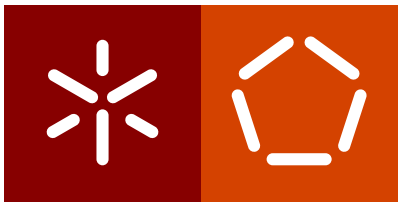
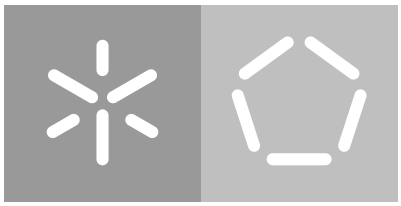


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Tiago Manuel Gonçalves Lameira

IntelliScaling
Serviço de gestão inteligente de capacidade

Maio 2022



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Tiago Manuel Gonçalves Lameira

IntelliScaling
Serviço de gestão inteligente de capacidade

Master in Informatics Engineering

Dissertation supervised by
José Orlando Pereira
Diogo Silva

Maio 2022

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

AGRADECIMENTOS

Nunca desconfiei do quão enriquecedora seria para a minha carreira pessoal e profissional a experiência de frequentar o prestigiado *Mestrado em Engenharia Informática (MEI)* de Engenharia Informática da Universidade do Minho.

Neste capítulo, queria em primeiro lugar agradecer ao Senhor Professor José Orlando Pereira por ter aceite o convite de orientador da minha dissertação, por toda disponibilidade, paciência, partilha de ideias e por toda a ajuda prestada ao longo da construção desta dissertação.

Tenho que agradecer ao meu supervisor da empresa Diogo Silva, que apesar do pouco tempo disponível, foi um verdadeiro líder, muito preocupado, atencioso e sempre disponível para ajudar na partilha de ideias e em algumas questões mais técnicas relacionadas projeto. Orgulho-me inclusive de ter tomado a decisão de concorrer à equipa de *DevOps* da 360imprimir, que apesar da distância que nos separava, houve sempre um excelente ambiente de respeito, entreajuda e camaradagem.

Dessa equipa destaco também os meus colegas Gustavo Balbino e Rui Matos, que me acolheram da melhor maneira possível, estiveram sempre disponíveis e me transmitiram muitos conhecimentos. Senti que fui muito bem recebido e integrado na equipa onde me deram a responsabilidade de também contribuir com meus conhecimentos na resolução de problemas da empresa, como por exemplo tickets de *LiveOps* e *DevOps* e evoluir nesta área de infraestrutura.

Não tenho palavras para agradecer aos meus pais pela sua postura e por toda a ajuda que deram, pela coragem e incentivo e pelas palavras certas nas horas certas.

Aos meus colegas de curso Pedro Silva, Diogo Fernandes e Daniel Silva pela ajuda e momentos bem passados durante a realização de trabalhos grupo. A todos os que mencionei e a todos os outros que me ajudaram mas que aqui não mencionei, o meu muito obrigado!

> "Aqueles que passam por nós não vão sós. Deixam um pouco de si, levam um pouco de nós."
>
> *Antoine de Saint-Exupery*

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração. Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

ABSTRACT

Cloud computing means using various computing resources shared by a *Cloud Solution Provider (CSP)*, as an alternative to using local servers. This model has been increasingly adopted by technology companies due to the ease of service management, from a mobile device with internet access.

One of the three service levels offered by the cloud corresponds to the *Infraestructure as a Service (IaaS)* layer. This model makes hardware resources (storage, network and computing) available to consumers for a rental period. As the use of these resources is usually charged per hour, an inefficient use implies some unnecessary costs in case the necessary benefits are not taken advantage of.

Many CSP, such as *Azure, Aws, Google Cloud Platform*, provide native applications responsible for automatically scaling cloud resources however, they may not be adapted to the reality of an organization if infrastructures have external cloud services, as in the case of *360imprimir*.

This dissertation consists of the development of an application, which taking advantage of the elasticity of the cloud, manages *360imprimir* cloud computing resources.

keywords: Nuvem, Cloud computing, Autoscalling, Dimensionamento, Cloud Service Provider, Azure, Infraestructure as a Code, Infraestructure as a Service, Balanceador de carga, Monitorização, Automação, Orquestração, Zabbix, HAProxy, ASP.NET Core (C#), Terraform, Azure DevOps

RESUMO

A *computação em nuvem* é um conceito que consiste no uso de vários recursos de computação partilhados por um CSP, em alternativa do uso de servidores locais. Este modelo tem sido cada vez mais adoptado pelas empresas tecnológicas devido à facilidade de gestão dos serviços, a partir de um dispositivo móvel com acesso à internet.

Um dos três níveis de serviço oferecidos pela nuvem corresponde a camada Infraestrutura como serviço IaaS, que consiste na disponibilização de recursos de hardware (armazenamento, rede e computação) para uma comunidade de consumidores por um período de aluguer. Como o uso destes recursos é normalmente cobrado por hora, uma ineficiente utilização implica alguns custos desnecessários caso não se tire o devido proveito em função das necessidades.

Muitos sistemas CSP, como o caso do *Azure, Aws, Google Cloud Platform*, disponibilizam aplicações nativas responsáveis pelo dimensionamento automático de recursos da nuvem no entanto, estas podem não se adaptar em concreto à realidade de uma organização,como por exemplo a *360imprimir*, que possui uma infraestrutura diversificada que utiliza serviços e tecnologias que não fazem parte do ecossistema da nuvem.

Esta dissertação consiste no desenvolvimento de uma aplicação, que tirando partido da elasticidade da nuvem, tem como objetivo gerir recursos de infraestrutura de nuvem da *360imprimir*.

Palavras-chave: Nuvem, Cloud computing, Autoscalling, Dimensionamento, Cloud Service Provider, Azure, Infraestructure as a Code, Infraestructure as a Service, Balanceador de carga, Monitorização, Automação, Orquestração, Zabbix, HAProxy, ASP.NET Core (C#), Terraform, Azure DevOps

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

A

API Application-Programming Interface.

C

CICD Continuous integration and delivery.

CLI Command-Line Interface.

CRUD Create, Read, Update, Delete.

CSP Cloud Solution Provider.

D

DNS Domain Name System.

DSL Custom Domain-Specific-Language.

DTO Data Transfer Object.

G

GUI Graphical User Interface.

H

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

I

IAAS Infraestructure as a Service.

IAC Infraestructure as a Code.

IIS Internet Information Services.

M

MEI Mestrado em Engenharia Informática.

R

REST Representational state transfer.

S

SAAS Software as a Service.

SDK Software Development Kit.

SSH Secure Shell.

T

TCP Transmission Control Protocol.

1

INTRODUÇÃO

A *Binary Subject*, também conhecida como *360imprimir*, é uma média empresa com sede em Torres Vedras e instalações em Lisboa e Braga. O seu principal foco consiste na promoção, simplificação e acesso a preços competitivos, a produtos e serviços de marketing através da plataforma de *e-commerce* online *"www.360imprimir.pt"*, destinada a micro e pequenas empresas, pequenos *designers*, profissionais liberais e particulares.

Neste momento a plataforma providencia soluções de marketing para 21 mercados do mundo (Portugal, Espanha, Reino Unido, Irlanda, França, Bélgica, Suiça, Alemanha, Áustria, Holanda, Itália, Dinamarca, Suécia, Finlândia, Noruega, República Checa, Polónia, Brasil, México e América do Norte), existindo um domínio diferente para cada mercado.

Existe uma grande infraestrutura tecnológica para disponibilizar o site que suporta todo este modelo de negócio nestes mercados, proporcionando uma plataforma ativa e disponível 24 horas por dia de forma a garantir elevada disponibilidade da aplicação. Convém salientar que uma falha nos servidores de produção implica perda de credibilidade e de potenciais clientes, que pode acarretar uma diminuição de vendas.

A nuvem é uma peça fundamental no funcionamento da empresa pois todos os servidores responsáveis pelo funcionamento do site nos vários mercados, desde os de produção, os de teste até aos de ambientes de equipa, se encontram lá alojados.

A existência de tantos servidores e serviços a correr em paralelo, acarreta um grande custo financeiro no final de cada mês.

Uma proposta recorrente para minimizar alguns custos consiste em fazer o *encerramento automático* de algumas máquinas que não necessitem de estar ligadas 24 horas por dia, como é o caso de servidores de ambientes de equipa e de testes. Já nos servidores de produção, se a taxa de utilização for baixa, tira-se proveito da elasticidade das nuvens, e faz-se um *Scale Down* manual das instâncias via *Azure Portal* de forma a reduzir os consumos indevidos. Todavia, essa configuração manual não é uma abordagem muito prática porque obriga a uma análise constante da infraestrutura por parte do administrador de sistemas.

Apesar dos vários CSP como a *AWS*, *Azure*, *Google Cloud Plataform* disponibilizarem aplicações de dimensionamento automático, essas não têm o poder de controlar alguns serviços externos utilizados pela infraestrutura da 360imprimir, como o caso dos servidores de balanceamento de carga e dos servidores de monitorização. A adaptação da aplicação *Azure*

AutoScale da nuvem, obrigaria a uma restruturação dos recursos de IaaS existentes na infraestrutura da 360imprimir para converter máquinas virtuais em instâncias do *Virtualmachine Scale Set*, para só para possuirem a capacidade de realizar operações de dimensionamento automaticamente.

Este projeto de dissertação permite automatizar a gestão da infraestrutura em nuvem, que consiste na monitorização atenta e intervenção cuidada por parte dos técnicos para criar, apagar e alterar o tamanho das máquinas virtuais em função da carga de trabalho que estas recebem. Com isto, a aplicação minimizará a necessidade de intervenção humana que assim permite desalocar intervenientes para outros projetos e anular possíveis causas de erros humanos na gestão da infraestrutura.

A adaptação desta aplicação permitirá economizar o valor gasto nas subscrições da nuvem, porque para cada máquina virtual será definida a melhor configuração em função da carga de trabalho recebida. Isto significa que caso o tamanho de cada máquina virtual seja demasiado grande em função da carga recebida, a aplicação terá a capacidade de diminuir o tamanho das máquinas virtuais para não estar a consumir em demasia.

Além disso, existem situações ao longo do ano (como por exemplo *Black Friday* ou em campanhas de lançamento de novos produtos) em que o site tem maior afluência de utilizadores e as máquinas poderão não estar preparadas para aguentar tantos pedidos. Nesse sentido pretende-se que a aplicação prepare a infraestrutura para suportar um grande número de pedidos.

1.1 OBJETIVO

O objetivo desta dissertação consiste em criar uma aplicação de gestão designada por *IntelliScaling* concebida para gerir máquinas virtuais existentes em nuvem e também os vários serviços ligados a uma infraestrutura, como o caso dos balanceadores de carga e dos servidores de monitorização.

A aplicação deve estar ligada a uma infraestrutura de nuvem com o propósito de realizar tomadas de decisão autonomamente como o caso de criar e apagar máquinas virtuais e também alterar as suas capacidades.

É um pedido expresso para o desenvolvimento da aplicação, que esta esteja organizada em várias camadas e bibliotecas isoladas, com um nível baixo de acoplamento que permita escalar e integrar novas bibliotecas com facilidade.

1.2 ESTRUTURA DO DOCUMENTO

Este documento é constituído pelos seguintes capítulos:

- Capítulo 2 - **Estado da arte** - são abordados os assuntos relacionados com a dissertação a ser desenvolvida
- Capítulo 3 - **Sistema** - descreve o sistema e a infraestrutura da 360imprimir, tipos de decisão perante uma menor ou maior carga de pedidos sobre as máquinas virtuais
- Capítulo 4 - **Desenvolvimento** - apresenta a parte mais técnica da criação da aplicação *"IntelliScaling"* e serviços distribuídos
- Capítulo 5 - **Casos de estudo e resultados** - onde são feitas simulações para analisar o comportamento da aplicação
- Capítulo 6 - **Conclusão** - apresenta um resumo dos objetivos alcançados com o projeto

2

ESTADO DA ARTE

2.1 SISTEMA DE MONITORIZAÇÃO

Um sistema de monitorização recolhe e regista métricas de componentes de hardware e software de uma ou várias instâncias que integram uma infraestrutura de computadores. É responsável por analisar em tempo real o desempenho de cada uma das instâncias que compõe o sistema, permitindo também detetar falhas e alertar o administrador acerca de possíveis erros que surjam nos vários componentes desde hardware, software, dispositivos de rede ou sistema operativo [55, 51, 58].

Para realizar operações de dimensionamento é importante recolher as seguintes métricas das instâncias sobre análise: percentagem de CPU, percentagem de RAM, disco e tráfego de rede.

Um aspeto a ser explorado para este trabalho de dissertação, é a capacidade de utilizar as ferramentas de monitorização para definir *triggers* (acções) para valores de *threshold* (limite), *triggers* esses que despoletam alertas quando a taxa de utilização das métricas recolhidas incide sobre esses limites.

Posteriormente serão utilizados *triggers* para dar feedback ao controlador da aplicação de que será necessário fazer um *Scale Up* ou *Scale Down* de acordo com o valor medido.

A configuração das ferramentas de monitorização implica a instalação de um software que adicionará um *daemon* na instância que interessa monitorizar e indicar ao servidor que existe uma nova máquina a ser monitorizada. Essa aplicação recolhe as várias informações provenientes do agente e envia-as para o servidor que as armazena numa base de dados.

Nesta secção é feito um estudo comparativo de algumas ferramentas de monitorização existentes. Deste modo, a Tabela 1 apresenta uma comparação entre alguns serviços de monitorização a nível de preço, modo de configuração, tipos de alerta, interface gráfica e que protocolos suportados.

Dos serviços de monitorização analisados, todos eles são gratuitos mas diferem uns dos outros em alguns pormenores. O zabbix, além de ser utilizada pela equipa da *360imprimir* para monitorização da infraestrutura, é uma ferramenta que oferece funcionalidades esperadas do sistema de monitorização. Apesar da documentação estar um pouco incompleta,

	Zabbix	**Naggios**	**Prometheus**
Licença	open source	open source	open source
Nível de configuração	*Graphical User Interface (GUI)*	Ficheiros de configuração	*Command-Line Interface (CLI)*
Alertas	SMS, Email, Messenger, *Custom Scripts*	Email, SMS	Email, Slack
Interface gráfica	Completa	Só leitura	Fraca
Protocolos de Suporte	HTTP, FTP, SSH, POP_3, SMTP, SNMP, MYSQL	HTTP, FTP, SSH, POP_3, SMTP, SNMP, MYSQL	HTTP

Tabela 1: Tabela com a alguns serviços de monitorização e suas principais funcionalidades

permite notificar os utilizadores de várias formas, monitoriza muitos protocolos e possui uma interface gráfica com funcionalidades de leitura e escrita, o que difere da ferramenta Naggios e Prometheus [59].

2.2 SERVIÇOS DE ATUAÇÃO

Os sistemas computacionais devem possuir a capacidade de reagir dinamicamente face a alterações existentes no seu meio de forma a garantir um bom desempenho. Estes sistemas podem ser controlados através do uso de interfaces programáticas para aprovisionar e libertar recursos de qualquer tipo.

Os serviços de atuação correspondem então às interfaces que permitem a troca de instruções entre dois serviços [52, 57].

2.2.1 *Recursos da nuvem*

A nuvem é um serviço pronto para entrega, multicliente, que agrega um conjunto de recursos computacionais e possui um papel ativo como serviço numa infraestrutura.

Nesta secção são apresentados alguns recursos da Azure (CSP utilizada neste tema de dissertação).

Subscrição

Uma subscrição é um contentor lógico onde vários recursos (tais como máquinas virtuais, contas de armazenamento, servidores de bases de dados) podem ser implantados na nuvem [37].

Grupo de Recursos

Consiste num contentor que agrupa recursos relacionados de uma subscrição do Azure. Esta permite agrupar conjuntos de recursos em grupos lógicos que facilitam o controlo de acessos, o aprovisionamento a a monitorização desses recursos [32]. Além das máquinas virtuais, redes, sub-redes e outros recursos são agrupadas em grupos de recursos para uma boa organização de uma infraestrutura em nuvem.

Availability Set

Traduzido como um *conjunto de disponibilidade*, consiste num agrupamento lógico de máquinas virtuais. O Azure garante assim que as máquinas virtuais estão alocadas em vários servidores físicos, *Rack*'s de computação e repetidores de rede permitindo redundância e alta disponibilidade, reduzindo a existência de pontos únicos de falha [4, 21].

Rede virtual

A rede virtual é a representação da rede isolada numa subscrição da nuvem que permite que vários recursos (por exemplo máquinas virtuais) tenham a capacidade de comunicar com segurança entre si, com a internet e com redes locais. Esta rede tem como benefícios a capacidade de escalar e a elevada disponibilidade [42, 41]. Fornece também um isolamento lógico dedicado à subscrição e permite a divisão em diferentes sub-redes virtuais [40].

Sub-Rede

É uma subdivisão lógica de uma rede virtual que corresponde a um conjunto de IP da rede virtual [49].

Grupo de segurança da rede

Com a sigla *NSG*, o grupo de segurança da rede é um sistema designado para prevenir acessos não autorizados, utilizado para filtrar o tráfego numa rede virtual. Este possui regras de segurança que permitem e rejeitam tráfego de entrada e de saída do recursos do Azure [27].

Conta de armazenamento

É um serviço dedicado ao armazenamento de dados de forma redundante, escalável, configurável e seguro dentro de uma subscrição, acessível via *Hypertext Transfer Protocol (HTTP)* ou *Hypertext Transfer Protocol Secure (HTTPS)*.

Permite armazenar diferentes tipos de ficheiros, desde backups de discos, até ficheiros mais pequenos como o caso de imagens e texto em secções como o caso de contentores, pastas de partilha, tabelas ou filas [34, 35].

VirtualMachine Scale Set

É um recurso de computação do Azure utilizado para aprovisionar um conjunto de máquinas virtuais que estejam definidas no seu grupo. É utilizado pela aplicação *Software as a Service (SaaS)*, designado por *Azure Autoscale* que com a interação de um balanceador de carga permite aprovisionar máquinas virtuais de forma automática em função da carga [15].

Azure Monitor

O *Azure Monitor* é a solução nativa de monitorização do Azure. Esta recolhe dados de várias fontes, como por exemplo aplicações, sistemas operativos, recursos do *Azure* e mostra o comportamento dos recursos sob a forma de gráficos, *Logs* e métricas [7].

Máquina virtual

Também designada como instância, é o recurso da nuvem com mais destaque ao longo desta dissertação.

Podendo ser definida como uma imagem de computador associado a uma máquina física de um centro de dados, a máquina virtual oferece flexibilidade de virtualização sem o utilizador se preocupar com a gestão hardware físico. Uma caraterística de destaque da máquina virtual no desenvolvimento deste projeto de dissertação é o tamanho das máquinas virtuais dado que, o tamanho de cada máquina tem impacto sobre o poder de processamento mas também sobre custo por hora de utilização [39].

Como as nuvens trabalham num modelo *custo por utilização*, cada recurso é cobrado por um preço por hora pela sua utilização. Portanto, quanto mais potentes são as máquinas virtuais que constituem a infraestrutura, mais custos financeiros estas acarretam.

A Tabela 2 ilustra os preços praticados pelo *Azure* pelas categorias de máquinas virtuais utilizadas pela infraestrutura. Verifica-se que à medida que o tamanho das máquinas virtuais vai crescendo, o seu custo de utilização vai aumentando também e que os preços de família.

Cada máquina virtual pode se encontrar num dos seguintes quatro estados: **a iniciar** (a máquina está a ligar), **em execução** (a instância está ligada e a correr normalmente), **parada** (está parada mas a consumir recursos) e **desalocada** (está parada e não está a consumir recursos).

	Nº Núcleos (vCPUs)	**Memória (GB)**	**Valor monetário aproximado por mês (€)**
Família D			
Standard_DS1_v2	1	3.5	34,46
Standard_DS2_v2	2	7	68,62
Standard_DS4_v2	8	28	135,21
Família F			
Standard_F2s_v2	2	4	50,30
Standard_F4s_v2	4	8	130,62
Standard_F8s_v2	8	16	200,90
Família B			
Standard_B1ms	1	2	10,83
Standard_B2ms	2	8	45,60
Standard_B4ms	4	16	92,63

Tabela 2: Especificação das máquinas virtuais por família de tamanho, recursos físicos e o valor monetário em euros

2.2.2 *Serviços da nuvem*

Todos os serviços de nuvem oferecem recursos para auxiliar na gestão da nuvem. Como tal esta secção explora as várias interfaces e formas para controlar os recursos do Azure.

- **Azure Portal**: Esta abordagem permite ao utilizador gerir a subscrição através da interface gráfica (GUI). É um modo de acesso útil para explorar, consultar informações ou desempenhar interações simples e manuais na nuvem. Esta abordagem não é a ideal para instalação de aplicações ou para desempenhar tarefas que necessitam de ser desempenhadas repetidamente [9].

- **Azure *PowerShell* / Azure *CLI***: O Azure *PowerShell* é uma funcionalidade que apresenta um conjunto de *cmdlets* [1] para gerir os recursos da conta, via linha de comandos *PowerShell* [11]. Já o modo de acesso Azure CLI difere da anterior pela utilização da linha de comandos *bash shell* para desempenhar comandos do Azure, detendo a vantagem de ser multi-plataforma.

- **Azure *REST API***: Através deste meio, é possível aceder a toda a gama de recursos de subscrição, utilizando qualquer linguagem de programação que execute pedidos *HTTP*, através da invocação dos *endpoints* [2] disponibilizados no site *Microsoft Azure* através do endereço *https://docs.microsoft.com/en-us/rest/api/azure/*.

- **AZURE SDK**:

1 comandos usados pelo *PowerShell* que implementam funções específicas
2 endereços que podem ser acedidos por uma aplicação cliente

Os SDK's do Azure são conjuntos de bibliotecas que permitem aprovisionar certos recursos da nuvem com suporte para as seguintes linguagens de programação: .NET, Java, JavaScript e Python [12]. As bibliotecas foram criadas para facilitar o uso de serviços Azure, com o foco na consistência e familiaridade da linguagem [11].

2.2.3 *Ferramentas de configuração e orquestração*

Embora seja possível utilizar as interfaces para controlo e aprovisionamento direto de recursos da nuvem, para uma configuração do ambiente de forma mais estruturada devem ser utilizadas ferramentas de *Infraestructure as a Code (IaC)*.

Infraestrutura como código (IaC) é um processo de aprovisionar e gerir recursos de uma infraestrutura através de programas armazenados normalmente num sistema de controlo de versões como o caso do software *Git* [46].

O uso de tecnologias de IaC permite automatizar tarefas como criar ambientes completos com a configuração de infraestrutura, redes e balanceadores, instalar pacotes e configurar aplicações.

Os seguintes exemplos enumeram um conjunto de ferramentas de configuração, que são mais direcionadas para gestão e instalação de software em servidores já existentes [13]:

- Puppet, Ansible, Chef, SaltStack

Já a seguinte lista enumera algumas tecnologias de orquestração de infraestrutura, concebidas principalmente para o aprovisionamento de instâncias na nuvem:

- Terraform, Pulumi, Azure Resource Manager

A Tabela 3 apresenta um estudo comparativo de algumas tecnologias de orquestração, dada a necessidade de utilização deste tipo de uma ferramenta para o âmbito de aprovisionamento de infraestruturas. [3]

A seleção da tecnologia de orquestração é uma tarefa que está dependente do CSP destino, portanto de acordo com a Tabela 3, o *Azure Resource Manager* (também designado ARM) tem compatibilidade de aprovisionamento de recursos em nuvem Azure, ao contrário do Terraform e do Pulumi. Como todas as ferramentas assentam num paradigma declarativo, os ARM *templates* são escritos em *JSON*, o Terraform possui uma *Custom Domain-Specific-Language (DSL)* própria designada por *HashiCorp Configuration Language*, mas por sua vez os templates Pulumi podem ser criados em Javascript, TypeScript, .NET, Python ou Go.

Todas as ferramentas presentes na tabela são *open-source*, mas a nível de comunidade e documentação, o Terraform destaca-se das outras 2.

3 Estudo realizado na seguinte bibliografia: https://stackshare.io/stackups/pulumi-vs-Terraform-vs-go-packages-cloudformation , https://blog.gruntwork.io/why-we-use-Terraform-and-not-chef-puppet-ansible-saltstack-or-cloudformation-7989dad2865c , https://dinarys.com/blog/azure-resource-manager-arm-shablony-vs-Terraform

	Terraform	**Pulumi**	**Azure Resource Manager**
Licença	open source	open source	open source
Linguagem	*Hashicorp Configuration Language* (HCL)	JavaScript, TypeScript, .Net, Python, Go.	JSON
Paradigma	Declarativo	Declarativo	Declarativo
Compatibilidade de nuvem	Todas	Todas	Azure
Documentação e Comunidade de suporte	Extensa	Pequena	Pequena

Tabela 3: Estudo comparativo de algumas ferramentas de orquestração

2.2.4 *Servidores de automação*

Uma ferramenta de aprovisionamento é necessária para criar, apagar e configurar instâncias na infraestrutura, no entanto, é necessário um servidor de automação para invocar os comandos da ferramenta. É neste contexto que é necessário aplicar ao projeto uma metodologia de integração e entrega contínua (*Continuous integration and delivery (CICD)*) com o objetivo de automatizar o processo de lançar a infraestrutura para a nuvem.

O uso de pipelines permite automatizar o processo de CICD pois invalida a necessidade de executar os comandos de Terraform de forma manual, podendo ser executados de forma automática e contínua [22].

Existem algumas soluções que permitem a criação de pipelines CICD, como é o caso do Jenkins, *Gitlab CI/CD* ou *AzureDevops*.

2.2.5 *Balanceadores*

Os balanceadores de carga são um serviço tecnológico utilizado para distribuir de forma eficiente o tráfego de rede em vários servidores. Atuam como um ponto único de acesso aos clientes, distribuindo depois o tráfego de entrada por um grupo de servidores [47]. Possuem também um papel importante num sistema tolerante a faltas, isto é, no caso de um servidor ficar inativo, o balanceador de carga redirecionará o tráfego para os restantes servidores que se encontram online. Num caso de um servidor ser adicionado ao grupo, o balanceador automaticamente passará a enviar-lhe solicitações. Os balanceadores de carga possuem como principais funções:

- Distribuir os pedidos dos clientes de maneira eficiente nos vários servidores;

	HAProxy	**Nginx**	**Apache mod_proxy**	**Azure Loadbalancer**
Nº métricas de configuração	Muitas	Poucas	Poucas	Sem suporte
Multi-tarefa	Não	Sim	Sim	Não
Página de estatísticas	Leitura e escrita	Só leitura	Leitura e escrita	Sem suporte
Complexidade de configuração	Médio	Fácil	Fácil	Muito fácil

Tabela 4: Estudo comparativo entre os algumas características dos principais serviços de balanceamento de carga

- Garantir elevada disponibilidade e confiabilidade, porque apenas envia pedidos para servidores ativos;
- Oferecer flexibilidade em adicionar ou remover servidores do grupo conforme carga em demanda;

Ferramentas de balanceamento de carga

Nesta secção é feito um estudo comparativo de algumas tecnologias de balanceamento de carga. A Tabela 4 expõe uma comparação entre os vários serviços de balanceamento de carga a nível de número de métricas suportadas na configuração, se funciona como um serviço multi-tarefa, tipo de página de estatísticas e complexidade de configuração do serviço.

Todas as tecnologias presentes na tabela desempenham função de balanceamento de carga mas destacam-se o HAProxy e o Azure Loadbalancer por serem utilizadas nas infraestruturas da *360imprimir* para distribuir os pedidos pelos vários servidores web.

O *Azure Loadbalancer* é um serviço de balanceamento de carga disponibilizado pelo Azure mas não permite implementar regras de controlo de tráfego. Por sua vez, o *HAProxy* é uma aplicação mais robusta que disponibiliza uma interface gráfica que permite gerir os vários nodos no modo leitura e escrita.

São apresentadas de seguida diferentes formas de interação sobre os balaceadores de carga:

- **Interface Gráfica**: Através da interface gráfica disponibilizada pelo serviço de balanceamento de carga, é possível configurar as máquinas que estão dentro do grupo de servidores. Essa interface permite manualmente colocar e retirar os servidores do grupo. Esta abordagem é interessante mas não é muito útil no caso de se pretender ter uma aplicação externa a controlar o balanceador de carga.
- **Linha de comandos**: É possível executar instruções na linha de comandos que despoletam acções diretamente sobre o ficheiro de configuração do balanceador, como

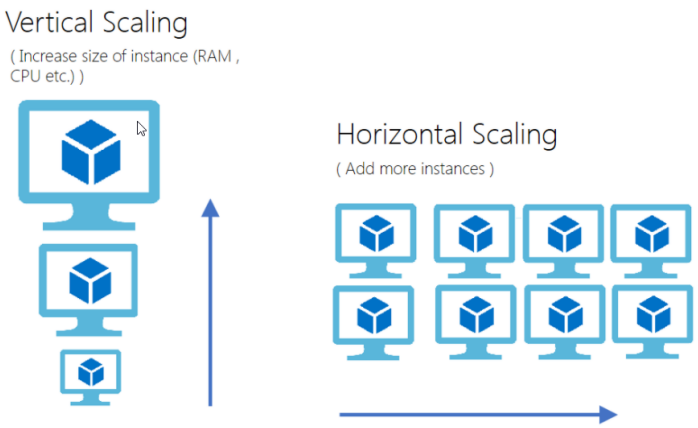


Figura 1: Diferenças dimensionamento vertical versus horizontal

por exemplo através de *Unix Socket commands* para interagir com o ficheiro socket do balanceador.

- **API Client**: Existem também bibliotecas ou *API*'s que controlam ficheiros de configuração do balanceador, como é o exemplo do módulo *http_api_module* que fornece uma REST API para aceder e alterar configurações do servidor em execução.

2.3 SERVIÇOS DE AUTOSCALING

Um recurso elástico tem a capacidade de ser dimensionado (horizontalmente ou verticalmente) para fazer o ajuste de acordo com a carga de trabalho de entrada. A Figura 1 evidencia as diferenças entre o dimensionamento vertical e o horizontal num contexto de máquinas virtuais.

O dimensionamento automático faz uma gestão de sobrecarga dos recursos tirando proveito da elasticidade dos ambientes hospedados na nuvem. Este reduz a necessidade de uma monitorização contínua por parte de um operador para tomar decisões de adicionar ou remover recursos no sistema. Os recursos podem escalar de duas formas [3]:

- *Dimensionamento vertical*, significa alterar a capacidade de um recurso em função da carga em demanda e faz-se sentir através de uma acção de ***Scale Up*** (aumentar tamanho) ou ***Scale Down*** (diminuir tamanho). Esta abordagem implica indisponibilidade do recurso quando se encontra em processo de atualização;

- *Dimensionamento horizontal*, também conhecido como ***Scale Out*** (adicionar recursos) ou ***Scale In*** (remover recursos), implica a adição ou remoção de instâncias de um grupo em função da carga em demanda. Esta abordagem não gera interrupção à medida que novos recursos são aprovisionados.

As técnicas de dimensionamento são divididas em dois tipos de abordagem: **reativa** ou **proativa** como será descrito posteriormente.

2.3.1 *Abordagem reativa*

A abordagem reativa é a mais comum, tendo como características o facto de impor que o sistema reaja a alterações, tomando uma acção de dimensionamento em função da carga em demanda [56]. O sistema reativo utiliza uma estratégia baseada em *threshold* (limiar) em que as máquinas virtuais são controladas com pouca intervenção manual. Esta abordagem implica que o sistema reaja com base em alterações detetadas na carga de trabalho recebidas, utilizando os valores obtidos através de um conjunto de variáveis de monitorização.

As aplicações SaaS de nuvem como o caso do *Azure AutoScale* estão desenvolvidas com esta abordagem para realização de dimensionamento automático. Nesta aplicação os utilizadores podem criar grupos de políticas de dimensionamento para definir acções de acordo com a carga recebida [60].

A abordagem *reativa* assenta num mecanismo de feedback que está constantemente a responder face a alterações recolhidas, durante períodos de tempo de leitura (como por exemplo a cada 5 minutos) [54]. Quando uma condição de dimensionamento é conhecida, é despoletado um alerta para dar origem a uma acção que como resultado dará origem a uma tomada de decisão de vertical (*Scale Up*, *Scale Down*) ou horizontal (*Scale In* ou *Scale Out*) sobre os recursos da nuvem.

2.3.2 *Abordagem proativa*

A abordagem *proativa* assenta na tomada de decisão baseada em análises preditivas, relativas a dados históricos através da integração de um conjunto de modelos de algoritmos que analisam os dados obtidos a fim de prever qual será o valor de determinadas métricas no futuro [20].

Análises preditivas usam dados, algoritmos estatísticos e técnicas de *Machine Learning* para identificar a probabilidade de resultados futuros, a partir de dados históricos.

O primeiro passo para adopção de uma estratégia preditiva consiste em armazenar e estruturar grandes quantidades de dados, a fim de facilitar a análise e o processamento dos mesmos. Após a recolha e organização dos dados, o desafio assenta na criação de modelos preditivos, ou seja, modelos matemáticos que quando aplicados sobre os dados consigam prever resultados futuros.

Machine learning é uma aplicação da inteligência artificial que fornece aos sistemas a capacidade de aprenderem automaticamente e adquirir experiência sem serem explicitamente

programados [48]. Serviços de dimensionamento automático utilizam *Reinforcing Learning* [4] de forma a realizarem a melhor tomada de decisão para um determinado estado do sistema através de dados históricos e tentativa erro [53].

2.3.3 *Exemplos*

Nesta secção são abordados alguns exemplos de serviços com a capacidade de controlar recursos de IaaS da nuvem.

Azure Autoscale

O *Azure Autoscale* é uma aplicação nativa (*SaaS*) do Azure que funciona como um serviço de alocação de recursos de forma dinâmica a fim de garantir os devidos requisitos de desempenho. Este tira partido da elasticidade dos serviços existentes num sistema de nuvem, facilitando a gestão de ocorrências de sobrecarga, deste modo reduz a necessidade de existência de um operador para monitorizar constantemente o desempenho de um sistema e tomar decisões sobre a adição ou remoção de recursos [3].

Segundo os princípios da elasticidade, existem 2 possíveis formas de dimensionamento:

- Vertical, que implica a alteração da capacidade dos recursos, inclui as acções de *Scale Up* e *Scale Down*.
- Horizontal, que implica adicionar ou remover instâncias de um grupo de recursos e inclui acções de *Scale Out* e *Scale In*;

A aplicação *Azure Autoscale* é constituído pelos seguintes componentes [3]:

- Serviço de instrumentação e monitorização de forma a capturar métricas tais como tempo de resposta, utilização de CPU e utilização de memória;
- Sistema de controlo de lógica de tomada de decisão que avalia as métricas em função de determinados *thresholds* e temporizadores de maneira a decidir se o sistema deve reagir;
- Sistema de verificação que garante que a acção de dimensionamento foi bem executada;

Esta abordagem possui todos os requisitos anteriormente enumerados que são necessários para escalar aplicações de uma infraestrutura, mas se houver necessidade de implementar regras específicas para fazer autoscaling, deve ser ponderada a aplicação de uma implementação personalizada [3].

4 Segundo [29], é um ramo da *Machine Learning* cuja forma de aprendizagem é baseada em reforço, onde o agente é recompensado por comportamentos corretos e punido por comportamentos incorretos

Kubernetes

Kubernetes é uma ferramenta de orquestração de contentores que elimina muito do processo manual de instalação e de dimensionamento das aplicações em contentores [18]. O serviço Kubernetes possui um mecanismo que se designa por *Horizontal Pod Autoscaler*, que automaticamente escala o número de *pods* presentes num *controller* de replicação, *deployment* ou num conjunto de réplicas baseado na percentagem de utilização de CPU, isto é, o autoscale é feito de forma automática se a taxa de CPU ultrapassar um determinado valor de *threshold*.

De acordo com a investigação, kubernetes gere um sistema de micro-serviços, portanto a sua implementação no sistema da empresa foge ao contexto que é pedido, porque pretende-se implementar uma solução que atue sobre máquinas virtuais e não sobre contentores.

3

SISTEMA

3.1 INFRAESTRUTURA

Este capítulo aborda como funciona o sistema existente nas infraestruturas tecnológicas da 360imprimir, contextualiza qual o problema existente e explica quais as intervenções realizadas no processo de dimensionamento.

3.1.1 *Funcionamento do sistema*

Na nuvem, a infraestrutura da *360imprimir* está divida em 3 grupos de servidores para estrategicamente disponibilizar o website para *vinte e um* mercados dos continentes onde atuam, que necessitam de estar com o menor tempo de indisponibilidade possível.

As máquinas virtuais presentes nesses mercados estão organizados em grupos e agem como servidores de *FrontEnd* de aplicações, que através do servidor aplicacional *Internet Information Services (IIS)* correm os *Websites* dos vários módulos do website da 360imprimir. Estes servidores de *FrontEnd* estão ligados a outros grupos de recursos como por exemplo, bases de dados relacionais e não relacionais garantem o funcionamento do *website*.

As máquinas virtuais estão de forma propositada organizadas em grupos com o objetivo de criar redundância e criar arquiteturas tolerantes a faltas. Para auxiliar esta prática, são introduzidos balanceadores de carga à cabeça dos grupos de máquinas virtuais, responsáveis por distribuir por elas os vários pedidos de acesso, dividindo a taxa de pedidos pelos vários servidores do grupo.

A Figura 2 mostra a arquitetura genérica que garante funcionamento dos vários *websites* que compõe a infraestrutura. Os utilizadores acedem ao site através do *Domain Name System (DNS)* do balanceador de carga que distribui os pedidos pelas máquinas de *FrontEnd* do grupo, pedidos esses que são reencaminhados para as máquinas de forma aleatória, graças ao algoritmo *round robin.* [1]

1 Algoritmo utilizado no balanceamento de carga para encaminhar os pedidos de forma cíclica pelos clientes. [1]

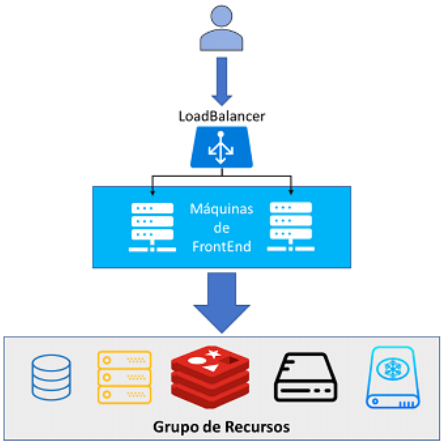


Figura 2: Esquema de funcionamento de um *website* alojado num servidor de *FrontEnd*

Os recursos como por exemplo *SQL Server*, *redis*, disco, etc. identificados no grupo de recursos presente na Figura 2 permitem o armazenamento e gestão dos dados existentes no site.

Principalmente nas máquinas virtuais do ambiente de *produção* é de extrema importância saber qual estado e o desempenho destas em execução. Como sistema de monitorização, a *360imprimir* utiliza o *zabbix* que permite a monitorização contínua das máquinas virtuais.

Em cada máquina virtual está instalado um agente do *zabbix* que envia informação dos vários componentes da máquina virtual para um servidor do *zabbix* que centraliza, armazena e disponibiliza de forma amigável ao utilizador.

O servidor de Zabbix recolhe os dados analíticos provenientes dos agentes configurados nas máquinas virtuais que compõe a infraestrutura e permite monitorizar em tempo real métricas relativas às máquinas virtuais. Assim, através dos diversos *Dashboards* e sistemas de alertas que o servidor disponibiliza, a equipa de gestão de infraestrutura consegue analisar e atuar nos diferentes recursos em caso necessidade.

