



Faculdade de Design,
Tecnologia e Comunicação
Universidade Europeia

Relatório Final – *Two-Factor Authentication*

Licenciatura em Engenharia informática

2021/2022

Discentes:

Rafael Santos 20190800

João Alves 20190908

Índice

1	ENQUADRAMENTO	3
2	ARQUITETURA DO PROJETO	4
3	IMPLEMENTACAO AZURE	6
3.1	VIRTUAL MACHINES & VIRTUAL NETWORKS	6
3.2	LOAD BALANCER	7
3.3	CONFIGURACAO DATABASES (LINUX)	10
3.4	LOAD BALANCER (MYSQL)	16
4	IMPLEMENTACAO AZURE (APP SERVICES)	17

Enquadramento

Neste relatório vamos abordar uma temática de Sistemas Distribuídos focada num projeto de 2FA (*Two-Factor Authentication*). Este desafio foi proposto na cadeira de Sistemas Distribuídos de forma a implementar não só uma aplicação, mas também toda a sua arquitetura de forma a relacionar-se com o conteúdo lecionado na cadeira.

A arquitetura tem de apresentar uma tolerância a falhas, isto é que será necessário existir alguma forma de proteger o sistema de apresentar falhas por exemplo caso um servidor falhe exista sempre forma de não ter a aplicação em “baixo”.

Vamos abordar várias abordagens e técnicas caso o projeto tivesse sido feito localmente falando da configuração *Master-Master* ou Ativo/Ativo em Bases de Dados, *Azure (Cloud)*.

Arquitetura do Projeto

Numa fase inicial decidimos implementar todo este sistema localmente fazendo recurso a duas *Firewalls*, dois servidores, um deles na DMZ (*Demilitarized Zone*) e outro numa rede “interna”. Na rede interna iria existir uma ligação direta às bases de dados que neste caso iriam usar *Cluster* ativo/passivo para existir uma tolerância a faltas.

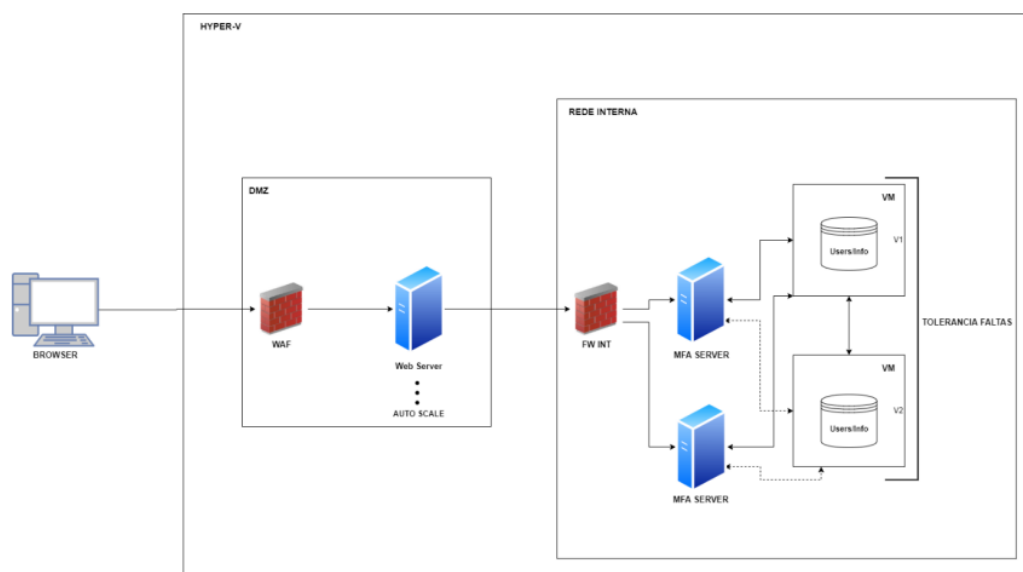


FIG.1- Arquitetura Geral

Conseguimos implementar as *Firewalls* usando *Pfsense* a correr numa máquina de *Linux*, as bases de dados foram implementadas em *MySQL* para melhor uso dos dados e das *queries* em *server side* na aplicação. Ambos os servers foram implementados em *Windows* necessitando de instalar *Node.js* para que o *CMD* esteja preparado para correr a aplicação.

Esta aplicação está dividida entre duas pequenas aplicações, a rede interna vai ter a solução de “Códigos” na porta 3000, esta solução apresenta uma pequena aplicação onde o utilizador depois de fazer o registo e o *login* consegue meter o seu *Secret* e ter acesso aos códigos para efetuar login no site.

Este site está hospedado na DMZ no *Web Server* exatamente na porta 3001, esta aplicação como já dito é o site onde esta hospedado o “site” onde o utilizador consegue efetuar e ativar o *Two-Factor Authentication*.

Com o desenvolvimento do projeto deparamo-nos com um problema/obstáculo de falta de recursos em termos físicos com o nosso hardware, não conseguindo fazer as máquinas todas necessárias para fazer o projeto como também a demonstração em aula.

Assim surgiu a ideia de implementar algo completamente fora da nossa zona de conforto, algo completamente novo e implementar e transformar esta arquitetura em *Cloud* utilizando *Azure*.

Decidimos assim construir três máquinas virtuais em *Azure* (Windows), uma simulando a DMZ com um IP publico para que possa ser acedida livremente bem como localmente usando a ligação remota do Windows e as outras duas com IP privado que mais a frente vamos explicar, mas estes surgindo das *Subnets* criadas.

