P.PORTO
INSTITUTO POLITECNICO DO PORTO Portugal

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

ASIST

SPRINT 2

Realizado por:

Diogo Ferreira, 1220829

Francisco Osório, 1220846

Sérgio Moreira, 1220890

Rafael Ferraz, 1221104

11/2024

Índice

Índice de Figuras

Índice de Tabelas

Divisão Tarefas

Tabela 1- Divisão das Tarefas

User Stories	Aluno
1	Francisco Osório
2	Diogo Ferreira
3	Rafael Ferraz
4	Sérgio Moreira
5	Francisco Osório
6	Diogo Ferreira
7	Rafael Ferraz
8	Sérgio Moreira

User Stories

User Story 1

Nesta *user story* era solicitado o *deployment* de um dos módulos do *RFP* numa *VM* do *DEI* e que este seja sistemático, validando de forma agendada com o plano de testes.

Assim, decidiu-se que seria o módulo *backend* a ser *deployment* na *VM* do *DEI*, devido a ser o módulo que contém as funcionalidades principais da aplicação.

Inicialmente criou-se uma máquina *Debian12* nos servidores do *DEI*, e configurou-se a máquina instalando-se o nano para criar *scripts* e instalar o *dotnet*. Posteriormente, criou-se um diretório designado por *"/root/sarm"* através do comando *"mkdir /root/sarm"*. Após termos o diretório criado, criou-se um script, dentro do diretório anterior, que será responsável pelo *deployment* do respetivo módulo, através do comando "*nano deployment_backend.sh*".

```
  GNU nano 7.2                                                deployment_backend.sh *
#!/bin/bash

# Parar o deploy anterior
echo "Parar o deploy anterior"
PID=$(fuser -n tcp 5001 2>/dev/null)

if [ -n "$PID" ]; then
  echo "Processo com o PID $pid está a utilizar a porta 5001. Terminar o processo $PID"
  kill -9 $PID
else
  echo "Nenhum processo está a correr na porta 5001."
fi

#Entrar no diretório do backend para realizar o deploy
echo "Entrar no diretório do backend para realizar o deploy"
cd ~/sarm/dddnetcore

#Build do projeto
echo "Build do projeto"
dotnet build

# Correr os testes
echo "Correr os testes"
dotnet test

# Redirecionar o tráfego TCP da porta 2224 para a porta 5001
echo "Redirecionar o tráfego TCP da porta 2224 para a porta 5001"
iptables -t nat -F PREROUTING
iptables -t nat -A PREROUTING -p tcp --dport 2224 -j REDIRECT --to-ports 5001

# Correr o backend na porta 5001 em segundo plano"
echo "Correr o backend na porta 5001"

LOG_FILE_DEPLOYMENT="../deployment_backend.log"

echo "-----------------------------------------------" >> $LOG_FILE_DEPLOYMENT
echo "Deployment backend realizado: $(date)" >> $LOG_FILE_DEPLOYMENT
echo " " >> $LOG_FILE_DEPLOYMENT
nohup dotnet run --urls "https://0.0.0.0:5001" > ~/sarm/deployment_backend_output.log 2>&1 &

echo "Deploy Realizado: $(date)"

# Terminar o script para indicar ao workflow para parar"
exit 0
```

Figura 1 - Script deployment_backend.sh

No *script* anterior, começa-se por verificar se existe algum processo a utilizar a porta 5001 (outro *deploy* anterior, por exemplo) através do comando "*fuser*", e caso exista através do comando "*kill -9*" terminamos esse processo para libertar a porta para o novo

deploy. Após isso entra-se no diretório que contém o *backend* (*"~sarm/dddnetcore"),* faz-se o *build* do módulo através do comando "*dotnet build",* e corre-se os testes com o comando "*dotnet test*". De seguida, redireciona-se o tráfego *TCP* da porta 2224 para a porta 5001, através do *iptables*, para ser possível aceder sem *vpn* á aplicação. Para ser possível visualizar quando são feitos *deploys* cria-se um diretório para os *logs* caso ele ainda não exista através do comando "*mkdir -p*", e guardando no ficheiro "*deployment_backend.log"* a data do *deploy*. Depois pelo comando "*nohup dotnet run --urls "https://0.0.0.0:5001" > ~/sarm/logs/deployment_backend_output.log 2>&1 &"*, executa-se o *backend* em segundo plano na porta 5001, redirecionando toda a informação da consola para o ficheiro "*deployment_backend_output.log"*. Por fim, através do *"exit 0"* terminamos o script, deixando o *backend* a correr em segundo plano.

Posteriormente, executou-se o comando "*chmod +x deployment_backend.sh*" para ser possível executar o script.

Depois de já temos o script que irá dar *deploy*, é preciso passar o conteúdo do módulo *backend* para a *VM* e para isso antes de tudo configurou-se umas *GitHub Secrets*, que será explicado mais á frente o intuito de cada uma.