3.1.2 *Contextualização do problema*

Os websites registam usualmente uma taxa de acesso constante durante a semana por parte dos utilizadores, no entanto nos fins de semana e em certas alturas do mês regista-se uma diminuição do número de acessos ao website, o que implica que não seja necessário ter máquinas virtuais tão potentes para aguentar a carga em demanda. Nestes casos, a equipa de gestão de infraestrutura analisa e age sobre o grupo de máquinas virtuais, diminuindo o tamanho ou reduz o número de máquinas virtuais que se encontram a servir os websites.

Por outro lado, quando ocorrem campanhas de lançamento de produtos ou dias de destaque como o caso da *Black Friday*, o número de acessos aos websites por parte dos

utilizadores dispara e são necessários mais recursos computacionais para dar resposta à afluência dos utilizadores.

Para analisar o comportamento das máquinas e os níveis de afluência aos websites, a equipa de manutenção da infraestrutura está atenta principalmente aos seguintes tipos de métricas: percentagem de utilização de CPU e de memória RAM, número de pedidos no IIS.

Nestas ocasiões quando se verificam comportamentos de stress, a equipa de gestão da Infraestrutura age sobre a infraestrutura, aumentando o número de máquinas virtuais ou aumentando o tamanho das maquinas virtuais para aguentar a carga em demanda.

Todavia alterar o tamanho das máquinas pode causar alguma indisponibilidade no site se o processo não for desempenhado com a devida atenção.

Quando é necessário mexer no tamanho da máquina virtual, esta é reiniciada e esse comportamento deve ser tido em atenção para não gerar indiponibilidade no *website* que se encontra dentro dela.

Os balanceadores de carga neste sentido também desempenham um papel importante porque no acto da alteração do tamanho da máquina virtual, é possível dar a informação ao balanceador de carga que a máquina ficará inativa por momentos e desativando-a do grupo do balanceador de carga que desta forma, o website nesta máquina não fica indisponível.

3.1.3 *Análise e manutenção das máquinas virtuais*

Para os servidores de *FrontEnd* de produção da infraestrutura da 360imprimir as famílias e tamanhos utilizadas nas máquinas virtuais são as do tipo D e F, que podem ser consultadas na Tabela 2 no Capítulo 2.

As famílias de máquinas virtuais do tipo *"F"* possuem uma alta relação memória em função de CPU, dando boa resposta para servidores web de tráfico intermédio, como por exemplo nos servidores de ambientes de equipa e de testes da *360imprimir*.

Por outro a famílias do tipo *"D"* possuem uma relação equilibrada de CPU e memória e são mais utilizadas para um contexto mais responsável e exigente, como por exemplo servidores produção.

A Tabela 2 na Secção 2.2.1 mostra os preços praticados pelo *Azure* pelas categorias de máquinas virtuais utilizadas pela infraestrutura. Verifica-se que à medida que o tamanho das máquinas virtuais vai crescendo, o seu custo por utilização também vai aumentando.

Em função do custo total e da capacidade de processamento necessária, a equipa de gestão de infraestrutura faz uma análise estratégica sobre se compensa adicionar/ remover máquinas virtuais ou alterar o tamanho das existentes no grupo de recursos.

Isto significa que mesmo automatizando o processo de gestão da infraestrutura com a aplicação *IntelliScaling*, a equipa de gestão de infraestrutura deve ter a liberdade de decidir qual a decisão a tomar.

3.1.4 *Procedimento em operações de dimensionamento*

Alterar recursos das máquinas em execução é um processo que requer algum cuidado pois há o risco de causar indisponibilidade nos websites se o processo não for desempenhado com a devida atenção. Por exemplo a alteração do tamanho de uma máquina virtual faz com que seja reiniciada para aplicar a nova alteração, ficando indisponível por breves momentos.

Nestas tarefas os balanceadores de carga desempenham um papel ativo porque permitem desviar o tráfego de pedidos para outra(s) máquina(s) ativa(s) segundo uma configuração, evitando indiponibilidade se aplicações que se encontrem a correr nelas.

Em ambiente de produção, este processo de dimensionamento de máquinas virtuais ainda requer mais cuidado. Os seguintes pontos, enumeram a ordem de tarefas desempenhada sempre que é necessário realizar um procedimento de **alteração do tamanho** das máquinas de forma manual:

1. Desativação do servidor web do grupo do balanceador de carga, através da interface de gestão do HAProxy;
2. Escolha do tamanho pretendido para a máquina virtual através do portal do Azure;
3. Cálculo e alteração dos pesos dos servidores web dentro do ficheiro de configuração do haproxy que corresponde a cada máquina virtual;
4. Ativação do servidor web no grupo do balanceador de carga através da interface do balanceador de carga;
5. Reinício do serviço de balanceador de carga.

A Figura 3 exemplifica o cenário esperado de uma operação de dimensionamento vertical (*Scale Up*) após aplicado o procedimento acima enumerado. A parte esquerda da Figura 3 mostra o ponto de partida onde existem duas máquinas virtuais com o tamanho *Standard_B2ms* que significa que ambas possuem dois núcleos, onde está indicado que ambas recebem 50% dos pedidos dos clientes que chegam até ao balanceador de carga.

A parte direita da Figura 3 mostra o resultado final após transformação de *Scale Up*, onde é visível que na máquina *PoolServer1* houve alteração do tamanho para *Standard_B4ms* e para acompanhar esta alteração, no balanceador de carga houve uma reconfiguração dos pesos para que a máquina mais forte receba 67% dos pedidos e a outra receba 33% dos pedidos por ter um tamanho inferior.

No caso de intervenção para realizar uma **alteração da quantidade de recursos** num grupo de máquinas virtuais, as equipas de gestão da infraestrutura realizam o seguinte procedimento:

1. Execução de um *script* Terraform utilizado para aprovisionar as máquinas virtuais num dado ambiente da infraestrutura;

Figura 3: Exemplo de uma operação de dimensionamento vertical (*Scale Up*)

2. Inclusão de um servidor web no grupo do ficheiro de configuração do balanceador de carga (apontando para a nova máquina virtual) no caso de acção de *scale out*, ou remoção do servidor web relativo à máquina virtual que será apagada, no caso de acção de *scale in*;

3. Cálculo e alteração dos pesos de todos os servidores web no ficheiro de configuração do balanceador;

4. Reinício do serviço de balanceador de carga.

5. Registo da máquina virtual no servidor do Zabbix e configuração do agente do Zabbix na máquina virtual para ter a possibilidade de ser monitorizada (em acção de *scale out*)

6. Adição da máquina virtual no servidor de configuração do *Octupus Deploy* [2], responsável por instalar e configurar o site na nova máquina virtual (se acção de *Scale Out*)

A Figura 4 exemplifica o cenário esperado após uma operação de dimensionamento horizontal, que mostra a finalidade de uma acção de *Scale In*, ocorrendo uma redução de três máquinas virtuais que se encontravam a servir para apenas duas. Numa operação de dimensionamento horizontal também existe o recálculo do peso do balancedor de carga para para existir uma distribuição correta do pedidos sobre as máquinas virtuais, onde é possível verificar que inicialmente todas as máquinas virtuais recebiam 33% dos pedidos e após a operação de dimensionamento, as duas máquinas passaram a receber 50% dos pedidos.

3.2 Configuração de serviços

Antes do desenvolvimento da aplicação *IntelliScaling* com a capacidade de realizar ajustes na infraestrutura e que consiga aplicar alterações com a lógica presente na Secção 3.1.4 de

2 De acordo com [2], *Octupus Deploy* é uma ferramenta de configuração que facilita a instalação automática de aplicações desenvolvidas em *ASP.NET*, *Java*, *NodeJS* em vários ambientes.

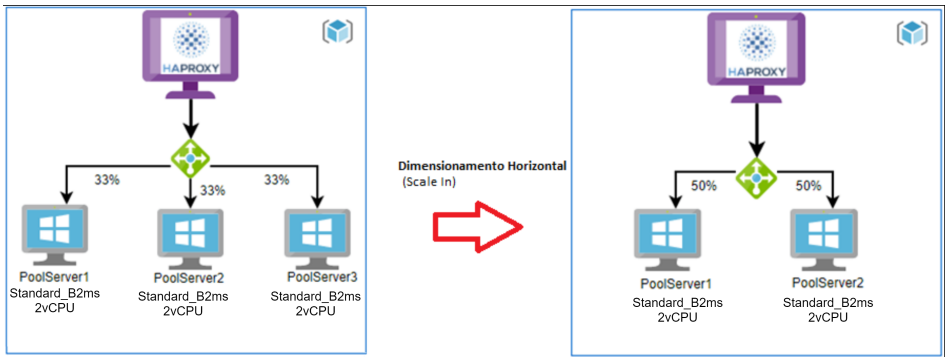


Figura 4: Exemplo de uma operação de dimensionamento horizontal (*Scale In*)

forma automática, inicialmente é necessário preparar uma infraestrutura para a aplicação ter o controlo sobre os vários componentes do sistema.

Portanto, esta secção aborda os componentes e serviços instalados e configurados para tornar possível o funcionamento *IntelliScaling* como um sistema de gestão.

Os requisitos essenciais para adaptar a *IntelliScaling* a uma infraestrutura são:

- Uma infraestrutura na nuvem constituída por máquinas virtuais organizadas em grupos de recursos com o papel de servidores de *FrontEnd*. As máquinas virtuais devem possuir o agente do Zabbix em execução para enviar métricas para o servidor de monitorização;
- Uma máquina virtual com o sistema operativo Linux com a aplicação Zabbix configurada na versão 4.4 a servir como servidor de monitorização.
- Uma ou mais máquinas virtuais com sistema operativo Linux com a aplicação HAProxy configurada para desempenhar função de balanceador de carga e reencaminhar a carga de trabalho pelos servidores de *FrontEnd*.
- Um script Terraform que contém as instruções para realizar aprovisionamento de máquinas virtuais e recursos da nuvem
- Um orquestrador de pipelines do *Azure DevOps* que permita executar tarefas de automação acerca do aprovisionamento e outras tarefas de aprovisionamento da infraestrutura.

3.2.1 *Componentes de monitorização*

Esta secção aborda a configuração de um um sistema de monitorização constituído por um servidor de Zabbix e vários agentes configurados de forma distribuída.

Cliente de Zabbix

O agente do Zabbix é um serviço necessário para ser instalado em todos as instâncias que necessitam de ser monitorizadas. O agente tem como função recolher taxas de utilização de componentes presentes nas máquinas virtuais pelo cliente e fornece as informações para o servidor por pedidos.

Para ser possível a monitorização de cada instância da infraestrutura, é necessário instalar e configurar o agente do Zabbix na máquinas virtual com o objetivo deste enviar informações para o servidor.

Nas máquinas virtuais a monitorizar foi realizado o seguinte procedimento para configuração do agente de modo ativo:

- Transferência do software agente do Zabbix versão igual ou superior a 4.4.
- Instalação e configuração do agente do Zabbix na máquina virtual. Após a instalação do software, é necessário aceder ao ficheiro de configuração "zabbix_agentd_win.conf"que se encontra numa pasta do disco criada após a instalação do software do agente, e alterar a configuração presente na Listagem 3.1 para apontar para o servidor do zabbix;

```
1 Server = ip_publico_zabbix_server
2 ServerActive = ip_publico_zabbix_server
3 Hostname = nome_maquinavirtual
```

Listagem 3.1: Configuração necessária no ficheiro zabbix_agentd_win.conf no agente do zabbix

- As portas 10050 e 10051 da máquina virtual devem estar abertas para ser possível estabelecer comunicação com a máquina servidor do zabbix.

É possível verificar que o agente do Zabbix está em execução, consultando a secção serviços do *gestor de tarefas* do windows, que como vísivel na Figura 5. Se o serviço "Zabbix Agent"estiver em execução significa que a configuração do lado da máquina virtual que interessa ser monitorizada está bem realizada e irá conseguir comunicar com o servidor do zabbix.

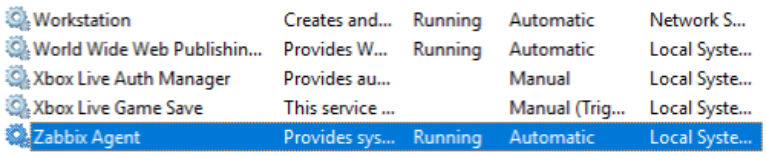


Figura 5: Exemplo do serviço do agente do Zabbix em execução numa máquina virtual

Servidor Zabbix

O servidor de monitorização utiliza a tecnologia Zabbix para conseguir recolher as informações enviadas pelo agente. Para recolher métricas de interesse da *IntelliScaling*, como por exemplo percentagem de utilização de CPU, percentagem de utilização de memória de uma máquina virtual, o servidor de monitorização deve possuir uma versão do Zabbix igual ou superior 4.4, já versões anteriores da não disponibilizam certas métricas de interesse.

Este componente existe na infraestutura da 360imprimir mas serve como uma ferramenta de análise do estado de cada máquina virtual, mas para a aplicação *IntelliScaling* conseguir cooperar com esta ferramenta, existem alguns configurações que devem ser realizadas:

- o servidor de Zabbix deve estar configurado com a mesma versão (>= 4.4) que os agentes do zabbix, para garantir compatibilidade.
- cada *host* [3] deve ser instalados alguns *templates* personalizados para devolver certas métricas de interesse como por exemplo, número de pedidos do IIS.

A parametrização e configuração dos recursos pode ser realizada via interface gráfica através do IP público da máquina virtual "`http://ip_publico_zabbix_server/zabbix`"[4] ou por intermédio da *API* que o Zabbix disponibiliza via *REST* para tarefas de automação como por exemplo em operações de dimensionamento horizontal, nas quais é necessário configurar o *host* no servidor de monitorização. A Figura 6 a interface gráfica do servidor de monitorização acedido através do DNS da máquina virtual.

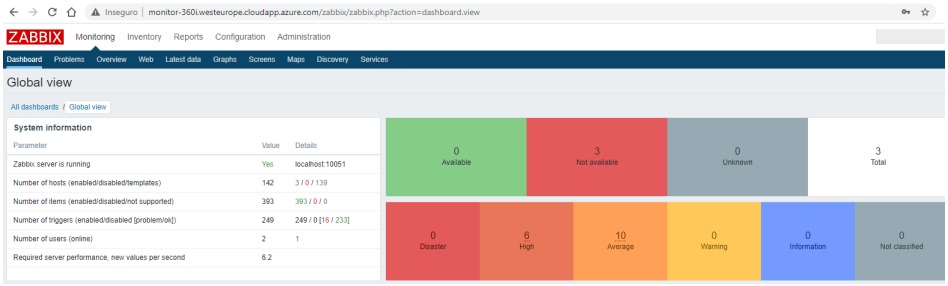


Figura 6: Exemplo interface gráfica do servidor de monitorização

Sobre interface gráfica ou via *REST API* é possível possível registar máquinas virtuais (*hosts*), templates, associar grupos de máquinas virtuais e consultar dados em tempo real ou em intervalos de tempo.

O servidor de monitorização constitui-se assim numa peça fundamental no propósito desta dissertação pois permite armazenar e transmitir as informações das máquinas virtuais

3 Um host corresponde à entidade que se pretende que seja monitorizada. Para o contexto de dissertação, o host corresponde às máquinas virtuais no servidor de zabbix
4 O *ip_publico_zabbix_server* corresponde ao IP público ou valor do DNS da máquina virtual

à aplicação *IntelliScaling*. Posteriormente na Secção A.1.8 do capítulo do desenvolvimento, é explicado como é feita a ligação entre o servidor de monitorização e como é possível recolher os valores de interesse de cada máquina virtual.

Templates de configuração

A escolha de templates no servidor do Zabbix permite definir o tipo de métricas a serem recolhidas das máquinas virtuais que passam a estar disponíveis no servidor de monitorização. Como tal, na criação de um *host* no servidor devem ser associados alguns *templates* para extrair métricas das máquinas virtuais.

Como esta dissertação o objetivo está direcionada em controlar grupos de máquinas virtuais com o sistema operativo Windows, os templates que se seguem permitem recolher métricas identificadas na Secção 4.1.1 relativas a memória RAM, CPU e pedidos do IIS de máquinas virtuais para serem utilizadas na aplicação *IntelliScaling*:

- Template OS Windows by Zabbix agent
- Template App IIS Service

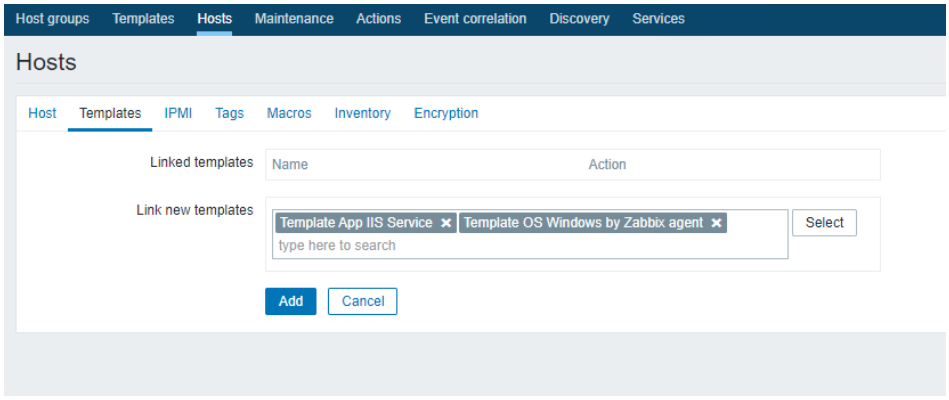


Figura 7: Exemplo interface gráfica do servidor de monitorização

Contudo, o template "Template App IIS Service"não vem instalado por defeito com o servidor do zabbix, necessita de ser importado externamente para servidor, para ser possível recolher métricas do IIS de cada máquina virtual.

3.2.2 *Balanceador de carga*

Acima dos grupos de máquinas virtuais, existem os balanceadores de carga que são máquinas com sistema operativo Linux e que possuem a aplicação HAProxy instalada para distribuirem os pedidos para os vários servidores web e assim oferecerem alta disponibilidade.

Esta secção aborda a configuração necessária realizada no HAProxy para a aplicação *IntelliScaling* gerir acções sobre o balanceador de carga e controlar os vários servidores web que estão organizados em grupos de backend [5].

O ficheiro de configuração da aplicação está localizado na diretoria de sistemas de ficheiros Unix em */etc/haproxy/haproxy.cfg* e neste ficheiro que são efetuadas todas as alterações a nível de grupos de backend, servidores web e respetivos pesos.

As seguintes instruções adicionadas na secção global do ficheiro *haproxy.cfg* atribuem à aplicação *IntelliScaling* a capacidade de comunicar com o balanceador de carga via *REST API*:

- stats socket /etc/haproxy/haproxysock level admin
- stats socket ipv4@haproxy_privateip_vm:8080 [6] level admin

```
global
        log /dev/log    local0
        log /dev/log    local1 notice
        chroot /var/lib/haproxy
        stats socket /run/haproxy/admin.sock mode 660 level
        stats timeout 30s
        user haproxy
        group haproxy
        daemon
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/private
        ssl-default-bind-ciphers ECDH+AESGCM:DH+AESGCM:ECDH
        ssl-default-bind-options no-sslv3
        stats socket /run/haproxy/admin.sock mode 660 level
        stats timeout 30s

        stats socket /etc/haproxy/haproxysock level admin
        stats socket ipv4@10.1.0.6:8080 level admin
defaults
```

Figura 8: Imagem do HAProxy com as configurações necessárias para permitir o controlo do IntelliScaling sobre o balanceador

A porta referida na instrução "*ipv4@haproxy_privateip_vm:8080*"deve estar aberta para existir interoperabilidade entre a máquina virtual e a aplicação de *IntelliScalling* para estabelecer comunicação entre os dois sistemas.

Com estas configurações a aplicação *IntelliScaling* está apta para comunicar com o servidor de balanceador via API.

A Figura 9 ilustra a existência de dois grupos de backend no ficheiro do haproxy com os nomes *backend.360imprimir.pt* e *backend.api-360imprimir.pt*, constituídos pelos seus servidores web que são inicializados com a palavra *"server"*. Por exemplo, o backend com o nome *backend.360imprimir.pt* possuir três servidores web que correspondem às instâncias de máquinas virtuais designadas por *FE-PoolServer1*,*FE-PoolServer2*, *FE-PoolServer3*. Destaca-se que em cada instância está presente o nome do servidor web, o seu ip privado e o seu peso que corresponde ao valor da percentagem de pedidos que cada servidor web está disposto a receber.

5 *backend* corresponde a um grupo de servidores que agrupam servidores web
6 *haproxy_privateip_vm* corresponde ao IP privado ou público da máquina virtual do balanceador de carga

```
frontend loadbalancer
        mode http
        stats enable
        stats uri /haproxy/stats
        default_backend backend.360imprimir.pt
        stats refresh 30s
        stats show-node
        bind *:80
        stats admin if TRUE   #responsible for attr checkboxes e set
backend backend.360imprimir.pt
        server FE-PoolServer3 10.1.0.8:80 weight 33 check
        server FE-PoolServer2 10.1.0.5:80 weight 33 check
        server F3-PoolServer1 10.1.0.4:80 weight 33 check
        balance roundrobin
        option forwardfor

backend backend.api-360imprimir.pt
        server FE-PoolServer2 10.1.0.5:80 weight 50 check
        server FE-PoolServer3 10.1.0.8:80 weight 50 check
        balance roundrobin
        option forwardfor
```

Figura 9: Exemplo de haproxy.cfg com 2 backends "backend.360imprimir.pt"e "backend.api-360imprimir.pt"e os respectivos servidores web

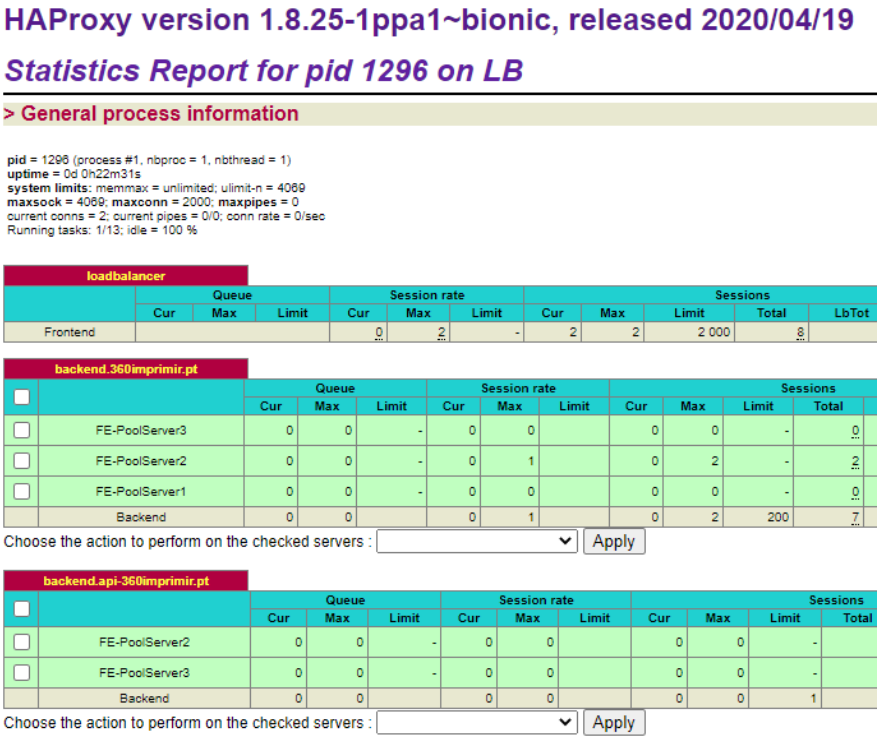


Figura 10: Interface gráfica de controlo do HAProxy

A Figura 10 mostra de forma gráfica as informações definidas no ficheiro do HAProxy onde é possível desempenhar algumas operações como por exemplo, ativar e desativar um servidor web. Esta figura disponibiliza graficamente as configurações presentes na Figura 9, onde é possível ver informação do grupo *backend* e os seus servidores web.

3.2.3 *Terraform*

De acordo com a Secção 2.2.3 do estado da arte, o aprovisionamento de máquinas virtuais e de componentes da nuvem pode ser conseguido através do Terraform que permite gerir recursos de um dado CSP [7] através de código.

O Terraform é uma ferramenta open source que permite criar, alterar e criar versões de uma infraestrutura de forma segura e eficiente através da invocação de *scripts* desenvolvidos numa linguagem declarativa designada por *HashiCorp Configuration Language* (HCL) [36].

O Terraform neste projeto de dissertação é invocado a partir de um servidor de automação para realização de operações de dimensionamento horizontal.

Foi então desenvolvido um script Terraform à medida deste projeto, pensado na infraestrutura de destino, onde estão declaradas grupos de variáveis que são fornecidas pela aplicação *IntelliScaling* para ser processadas pelo Terraform.

A Listagem 3.2 ilustra algumas partes do script desenvolvido.

```
1  [...]
2  ########### Resoure Group ##########################
3  data "azurerm_resource_group" "resource_gp" {
4    name = "${var.rgoup}"
5  }
6  ################ Public IP ##################
7  resource "azurerm_public_ip" "example" {
8    count                   = length(local.vmsName)
9    name                    = "${local.vmsName[count.index]}-pip"
10   location                = "${var.region}"
11   resource_group_name     =  data.azurerm_resource_group.resource_gp.name
12   allocation_method       = "Dynamic"
13   domain_name_label       = lower(local.vmsName[count.index])
14   }
15 [...]
16 ################ Virtual Machines ##################
17 resource "azurerm_virtual_machine" "main" {
18   count                 = length(local.vmsName)
19   name                  = "${local.vmsName[count.index]}"
20   location              = "${var.region}"
21   availability_set_id   = data.azurerm_availability_set.avset.id
22   resource_group_name   = data.azurerm_resource_group.resource_gp.name
23   network_interface_ids = [azurerm_network_interface.main[count.index].id]
24   vm_size               = "${var.vmsize}"
```

7 *Cloud Service Provider* - fornecedor de serviços de nuvem

```
25    delete_os_disk_on_termination = true
26    delete_data_disks_on_termination = true
27    depends_on = [azurerm_network_interface.main]
28
29    storage_image_reference {
30      id = "${data.azurerm_image.WindowsServer2020.id}"
31    }
32  [...]
```

Listagem 3.2: Ilustração de partes de recursos responsáveis pela criação e importação de alguns componentes na nuvem

O script Terraform utilizado pela aplicação tem como objetivo efetuar as seguintes configurações:

- aprovisionamento de máquinas virtuais de acordo com o estado da infraestrutura e configuração dos recursos associados como por exemplo grupo de recursos, ip privado, DNS, entre outros recursos;
- inclusão da imagem do disco de referência na nova máquina virtual;
- registo do agente do Zabbix na nova máquina virtual;
- instalação do website na nova máquina virtual criada.

Criação de imagem referência para o disco

Uma máquina virtual após ser criada de forma manual esta encontra-se completamente vazia, o que obriga a um conjunto de configurações para torná-la num servidor de *FrontEnd* [14].

Após uma operação de dimensionamento horizontal de *Scale Out*, espera-se que as novas máquinas virtuais possuam o estado de um servidor de *FrontEnd* e sejam configuradas de forma automática. O que implica que no processo de *Scale Out* através do Terraform a imagem do disco é é atribuída na instrução *"azurerm_virtual_machine"* durante aprovisionamento da máquina virtual.

Esta imagem do disco possui alguns pacotes e serviços que são enumerados de seguida:

- IIS, Inkscape, 7zip, elasticSearch, fontes de letra, AzCopy.

Para garantir tal configuração é tirada uma cópia instantânea (*snapshot*) a partir de uma máquina devidamente configurada com o registo do seu estado e é guardada como uma imagem de referência.

Assim, a imagem designada por *"WindowsServer2020"* será carregada na implantação de novas máquinas virtuais e já possuirão o estado que se pretende de um servidor de *FrontEnd*.

Configuração e armazenamento do ficheiro de estado do Terraform

A gestão de estado da nuvem é uma aspeto fundamental no correto funcionamento dos programas de *Terraform*. O *ficheiro de estado* é um ficheiro (com extensão *.tfstate*) responsável pela gestão de estado da infraestrutura realizada pelo Terraform funcionando como uma base de dados do *Terraform*, que com a configuração indicada na Listagem 3.3, este é transferido para uma conta de armazenamento na nuvem.

Sempre que existe a execução do *Terraform init*, um ficheiro com o nome de *Terraform.state* é criado automaticamente no disco do computador que possui informações acerca do histórico de operações realizados pelo Terraform.

Os comandos presentes na Listagem 3.3 do programa de instruções do Terraform, permitem a invocação do *ficheiro de estado* onde são recolhidos os atributos nome do grupo de recursos da nuvem, conta de armazenamento, nome do contentor e uma chave para o contentor, que possuem o ficheiro de estado designado por *tfstate*.

```
1 Terraform {
2   backend "azurerm" {
3     resource_group_name  = "DevOps-AutoScalling"
4     storage_account_name = "autoscale123sa"
5     container_name       = "tfstate"
6     key                  = "Terraform.tfstate"
7   }
8 }
```

Listagem 3.3: Configuração da secção backend no *Terraform* para persistência das configurações para autenticação com o CSP nuvem

O uso desta configuração evita a exposição de dados sensíveis relativos da conta do nuvem como o caso do *subscription_id* e *tenant_id* no Terraform. Como os scripts Terraform estão alojados na nuvem, esta configuração previne o envio desses dados sensíveis para o serviço de controlo de versões quando enviado para o servidor.

Configuração dos contentores para armazenamento do ficheiro de estado

No entanto, o uso de um *backend* para especificar configurações do estado do Terraform implica ao uso de um armazenamento, como por exemplo um contentor para registar o ficheiro de estado na nuvem e como tal, à existência de uma conta de armazenamento e de um contentor onde será registado este ficheiro de estado.

A Listagem 3.3 aponta para a atribuição do nome do ficheiro de estado, do grupo de recursos e do contentor onde será armazenado este ficheiro no Azure. Esta conta de armazenamento permite a criação de contentores, que permite por exemplo o armazenamento do ficheiro *Terraform.tfstate* que contém as definições do o estado do *script* Terraform. Através

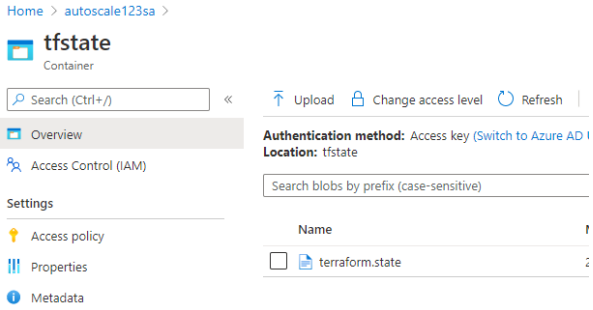


Figura 11: Ilustração do contentor responsável pelo armazenamento o ficheiro de estado do Terraform

da configuração presente na Listagem 3.3, o Terraform tem a capacidade transferir configurações do *Ficheiro de Estado* produzido pelo Terraform para um contentor existente na nuvem, fazendo com que este ficheiro se encontre situado num nível que possa ser acedido e modificado por um serviço remoto como é o caso do *Azure DevOps*. O uso deste tipo de configuração presente na Listagem 3.3 evita a exposição em pleno texto de credenciais e valores relativos às contas de acesso do CSP, como é o caso do *subscription_id*, *client_id*, *client_secret*, *tenant_id*.

3.2.4 *Azure Devops*

De acordo com a Secção 2.2.4, o Azure DevOps é util para executar tarefas de automação e invocar comandos de ferramentas de configuração da infraestrutura.

O Azure Devops, como servidor de automação, é o agente responsável pelas operações de dimensionamento horizontal, pois centraliza e executa de forma sequencial as tarefas que dão origem a acções de *Scale Out* ou *Scale In*. Esta ferramenta disponibiliza uma *Application-Programming Interface (API)* que via *Representational state transfer (REST)*, o utilizador consegue interagir com as pipelines do projeto.

A aplicação *IntelliScaling* via *REST API* tem possibilidade de executar as *pipelines* de um projeto do servidor de automação e de lhe passar parâmetros que são utilizados nas tarefas, a fim de realizar uma dimensionamento horizontal com sucesso. Além de executar tarefas relacionadas com o Terraform (*Terraform init* e *Terraform apply*) que são usados para efetuar alterações na infraestrutura, este permite invocar métodos que foram desenvolvidos na aplicação IntelliScaling, como por exemplo registar de *hosts* no servidor de monitorização e efetuar ajustes no servidor de balanceador de carga.

Mas criar e gerir *pipelines* via utilizando a API da plataforma implica algumas configurações necessárias no portal do Azure Devops [19]:

- criação de uma organização e um projeto no portal do Azure Devops

- o código das pipelines do Azure Devops deve estar inserido num sistema de controlo de versões, como por exemplo *GitHub*;
- o utilizador deve criar um código pessoal de acesso dentro do portal

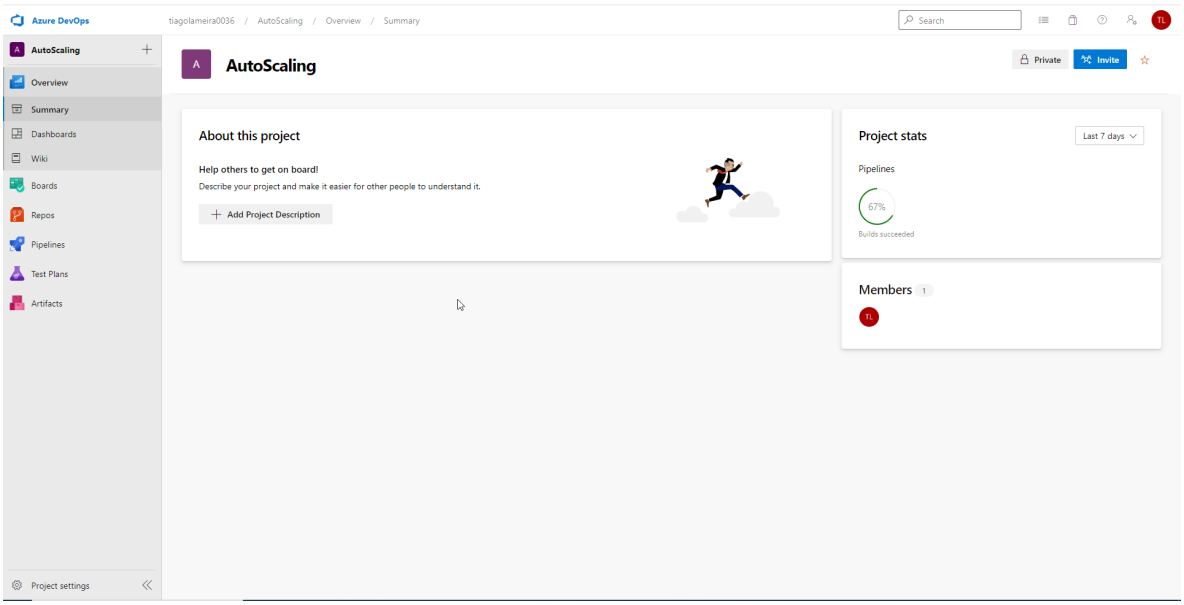


Figura 12: Imagem referente ao projeto criado no Azure Devops

Em Apêndice A.2 é possível consultar algumas configurações realizadas na configuração do projeto e das pipelines no Azure DevOps.

Invocação do Terraform a partir do Azure Devops

Sempre que o Terraform é executado de forma não interativa, como por exemplo através de um servidor de automação, é recomendado o uso de um serviço principal ou de um serviço de identidade gerida para o processo de autenticação [10].

Para cumprir esse requisito, a Listagem 3.4 representa um extrato de código responsável pela autenticação realizada pelo Terraform utilizando o serviço principal do azure. Nesta instrução são invocadas a partir do grupo de variáveis, as variáveis *client_id*, *client_secret*, *subscription_id* e *tenant_id*, implementadas numa secção do portal do Azure DevOps designada por "Biblioteca"que é destinada a esse efeito. No Apêndice A.2 está representada a Figura 72 que mostra a configuração dessas variáveis.

```
1 jobs:
2 - job: Login_Devtest
3   variables:
4   - group: Devtest_login
5   steps:
```

```
6   - script:   az login --service-principal --username $(ARM_CLIENT_ID)
        --password $(ARM_CLIENT_SECRET) --tenant $(ARM_TENANT_ID)
7   - script:   az account set --subscription $(ARM_SUBSCRIPTION_ID)
```

Listagem 3.4: Tarefa 1 responsável pela autenticação com o CSP do Azure através do serviço principal

Estas possuem um prefixo *"ARM"* para serem definidas como variáveis de ambiente e assim serem reconhecidas pelo Terraform na invocação do comando Terraform init.

O serviço principal consiste numa aplicação existente no serviço designado por *Azure Active Directory* que fornece códigos de autenticação utilizados pelo Terraform através dos atributos *client_id*, *client_secret* e *tenant_id*. O uso deste serviço obriga ao registo de uma aplicação no Azure Active Directory, passo necessário para existir possibilidade de extrair estes recursos.

```
1  - job: DevTest_Pipeline
2    dependsOn: Login_Devtest
3    variables:
4     - group: Devtest_backend
5    steps:
6    - script: |
7        Terraform init -backend-config="storage_account_name=
             $(AZURE_STATE_STORAGE_NAME)" -backend-config="resource_group_name=
             $(AZURE_STATE_GROUP_NAME)" -backend-config="container_name=
             $(AZURE_CONTAINER_NAME)" -backend-config="key=
             $(AZURE_BLOB_CONTAINER) "
8      workingDirectory: ./Terraform/Terraform_360_windows/devtest
9      displayName: 'Terraform INIT: Run Terraform Init'
10 [...]
```

Listagem 3.5: Invocação dos comandos Terraform init a partir da pipeline do Azure DevOps

O código explícito na Listagem 3.5 permite executar o comando *Terraform init*, que através do serviço *backend* que permite inicializar o Terraform e carregar as configurações armazenadas na conta de armazenamento da nuvem e assim ser possível executar tarefas de execução do Terraform.

4

DESENVOLVIMENTO

4.1 APLICAÇÃO

A aplicação de gestão *IntelliScaling* atua como agente controlador com a possibilidade de realizar ajustes de capacidade de máquinas virtuais presentes em nuvem e gerir os serviços externos como o caso dos balanceadores de carga, servidores de monitorização e a nuvem.

A aplicação necessita dos seguintes elementos base para ter capacidade de controlar os recursos externos com uma abordagem reativa:

- gestão e sincronização de atributos dos recursos da nuvem e dos balanceadores de carga para a base de dados da aplicação;
- acesso às métricas atualizadas de cada máquina virtual provenientes do sistema de monitorização;
- gestão de regras que comandam de tomadas de decisão;
- um motor de decisão que permite a aplicação tomar decisões de dimensionamento autonomamente;
- comunicação com os serviços externos (identificados na Secção 3.2), com capacidade leitura e escrita;
- dimensionamento vertical e horizontal.

A nível de tecnologias, a aplicação foi desenvolvida com suporte à framework ASP.NET Core (C#) com as tecnologias do lado cliente HTML (*razor engine*), bootstrap, jQuery. A escolha da versão *Core* da framework permite a instalação da aplicação em vários ambientes como o caso dos sistemas operativos MacOS, Linux ou Windows.

