zookeeper-client, foi permitida que fossem estabelecidas ligações TCP na porta 2181 para a subrede 10.0.4.0/22 (a subrede dos zookeepers).

A entrada de ligações para os zookeepers foi também implementada, apenas permitindo ligações do mesmo tipo, partindo de máquinas zookeeper-client, para máquinas zookeeper-server.

2.11.3.2.2 Balanceadores

Existe também uma regra que permite as portas 8080 e 8081 aplicadas a máquinas da subrede 10.0.0.0/22 (loadbalancers) para a subrede 10.0.192.0/18 (webservers).

Para o tráfego VRRP foi permitido estabelecimento de ligações do protocolo 112 (id IANA para VRRP) entre máquinas com tag vrrp-loadbalancer.

2.11.3.2.3 Bases de dados

A mesma regra de VRRP foi aplicada às bases de dados, com a diferença da tag, sendo esta a vrrp-database. Assim impede que o tráfego VRRP das bases de dados alguma vez interfira com o tráfego VRRP dos balanceadores.

Para replicação foi permitido tráfego TCP na porta 3306 para máquinas com tag mysql e que têm origem numa das seguintes subredes: 10.0.192.0/18 (webservers, para permitir aceder aos dados), 10.0.128.0/18 (databases para replicação), 10.0.112.0/20 (database backups para criação de dumps)

Para a transferência de ficheiros entre máquinas, usado para os backups, foi permitido tráfego SSH com origem em 10.0.128.4 e 10.0.128.3 (máquinas das bases de dados) e com destino a 10.0.112.2 (máquina backups)

2.12 APIs externas – (2.0%)

Firestore

No que refere à autenticação, foram definidos na conta de produção do Firestore (o desenvolvimento e a produção são feitas em contas distintas do Firestore) o domínio da aplicação, o método de Sign-In como Email/Senha, o tipo de ambiente como “Produção” e ativado o Google Analytics de forma a obter os dados de tráfego e de utilização do site.

Para configuração do Firestore, todos os dados necessários para a comunicação entre a aplicação e o projeto registado no serviço foram gravados em variáveis ambiente e o token disponibilizado guardado na store (em memória) e persistido nas cookies dos browsers dos clientes.

Para além de facilitar o processo de dados mais sensíveis como as senhas, esta API retorna avisos e erros (como um email não registado, uma senha incorreta, etc.) que são posteriormente convertidos em mensagens para linguagem de utilizador e mostrados num pop-up. Outra funcionalidade que a aplicação utiliza deste serviço é a desativação e reativação de contas.

Stripe

Para a gestão de pagamentos e faturação na Internet utilizou-se o Stripe. A escolha do Stripe foi acertada por diversos motivos como a sua facilidade de integração (devido à documentação detalhada e API amigável), segurança pela adoção de medidas de proteção de informações dos

clientes, tais como o certificado nível 1 (o mais rigoroso e disponível do setor de pagamentos), TLS para garantir conexões seguras e ainda criptografia de dados sigilosos com AES-256, guardando as chaves em máquinas separadas.

A configuração do Stripe para o website consistiu na inicialização de uma variável, através da chave secreta fornecida pela plataforma e guardada em variáveis de ambiente.

Foi ainda utilizada a funcionalidade webhook, que consiste na receção de eventos enviados pela plataforma, como por exemplo quando um pagamento é efetuado ou cancelado.

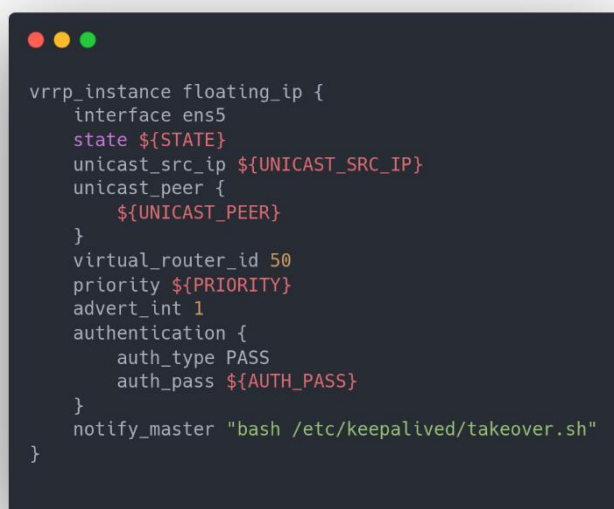
Nota: Todas as verificações relativamente a cartões bancários (validade) são feitas pela plataforma. Para além do já mencionado, o Stripe é também utilizado para processar devoluções.