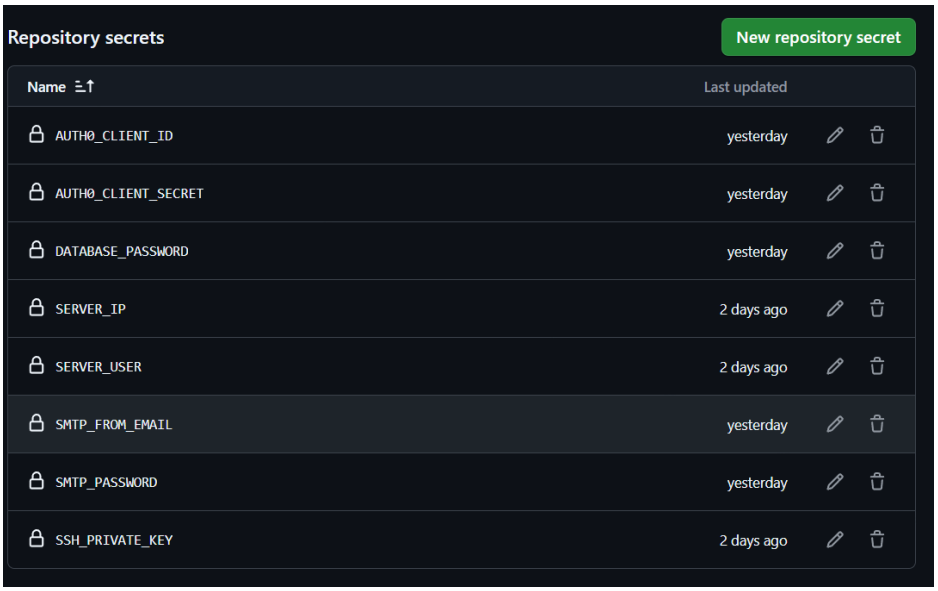

Figura 2 - Github Secrets

De seguida, criou-se um *workflow* designado "*deploy.yml*" (*Github Actions*) que irá automatizar o processo de *deployment* do *backend* na máquina virtual, e será acionado todos os sábados às 2 da manhã.

```yaml
name: Deploy Backend Module in VM DEI

on:
  schedule:
    - cron: '0 2 * * 6'

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Repository
        uses: actions/checkout@v3

      - name: 'Configuring a .env file for the backend'
        run: |
          cd dddnetcore
          touch .env
          echo "Auth0_Domain=dev-5hgod7guea48z3kl.us.auth0.com" >> .env
          echo "Auth0_Domain_URI=https://dev-5hgod7guea48z3kl.us.auth0.com" >> .env
          echo "Auth0_Audience=https://api.sarm" >> .env
          echo "Auth0_ClientId=${{ secrets.AUTH0_CLIENT_ID }}" >> .env
          echo "Auth0_ClientSecret=${{ secrets.AUTH0_CLIENT_SECRET }}" >> .env
          echo "Auth0_Connection=Username-Password-Authentication" >> .env
          echo "Auth0_Namespace_Roles=https://sarm.com" >> .env

          echo "Database_Name=sem5pi" >> .env
          echo "Database_HostName=vsgate-s1.dei.isep.ipp.pt" >> .env
          echo "Database_Port=11372" >> .env
          echo "Database_User=root" >> .env
          echo "Database_Password=${{ secrets.DATABASE_PASSWORD }}" >> .env

          echo "Smtp_Server=smtp.office365.com" >> .env
          echo "Smtp_Port=587" >> .env
          echo "Smtp_From_Email=${{ secrets.SMTP_FROM_EMAIL }}" >> .env
          echo "SMTP_PASSWORD=${{ secrets.SMTP_PASSWORD }}" >> .env

      - name: Upload Backend to VM DEI
        uses: appleboy/scp-action@v0.1.5
        with:
          host: ${{ secrets.SERVER_IP }}
          username: ${{ secrets.SERVER_USER }}
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          source: ./dddnetcore/*
          target: ~/sarm

      - name: Deploy Backend in VM DEI
        uses: appleboy/ssh-action@v0.1.5
        with:
          host: ${{ secrets.SERVER_IP }}
          username: ${{ secrets.SERVER_USER }}
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          script: |
            ~/sarm/deployment_backend.sh
```

Figura 3 - deploy.yml (workflow)

Explicando o *workflow* anterior, inicialmente utilizou-se o evento *schedule* com a expressão "*0 2 * * 6*" definindo assim que o workflow será apenas executado aos sábados às 2h da manhã. Após isso através "*actions/checkout@v3"* clona-se o repositório *GitHub* em um ambiente de execução, para ser possível aceder aos ficheiros do *backend*. Depois disso, configura-se um ficheiro *.env* para o *backend*, entrando-se no diretório *"dddnetcore"*, que é onde se encontra o *backend*. Esse ficheiro irá guardar as variáveis de ambiente necessárias para o funcionamento do *backend*, como configuração do *auth0*, da base de dados *MySQL* e do email *SMTP*. Na configuração deste ficheiro, foram utilizadas algumas das *secrets* que foram criadas anteriormente, como *"Auth0_ClientId", "Auth0_ClientSecret", "Database_Password", "Smtp_From_Email", "SMTP_PASSWORD"*, uma vez que consideradas são informações sensíveis. A próxima etapa refere-se a dar *upload* do *backend* para a *VM* utilizando a ação "*appleboy/scp-action@v0.1.5"* (utiliza protocolo *SCP*), definindo -se o *source* como "*dddnetcore*", pois é onde se encontra o código do *backend* e o *target* como "~/sarm", que foi o diretório criado inicialmente, sendo que o acesso à máquina é realizado via *SSH*, utilizando o *host, username, key*, da máquina que foram definidas nas *secrets* também *("SERVER_IP", "SERVER_USER", " SSH_PRIVATE_KEY"*). O último passo do *workflow* é executar o script que foi criado anteriormente, utilizando a mesma ação anterior (protocolo *SCP*), tendo acesso também via *SSH*, realizando assim o *deploy*.