A Figura 13 mostra a arquitetura da aplicação que, sendo multi-camada, possui várias bibliotecas independentes para dar a possibilidade futuramente integrar outros serviços externos, sem existir necessidade de alterar em bibliotecas já existentes.

A Figura 13 apresenta os seguintes módulos:

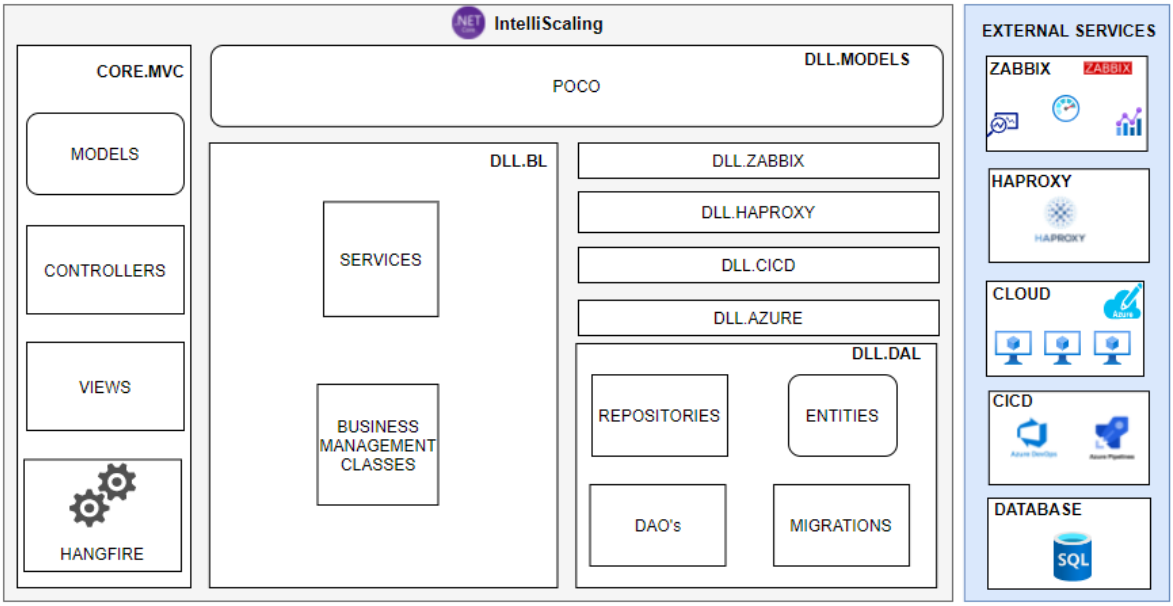


Figura 13: Arquitetura da aplicação IntelliScaling

- Core.MVC - Corresponde à camada de apresentação do projeto, desenvolvida com recurso ao template *ASP.NET Core Web App* que utiliza o paradigma *Model-View-Controller*. Esta camada é constituída pelos componentes *Models*, *Controllers*, *Views* e o Hangfire. As *Views* correspondem à interface gráfica do utilizador, sendo responsáveis por apresentar sob a forma visual os dados provenientes dos *Models*. Os *Controllers* desempenham a função de troca de pedidos entre as *Views* e a camada *Services* da biblioteca "DLL.BL". Os Models são utilizados para armazenar o estado dos dados recebidos pela camada de apresentação. Nesta camada, mais concretamente na classe de arranque, está configurado o *Hangfire*, que como serviço de agendador de tarefas, permite o agendamento de alguns pedidos que são utilizados para operações de dimensionamento automático.

- DLL.BL (biblioteca lógica de negócio) - corresponde à camada responsável pela lógica de negócio e funciona como uma ponte entre a camada *CORE.MVC* e as dependências com o prefixo *DLL*. Possui dois componentes: os *Services*, que são interfaces entre a camada *CORE.Mvc* através dos *Controllers* e a *Business Management Classes*. O outro componente designa-se por *Business Management Classes* que são classes utilizadas para invocar métodos das várias dependências da aplicação (DLL.Zabbix, DLL.Haproxy, DLL.Cicd , DLL.Azure e DLL.DAL).

- Biblioteca DLL.Azure (biblioteca de acesso à nuvem) - oferece métodos para ler e modificar os diversos recursos da CSP da Microsoft - *Azure*. Para mais informações técnicas consultar a Secção A.1.9 do manual de utilizador em anexo.

- Biblioteca DLL.Zabbix (biblioteca de monitorização) - é biblioteca responsável por disponibilizar métodos de gestão dos recursos do servidor de monitorização. Para mais informações técnicas consultar a Secção A.1.8 do manual de utilizador em anexo.
- Biblioteca DLL.Haproxy (biblioteca de balanceador de carga) - contém métodos que permite ler de grupos e atualizar o estado dos servidores web que os constituem. Para Mais informações sobre esta biblioteca estão disponíveis na Secção A.1.10 do manual de utilizador em anexo.
- Biblioteca DLL.CICD (biblioteca de servidores de automação) - disponibiliza métodos para obter informações e executar pipelines de uma dada organização num servidor de automação - *Azure Devops*. Mais informações técnicas disponíveis na Secção A.1.9 do manual de utilizador em anexo.
- Biblioteca DLL.DAL (biblioteca de acesso a dados) - corresponde à camada de acesso a dados, funcionando como um elo de ligação entre a camada de negócio e a base de dados. Utiliza a tecnologia *entity framework* como unidade de trabalho através dos repositórios, disponibiliza métodos para ler e modificar dados na base de dados através das entidades [38] [30]. Contém uma pasta de DAO's, que com base no princípio de separação de responsabilidades, além de permitir desacoplar o código de acesso com a lógica de aplicação, permite criar classes de dados independentes da fonte de dados de uma base de dados relacional [26].
- Biblioteca DLL.Models - correspondem às classes utilizadas para transferir dados por toda a aplicação desde a CORE.WEB até às dependencias designadas por DLL, que com a recurso a um software de mapeamento de objetos designado por "automapper", é realizada a mapeamento de dados nas camadas da aplicação.

Para além destes componentes, a aplicação está ligada a uma base de dados que se encontra na nuvem e armazena métricas, regras, configurações e dados que são recolhidos dos vários serviços externos.

Como suporte ao projeto, num servidor de controlo de versões estão alojados templates das ferramentas de orquestração de infraestrutura e de automação, utilizadas pela aplicação em operações de dimensionamento horizontal.

4.1.1 *Métricas de análise*

Para tomar decisões de forma reativa, a aplicação necessita de ter acesso de forma contínua a um conjunto de métricas de cada máquina virtual. A Figura 14 mostra as várias métricas recolhidas do servidor Zabbix relativas a uma máquina virtual:

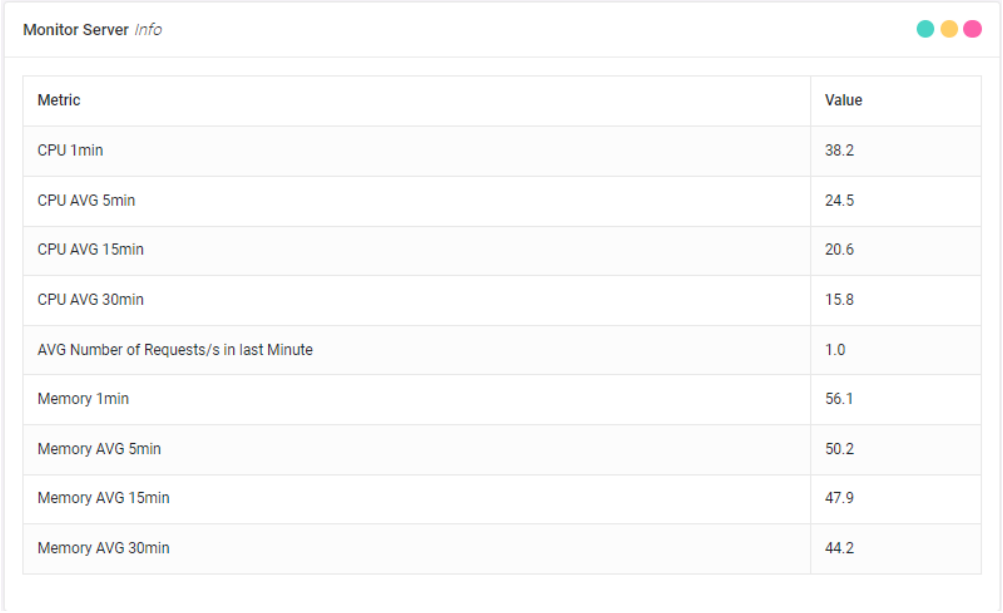


Figura 14: Lista de métricas recolhidas para uma máquina virtual

Os valores na Figura 14 correspondem de forma ordenada às seguintes métricas: valor percentual médio de utilização de CPU nos últimos (1, 5, 15 e 30) minutos, número médio de pedidos registados pelo IIS no último minuto e valor percentual médio de utilização de memória nos últimos (1, 5, 15 e 30) minutos.

4.1.2 *Regras*

Uma regra consiste numa instrução armazenada pela aplicação que define o tipo de métrica, decisor, valor e acção a ser despoletada para tomar uma acção de dimensionamento que pode ser vertical ou horizontal.

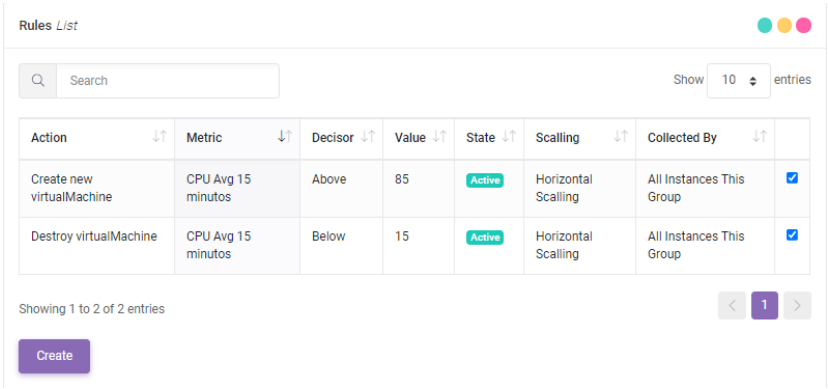


Figura 15: Exemplo de lista de regras ativas para um grupo de máquinas virtuais

A nível lógico de dados este motor de regras está associado a um grupo de máquinas virtuais, permitindo criação de regras, como por exemplo : *"Num determinado grupo de máquinas virtuais, quando a % média de CPU durante 5minutos de uma máquina virtuais tiver um valor percentual inferior a 15%, tome a decisão de diminuir a capacidade da máquina virtual para um tamanho inferior"*. A Figura 16 mostra o formulário para criação da regra para este exemplo atrás descrito.

O formulário de criação da regra permite definir a acção de dimensionamento, um valor de limite para o tipo de operação definido que fica associado a um conjunto de máquinas virtuais e o estado da regra que pode ser ativo ou inativo.

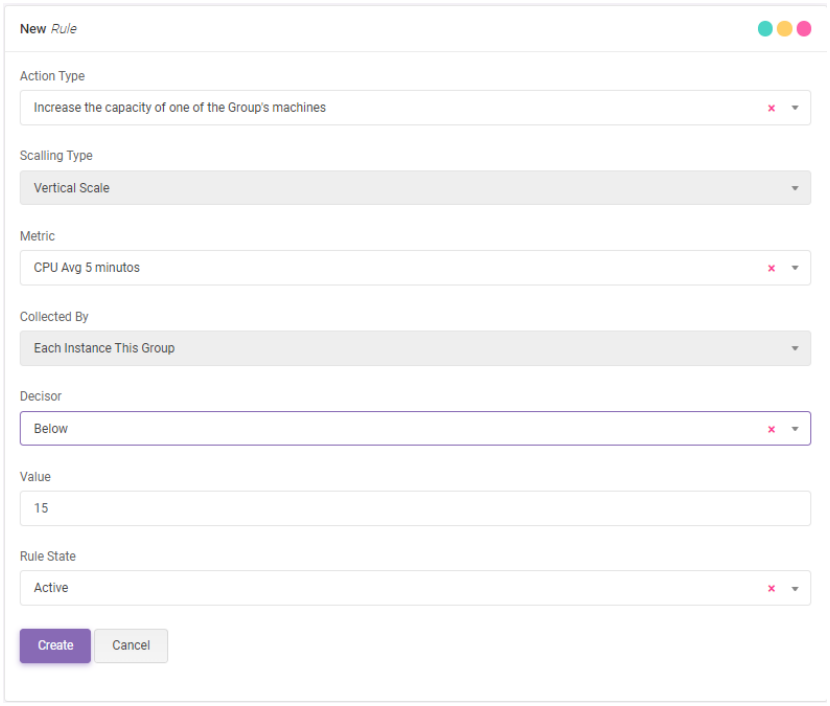


Figura 16: Exemplo de criação de uma regra para acção de *Scale Down*

Após criada, a regra é acrescentada à lista de regras existentes e a Figura 15 exemplifica a lista de regras para um dado grupo. As regras ativas são posteriormente processadas pela função *"Apply Decision"* do agendador de tarefas que verifica se existe alguma tomada de decisão a ser aplicada num grupo de máquinas virtuais. No entanto, para o motor de decisão executar uma decisão das regras presentes num grupo, estas devem obrigatoriamente de estar no estado ativo.

4.1.3 *Motor de decisão*

O motor de decisão é um componente da aplicação com capacidade executar decisões de forma autónoma. De seguida são mostradas duas formas de construir este componente para tornar a aplicação capaz de tomar decisões sem intervenção humana:

- adoptando uma abordagem passiva, no qual a lógica de gestão de regras para os limites estaria no lado do servidor do zabbix.
- adoptando uma abordagem ativa, que desta forma seria a aplicação a aceder ao sistema de monitorização para recolher as métricas e a lógica de criação das regras ficaria do lado da aplicação.

Numa abordagem passiva, as regras são definidas no próprio sistema de monitorização e quando algum limite é ultrapassado, o sistema de monitorização notifica a aplicação IntelliScaling através de *webhooks* e por fim a aplicação tomava uma decisão de dimensionamento vertical ou horizontal.

Esta abordagem apresenta como desvantagem a perda de controlo das regras de decisão e a aplicação *IntelliScaling* estaria completamente dependente de um software de monitorização, impossibilitando de escalar ou trocar para outros serviços de monitorização.

Numa abordagem ativa, compete à aplicação ir recolher métricas necessárias ao servidor de monitorização e tem de ser a própria a gerir regras de decisão e de executar decisões. Esta abordagem implica a adaptação de um agendador de tarefas no projeto que ficará em execução em segundo plano com o pretexto de executar os métodos de recolha de métricas do servidor de monitorização e de aplicar uma decisão de dimensionamento.

Face a estas hipóteses, a decisão recaiu sobre a implementação de uma abordagem ativa com a integração de um agendador de tarefas, pois permite maior controlo sobre o sistema.

4.1.4 *Integração de um agendador de tarefas*

Um agendador de tarefas é um serviço utilizado para executar automaticamente tarefas em segundo plano.

O Hangfire é um agendador de tarefas com suporte a aplicações *ASP.NET Core* que permite executar tarefas em segundo plano do tipo "despoleta e apaga".

A adaptação deste serviço gera a autonomia esperada da aplicação, evitando a necessidade da tomada de decisões manuais por parte de um agente humano.

É utilizado para executar de forma recorrente duas funções da camada de biblioteca de lógica de negócio de forma autónoma, como mostra a Figura 17:

- Função para recolha de métricas

- Função para tomada tomada de decisão

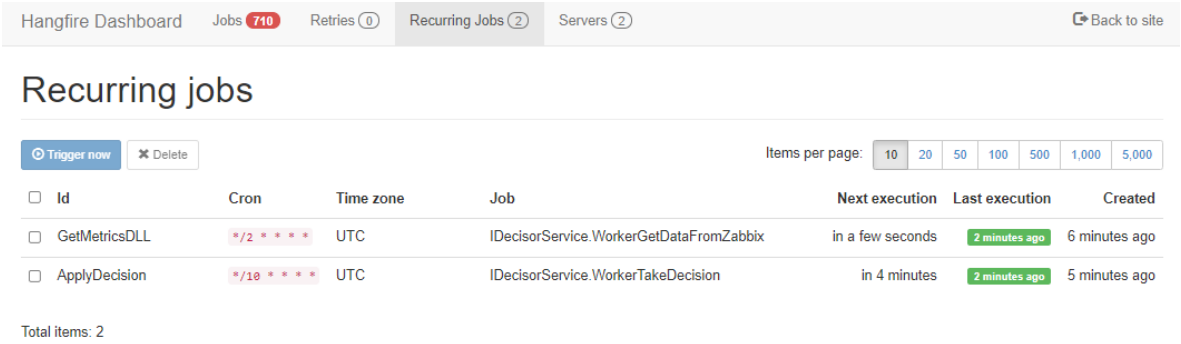


Figura 17: Ilustração das tarefas configuradas no hangfire com a especificação do tempo de recorrência

Função da recolha de métricas

A função presente na Figura 17 designada por *GetMetrics_DLL* é invocada para transferir valores das métricas das máquinas virtuais que se encontram no servidor de monitorização para a base de dados da aplicação de 2 em 2 minutos.

Na parte inicial do desenvolvimento, a consulta das métricas das máquinas virtuais era feita diretamente ao servidor de monitorização no acto da tomada de decisão através da função *Apply Decision*.

Verificou-se após a realização de alguns testes, que a função *Apply Decision* demorava cada vez mais tempo à medida que o número de máquinas virtuais ia aumentando, porque por cada máquina virtual estava a ser feito um pedido ao servidor de monitorização.

Para evitar gargalos de desempenho na função de tomada de decisão, foi criada uma nova tarefa no Hangfire identificada como *GetMetricsDLL*, visível na Figura 17, responsável por invocar uma função com o objetivo de povoamento dos valores das métricas de cada máquina virtual. Ficou definido que o Hangfire de 2 em 2 minutos iria transferir as métricas das máquinas virtuais para a base de dados da aplicação.

A Figura 18 exemplifica através de uma visão geral, como funciona o procedimento invocado no Hangfire designado por *GetMetricsDLL*, para importar para a aplicação, as métricas das máquinas virtuais presentes no servidor de monitorização, de 2 em 2 minutos.

Assim a aplicação possui os valores de monitorização das máquinas virtuais atualizados na base de dados.

Função da tomada de decisão

Tendo acesso a todos os valores necessários e atualizados em base de dados, é através do método *Apply_Decision* que a aplicação verifica se os valores em tempo real das métricas de cada máquina virtual dos grupos superam as regras definidas no motor de decisões a fim de

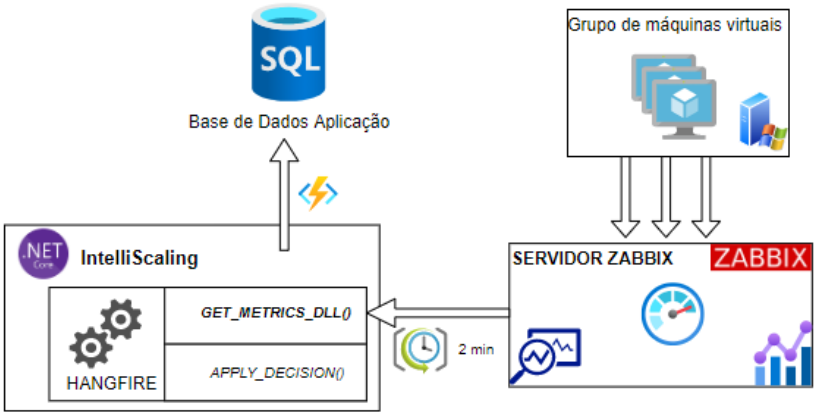


Figura 18: Esquema com o processo de recolha de métricas

levar a cabo uma tomada de decisão. É despoletado pelo Hangfire de 10 em 10 minutos para executar uma tomada de decisão que pode levar a cabo uma operação de dimensionamento vertical ou horizontal.

A escolha do tempo de execução da tarefa foi ponderado em função do tempo médio de execução de uma tarefa de dimensionamento horizontal (é mais demorada que a de dimensionamento vertical e demora aproximadamente 8 minutos) e também para dar hipótese ao servidor de monitorização atualizar as métricas após operação de dimensionamento ocorrida.

No caso de ser escolhido um período de tempo mais curto de execução para esta tarefa, o servidor de monitorização poderia não ter tempo suficiente para atualizar as métricas e poderia desta forma voltar despoletar uma nova operação de dimensionamento sem necessidade.

Algoritmo

Pretende-se verificar se dentro de um grupo de máquinas virtuais existe alguma métrica que supere alguma das regras existentes. As entidades foram definidas no modelo de dados para relacionar os dados referentes a grupos, regras e máquinas virtuais deste modo: Um grupo pode ter várias regras e este é constituído por várias máquinas virtuais e cada máquina virtual tem a dispor os valores das métricas visíveis na Figura 17.

O algoritmo de tomada de decisão possui as seguintes instruções:

1. Recolher de todas as regras activas da base de dados
2. Para cada regra, recolher todos os grupos de máquinas virtuais associados
3. Para cada grupo, buscar todas as suas máquinas virtuais
4. Para cada máquina virtual, verificar se existe alguma métrica que supere as regras:

- Se sim, executa uma decisão
- caso contrário, o pedido é ignorado

No caso de ocorrência de tomada de decisão acima descrita, a aplicação despoleta uma das seguintes acções:

- Desligar máquina virtual
- *Scale Up* (operação de dimensionamento vertical) que consiste num aumento do tamanho da máquina virtual;
- *Scale Down* (operação de dimensionamento vertical) que consiste na diminuição do tamanho da máquina virtual;
- *Scale Out* (operação de dimensionamento horizontal) que resulta na criação de novas máquinas virtuais num grupo de recursos na nuvem.
- *Scale In* (operação de dimensionamento horizontal) que consiste na remoção máquinas virtuais de um grupo de recursos na nuvem.

4.1.5 *Gestão de pedidos concorrentes na tomada de decisão*

O hangfire executa o método *Apply Decision* referente à tomada de decisão de 10 em 10 minutos e no caso de uma operação de dimensionamento demorar mais que 10 minutos a ser executada, o hangfire pode voltar a invocar novamente esta função sem que a anterior tenha sido finalizada.

Pode revelar-se um problema se ocorrer mais do que um pedido concorrente para realizar operação de dimensionamento em simultâneo para máquinas virtuais do mesmo grupo, podendo dar origem a conflitos ou a configurações indesejáveis.

Portanto para evitar tais problemas de concorrência foi necessário efetuar alguns ajustes no algoritmo da função *Apply Decision* para incorporar as seguintes instruções:

- Integração de um semáforo que limita o número de acessos de forma concorrente;
- Criação de um atributo do tipo boleano na tabela do grupo designado por *"In_Progress"*, que permite controlar se existe alguma ação em progresso para o grupo.

A classe *SemaphoreSlim*, consiste num semáforo para limitar o número de acessos permitidos de forma concorrente, para que a função de tomada de decisão seja executada uma vez num intervalo de tempo. No caso de se existir uma operação em progresso, o semáforo

implementado bloqueia a *thread* para descarta novas operações até que a operação seja concluida. Assim que a operação está concluída, a thread é libertada.

Um semáforo é uma primitiva usado para controlar o acesso a um recurso usado por várias *threads* e evita problemas de concorrência num sistema de várias tarefas.

O semáforo permite gerir os pedidos concorrentes, garantindo que apenas é executado um pedido de dimensionamento para um grupo de cada vez.

No caso de ser iniciada uma operação de dimensionamento para uma máquina virtual de um grupo, esse grupo fica bloqueado para não ocorrer mais nenhuma operação de dimensionamento no grupo durante a operação em execução.

Para garantir esse comportamento foi criada na tabela de grupos da base de dados, uma variável designada por "In_Progress"do tipo booleana que possui o valor verdadeiro no caso de estar a ocorrer uma operação de dimensionamento para aquele grupo.

Algoritmo

O algoritmo de controlo de controlo de concorrência é constituído pelas seguintes instruções:

1. Ler todas as regras ativas na base de dados
2. Recolher todos os grupos das regras existentes
3. Ler todas as máquinas virtuais dos grupos
4. Verificar por cada regra ativa se a variável In_progress daquele grupo = false e existir alguma métrica a ser despoletada para as máquina virtuais do grupo
    - Se sim, alterar o estado da variavel In_progress do grupo para 1 que assim pode dar desempenhar uma tomada de decisão e bloqueia próximas decisões
    - caso contrário, manter valor variável In_progress continua a zero

4.1.6 *Cálculo de pesos para o balanceador de carga*

Os servidores web registados num grupo do *HAProxy* recebem uma carga de pedidos proporcional ao seu peso em relação à soma de todos os pesos do grupo, o que significa quanto maior o peso, mais a carga em demanda. Portanto, o valor dos pesos para aplicar em cada servidor web no balanceador de carga tem em consideração o total de servidores web que compõe os grupos *backend* do balanceador de carga e o número de núcleos que diz respeito ao tamanho de cada máquina virtual. O número de núcleos correspondente ao tamanho de uma máquina virtual pode ser consultado na Tabela 2 da Secção 2.2.1.

A fórmula para o cálculo do peso de cada servidor web consiste na divisão entre o número de núcleos de cada máquina virtual com o somatório do número de núcleos de todas as

máquinas virtuais presentes no grupo dos servidores web. Esse valor é multiplicação por cem para devolver um valor inteiro entre zero e cem. No caso de uma máquina virtual estar desligada, o servidor web é considerado inativo e nesse caso é-lhe atribuído um peso de zero. Esta fórmula encontra-se abaixo representada.

$$Peso\_servidor\_web = \frac{Numero\_nucleos\_1VM}{Numero\_nucleos\_todas\_VMs\_Grupo} * 100$$

Esta fórmula para o cálculo do peso está num método dentro da biblioteca de balanceador de carga e é invocada em operações de dimensionamento automático.

4.1.7 *Padrões e Técnicas de Desenvolvimento*

Além de uma solução funcional, a aplicação deve estar para preparada para crescer. Para tal, esta possui o código estruturado em várias bibliotecas e camadas separadas de acordo com as suas responsabilidades, tentando mantendo um nível de acoplamento baixo entre elas.

O acoplamento mede o grau de interdependência entre os módulos de software. Baixo acoplamento é indicador que o software está bem desenvolvido pois significa que são fáceis de manter, reutilizar e substituir, uma vez que código fracamente acoplado oferece maior flexibilidade e maior capacidade de ser reutilizável [33]. Esta secção aborda algumas técnicas e padrões adoptados no desenvolvimento.

Padrão de repositórios

Repositórios são classes ou componentes que encapsolam a lógica necessária para aceder às fontes de dados. Estes centralizam a funcionalidade comum de acesso a dados que assim fornecem melhor capacidade de manutenção, abstraindo-se da infraestrutura ou da tecnologia utilizada para acesso às bases de dados.

Um repositório executa tarefas de intermediário entre a camada do modelo de domínio e o mapeamento de dados, agindo de maneira semelhante a um conjunto de objectos de domínio na memória [25, 31]. Conceptualmente, um repositório encapsula um conjunto de objetos armazenados na base de dados e as operações que podem ser realizadas neles, criando desta forma proximidade com a camada de persistência, permitindo que o programador se concentre apenas na lógica de persistência de dados.

A utilização do padrão de repositórios permite a definição de métodos genéricos por entidade como por exemplo as operações CRUD que sendo transversais às entidades evita a redundância de métodos. A Figura 19 aborda um exemplo de invocação de métodos genéricos para as entidades.

A centralização da lógica de acesso a dados, a facilidade de manutenção e a escalabilidade constituem outras vantagens da utilização de padrão de repositórios no código [31].

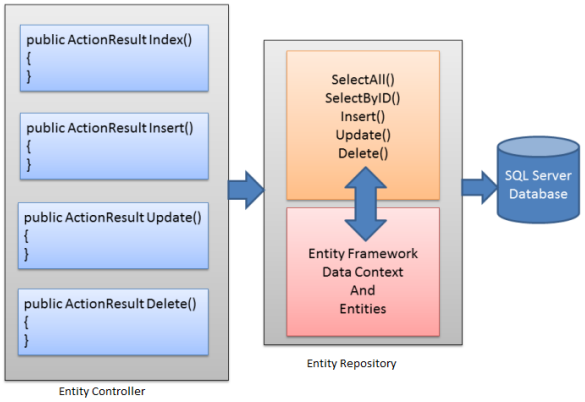


Figura 19: Esquema ilustrando o funcionamento genérico de padrões de repositórios

Injecção de Dependências

A injeção de dependências é um *padrão de design* que permite melhorar a capacidade de reaproveitamento de código [24].

A injeção de dependências é um tipo de inversão de controlo para manipular o controlo de objetos que significa que uma classe deixar de ser responsável por criar ou instanciar objetos dos quais depende [16]. Isto substitui a prática de instânciar classes no código pela injeção da classe diretamente no contrutor do ficheiro, que deste modo permite desacoplar classes, anulando dependência entre elas [23] .

A Listagem 4.1 mostra um exemplo da injeção de dependência de uma classe via construtor.

```
1  public class LoadbalancerManagement{
2      private readonly ILoadbalancerRepository LbRepository;
3      public LoadbalancerManagement(ILoadbalancerRepository LbRepository){
4          this.LbRepository = LbRepository;
5      }
```

Listagem 4.1: Exemplo de injeção de dependência da interface ILoadbalancerRepository no construtor da classe LoadbalancerManagement

Data Transfer Object

Data Transfer Object (DTO) é um padrão de projeto que consiste na criação de classes cujo os seus atributos são utilizado para transportar dados entre diferentes camadas do projeto [44]. Permitem que os dados transitem desde as diversas bibliotecas do projeto até chegarem à aplicação *Core.MVC* e vice-versa, saltando de camada em camada.

Enquanto que uma entidade de domínio está inserido na camada de acesso a dados e mapea os atributos com a base de dados e as validações (*data annotations*), os DTOs estão

situados numa biblioteca independente, estes navegam dentro das bibliotecas e vão mapear os dados das várias bibliotecas, circulando-os por toda a aplicação sem gerar necessidade de extrair tipos de variáveis das bibliotecas.

A Listagem 4.3 representa uma classe DTO para uma entidade designada "Cargo"e a Listagem 4.2 corresponde a uma entidade "Cargo"da camada de acesso aos dados responsável por mapear os dados com a base de dados.

```
1 public class Role{
2     [Key]
3     public int RoleId { get; set; }
4     [MaxLength(255), Required]
5     public string Name { get; set; }
6 }
```

Listagem 4.2: Exemplo de uma entidade de Cargo ligada ao modelo de dados

```
1 public class RoleDTO{
2         public int RoleId { get; set; }
3         public string Name { get; set; }
4 }
```

Listagem 4.3: Exemplo de uma classe DTO de Cargo

Modelo de programação assíncrona

O modelo de programação assíncrona da linguagem *C#* aproveita o suporte assíncrono da framework *.NET*, que consiste numa forma programação paralela que permite que uma unidade de trabalho seja executada separadamente da *thread* principal da aplicação, servindo para evitar gargalos de desempenho e para aprimorar a capacidade de resposta da aplicação.

No modelo síncrono, a aplicação serve-se da *thread* principal e no caso desta ficar bloqueada, a aplicação pára de responder e tem que esperar até que seja finaliza a operação. O mesmo não acontece com os métodos assíncronos pois a interface gráfica continua a responder e o utilizador pode continuar a efetuar operações na aplicação enquanto a atividade continua a processada sem existir o bloqueio desta [28].

Este modelo é essencial para tarefas de acessos a bibliotecas na web ou a ficheiros de imagens, que possam ser demorados evitando que bloqueiem a aplicação.

IntelliScaling está desenvolvida com base no modelo assíncrono devido à necessidade de efetuar acções em simultâneo sobre os componentes distribuídos (nuvem, servidor de monitorização, servidor de balanceador de carga e servidor de automação), sem existir necessidade do utilizador aguardar que a aplicação finalize a operação.

```
1 public async Task TurnOnAsync(){
```

```
2       await ivm.StartAsync();
3       ivm.Refresh();
4   }
5   public void TurnOnSync(){
6       ivm.Start();
7       ivm.Refresh();
8   }
```

Listagem 4.4: Distinção entre o método assíncro *versus* síncrono

A Figura 4.4 ilustra o método síncrono *versus* assíncrono para o contexto de ligar uma máquina virtual. Apesar de ambos os métodos terem a mesma finalidade, o *TurnOnSync()* quando é despoletado a aplicação só voltará a reagir quando a máquina virtual estiver ligada. O mesmo nao acontece com o método assíncrono *TurnOnAsync()* que mesmo que a tarefa não esteja finalizada, o utilizador pode navegar na aplicação enquanto a máquina virtual está a ser ligada.

4.2 DIMENSIONAMENTO VERTICAL

Dimensionamento vertical consiste em alterar o tamanho da máquina virtual em resposta à carga de trabalho que esta recebe e tem associadas duas possíveis acções: *Scale Up* - aumenta o tamanho de máquinas virtuais e *Scale Down* - reduz o tamanho de máquinas virtuais [43]. Este processo de dimensionamento é realizado pela aplicação que foi programada para aceder diretamente ao CSP da nuvem modificar o tamanho da máquina virtual. No entanto, o utilizador pode configurar operações de dimensionamento vertical de duas formas possíveis:

- através de abordagem reativa - o utilizador define um conjunto de regras e a aplicação através do motor de decisão (referido na Secção 4.1.3) analisa as regras e toma decisões de forma automática (esquema da Figura 20);
- através de uma abordagem manual - sempre que o utilizador quiser, através de um botão para cada acção, este tem a liberdade de acionar no imediato uma acção de *Scale Up* ou *Scale Down* sobre máquinas virtuais de um grupo.

A Figura 20 ilustra de modo geral a forma como a aplicação interage o sistema numa operação de dimensionamento vertical para a abordagem reativa. A aplicação está ligada à base de dados, pois segundo a Secção 4.1.4, a aplicação tem na sua base de dados as métricas de cada máquina virtual atualizadas a cada dois minutos. De acordo com a Secção 4.1.4, através do hangfire, a função de tomada de decisão do hangfire de 10 minutos em 10 minutos confirma se existe alguma operação de dimensionamento vertical ser despoletada.

Se existir, a aplicação comunica com o balanceador de carga e com a nuvem via *REST API* para realizar as alterações.

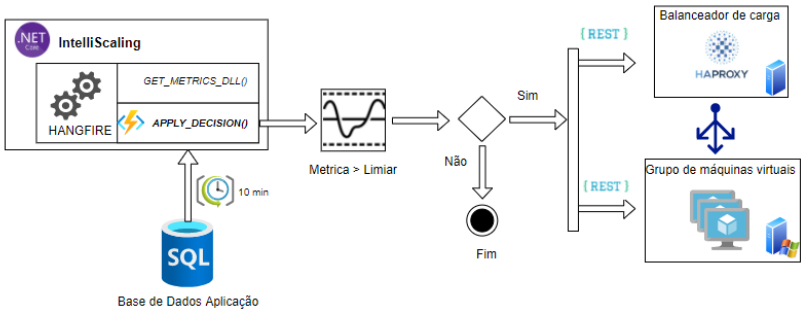


Figura 20: Esquema no dimensionamento vertical para a abordagem reativa

Sempre que esta ação é despoletada, a aplicação desempenha um procedimento garantir elevada disponibilidade, as máquinas virtuais não podem ficar desligadas sem um prévio aviso ao servidor de balanceador de carga porque alterar a capacidade da máquina implica que a máquina seja reiniciada para aplicar a acção.

Para evitar indisponibilidade do site, durante esta acção é necessário efetuar um procedimento sobre o balanceador de carga, desativando o webserver enquanto a respetiva máquina virtual está a ser dimensionada. Assim que a máquina virtual esteja novamente operacional, o webserver pode voltar a ser ativado.

A aplicação tanto na abordagem reativa como a manual efetua a seguinte ordem de tarefas para a evitar *indisponibilidade* do site que está a correr nas máquinas virtuais.

1. Desativar o servidor web correspondente à máquina virtual do grupo do balanceador de carga.
2. Alterar o tamanho da máquina (operação que provoca reinício da máquina virtual)
3. Atualizar os pesos dos servidores web para adaptar a carga tendo em conta do tamanho da máquina virtual
4. Ativar novamente a máquina virtual no grupo do balanceador de carga

Desativação do servidor web

A primeira acção do algoritmo de dimensionamento vertical consiste na desativação do webserver do grupo do balanceador de carga, a que corresponde à máquina virtual que será dimensionada.

Dentro da biblioteca de gestão de balanceadores de carga está presente um método que dado um grupo e uma máquina virtual, existe forma de desativar o servidor web do balanceador de carga. O método *GetWebServerByVM(VirtualMachineDTO vm, GroupDTO g)*

tem a capacidade de dada uma máquina virtual, é possível retornar um webserver cujo o *ip privado* seja igual ao da máquina virtual. Assim na função de *DisableServer(string grupo, string webserver)* é possível indicar qual o servidor web a ser desativado.

```
1 public void TakeOutWebServerGroup(VirtualMachineDTO vm, GroupDTO g){
2     WebServerDTO w = GetWebServerByVM(vm, g);
3     this.client.DisableServer(g.Name, w.Name);
4 }
```

Listagem 4.5: Método de desativação do servidor web presente na biblioteca de gestão do balanceador de carga

A partir deste momento, aquela máquina virtual deixa de receber pedidos sempre que os utilizadores acedem ao site.

Scale Up / Scale Down

Dada a informação que o webserver se encontra desativado do balanceador de carga, é seguro proceder para a alteração do tamanho da máquina virtual na nuvem.

O objetivo consiste em aceder à nuvem através dos métodos *Scale Up* ou *Scale Down* presentes na biblioteca de gestão de nuvem, presentes na Secção A.1.9, e aplicar o tamanho pretendido. Inicialmente é feita a leitura do atributo tamanho da máquina virtual no momento atual, para recolher o tamanho anterior ou seguinte dependente do tipo de acção. Como suporte à seleção de tamanho, foi definida uma lista de tamanhos por família que as máquinas virtuais podem ter.

Os métodos *Scale Up* e *Scale Down* apenas diferem entre si na seleção do tipo de tamanho da máquina virtual, isto é, o método *Scale Up* está preparado para ser selecionado tamanho seguinte, enquanto que o método *Scale Down* seleciona o tamanho anterior em relação ao tamanho atual que a máquina virtual possui no momento.

```
1 public async Task UpScale(){
2     VirtualMachineSizeTypes AtualSizeVM =
          CastStringToVirtualMachineSizeTypes(this.Size);
3     VirtualMachineSizeTypes GetNewSizeVM = SelectNewSize("up", AtualSizeVM);
4     await ivm.Update().WithSize(GetNewSizeVM).ApplyAsync();
5 }
```

Listagem 4.6: Exemplo método *Scale Up*

Selecionado o tamanho pretendido, o próximo passo consiste em aplicar esse novo tamanho e atualizar máquina, que com esta alteração, a máquina virtual será reiniciada e possuirá então o novo tamanho.

Cálculo dos pesos no balanceador

Feito o processo de atualização do tamanho da máquina virtual na nuvem o próximo passo consiste na cálculo dos pesos que cada webserver irá possuir no grupo do balanceador de carga.

A fórmula utilizada para o cálculo dos pesos do balanceador de carga está presente na Secção 4.1.6

Registo das alterações na aplicação

Uma vez atualizados os valores dos pesos dos webservers e do tamanho da máquina virtual na nuvem, falta registar as alterações efetuadas relativas a tamanho da máquina virtual, número de núcleos e peso na base de dados da aplicação. Para criar um histórico das alterações que ocorreram a nível do grupo, na tabela de *Logs* foi criada um novo registo a informar que o tamanho da máquina foi atualizado.