Implementamos um *Load Balancer* em *Azure*, fazendo toda a configuração necessária desde a pool do *backend* (*Virtual Machines*), o IP do *frontend* (IP este que iria dar acesso às máquinas da rede interna), regras de *Load balancing* (neste caso na porta 3000) e finalmente o *health probe* para validar se existe uma resposta das *Virtual Machines* na porta 3000. De lembrar que o *Azure* ajuda na implementação deste serviço nos servidores do mesmo, mas todos os tipos de configurações são feitos pelos utilizadores não tornando este serviço *plug and play*.

Em termos de base de dados mais tarde no projeto criamos mais duas máquinas em Linux, instalando o MySQL e o Putty para conseguir aceder às mesmas. Configuramos como vai ser explicado futuramente neste relatório uma configuração Ativa/Ativa ou realmente com o nome de Master-Master, que nos deu possibilidade sempre que usamos uma da base de dados a outra é atualizada no momento, para que exista uma tolerância a falhas.

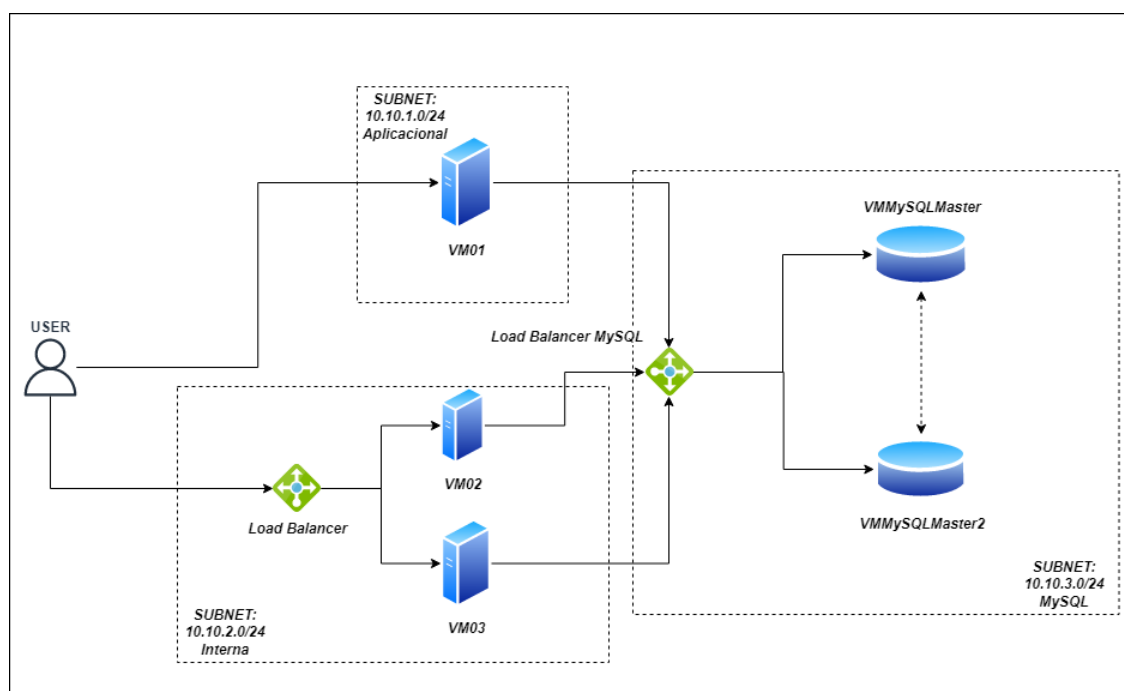


Fig 2 – Arquitetura Cloud.

Implementação Azure

É necessário explicar que algumas destas variáveis para criação da mesma são necessárias para criação mais a frente do *Load Balancer* bem como das restantes máquinas virtuais. Entre estas variáveis está o *Resource Group* que é um repositório de componentes do nosso projeto, a região de todas as máquinas e componentes que foi escolhido *North Europe* e finalmente o tipo de autenticação de acesso às mesmas foi feito por uma conta de administrador bem como o “*Azure Spot Instance*” ativado, visto que temos uma subscrição com crédito limitado esta serve assim para reduzir custos.

Virtual Machines & Virtual Networks

Antes de explicar a criação das máquinas virtuais é necessário criar uma rede para que as mesmas comuniquem, então criamos de uma rede. Para criação desta Virtual Network para além do nome da mesma, é apenas necessário escolher o IP da mesma e o prefixo neste caso foi escolhida a rede 10.10.0.0/16. Depois em “*Add Subnet*” encontramos uma página em que precisamos só de meter o nome e o IP de cada *Subnet* em que no nosso trabalho apenas necessitamos de criar duas *Subnets* a Aplicacional e a Interna (10.10.1.0 e 10.10.2.0).