Para testar, definiu-se um *schedule* diferente, para uma data e hora e obteve-se os seguintes resultados:

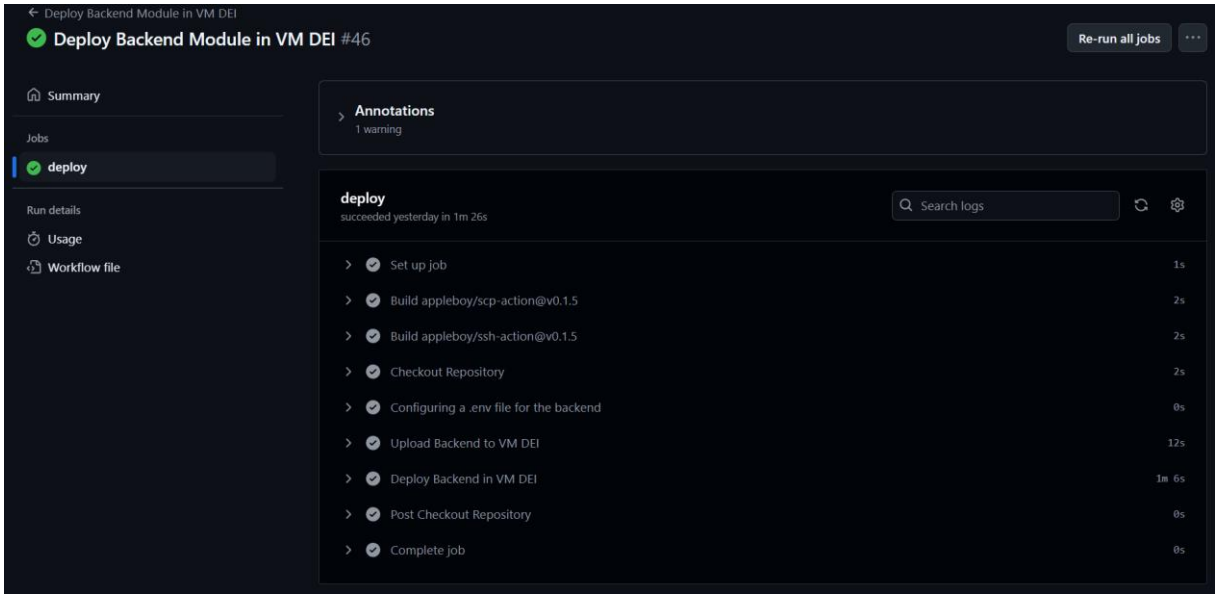

Figura 4 - Resultado do workflow

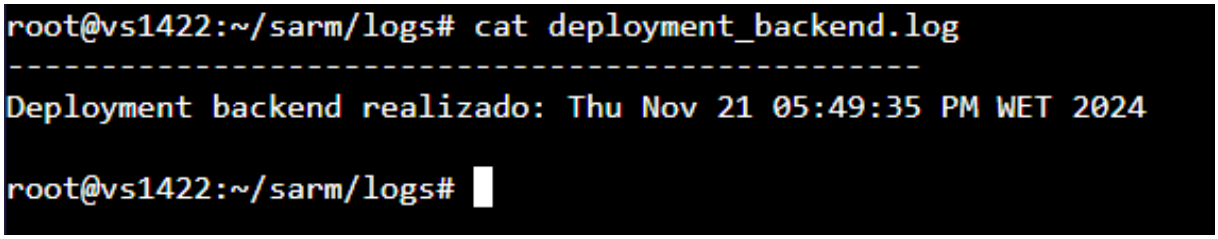

Figura 5 - Ficheiro deployment_backend.log com data do deploy

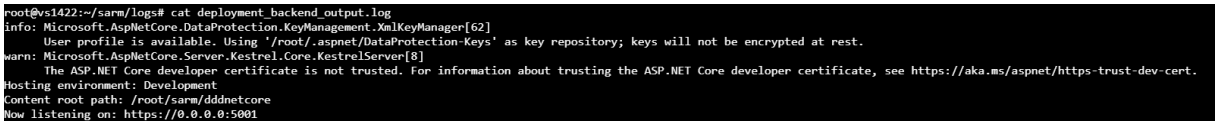

Figura 6 - Ficheiro deployment_backend_output.log com output do deploy

Figura 7 - Teste de uma rota do backend para verificar que ele estava a correr

User Story 2

Nesta *user story* era pedido que apenas os utilizadores da rede interna do DEI tivessem acesso à solução.

A solução foi desenvolvida na VM da *us* anterior.

De modo a restringir o acesso à rede interna do DEI temos de limitar o acesso aos endereços pertencentes a esta rede (e outros necessários à execução de outras tarefas).

Começamos por instalar o *iptables-persistent* de modo a guardar a solução mesmo em caso de reiniciação da máquina.