Ativação do servidor web

Finalizado a alteração de tamanho, o último passo consiste em ativar novamente o servidor o web relativo à máquina virtual, através do método visível na Listagem 4.7, que deste modo a máquina passará novamente a disponibilizar o site e a receber pedidos dos utilizadores.

```
1 public async Task ScaleUp(VirtualMachineDTO vm, GroupDTO singleGroup){
2     [...]
3     await this.LbMang.TakeInWebServerFromPool(vm, singleGroup);
4 }
```

Listagem 4.7: Reativação do servidor web do balanceador de carga

4.3 Dimensionamento Horizontal

O dimensionamento horizontal consiste em adicionar ou remover recursos de um sistema e pode fazer-se sentir de duas formas: *Scale Out* - adicionar máquinas virtuais ou *Scale In* - remover máquinas virtuais. Neste processo, o desafio consiste em executar a ordem de tarefas indicadas na Secção 3.1.4 e também tornar a aplicação capaz de criar várias instâncias com base num determinado critério:

- Criar instâncias idênticas a máquinas de produção, num determinado grupo de recursos de uma dada subscrição da nuvem, que implica que as novas instâncias possuam certas características antes de serem colocadas em produção, atualizar os pesos no balanceador de carga e por fim registar as instâncias no servidor de monitorização.

A automatização deste processo de dimensionamento horizontal pode ser conseguido através de uma ferramenta de orquestração como o caso do *Terraform*, que permite indicar quais os componentes que se pretende que uma dada infraestrutura tenha através de ficheiros de código.

A aplicação necessita de um servidor de automação como o *Azure DevOps* para executar scripts Terraform e efetuar alterações nos serviços externos através de templates de formato *yaml* que se encontram num servidor de controlo de versões como o caso do *GitHub*. Permite executar várias tarefas de automação utilizadas para o processo de dimensionamento horizontal automático de forma sequencial, como exemplificadas na Figura 23

Como a IntelliScaling e o Azure DevOps são entidades independentes, a comunicação é feita via *REST API*, que deste modo a aplicação tem a capacidade de transferir para o servidor um conjunto de variáveis de interesse e também de executar a pipeline. É a camada da aplicação DLL.CICD que está preparada para executar estas funções.

Tal como no dimensionamento vertical, as operações de dimensionamento horizontal podem ser despoletadas através de uma abordagem manual (que através de um botão permite gerar no imediato uma acção de *Scale Out* ou *Scale In*) ou através de uma abordagem reativa. Através da abordagem reativa, a aplicação analisa o motor de regras através do agendador de tarefas e com base nas métricas recebidas das várias máquinas virtuais, pode ser executada uma acção de *Scale Out* ou *Scale In*.

A Figura 21 mostra o sistema desenvolvido, mostrando a adaptação da aplicação ao Azure DevOps, com o impacto nos serviços externos.

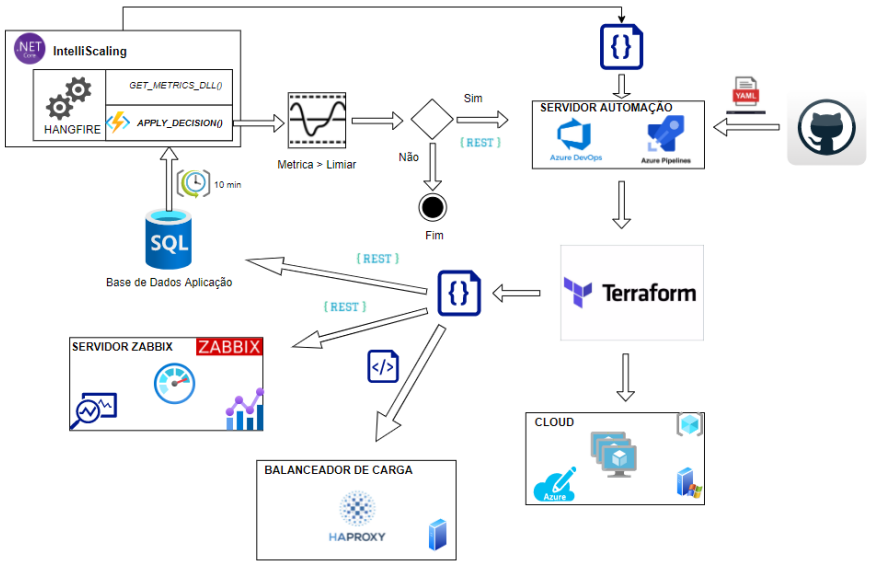


Figura 21: Esquema no dimensionamento horizontal para a abordagem reativa

Processo de dimensionamento

Uma acção de dimensionamento horizontal gerida pela *IntelliScaling* assenta em algumas operações executadas numa pipeline do Azure DevOps.

De acordo com a Figura 21, o primeiro passo do processo compete à aplicação invocar a pipeline do portal *Azure DevOps*, com o tipo de acção a ser despoletada (*Scale In* ou *Scale Out*).

A Figura 22 mostra o estado da pipeline quando é despoletada e se encontra em processo de execução.

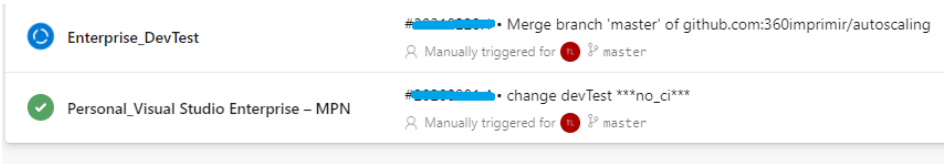


Figura 22: Exemplo da pipeline com a designação de "Enterprise_DevTest"do Azure DevOps no estado de execução

Sempre que ocorre uma operação de dimensionamento horizontal, a aplicação prepara e transfere uma estrutura de dados com informação relativa da nuvem, do balanceador de carga e do servidor de monitorização para o Azure DevOps, que são enumerados de seguida:

- **Tipo de operação:** Scale In ou Scale Out;
- **Informações da nuvem:** nome da subscrição, *nome do grupo de recursos da nuvem*, nome da *região*, nome do *conjunto de disponibilidade*, *máquinas virtuais* de um determinado grupo de recursos, *tamanho* das máquinas virtuais, nome do *subgrupo de rede*, nome da *rede virtual*, IP Privado;
- **Informações do balanceador de carga:** *nome do grupo* do balanceador de carga, DNS do servidor de balanceador de carga, nome dos servidores web;
- **Informações do servidor de monitorização:** Nome dos Hosts, IP Privado da máquina do servidor de monitorização, DNS da máquina do servidor de monitorização.

A partir do momento que é desencadeada a acção, a *pipeline* fica em execução, como exemplifica a Figura 22 e prepara-se para executar o conjunto de tarefas indicado na Figura 23.

Dada a necessidade do Terraform aceder à infraestrutura que será aprovisionada, a primeira tarefa a ser executada pela pipeline, consiste na autenticação via *"service principal"* numa determinada subscrição do CSP onde serão publicadas as alterações, visível na Figura 23, processo que se encontra explicado na Secção 3.2.4 do capítulo do Sistema.

Os próximos passos desempenhados na pipeline consistem na execução do Terraform, através dos comandos *Terraform Install, Init, Plan* e *Apply*, que permitem instalar o serviço Terraform no servidor de automação e executar os script de Terraform, referenciado na Secção 3.2.3, que possuem as instruções definidas para aprovisionar recursos na infraestrutura.

De acordo com a Figura 21 referente ao esquema de processo de dimensionamento, ao ser executado o comando de Terraform Apply gera dois resultados:

- aprovisionamento dos recursos na nuvem como o caso das máquinas virtuais, que perante uma operação de *Scale Out* é criada uma nova máquina virtual e os seus componentes associados (interface de rede, firewall, entre outros que são explicados na Secção 4.3.
- uma variável com o resultado do dicionário e IPs Privados das máquinas virtuais de um grupo

Obtendo-se do resultado do Terraform, as últimas tarefas consistem na utilização na configuração dos serviços externos (balanceador de carga, servidor de monitorização e registo em base de dados). A configuração dos serviços externos como o caso do servidor de monitorização e registo de base de dados são feitas invocando os endereços da aplicação via *REST API* responsáveis por registar estas alterações, no entanto para registar novos servidores web devem ser feitas diretamente no ficheiro de configuração do *haproxy*.

Aprovisionamento das Máquinas Virtuais através do comando Terraform apply

Como os scripts Terraform são escritos numa linguagem declarativa, devem indicar o número de recursos esperado na infraestrutura final após operação de dimensionamento horizontal.

Por exemplo, no caso de existirem duas máquinas virtuais num grupo de recursos na nuvem com o nome *PoolServer1*, *PoolServer2* e a aplicação despoleta uma operação de *ScaleOut* com o objetivo de criar outra nova máquina virtual, então o Terraform deve receber como parâmetro o nome da nova máquina virtual que se pretende que seja criada (por exemplo PoolServer3)

O nome das máquinas virtuais num grupo de recursos é atribuído pela equipa de gestão de infraestrutura com base no número de servidores web existentes num grupo de balanceadores de carga de um determinado mercado. Por exemplo, se existem duas máquinas virtuais com o nome *PoolServer1* e *PoolServer2*, uma terceira máquina virtual deve ser nomeada como *PoolServer3*.

O comando *Terraform apply* utiliza o objeto das variáveis de fonte de dados transferidas desde a aplicação IntelliScaling até ao servidor do *Azure DevOps* que posteriormente são recolhidas pelo Terraform através do ficheiro *variables.tf* utilizadas na configuração de recursos no acto de criação ou remoção de máqinas virtuais numa subscrição da nuvem.

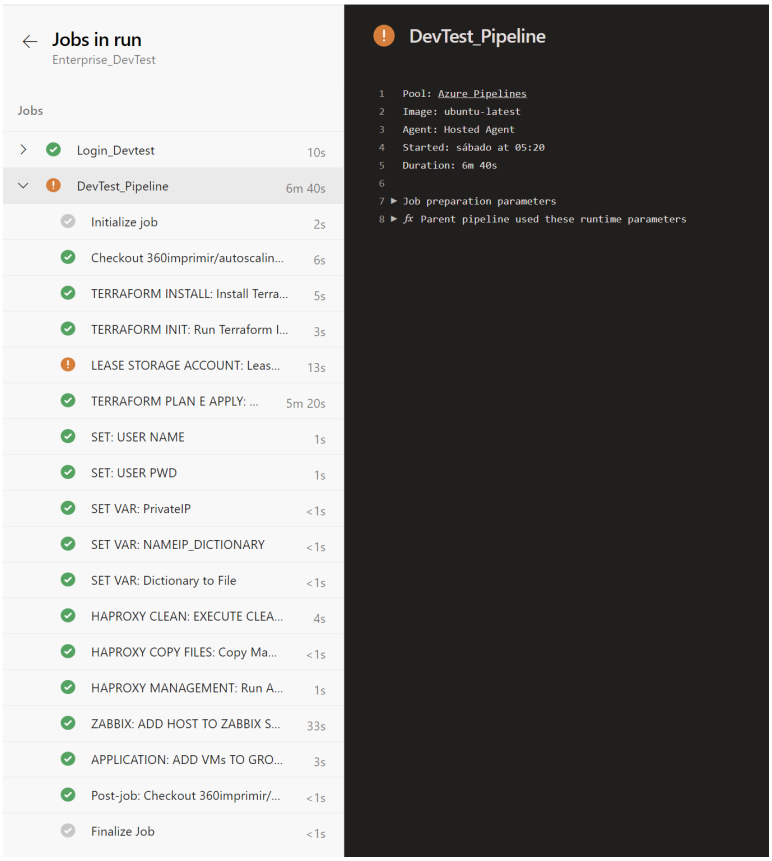


Figura 23: Lista de tarefas executadas pelo Azure DevOps no processo de dimensionamento horizontal

A Listagem 3.2 presente na Secção 3.2.3 apresenta uma parte do *script Terraform* que aprovisiona máquinas virtuais com um determinado tamanho e configura as num determinado grupo de recursos, rede, sub-rede, *firewall*, conjunto de disponibilidade da nuvem e região.

A instrução *"TF_VAR"* permite que as variáveis sejam transferidas por ambiente e tenham a capacidade chegar como parâmetro ao *script* de *Terraform* de forma automática para aprovisionar as máquinas virtuais.

O código presente na Listagem A.24 ilustra como exemplo a instrução *Terraform apply* executado a partir da pipeline do *Azure DevOps* onde são transferidas as variáveis do *Azure Devops* para o *Terraform*.

```
1 variable "vmnameList" {
2     type = string
3 }
4 [...]
5 locals {
6   vmsName = tolist(split(",", var.vmnameList))
```

```
7 }
```

Listagem 4.8: Definição das variáveis num ficheiro designado por *variables.tf* usado depois pelo script principal do *Terraform*

Contudo, antes da lista de máquinas virtuais ser afixada à secção responsável, a Listagem 4.8 ilustra o processo efetuado para converter a string que contém a lista com os nomes das máquinas virtuais para uma lista, para ser processada pelo resto do *script*. Assim, a variável com o nome *vmsName* já se encontra no tipo de variável ideal para ser trabalhado pelo resto do script.

Um aspeto necessário durante o aprovisionamento de uma máquina virtual consiste na atribuição de algumas das características, isto é, a nova máquina virtual ao ser criada deve ser associada a recursos já existentes na infraestrutura como por exemplo:

- região, grupo de recursos, rede virtual, sub-rede, conjunto de disponibilidade.

Para associar a máquina virtual a estes recursos, é necessário realizar a importação destes recursos dentro do Terraform que são enviados a partir da aplicação. A Listagem 4.9 mostra como é feita a importação de um grupo de recursos.

```
1 data "azurerm_resource_group" "resource_gp" {
2   name = "${var.rgoup}"
3 }
```

Listagem 4.9: Exemplo da importação do grupo de recursos

Com estes recursos já importados, o Terraform possui as informações necessárias para criar a máquina virtual. No entanto, quando é aprovisionada a máquina virtual, são também criados outros recursos que estão associados à máquina virtual como o caso do **IP Público**, **interface da placa de rede** (*NIC*) e **grupo de segurança da rede** (*NSG*).

Além dos comandos levados a cabo pelo script Terraform para configurar todos os recursos associados à maquina virtual é fornecida uma imagem do disco para a nova máquina virtual.

A Listagem 4.10 exemplifica o processo de configuração dum disco a partir de uma imagem de uma máquina virtual durante processo aprovisionamento exercido pelo Terraform.

Esta imagem foi preparada pelo processo *Sysprep* antes da realização do *Snapshot* da máquina virtual e possui o sistema operativo e algumas configurações que disponibilizam um servidor web em funcionamento.

```
1 data "azurerm_image" "WindowsServer2019" {
2   name                = "Teams-Base-Image-WS2019v2-01042020"
3   resource_group_name = "ImagesRG"
4 }
```

Listagem 4.10: Secção responsável por criar um disco a partir de uma imagem existente de uma máquina virtual durante o aprovisionamento

Essa imagem permite que a máquina virtual tenha ao dispor software necessário como *7zip*, IIS, ElasticSearch, para tornar esta máquina virtual com o mesmo estado que os outros servidores de *FrontEnd*. Para mais informações consultar a Secção 3.2.3.

Após execução do comando Terraform apply, é devolvido um *dicionário* que associa o nome da máquina virtual com o seu IP privado (Figura 24), estrutura de dados utilizada nas tarefas seguintes da pipeline.

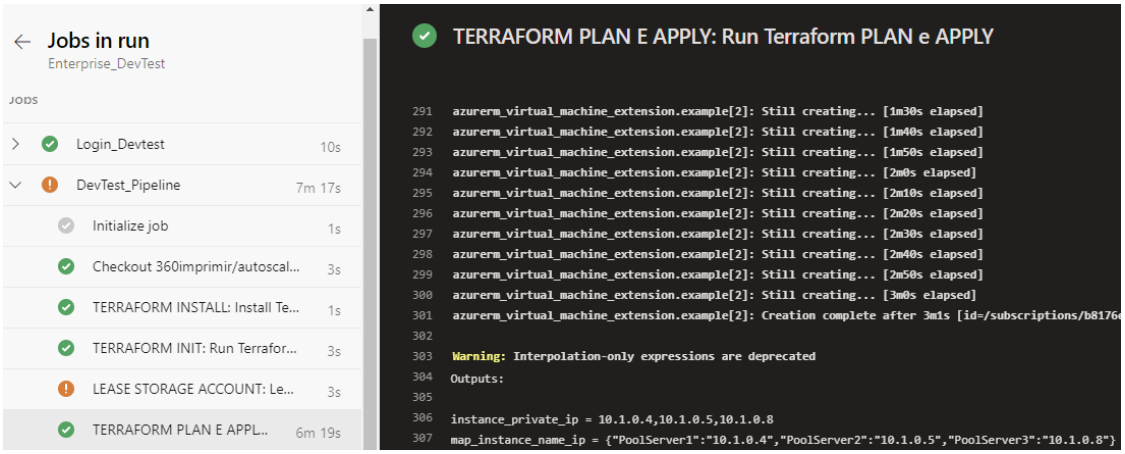


Figura 24: Resultado da tarefa de execução do Terraform apply com obtenção do output

Configuração do agente do Zabbix nas máquinas virtuais

Como o processo de criação de máquinas virtuais é automático sem intervenção humana, a instalação do agente do Zabbix é automatizado a fim da máquina virtual ser monitorizada.

Como referido na Secção 3.2.1, numa acção de *Scale Out*, é necessário configurar o agente do Zabbix na nova máquina virtual, para a aplicação receber métricas desta.

O Terraform através do recurso *azurerm_virtual_machine_extension* permite o aprovisionamento das máquinas virtuais, mas também permite efetuar tarefas de automação como o caso de configuração de componentes e instalação de software nas máquinas virtuais *"on the fly"*, como o caso do agente do zabbix.

A Listagem 4.11 ilustra uma parte do script Terraform usado para configurar o agente do zabbix.

```
1  resource "azurerm_virtual_machine_extension" "example" {
2    name                 = "CreateZabbixAgent_${local.vmsName[count.index]}"
3    virtual_machine_id   = azurerm_virtual_machine.main[count.index].id
4    [...]
5    settings = <<SETTINGS{
6    "fileUris": ["https://www.zabbix.com/downloads/4.4.10/zabbix_agent
         -4.4.10-windows-amd64-openssl.msi"],
```

```
7       "commandToExecute": "msiexec /I
            zabbix_agent-4.4.10-windows-amd64-openssl.msi
            server=${var.monitor_privateip} sport=10050 lport=10050
            SERVERACTIVE=${var.monitor_privateip} rmtcmd=0 /qn"}
8   SETTINGS}
```

Listagem 4.11: Método de registo do agente do Zabbix nas novas máquinas do virtuais

Para configuração do agente do zabbix, esta extensão encontra o caminho *url* do executável que pretende ser instalado na máquina virtual através da instrução *"fileUris"* e o comando *msiexec* desempenha a função de instalar e configurar o ficheiro de instalação.

Como se verifica na Figura 25, o agente do Zabbix encontra-se em execução e permite que a máquina virtual passe a ser monitorizada pelo servidor do zabbix.

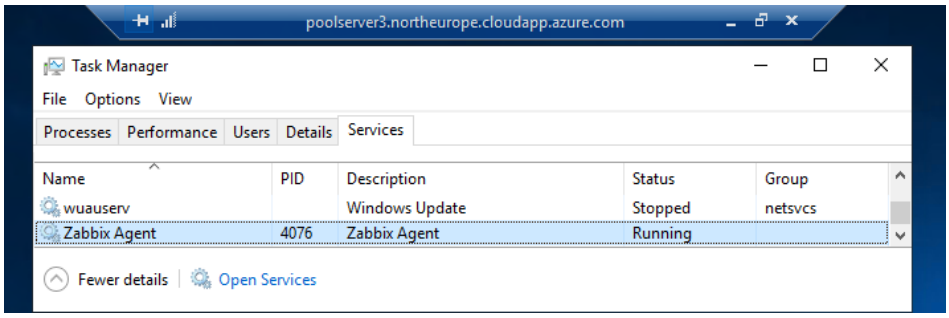


Figura 25: Agente do Zabbix em execução na máquina aprovisonada

Implantação do website

Uma tarefa realizada numa ação de *Scale Out* durante o aprovisionamento da máquina virtual consiste na instalação do website no servidor aplicacional da máquina virtual, a fim da máquina virtual ficar pronta para servir.

Uma abordagem para instalação do website na máquina virtual consiste no uso de ferramentas de *Continuous Deployment* para configuração de software como o caso do *Octopus Deploy*, que é especializada na distribuição de pacotes de software em máquinas virtuais de forma simples .

No entanto, para tornar a instalação do software um processo mais simples, ficou decidido que a instalação da aplicação seria realizado com a utilização de uma configuração partilhada (*Shared Configuration*) que fica armazenada numa conta de armazenamento (*Storage Account*) na nuvem, centralizando os ficheiros do website num só sítio.

A funcionalidade de configuração partilhada permite aos administradores colocar a configuração do IIS num volume partilhado e disponibilizar para múltiplos servidores web.

Utilizando um recurso partilhado de rede, qualquer alteração nos ficheiros de configuração central serão propagadas pelos diferentes servidores sem necessidade de outras ferramentas

ou de reprogramação, promovendo escalabilidade, alta disponibilidade, confiabilidade e capacidade de gestão de aplicações. A configuração partilhada faz com que máquinas virtuais acedam ao ficheiro de configuração presente na rede como se tratasse de uma configuração local, o que significa que qualquer ajuste nos ficheiros de servidor aplicacional estarão disponíveis para outros servidores.

Este conceito consiste em associar uma unidade de rede no computador à conta de armazenamento de ficheiros da nuvem através de uma partilha de ficheiros *File Share* do Azure que contém os ficheiros de configuração do IIS e da pasta do website. Isto permite partilhar ficheiros centralizados na nuvem por vários computadores de destino de forma simultânea.

Esta funcionalidade implica a existência de 2 recursos:

- Uma conta de armazenamento, mais concretamente uma Partilha de Ficheiros (File Share) na nuvem, responsável por armazenar a pasta dos componentes do website e da configuração do IIS. Da partilha de ficheiros (*File Share*) é extraído caminho a apontar para a localização da partilha de ficheiros.

- Uma configuração do IIS extraída de uma máquina virtual que foi configurada com configuração partilhada. Essa configuração é depois utilizada em todas as máquinas virtuais criadas sobre o processo de dimensionamento horizontal.

Na conta de armazenamento está presente a pasta com os ficheiros do site responsáveis pelo seu funcionamento. Esta pasta é configurada para ficar presente na diretoria do disco da máquina virtual configurada em modo de atalho ou *symlink*, que é uma pasta virtual no disco a apontar para um caminho físico da conta de armazenamento.

Configuração necessária:

Para fazer uso da configuração partilhada na nuvem, o primeiro passo consiste em colocar os ficheiros e pacotes do website numa pasta na conta de armazenamento. Essa será a pasta central lida pelo IIS que contém os ficheiros de instalação do website.

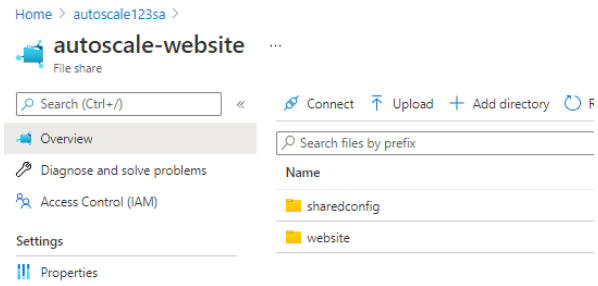


Figura 26: Código que permite definir caminho de rede para a conta de armazenamento

O próximo ponto consiste em configurar a unidade de rede para a localização da conta de armazenamento num servidor em funcionamento, com as credenciais da conta de armazenamento, cujo o **nome de utilizador** e a **palavra passe** correspondem ao nome da conta de armazenamento e à chave da conta de armazenamento respetivamente. A *File Storage*, sendo um serviço da conta de armazenamento da nuvem, oferece armazenamento partilhado que permite ao utilizador mapeá-la como uma unidade de rede para o computador destino, disponibilizando através de um link, dando a possibilidade de montar a partilha de ficheiros, como mostra a seguinte imagem:

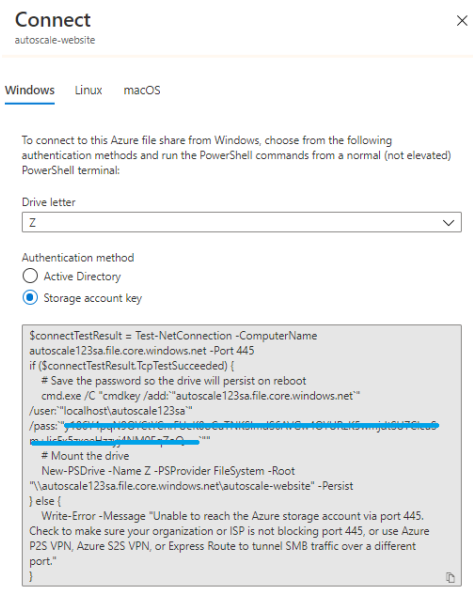


Figura 27: Código que permite definir caminho de rede para a conta de armazenamento

Este ponto corresponde à parametrização da configuração partilhada que é realizada no servidor aplicacional (IIS) de uma máquina com o objetivo de lhe extrair a configuração de uma máquina funcional para apontar para o volume de rede da conta de armazenamento.

No IIS, o primeiro passo consiste em definir na *Pool* do website a pasta da unidade de rede. Para definir a configuração partilhada, o IIS disponibiliza uma opção "Shared Configuration"onde o utilizador deve realizar as seguintes alteraçoes:

- inserir o **caminho físico dos ficheiros**, que corresponde ao caminho da unidade de volume de rede que aponta para a conta de armazenamento onde será armazenado este ficheiro da configuração partilhada.
- inserir os dados da conta de utilizador, que dizem respeito ao nome de utilizador e palavra-passe da conta de armazenamento.
- ativar a opção "configuração partilhada".

Com estes dados preenchidos, o utilizador pode selecionar a opção para exportar configuração partilhada como mostra a imagem seguinte.

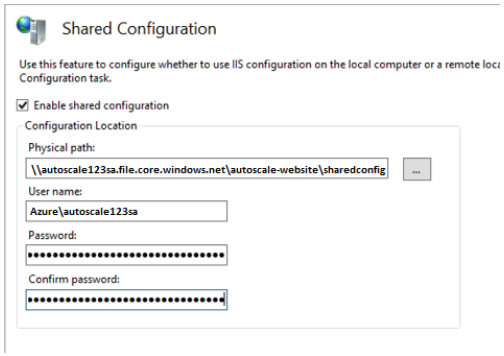


Figura 28: Exemplo de exportação da configuração partilhada para a nuvem

Ao criar uma nova máquina virtual, desde que esta possua o IIS configurado, se lhe for fornecida este ficheiro de configuração, automaticamente este servidor aplicacional consegue disponibilizar o website através do browser, após o seu reinício.

Para tornar este processo automático num processo de *Scale Out* é conseguido com a ajuda do Terraform, que faz esta instalação do website em segundo plano.

Automatizar este processo de implementação da configuração partilhada pode ser conseguido através do Terraform que por intermédio de scripts com tarefas automação em *Powershell*.

O Terraform através da extensão *"azurerm_virtual_machine_extension"* permite copiar e executar scripts powershell mas que inicialmente necessita de os copiar para a máquina virtual destino com o sistema operativo *Windows* que se encontra no estado de aprovisionamento. Esses ficheiros powershell após serem executados vão desempenhar as tarefas abordadas na Secção 4.3 referentes à configuração partilhada na máquina virtual, que após executadas coloca o website a servir na nova máquina de forma automática.

Interação com o servidor de balanceador de carga

Após concluída a execução da tarefa do Terraform para acrescentar ou remover máquinas virtuais num grupo de recursos na nuvem, a próxima tarefa consiste em atualizar o ficheiro de configuração do haproxy no servidor de balanceador de carga com as máquinas virtuais resultantes do processo de aprovisionamento.

A Figura 29 ilustra uma parte do ficheiro de configuração *haproxy.cfg* onde estão definidas as seguintes instruções relativas aos servidores web agrupados por cada grupo-backend:

- nome grupo backend, nome do servidor web / máquina virtual, valor peso de cada servidor web, ip privado da máquina virtual

```
frontend loadbalancer
        mode http
        stats enable
        stats uri /haproxy/stats
        default_backend backend.360imprimir.pt
        stats refresh 30s
        stats show-node
        bind *:80
        stats admin if TRUE   #responsible for attr checkboxes e set
backend backend.360imprimir.pt
        server FE-PoolServer3 10.1.0.8:80 weight 33 check
        server FE-PoolServer2 10.1.0.5:80 weight 33 check
        server F3-PoolServer1 10.1.0.4:80 weight 33 check
        balance roundrobin
        option forwardfor

backend backend.api-360imprimir.pt
        server FE-PoolServer2 10.1.0.5:80 weight 50 check
        server FE-PoolServer3 10.1.0.8:80 weight 50 check
        balance roundrobin
        option forwardfor
```

Figura 29: Exemplo de haproxy.cfg com 2 backends "backend.360imprimir.pt"e "backend.api-360imprimir.pt"e os respectivos servidores web

Como a biblioteca de gestão de balanceador de carga da IntelliScaling (*DLL.Haproxy*), não possui um método para inserir e remover servidores web via REST API, recorreu-se a técnicas de processamento de texto para efetuar alterações do ficheiro de configuração para inserir e apagar servidores web (relativa às máquinas virtuais) dentro dos grupos backend do balanceador de carga.

Inicialmente é armazenado o resultado do *output* do comando *Terraform apply* para um ficheiro de formato de texto, que se nomeou como *DictionaryWithIpAndName* (presente na Listagem A.27) porque entre as tarefas da pipeline as variáveis de *output* do Terraform perdem o estado (*stateless*). Como o ficheiro de texto *DictionaryWithIpAndName.txt* possui numa estrutura de dados a informação relativa aos IP's privados e nome das máquinas virtuais, este pode ser transferido para um script que efetue o processamento necessário para registar as máquinas virtuais num grupo do balanceador de carga.

Esse script designado por *manageHaproxy.sh* possui comandos *Sed* responsáveis por pesquisar e atualizar padrões de texto de cada linha de configuração do ficheiro *haproxy.cfg*. Para isso, o script necessita de obter o valor do conjunto de variáveis descritas anteriormente (nome grupo backend, nome da máquina virtual, peso, ip privado da máquina virtual) para aplicar essas alterações no ficheiro destino.

```
1 sample=$1
2 for instance in $(jq '. | keys | .[]' <<< $sample); do  # get json
3     sed -i "/$(jq -r ".[$instance]" <<< $sample):80 /d"
          /etc/haproxy/haproxy.cfg
4     sed -i -e "/^backend $3/a\ \t server $(jq -r "." <<< $instance) $(jq -r
          ".[$instance]" <<< $sample):80 weight $4 check"
          /etc/haproxy/haproxy.cfg
5 done
```

```
6  sudo systemctl restart haproxy
```

Listagem 4.12: Parte do script designado *manageHaproxy.sh* para adicionar e remover instâncias de máquinas virtuais no ficheiro de configuração haproxy.cfg

O código presente na Listagem 4.12 mostra o script *manageHaproxy.sh*, que se encontra armazenado no servidor de controlo de versões e é enviado para o servidor de balanceador de carga com a função de atualizar o ficheiro *haproxy.cfg* com as informações de cada servidor web.

A tarefa da pipeline referenciada na Listagem 4.13 permite enviar o script via *Secure Shell (SSH)* para a máquina do servidor de balanceador de carga, que para além de lhe serem fornecidas permissões de leitura e escrita, executa o script com o propósito de incluir os seguintes argumentos no ficheiro de configuração *haproxy.cfg*:

- $1 - dicionário que contém o nome da máquina virtual seguido do IP, proveniente do output do Terraform;
  $2 - tipo de acção, que possuem os valores *add* no caso de *Scale Out* ou *remove* no caso de *Scale In*, que corresponde à flag que indica se é para adicionar ou remover servidores web do haproxy;
  $3 - nome do grupo do balanceador de carga onde serão definidos os servidores web;
  $4 - peso atribuído ao servidor web.

O envio deste processo foi conseguido através de uma tarefa na pipeline do Azure DevOps que se encontra representada na Listagem 4.13 que permite automatizar o registo de servidores web relativo às máquinas virtuais aprovisionadas na nuvem sem que haja necessidade do utilizador configurar manualmente os servidores web num determinado grupo do servidor de balanceador de carga.

```
1  - script: |
2    sshpass -p $VMAP ssh -tt $VMAU@${{parameters.lb_dns}} 'sudo chmod +x
         /tmp/manageHaproxy.sh'
3    sshpass -p $VMAP ssh -tt $VMAU@${{parameters.lb_dns}} "sudo
         /tmp/manageHaproxy.sh '$(cat DictionaryWithIpAndName.txt)' add
         ${{parameters.lb_groupname}} ${{parameters.peso}}"
4  workingDirectory: ./Terraform/Terraform_360_windows/devtest
5  displayName: 'HAPROXY MANAGEMENT: Run Add Instances to HAProxy'
6  condition: eq('${{parameters.operation}}', 'ScaleOut')
```

Listagem 4.13: Registo no servidor do balanceador de carga das atualizações das máquinas virtuais provisionadas pelo Terraform

Atualização dos registos no servidor de monitorização

As máquinas virtuais criadas pelo Terraform ainda não estão a ser monitorizadas pelo Zabbix, apesar de cada uma delas já possuir um agente configurado que lhe permite enviar as métricas para o destinatário, porque estas ainda não foram registadas no servidor do zabbix.

No servidor de monitorização a entidade a ser registada designa-se por *Host* e para o seu registo é necessária a informação do nome e ip privado da máquina virtual, lista de templates a que se pretende associar o host (Anexo 3.2.1) e o grupo do servidor de monitorização. Em anexo encontra-se a Listagem A.29 que mostra o pedido HTTP a partir da tarefa da pipeline do *Azure DevOps* com destino à aplicação IntelliScaling para registar máquinas virtuais no servidor de monitorização.

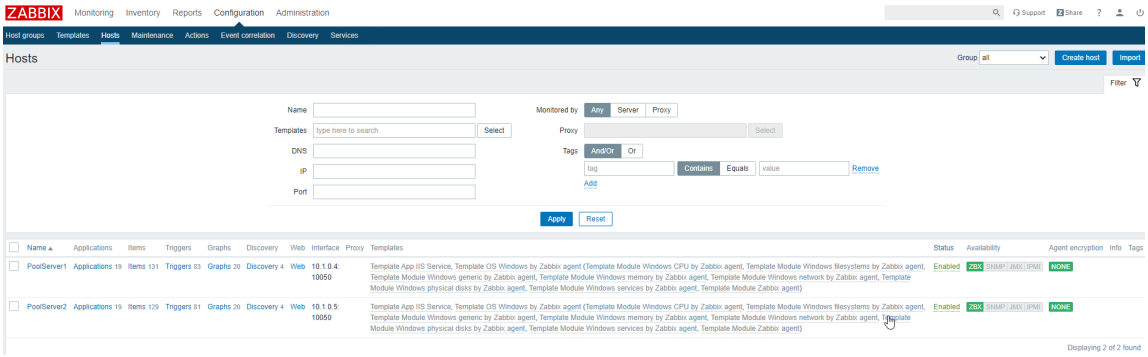


Figura 30: Listagem dos hosts registados no servidor de monitorização

Todos estes atributos são enviados para a biblioteca de lógica de negócio da aplicação que antes de serem registados no servidor de monitorização, são processados conforme o valor do tipo de operação e também validados para não inserir em duplicado máquinas virtuais com o mesmo nome e IP pois compromete a monitorização das mesmas no servidor.

A partir do registo do host, este passa a estar ativo no servidor de monitorização e assim a máquina virtual passa a ser monitorizada pelo zabbix, que feito isto, a aplicação IntelliScaling já conseguirá aceder às suas métricas.

Registo das alterações na aplicação

O processo do dimensionamento horizontal automático só fica concluído após registo das entidades criadas ou apagadas na base de dados, para a aplicação IntelliScaling ter acesso ao estado do sistema após as tarefas de dimensionamento automático.

Este passo é essencial para o controlo aplicação sobre os recursos existentes, disponibilizando ao utilizador a informação via interface gráfica alterações que ocorreram em segundo plano e o estado atual do sistema.

Sempre que ocorre uma operação de dimensionamento automático, é registado numa tabela de Logs a hora de início da operação, tipo de operação e descrição da operação de dimensionamento ocorrida. Isto permite que o utilizador da aplicação tenha conhecimento das operações de dimensionamento ocorridas num grupo de máquina de virtuais na nuvem.

Além disso, com o registo das novas máquinas virtuais na camada de persistência, estas passam recolher métricas pelo agendador de tarefas e seja atualizado o fator de peso do balanceador de carga em função do numero de máquinas virtuais e número de núcleos que cada uma possui.

Em sentido inverso, quando existe uma operação de remoção de máquinas virtuais, estas são removidas da base de dados para não fornecer informações enganosas aos serviços terceiros, acerca de nome, número e pesos das máquinas virtuais existentes na camada de persistência.

```
1 [...]
2 - script: |
3   curl -k -H 'Content-Type: application/form-data' -H 'Content-Length: 0' -X
        POST
        "https://bizayautoscale-devops.azurewebsites.net/WebAPI/ImportVMToDAL?
        PrivateipList=$PIP&GroupName=${{parameters.lb_groupname}}&
        operation=${{parameters.operation}}"
4 displayName: 'APPLICATION: ADD VMs TO GROUP APPLICATION'
5 workingDirectory: ./Terraform/Terraform_360_windows/devtest
6 name: ImportToApplication
```

Listagem 4.14: Pedido à aplicação responsável por registar as máquinas com os IPs privados fornecidos à aplicação

A Listagem 4.14 ilustra a comunicação entre a tarefa da pipeline e o método da aplicação responsável por receber os atributos com a função de encaminhar para a função de atualização dos registos na base de dados. A aplicação recebe os atributos "Lista de IPs privados das máquinas virtuais, nome do Grupo do balanceador de carga necessários para efetuar o processamento dos dados antes de serem adicionados em base de dados.

No contexto de uma operação de *Scale In* para apagar as máquinas virtuais obsoletas, é necessário realizar uma associação das máquinas da nuvem que correspondem à lista de IPs privados que são enviados por parâmetro, com o objetivo de retornar uma lista de objetos de máquinas virtuais para serem comparadas com as existentes no modelo de dados. Isto é, o codigo presente na Listagem A.30 realiza uma intersecção entre as máquinas que correspondem aos IPs provenientes por parâmetro e as existentes em base dados, com o objetivo de apagar todas resultantes da interseção. Assim apenas se mantém em armazenamento as máquinas virtuais cujo os IPs chegam por parâmetro e as restantes são eliminadas.

5

CASOS DE ESTUDO E RESULTADOS

Este capitulo mostra o caso de estudo e os resultados de uma atividade experimental com o objetivo de analisar se a aplicação possui a capacidade de despoletar operações automáticas de dimensionamento vertical e horizontal em máquinas virtuais alojadas em nuvem e realizar ajustes nos vários serviços externos.