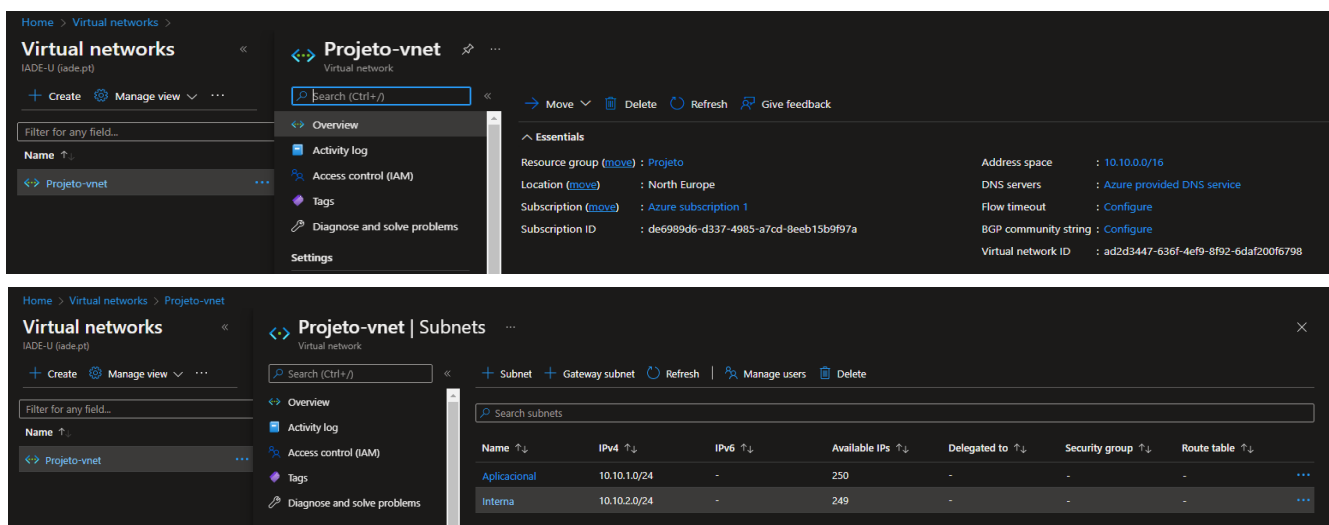
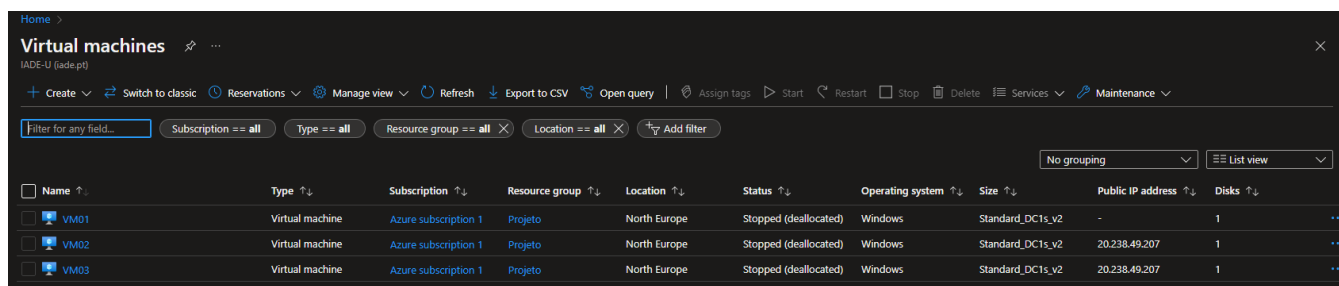


Fig 3 – Criação de Rede e Subnets.

Passando para a vertente da criação das máquinas virtuais, uma das escolhas mais importantes e mais difíceis no nosso trabalho foi o tipo de máquina visto que a subscrição que o nosso grupo tem no *Azure* não nos possibilita ter mais de três vCPUs, isto é, no máximo poderíamos apenas ter três máquinas com um CPU cada uma, assim escolhemos a máquina do tipo de DC1s_V2.

Em termos de definições de disco apenas foi escolhido o “*Standard HDD*” apenas por custos associados aos mesmos e nas definições de rede uma vez criada a rede e as *Subnets* associamos a nossa primeira máquina à *Subnet* “Aplicacional” como vemos na figura 1, e as outras duas máquinas à *Subnet* “Interna”.



Name	Type	Subscription	Resource group	Location	Status	Operating system	Size	Public IP address	Disks
VM01	Virtual machine	Azure subscription 1	Projeto	North Europe	Stopped (deallocated)	Windows	Standard_DC1s_v2	-	1
VM02	Virtual machine	Azure subscription 1	Projeto	North Europe	Stopped (deallocated)	Windows	Standard_DC1s_v2	20.238.49.207	1
VM03	Virtual machine	Azure subscription 1	Projeto	North Europe	Stopped (deallocated)	Windows	Standard_DC1s_v2	20.238.49.207	1

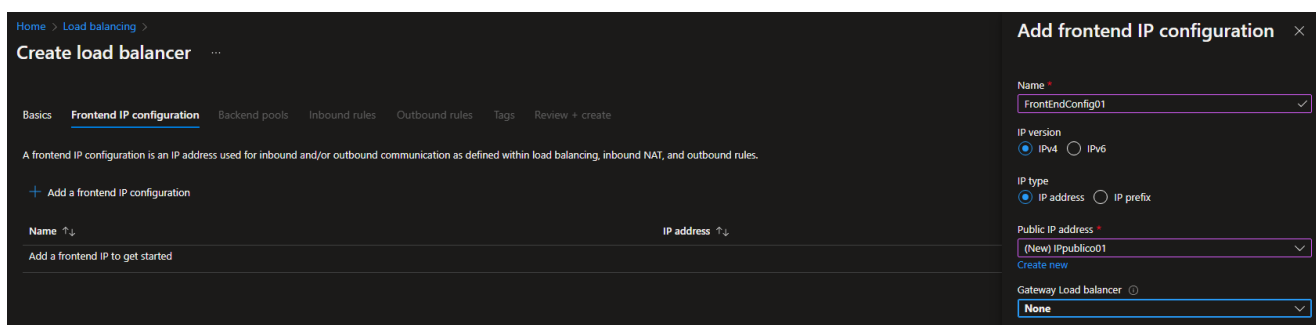
Fig 4 – Virtual Machines.