Mapbox

Em relação aos mapas, é utilizada a API do Mapbox, que disponibiliza gratuitamente mapas fiéis à realidade e com muitas informações. Os pedidos à API são feitos com as coordenadas dos utilizadores, para mostrar uma localização em concreto, através de Markers/Pins e uma linha entre estes.

2.13 Gestão de credenciais e IPs na instalação – (5.0%)

De forma geral, as configurações, credenciais, e IPs são salvaguardados em variáveis de ambiente, em ficheiros com nomes *.env* (ou nalguns casos nomes parecidos). Como o código de configuração da infraestrutura é publicado na totalidade no repositório [infrastructure](#), é necessário que esses ficheiros não sejam monitorizados pelo controlo de versão, mas para facilitar aos utilizadores desse repositório, são publicados ficheiros exemplares (*.env.example*) com dados fictícios, sendo esses corretos e coerentes entre si.



```
vrrp_instance floating_ip {
    interface ens5
    state ${STATE}
    unicast_src_ip ${UNICAST_SRC_IP}
    unicast_peer {
        ${UNICAST_PEER}
    }
    virtual_router_id 50
    priority ${PRIORITY}
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass ${AUTH_PASS}
    }
    notify_master "bash /etc/keepalived/takeover.sh"
}
```

Alguns ficheiros (configuração do keepalived e haproxy, ficheiros sql das bases de dados) não têm a possibilidade de ler de variáveis de ambiente, sobretudo de ficheiros próprios como os usados, os *.env*. Para resolver essa situação, foram desenvolvidos *templates*.

Por exemplo, o ficheiro *keepalived.template.conf* é o seguinte:

E existem os placeholders (ex: *\${STATE}*) que são depois populados pelos scripts **computeTemplate**, estes que lêem do ficheiro *.env* e fazem uso do *envsubst* para substituir os valores.

Como referido anteriormente, para que o servidor web conheça o seu próprio IP para o anunciar ao *zookeeper*, existe um *script* que altera o ficheiro *.env* para colocar esse valor.

2.14 Automação e scripts para construção e lançamento da aplicação – (2.0%)

A aplicação (backend e frontend) está disponibilizada no Docker Hub, e deve ser previamente construída e publicada. Para tal, foi definida uma Github Action (*workflow*) que compila a imagem (internamente irá compilar a aplicação, parando e notificando se houver algum erro) e depois a publica no Docker Hub. Existem dois momentos em que isto é feito: quando há *pushes* (*commits* “empurrados” para o Github) num *pull request* ou no branch *main*. Quando é feito num *pull request* a *tag* atribuída à imagem corresponde ao número do PR, e quando é feito na branch *main* a *tag* é “main”. Isto pode ser verificado em [hivetown/backend](#) e [hivetown/frontend](#) e na seguinte imagem.

TAG main Last pushed in 18 minutes by hivetown	DIGEST 1c9ef570fe4a	OS/ARCH linux/amd64	SCANNED ---	LAST PULL in 18 minutes	COMPRESSED SIZE ⓘ 69.92 MB	
TAG pr-38 Last pushed 40 minutes ago by hivetown	DIGEST c41e95cd446d	OS/ARCH linux/amd64	SCANNED ---	LAST PULL ---	COMPRESSED SIZE ⓘ 69.91 MB	
TAG pr-20 Last pushed an hour ago by hivetown	DIGEST 1f4a34a52cf8	OS/ARCH linux/amd64	SCANNED ---	LAST PULL ---	COMPRESSED SIZE ⓘ 69.92 MB	
TAG pr-54 Last pushed 2 hours ago by hivetown	DIGEST e3dda689ce3f	OS/ARCH linux/amd64	SCANNED ---	LAST PULL ---	COMPRESSED SIZE ⓘ 69.86 MB	

Os servidores web, como mencionado anteriormente, têm um container que automatiza a atualização dos *containers* ao verificar se existe uma imagem mais recente, fazendo pull do Docker Hub, e reiniciando o container com a nova imagem. Como visível acima, as imagens têm cerca de 70MB, o que é bastante rápido de transferir, sobretudo por se estar localizado num *Datacenter* da Google.