5.1 CASO DE ESTUDO

Como caso de estudo, é analisado comportamento do sistema num dia de *Black Friday* em que existe uma grande afluência de utilizadores às lojas online. As promoções da campanha provocam um aumento do número de acessos aos websites da loja online e como consequência um aumento da carga computacional nas máquinas do sistema.

Para simular esse processo, são realizados testes de carga com o intuito de saturar as máquinas virtuais presentes no sistema e avaliar a capacidade que a aplicação tem de reagir para a equilibrar o poder de processamento em função da carga recebida.

À medida que a aplicação recolhe métricas de monitorização das máquinas virtuais, esta deve ter a capacidade de agir na infraestrutura e ajustar os seus recursos de forma autónoma com base na carga recebida, aplicando operações de dimensionamento vertical ou horizontal, de acordo com as regras definidas - abordagem reativa.

Inicialmente, a aplicação é configurada com regras correspondendo a limiares de *threshold* para o tipo de métrica *percentagem de utilização de CPU* proveniente das máquinas virtuais, pois a utilização de CPU das máquinas virtuais é uma métrica muito analisada pelas equipas de manutenção de infraestrutura para estimar a carga.

Os testes de carga são realizados com o apoio da aplicação *Apache JMeter*, que permite realizar testes de carga e medir o desempenho de um website, com a elaboração de um plano de testes no qual é definido um número de utilizadores que irá aceder ao website de forma concorrente durante um intervalo de tempo.

Configuração do ambiente de testes

A Figura 31 mostra a infraestrutura de testes que serve como ponto de partida para as 2 simulações que serão realizadas. A aplicação IntelliScaling está ligada a cada um dos componentes da infraestrutura que se encontra montada de acordo com os pontos referidos na Secção 3.2 relacionada com a configuração do sistema.

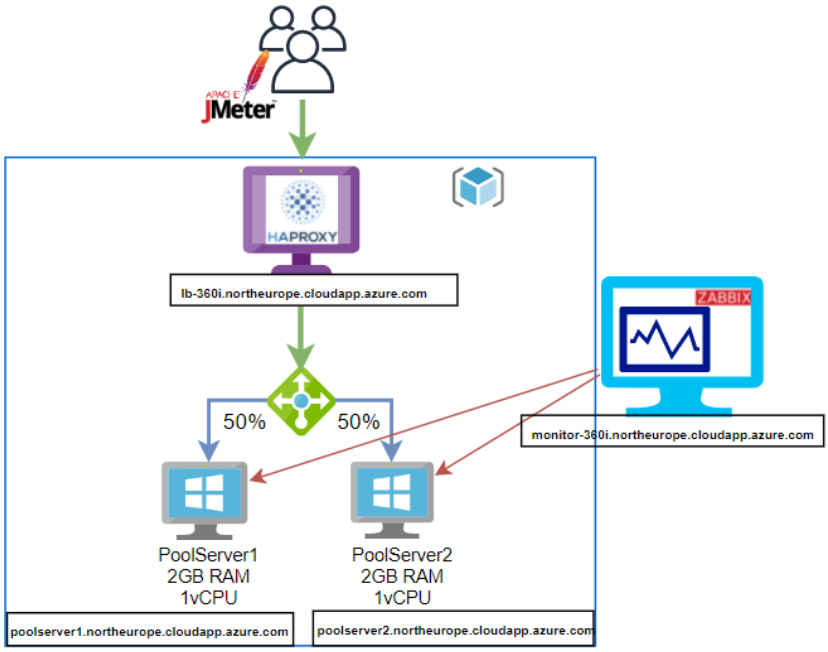


Figura 31: Imagem da Infraestrutura de caso de estudo para realização de testes de stress

Esta infraestrutura para a realização das simulações é constituída pelos seguintes componentes:

- Uma máquina virtual com o sistema operativo Linux que possui o *HAProxy* configurado para servir de balanceador de carga para os dois servidores de FrontEnd, configurado para distribuir 50% dos pedidos, por cada um possuir o mesmo tamanho. Este balanceador de carga possui como DNS: *"lb-360.northeurope.cloudapp.com"*.
- Duas máquinas virtuais denominadas por *PoolServer1* e *PoolServer2*, com um tamanho de *Standard_B1ms* (designação Azure), possuem o sistema operativo Windows e atuam como servidores de FrontEnd com um DNS: "*poolserver1.northeurope.cloudapp.com*" e "*poolserver2.northeurope.cloudapp.com*" respetivamente. Estas máquinas virtuais possuem apenas um servidor web configurado e em cada uma existe um agente do Zabbix que envia métricas para o servidor de monitorização.

- Uma máquina virtual com o sistema operativo Linux a atuar como servidor de monitorização, utilizada para monitorizar os dois servidores de FrontEnd Esta máquina possui como DNS *"monitor-360i.northeurope.cloudapp.azure.com"*.

Fluxo do tráfego

No JMeter é definido um plano de teste de carga onde são definidos um número de utilizadores que acedem a um endereço do site que se encontram dentro de cada máquina virtual.

O plano de teste de carga é configurado para o tráfego de pedidos ser apontado para o DNS do balanceador de carga, que por sua vez, os distribui para cada uma das máquinas virtuais.

Como o HAproxy atua como um *reverse proxy*, um utilizador ao aceder ao DNS do balanceador de carga, o pedido é reencaminhado para um dos servidores web também designados como FrontEnds, fazendo com que os pedidos os clientes ou utilizadores cheguem até às máquinas virtuais.

O servidor de monitorização recolhe e armazena constantemente métricas de cada máquina virtual, e envia esses dados para a aplicação *IntelliScaling* interpretar esses registos.

Cenário Inicial

A Tabela 5 mostra a informação inicial e o ponto de partida da infraestrutura antes da realização das simulações 1 e 2.

Informação no Azure			Informação no HAProxy		
Nome VM	**Tamanho**	**Nº núcleos**	**WebServer**	**IP privado**	**Peso**
PoolServer1	Standard_B1ms	1	poolserver1	10.1.0.4	50
PoolServer2	Standard_B1ms	1	poolserver2	10.1.0.5	50
Grupo de recursos: DevOps-AutoScaling			Grupo: backend.360imprimir.pt		

Tabela 5: Informação das máquinas virtuais na nuvem e no HAProxy

Na nuvem está definido um grupo de recursos designado por *"DevOps-AutoScaling"* com duas máquinas virtuais designadas por *PoolServer1* e *PoolServer2*, com a configuração *Standard_B1ms* (2GB de RAM e 1 vCPU, que significa que cada uma possui um único núcleo de acordo com Tabela 2 das especificações).

No servidor do balanceador de carga, através do ficheiro de configuração *haproxy.cfg*, as duas máquinas estão instanciadas dentro de um grupo designado por *backend.360imprimir.pt*, cada uma com um peso de 50%, que significa que a quantidade de pedidos é dividida de forma igual.

Para a realização das simulações, as máquinas virtuais utilizam tamanhos da família *tipo B* por se tratar de uma família de tamanhos equilibrada para cargas de trabalho porém consegue ser mais económica que as família *D* ou *F*, como descrito na Tabela 2.

Como cenário inicial no servidor de monitorização encontram-se definidos dois *hosts* (consultar Figura 32) que correspondem às duas máquinas virtuais *PoolServer1* e *PoolServer2*, permitindo agregar métricas de cada máquina.

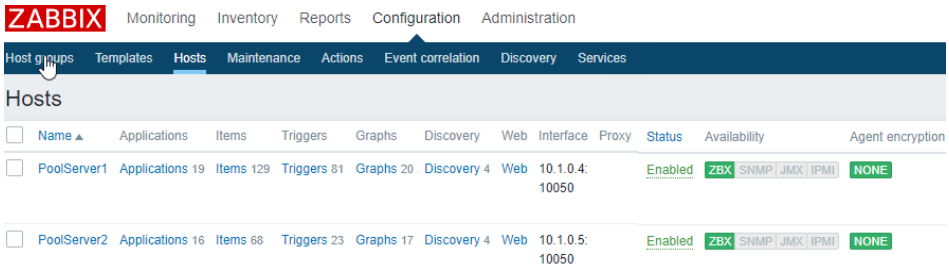


Figura 32: Número de hosts definidos no cenário inicial dos testes de carga

5.2 SIMULAÇÕES

Nesta secção são descritas as duas simulações para testar operações de dimensionamento automático, vertical e horizontal.

A aplicação *IntelliScaling* de forma autónoma está atenta a variações de carga ocorridas neste conjunto de máquinas e toma açcões em função das regras definidas. Estas operações ocorrem graças ao agendador de tarefas (*Hangfire*), que em segundo plano executa a tarefa de tomada de decisão a cada 10 minutos.

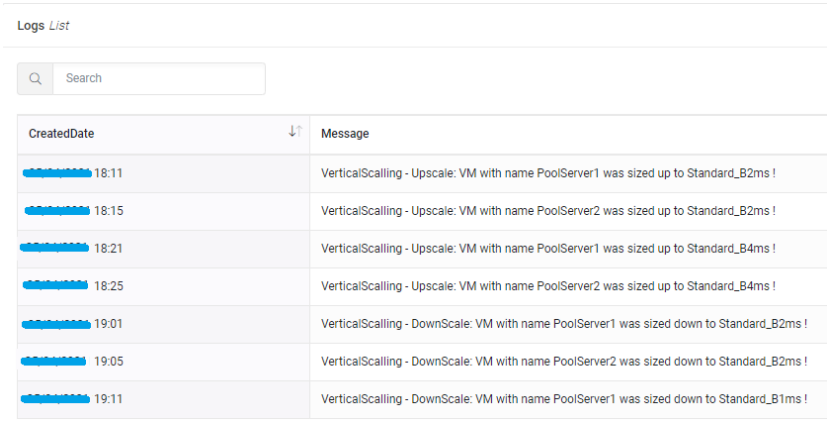
No caso de ocorrer uma operação, é registada numa tabela de *Logs* da base de dados, a qual é lida e disponibilizada de forma visual aos utilizadores através de uma listagem, mostrando eventos ocorridos sobre cada grupo de máquinas.

A Figura 33 mostra como exemplo a listagem indicando as acções de *Scale Up* e *Scale Down* ocorridas num grupo de recursos, indicando a data de operação e qual o tipo de operação.

5.2.1 *Simulação 1 – Dimensionamento vertical*

Nesta simulação é testada a capacidade da aplicação provocar acções de dimensionamento vertical, *Scale Up* ou *Scale Down*, perante situações de instabilidade provocadas pelo teste de carga sobre o sistema de máquinas definidas no cenário inicial presente na Secção 5.1.

A aplicação *IntelliScaling* de forma autónoma está atenta a estas variações de carga ocorridas neste conjunto de máquinas e tomará ações se necessário em função das regras definidas mas é importante salientar que apenas pode acontecer uma operação de dimensionamento



Logs *List*

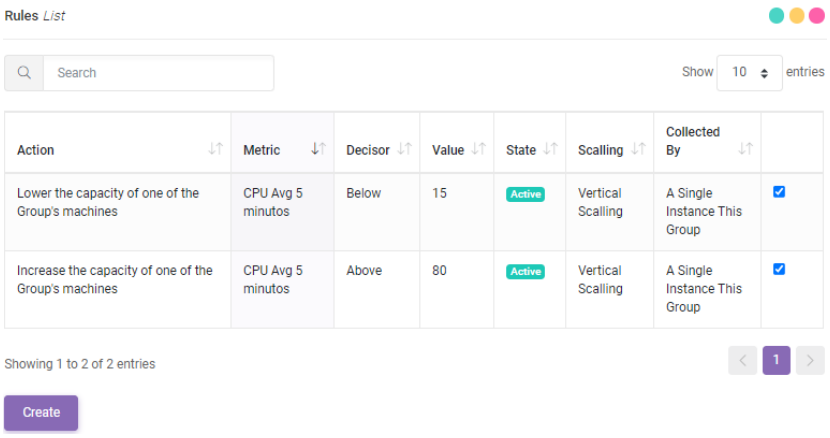
Search

CreatedDate	Message
[redacted] 18:11	VerticalScalling - Upscale: VM with name PoolServer1 was sized up to Standard_B2ms !
[redacted] 18:15	VerticalScalling - Upscale: VM with name PoolServer2 was sized up to Standard_B2ms !
[redacted] 18:21	VerticalScalling - Upscale: VM with name PoolServer1 was sized up to Standard_B4ms !
[redacted] 18:25	VerticalScalling - Upscale: VM with name PoolServer2 was sized up to Standard_B4ms !
[redacted] 19:01	VerticalScalling - DownScale: VM with name PoolServer1 was sized down to Standard_B2ms !
[redacted] 19:05	VerticalScalling - DownScale: VM with name PoolServer2 was sized down to Standard_B2ms !
[redacted] 19:11	VerticalScalling - DownScale: VM with name PoolServer1 was sized down to Standard_B1ms !

Figura 33: Tabela de Logs da aplicação que mostra os eventos ocorridos na realização do teste de carga

vertical numa máquina de cada vez, para garantir que fica pelo menos uma operacional para servir o website.

Esta simulação permite analisar o comportamento da aplicação a realizar operações de dimensionamento vertical sobre as máquinas virtuais, que de forma automática tem a capacidade de alterar o seu tamanho mas também de alterar os pesos no balanceador de carga em função do tamanho de cada máquina.



Rules *List*

Search

Show 10 entries

Action	Metric	Decisor	Value	State	Scalling	Collected By	
Lower the capacity of one of the Group's machines	CPU Avg 5 minutos	Below	15	Active	Vertical Scalling	A Single Instance This Group	☑
Increase the capacity of one of the Group's machines	CPU Avg 5 minutos	Above	80	Active	Vertical Scalling	A Single Instance This Group	☑

Showing 1 to 2 of 2 entries

1

Create

Figura 34: Sistema de regras ativas definidas para dimensionamento vertical

Regras definidas

A Figura 34 mostra as duas regras configuradas que correspondem aos limites, para a aplicação despoletar acções *Scale Up* e o *Scale Down*.

Regra nº1: a primeira regra presente na figura provoca uma tomada de acção sempre que o valor médio percentual dos últimos 5 minutos de qualquer uma máquinas virtuais que compõe o grupo for inferior a 15%. A aplicação deve tomar a iniciativa de baixar o tamanho

dessa máquina virtual para o nível inferior, isto é, se nos últimos 5 minutos a percentagem média de CPU for inferior a 15%, deve existir uma operação de *Scale Down* nessa máquina.

Regra nº2: A segunda regra presente na figura implica que quando o valor percentual dos últimos 5 minutos de qualquer uma das máquinas que compõe o grupo for superior a 80%, a aplicação deve tomar a iniciativa de aumentar o tamanho dessa máquina virtual para um nível superior. Isto é, se nos últimos 5 minutos a percentagem média de CPU for superior a 80%, deve acontecer uma operação de *Scale Up* sobre a máquina virtual.

Com as regras ativas, a aplicação já possui controlo sobre a infraestrutura. É possível iniciar os testes de carga com auxílio da ferramenta *Apache JMeter* com o objetivo de saturar as máquinas virtuais, cujas características podem ser consultadas na Tabela 5.

Plano de teste

O gráfico presente na Figura 35 representa o plano do teste stress definido no *JMeter*, que relaciona o número de utilizadores em função da duração, com o objetivo de induzir carga nas máquinas virtuais durante 1h e 20 minutos.

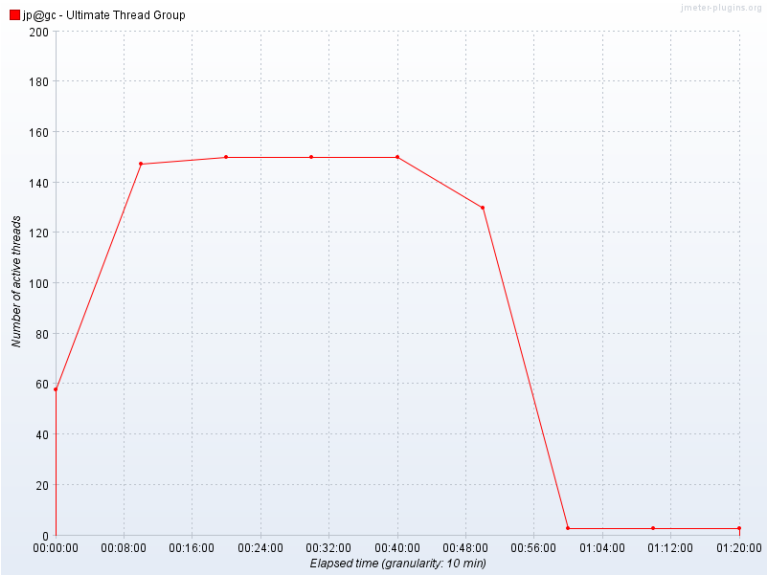


Figura 35: Gráfico com plano de teste do número de utilizadores definido no *JMeter* para simulação do dimensionamento vertical

Definiu-se 150 como valor máximo de utilizadores a acederem à aplicação, que ocorre entre o minuto 10 e 50 do plano de execução.

A partir do minuto 50 até ao último minuto do teste, há uma redução do número de utilizadores de 150 para apenas 2 utilizadores para analisar o comportamento da aplicação perante uma redução da carga repentina sobre o website e respetivas máquinas virtuais.

A Figura 75 e 76 mostram os valores de monitorização, mais concretamente a percentagem de CPU recolhidas do servidor de monitorização ao longo da duração do teste relativas às duas máquinas virtuais.

Com base no plano de teste elaborado, o *JMeter* devolveu como resposta informações como por exemplo o gráfico tempo de resposta em função dos pedidos, que pode ser consultado na Figura 36.

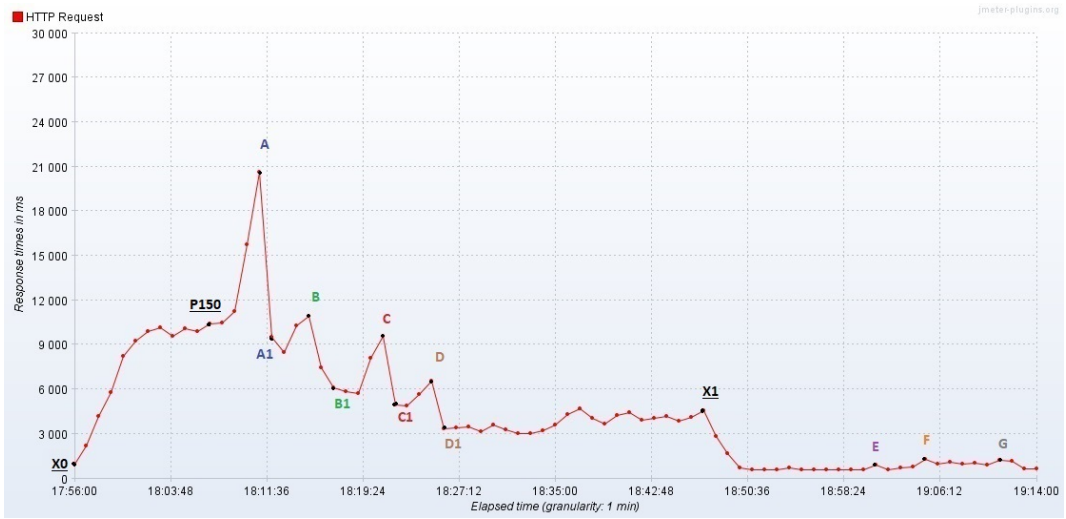


Figura 36: Gráfico com informação do tempo de resposta de pedidos da aplicação em função do tempo para a simulação 1

No gráfico da Figura 36 estão identificados os momentos chave (com os pontos A até G) relativos à simulação realizada. Essas letras do gráfico de tempo de resposta foram também assinaladas nos gráficos de monitorização, com o objetivo de relacionar o tempo de resposta obtido com a percentagem de CPU utilizado em cada máquina virtual.

Resultados

O instante designado por **Xo** da Figura 36 indica o início do teste de carga.

Os primeiros 10 minutos do teste de execução corresponde ao período de aquecimento dos clientes e a partir do instante P150 começam a entrar os 150 utilizadores de forma concorrente na aplicação. Nesse exato momento, os utilizadores demoram cerca de 10 segundos a acederem ao endereço definido no teste de carga.

No entanto, no instante A (às 18:11) ocorreu a primeira operação de *Scale Up* na máquina *PoolServer1*.

Como a esta máquina possuía uma percentagem de CPU acima dos 80%, esta sofreu alteração do tamanho passando de Standard_B1ms para Standard_B2ms. Essa acção pode ser consultada no gráfico de monitorização da Figura 37.

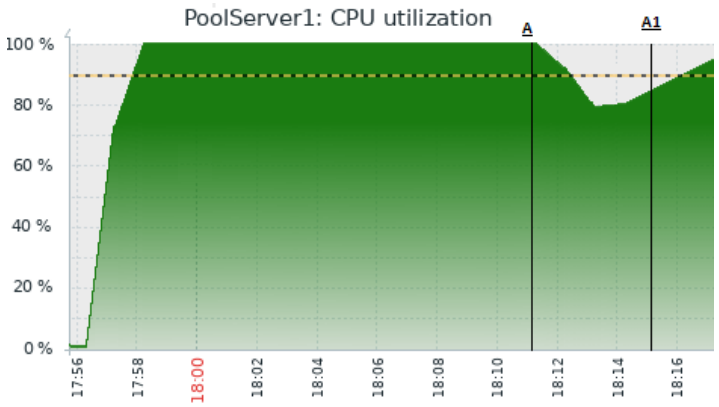


Figura 37: Monitorizaçao de CPU da máquina PoolServer1 desde o início do teste até ao momento ação de *Scale Up* do ponto A

Nesse momento (instante A às 18:11), a máquina PoolServer2 passou a receber a totalidade dos pedidos pois corresponde ao momento de indisponibilidade da máquina *PoolServer1* que teve que reiniciar para atualizar o seu tamanho.

No instante **A** (das 18:11) no gráfico da Figura 36 verifica-se um aumento do tempo de resposta, pois como a máquina *PoolServer1* teve que reiniciar para atualizar o seu tamanho, os pedidos do website foram todos redirecionados para a *PoolServer2*, originando um aumento do tempo de resposta para 21 segundos.

Porém no instante de tempo **A1**, ambas as máquinas já se encontravam a servir o website, mas agora com as seguintes características visíveis na Tabela 6:

Informação das máquinas virtuais (após *Scale Up* do instante 18:11)					
Informação no Azure			Informação no HAProxy		
Nome VM	**Tamanho**	**Nº núcleos**	**WebServer**	**IP privado**	**Peso**
PoolServer1	Standard_B2ms	2	poolserver1	10.1.0.4	67
PoolServer2	Standard_B1ms	1	poolserver2	10.1.0.5	33
Grupo de recursos: DevOps-AutoScaling			Grupo: backend.360imprimir.pt		

Tabela 6: Informação das máquinas virtuais na nuvem e no HAProxy

Uma acção de *Scale Up* demora aproximadamente 4 minutos a ser executada, modificando o tamanho na nuvem e a alterar os pesos no balanceador de carga. Esta acção demorou 4minutos e 18 segundos a ser executada, conforme indica a Figura 38 referente à tarefa do *Hangfire*.

Desde o início do teste, a máquina *PoolServer2* também recebeu carga e de acordo com a Figura 39, como a percentagem média de CPU dos últimos 5 minutos foi superior a 80%, ocorreu uma acção de *Scale Up* no instante B (das 18:15), atualizando o tamanho da máquina *PoolServer2* de *Standard_B1ms* para *Standard_B2ms*.

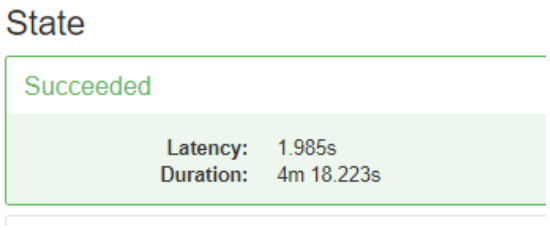


Figura 38: Tempo de execução de uma operação de *Scale Up* extraído do hangfire

Nesse instante **B** enquanto há a atualização do tamanho da *PoolServer2*, a máquina é reiniciada e os pedidos são reencaminhados para a *PoolServer1*, que segundo o gráfico da Figura 36, o website demorava cerca de 10 segundos a responder aos pedidos com uma máquina apenas a servir.

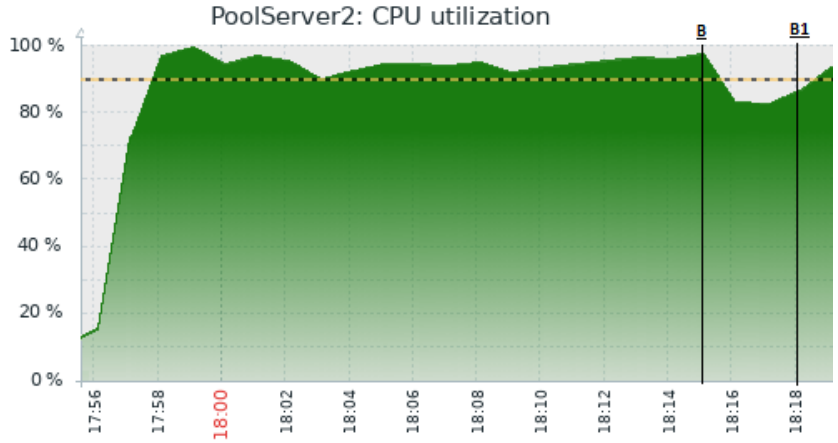


Figura 39: Monitorizaçao de CPU da máquina PoolServer2 desde o início do teste até ao instante B

A Figura 39 ilustra também o ponto **B1** que corresponde ao momento em que as duas máquinas virtuais estão novamente a servir após acção de *Scale Up* da *PoolServer2*, mas agora com as seguintes características visíveis na Tabela 7.

Informação das máquinas virtuais (após *Scale Up* do instante 18:15)					
Informação no Azure			Informação no HAProxy		
Nome VM	**Tamanho**	**Nº núcleos**	**WebServer**	**IP privado**	**Peso**
PoolServer1	Standard_B2ms	2	poolserver1	10.1.0.4	50
PoolServer2	Standard_B2ms	2	poolserver2	10.1.0.5	50
Grupo de recursos: DevOps-AutoScaling			Grupo: backend.360imprimir.pt		

Tabela 7: Informação das máquinas virtuais na nuvem e no HAProxy

Após o instante **B1** da Figura 36, o website passou a demorar cerca de 6 segundos a responder aos pedidos solicitados com as duas máquinas a possuir um tamanho de *Standard_B2ms*.

Como ambas as máquinas estavam com uma percentagem superior a 80%, ambas necessitaram de potenciar os seus recursos para aguentar com a carga recebida, no entanto o algoritmo que controla as operações de dimensionamento teve o cuidado para não despoletar a operação de dimensionamento vertical nas duas máquinas ao mesmo tempo. Isto é ponto de destaque para não deixar as duas máquinas virtuais a servir e não gerar indisponibilidade no website.

Continuando o teste de carga, as máquinas continuam receber pedidos dos utilizadores que acedem ao site e os gráficos da Figura 40 ilustram a percentagem de utilização de CPU das máquinas *PoolServer1* e *PoolServer2*. Conforme a Figura 40, as máquinas virtuais mostram não possuir o tamanho suficiente para suportar a carga recebida, e no instante C (minuto 18:21) ocorreu nova acção de *Scale Up* na máquina *PoolServer1* e no instante D (minuto 18:27) ocorreu outra acção de *Scale Up* na máquina *PoolServer2*.

Após estas estas alterações do ponto **D**, o tempo de resposta do website baixou para aproximadamente 4 segundos, segundo a Figura 36.

Após estas infraestrutura possuía as características visíveis na Figura 8.

Informação das máquinas virtuais (após *Scale Up* do instante 18:25)					
Informação no Azure			Informação no HAProxy		
Nome VM	**Tamanho**	**Nº núcleos**	**WebServer**	**IP privado**	**Peso**
PoolServer1	Standard_B4ms	4	poolserver1	10.1.0.4	50
PoolServer2	Standard_B4ms	4	poolserver2	10.1.0.5	50
Grupo de recursos: DevOps-AutoScaling			Grupo: backend.360imprimir.pt		

Tabela 8: Informação das máquinas virtuais na nuvem e no HAProxy

Após a acção *Scale Up* relativo ao instante **D**, as máquinas virtuais já estavam configuradas com um tamanho estável para aguentar a carga recebida, mais nenhuma operação foi despoletada.

Com isto é possível concluir que os recursos existentes nas máquinas eram suficientes para processar o número de pedidos dos 150 utilizadores. Esta estabilidade verificou-se entre o minuto 18:27 e 18:46 e pode ser consultada na Figura 40.

O ponto X1 (minuto **18:46**), visível na Figura 36 relativa ao tempo resposta da aplicação e na Figura 40 relativa a gráficos de monitorização das duas máquinas virtuais, corresponde ao momento que o número de acessos ao website começa a baixar, contabilizando-se a apenas 2 acessos a partir do minuto **18:50**.

Esta diminuição de acessos resultou numa drástica diminuição da percentagem de utilização de CPU das máquinas virtuais como comprova a Figura 41.

Como se encontrava ativa a regra para existir uma acção de *Scale Down* no caso da percentagem de CPU ser inferior a 15%, com base nos valores recolhidos ocorreu a primeira a acção de *Scale Down* no ponto E (minuto 19:01) sobre a máquina *PoolServer1*, passando do tamanho *Standard_B4ms* para *Standard_B2ms*.

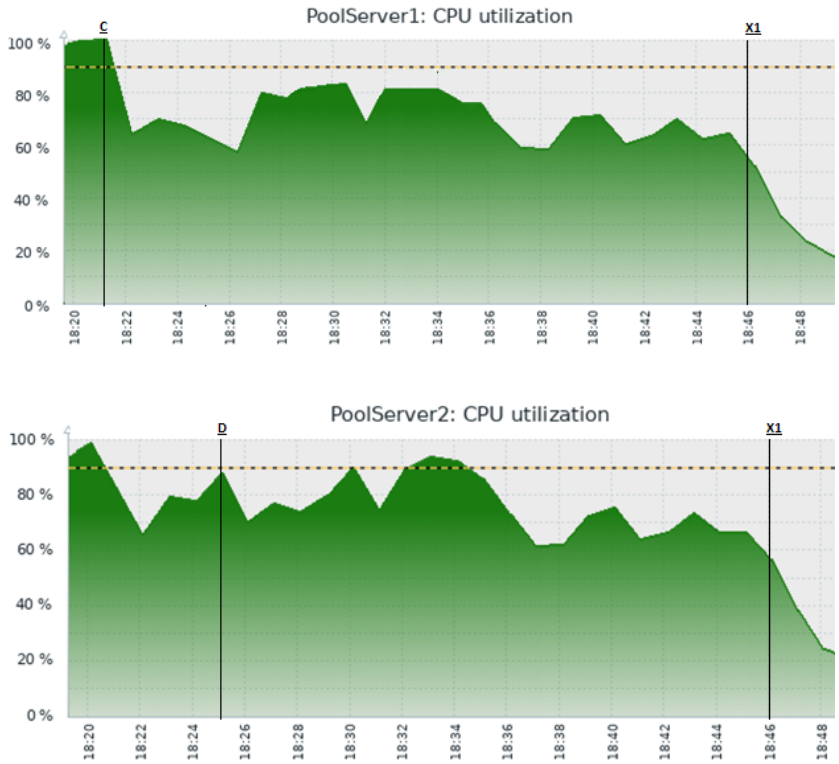


Figura 40: Monitorização das máquinas *PoolServer1* e *PoolServer2* entre o minuto 18:20 e 18:48, com destaque aos pontos C e D

Informação das máquinas virtuais (após *Scale Down* do instante 19:01)					
Informação no Azure			Informação no HAProxy		
Nome VM	**Tamanho**	**Nº núcleos**	**WebServer**	**IP privado**	**Peso**
PoolServer1	Standard_B2ms	2	poolserver1	10.1.0.4	33
PoolServer2	Standard_B4ms	4	poolserver2	10.1.0.5	67
Grupo de recursos: DevOps-AutoScaling			Grupo: backend.360imprimir.pt		

Tabela 9: Informação das máquinas virtuais na nuvem e no HAProxy após *Scale Down* (relativo ao ponto **E** - minuto 19:01)

Como consequência desta redução de acessos ao website, no instante F (minuto 19:05) ocorreu a acção de *Scale Down* na máquina *PoolServer 2*. Esta acção ocorreu porque a taxa de utilização de CPU nos últimos 5 minutos desta máquina era de aproximadamente 10%, obrigando a aplicação a descer ao seu tamanho.

A última operação de dimensionamento para este simulação com a duração de 1hora e 20 ocorreu no instante G com uma acção de *Scale Down* sobre a máquina *PoolServer 2*.

A Tabela 10 mostra a configuração final da infraestrutura após a realização desta simulação1 para teste de dimensionamento vertical.

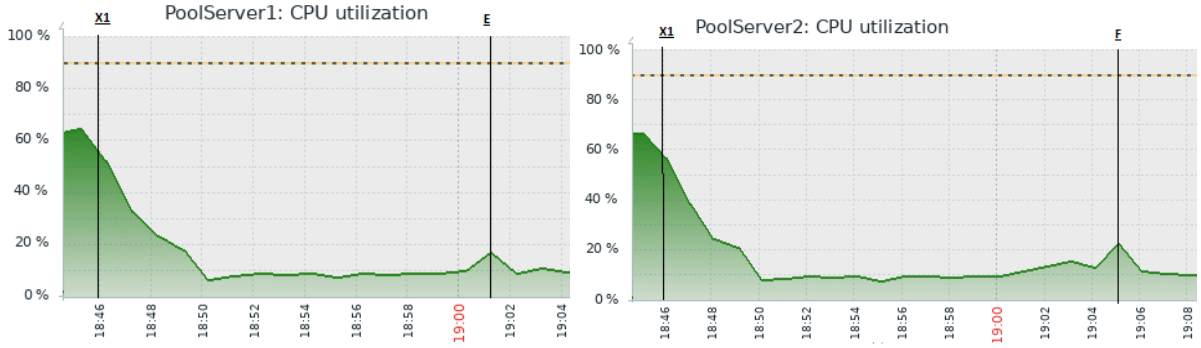


Figura 41: Momento da redução da utilização de CPU nas duas máquinas virtuais (X1) e identificação dos pontos E e F que correspondem a duas ações de *ScaleDown*

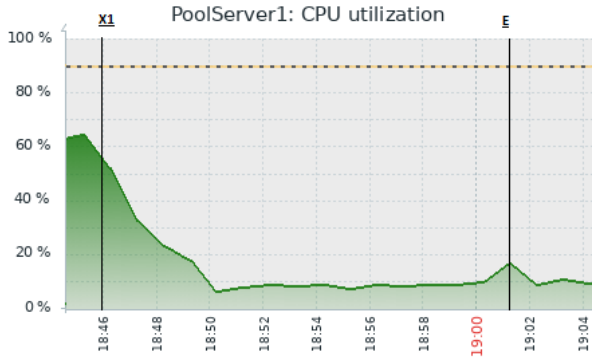


Figura 42: Monitorizaçao de CPU da máquina PoolServer2 desde o início do teste até ao momento da acção de *Scale Up* do instante B

Informação das máquinas virtuais (após *Scale Down* do instante 19:11)					
Informação no Azure			Informação no HAProxy		
Nome VM	**Tamanho**	**Nº núcleos**	**WebServer**	**IP privado**	**Peso**
PoolServer1	Standard_B1ms	1	poolserver1	10.1.0.4	33
PoolServer2	Standard_B1ms	2	poolserver2	10.1.0.5	67
Grupo de recursos: DevOps-AutoScaling			Grupo: backend.360imprimir.pt		

Tabela 10: Informação das máquinas virtuais na nuvem e no HAProxy após *Scale Down* (relativo ao ponto **G** - minuto 19:11)

Análise

Como o número de máquinas se mantém igual durante esta Simulação 1 e o tamanho de cada máquina virtual tem associado um número de núcleos, a forma de avaliar a capacidade de processamento do grupo de recursos faz-se através do número de núcleos total das máquinas virtuais presentes no grupo de recursos da nuvem. A Figura 43 mostra a evolução do número total de núcleos das 2 máquinas virtuais que compõe o grupo de recursos da nuvem designado por *DevOps-AutoScaling* após sofrerem operações de dimensionamento vertical.

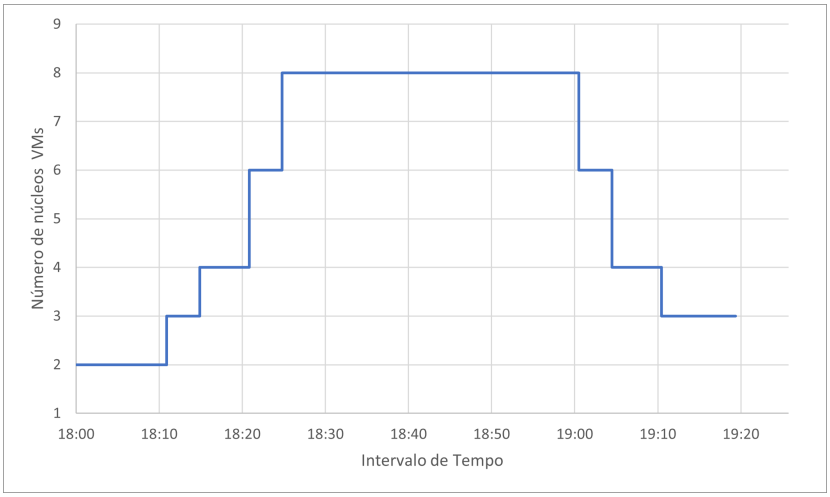


Figura 43: Variação do número de núcleos de máquinas virtuais no grupo *DevOps-AutoScaling* em função do teste de carga

De acordo com o gráfico da Figura 43, a simulação começa com 2 máquinas virtuais configuradas com o tamanho *Standard_B1ms* (1 núcleo cada), isso significa que o grupo de recursos possui 2 núcleos no total.

Até ao minuto 18:25 ocorrem acções de *Scale Up* até existiram 8 núcleos no total do grupo, momento em que ambas as máquinas possuem o tamanho *Standard_B4ms* (4 núcleos cada uma).

Entre o intervalo 18:27 e 18:47 foi encontrado um ponto de estabilidade entre os tamanhos das máquinas virtuais da infraestrutura e o número de acessos por parte dos utilizadores, o que significa que as máquinas tinham capacidade de processamento suficiente para suportar os pedidos recebidos.

No entanto com a redução do número de utilizadores, as máquinas virtuais estavam demasiado fortes para a carga relativa aos pedidos dos dois clientes e não justificavam o seu tamanho excessivo que quanto maior for, mais custos acarreta. Assim, a partir do minuto 19:00, ocorreram 3 acções de *Scale Down* até o término de teste de carga que ocorreu ao minuto 19:14, mostrando a Tabela 10 o estado final da infraestrutura da nuvem e do balanceador de carga após realização da Simulação 1.

5.2.2 *Simulação 2 – Dimensionamento horizontal*

Esta simulação tem como objetivo testar a capacidade da aplicação *IntelliScaling* realizar operações de dimensionamento horizontal, *Scale Out* ou *Scale In* dentro de um grupo de recursos definido na Secção 5.1, perante as variações de carga provocadas pelo número de acessos ao website, simulados por um plano de teste definido no JMeter.

Tendo com base a arquitetura de testes definida na Secção 5.1, o grupo de recursos da nuvem designado por *"DevOps-AutoScaling"* é constituído por 2 máquinas virtuais designadas por *PoolServer1* e *PoolServer2*, configuradas com um tamanho de *Standard_B1ms*.

