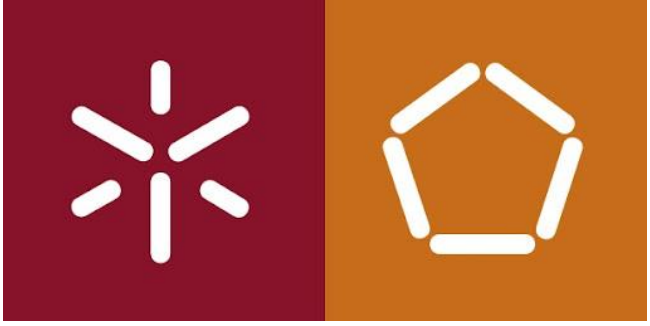


Universidade do Minho

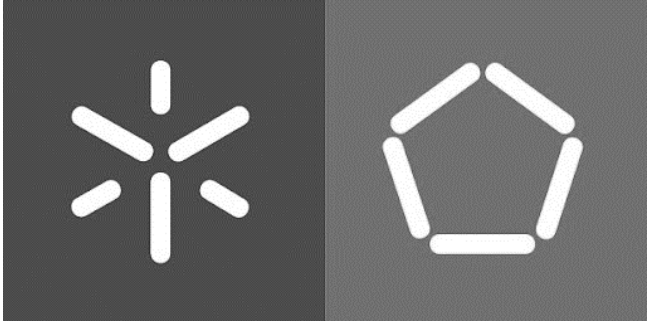
Escola de Engenharia

Departamento de Informática

João Pedro de Vasconcelos Cadavez Ferreira

Desenho e Implementação de Processos de Automação de IaaS em Ambientes Cloud

outubro 2022



Universidade do Minho

Escola de Engenharia

Departamento de Informática

João Pedro de Vasconcelos Cadavez Ferreira

Desenho e Implementação de Processos de Automação de IaaS em Ambientes Cloud

Dissertação de Mestrado

Mestrado em Engenharia de Redes e Serviços Telemáticos

Trabalho Realizado Sobre a Orientação de:

Professor Pedro Nuno Miranda de Sousa

Supervisor na entidade:

Engenheiro Marco André Magalhães

outubro 2022

Desenho e Implementação de Processos de Automação de IaaS em Ambientes Cloud

João Pedro de Vasconcelos Cadavez Ferreira

Dissertação apresentada à Universidade do Minho para obtenção do grau de Mestre em Engenharia de Redes e Serviços Telemáticos sob a supervisão científica do Professor Doutor Pedro Nuno Miranda de Sousa, e com coordenação na entidade pelo Engenheiro Marco André Magalhães.

Universidade do Minho

Escola de Engenharia

Departamento de Informática

outubro 2022

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.

Agradecimentos

O presente documento é o culminar de vários meses de dedicação e trabalho, que, no entanto, não seriam possíveis sem o constante apoio do ciclo das pessoas corretas. Gostaria de destacar e agradecer o apoio constante do Professor Pedro Nuno Miranda de Sousa na orientação académica do documento e na constante disponibilidade.

Agradeço também a toda a equipa Datacenter dos Serviços Partilhados do Ministério da Saúde que possibilitou o ambiente ideal para a realização do trabalho, assim como o constante apoio técnico nos constrangimentos do projeto. Um especial obrigado ao Engenheiro Marco André Magalhães e ao Engenheiro Ricardo Miguel Cardoso Maia pela orientação e disponibilidade.

Por último, quero agradecer à minha mãe, pai, irmã e à Joana pela ajuda constante, assim como de todos os familiares e amigos pelo constante apoio pessoal e carinho demonstrado, sem o qual não seria possível alcançar esta meta.

Resumo

Com o crescimento exponencial dos serviços hospedados em ambientes digitais e consecutivamente com o aumento da criticidade dos mesmos, tem-se verificado uma procura constante por parte de desenvolvedores e gestores de projetos por plataformas que disponibilizem agilidade, flexibilidade e baixa complexidade para a disponibilização ao público das suas soluções o mais rápido possível com o mínimo de esforço por parte dos mesmos.