2.15 Inclusão de testes unitários integrados ao processo de construção da aplicação (GitHub) – (3.0%)

Os testes unitários são executados em paralelo e com a mesma frequência que o *workflow* anterior. Foi, da mesma forma, definido como um *workflow* para *Continuous Integration* que inclui os testes unitários e o *linting*, que força uma formatação padrão no código assim como verifica alguns erros sintáticos e outras regras.

Os testes unitários foram executados a ferramentas de suporte (que não necessitam conexão à base de dados) de modo a simplificar a interação entre o ambiente de testes e a integração com o Github.

2.16 Testes de aceitação com a API – (3.0%)

A especificação da API é referente à versão 1.2.1, e no momento da escrita do relatório a implementação da API está na versão 1.2.2. Como a diferença de versões apenas representa um *minor patch*, a especificação e a implementação estão sincronizadas.

Ainda, ao longo do desenvolvimento foi desenvolvido um projeto no Postman para testar a API e manualmente confirmar se está de acordo com a especificação.

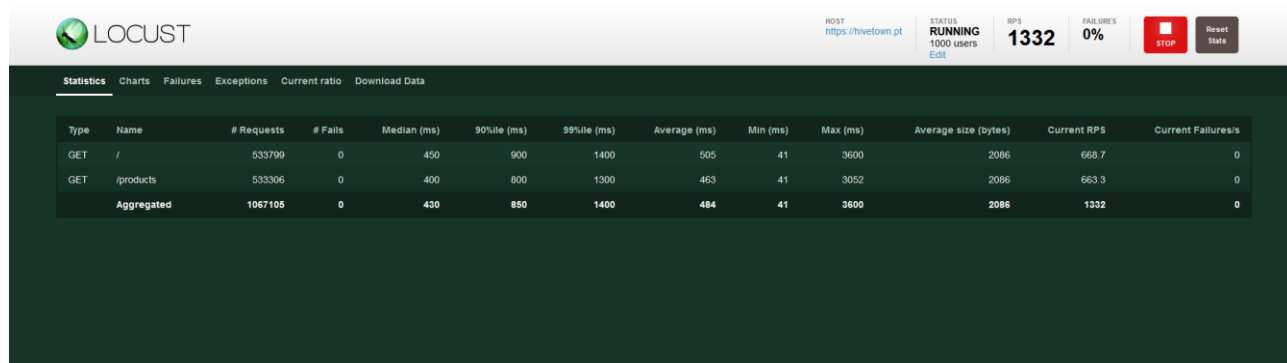
2.17 Testes de carga – (3.0%)

De modo a simular a carga de vários utilizadores no nosso website para medir e/ou identificar possíveis problemas de desempenho utilizámos a ferramenta Locust.

Os testes foram realizados com acessos em duas páginas: hivetown.pt (página inicial, simulando acessos ao website) e hivetown.pt/products (simulando o acesso à página de produtos que contém também maior interação com a API).

Deste modo, fomos aumentando progressivamente o volume de pedidos a ambas as páginas (ou seja, aumentando o número de acessos) até encontrarmos um limite que consideremos aceitável, nomeadamente respostas, em média, até 500ms.

As respostas em média com mais de 500ms surgiram com 1000 utilizadores e aproximadamente 650 RPS em cada página e com 0% de falhas.



De seguida procedemos aos testes de stress de modo a perceber como o sistema iria se comportar em caso de excesso de carga, para isto aumentámos ainda mais o número de utilizadores para 1200 que originando aproximadamente 700 RPS. As primeiras falhas surgiram com pouco mais de 1100 utilizadores e com um número total de 1356 RPS.