Acima destas encontra-se uma máquina a servir de balanceador de carga com um grupo com 2 servidores web designados por "*poolserver1*" e "*poolserver2*" com o peso de 50% cada uma.

Regras definidas

À semelhança da Simulação nº1, o primeiro passo da simulação consiste em definir as regras que correspodem aos limites para a aplicação despoletar acções *Scale Out* e *Scale In* quando a métrica despoletar algum dos valores definidos.

A Figura 44 mostra a imagem das regras definidas que permitem realizar acções de forma automática.

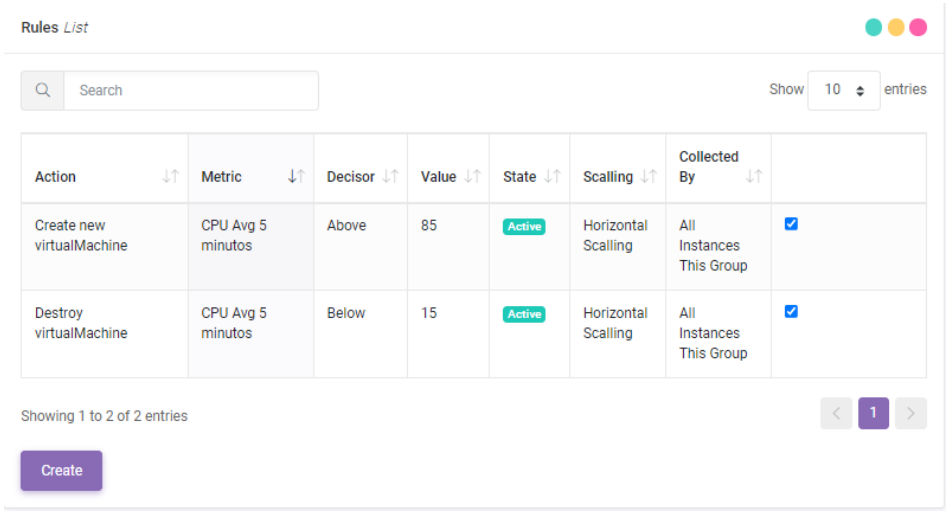


Figura 44: Regras definidas de acções de *Scale Out* e *Scale In*

Regra 1: A primeira regra consiste em criar uma nova máquina virtual assim que a percentagem média de CPU dos últimos 5 minutos de todas as máquinas for superior a 85%.

Regra 2: A outra regra consiste em apagar uma máquina virtual do determinado grupo, quando a percentagem média de CPU durante 5 minutos do total das máquinas virtuais tiver um valor percentual inferior a 15%.

A partir do momento que as regras se encontram ativas, a aplicação já está apta para tomar decisões de dimensionamento horizontal. Portanto, o próximo passo consiste em realizar um teste de carga realizado com auxílio da ferramenta Apache JMeter com o objetivo de saturar as máquinas do cenário inicial referidas na Tabela 5.

Plano de teste

Figura 45: Gráfico com o plano de utilizadores para simulação de dimensionamento horizontal

O gráfico da Figura 45 mostra o plano do teste de carga definido no *JMeter*, que relaciona o número de utilizadores em função do tempo de duração do teste de stress, com o objetivo de induzir carga nas máquinas virtuais durante 50 minutos com um valor máximo de 150 utilizadores.

O teste teve início às 21:12 e durante os 30 minutos iniciais foram introduzidos 150 utilizadores de forma concorrente. Nos últimos 20 minutos, houve uma redução do número de utilizadores de 150 para 2.

Resultados

Com base no teste definida, o *JMeter* o gráfico da Figura 46 que ilustra os tempos de resposta em função dos pedidos executados à aplicação, onde são visíveis espécies de degraus no tempo de resposta que correspondem aos momentos da ocorrência de operações de dimensionamento horizontal (*Scale Out* e *Scale In*) sobre o grupo de máquinas virtuais.

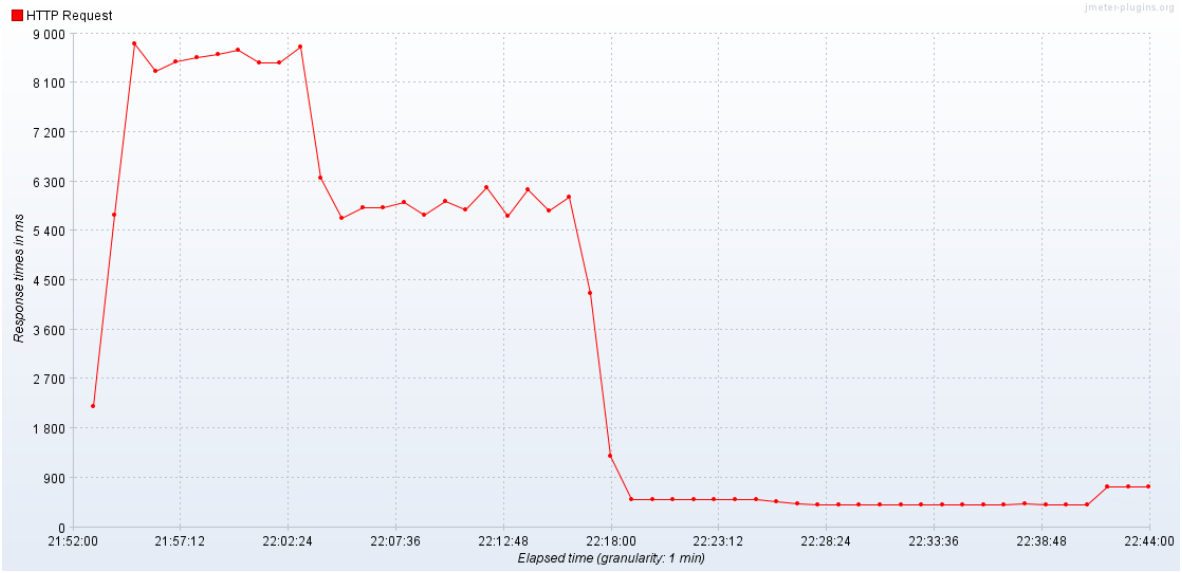


Figura 46: Gráfico com o tempo de resposta em função dos pedidos enviados pelos utilizadores do plano de teste definido

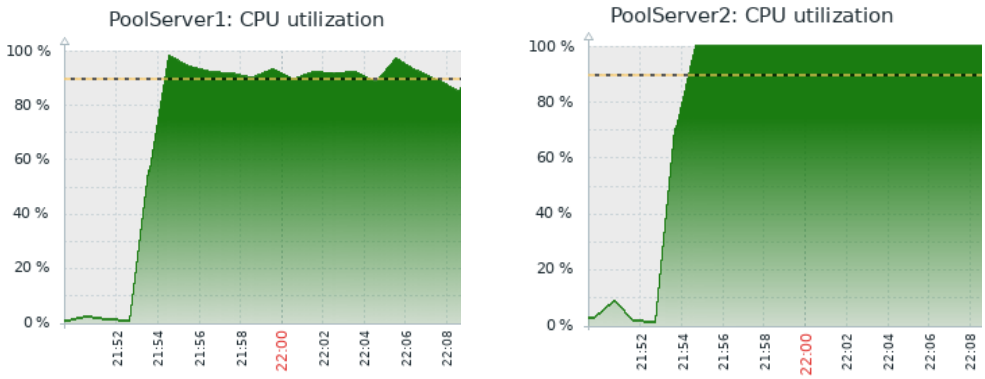


Figura 47: Percentagem de utilização de CPU das duas máquinas desde o início do teste de carga (minuto 21:52) até à primeira operação de *Scale Out* (minuto 22:04)

A Figura 47 mostra a monitorização das duas máquinas virtuais desde o início da realização do teste (minuto 21:52), que a partir desse instante é notório o aumento da taxa de utilização de CPU das máquinas virtuais causado pelo acesso dos vários clientes. Como possuem um poder de processamento bastante fraquinho devido ao seu tamanho, rapidamente começaram a saturar.

Como a percentagem média de CPU nos últimos 5 minutos destas 2 máquinas é superior a 85% (confrontando a **Regra 1** da Figura 44), ao minuto 22:04 ocorre automaticamente a primeira operação de *Scale Out*, que significa que será criada mais uma nova máquina virtual ao grupo de recursos para dar mais poder de processamento.

Em segundo plano, nesse instante a aplicação invoca a *pipeline* do *Azure DevOps*, que através do *Terraform* dá início ao aprovisionamento de uma nova máquina virtual no grupo

de recursos "*DevOps-AutoScaling*", aplicando um tamanho igual às outras duas máquinas virtuais (*Standard_B1ms*).

Além da criação da máquina virtual, uma tarefa na *pipeline* no servidor do *Azure DevOps* efetua alterações no ficheiro de configurações do servidor de balanceador de carga, onde é adicionado um novo servidor web no grupo designado por *"backend.360imprimir.pt"* e atualiza os pesos de todos os servidores web do grupo para 33%.

Informação das máquinas virtuais (após *Scale Out* do instante 22:04)					
Informação no Azure			Informação no HAProxy		
Nome VM	**Tamanho**	**Nº núcleos**	**WebServer**	**IP privado**	**Peso**
PoolServer1	Standard_B1ms	1	poolserver1	10.1.0.4	33
PoolServer2	Standard_B1ms	1	poolserver2	10.1.0.5	33
PoolServer3	Standard_B1ms	1	poolserver3	10.1.0.8	33
Grupo de recursos: DevOps-AutoScaling			Grupo: backend.360imprimir.pt		

Tabela 11: Informação das máquinas virtuais na nuvem e no HAProxy após Scale Out ocorrido no instante 22:04

Durante esta acção de *Scale Out*, foi adicionado um novo *host* com o mesmo nome da máquina virtual no servidor de monitorização, como visível na Figura 48.

Com esta configuração e como na nova máquina virtual está presente um agente do zabbix, esta passa a ser monitorizada e a IntelliScaling consegue ler as suas métricas, como por exemplo, a percentagem de utilização de CPU dos últimos 5 minutos (métrica analisada nesta simulação).

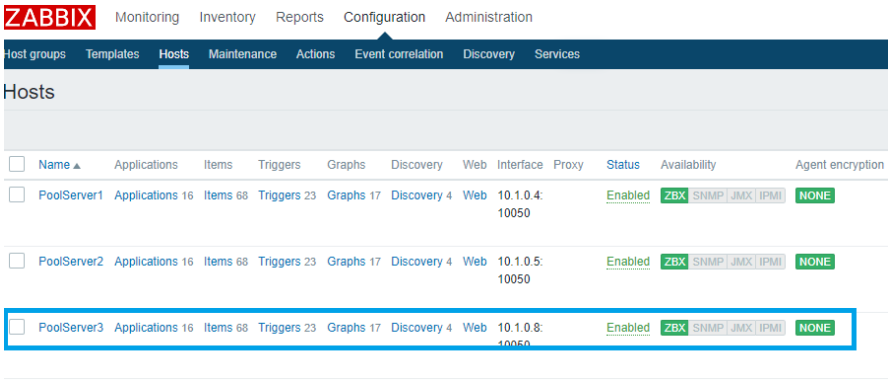


Figura 48: Número de *hosts* ativos no servidor de monitorização após operação de *Scale Out* ocorrida.

Após o registo deste *host* no servidor de monitorização, é possível verificar a percentagem de utilização de CPU desta nova máquina virtual a ser apresentada a partir do minuto 22:10, como ilustra a Figura 49.

Esta acção de *Scale Out* despoletada pelo *Hangfire* demorou aproximadamente 7 minutos e 25 segundos (conforme a Figura 50) a executar todas as configurações na máquina virtual, na nuvem, no servidor de monitorização e no balanceador de carga.

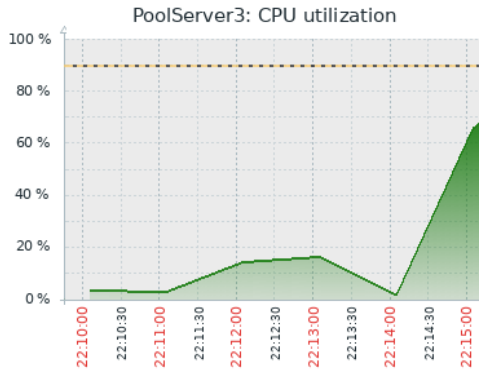


Figura 49: Monitorização imediatamente após a PoolServer3 estar aprovisionada e registada no servidor do Zabbix

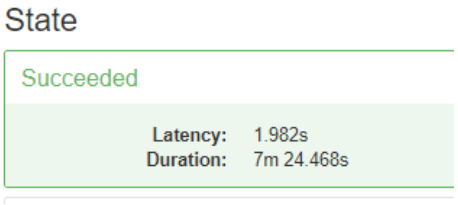


Figura 50: Tempo médio de execução de uma operação de *Scale Out* registado pelo *Hangfire*

Com 3 máquinas em execução, o tempo de resposta ao minuto 22:05 passou para aproximadamente *5 segundos e meio*, como se pode verificar na Figura 46.

A Figura 51 mostra, entre o minuto 22:14 e 22:19, a percentagem de utilização de CPU destas 3 máquinas com o website a correr.

Como a percentagem média de utilização de CPU das 3 máquinas continuava a superar os 85% nesse intervalo de 5 minutos, ocorreu ao minuto 22:19, nova acção de *Scale Out*.

Informação das máquinas virtuais (após *Scale Out* do instante 22:19)					
Informação no Azure			Informação no HAProxy		
Nome VM	**Tamanho**	**Nº núcleos**	**WebServer**	**IP privado**	**Peso**
PoolServer1	Standard_B1ms	1	poolserver1	10.1.0.4	25
PoolServer2	Standard_B1ms	1	poolserver2	10.1.0.5	25
PoolServer3	Standard_B1ms	1	poolserver3	10.1.0.8	25
PoolServer4	Standard_B1ms	1	poolserver4	10.1.0.9	25
Grupo de recursos: DevOps-AutoScaling			Grupo: backend.360imprimir.pt		

Tabela 12: Informação das máquinas virtuais na nuvem e no HAProxy após Scale Out ocorrido no instante 22:19

A Tabela 12 mostra o resultado após a operação de *Scale Out* com o aprovisionamento da máquina *PoolServer4*. No balanceador de carga foi adicionado um novo servidor web com o nome *poolServer4* e além disso foram recalculados os pesos dos servidores web para 25%

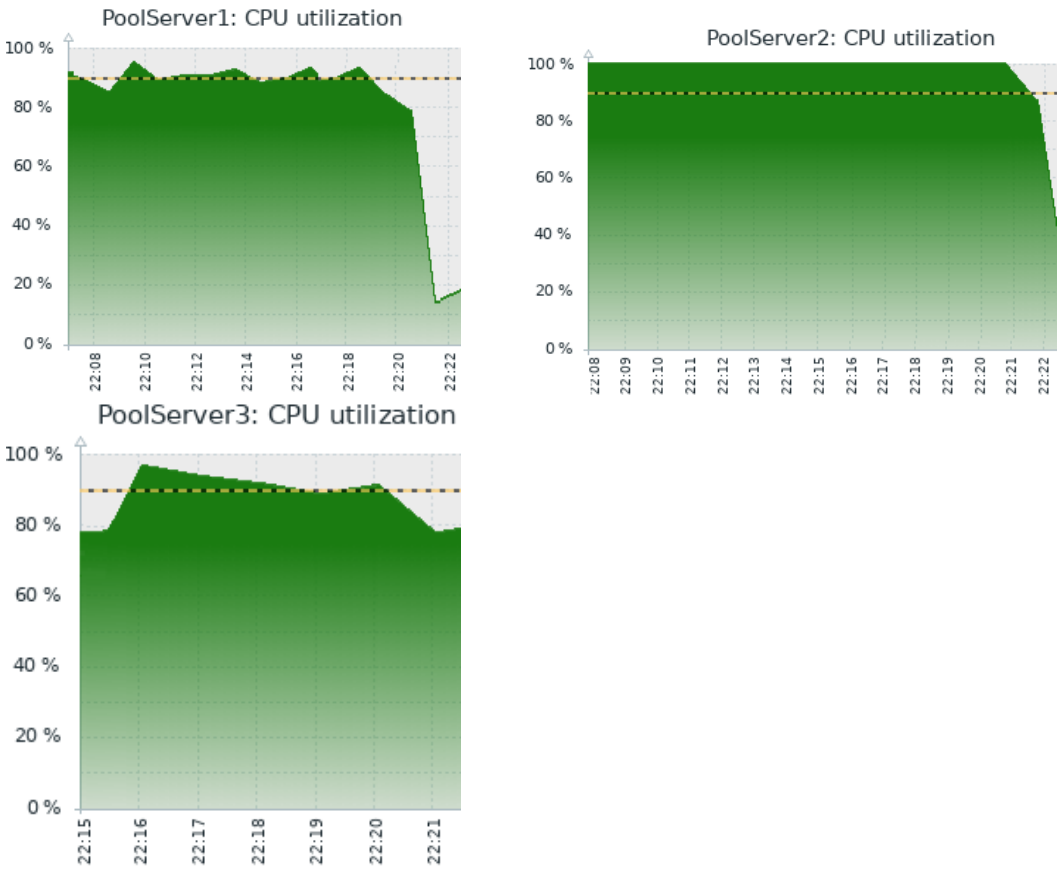


Figura 51: Monitorização das 3 máquinas em execução com destaque entre o minuto 22:15 e o 22:21

com o objetivo destes receberem a mesma percentagem de pedidos por possuírem o mesmo tamanho.

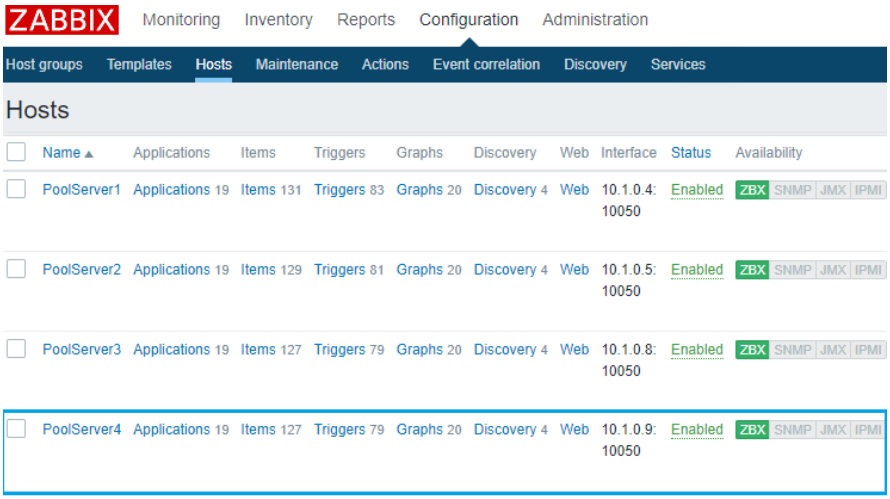


Figura 52: Informação do número de hosts existentes no servidor do Zabbix após segunda ação de *Scale Out*

A partir do momento em que foi adicionado o *host* ao servidor do zabbix, este passou a recolher métricas da nova máquina virtual a partir do minuto 22:26, como se verifica na Figura 53.

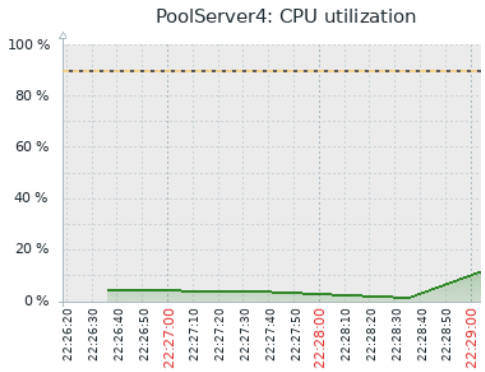


Figura 53: Gráfico da percentagem de utilização de CPU da *PoolServer4*

Continuando o teste de stress, as 4 máquinas continuam a receber pedidos e de acordo com a Figura 46, a partir do minuto 23:20 houve uma redução do número de utilizadores a aceder ao website e como consequência, as taxas de utilização das quatro máquinas virtuais baixaram de forma considerável.

A Figura 54 mostram os gráficos relativos às quatro máquinas virtuais neste intervalo de tempo e como tal, no momento 23:37 poder de computação não justifica a existência de 4 máquinas virtuais a processar os pedidos da aplicação como se pode comprovar nas taxas de monitorização abaixo.

Como a média da percentagem de CPU das quatro máquinas virtuais entre o minuto 23:30 até ao minuto 23:37 situa-se abaixo dos 15% (confrontando a **Regra 2** definida na Figura 44), ocorreu uma operação de *Scale In*, mostrando que não são necessárias tantas máquinas virtuais para processar a carga presente no website.

Informação das máquinas virtuais (após *Scale In* do instante 23:37)					
Informação no Azure			Informação no HAProxy		
Nome VM	**Tamanho**	**Nº núcleos**	**WebServer**	**IP privado**	**Peso**
PoolServer1	Standard_B1ms	1	poolserver1	10.1.0.4	33
PoolServer2	Standard_B1ms	1	poolserver2	10.1.0.5	33
PoolServer3	Standard_B1ms	1	poolserver3	10.1.0.8	33
Grupo de recursos: DevOps-AutoScaling			Grupo: backend.360imprimir.pt		

Tabela 13: Informação das máquinas virtuais na nuvem e no HAProxy após *Scale In* ocorrido no instante 23:37

Esta acção de *Scale In* efetuou as seguintes alterações:

- A máquina *PoolServer 4* foi apagada da nuvem, passando a existir 3 máquinas virtuais no grupo de recursos "*DevOps-AutoScaling*". O servidor web *poolserver4* foi removido

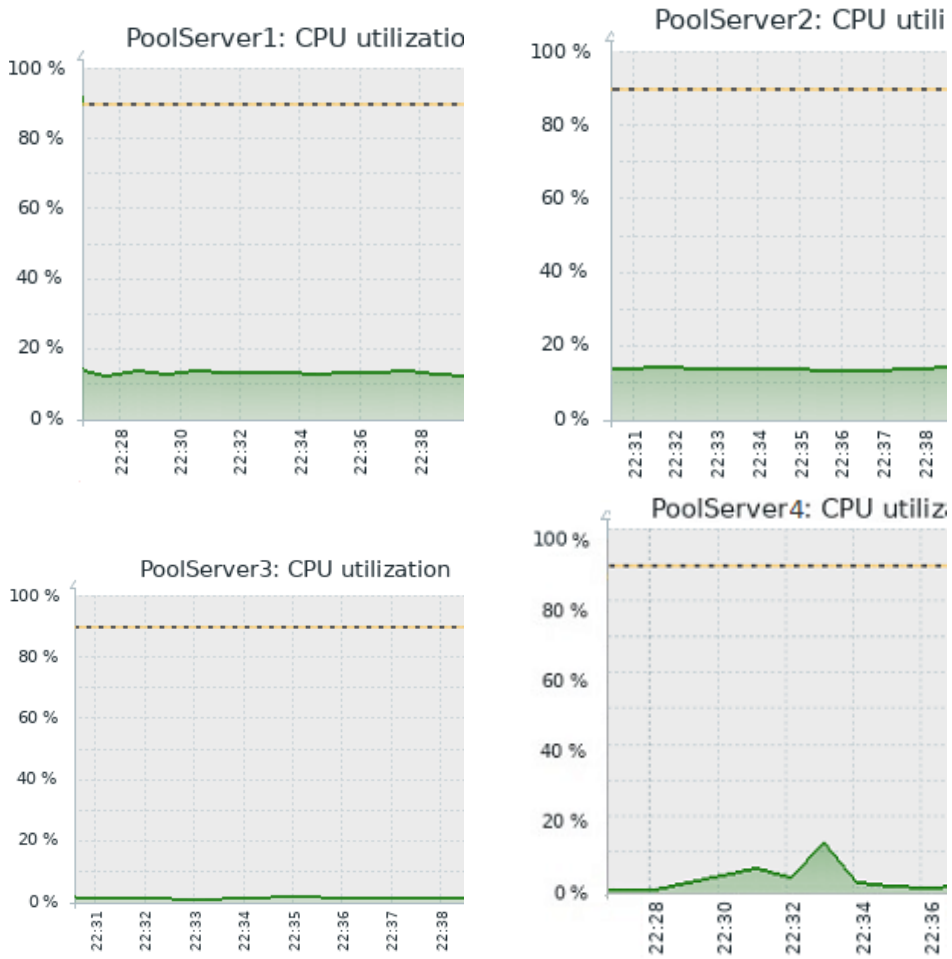


Figura 54: Monitorização das 4 máquinas em execução com destaque entre o minuto 22:30 e o 22:36

do grupo e voltaram a existir 3 servidores web no grupo *backend.360imprimir.pt* do balanceador de carga no estado ativo. O *host PoolServer4* foi apagado do servidor de monitorização, como consequência da remoção da máquina virtual da nuvem.

Análise

O gráfico da Figura 55 mostra a variação do número de de máquinas virtuais que constituem o grupo de recursos durante o período de execução deste teste de carga.

Analisando a Figura 46, os degraus presentes no gráfico do tempo de resposta do website estão relacionados com os eventos de criação ou remoção de máquinas virtuais na nuvem desta simulação 2.

O gráfico 55 mostra de forma resumida os momentos da ocorrência de acções de *Scale Out* (minuto 22:04 e minuto 22:19) e *Scale In* (minuto 23:37) estando relacionadas com as variações analisadas na Figura 46.

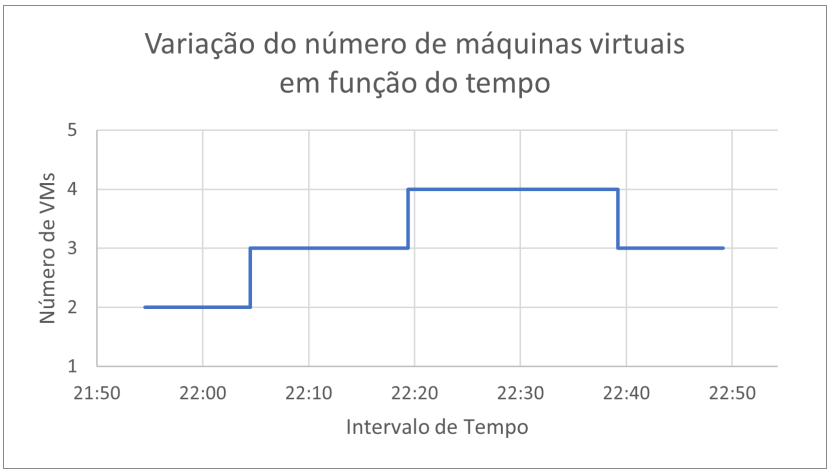


Figura 55: Variação do número de máquinas virtuais durante o teste de carga

Após o término do teste de carga, a infraestrutura de testes ficou constituída por 3 máquinas virtuais com as características indicadas na Tabela 13, após conclusão da Simulação 2 relativa ao dimensionamento horizontal.

6

CONCLUSÃO

A equipa de *DevOps* da *360imprimir* tem em mãos o desafio de gerir vários servidores de uma infraestrutura heterogénea composta por várias instâncias em nuvem e outros serviços externos a elas ligados.

Perante situações de muita ou pouca de carga nas máquinas virtuais, a gestão da sua capacidade é feita de forma manual podendo dar origem a falhas humanas sempre que são necessárias intervenções, além da necessidade regular consulta da monitorização por parte de um analista.

As soluções de dimensionamento automático, onde são encontrados serviços dedicados ao controlo de IaaS, são aplicações de nuvem como o caso do *Azure Autoscale*, que implica uma determinada configuração própria de máquinas virtuais (como por exemplo, uso de *VirtualMachine Scale Set*) e apresentam limitações da disponibilidade de hardware numa determinada região [5].

O resultado desta dissertação consiste numa aplicação prova de conceito designada por *IntelliScaling*, que permite gerir infraestruturas heterogéneas de máquinas virtuais da nuvem *Azure* e de serviços externos como o caso balanceadores de carga do *HAProxy* e servidores de monitorização *Zabbix*, podendo ser alargada a outras ferramentas com as mesmas funções.

Das duas possíveis abordagens (reativa e proativa) para despoletar acções de dimensionamento automático, desenvolveu-se a abordagem reativa. Através da integração de um motor de decisão de decisão, a aplicação consegue executar operações de dimensionamento vertical e horizontal de forma automática, tendo-se descartado a abordagem proativa por não oferecer muitas garantias de fiabilidade para um contexto empresarial.

Como caso de estudo, a aplicação foi ligada a uma infraestrutura de testes com várias máquinas virtuais e estas foram submetidas a testes de carga com o objetivo de simular o comportamento do sistema num dia em que existe uma grande afluência de utilizadores, como o caso de *Black Friday*. Com base nas variações de carga registados, a aplicação conseguiu gerir operações de dimensionamento automático vertical, horizontal e também efetuar as alterações nos serviços externos como o caso do balanceador de carga e no servidor de monitorização.

Como trabalho futuro, como a aplicação escala com facilidade, podem ser integradas mais bibliotecas para gestão de recursos de outros CSP, como por exemplo, AWS ou GCP ou então mais ferramentas de monitorização ou de balanceamento de carga.

Além disso, pode ser integrado um sistema de alertas (por SMS, email) que notifique os utilizadores da aplicação sempre ocorra alguma operação de dimensionamento ou mudança de estado nos recursos da infraestrutura de nuvem.

BIBLIOGRAFIA

[1] An Introduction to HAProxy and Load Balancing Concepts. https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts. Consultado em 2/12/2019.

[2] Automating Deployments from Azure Repos with Octopus Deploy. https://azuredevopslabs.com/labs/vstsextend/octopus/. Consultado em 13/12/2019.

[3] Autoscaling. https://docs.microsoft.com/en-us/azure/architecture/best-practices/auto-scaling. Consultado em 10/11/2019.

[4] Availability sets on azure. https://www.softnas.com/docs/softnas/v3/html/azure_availability_sets.html. Consultado em 20/8/2020.

[5] Azure Autoscaling supports vertical Scaling? https://docs.microsoft.com/en-us/azure/azure-monitor/platform/autoscale-overview. Consultado em 29/09/2020.

[6] Azure DevOps Pipelines. https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops#:~:text=Azure%20Pipelines%20is%20a%20cloud,it%20available%20to%20other%20users.&text=Azure%20Pipelines%20combines%20continuous%20integration,ship%20it%20to%20any%20target.. Consultado em 23/8/2020.

[7] Azure Monitor overview. https://squaredup.com/blog/azure-monitor-part-1-what-is-it-and-how-does-it-work/. Consultado em 19/03/2020.

[8] Azure Pipelines. https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/pipelines-get-started?view=azure-devops. Consultado em 23/8/2020.

[9] Azure portal overview. https://docs.microsoft.com/en-us/azure/azure-portal/azure-portal-overview. Consultado em 29/10/2019.

[10] Azure Provider on Terraform. https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/guides/azure_cli. Consultado em 20/7/2020.

[11] Azure SDK 2.8 for .NET Developers. https://azure.microsoft.com/en-us/downloads/dotnet-sdk-28/. Consultado em 29/10/2019.

[12] Azure SDK for .NET. https://azure.github.io/azure-sdk-for-net/",note="Consultadoem29/10/2019.

[13] Configuration Tools Infraestructure. https://blog.gruntwork.io/why-we-use-terraform-and-not-chef-puppet-ansible-saltstack-or-cloudformation-7989dad2865c. Consultado em 15/01/2020.

[14] Create virtual machine on Azure via portal. https://docs.microsoft.com/en-us/azure/virtual-machines/windows/quick-create-portal. Consultado em 20/10/2019.

[15] Definição VirtualMachine ScaleSet. https://azure.microsoft.com/en-us/services/virtual-machine-scale-sets/. Consultado em 12/04/2020.

[16] Dependency Injection and Inversion of Control. https://alexalvess.medium.com/dependency-injection-and-inversion-of-control-on-net-core-3136fe98b72. Consultado em 3/11/2019.

[17] Group of variables on Azure DevOps. https://docs.microsoft.com/en-us/azure/devops/pipelines/library/variable-groups?view=azure-devops&tabs=yaml. Consultado em 22/8/2020.

[18] Horizontal Pod Autoscaler. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/. Consultado em 8/11/2019.

[19] How create pipelines. https://docs.microsoft.com/en-us/azure/devops/pipelines/create-first-pipeline?view=azure-devops&tabs=java%2Ctfs-2018-2%2Cbrowser. Consultado em 20/8/2020.

[20] How Machine Learning Algorithms Work (they learn a mapping of input to output). https://machinelearningmastery.com/how-machine-learning-algorithms-work/. Consultado em 08/01/2020.

[21] How to use availability sets. https://docs.microsoft.com/en-us/azure/virtual-machines/windows/tutorial-availability-sets. Consultado em 20/8/2020.

[22] Integrate terraform with Gitlab. https://blog.marcelocavalcante.net/criando-uma-pipeline-para-terraform-com-o-gitlab/. Consultado em 21/8/2020.

[23] IoC. https://www.devmedia.com.br/inversao-de-controle-x-injecao-de-dependencia/18763. Consultado em 3/11/2019.

[24] IoC and DI. https://medium.com/@eduardolanfredi/inje%C3%A7%C3%A3o-de-depend%C3%AAncia-ff0372a1672. Consultado em 3/11/2019.

[25] Layer Patterns. https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design#:~:text=of%20Work%20patterns.-,The%20Repository%20pattern,from%20the%20domain%20model%20layer.. Consultado em 7/3/2020.

[26] .NET - Apresentando o padrão DAO - Data Access Object. http://www.macoratti.net/11/10/pp_dao1.htm. Consultado em 17/01/2020.

[27] NSG. https://docs.microsoft.com/pt-br/azure/virtual-network/network-security-groups-overview#:~:text=A%20network%20security%20group%20contains,destination%2C%20port%2C%20and%20protocol.. Consultado em 20/03/2020.

[28] Programming using Tasks. https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/task-asynchronous-programming-model. Consultado em 5/1/2020.

[29] Reinforcement Learning. https://searchenterpriseai.techtarget.com/definition/reinforcement-learning. Consultado em 24/09/2019.

[30] Repository Pattern in .NET Core. https://www.programmingwithwolfgang.com/repository-pattern-net-core/. Consultado em 17/01/2020.

[31] Repositoy Patterns. https://garywoodfine.com/generic-repository-pattern-net-core/#:~:text=The%20Repository%19Pattern%20is%20an,the%20rest%20of%20the%20application.&text=It%20queries%20the%20data%20source,entity%20to%20the%20data%20source.. Consultado em 6/3/2020.

[32] Resource Groups explanation. https://www.otava.com/reference/how-to-use-azure-resource-groups-a-simple-explanation/. Consultado em 20/8/2020.

[33] Software Development Approachs. https://www.open.edu/openlearn/science-maths-technology/approaches-software-development/content-section-1.5.4. Consultado em 10/3/2020.

[34] Storage Acount Information. https://medium.com/faun/azure-storage-account-68c478be485d. Consultado em 18/03/2020.

[35] Storage Acount overview. https://docs.microsoft.com/en-us/azure/storage/common/storage-account-overview. Consultado em 18/03/2020.

[36] Terraform to manage infrastructure deployment. https://docs.microsoft.com/en-us/azure/devops/pipelines/release/automate-terraform?view=azure-devops. Consultado em 29/09/2020.

[37] Understand organization of subscscriptions in azure. https://blog.siliconvalve.com/2018/08/27/understanding-tenants-subscriptions-regions-and-geographies-in-azure/. Consultado em 20/8/2020.

[38] Unit of Work in Repository Pattern . https://www.c-sharpcorner.com/UploadFile/b1df45/unit-of-work-in-repository-pattern/. Consultado em 16/01/2020.

[39] Virtual Machines Azure. https://docs.microsoft.com/pt-br/azure/virtual-machines/linux/overview. Consultado em 13/02/2020.

[40] Virtual Network and Subnets. https://devblogs.microsoft.com/premier-developer/understanding-cidr-notation-when-designing-azure-virtual-networks-and-subnets/. Consultado em 21/8/2020.

[41] Virtual Network FAQ. https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-faq. Consultado em 21/8/2020.

[42] Virtual Network Overview. https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-overview. Consultado em 21/8/2020.

[43] VMSS Provision. https://docs.microsoft.com/pt-br/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-vertical-scale-reprovision. Consultado em 30/5/2020.

[44] What are DTO's. http://www.macoratti.net/19/07/c_dtovopc1.htm. Consultado em 30/11/2019.

[45] What is Azure Active Directory authentication. https://docs.microsoft.com/en-us/azure/active-directory/authentication/overview-authentication. Consultado em 21/07/2020.

[46] What is IAC. https://medium.com/sysadminas/infrastructure-as-code-42537d03e021. Consultado em 14/01/2020.

[47] What Is Load Balancing? https://www.nginx.com/resources/glossary/load-balancing/. Consultado em 30/10/2019.

[48] What is Machine Learning? A definition. https://expertsystem.com/machine-learning-definition/. Consultado em 12/01/2020.

[49] What is subnet in terraform. https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/subnet. Consultado em 21/8/2020.

[50] Zabbix C API. https://github.com/HenriqueCaires/ZabbixApi. Consultado em 30/10/2020.

[51] How To Install and Configure Zabbix to Securely Monitor Remote Servers on Ubuntu 18.04. https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-zabbix-to-securely-monitor-remote-servers-on-ubuntu-18-04. Consultado em 27/10/2019.

[52] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[53] Laura R Moore, Kathryn Bean, and Tariq Ellahi. Transforming reactive auto-scaling into proactive auto-scaling. In *Proceedings of the 3rd International Workshop on Cloud Data and Platforms*, pages 7–12, 2013.

[54] Fabio Jorge Almeida Morais, Francisco Vilar Brasileiro, Raquel Vigolvino Lopes, Ricardo Araújo Santos, Wade Satterfield, and Leandro Rosa. Autoflex: Service agnostic auto-scaling framework for iaas deployment models. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 42–49. IEEE, 2013.

[55] Sérgio Manuel Maia Torres Moreira. Monitorização de redes e sistemas informáticos. 2014.

[56] EG Radhika, G Sudha Sadasivam, and J Fenila Naomi. A rnn-lstm based predictive autoscaling approach on private cloud. 2018.

[57] Eric Rutten, Nicolas Marchand, and Daniel Simon. Feedback control as mape-k loop in autonomic computing. In *Software Engineering for Self-Adaptive Systems III. Assurances*, pages 349–373. Springer, 2017.

[58] The importance of having a good monitoring system. `https://pandorafms.com/blog/why-you-need-a-monitoring-system/`. Consultado em 27/10/2019.

[59] Zabbix vs Nagios comparison. `https://kkovacs.eu/zabbix-vs-nagios/`. Consultado em 28/10/2019.

[60] Fan Zhang, Xuxin Tang, Xiu Li, Samee U. Khan, and Zhijiang Li. Quantifying cloud elasticity with container-based autoscaling. *Future Generation Computer Systems*, 98:672–681, 2019.

A

MANUAL DE UTILIZADOR

Esta secção serve como manual de utilização da aplicação onde são abordadas algumas partes mais técnicas do desenvolvimento da *IntelliScaling*.

A.1 COMPONENTES DA APLICAÇÃO

A aplicação possui algumas funcionalidades tais como:

- gestão de grupos, de cargos, de balanceadores de carga, de servidores de monitorização, de grupos de máquinas virtuais, de regras de decisão e listagem de Logs.
- wizard de importação de recursos para a aplicação
- serviço de autenticação via *Active Directory*

A.1.1 *Autenticação via Azure Active Directory*

O *ActiveDirectory* (Azure AD) é o sistema de autenticação que controla e permite o acesso de utilizadores na aplicação, através das credenciais definidas na organização.