Este trabalho tem como objetivo precípuo o estudo e implementação de uma plataforma de IaaS em *Cloud* privada, com a finalidade de disponibilizar a várias entidades da área da saúde uma plataforma centralizada de gestão de ativos de computação e de rede, com o intuito de facilitar, perante o paradigma passado, a logística associada à criação e coordenação dos ativos por parte das mesmas. Para tal o presente trabalho propõe um estudo de mercado, seguido de uma análise de formas e plataformas de automação de processos a serem implementados intrinsecamente e/ou extrinsecamente à plataforma de *Cloud* privada, de modo a trabalharem em simbiose. São também apresentadas metodologias de desenho de *scripting* necessário para a realização dos casos de uso propostos pela entidade SPMS, assim como o processo utilizado para a integração da solução com plataformas e serviços terceiros.

Com este trabalho intenciona-se proporcionar à SPMS a otimização dos seus recursos computacionais e de rede, bem como diminuir drasticamente as horas humanas dedicadas a processos repetitivos e iterativos, canalizando-as para processos mais nobres.

Palavras-chave: IaaS, Cloud Privada, Automação, Scripting, Otimização

Abstract

With the exponential growth of services hosted in digital environments and the increase in their criticality, it has created a constant demand on the part of developers and project managers for platforms that provide agility, flexibility and low complexity to make their services available to the public solutions as quickly as possible with minimal effort on their part.

The main objective of this work is the study and implementation of an IaaS platform in a private Cloud with the purpose of providing several entities in the health area with a centralized platform for the management of computing and network assets, in order to facilitate, in the face of the past paradigm, the logistics associated with the creation and coordination of assets by them. With that in mind, the present work proposes a market study, followed by an analysis of processes automation forms and platforms to be implemented intrinsically and/or extrinsically to the private Cloud platform, in order to work in symbiosis. Methodologies for designing the scripting necessary to carry out the use cases proposed by the company are also presented, as well as the process used to integrate the solution with third-party platforms and services.

With this work, the intention is to provide the company with the optimization of its computational and network resources, as well as drastically reducing the human hours dedicated to repetitive and iterative processes, channeling them to more noble processes.

Keywords: IaaS, Private Cloud, Automation, Scripting, Optimization

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos e Estrangeirismos

ACI · Application centric infrastructure

API · Application programming interface

Active Directory – Diretório ativo

Ad aeternum · Para a eternidade

Agent · Agente

As-a-Service – Como serviço

Backend · Processo interno

Backups · Cópia de segurança

Baremetal – Servidor físico

Block Based Coding – Código baseado em blocos

Block programming – Programação em blocos

Boolean – Boleano

Broadcast · Comunicação todos para todos

Brown field – Campos Castanhos (Ambientes já em produção)

CSV · Comma-separated values

Case sensitive – Sensível ao tipo de letra

Checkpoints – Ponto de restauro

Client-based · Baseado no cliente

Closed-source – Programa pago

Cloud Computing – Computação em nuvem

Cloud Providers – Provedores de computação em nuvem

Cloud – Nuvem

Code as a Service – Código como serviço

Community Cloud – Nuvem comunitária

Constraints · Restrições

Container as a Service – Contentores como serviços

Containers · Contentores

Continuous integration / continuous delivery – Integração continua / Entrega continua