# fails	Method	Name	Type
144	GET	/	SSLError(OSError(24, 'Too many open files'))
3720	GET	/	OSError(24, 'Too many open files')
35	GET	/	BlockingIOError(11, 'Resource temporarily unavailable')
102	GET	/products	SSLError(OSError(24, 'Too many open files'))
3712	GET	/products	OSError(24, 'Too many open files')
17	GET	/products	BlockingIOError(11, 'Resource temporarily unavailable')

O Locust permite distribuir a carga por várias máquinas, mas foi testada apenas a partir de uma. Num caso real, a utilização de *ratelimits* iria melhorar o desempenho do serviço, sobretudo na API, por bloquear acesso temporário a clientes que excedam a quantidade de pedidos considerados normais. Por consequência, ao minimizar a quantidade de pedidos considerados como maliciosos, como se tratasse de um ataque de disponibilidade, as operações mais intensas não seriam realizadas o daria mais folgo aos servidores para responder a pedidos “normais”. Esse *ratelimiting* podia ser primeiramente implementado no balanceador de carga, por cliente/ip, e também nos servidores da API nas rotas onde os pedidos são mais intensos e menos frequentes.

2.18 Testes de vulnerabilidades – (3.0%)

Para realizar os testes de vulnerabilidades utilizámos duas ferramentas: o Nmap e o Zaproxy.

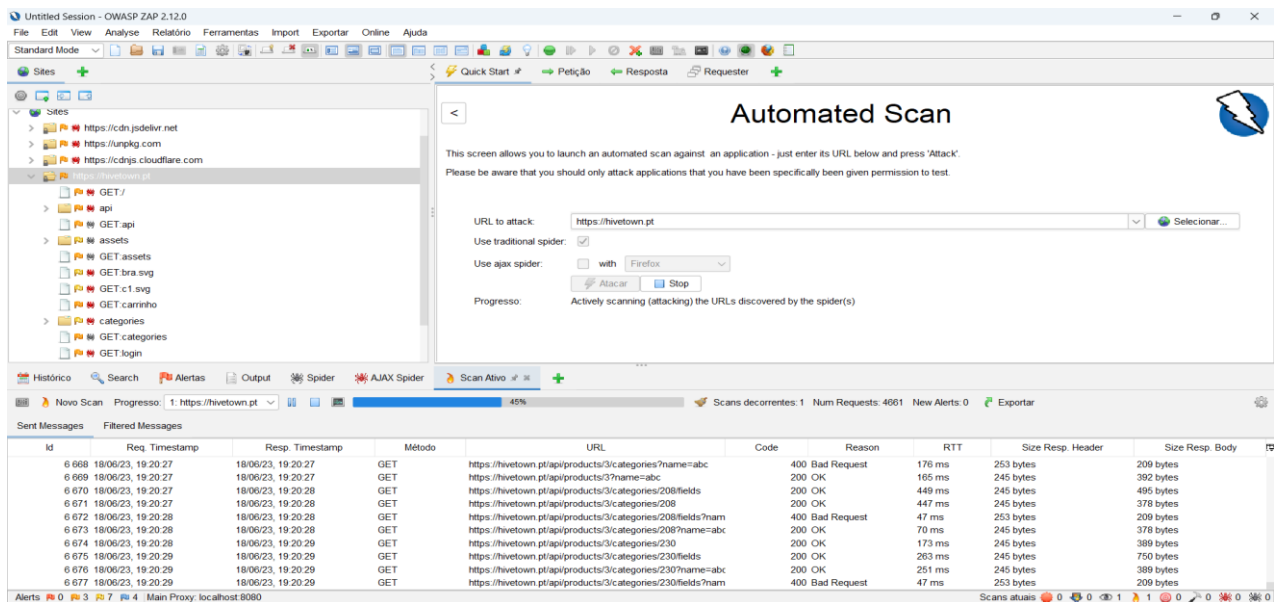
O Nmap é uma ferramenta de mapeamento de rede que pode ser usada para descobrir *hosts* ativos, portas abertas, e o software e versão prováveis do *host* e dos serviços em execução.

Zaproxy (também conhecido como OWASP ZAP): É uma ferramenta amplamente usada para testes de segurança em aplicações web. Ela permite encontrar e explorar vulnerabilidades comuns, como injeção de SQL, cross-site scripting (XSS), entre outras. O Zaproxy possui uma interface gráfica amigável e recursos abrangentes.