Este mecanismo de autenticação inclui alguns dos seguintes componentes tais como serviço de redefinição de password, autenticação sem password, autenticação via multi-factor, do qual o utilizador é questionado sobre informações adicionais, como por exemplo através de um código de utilizador que este recebeu no telefone [45].

Wizard de Instalação - Importação de grupos

Esta funcionalidade tem como objetivo importar os vários recursos pertencentes a uma subscrição na nuvem, do servidor de monitorização e do balanceador de carga.

Este é constituído por três formulários sequenciais que possibilita ao utilizador importar vários dados dos serviços externos para a camada de persistência da *IntelliScaling*:

- Formulário 1: Seleção da subscrição e do servidor de balanceador (Figura 57)

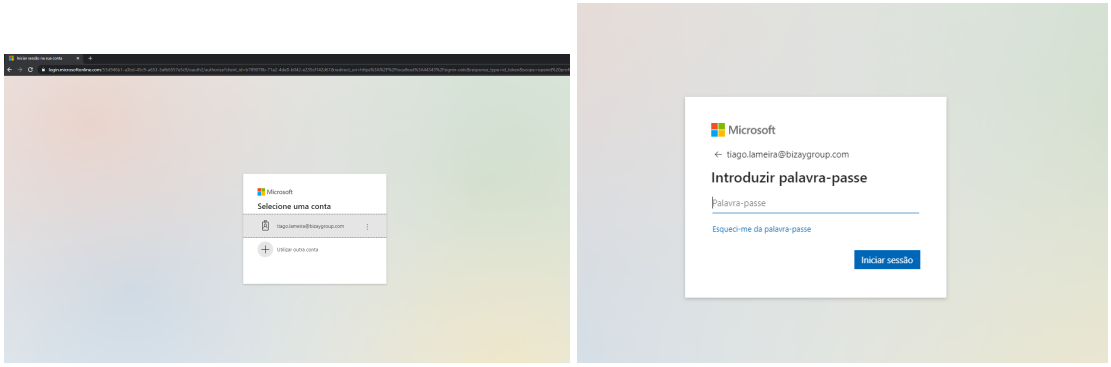


Figura 56: Autenticação via Active Directory

- Formulário 2: Seleção das instâncias com base nos servidores de *FrontEnd* (Figura 59)
- Formulário 3: Seleção do cargo e do servidor de monitorização (Figura 60)

A Figura 57 representa o Formulário 1, exibindo a lista de subscrições que o utilizador autenticado tem acesso na nuvem com a finalidade de importar recursos.

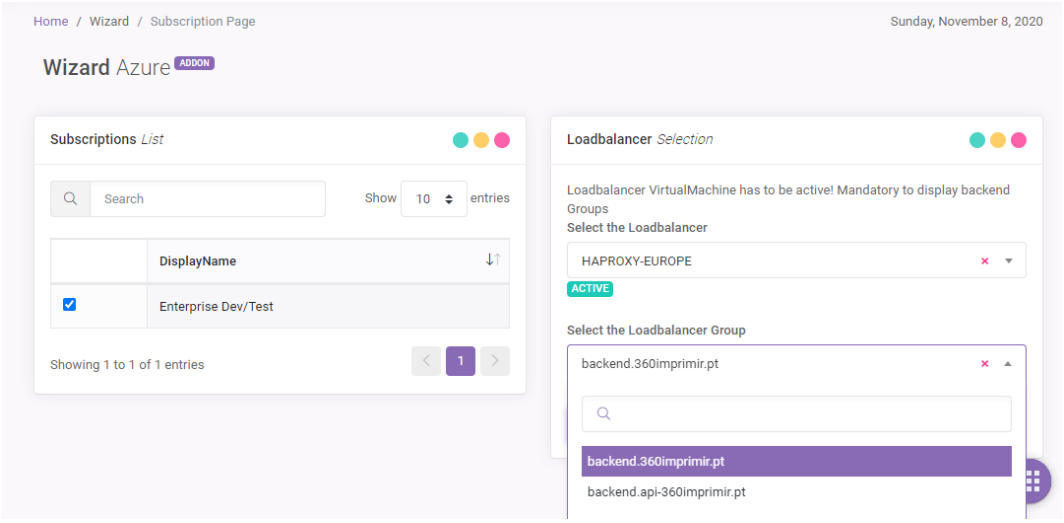


Figura 57: Formulário 1 - Seleção da subscrição e dos backends existentes no balanceador de carga

Esta mostra em formato tabela as subscrições que tem permissão de ser consultadas e alteráveis via API.

A Figura 57 possui duas *Dropdown Lists* para seleção dos vários backends que estão no ficheiro de configuração haproxy.cfg presentes no servidor de balanceador de carga.

Após seleção da primeira *Dropdown List* relativo ao servidor de balanceador, no segundo controlo são carregados os grupos de *backends* que se encontram no ficheiro do balanceador de carga, mas que só aparecem resultados na segunda *Dropdown List* se o servidor estiver no estado *ativo*. Para o utilizador ultrapassar a validação e conseguir aceder ao passo seguinte do Wizard de instalação deve selecionar uma subscrição através da *"checkbox"* associado

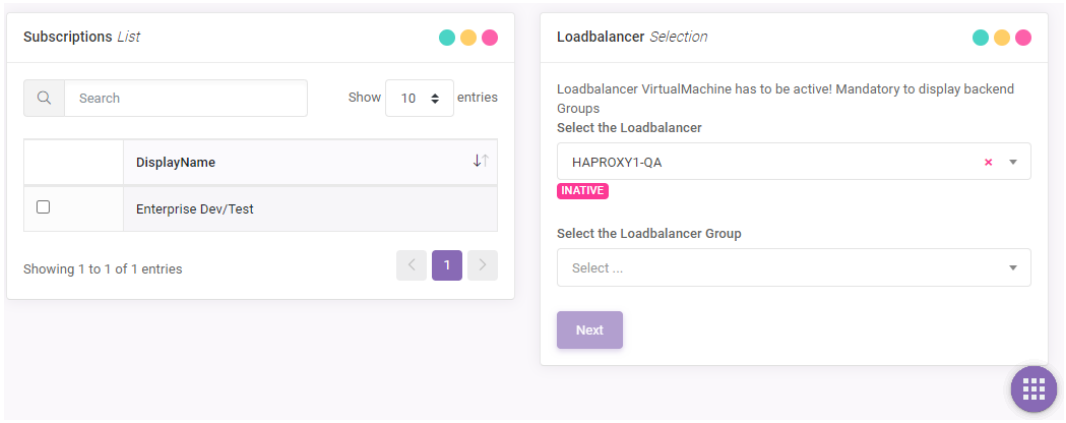


Figura 58: Ilustração do estado do servidor de balanceador de carga no estado inativo

ao registo e tem a obrigatoriedade de selecionar o grupo de*"backend"* que se encontrará na segunda *dropdown*. Caso contrário, o botão "Next"continua bloqueado.

O Figura 59 representa o Formulário 2 que permite associar os servidores web existentes no balanceador de carga às máquinas virtuais existentes numa dada subscrição na nuvem.

De acordo com a Figura 57, o ficheiro haproxy.cfg presente no servidor de carga tem uma secção onde são definidos os servidores web, o ip público ou privado e o respectivo peso que diz respeito à percentagem de pedidos que entram via balanceador de carga.

Todavia esses servidores web e as máquinas virtuais presentes na nuvem são duas entidades completamente independentes um do outro, pois por exemplo, o nome do servidor web pode ser diferente do nome da respectiva máquina virtual da nuvem.

O **IP privado** é o atributo que permite correlacionar máquinas virtuais com os *servidores web*, pois é o único atributo estático comum entre as duas entidades.

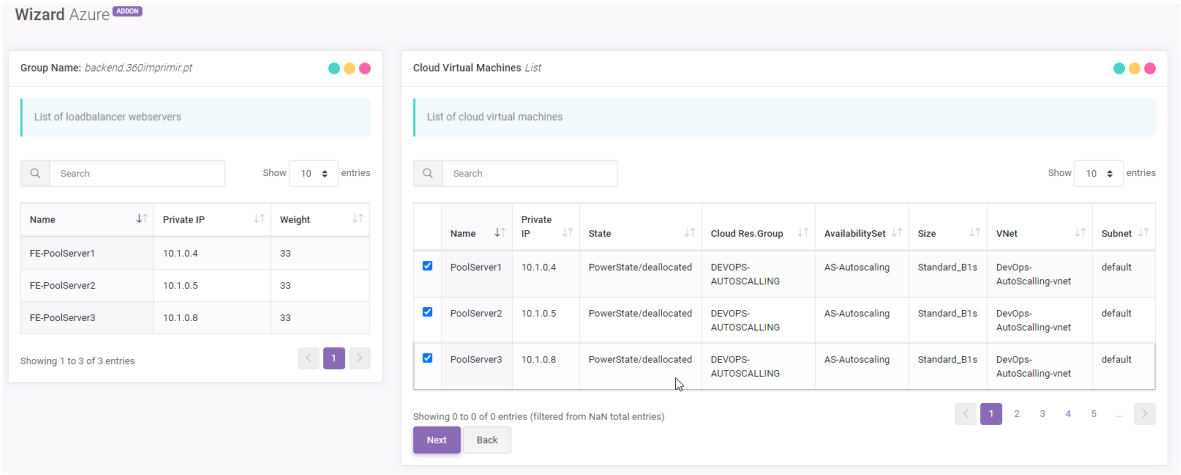


Figura 59: Formulário 2 - Associação dos servidores presentes no grupo de backend às máquinas virtuais de um determinado grupo de recursos da nuvem

Após a seleção da subscrição e do backend do servidor de balanceador de carga e o utilizador pressionar o botão "*next*" presente na figura 57, o sistema mostrará uma interface gráfica como é visível na figura 59.

A Figura 59 é constituída por duas secções.

A secção do lado esquerdo disponibiliza uma listagem de servidores web presentes no ficheiro do *haproxy.cfg* e estes possuem os atributos o nome do webserver, o ip privado e o peso, que também são atributos de interesse para a aplicação de gestão.

A secção do lado direito da figura 59 corresponde às máquinas virtuais relacionadas com os três servidores web que apareceram de forma automática.

Deste processo são retirados os seguintes atributos: Nome da instância, IP privado, estado, nome do grupo de recursos, conjunto de disponibilidade, nome da rede virtual, nome da sub-rede interna do grupo de recursos, IP público, tamanho, sistema operativo e número de núcleos das máquinas virtuais.

A Figura 60 ilustra o Formulário 3 e final do processo de importação de grupos no qual é possível selecionar o cargo e o servidor de monitorização.

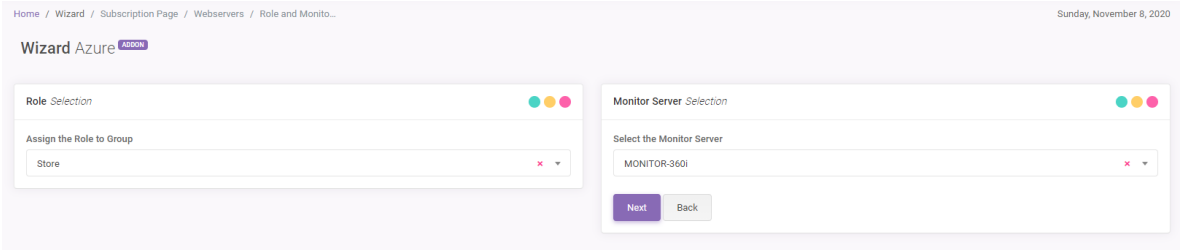


Figura 60: Figura 3 - Seleção do cargo e do servidor de monitorização

Após finalização, na base de dados é criado na entidade *grupo* o registo com exatamente o mesmo valor do grupo de *"backend"* definido na **fase 1** e são importados para este grupo todos os recursos acima enumerados.

A.1.2 *Gestão de Cargos*

A entidade Cargos consiste num agrupamento lógico de websites implantados no IIS nas máquinas virtuais e uma máquina virtual pode ter alojado vários websites em simultâneo, como por exemplo, os website da *Store*, *API* e o *Mailer*. Este grupo permite uma abstracção da aplicação final em relação à máquina virtual, o que significa que um dado grupo de máquinas virtuais pode pertencer a vários Cargos em simultâneo.

A nível de interface gráfica a funcionalidade de gestão de cargos foi planeada da seguinte forma:

- Criar Cargo

Esta funcionalidade pode ser acedida quando se pressiona o botão *New Role* e logo de seguida será mostrado o seguinte formulário. Nesse formulário existe o campo para adicionar o nome do e descrição do cargo. Como ilustra a seguinte figura.

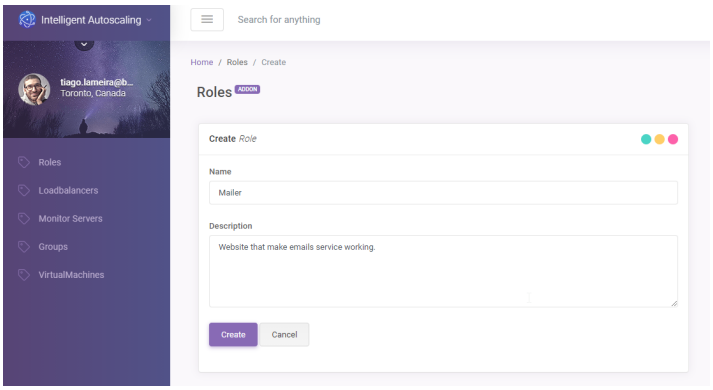


Figura 61: Formulário de criação de um cargo

- Lista de Cargos

  Janela com o propósito de listar todas as roles existentes na base de dados. Em cada linha da tabela é possível verificar o botão de apagar o cargo.

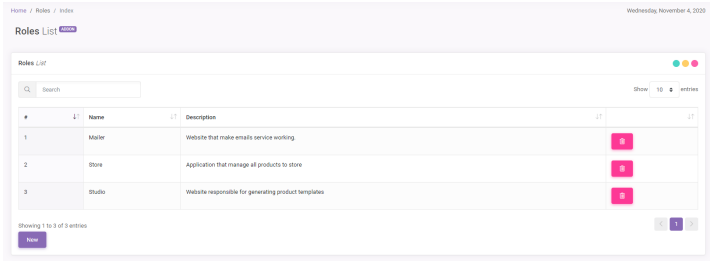


Figura 62: Tabela de listagem de cargos

- Apagar Cargo

  Esta funcionalidade permite apagar o Cargo.

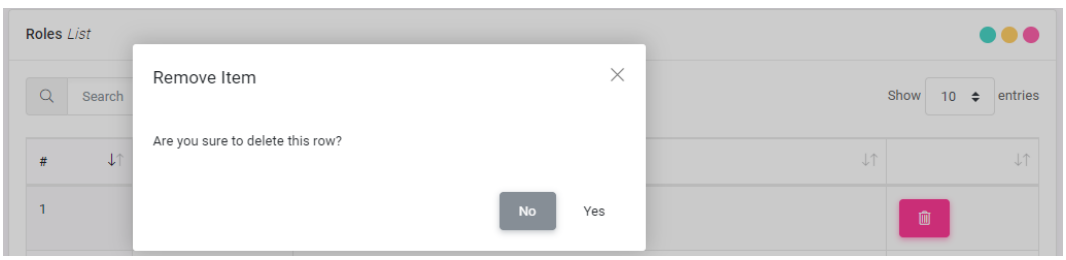


Figura 63: Modal de confirmação para apagar registo

Todavia, se o cargo estiver associado a algum *Grupo*, é enviado uma mensagem ao utilizador, a informar que o registo está associado a uma determinada entidade e a operação de remoção será cancelada.

A.1.3 *Gestão de Balanceadores de Carga*

- Listagem de balanceadores de carga

  Nesta tabela são apresentados todos os balanceadores de carga que foram importados para a base de dados. Na tabela aparece como cabeçalho o nome e o estado atual do balanceador de carga.

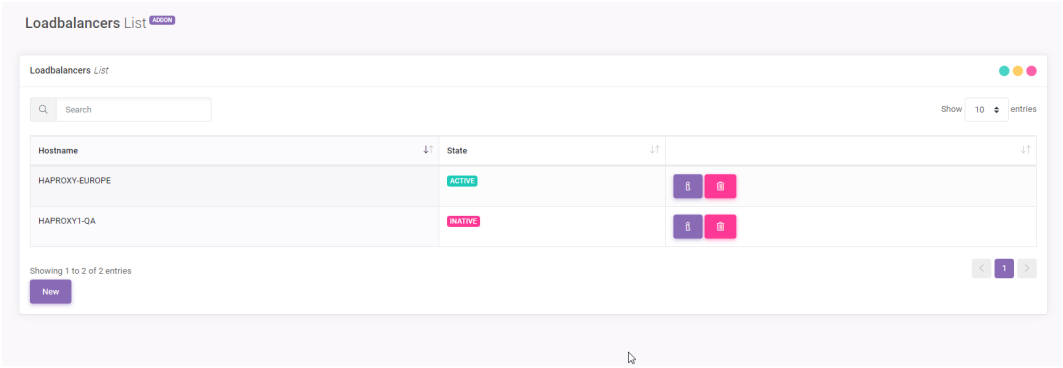


Figura 64: Tabela com listagem de balanceadores de carga

- Criar balanceador de carga

  Esta funcionalidade serve para inserir alguns atributos com o objetivo de registar um balanceador de carga na base de dados da aplicação:

  - Designação - campo de texto para atribuir o nome do balanceador de carga.
  - Ip Público / DNS - campo que deve ser preenchido com um dado estático, isto é, na eventualidade do Ip público da máquina virtual ser dinámico, este campo deve ser preenchido com o DNS da máquina virtual.
  - Ip privado - utilizado para preencher o Ip privado da máquina virtual. Este atributo é utilizado para operações de dimensionamento vertical e horizontal.
  - Nome Servidor - permite definir um *link* para estabelecer comunicação com o servidor de balanceador de carga e recolher informação do ficheiro haproxy.cfg (mais contretamente stats socket ipv4@IPPrivadoHaproxy:8080).
  - Porta - permite definir o valor da porta para estabelecer comunicação com a máquina virtual. O valor da porta deve estar presente na excepção da firewall da máquina virtual.

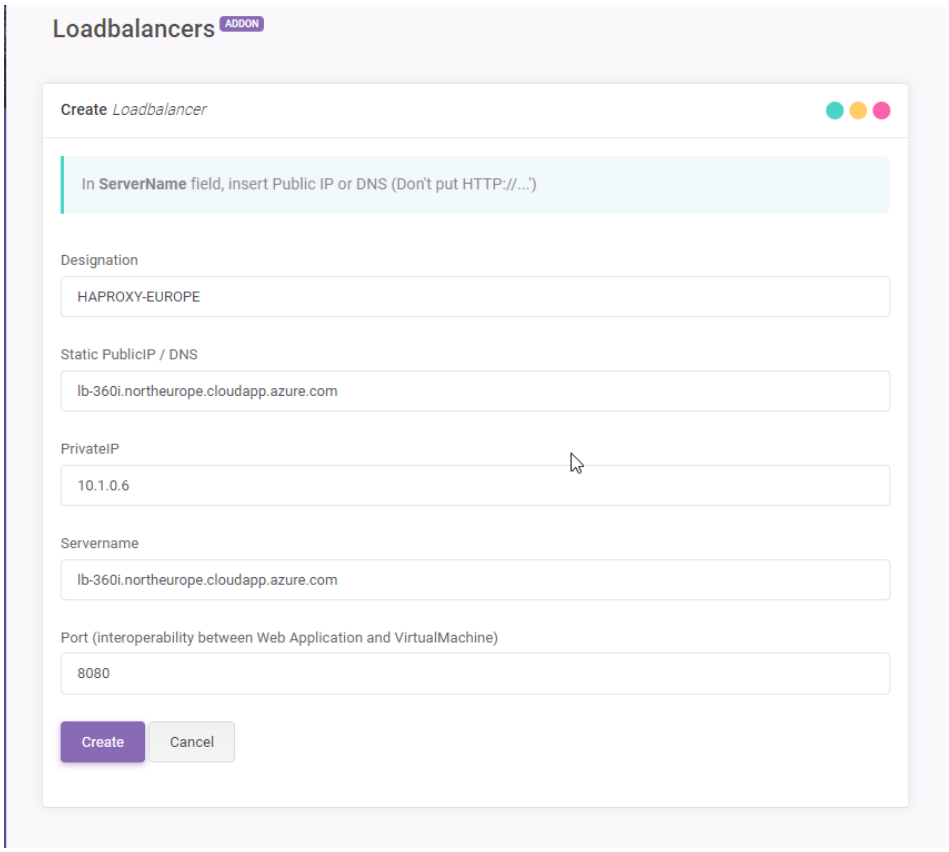


Figura 65: Formulário de registo de máquina virtual de balanceador de carga

- Apagar balanceador de carga

  Esta funcionalidade permite apagar o balanceador de carga.

Biblioteca de camada de acesso a dados

A camada de persistência de dados foi criada a partir da abordagem *Entity Framework* code first, que foca na criação das tabelas e atributos na base de dados a partir das classes.

O uso da abordagem Code First obriga à instalação da EntityFramework que pode ser feito via "Nuget Packages".

O *DbContext* é a classe primária responsável por interagir com os dados, o qual analisa mudanças de estado, persistência dos dados, gestão de relacionamentos e controlo de acessos e pesquisa na base de dados.

A base de dados tem como principal função armazenar dados de interesse e configurações da infra-estrutura e a Figura 66 mostra o modelo desenhado para armazenar os dados de suporte à aplicação IntelliScaling. .

Figura 66: Modelo Entidade-Relação da aplicação

Biblioteca de Modelos

Dentro da solução existe uma biblioteca que possui apenas classes derivadas das entidades designadas por DTO cuja sua função consiste na transferência de dados de entidades entre camadas.

O uso dos DTO's permite manter o acoplamento baixo das bibliotecas da aplicação, através da transferência de dados camada a camada da aplicação.

A transferência de dados classes é feita através do uso de um serviço *automapper* utilizada para mapear objetos entre as camadas da aplicação.

A.1.4 *Gestão de Grupos*

O grupo é a entidade central da aplicação, agrupa e mostra a informação das várias entidades como o caso de máquinas virtuais, servidores web, regras de decisão, cargos e logs.

A Figura 67 mostra as informações associadas a um grupo. Nesta janela, a aplicação dá a possibilidade de despoletar operações manuais de dimensionamento horizontal e vertical.

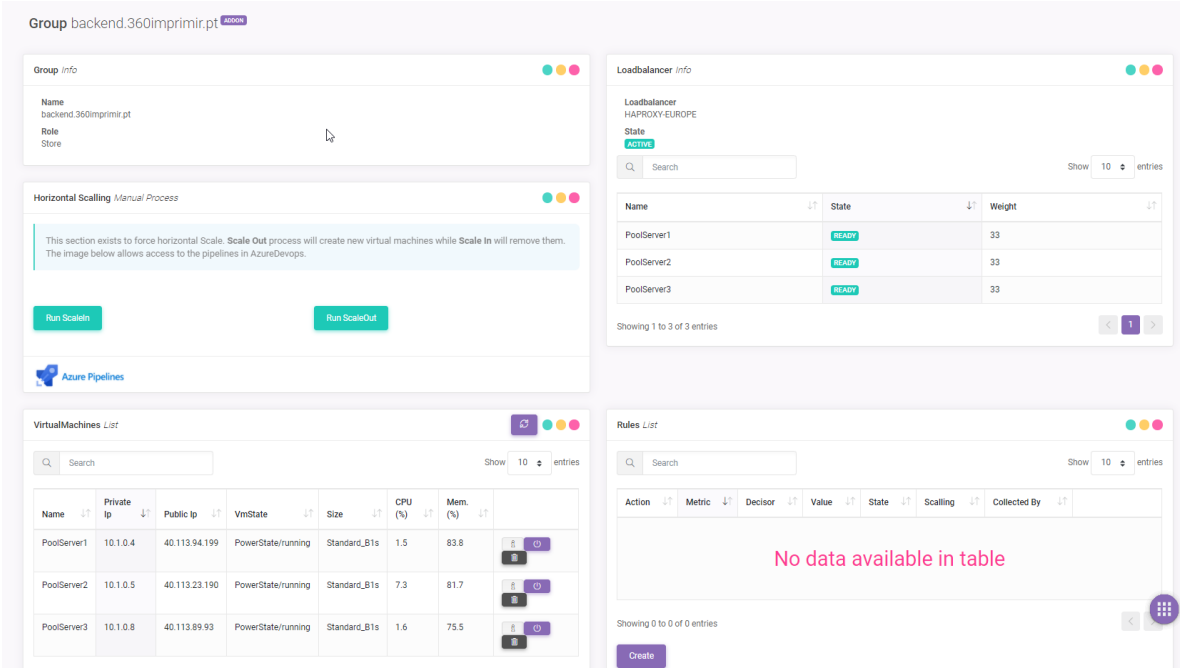


Figura 67: Detalhe do grupo que agrega todas as entidades

A.1.5 *Gestão de Regras de Decisão*

Uma vez criado o grupo e importado os seus constituintes, é possível criar, apagar e configurar regras que serão utilizadas pela aplicação para tomar decisões de forma automática. Este processo de gestão de regras encontra-se na página de detalhe de o grupo como é possível verificar a seguinte figura. A metodologia de criação de máquinas virtuais é completamente parametrizável dando a possibilidade ao utilizador de criar qualquer tipo regra que permita definir acções *Scale Up*, *Scale Down*,*Scale Out* e *Scale In*. Após a regra estar criada o utilizador tem possibilidade de ativá-la ou desativá-la.

- Nova Regra

  Na página de detalhe de um grupo, existe um botão com a designação "New Rule", com o objetivo de mostrar um formulário de criação de uma nova regra.

  Neste formulário serão apresentados os seguintes controlos de seleção: A opção designada por Metric mostra algumas métricas provenientes do sistema de monitorização na qual a aplicação pretende ler. De seguida são enumeradas as seguintes métricas lidas pelo sistema: percentagem média de CPU nos últimos 5 minutos, percentagem

média de CPU nos últimos 15 minutos, percentagem média de CPU nos últimos 30 minutos, percentagem média de memória nos últimos 5 minutos, percentagem média de memória nos últimos 15 minutos, percentagem média de memória nos últimos 30 minutos e número de pedidos do IIS por minuto.

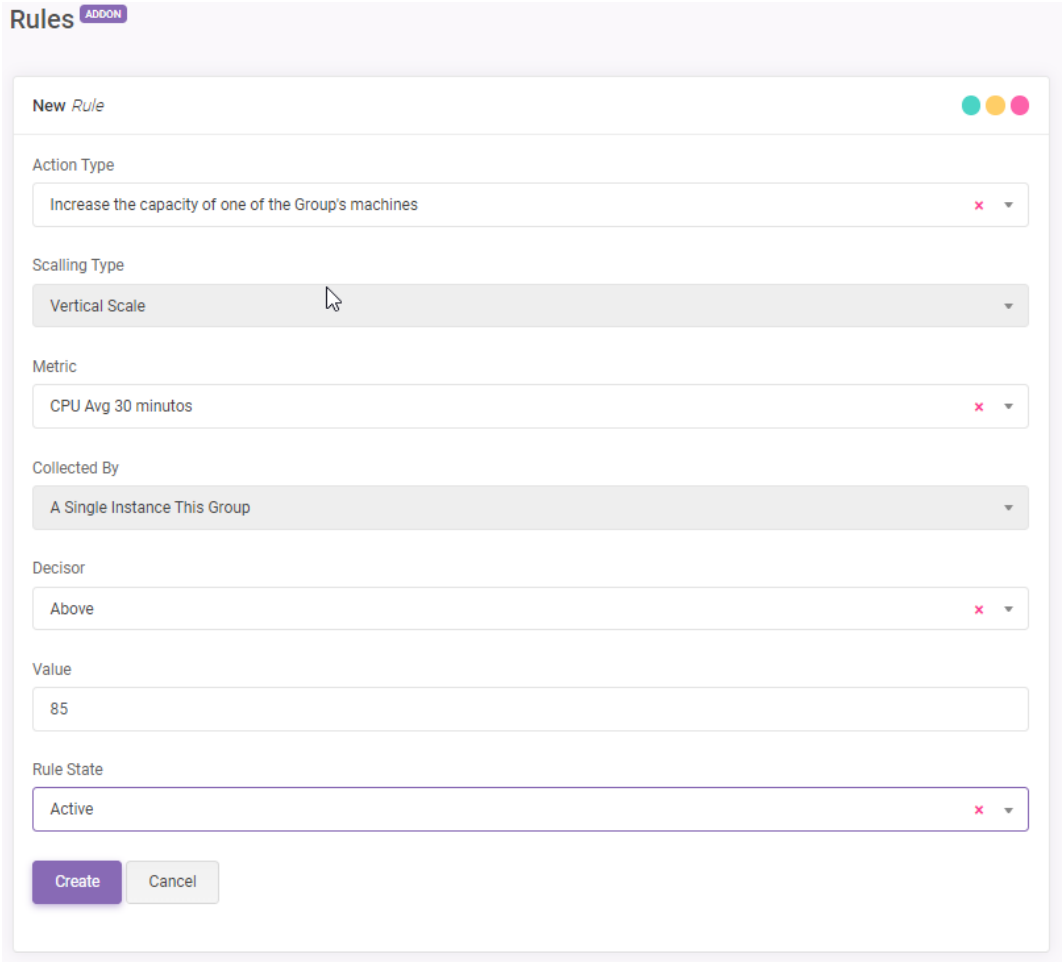


Figura 68: Exemplo de criação de uma regra para acção de *Scale Up*

- Ativar / Desativar Regra

  Esta funcionalidade consiste no controlo do estado da regra, que pode ser ativo ou inativo, através de uma variável definida na base de dados.

- Apagar Regra

  Esta funcionalidade consiste em apagar esta regra que se encontra associada ao grupo. Porém para ser possível apagar a regra, esta tem obrigatoriamente que estar desativada do grupo.

A.1.6 *Gestão de Máquinas virtuais*

Relativamente às máquinas virtuais que foram importadas é possível efetuar algumas interações sobre as mesmas. Sobre cada máquina virtual, o utilizador tem possibilidade de efetuar as seguintes acções manuais:

- Ligar e desligar

  No detalhe da máquina virtual existe um botão que conforme o estado da máquina virtual possui a ação "Ligar / desligar". Uma vez pressionado o botão, a aplicação comunica com a nuvem e dá informação que a máquina virtual vai ser ligada ou desligada. Quando esta acção é despoletada, o sistema foi programado para verificar qual o *webserver* que corresponde à máquina virtual, e ligar ou desligar aquele *webserver* do balanceador de carga.

- Realizar o *Scale Up* manual

  O utilizador tem a capacidade de tornar uma máquina de cada vez mais poderosa. para isso, este tem a possibilidade de realizar o *Scale Up* pressionando o devido botão que se encontra nas acções da tabela de listagem de maquinas virtuais. Perante tal acção, o tamanho da máquina virtual será aumentado para o nível seguinte. Como esta máquina ficará mais poderosa, se estiver na presença de mais mais máquinas virtuais dentro do balanceador de carga, pode receber mais pedidos. Dessa forma, quando o utilizador carrega no botão para realizar o *Scale Up*, é necessário calcular os pesos no balanceador de carga para todos os *webservers* estarem equiparados no contexto número de núcleos da máquina virtual e o seu tamanho.

- Realizar o *Scale Down* manual

  Também existe a funcionalidade para aplicar a acção de *Scale Down* das máquinas virtuais de forma manual. A nível de interface gráfica, existe um botão na listagem de máquinas virtual que perante tal comportamento. Ao contrário do *Scale Up*, quando pressionado o botão referente ao a esta acção, a máquina virtual será alterada para o nível de tamanho abaixo do que estava antes configurada. Tal como descrito no ponto anterior referente ao *Scale Up* manual, foi desenvolvida uma verificação se necessidade de recalcular os pesos do balanceador de carga perante diminuição do tamanho da máquina virtual.

- Consultar detalhe da máquina virtual

O utilizador pode consultar o detalhe da máquina virtual, pressionando o botão cinzento que possui uma *tooltip* [1] com a mensagem *"Detalhe da VM"*. Dentro da página de detalhe são mostradas os valores das métricas de cada máquina virtual.

- Apagar máquina virtual

  Na listagem de máquinas virtuais, existe um botão para apagar a máquina virtual. Ao pressionar esse botão, o sistema mostrará um modal para confirmar se o utilizador realmente pretende apagar a máquina virtual.

A.1.7 *Gestão de Logs*

Dentro do grupo está presente uma tabela com a listagem de Logs que relata todas as acções que ocorreram dentro daquele grupo relativo à criação de grupos, gestão de máquinas virtuais e gestão de regras. Os logs serão apresentados numa tabela na página de detalhe do grupo e podem ser filtrados e ordenados por descrição e data.

A.1.8 *Biblioteca DLL.Zabbix*

A biblioteca de monitorização é uma biblioteca isolada do projeto de gestão que permite gerir tarefas com o servidor de monitorização. Possui um SDK designado *C# ZabbixAPI* responsável por estabelecer tal comunicação com o servidor de monitorização.

Segundo [50], permite recolher e modificar a configurações e realizar operações *Create, Read, Update, Delete (CRUD)* via *REST* API.

Para manter o baixo acoplamento houve algum cuidado em criar métodos privados e para não retornar tipos de variáveis para fora deste SDK, foi criada uma classe que serve como modelo de dados nesta biblioteca onde são definidos os atributos necessários a serem povoados.

Com este procedimento, ao invés de carregar todos os atributos relativos às métricas existentes, apenas são carregados para memória atributos filtrados por tipo provenientes do servidor de monitorização, implicando uma melhoria em questões de desempenho no acto de consumo de dados da API.

A inicialização do serviço que permite a conexão à API pode ser feita de duas formas:

- Definir uma conexão estática no ficheiro de configurações do projeto

```
1 "ZabbixApi": {
2     "url": "http://EnderecopublicoZabbix/api_jsonrpc.php",
3     "user": "nome_utilizador_frontend_zabbixserver",
```

1 A tooltip é um elemento da interface gráfica de utilizador utilizada para mostrar informação de texto quando o utilizador posiciona o rato em cima de um componente *Hypertext Markup Language (HTML)*

```
4     "password": "password_utilizador_frontend_zabbixserver"
5 }
```

Listagem A.1: Exemplo da inicialização estática do serviço *C# ZabbixAPI* via appsettings.json

Esta forma possibilita o uso do princípio de injeção de dependências, pois a conexão é sempre estática e pode ser feita através de um contexto de construtor vazio. Desta forma evita-se a necessidade de instanciar a classe a ser utilizada, como é ilustrado na Listagem A.2.

Porém esta abordagem obriga à alteração do ficheiro de configuração da aplicação sempre que haja necessidade de estabelecer com um servidor de monitorização diferente.

```
1 using(Context context = new Context()){
2   Host host = context.Hosts.Get(new{hostid = "1"});
3 }
```

Listagem A.2: Inicialização do contexto estático da API C# ZabbixAPI

- Instanciar o contexto e o serviço por pedido

  Neste caso o contexto é inicializado com as credenciais de acesso ao servidor de monitorização.

  Esta abordagem dinâmica apesar de menos prática, permite que a aplicação consiga estabelecer conexão com diferentes servidores de Zabbix, podendo assim que seja a lógica definida na aplicação a estabelecer configurações (estratégia utilizada).

```
1 using (Context context = new Context(monitor.Url, monitor.User,
      monitor.Password)){
2     Host host = context.Hosts.Get(new{hostid = "1"});
3 }
```

Listagem A.3: Inicialização do contexto dinâmico

No entanto o SDK não disponibilza todos os métodos necessários para desenvolvimento, como por exemplo, determinar o estado do servidor, pois não possui nenhum método que permita identificar se o servidor se encontra ligado ou desligado.

```
1 public Zabbix(MonitorDTO monitor){
2     try{
3         using (var context = new Context(monitor.Url, monitor.User,
              zabbixModel.Password)){
4             zabbixModel.Groups = context.HostGroups.Get();
5             zabbixModel.State = true;
6         }
```

```
7       }catch (Exception e){
8           zabbixModel.State = false;
9           if (e.HResult == -2147467259){
10              zabbixModel.result = "Server Is Disconnected " +
                    e.InnerException.Message;
11          }
12      }
13  }
```

Listagem A.4: Exemplo do construtor da classe Zabbix que permite obter informação do estado do servidor de monitorização

```
1   public class Zabbix{
2       public Zabbix(MonitorDTO monitor);
3       private List<string> GetGroupsZabbix();
4       private async Task<IEnumerable<Item>> LoadModelCPU(Host host);
5       private async Task<IEnumerable<Item>> LoadModelMemory(Host host);
6       private async Task<IEnumerable<Item>> LoadModelRequests(Host host);
7       private async Task<string> GetCPUByDuration(ZabbixModel model, int
            duration);
8       private async Task<string> GetMemoryByDuration(ZabbixModel model, int
            duration);
9       private async Task<string> GetNumberRequestsByDuration(ZabbixModel model,
            int duration);
10      private async Task<ZabbixModel> GetHistoryDataByHost(string nameVM,
            string operativeSystem);
11      private async Task<Host> GetHostByFilter(string filterName);
12      public async Task<ZabbixModel>
            GetLoadAllModelsMetricsAsync(VirtualMachineDTO VM);
13      public async Task<int> DeleteHostByFilter(string filterName);
14      public async Task<int> CreateHost(string Hostname, string PrivateIp,
            string Port);
15  }
```

Listagem A.5: Exemplo dos principais métodos desenvolvidos de interação com o servidor de monitorização

A.1.9 *Biblioteca de acesso à Cloud*

A aplicação possui uma biblioteca designada por DLL.Azure, com várias classes com a função de estabelecer comunicação com o CSP Microsoft Azure, e está apta para reco-

lher informações e efectuar acções para tornar possível um sistema de dimensionamento automático.

A aplicação consegue interagir com a nuvem através dos seguintes pacotes:

- Microsoft.Azure.Management.Compute - fornece bibliotecas atualizadas do gestor de recursos do Azure para realizar acções de implantação de máquinas virtuais, acções sobre *conjuntos de disponibilidade* de máquinas virtuais, entre outro tipo de acções diretamente sobre estas tais como iniciar, reiniciar, desligar ou escalar. A nível da hierarquia da subscrição do azure, esta biblioteca atua sobre as máquinas virtuais que se encontram no nível abaixo do *gestor de recursos do Azure*

- Microsoft.WindowsAzure.Management.Libraries - Este pacote fornece um conjunto de bibliotecas de gestão de serviços do Microsoft Azure, oferencendo o poder de automatizar, deploy e testar a infraestrutura com facilidade. Permite devolver informação da subscrição, interagir sobre bases de dados, redes e contas de armazenamento.

Dentro desta biblioteca existem as seguintes classes *CloudResourceGroup* e *CloudVirtualMachine*.

A biblioteca *CloudResourceGroup* possui métodos para extração dos vários recursos da subscrição e dos vários grupos de recursos que estão presentes para conseguir comunicar e inicializar com uma subscrição foi adicionado a um ficheiro de configuração várias credenciais relativas a esta. De forma a ser a possível alterar de subscrição, esses ficheiros de configuração encontram-se numa pasta específica do projeto da aplicação de gestão para uma melhor gestão de ficheiros de credenciais.