Load Balancer

Criamos um *public Load Balancer* que servirá como ponto de entrada das máquinas que contêm a componente dos códigos 2FA da nossa solução. Desta forma criamos uma redundância entre duas máquinas que funcionam como ativa/ativa onde qualquer uma pode responder ao pedido do utilizador devolvendo e atualizando os códigos 2FA.

Esta solução permite que em caso de falha de uma das máquinas não seja necessário nenhum tipo de *Failover* manual, visto que, o próprio *Load Balancer* tem configurações e métricas próprias para validar quais das máquinas estão ativas e reencaminhando o tráfego para as máquinas em funcionamento.

Estes são os passos de configuração do *Load Balancer* começando pelo IP publico de *Frontend*.



Create load balancer

Basics **Frontend IP configuration** Backend pools Inbound rules Outbound rules Tags Review + create

A frontend IP configuration is an IP address used for inbound and/or outbound communication as defined within load balancing, inbound NAT, and outbound rules.

+ Add a frontend IP configuration

Name	IP address
Add a frontend IP to get started	

Add frontend IP configuration

Name * FrontEndConfig01

IP version ☒ IPv4 ☐ IPv6

IP type ☒ IP address ☐ IP prefix

Public IP address * (New) IPpublic01

Gateway Load balancer ☐ None

Fig 5 – Criação de IP publico frontend.

Em seguida é necessário criar o *backend pool* para associar as máquinas virtuais necessárias a este *Load Balancer*, e escolhemos as máquinas baseadas na sua NIC (*Network Interface Card*) visto que assim no futuro caso seja necessário alterar os IP's destas máquinas esta *backend pool* irá atualizar automaticamente as configurações pois está conectada diretamente às NIC's.

Home > Load balancing > Create load balancer >

Add backend pool

Name * Backendpool01

Virtual network * Projeto-vnet (Projeto)

Backend Pool Configuration

- ☒ NIC
- ☐ IP Address

IP Version

- ☒ IPv4
- ☐ IPv6

Virtual machines

You can only attach virtual machines in northeurope that have a standard SKU public IP configuration. All IP configurations must be on the same virtual network.

+ Add - Remove

Virtual machine	IP Configuration	Availability
No virtual machines selected		

Information: You can only attach virtual machines that are in the same location and on the same virtual network. Virtual machines must have a standard SKU public IP or no public IP.

Filter by name... Location == northeurope

Virtual machine	Resource group	IP Configuration	Availability
VM02	Projeto	ipconfig1 (10.10.2.4)	-
VM03	Projeto	ipconfig1 (10.10.2.5)	-

Fig 6– Criação da backend pool.

Em seguida é necessário criar uma regra de *Load Balancing* começando primeiro por definir uma *Health Probe*, que irá testar se os serviços estão ativos nas máquinas do *backend* através da porta definida (neste caso a 3000). Será com esta *Health Probe* que em caso de falha ou erro de uma das máquinas irá detetar que esta porta 3000 não está acessível e que não deverá reencaminhar o tráfego dos utilizadores para esta máquina problemática.

Add health probe

Name * HealthProbe01 ✓

Protocol * TCP

Port * 3000 ✓

Interval * 5 seconds

Unhealthy threshold * 2 consecutive failures

Used by

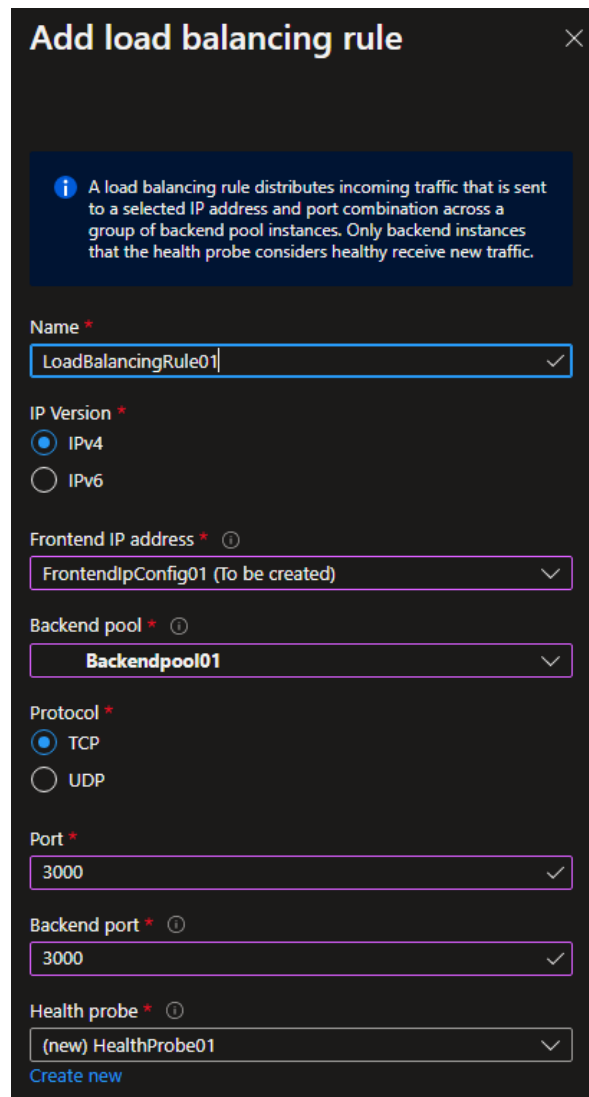
Not used

OK Cancel

Fig 7 – Criação da Health Probe.