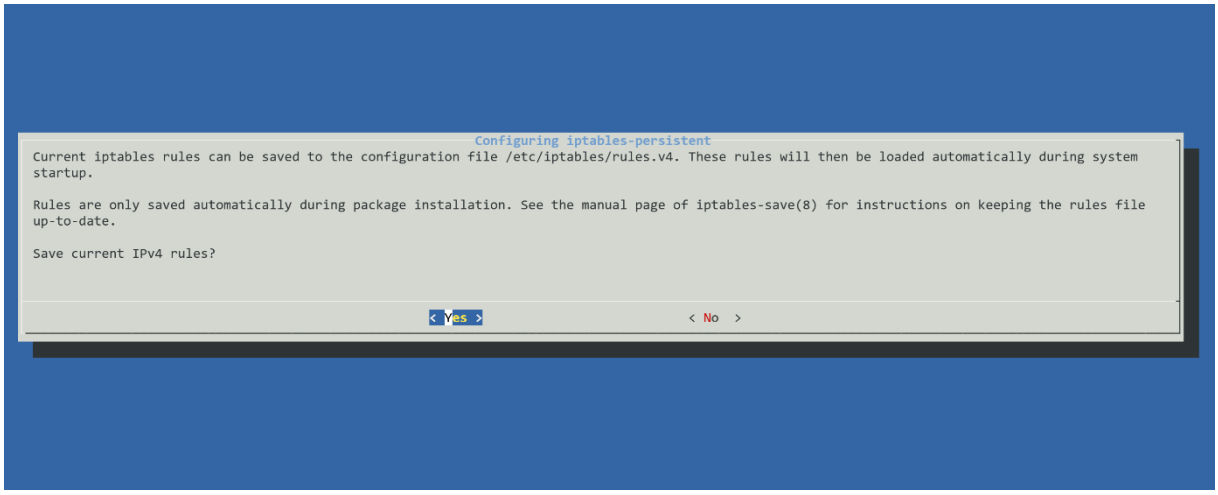

Figura 8 - Instalação iptables persistence

Para desenvolver a solução criamos um script com nome “myfirewall.sh” e demos permissões de execução.

```
root@vs1422:~# cd /root/config/firewall/
root@vs1422:~/config/firewall# touch firewall.sh
root@vs1422:~/config/firewall# chmod +x firewall.sh
root@vs1422:~/config/firewall# ls -l
total 0
-rwxr-xr-x 1 root root 0 Nov 21 18:58 firewall.sh
root@vs1422:~/config/firewall#
```

Figura 9 - Criação script para o firewall

Primeiramente começamos por limpar a regras de INPUT antigas.

Damos permissões *SSH* às ligações provenientes de *IPs* da rede interna do DEI, permissões a *IPs* do *GitHub*, para permitir o *Actions*, permissões à porta 80 e 443 para obter recursos do exterior, permissões para o *frontend*, permissões *loopback* e *icmp*.

Por fim definimos como política de INPUT DROP, para descartar tudo o que não faz parte da *INPUT chain*.

```
  GNU nano 7.2                                          config/firewall/firewall.sh *
#!/bin/bash

# PORTS:
# 22 -> SSH
# 5001 -> Frontend

iptables -F INPUT

#Accept SSH from DEI's internal Network
iptables -A INPUT -p tcp -s 10.8.0.0/16 --dport 22 -j ACCEPT

#Accept from GitHub
iptables -A INPUT -p tcp -m conntrack -i ens32 -s 185.199.108.0/22 --ctstate NEW,ESTABLISHED -j ACCEPT
iptables -A INPUT -p tcp -m conntrack -i ens32 -s 192.30.252.0/22 --ctstate NEW,ESTABLISHED -j ACCEPT
iptables -A INPUT -p tcp -m conntrack -i ens32 -s 140.82.112.0/20 --ctstate NEW,ESTABLISHED -j ACCEPT

#Accept HTTP port 80 and 443 to Install and Update
iptables -I INPUT -p tcp -m tcp -i ens32 --dport 80 -j ACCEPT
iptables -I INPUT -p tcp -m tcp -i ens32 --dport 443 -j ACCEPT

#Accept frontend
iptables -I INPUT -p tcp -m conntrack -i ens32 -s 10.8.0.0/16 --dport 5001 --ctstate NEW,ESTABLISHED -j ACCEPT

#Accept responses
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

#Accept icmp
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT

#Accept loopback
iptables -A INPUT -i lo -j ACCEPT

iptables -P INPUT DROP
```