DHCP · Dynamic Host Configuration Protocol

DNS – Domain Name Server

Datacenter – Centro de dados

Debug – Pesquisa de erros

Default gateway – Rota padrão

Deployments · Implantações

Design · Desenho

Distributed denied of service · Ataque de negação de serviço distribuído

Downtime – Tempo em baixo

Drag-and-drop – Pegar e largar

Enterprise · Empresarial

Firewalls – Parede de fogo

Firmwares – Controlo de baixo nível do equipamento

Flavor – Sabor, tipo

Framework – Abstração de código

Frontend – Processo de primeira linha

Green field – Campo verde

HDD – Hard Disk Drive

HTTP · Hypertext Transfer Protocol

HTTPS · Hypertext Transfer Protocol Secure

Hardware · Equipamento

Hostname – Nome do utilizador

Hosts · Hospedeiro

Hypervisors · Monitor de máquina virtual

IaaS · Infrastructure as a Service

Infrastructure as a Service – Infraestrtura como serviço

Input · Entrada

Integrated Development Environment · Ambiente de desenvolvimento integrado

Kernel · Núcleo

LDAP · Lightweight Directory Access Protocol

Layer · Camada

Lease · Alugar

Legacy – Desatualizado

Length · Comprimento

Login · Entrar

Logs · Registos

Loop · Ciclo

Low-code – Baixo código

MIB · Management information base

Mailing list – Lista de email

Malware · Software malicioso

Manager · Gestor

Member · Membro

Middleware – Software de ponte

Networking · Rede

Number · Número

OID · Identificador de objeto

Off-premise – Não hospedado localmente

On-demand – Em procura

On-premises – Hospedado localmente

On-site – Hospedado localmente

On-the-go – Sem interrupções

One-size-fits-all – Um tamanho serve a todos

Online – Em linha

Open-source – Código aberto

Output – Saída

Overhead · Sobrecarga

Pay-As-You-Go · Pagamento conforme o uso

Pay-as-you-consome · Pague conforme o consumo

Plataform-as-a-Service – Plataforma como serviço

Proxy Server – Servidor proxy (representante)

Queries – Consulta

RDP – Remote Desktop Protocol

Reboot · Reinício

Rollback · Retornar

SDN – Software Defined Networking

SLAs – Service Level Agreement

SMTP · Simple Mail Transfer Protocol

SSD – Solid State Disk

SSH · Secure Shell

Scheduled · Agendado

Scripting – Roteiro, Algoritmo

Server-based – Baseado no servidor

Single Responsability Principle – Princípio de responsabilidade única

Software Defined Networking – Rede definida por computação

Software – Suporte lógico

Software-as-a-Service – Programa como serviço

Steakholder · Grupo de interesse

Streaming – Transmissão em direto

Strings – Cadeia de caracteres

Subscriptions · Subscritores

Tags · Etiqueta

Templates · Modelo de documento

Threshold · Limite

Tokens · Fichas

Troubleshooting – Resolução de problemas

URL · Uniform Resource Locator

Username – Nome do utilizador

Viewer – Espectador

Virtual Machine – Máquina virtual

Workshops – Oficina de trabalho

XML · Extensible Markup Language

YAML · YAML Ain't Markup Language

Zero trust – Confiança Zero

Capítulo 1 - Introdução

O presente capítulo propõe difundir a ideia por detrás do tema abordado no presente trabalho, enquanto explica o enquadro motivacional pessoal e coletivo que levou à realização do estudo e da implementação. São também apresentados os principais objetivos traçados, como também as contribuições espectáveis definidas no início do projeto. Para melhor entendimento da estruturação do documento, é ainda incluído no capítulo uma breve descrição da mesma.

1.1 Enquadramento e Motivação

Com o rápido crescimento da Internet e aumento da criticidade e dependências dos serviços que nela estão disponíveis, também se verificou um crescimento do esforço humano, por parte dos técnicos e engenheiros informáticos, para responder à procura exponencial de novos serviços e, ao mesmo tempo, efetuar a gestão dos serviços já existentes. Em adenda, pequenas instituições e empresas que necessitam de disponibilizar serviços *online,* mas não possuem meios financeiros nem recursos humanos especializados, ficam limitadas à contratação de terceiros com SLAs abaixo do padrão da competitividade ou mesmo inexistentes. Isto resulta em aplicações desenvolvidas sobre plataformas desatualizadas, devido às elevadas dependências criadas sobre as aplicações sem manutenção, a equipamentos de segurança mal configurados criando falhas de segurança para as entidades, ou equipamentos de rede com configurações *out-of-the-box*[1], não aproveitando o potencial de segmentação e granularidade fornecida por estes dispositivos. Em conjunção, as entidades procuram reduzir custos com a compra de equipamentos e licenças, de modo a focarem os seus esforços no seu negócio.

