Ciências ULisboa | Informática

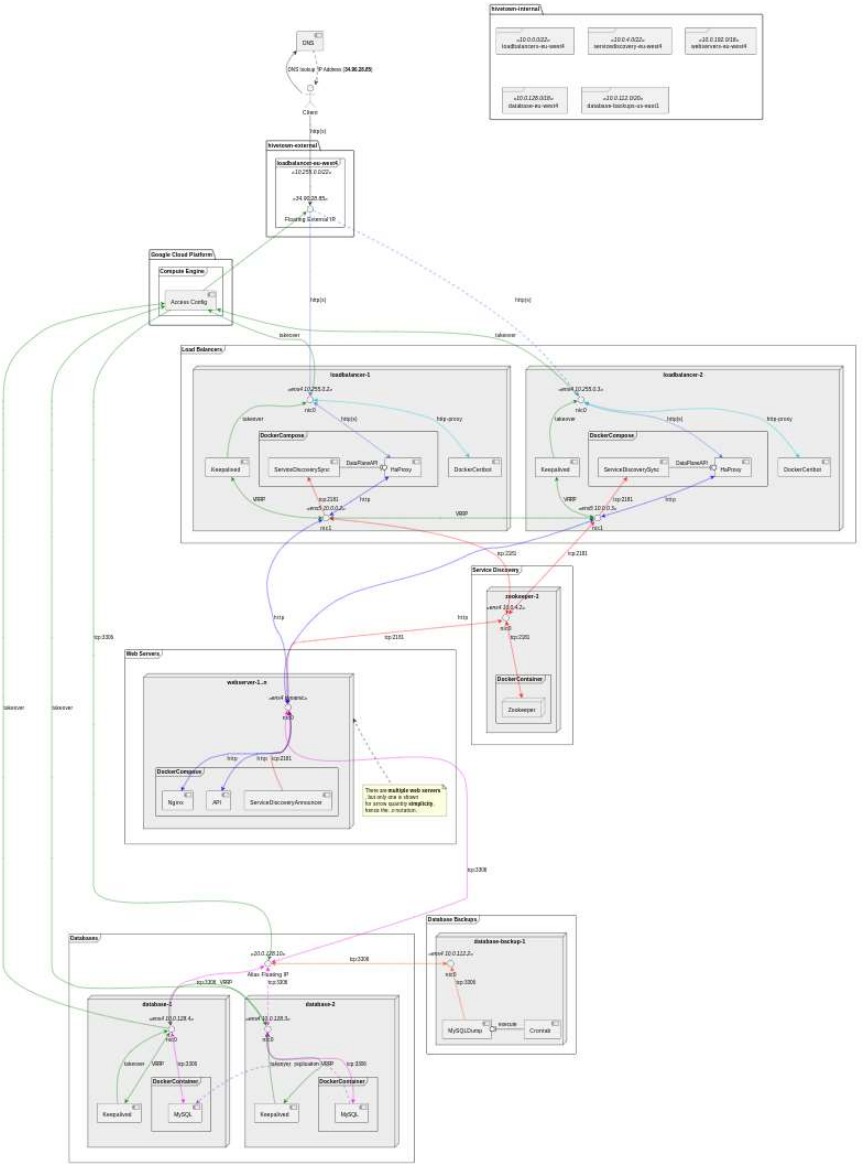
Projeto de Tecnologias de Redes

Relatório de Situação do Projeto
(3ª 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 Desenho atualizado da arquitetura distribuída



As alterações que se devem notar relativamente à iteração anterior são a adição do componente Certbot nos balanceadores de carga, este executado manualmente num Docker Container, para geração e renovação de certificados.

3 Segurança

3.1 Implementação de Mecanismo de Autenticação e Autorização (Firebase)

3.1.1 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.

3.1.2 Autorização

A autorização tem por base um controlo de acesso baseado em cargos (RBAC), em que um utilizador pode possuir um cargo, este 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**) 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.

3.2 Canais seguros, DNS e configuração de firewall

3.2.1 Uso de TLS com certificado assinado por uma Autoridade Certificadora (AC)

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, estes 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) 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 webservers). 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, redireccionamento de acme challenges para o certbot, e a definição do backend do cerbot. *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 5s

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:8888

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-
to=stdout --log-level=trace
  no option start-on-reload
```

3.2.2 Nome de domínio registado e associado ao IP estático

WHOIS: https://www.pt.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**

3.2.3 Configurações de Firewall

Para informação mais detalhada de implementação e comandos ver READMEs do repositório infrastructure.

3.2.3.1 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 aceder ao projeto pode aceder às máquinas.

3.2.3.2 Rede interna

A mesma regra de ssh foi permitida nesta rede VPC.
É permitido tráfego ICMP dentro da subrede interna

3.2.3.2.1 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.

3.2.3.2.2 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.

3.2.3.2.3 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)

3.3 Configurar uso de APIs externas

Firebase
No que refere à autenticação, foram de definidos na conta de produção do Firebase (o desenvolvimento e a produção são feitas em contas distintas do Firebase) 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 Firebase, 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.

3.4 Gestão de credenciais e IPs na instalação

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 infrastrutura é publicado na totalidade no repositório infrastructure, é necessário que esses ficheiros não sejam monitorados 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.

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:

```
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"
}
```

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.