Figura 10 - Script firewall.sh

Após correr o script é possível verificar as regras através do comando "*iptables -L*".

```
root@vs1422:~# ./config/firewall/firewall.sh
root@vs1422:~# iptables -L
# Warning: iptables-legacy tables present, use iptables-legacy to see them
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  10.8.0.0/16          anywhere             ctstate NEW,ESTABLISHED tcp dpt:5001
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:https
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
ACCEPT     tcp  --  10.8.0.0/16          anywhere             tcp dpt:ssh
ACCEPT     tcp  --  185.199.108.0/22     anywhere             ctstate NEW,ESTABLISHED
ACCEPT     tcp  --  192.30.252.0/22      anywhere             ctstate NEW,ESTABLISHED
ACCEPT     tcp  --  140.82.112.0/20      anywhere             ctstate NEW,ESTABLISHED
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
ACCEPT     icmp --  anywhere             anywhere             icmp echo-request
ACCEPT     all  --  anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Figura 11 - Verificação das regras

Para persistir as regras basta correr o comando "*netfilter-persistent save*".

User Story 3

Para esta *user story*, temos de fazer com que seja possível definir os clientes do *DEI*, simplesmente mudando um ficheiro de texto.

Para isso, começamos por criar o tal ficheiro de texto, que vai conter os endereços permitidos. Fazemos isso com o comando "*nano ips.txt*". Dentro do ficheiro colocamos o IP da máquina (linha 1) e o IP da rede interna do *DEI*, que os clientes usariam (linha 2):

Figura 12 - ips.txt

Em seguida, criamos um pequeno script que permite acesso aos clientes presentes no "*ips.txt*", usamos "*nano script_us6-4-3.sh*" para criar o ficheiro, que contem um ciclo *for* para cada *ip* encontrado em "*ips.txt*", adiciona uma regra no *iptables* que permite tráfego TCP na porta 22, que corresponde a *SSH*. *Finalmente, é adicionada uma regra que bloqueia o trafico de outras ligações*:

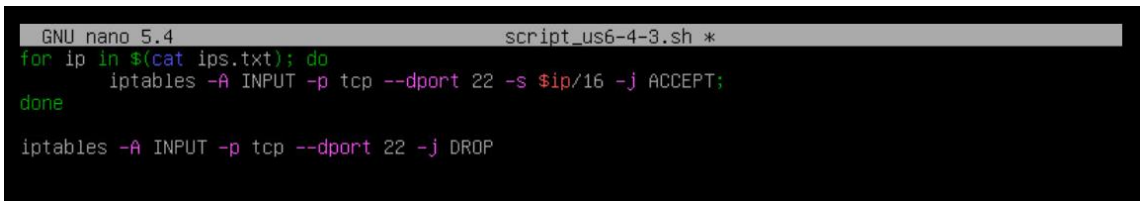

Figura 13 - Script para as iptables

Não nos podemos esquecer de dar permissão de execução ao ficheiro com o comando "*chmod +x script_us6-4-3.sh*". Agora, corremos o script com "*./script_us6-4-3.sh*". Podemos verificar que o script fez o pretendido:

Figura 14 - iptables depois de correr o script

Por último, devemos guardar as mudanças de modo a se manterem mesmo após reiniciar o sistema. Para isso, corremos "*sudo netfilter-persistent save*"

```
root@uvm020:~# sudo netfilter-persistent save
run-parts: executing /usr/share/netfilter-persistent/plugins.d/15-ip4tables save
run-parts: executing /usr/share/netfilter-persistent/plugins.d/25-ip6tables save
```

Figura 15- Guardar permanentemente os iptables

Como podemos ver abaixo, após reiniciar o sistema as configurações ainda estão presentes:

```
 ------------------------------------------------------------------------------
|                              ASIST UVM 20                                    |
 ------------------------------------------------------------------------------

Welcome root

Date: Tuesday, November 12, 2024
Hours: 07:53:00

root@uvm020:~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  10.9.0.0/16          anywhere             tcp dpt:ssh
ACCEPT     tcp  --  10.8.0.0/16          anywhere             tcp dpt:ssh
DROP       tcp  --  anywhere             anywhere             tcp dpt:ssh

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Figura 16 - iptables depois de reiniciar o sistema

User Story 4

O RA (*Risk Assessment*) é constituído por cenários que podem afetar a continuidade do negócio, bem como a probabilidade de tal item acontecer e o seu impacto. O RA é representado por uma matriz, onde cada cenário é associado a uma probabilidade e a uma severidade e o risco correspondem à multiplicação destes dois componentes, isto é, esta matriz é uma ferramenta que permite qualificar visualmente os riscos que devem ter mais atenção.