Depois de criada a *Health Probe* é necessário também efetuar o resto da configuração desta regra de *Load Balancing* utilizando os componentes previamente criados como o IP de *Frontend* e a *Backend Pool* e adicionalmente definir a porta que será utilizada para a distribuição de serviço, tanto no *frontend* como no *backend* que será a 3000.

Esta será uma regra do tipo de *Inbound* pois o principal objetivo é o utilizador conseguir comunicar com o serviço que está publicado nos servidores.



The screenshot shows the 'Add load balancing rule' dialog box in the Azure Portal. It contains the following fields and options:

- Name ***: LoadBalancingRule01
- IP Version ***: IPv4 (selected), IPv6
- Frontend IP address ***: FrontendIpConfig01 (To be created)
- Backend pool ***: Backendpool01
- Protocol ***: TCP (selected), UDP
- Port ***: 3000
- Backend port ***: 3000
- Health probe ***: (new) HealthProbe01
- [Create new](#)

An informational message at the top states: 'A load balancing rule distributes incoming traffic that is sent to a selected IP address and port combination across a group of backend pool instances. Only backend instances that the health probe considers healthy receive new traffic.'

Fig 8– Criação da regra de Load Balancing.

Configuração Databases (Linux)

Para existir uma tolerância a falhas também do lado do servidor em vez de publicar a base de dados online, decidimos criar duas máquinas em Linux parecido com a explicação já anteriormente falada. Estas máquinas vão ter um IP privado, neste caso da terceira *Subnet* (10.10.3.0/24) que irá servir para a parte de MySQL.

Acedemos pela máquina VM01, através do programa *Putty*, e fizemos as seguintes configurações:

- Instalamos o *MySQL-Server* em ambas as máquinas em primeiro lugar;

```
db1:~$ sudo apt install mysql-server
```

Máquina nº1

- Acedemos ao *config* do *mysql* para mudar alguns dados, através do editor *Vi*, o **bind-address**, e o **log-bin** e **server-id** é necessário “descomentar” no final de mudar os dados introduzir (:wq!), para o ficheiro escrever, guardar e dar *quit* no ficheiro;

```
db1~$ sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf
```

```
user = mysql
```

```
bind-address = 0.0.0.0
```

```
mysqlx-bind-address = 127.0.0.1
```

```
key_buffer_size = 16M
```

```
myisam-recover-options = BACKUP
```

```
log_error = /var/log/mysql/error.log
```

```
server-id = 1
```

```
log_bin = /var/log/mysql/mysql-bin.log
```

```
max_binlog_size = 100M
```

- Necessário de seguida, dar um pequeno *restart* ao *MySQL-Server*;

```
db1:~$ sudo systemctl restart mysql
```

- É necessário fazer um User para ligar esta máquina à segunda máquina, temos de ter em conta o IP então da segunda;

```
db1:~$ sudo mysql -u root -p
```

```
mysql-db1> CREATE USER 'repl'@'10.10.3.5' IDENTIFIED BY 'secret';
```

```
mysql-db1> GRANT REPLICATION SLAVE ON *.* TO 'repl'@'10.10.3.5';
```

- O último passo nesta primeira máquina, é guardar alguns dados para usar depois mais a frente na configuração na segunda máquina, esses dados são o *File* e a *Position*;

```
db1> SHOW MASTER STATUS G
```

```
***** 1. row *****
```

```
File: mysql-bin.000001
```

```
Position: 946
```

```
Binlog_Do_DB:
```

```
Binlog_Ignore_DB:
```

```
Executed_Gtid_Set:
```

```
1 row in set (0.00 sec)
```

Máquina nº2

- Acedemos ao *config* do *mysql* para mudar alguns dados, através do editor Vi, o **bind-address**, e o **log-bin** e **server-id** é necessário “descomentar”, desta vez meter o ID a 2 e no final de mudar os dados introduzir (:wq!), para o ficheiro escrever, guardar e dar *quit* no ficheiro;

```
db2:~$ sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf

user = mysql

bind-address = 0.0.0.0

mysqlx-bind-address = 127.0.0.1

key_buffer_size = 16M

myisam-recover-options = BACKUP

log_error = /var/log/mysql/error.log

server-id = 2

log_bin = /var/log/mysql/mysql-bin.log

max_binlog_size = 100M
```

- Necessário de seguida, dar um pequeno *restart* ao *MySQL-Server*;

```
db2:~$ sudo systemctl restart mysql
```

- De seguida é necessário ter em conta como já referido aos dados que já tinham sido mostrados na primeira máquina, e executar o comando de “*CHANGE REPLICATION SOURCE*”;

```
db2:~$ sudo mysql -u root -p

mysql-db2> CHANGE REPLICATION SOURCE TO SOURCE_HOST='10.10.3.4',

        SOURCE_LOG_FILE='mysql-bin.000001',

        SOURCE_LOG_POS=946,

        SOURCE_SSL=1;

Query OK, 0 rows affected (0.02 sec)
```