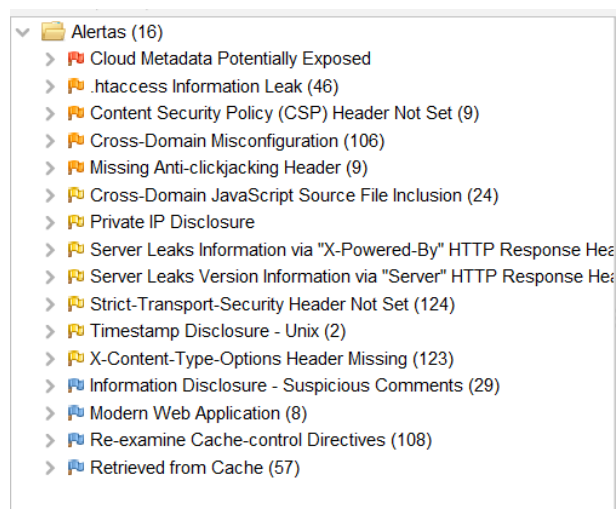
Ao combinar o Nmap com o Zaproxy podemos obter uma visão mais completa da segurança do nosso serviço. Enquanto o Nmap fornece informações sobre a infraestrutura, o Zaproxy pode identificar vulnerabilidades específicas nas aplicações web em execução nesses hosts.

Zaproxy

Foi utilizado o scan automático do Zaproxy em que inicialmente ele identificou todos os caminhos possíveis e de seguida testou-os através de pedidos corretos e incorretos.



Uma vez acabado o scan recebemos os seguintes alertas:



De seguida, é feita uma análise não detalhada dos alertas marcados com bandeira laranja e vermelha (respetivamente de média e alta prioridade), que embora não tenha existido tempo para os corrigir, estão devidamente identificados.

Nmap

Para efetuar a análise com o Nmap efetuámos os seguintes passos:

1. Identificar o sistema operativo

```
sudo nmap -O hivetown.pt
```

```
Starting Nmap 7.80 ( https://nmap.org ) at 2023-06-18 09:54 WEST
Nmap scan report for hivetown.pt (34.90.28.85)
Host is up (0.12s latency).
Not shown: 998 filtered ports
PORT      STATE SERVICE
80/tcp    open  http
443/tcp   open  https
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Aggressive OS guesses: Linux 2.6.32 (94%), Linux 3.10 - 4.11 (94%), Linux 3.2 - 4.9 (94%), Linux 3.4 - 3.10 (94%), Linux
 2.6.32 - 3.10 (93%), Linux 2.6.32 - 3.13 (93%), Linux 3.10 (92%), Linux 2.6.22 - 2.6.36 (91%), Synology DiskStation Man
ager 5.2-5644 (91%), Linux 2.6.39 (91%)
No exact OS matches for host (test conditions non-ideal).

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 22.87 seconds
```

É perceptível que o Nmap não conseguiu determinar com clareza o sistema operativo que o servidor web utiliza, o que provavelmente se deve às regras de firewall definidas.

Mesmo os palpites do Nmap, que são baseados em padrões encontrados para pedidos e respostas específicas, foram próximos mas incorretos, uma vez que as máquinas são todas Ubuntu 22.04.2 LTS e todos os palpites foram de máquinas *Linux*, com as versões do *kernel* visíveis na imagem.

2. Deteção de serviços

```
nmap -sV hivetown.pt
```

```
Starting Nmap 7.80 ( https://nmap.org ) at 2023-06-18 10:06 WEST  
Nmap scan report for hivemont.pkt (34.90.28.85)  
Host is up (0.13s latency).  
Not shown: 998 filtered ports  
PORT      STATE SERVICE VERSION  
400/tcp    open  http     nginx 1.23.4
```