Com isto em mente, verifica-se que existe uma necessidade emergente de plataformas de caráter genérico e centralizadas, que disponibilizem a diferentes clientes soluções customizáveis e/ou padronizadas que possam ser geridas por uma ou mais equipas especializadas.

Para tal, o uso de plataformas *IaaS* possibilita impulsionar para as extremidades a criação e gestão das máquinas virtuais de uma forma segura e acessível, permitindo aos engenheiros, de forma transparente para o subscritor, a gestão do equipamento *baremetal*, da *networking* e das plataformas de virtualização, assim como a instalação

[1] Configurações introduzidas por defeito nos equipamentos

de novos nós de processamento e a migração em tempo real das máquinas entre os diferentes nós, sem que tal seja notado pelo cliente ou subscritor final.

1.2 Objetivos

Pretende-se, com a presente dissertação, efetuar a configuração de uma plataforma *IaaS* nos Serviços Partilhados do Ministério da Saúde (SPMS), de modo a disponibilizar às diferentes entidades parceiras, tais como hospitais, Santas Casas da Misericórdia ou clientes do ramo da saúde, a possibilidade de migrar os seus serviços para uma solução centralizada e gerida pela SPMS, com o intuito de melhorar o serviço prestado ao cliente, facilitar o *troubleshooting* da equipa Datacenter da SPMS e padronizar processos que, à medida do atual, não se encontram coerentes. A adoção de uma plataforma IaaS permitirá também poupar horas dos recursos humanos da empresa, possibilitando o foco desses engenheiros e técnicos em assuntos de caráter mais produtivo, ao mesmo tempo que permite otimizar os recursos computacionais em uso.

Para tal, será necessário entender os principais conceitos inerentes às soluções de *Cloud*, assim como o atual estado da arte. Em seguida, será necessário entender a organização e o funcionamento da solução previamente adquirida pela SPMS, o *vRealize* da *VMware*, sendo o principal foco desta dissertação a ferramenta *Automation*. Logo após, será efetuado um levantamento dos casos de uso necessários à SPMS, consequente desenho dos mesmos e propostas de abordagem para cada um. Por fim, será efetuada a implementação dos mesmos e efetuados os testes necessários para garantir a coesão da solução proposta.

1.3 Principais Contribuições

Com a presente dissertação pretende-se disponibilizar à SPMS uma panóplia de módulos genéricos capazes de serem usados pela entidade e pelos subscritores que adotarem a solução, de modo a simplificar e organizar o processo atual de criação de máquinas e gestão de ambientes.

Será também possível, com o êxito da implementação, poupar inúmeras horas de operação dos engenheiros assim como melhorar a experiência de serviço dos utilizadores finais e dos parceiros que adotarem a solução.

1.4 Organização do Documento

O presente documento encontra-se dividido em seis principais capítulos. No primeiro capítulo encontra-se descrita a introdução do documento, no qual são resumidas as principais motivações do projeto, bem como os seus objetivos e as suas notáveis contribuições para a SPMS.

Em seguida, o capítulo de Estado da Arte disponibiliza uma revisão sobre alguns conceitos inerentes à plataforma em questão, tal como uma revisão dos trabalhos relacionados e de algumas plataformas IaaS disponíveis no mercado.

Logo após, encontra-se disponível o capítulo de Especificação e Proposta Arquitetural, no qual é especificada a abordagem tomada perante os casos de uso propostos, assim como a base de trabalho e a explicação dos mecanismos disponibilizados pela plataforma.