$$Risco = Impacto * Probabilidade$$

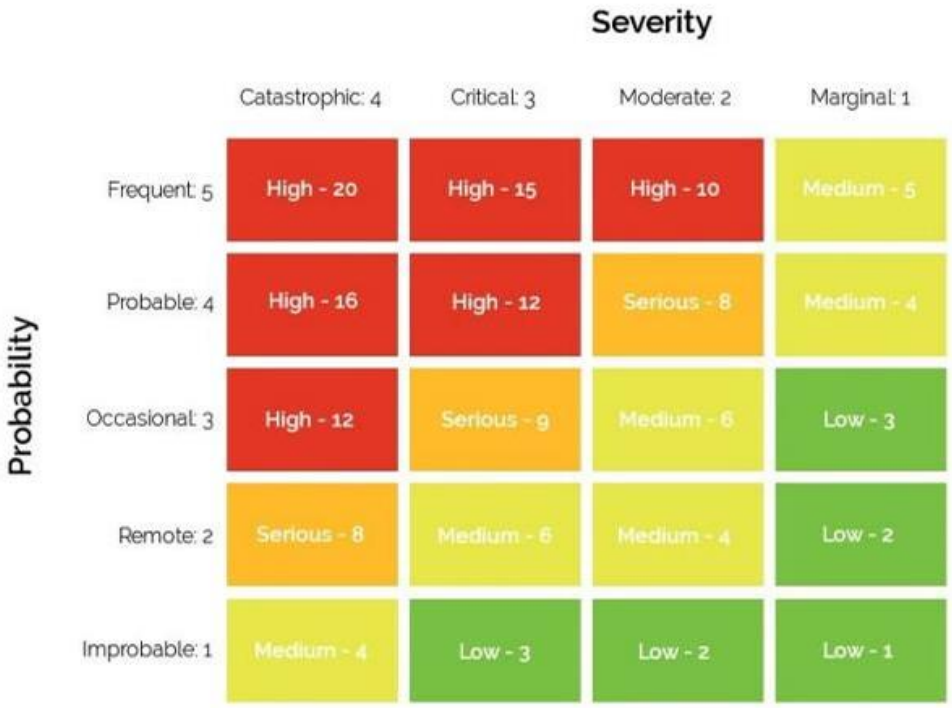

Figura 17 - Matriz de Riscos (Fonte: slides)

Tabela 2 - Riscos

Ameaça	Probabilidade	Consequência	Risco
Exposição de Dados Sensíveis	2	4	8
Acesso indevido por falta de autenticação	2	4	8
Perda da Base de Dados	3	5	15
Falha de conexão à internet	4	3	12
Falha de eletricidade	4	3	12

Falhas de Hardware e Software	2	2	4

User Story 5

Nesta user story era pedido que se defini-se o *MBCO (Minimum Business Continuity Objective)* a propor aos stakeholders.

O *Mininum Business Continuity Objective (MBCO)* especifica o nível mínimo de operacionalidade que deve ser mantida durante uma disrupção na infraestrutura.

A aplicação que está a ser desenvolvida encontra-se divida em vários módulos: o *backend*, o *frontend*, incluindo dentro o *3d visualization*, e o *planning*. Sendo que cada módulo tem uma função distinta e consequentemente maior importância, é necessário ter em atenção ao *Maximum Tolerable Downtime (MTD)* e ao *Maximum Tolerable Period of Disruption (MTPD)* para cada um.

Começando pelo módulo de *backend*, este é o responsável por processar e armazenar os dados, permitindo a gestão de pacientes, staffs, tipos de cirurgias, pedidos de cirurgias e marcação destas, sendo assim considerado o core da nossa aplicação. Desta forma, foi definido que o *MTD* para este módulo seria de 1 hora, e após esse tempo este módulo apresentará funcionalidades mínimas como gestão de pacientes, pedidos de cirurgias e suas marcações até ao *MTPD,* que foi definido para 12 horas.

Passando agora para o módulo de *frontend*, este é o que permite aos utilizadores interagirem com o sistema e realizarem funcionalidades como gestão de pacientes, staffs, tipos de cirurgias, pedidos de cirurgias e respetivas marcações, funcionalidades baseadas no módulo *backend*. Sendo assim, o *frontend* é considerado um módulo essencial para a continuidade das operações da aplicação. Assim, como no *backend*, definiu-se também um *MTD* de 1 hora. Após esse tempo, funcionalidades como gestão de pacientes, pedidos de cirurgias e suas marcações, estarão disponíveis, podendo estar outras ainda indisponíveis. Para recuperar todas as funcionalidades foi definido um período também de 12 horas, ou seja, para o *MTPD*.

O módulo *3d visualization* encontra-se integrado no *frontend*, e este permite aos utilizadores uma visão em tempo real do hospital, sendo possível ver os quartos e as respetivas operações que se encontram a decorrer. Apesar deste módulo ser útil, é secundário em relação aos módulos anteriores. Desta forma, foi definido um *MTD* de 6 horas, sendo que após este já deve ser possível os utilizadores verem um layout simplificado com o status de ocupação dos quartos, ainda com poucos detalhes. Já o *MTPD* foi definido para 48 horas, permitindo que este seja restaurado integralmente dentro desse prazo.

Por fim, o módulo *planning* é o que é responsável por gerar o agendamento das cirurgias e otimizar estes horários de acordo com diferentes critérios e disponibilidade de profissionais médicos e quartos. Este módulo também tem a sua importância, como a otimização, apesar disso, pode também ser considerado secundários, pois é possível realizar agendamentos manuais temporariamente. Assim, determinou-se que o tempo

de inatividade deste módulo será de 3 horas, ou seja, o *MTD*. Após o *MTD* e até o *MTPD* a sua funcionalidade mínima, permitirá o agendamento básico de cirurgias, deixando otimização automática desativada temporariamente, sendo que o definiu-se o *MTPD* para 24 horas.

Concluindo, garante-se que as funcionalidades críticas da aplicação, continuem disponíveis, com tolerâncias razoáveis para restabelecimento de cada módulo em caso de disrupção, minimizando assim os impactos.

User Story 6

Nesta *user story* era pedido que seja proposta, justificada e implementada uma estratégia de cópia de segurança que minimize o RTO e o WRT.

Decidimos então aplicar uma estratégia em que realizamos uma cópia completa aos domingos e cópias incrementais nos restantes dias. As cópias acontecem às 2:30 de modo aproveitar um horário de menor utilização. Esta estratégia oferece um equilíbrio entre eficiência e tempo de recuperação.