I service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at <https://nmap.org/cgi-bin/submit.cgi?new-service>

```
SF:Port=80;V:=7.80A1T7P6(nginx)S=x86_64-pc-linux-gnuR(GetH  
SF:request:6A,"HTTP/1.1\x20302\x20Found\r\ncontent-length:\x20B\r\nlocation:  
SF:n;\x20htts:///r\ncache-control:\x20no-cache\r\nconnection:\x20close\r  
SF:\n\r\n")&r(HTTPOptions,6A,"HTTP/1.1\x20302\x20Found\r\ncontent-length:  
SF:\x20B\r\nlocation:\x20htts:///r\ncache-control:\x20no-cache\r\nconnec  
SF:tion:\x20close\r\n\r\n")&r(RTSRequest,CF,"HTTP/1.1\x20400\x20Bad\x20E  
SF:request\r\nContent-Length:\x2090\r\nCache-Control:\x20no-cache\r\nConnec  
SF:Tion:\x20close\r\nContent-Type:\x20text/html\r\n\r\n<html><body><h1>400  
SF:F:\x20BAD\x20request</h1>\nYour\x20browser\x20sent\x20an\x20invalid\x20Re  
SF:quest.\n</body></html>\n")&r(XIProbe,CF,"HTTP/1.1\x20400\x20Bad\x20E  
SF:request\r\nContent-Length:\x2090\r\nCache-Control:\x20no-cache\r\nConnec  
SF:tion:\x20close\r\nContent-Type:\x20text/html\r\n\r\n<html><body><h1>400  
SF:F:\x20BAD\x20request</h1>\nYour\x20browser\x20sent\x20an\x20invalid\x20Re  
SF:quest.\n<n</body></html>\n")&r(FourOhFourRequest,8D,"HTTP/1.1\x20302\x2  
SF:0Found\r\ncontent-length:\x20B\r\nlocation:\x20htts:///nice%20ports%2C  
SF:/Tri&Eity.txt%20ebak\r\ncache-control:\x20no-cache\r\nconnection:\x20C  
SF:lose\r\n\r\n")&r(RPCCheck,CF,"HTTP/1.1\x20400\x20BAD\x20request\r\nCon  
SF:ent-length:\x2090\r\nCache-Control:\x20no-cache\r\nConnection:\x20clos  
SF:e\r\nContent-Type:\x20text/html\r\n\r\n<html><body><h1>400\x20BAD\x20re  
SF:quest<h1>\nYour\x20browser\x20sent\x20an\x20invalid\x20request.\n<n</bo  
SF:dys</html>\n")&r(OSVersionInReqTCP,CF,"HTTP/1.1\x20400\x20Bad\x20E  
SF:gust\r\nContent-Length:\x20B\r\nCache-Control:\x20no-cache\r\nConnectio  
SF:ion:\x20close\r\nContent-Type:\x20text/html\r\n\r\n<html><body><h1>400)\nSF:f:\x20bad\x20request</h1>\nYour\x20browser\x20sent\x20an\x20invalid\x20req  
SF:uest.\n<n</body></html>\n")&r(DNSStatusRequestTCP,CF,"HTTP/1.1\x20400\x  
SF:20BAD\x20request\r\nContent-Length:\x2090\r\nCache-Control:\x20no-cache  
SF:r\nConnection:\x20close\r\nContent-Type:\x20text/html\r\n\r\n<html><bO  
SF:dys<h1>400\x20BAD\x20request</h1>\nYour\x20browser\x20sent\x20an\x20inv  
SF:salid\x20request).\n<n</body></html>\n")&r(Help,CF,"HTTP/1.1\x20400\x20BA  
SF:D):\x20request\r\nContent-Length:\x2090\r\nCache-Control:\x20no-cache\r\n  
SF:Connection:\x20close\r\nContent-Type:\x20text/html\r\n\r\n<html><body>  
SF:h1>400\x20BAD\x20request</h1>\nYour\x20browser\x20sent\x20an\x20invalid  
SF:\x20request.\n<n</body></html>\n");
```

Service detection performed. Please report any incorrect results at <https://nmap.org/submit/>.

Nmap done: 1 IP address (1 host up) scanned in 37.39 seconds

O Nmap conseguiu detectar dois serviços em execução no nosso host: um servidor HTTP na porta 80/tcp e um servidor SSL/HTTP (HTTPS) na porta 443/tcp, este último com a indicação correta de que está usando o servidor Nginx na 1.23.4. Esta detecção deve-se, provavelmente, pois as respostas do servidor web do *frontend*/interface possuírem um *header* com essa informação (x-served-by)