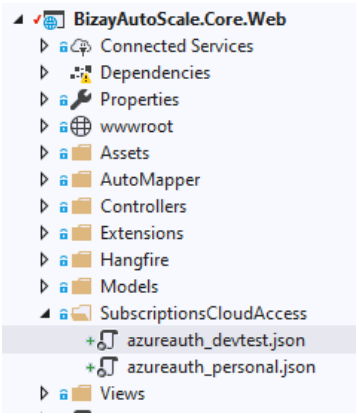


Figura 69: Ficheiros de configuração para comunicação do SDK com a subscrição do Azure

Esse ficheiro contém os dados relativos às credenciais *client_id*, *client_secret*, *tenant_id*, *subscription_id* que permitem identificar a subscrição da nuvem. Esse ficheiro é utilizado para inicializar a subscrição a partir da aplicação.

A imagem seguinte mostra o método de inicialização da subscrição a partir do ficheiro em causa:

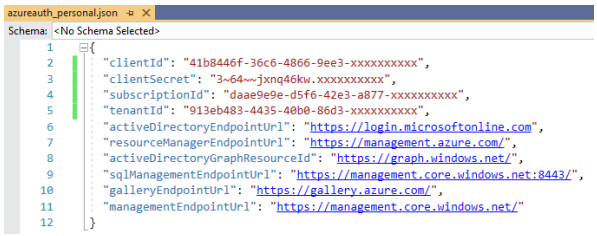


Figura 70: Configurações para o *Azure Credentials*

```
public void InitSubscription(string SubscriptionKey){
    this.azure = Azure.Configure().
    WithLogLevel(HttpLoggingDelegatingHandler.Level.Basic)
  .Authenticate(SdkContext.AzureCredentialsFactory
    .FromFile("SubscriptionsCloudAccess/azureauth_personal.json"))
    .WithSubscription(SubscriptionKey);
}
```

Listagem A.6: Método de inicialização da subscrição

Após a aplicação estabelecer ligação com a nuvem via SDK, já é possível recolher dados através dos seguintes métodos:

```
public interface ICloudResourceGroup{
    void InitSubscription(string SubscriptionKey);
    Task<IEnumerable<CloudSubscription>> ListSubscriptionsAccount();
    Task<CloudSubscription> GetSubscriptionByKey(string SubscriptionKey);
    Task<IEnumerable<GroupDTO>> GetResourceGroupsFromSubscription(string
        SubscriptionKey);
    Task<IEnumerable<CloudVirtualMachine>>
        GetVirtualMachinesByResourceGroup(string ResourceGroupName);
    Task<IEnumerable<IVirtualMachineSize>>
        GetSizesVirtualMachineFromRegion(string VmRegion);
    Task<VNetGroupDTO> GetVirtualNetworksFromResourceGroup(GroupDTO g);
    Task<List<SubnetGroupDTO>> GetSubnetFromResourceGroup(VNetGroupDTO
        VNetGroupDTO);
}
```

Listagem A.7: Interface com os principais métodos de recolha de recursos da subscrição da nuvem

Já a classe *CloudVirtualMachine* tem como objetivo fazer uma transferência dos dados importantes de uma máquina virtual nuvem para uma classe interna do projeto.

Esta classe possui atributos e métodos para tomar acções sobre uma máquina virtual, como por exemplo, ligar, desligar e alterar o seu tamanho.

```
public interface ICloudVirtualMachine{
```

```
2       Task Up_DownScale(VirtualMachineSizeTypes VmSizeType);
3       Task TurnOn();
4       Task TurnOff();
5   }
```

Listagem A.8: Interface com os métodos de interação com uma instância da máquina virtual da nuvem

A classe *CloudVirtualMachine* instancia um objeto da nuvem, que é diferente da entidade *VirtualMachine* do modelo de dados ou da *VirtualMachineDTO*.

Esta possui os métodos necessários provenientes da interface *IVirtualMachine* que consiste numa representação de uma máquina virtual da nuvem. para manter o baixo acoplamento, houve um cuidado para não retornar variáveis do tipo *IVirtualMachine* para fora da biblioteca de acesso à nuvem, caso contrário, seriam criadas dependências de bibliotecas externas nas outras camadas do projeto.

```
1   public CloudVirtualMachine(IVirtualMachine ivm)   {
2       this.Name = ivm.ComputerName;
3       this.PrivateIP = ivm.GetPrimaryNetworkInterface().PrimaryPrivateIP;
4       this.Size = ivm.Size.ToString();
5       [...]
```

Listagem A.9: Exemplo da transferência de registos da *IVirtualMachine* para a classe CloudVirtualMachine

Uma vez estabelecida interoperabilidade com a CSP, a aplicação consegue decisões sobre as máquinas virtuais da nuvem com base no motor de regras de decisão na aplicação de gestão.

A.1.10 *Biblioteca de balanceador de carga*

Na solução é adicionado um novo projeto, à parte do aplicação web para dar a possibilidade de acrescentar novas bibliotecas de balanceamento de carga neste projeto de IntelliScalling.

A biblioteca tem como objetivo de gerir grupos e servidores web presentes no ficheiro de configuração do HAProxy.

No cenário utilizado, o servidor de balanceador de carga consiste numa máquina virtual com sistema operativo Linux que possui uma aplicação designada por HAproxy que oferece alta disponibilidade e balanceamento de carga para aplicações baseados nos protocolos *Transmission Control Protocol (TCP)* e HTTP.

No lado da aplicação para facilitar a interação com o servidor balanceador de carga existe um pacote nesta biblioteca que disponibiliza uma API de controlo do HAProxy. Este pacote tem o nome de *HAProxy Api Client For .Net*.

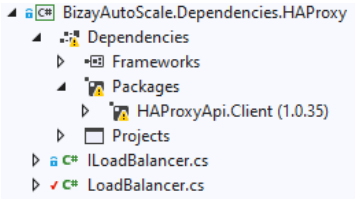


Figura 71: Biblioteca com o nome BizayAutoScale.Depencies.HAProxy com a classe Loadbalancer e package do HAProxyAPI.Client instalado

Nesta bilioteca pretendem-se existem métodos ilustrados na Listagem A.10 que desempenhem as seguintes funções:

- desativar e ativar servidores web
- alterar o peso servidores web
- gerir grupos backend
- adicionar e remover servidores web

```
1 IEnumerable<BackendServerDTO> GetListBackendServers();
2 IEnumerable<BackendServerDTO> GetListBackendServerByName();
3 IEnumerable<WebServerDTO> GetListWebServers();
4 WebServerDTO GetWebServerByIpPrivate(string PrivateIp);
5 IEnumerable<WebServerDTO> GetListWebServersByGroup(string GroupName);
6 void EnableWebServerPool(WebServerDTO webserver);
7 void DisableOutWebServerPool(WebServerDTO webserver);
8 void UpdateWeightWebServer(WebServerDTO webserver);
```

Listagem A.10: Métodos criados para gestão do HAproxy

Desta forma, os métodos que foram ilustrados na Listagem A.10 conseguem devolver valores e efetuar alterações no ficheiro de configuração do balanceador de carga e com isto torna-se possível criar alguma lógica e tomar decisões baseadas em condições.

Limitação do pacote instalado no balanceador de carga

A ferramenta configurada para a comunicação com o balanceador de carga com o nome *HAProxy Api Client For .Net* não permite adicionar ou remover servidores web. Este detalhe origina uma limitação importação importante para acções de *Scale Out* ou *Scale In* no dimensionamento horizontal. A Secção 4.3 aborda de que forma esta limitação foi ultrapassada.

A.1.11 *Biblioteca de gestão de CICD*

A biblioteca de gestão de *CICD* serve para gerir pipelines dos vários projetos de automação de um serviço de *CICD*, como o caso do *Jenkins* ou do *AzureDevops*. O projeto InteliiScaling está configurado para gerir pipelines do serviço *AzureDevops*, por ser o serviço utilizado pela equipa de infraestrutura da 360imprimir para executar tarefas de *CICD*.

Com a configuração realizada no ponto 3.2.4, o Azure DevOps encontra-se apto para servir via *REST API* pedidos relativos ao projeto definido e para esta biblioteca possui alguns métodos que permitem recolher informações das pipelines e também para despoletar acções sobre as mesmas, como exemplifica a Listagem A.11:

```
1 public async Task<IEnumerable<PipelineDTO>> GetPipelinesOrganizationAccount();
2 public async Task<List<PipelineDTO>>
    GetPipelineBySubscription(SubscriptionDTO S);
3 public async Task<int> RunPipelineOrganization(PipelineDTO Pip, dynamic
    DataVMObject);
```

Listagem A.11: Métodos presentes na biblioteca de gestão de CICD

Os métodos existentes na Listagem A.11 dão capacidade da aplicação ler e executar pipelines existentes num projeto no Azure DevOps, com o objetivo de correr comandos Terraform para aprovisionar máquinas virtuais e interagir com componentes externos como caso do servidor de balanceador de carga e de monitorização.

Esses métodos trabalham com objeto de dados do tipo *SubscriptionDTO* que contêm atributos relativos à organização, nome da pipeline, projeto, que permitem identificar a pipeline a ser executada.

O método com o nome *GetPipelineBySubscription* devolve a pipeline da subscrição com o objetivo de ser executada. O método *RunPipelineOrganization* permite executar a pipeline com os atributos relativos às máquinas virtuais a serem fornecidos passados por parâmetro, como é o caso do nome e das máquinas virtuais a serem provisionadas.

Biblioteca de Lógica de Negócio

Por fim, existe a biblioteca de lógica de negócio que tem como objetivo relacionar todas as outras bibliotecas. Esta biblioteca funciona como ponto cental das outras bibliotecas e permitindo a invocação dos métodos presentes das outras bibliotecas da aplicação.

Esta biblioteca tem uma camada de serviços que possui métodos para transferir dados entre a camada de apresentação e as diferentes bibliotecas.

A.2 CONFIGURAÇÕES NO AZURE DEVOPS

Criação da organização e de um projeto interno dentro do portal do Azure DevOps

O Azure Devops neste projecto tem como papel disponibilizar as pipelines para serem fornecidos os comandos de execução do Terraform. Para que seja possível a criação de pipelines para correr os comandos de automação referentes ao *Terraform*, inicialmente é necessária a existência de uma organização da e de um projeto dentro do portal do *Azure DevOps*. A existência desse projeto é que permite a criação numa secção própria de pipelines onde é feito o desplotamento e configuração das pipelines.

Criação do código pessoal de acesso

O código pessoal é uma configuração consiste num código que funciona como alternativa à mecanismo autenticação básico da indicação email e palavra passe do utilizador do AzureDevops, e que pode ser utilizado em componentes terceiros que não suportem contas de *Microsoft* ou de *Azure Active Directory*. No acto de criação do código de acesso é possível especificar qual o tipo de acesso que o este consegue desempenhar, além do tipo de duração (até uma duração máxima de dois anos). A criação do código de acesso pessoal é um requisito exigido quando é feito o uso de comunicações entre serviços terceiros e o Azure DevOps via *REST API*, porque deve inserido na cabeçalho de autorização do pedido para aceder ao método via *REST API*.

Grupos de variáveis

A inclusão dos *grupos de variáveis* às pipelines é uma mais valia pois possibilita o armazenamento de valores que é necessário controlar e torná-las disponíveis pelas diferentes pipelines que constituem uma organização.

É possível utilizar o *grupos de variáveis* para armazenar chaves ou dados sensíveis e outros valores que seja necessário serem passadas para o ficheiro pipeline *YAML*.

Os grupo de variáveis são definidos e geridos através da página Biblioteca dentro da secção Pipelines no Azure DevOps e o uso deste mecanismo evita a exposição de dados sensíveis no template da pipeline, principalmente quando este for enviado para o servidor de controlo de versões [17].

```
1 trigger:
2   - master
3 variables:
4     - group: Devtest_login
5     - name: VMAU
```

```
6       - name: VMAP
```

Listagem A.12: Inicialização das variáveis pelo template da pipeline

O código presente na Listagem A.12 permite associar na pipeline os valores provenientes das variáveis de grupo expressas na Figura 72.

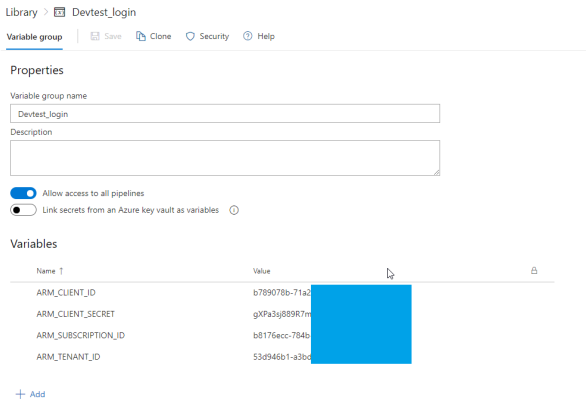


Figura 72: Credenciais definidas num grupo com o nome "Devtest_login"utilizadas no processo de autenticação com a subscrição da nuvem

Criação da pipeline

As pipelines no Azure DevOps podem ser desenvolvidas por meio de templates com recurso à sintaxe YAML ou por meio de configurações através de interface clássica de utilizador (método clássico). O uso de templates torna o processo de configuração mais personalizável e mais livre para o contexto de invocação de, por exemplo, comandos cURL [6, 8].

O projeto possui duas pipelines com o propósito executar acções de dimensionamento horizontal de acordo com a subscrição da nuvem onde estão situados os recursos a configurar.

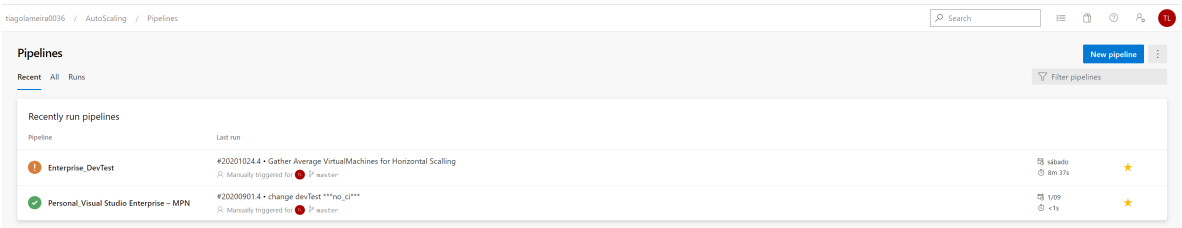


Figura 73: Interface gráfica que ilustra as duas pipelines definidas de acordo com tipo de ambiente da nuvem

A.3 CONFIGURAÇÃO DO AGENDADOR DE TAREFAS

Como *background worker* a ferramenta escolhida é o hangfire. O primeiro passo da configuração consiste na instalação do pacote do hangfire no projeto do visual studio. A forma mais simples de instalação de pacotes *Nugget* consiste no uso da consola gestora de pacotes do Visual Studio para correr os seguintes comandos:

```
1    PM> Install-Package Hangfire.Core
2    PM> Install-Package Hangfire.SqlServer
3    PM> Install-Package Hangfire.AspNet
```

Listagem A.13: Exemplo de comandos para instalar o hangfire via *Package Manager Console*

Com os módulos já instalados na aplicação, o segundo passo consiste na criação uma base de dados vazia. A sua função consiste em persistir tarefas e eventos de segundo plano num armazenamento persistente, para que esses mesmos eventos tenham a capacidade de sobreviver quando a aplicação é reiniciada.

```
1      "AllowedHosts": "*",
2      "ConnectionStrings": {
3            [...]
4            "HangfireContext":
                 "Server=tcp:autoscale360.database.windows.net,1433;Initial
                 Catalog=Hangfire;Persist Security Info=False;User
                 ID=user;Password=user_password;MultipleActiveResultSets=False;"
5      }
```

Listagem A.14: Configuração da *ConnectionString* para a base de dados do Hangfire no ficheiro de configurações appsettings.json da aplicação de gestão

Concluída a configuração anterior, falta apenas instanciar o contexto da base de dados nas secções de configuração e configuração de serviços no ficheiro *"startup.cs"* da aplicação de gestão e feito isto, o hangfire já está pronto para ser utilizado.

```
1  public void ConfigureServices(IServiceCollection services){
2      services.AddHangfire(x => x.UseSqlServerStorage
           (Configuration.GetConnectionString("HangfireContext")));
3      services.AddHangfireServer();
4  }
```

Listagem A.15: Exemplo da secção Configure Services do Startup.cs da aplicação para a configuração do hangfire

```
1  public void Configure(IApplicationBuilder app, IWebHostEnvironment env){
2      app.UseHangfireServer();
```

```
3       app.UseHangfireDashboard("/hangfire");
4       try{
5           RecurringJob.AddOrUpdate<IDecisorService>("GetMetricsDLL", x =>
                x.WorkerGetDataFromZabbix(), Cron.MinuteInterval(2));
6           RecurringJob.AddOrUpdate<IDecisorService>("ApplyDecision", x =>
                x.WorkerTakeDecision(), Cron.MinuteInterval(5));
7       }
8   }
```

Listagem A.16: Exemplo da configuração do hangfire numa secção do Startup.cs da aplicação designada por Configure

A Listagem A.16 mostra o procedimento para a configuração de duas tarefas no *hangfire* responsável por

Uma tarefa tem o nome de *GetMetricsDLL* e invoca uma função designada por *WorkerGetDataFromZabbix()* responsável por recolher métricas do servidor de monitorização.

A outra tarefa designa-se por *ApplyDecision* e invoca a função *WorkerTakeDecision()* responsável por tomar uma decisão de dimensionamento vertical ou horizontal.

O componente do servidor *Hangfire* verifica tarefas recorrentes baseados no intervalo de tempo definidos e depois coloca a tarefas numa fila de dados do tipo despoleta e esquece.

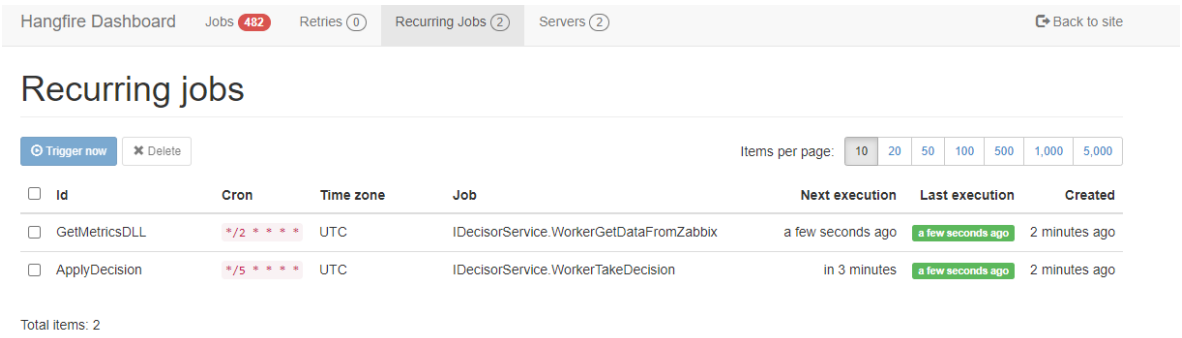


Figura 74: Ilustração das tarefas configuradas no hangfire com a especificação do tempo de recorrência

Algoritmo de recolha de métricas

Este ponto ilustra o algoritmo desenvolvido para recolha de métricas do sistema de monitorização.

```
1   public async Task GetMetrixFromZabbix(){
2       IEnumerable<MonitorDTO> monitorList = await
            this.MonitorMan.GetAllMonitorServersActives();
3       [...]
4       IEnumerable<VirtualMachineDTO> VMSList = await
            this.VMManag.GetVMSByGroupFromDAL(group.GroupId);
```

```
5       Zabbix zabbixCls = new Zabbix(group.MonitorServer);
6       foreach (VirtualMachineDTO Vm in VMSList){
7           [...]
8           Task<VirtualMachineDTO> TaskVM = GetMetricsSingleVM(Vm, zabbixCls);
9           VMsListOfTasks.Add(TaskVM);
10      }
11      VirtualMachineDTO[] VmsArray = await Task.WhenAll(VMsListOfTasks);
12      foreach (VirtualMachineDTO v in VmsArray){
13          await this.VmRepository.UpdateVirtualMachine(v);
14      }
15  }
16  public async Task<VirtualMachineDTO> GetMetricsSingleVM(VirtualMachineDTO v,
        Zabbix z){
17      ZabbixModel z = await m.GetLoadAllModelsMetricsAsync(v);
18      v.kpi_Memory15min = m.GetMemoryAsync(z, 15);
19      v.kpi_CPU30min = m.GetCPUAsync(z, 30);
20      v.kpi_NumberRequests = m.GetRequestsFromVM(v);
21      ...
22  }
```

Listagem A.17: Algoritmo responsável pela transferência de valores do servidor de monitorização para a base de dados

O algoritmo exposto na Listagem A.17 foi dividido em duas partes. O primeiro passo consistiu em recolher os servidores de monitorização que estão activos. A nível de modelo entidade - relação, um servidor de monitorização, possui vários grupos e um grupo possui várias máquinas virtuais. Portanto, foi necessário percorrer as várias colecções desde o servidor de monitorização até chegar à colecção de várias máquinas virtuais que são monitorizadas pelo servidor. Dentro da colecção de máquinas virtuais, foi invocado o método *GetMetricsSingleVM(VirtualMachineDTO v, Zabbix z)* cuja sua função é retornar uma máquina virtual, com os valores das métricas memória, CPU e pedidos dos IIS para os devidos intervalos de tempo.

Recolha de dados para a tomada de decisão

Tal como a função de recolha de métricas, esta encontra-se na camada lógica de negócio do projeto, numa classe designada por *BackgroundWorkerManagement* e foi criada para ser executada recurrentemente pelo hangfire, sendo configurada a tarefa no agendador de tarefas para que seja executada de 10 em 10 minutos.

O motivo pela escolha do intervalo de tempo de execução foi devido à análise efetuada no sistema de monitorização, após uma ação de dimensionamento vertical.

A primeira componente lógica da acção do algoritmo consiste em recolher uma coleção de todas as regras que se encontrem ativas no motor de decisão da aplicação. A partir desse conjunto de regras, foi necessário recolher de forma distinta os valores dos identificadores dos grupos (ID's) que estão associados a essa coleção de regras. Obteve-se uma lista de ID's dos grupos, o objetivo consiste em recolher uma coleção de máquinas virtuais que se encontrem ativas filtradas por identificador do grupo, isto é, cujo o seu estado seja igual a "ligado". Assim neste ponto, o sistema tem acesso a todas as máquinas virtuais que vão ser afetadas pela ordem de tomada de decisão.

```
1 List<VirtualMachineDTO> VmsList = new List<VirtualMachineDTO>();
2 IEnumerable<RuleDTO> ListOfRulesEnabled = await
      this.RuleService.GetAllElementsActive();
3     List<int> ListGroupsId = ListOfRulesEnabled.Select(t =>
          t.GroupId).ToList();
4     foreach (int groupId in ListGroupsId){
5         IEnumerable<VirtualMachineDTO> vmsList = await
              this.VMMan.GetVMSByGroupFromDAL(groupId);
6         VmsList = vmsList.Where(t => t.VmState.Contains("running")).ToList();
7 }
```

Listagem A.18: Recolha de máquinas virtais de grupos com regras ativas

A função *this.VMMan.GetVMSByGroupFromDAL(groupId)* foi criada numa classe designada por *VirtualMachineManagement* com a função de possuir vários métodos que permitem gerir máquinas virtuais na camada de lógica negócio. Este método devolve uma coleção de máquinas virtuais que se encontram na camada de acesso a dados, mas neste método foi feita uma modificação para que não se limite a recolher o conjunto de máquinas vrtuais persistentes em armazenamento.

Ir buscar exclusivamente os dados à base de dados pode não garante a recolha das máquinas virtuais com o estado igual ao estado que possuem na nuvem, isto, as variáveis estado, número de núcleos e tamanho podem possuir valores diferentes daquele que estão a nuvem para lidar com este detalhe, na camada de *lógica de negócio* existe uma classe, com o nome *VirtualMachineManagement*, que que possui métodos para gerir coleções de máquinas virtuais oriundas da biblioteca de *acesso a dados* mas que necessitam de misturar dados provienites da biblioteca *de acesso à cloud*, da biblioteca de *acesso ao servidor de monitorização* e da biblioteca de *balanceador de carga*. Assim para recolher informação do estado da máquina virtual que se encontra na base de dados foi desenvolvido o seguinte método:

```
1 public async Task<IEnumerable<VirtualMachineDTO>> GetVMSByGroupFromDAL(int
      groupId){
2     GroupDTO group = await this.GroupService.GetElementById(groupId);
```

```
3       IEnumerable<VirtualMachineDTO> vmDTOs = await
            this.VmRepository.GetAllByGroup(group);
4       this.CloudResourceGroup.InitSubscription(group.SubscriptionKey);
5       foreach (VirtualMachineDTO v in vmDTOs){
6           CloudVirtualMachine ivm = await
                this.CloudResourceGroup.GetSingleVirtualMachineByKey(v.Uuid)
7           v.VmState = ivm.VmState;
8           v.NumberOfCores = ivm.NumberOfCores;
9           v.Size = ivm.Size;
10      }
11      return vmDTOs;
12  }
```

Listagem A.19: Método da camada lógica de negócio que permite recolher as máquinas virtuais da base de dados com inclusão de detalhes das outras bibliotecas desenvolvidas

A Listagem A.19 foi desenvolvida com o seguinte propósito, recebido valor do identificador do grupo por parâmetro, é possível ter acesso ao objeto grupo, que contém atributos com o caso da chave de subscrição (*subscription key*) para a aceder à nuvem através do método *InitSubscription*.

O método *this.VmRepository.GetAllByGroup(group)* permite devolver todas as máquinas virtuais em base dados que filtradas pelo grupo e é sobre cada uma dessas máquinas virtuais que se pretende buscar atributos da nuvem.

O método *this.CloudResourceGroup.GetSingleVirtualMachineByKey(v.Uuid)* permite devolver um objeto do tipo *CloudVirtualMachine* que corresponde à instância da máquina virtual da nuvem com aquele identificador único universal (uuid). Sobre cada máquina virtual é possível recolher o estado, o número de núcleos e tamanho provenientes do objeto *CloudVirtualMachine*, que assim é possível associar à coleção de máquinas virtuais.

A Listagem A.20 expõe o segundo passo do algoritmo de decisão que diz respeito à recolha de métricas e seus respetivos valores para confronto com o motor de regras de decisão e nesta secção também é feita gestão de concorrência através de um semáforo.

Mas antes disso, a condição *(r.MetricCollectedBy.Equals("AllInstances"))* verifica se cada regra presente na coleção é do tipo agregada ou não-agregada, isto é, se é contabilizado os valores das métricas de todas as máquinas virtuais do grupo para cálculo de média global ou se é recolhido o valor por cada máquina virtual singular.

```
1   try{
2       if (r.MetricCollectedBy.Equals("AllInstances")){
3           if (!groupSingle.In_Progress && vms.Any(v =>
                Generic.GetAverageMetricAllInstances_VariableValue(r.Metric,
                r.Group).Count != 0)){
4               toBlock = true;
```

```
5       }else{
6           if (!groupSingle.In_Progress && vms.Any(v =>
                Generic.GetVariable_ValueFromClass(r.Metric, v).Count != 0)){
7               toBlock = true;
8           }
9       }
10  [...]
```

Listagem A.20: Secção 2 do algoritmo de tomada de decisão - Confronto das métricas das máquinas virtuais e gestão de concorrência

O método *Generic.GetVariable_ValueFromClass(r.Metric, v)* foi criado com o objetivo de recolher todas as métricas e respetivos valores de cada máquina virtual. Este método permite retornar uma estrutura de dados do tipo dicionário, que em cada linha será feita uma atribuição da chave e o seu respetivo valor de todos os atributos que digam respeito a métricas nas máquinas virtual. Assim na eventualidade de serem adicionadas mais atributos que digam respeito a novas métricas, este método tem a capacidade de extrair a métrica e o seu respetivo valor através da técnica de *Pattern Reflection*, explicada na Secção **??**.

A.4 Interação com o balanceador de carga

Através da aplicação a operação de cálculo dos pesos dos servidores web é feita com recurso ao algoritmo da Listagem A.21.

```
1  int TotalCores = g.VirtualMachines.Where(t =>
       t.VmState.Contains("running")).Sum(t => t.NumberOfCores);
2  foreach (VirtualMachineDTO vm in g.VirtualMachines){
3      WebServerDTO w = GetWebServerByVM(vm, g);
4          w.Weight = Generic.FormatToPercentage(vm.NumberOfCores, TotalCores);
5      UpdateWeightsWebServers(w);
6  }
```

Listagem A.21: Cálculo de pesos de cada grupo de servidores web no balanceador de carga

Do resultado dos pesos calculados de cada webserver da Listagem A.21, é necessário aplicar a alterações de cada webserver no servidor de balanceador de carga, que é feito através da instrução *"UpdateWeightsWebServers(w)"*. Essa instrução presente na biblioteca de gestão do balanceador de carga permite definir o tamanho que o dado webserver possuirá naquele grupo (ver Listagem A.22).

```
1  public void UpdateWeightsWebServers(WebServerDTO w){
2      this.client.SetWeight(w.backend.Name, w.Name, w.Weight);
```

```
}
```

Listagem A.22: Função que permite a atualização dos peso num determinado webserver

```
#!/bin/bash
sudo sed -i '/^[[:space:]]*$/d' /etc/haproxy/haproxy.cfg    # apaga todas as linhas em branco
cat /etc/haproxy/haproxy.cfg | grep -P -B5000 '^backend webservers' > /tmp/aux_haproxy.txt  #reserva espaco para as linhas 5000
cat /etc/haproxy/haproxy.cfg |  grep -P -A5000 '^backend webservers' | sed "s/.*server.*//g" >> /tmp/aux_haproxy.txt #reserva espaco para as linhas 5000
sudo mv /tmp/aux_haproxy.txt /etc/haproxy/haproxy.cfg
sudo sed -i '/^[[:space:]]*$/d' /etc/haproxy/haproxy.cfg
sudo systemctl restart haproxy
```

Listagem A.23: Script com o nome *cleanGroupHaproxy.sh* responsável por apagar servidores web do haproxy.cfg

A.5 TERRAFORM COM O AZURE DEVOPS

Transferência de variáveis

O *Azure DevOps* utiliza templates de formato *YAML* onde estão presentes instruções como por exemplo invocação do comando de execução do *Terraform*.

```
[...]
- script: Terraform plan && Terraform apply -auto-approve
workingDirectory: ./Terraform/Terraform_360_windows/devtest
env:
  TF_VAR_region: "${{parameters.region}}"
  TF_VAR_aset: "${{parameters.aset}}"
  TF_VAR_rgoup: "${{parameters.group}}"
  TF_VAR_vmnameList: "${{parameters.vmname}}"
  TF_VAR_vmsize: "${{parameters.vmsize}}"
  TF_VAR_subnet: "${{parameters.subnet_name}}"
  TF_VAR_vnet: "${{parameters.vnet_name}}"
  TF_VAR_lb_dns: "${{parameters.lb_dns}}"
  TF_VAR_lb_privateip: "${{parameters.lb_privateip}}"
  TF_VAR_lb_groupname: "${{parameters.lb_groupname}}"
  TF_VAR_monitor_privateip: "${{parameters.monitor_privateip}}"
displayName: 'Terraform PLAN E APPLY: Run Terraform PLAN e APPLY'
```

```
17 [...]
```

Listagem A.24: Passo da pipeline que permite execução dos comandos *Terraform plan* e *Terraform apply* para aplicar as alterações do ficheiro *Terraform*

A Listagem A.25 ilustra o processo de importação do grupo de recursos com base na variável com o nome *rgoup* existente no ficheiro *variables.tf*.

```
1 data "azurerm_resource_group" "resource_gp" {
2   name = "${var.rgoup}"
3 }
```

Listagem A.25: Carregamento do grupo de recursos com base do nome fornecido por parâmetro

```
1 output "map_instance_name_ip" {
2   value       =  jsonencode(zipmap("${azurerm_virtual_machine.main.*.name}",
        "${azurerm_network_interface.main.*.private_ip_address}"))
3   description = "Create a map with name and ip."
4 }
```

Listagem A.26: variável de output do Terraform que origina um dicionário com o nome de cada máquina virtual seguido do seu IP privado

```
1 - bash: |
2   nameIP=$(Terraform output map_instance_name_ip)
3   echo "##vso[task.setvariable variable=DICTNAMEIP;]$nameIP"
4   echo $DICTNAMEIP > DictionaryWithIpAndName.txt
5 workingDirectory: ./Terraform/Terraform_360_windows/devtest
6 displayName: 'SET VAR: NAMEIP_DICTIONARY'
```

Listagem A.27: Atribuição da variável extraída pelo Terraform para uma variável da pipeline do Azure DevOps

```
1 - script: |
2         sshpass -p $VMAP scp -o StrictHostKeyChecking=no cleanGroupHaproxy.sh
              $VMAU@${{parameters.lb_dns}}:/tmp
3         sshpass -p $VMAP ssh -tt $VMAU@${{parameters.lb_dns}} 'sudo chmod +x
              /tmp/cleanGroupHaproxy.sh'
4         sshpass -p $VMAP ssh -tt $VMAU@${{parameters.lb_dns}} "sudo
              /tmp/cleanGroupHaproxy.sh ${{parameters.operation}}
              ${{parameters.lb_groupname}}"
5 workingDirectory: ./Terraform/Terraform_360_windows/extra
6 displayName: 'HAPROXY CLEAN: EXECUTE CLEAN HAProxy to remote server via SSH'
7 condition: eq('${{parameters.operation}}', 'ScaleIn')
```

Listagem A.28: Tarefa da pipeline que copia e executa o script com o nome

Tarefa com pedido para registo de hosts no servidor de monitorização

```
[...]
- script: |
  curl -k -H 'Content-Type: application/form-data' -H 'Content-Length: 0' -X
      POST
      "https://bizayautoscale-devops.azurewebsites.net/WebAPI/AddHostZabbixServer
      ?GroupName=${{parameters.lb_groupname}}&
      MonitorServerIp=${{parameters.monitor_privateip}}&PrivateipList=$PIP&
      operation=${{parameters.operation}}"
displayName: 'ZABBIX: ADD HOST TO ZABBIX SERVER'
workingDirectory: ./Terraform/Terraform_360_windows/devtest
name: AddHostZabbixServer
[...]
```

Listagem A.29: Pedido à aplicação responsável por registar as máquinas virtuais no servidor do zabbix

A.6 Dimensionamento horizontal - Registo em base de dados

```
IEnumerable<CloudVirtualMachine> ListVMsCloud = await
    this.CloudResourceGroup.GetVirtualMachinesFromSubscriptionByIP(group.SubscriptionKey,
    PrivateIpArr);
List<string> vmThatExistinBD = ListVMsCloud.Where(azurevm =>
    group.VirtualMachines.Any(vm =>
    vm.PrivateIP.Equals(azurevm.PrivateIP))).Select(t =>
    t.PrivateIP).ToList();
List<VirtualMachineDTO> VmsToDelete = group.VirtualMachines.Where(t =>
    !vmThatExistinBD.Contains(t.PrivateIP)).ToList();
foreach (VirtualMachineDTO vm in VmsToDelete){
    await this.VmRepository.DeleteVirtualMachine(vm);
    [...]
}
```

Listagem A.30: Processamento das máquinas virtuais na operação de *Scale In* antes da persitência das máquinas virtuais

No contexto de uma operação de *Scale Out* a Listagem A.31 aborda o processo de persistir máquinas virtuais, evitando que haja duplicação de registos e que a aplicação tenha de registar alguma atualização que exista.

No método existe uma validação se o IP privado de uma máquina virtual existe na base dados, então o registo relativo a esta máquina virtual é atualizado, caso contrário é criado.

```
List<VirtualMachineDTO> vmsGroup = group.VirtualMachines;
foreach (CloudVirtualMachine z in ListaTodasVMsCloud){
    VirtualMachineDTO v = new VirtualMachineDTO(){[...],
        WasProvisioned = true};
    if (vmsGroup.Any(t => t.PrivateIP.Equals(z.PrivateIP))){
        await this.VmRepository.UpdateVirtualMachine(v);[...]
    }else{
        int VmId = await this.VmRepository.CreateVirtualMachine(v);[...]
    }[...]
```

Listagem A.31: Processamento das máquinas virtuais na operação de Scale Out antes da persistência de maquinas virtuais

A.7 ZABBIX

Registo da máquina virtual nos servidor do Zabbix

O método *CreateHost(string Hostname, string PrivateIp)* permite registar o *host* no servidor do Zabbix, que aceita como parâmetro o nome e o IP privado da máquina virtual, atributos necessários para a construção do objeto *host*.

Os valores dos templates são definidos estaticamente com o valor dos dois templates definidos na secção 3.2.1 pois apenas estes dois são os utilizados pela equipa de infraestrutura da 360imprimir para configuração de cada máquina virtual, no entanto para trabalho futuro a aplicação permite configurar os *templates* de forma remota.

```
string[] templateNameArr = { "Template OS Windows by Zabbix agent", "Template
    App IIS Service" };
[...]
HostInterface HostInterfaceObj = new HostInterface(){
    type = HostInterface.InterfaceType.Agent,
    ip = PrivateIp,
};
Host HostObj = new Host(){
    name = Hostname,
    interfaces = HostInterfaceList,
    groups = HostGroupList,
    parentTemplates = TemplatesList,
};
```

```
13  var host = context.Hosts.Create(HostObj);
```

Listagem A.32: Método *CreateHost* que serve para realizar a configuração do Host no servidor de zabbix

B

ANEXOS

Simulação 1 - dimensionamento vertical - monitorização global das 2 máquinas virtuais

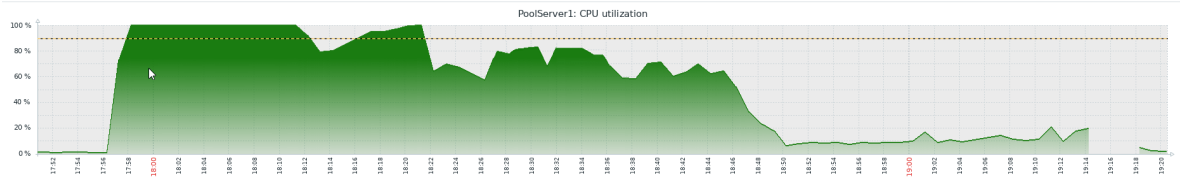


Figura 75: Gráfico de percentagem de utilização de CPU extraído da monitorização da *PoolServer1* durante a realização da simulação 1

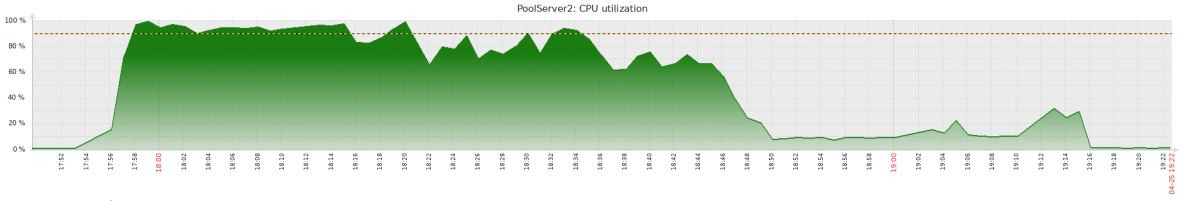


Figura 76: Gráfico de percentagem de utilização de CPU extraído da monitorização da *PoolServer2* durante a realização da simulação 1

NB: place here information about funding, FCT project, etc in which the work is framed. Leave empty otherwise.