Começamos por criar um script com o nome "*backup.sh*" e demos permissão de execução.

```
root@vs1422:~# cd config/
root@vs1422:~/config# ls
firewall
root@vs1422:~/config# mkdir backup
root@vs1422:~/config# cd backup/
root@vs1422:~/config/backup# touch backup.sh
root@vs1422:~/config/backup# chmod +x backup.sh
```

Figura 18 - Criação do script de backup e permissões

O *script* faz *backup* à base de dados, completa aos domingos e incremental nos restantes dias. Também aos domingos os scripts da semana vão para a pasta "*old*".

```
 GNU nano 7.2                                    config/backup/backup.sh
#!/bin/bash

USER="root"
PASSWORD="Wfd1oionqJkp"
DB_NAME="sem5pi"
HOST="vsgate-s1.dei.isep.ipp.pt"
PORT="11372"
DIR_BACKUP="/backup_script/database"
DIR_BACKUP_OLD="$DIR_BACKUP/old"
DATE=$(date '+%Y-%m-%d_%H-%M-%S')
TODAY=$(date '+%u')

mkdir -p "$DIR_BACKUP"
mkdir -p "$DIR_BACKUP_OLD"

# Arquivo que rastreia a última execução de backup
LAST_BACKUP_TIME_FILE="$DIR_BACKUP/last_backup_time.txt"

# Verifica o último horário de backup ou define um padrão inicial
if [ ! -f "$LAST_BACKUP_TIME_FILE" ]; then
    LAST_BACKUP_TIME="1970-01-01 00:00:00"
else
    LAST_BACKUP_TIME=$(cat "$LAST_BACKUP_TIME_FILE")
fi

# Backup completo aos domingos
if [ "$TODAY" -eq 7 ]; then
    echo "A Criar backup completo..."

    # Backup completo de todas as tabelas
    mydumper -h $HOST --port=$PORT -u $USER -p$PASSWORD $DB_NAME > "$DIR_BACKUP/full_backup_${DATE}.sql"

    # Compactar backup completo
    tar -czf "$DIR_BACKUP/full_backup_${DATE}.tar.gz" -C "$DIR_BACKUP" "full_backup_${DATE}.sql"
    rm -rf "$DIR_BACKUP_OLD"/*
```

Figura 19 - Primeira parte do script backup.sh

```
    # Compactar backup completo
    tar -czf "$DIR_BACKUP/full_backup_${DATE}.tar.gz" -C "$DIR_BACKUP" "full_backup_${DATE}.sql"
    rm -rf "$DIR_BACKUP_OLD"/*
    mv "$DIR_BACKUP"/*.tar.gz "$DIR_BACKUP_OLD/"

    # Atualizar o último horário de backup
    echo "$(date '+%Y-%m-%d %H:%M:%S')" > "$LAST_BACKUP_TIME_FILE"

    echo "Backup completo movido para $DIR_BACKUP_OLD."
else
    echo "A criar backup incremental desde $LAST_BACKUP_TIME..."

    # Backup incremental apenas de registros alterados
    TABLES=$(mysql -h $HOST --port=$PORT -u $USER -p$PASSWORD -D $DB_NAME -e "SHOW TABLES;" | tail -n +2)
    for TABLE in $TABLES; do
        # Adicionar condição de data para capturar registros modificados
        QUERY="SELECT * FROM $TABLE WHERE updated_at > '$LAST_BACKUP_TIME' OR created_at > '$LAST_BACKUP_TIME';"
        mysql -h $HOST --port=$PORT -u $USER -p$PASSWORD -D $DB_NAME -e "$QUERY" > "$DIR_BACKUP/incremental_${TABLE}_${DATE}.sql"
    done

    # Compactar backups incrementais
    tar -czf "$DIR_BACKUP/incremental_backup_${DATE}.tar.gz" -C "$DIR_BACKUP" "incremental_*.sql"
    rm "$DIR_BACKUP/incremental_*.sql"

    # Atualizar o último horário de backup
    echo "$(date '+%Y-%m-%d %H:%M:%S')" > "$LAST_BACKUP_TIME_FILE"

    echo "Backup incremental concluído: $DIR_BACKUP/incremental_backup_${DATE}.tar.gz"
fi
```

Figura 20 - Segunda parte do script de backup.sh

```
root@vs1422:~# ./config/backup/backup.sh
A Criar backup completo...
```

Figura 21 - Verificação do funcionamento do script

Para executar o script diariamente utilizamos o *crontab*.

```
  GNU nano 7.2                                    /tmp/crontab.Sr0kTi/crontab *
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command

30 2 * * * /root/config/backup/backup.sh
```

Figura 22 - Definição do crontab

User Story 7

Nesta US era-nos pedido que se define uma pasta pública para todos os utilizadores registados no sistema, de modo a poderem ler o que lá está.