- Finalmente conseguimos correr o comando de replica na segunda máquina;

```
mysql-db2> START REPLICATION USER='repl' PASSWORD='secret';
```

- Para verificar que os comandos e a própria configuração foi bem executada;

```
mysql-db2> SHOW REPLICATION STATUS G

***** 1. row *****

        Replica_IO_State: Waiting for master to send event

                Source_Host: 192.168.56.11

                Source_User: repl

                Source_Port: 3306

                Connect_Retry: 60

                Source_Log_File: mysql-bin.000001
```

- Para existir uma arquitetura de Ativo/Ativo e que ambas as bases de dados consigam-se atualizar uma à outra, é necessário criar também um *User* desta vez da segunda máquina para a primeira;

```
mysql-db2> CREATE USER 'repl'@'10.10.3.4' IDENTIFIED BY 'secret';
```

```
mysql-db2> GRANT REPLICATION SLAVE ON *.* TO 'repl'@'10.10.3.4';
```

- O último passo nesta segunda máquina, é guardar alguns dados para usar depois na configuração na primeira máquina, esses dados são o *File* e a *Position*;

```
mysql-db2> SHOW MASTER STATUS G
```

```
***** 1. row *****
```

```
File: mysql-bin.000001
```

```
Position: 904
```

```
Binlog_Do_DB:
```

```
Binlog_Ignore_DB:
```

```
Executed_Gtid_Set:
```

```
1 row in set (0.00 sec)
```

Máquina nº1

- De seguida é necessário ter em conta como já referido aos dados que já tinham sido mostrados na segunda máquina, e executar o comando de “*CHANGE REPLICATION SOURCE*”;

```
mysql-db1> CHANGE REPLICATION SOURCE TO SOURCE_HOST='10.10.3.5',  
  
        SOURCE_LOG_FILE='mysql-bin.000001',  
  
        SOURCE_LOG_POS=904,  
  
        SOURCE_SSL=1;
```

- Finalmente já temos esta configuração feita e para finalizar corremos esta configuração também na primeira máquina como já feito anteriormente na segunda;

```
mysql-db1> START REPLICA USER='repl' password='secret';
```

Load Balancer (MySQL)

Para existir uma facilidade na conexão no ficheiro de JavaScript de ambas as soluções, onde irá existir a conexão para estas máquinas, uma das formas de fazer para além da mudança destas máquinas durante uma “falha” bem como o uso apenas de um IP na conexão, decidimos então implementar um *Load Balancer* para as bases de dados.

Única preocupação na criação da mesma é trocar no *Health Probe* e na *Load Balancing Rule* o *port* para 3306, que é o *port default* utilizado pelo *MySQL*. Na *backend pool* é necessário seleccionar as máquinas neste caso as duas de *Linux* criadas para as bases de dados.

```
var mysql = require("mysql");
var util = require("util");

var pool = mysql.createPool({
  connectionLimit: 10,
  host: "10.10.3.10",
  user: "usercon",
  password: "secret",
  database: "mfa",
  port: 3306,
  multipleStatements: true,
});

pool.query = util.promisify(pool.query);

module.exports = pool;
```

Fig 9– *Connection.JS* (3000 e 3001).

Assim conseguimos ficar com uma arquitetura tolerante a falhas bem como ambas as bases de dados a comunicar entre si com a arquitetura (Ativa/Ativa ou Master/Master).



Fig 10– *Arquitetura BD's (Master-Master)*.

Alternativa de Implementação Azure (App Services)

Uma outra alternativa para implementação deste projeto em *Azure* seria utilizando *App Services*. Os *App Services* são uma plataforma como serviço (PaaS) especialmente desenhada para hospedar aplicações *Web* tendo várias ferramentas de suporte que facilitam a construção e manutenção destes aplicativos.

Criamos assim dois *App Services* um para a aplicação de *Login e Registo* e outro para a aplicação de Códigos, depois de desenvolver ambas as aplicações com ajuda de uma simples extensão do *VSCode* conseguimos dar um *deploy* dessas mesmas aplicações para o *App Service* respetivo.