3. Varrimento de portos

Inicialmente tentou fazer-se a varrimento em todos os portos possíveis, porém não foi possível obter nenhuma resposta em tempo útil. Por isso, mudou-se de estratégia para um varrimento de portos comuns de estarem abertos.

```
nmap -p 20,21,22,23,25,53,80,443,110,143,161,162,3389,3306,5900,5432,1521,1433,2181 hivetown.pt
```

```

Starting Nmap 7.80 ( https://nmap.org ) at 2023-06-19 00:03 WEST
Nmap scan report for hivetown.pt (34.90.28.85)
Host is up (0.098s latency).

PORT      STATE      SERVICE
20/tcp    filtered  ftp-data
21/tcp    filtered  ftp
22/tcp    filtered  ssh
23/tcp    filtered  telnet
25/tcp    filtered  smtp
53/tcp    filtered  domain
80/tcp    open      http
110/tcp   filtered  pop3
143/tcp   filtered  imap
161/tcp   filtered  snmp
162/tcp   filtered  snmptrap
443/tcp   open      https
1433/tcp  filtered  ms-sql-s
1521/tcp  filtered  oracle
2181/tcp  filtered  eforward
3306/tcp  filtered  mysql
3389/tcp  filtered  ms-wbt-server
5432/tcp  filtered  postgresql
5900/tcp  filtered  vnc

Nmap done: 1 IP address (1 host up) scanned in 2.78 seconds

```

Como esperado apenas as portas 80 e 443 estão abertas, respetivamente http e https. O estado "filtered" indica que o Nmap não recebeu uma resposta definitiva do host alvo para determinar o estado da porta. Isso pode acontecer quando uma firewall bloqueia ou filtra o tráfego da porta e deste modo não estão abertas para comunicação direta com a Internet.

4. Execução de alguns scripts de segurança que o nmap disponibiliza

```
nmap --script <nome_do_script> hivetown.pt
```

http-vuln-*

Este é uma família de scripts que verifica vulnerabilidades específicas em servidores web, como o [http-vuln-cve2015-1635](#) que verifica a vulnerabilidade do IIS HTTP.sys (MS15-034).

```

Starting Nmap 7.80 ( https://nmap.org ) at 2023-06-18 18:24 WEST
Nmap scan report for hivetown.pt (34.90.28.85)
Host is up (0.14s latency).
Not shown: 998 filtered ports
PORT      STATE SERVICE
80/tcp    open  http
443/tcp   open  https
|_ http-vuln-cve2011-3192:
|   VULNERABLE:
|   Apache byterange filter DoS
|   State: VULNERABLE
|   IDs: CVE:CVE-2011-3192 BID:49303
|   The Apache web server is vulnerable to a denial of service attack when numerous
|   overlapping byte ranges are requested.
|   Disclosure date: 2011-08-19
|   References:
|   https://www.securityfocus.com/bid/49303
|   https://www.tenable.com/plugins/nessus/55976
|   https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3192
|_ https://seclists.org/fulldisclosure/2011/Aug/175
|_ http-vuln-cve2017-1001000: ERROR: Script execution failed (use -d to debug)

Nmap done: 1 IP address (1 host up) scanned in 22.70 seconds

```

O script `http-vuln-cve2011-3192` do Nmap identificou uma vulnerabilidade conhecida no servidor Apache, que é relacionada a um ataque de negação de serviço (DoS) que ocorre quando várias faixas de bytes sobrepostas são solicitadas.

Porém, deve tratar-se de um falso positivo uma vez que o Apache não se encontra na nossa *stack*, mas talvez o nginx possa ser igualmente vulnerável.

ssl-heartbleed

Este script verifica se o servidor SSL/TLS é vulnerável à falha de segurança Heartbleed (CVE-2014-0160).

```

Starting Nmap 7.80 ( https://nmap.org ) at 2023-06-18 18:27 WEST
Nmap scan report for hivetown.pt (34.90.28.85)
Host is up (0.13s latency).
Not shown: 998 filtered ports
PORT      STATE SERVICE
80/tcp    open  http
443/tcp   open  https

Nmap done: 1 IP address (1 host up) scanned in 16.88 seconds

```

Uma vez que não retornou nenhuma informação específica sobre a presença da vulnerabilidade Heartbleed (CVE-2014-0160) no servidor HTTPS, isso indica que o servidor não é vulnerável a esta falha.