Primeiro, precisamos de criar a tal pasta, que depois será configurada como pública. Para isto usamos o comando "*mkdir*":

```
root@uvm020:~# mkdir /shared
```

Figura 23 - Criar a pasta partilhada

Em seguida, precisamos de mudar as permissões da pasta, o objetivo é que todos tenham permissão de leitura, para isso, temos de usar o comando "*chmod -R a+r /shared"*:

```
root@uvm020:~# chmod -R a+r /shared
```

Figura 24 - Dar permissão de leitura para ficheiros na pasta partilhada

O comando "*chmod*" é usado para mudar as permissões de acesso, leitura ou execução. "*-R*" muda as permissões recursivamente (a tudo o que se encontra dentro do diretório), "*a+r*" indica que todos (a) devem receber (+) permissão de leitura (r), e depois metemos a nossa pasta. Podemos rapidamente ver se as permissões estão a funcionar corretamente criando um ficheiro de texto e correndo o comando "*ls -l*":

```
root@uvm020:~# nano /shared/teste.txt
```

Figura 25 - Criar um ficheiro de teste

```
root@uvm020:/# cd /shared
root@uvm020:/shared# ls -l
total 4
-rw-r--r-- 1 root root 9 Nov  5 07:12 teste.txt
```

Figura 26 - Ficheiro de teste com permissão de leitura para todos

O ficheiro criado tem permissões de leitura e escrita para o dono, e apenas de leitura para o grupo e para todos, tornando isto uma pasta que todos podem aceder.

User Story 8

Nesta *user story* é necessário analisar o ficheiro /var/log/auth.log pois apresenta os *logs* de autenticação dos utilizadores, por quem é o estado de sucesso do acesso.

Figura 27 - Ficheiro /var/log/auth.log

Para melhorar o processo de analise de identificação de utilizadores, criamos um Shell script com o nome "*acessos_incorretos.sh*" que efetua a filtragem de modo a contar o número de vezes que cada utilizador falhou a password incrementando mais 1 o número de logins falhados pelo utilizador.

Se um utilizador exceder o número limite de logins errados ele ira escrever na Shell o usuário e o número de acessos incorretos.

```
 GNU nano 5.4                          acessos_incorretos.sh
#!/bin/bash

declare -A user_counts

while read -r line; do
        if [[ $line == *"Failed password"* ]]; then
                user=$(echo "$line" | awk '{print $9}')
                ((user_counts[$user]++))
        fi

done < /var/log/auth.log

echo " User     | Numero De Acessos"
echo "-------------------------"

for user in $(echo "${!user_counts[*]}"); do
        count="${user_counts[$user]}"
        if [ "$count" > 3 ]; then
                printf "%-8s  | %d\n" "$user" "$count"
        fi
done
```

Figura 28 - Script accessos_incorretos.sh

Agora com o utilizador luser6 iremos realizar o login com a palavra-passe que seja incorreta mais de 3 vezes e com o utilizador luser5 iremos realizar 2 tentativas de login de modo a comprovar que o script funciona.

```
C:\Users\sergi>ssh luser6@uvm020
The authenticity of host 'uvm020 (10.9.10.20)' can't be established.
ED25519 key fingerprint is SHA256:VHQ89VofvEp3IoSsOc88WbttRKLWEI5lXcr1dd/Hhc
I.
This host key is known by the following other names/addresses:
    C:\Users\sergi/.ssh/known_hosts:1: 10.9.10.20
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'uvm020' (ED25519) to the list of known hosts.
------------------------------------------------------------------------
|                          ASIST-24-25-UVM 20                          |

------------------------------------------------------------------------
Welcome!
This is our system uvm20!
luser6@uvm020's password:
Permission denied, please try again.
luser6@uvm020's password:
Permission denied, please try again.
luser6@uvm020's password:
luser6@uvm020: Permission denied (publickey,password).

C:\Users\sergi>ssh luser6@uvm020
------------------------------------------------------------------------
|                          ASIST-24-25-UVM 20                          |

------------------------------------------------------------------------
Welcome!
This is our system uvm20!
luser6@uvm020's password:
Permission denied, please try again.
luser6@uvm020's password:
Permission denied, please try again.
luser6@uvm020's password:
luser6@uvm020: Permission denied (publickey,password).
```

Figura 29 - Tentativas login luser6

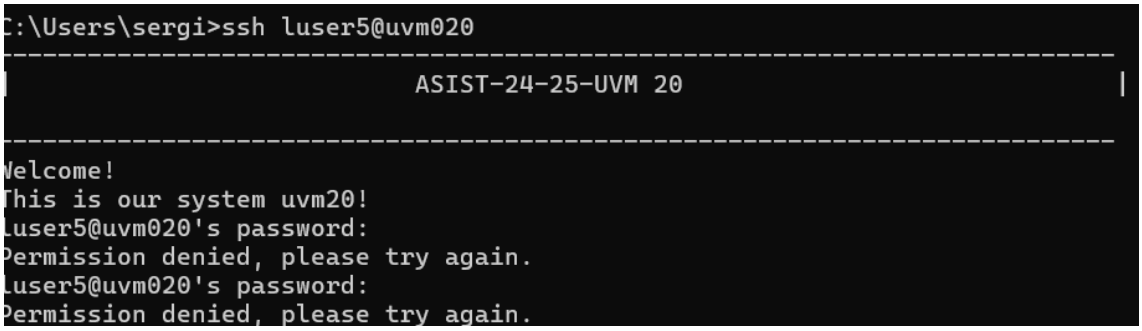

Figura 30 - Tentativas login luser5

Após estes passos iremos executar o ficheiro *acessos_incorretos.sh* e verificar que o luser6 errou 6 vezes o login e o luser5 errou 2 vezes por isso no ficheiro é apresentado só o luser6 com 6 login incorretos.

Figura 31 -Execução do script acessos_incorretos.sh