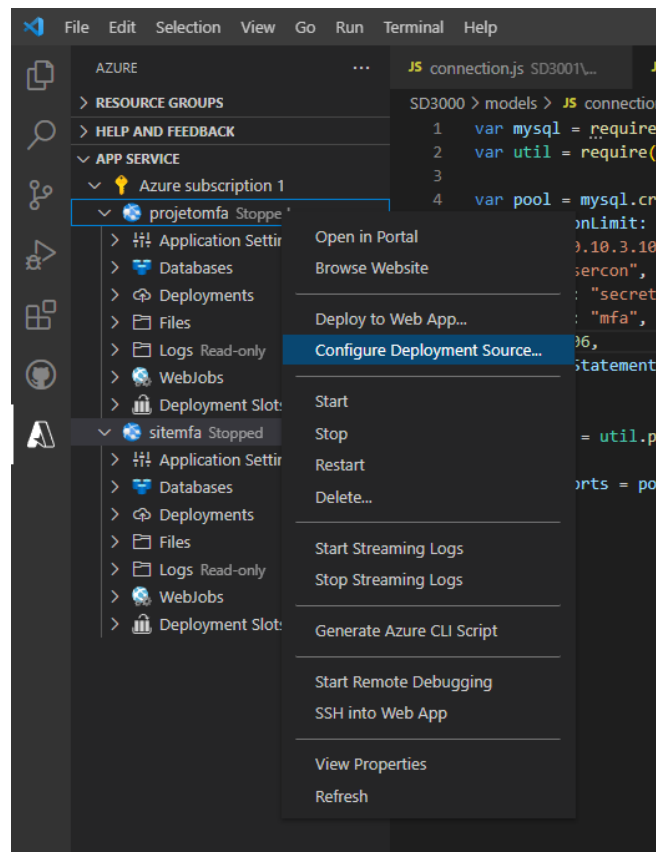
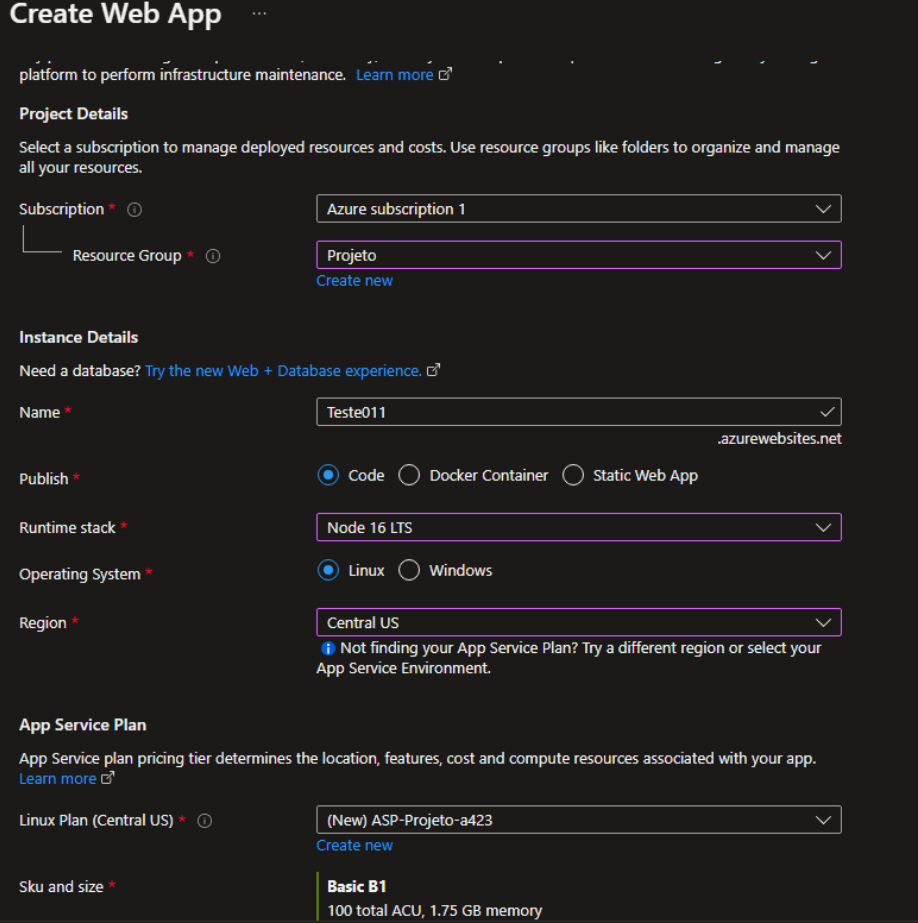


Fig 11– Deploy App Service.

Desta forma torna-se muito mais facilitado a gestão do código bem como a sua atualização e publicação no *App Service*, não tendo assim de implementar máquinas virtuais, ter de efetuar uma gestão do OS e *software* de suporte e toda a sua arquitetura.

Outra vantagem da utilização dos *App Services* é o facto de o *Azure* atribuir automaticamente *URL*'s a estas aplicações não sendo necessário aceder a estes “sites” pelos seus *IP*'s públicos.

Para criar um *App Service* através do portal do *Azure* é necessário atribuir-lhe um nome único que vai ser utilizado para o URL, escolher o software que vai “correr” o código, neste caso será o *Node*, também será necessário escolher qual o sistema operativo que este *App Service* terá e o plano associado à máquina que no nosso caso foi o plano free que disponibiliza uma hora diária.



Create Web App ...

platform to perform infrastructure maintenance. [Learn more](#) ↗

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ Azure subscription 1

Resource Group * ⓘ Projeto [Create new](#)

Instance Details

Need a database? [Try the new Web + Database experience.](#) ↗

Name * Teste011 ✓
.azurewebsites.net

Publish * ☒ Code ☐ Docker Container ☐ Static Web App

Runtime stack * Node 16 LTS

Operating System * ☒ Linux ☐ Windows

Region * Central US
ⓘ Not finding your App Service Plan? Try a different region or select your App Service Environment.

App Service Plan

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app. [Learn more](#) ↗

Linux Plan (Central US) * ⓘ (New) ASP-Projeto-a423 [Create new](#)

Sku and size * **Basic B1**
100 total ACU, 1.75 GB memory

Fig 12– Criação App Service

Depois de criado o *App Service*, esta é a sua interface onde conseguimos visualizar algumas métricas sobre a sua performance, o URL de acesso e o seu estado de funcionamento. Aqui também encontramos algumas definições da mesma, onde poderá ser configurado domínios específicos e garantir tanto a sua escalabilidade bem como a sua redundância em outras regiões utilizando o *Scale up* e o *Scale out*.

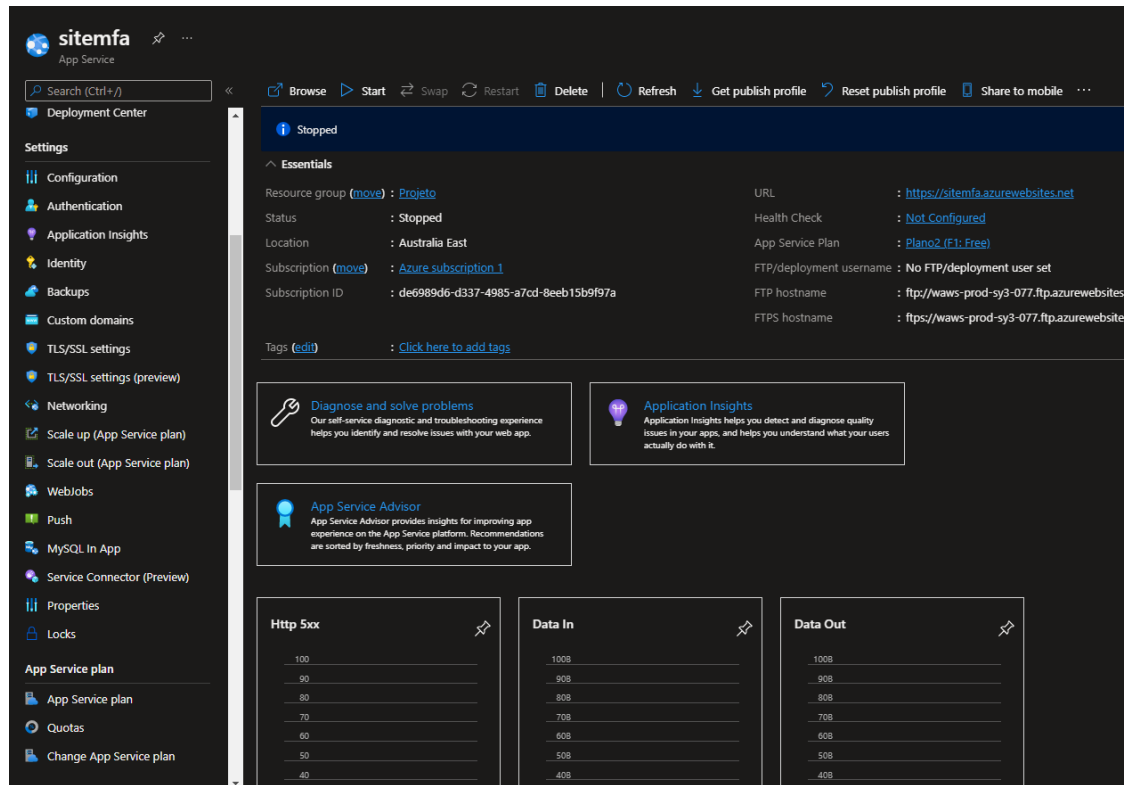


Fig 13–App Service (Sitemfa).

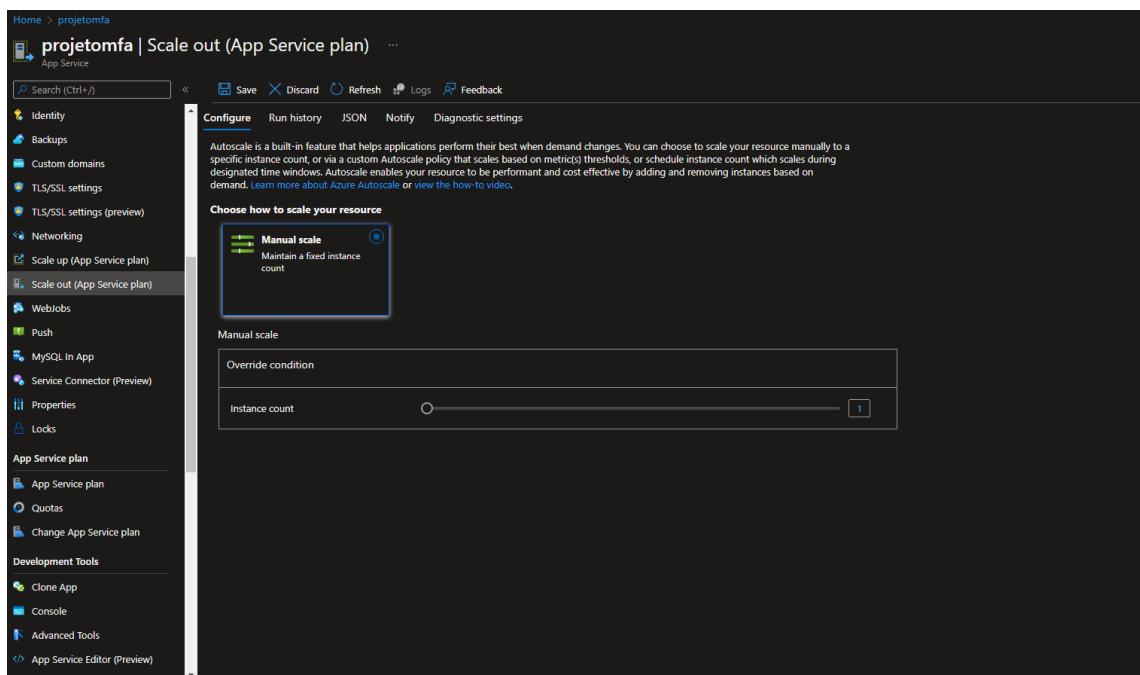


Fig 14– Scale out (App Service).