No capítulo Implementação da Solução foram colocados em prática os casos de uso discutidos no capítulo antecedente, ao mesmo tempo que foram disponibilizados os diagramas de sequência que possibilitam entender o fluxo de informação na solução e a relação entre os diferentes componentes.

Na sequência, apresentam-se os Testes e Exame de Resultados para cada um dos casos de usos, avaliando o desempenho de cada solução perante o objetivo inicialmente traçado. Nesta fase, são também apresentados os resultados e algumas conclusões de cada caso de uso.

Por último, é efetuada uma avaliação final do trabalho desenvolvido e apresentada uma sinopse das contribuições obtidas no âmbito do projeto. São também avaliados os entraves do trabalho apresentado, assim como os trabalhos futuros a desenvolver neste âmbito.

Durante o documento são utilizados termos em línguas estrangeiras que se encontram identificados pelo tipo de letra itálico, sendo possível confirmar o seu significado ou tradução direta na secção de Acrónimos e Estrangeirismos. É ainda importante salientar que são usadas palavras-chave para identificar os diferentes atores. No caso da palavra “cliente” destina-se aos clientes finais da solução (por exemplo quem usa o serviço, visita o site, etc.), o “subscritor” é o agente que subscreve a ação e que tem como objetivo servir o “cliente”. Por último o “administrador” gere e mantém a plataforma de IaaS, prestando serviços diretamente ao “subscritor” e indiretamente ao “cliente”.

Capítulo 2 - Estado da Arte

No presente capítulo é apresentado o estado de arte dos temas Automação em *Cloud* e Plataformas IaaS, identificando as opções de plataforma atuais que mais se destacam no mercado, assim como as suas características mais relevantes, de modo a, antes de inicializar a implementação, entender o trabalho já efetuado na área e a possibilitar análise compreensiva sobre as decisões efetuadas nos capítulos futuros.

2.1 *Cloud Computing*

Com a necessidade crescente de poder computacional espoletado pela migração em massa dos serviços para ambientes digitais [1], o *Cloud Computing* apresenta-se como um método viável para muitas pequenas e médias empresas. Como descrito em *Cloud Computing – Computação em Nuvem* [2], o conceito de *Cloud*[2] antecede da representação dos diagramas de rede em que se referia à Internet como uma nuvem pelo facto do seu funcionamento ser nubloso para o utilizador (e mesmo para os engenheiros de rede), mas por, ao mesmo tempo, ser um elemento fulcral. Assim, com o aparecimento de plataformas que disponibilizam poder computacional com a mesma proposta de turvação, foi adotado o nome de *Cloud Computing*.

O *Cloud Computing* tem como principal característica ser agnóstico ao local geográfico onde o poder computacional e o armazenamento se encontram. Desta forma, muitas das preocupações inerentes à posse, gestão e manutenção dos recursos computacionais passam para os gestores (ou plataforma) da solução *Cloud* em questão, apresentando-se como uma vantagem para os subscritores que não têm capacidade para contratar profissionais especializados e/ou não lhes compensa o investimento de *hardware* e *software* necessário para implementar as suas soluções. Outras vantagens passam também pela melhor relação preço/qualidade da segurança virtual da informação armazenada pelo cliente, como reportado pela *Palo Alto Networks*[3] em 2020, noventa e nove porcento das falhas de segurança a nível das *firewalls* são de origem humana por erros na configuração das mesmas [3]. Por outro lado, a nível de segurança física também é possível fornecer serviços de segurança de perímetro, redundância de energia e de poder computacional, assim como vigilância 24/7, representando para o subscritor uma pequena percentagem do preço final em solução *Cloud*, enquanto em ambiente *on-premises* ou *on-site*, para um mesmo nível de segurança, representa uma

[2] No presente trabalho quando é referido o conceito *Cloud* possui a mesma denotação de *Cloud Computing*.

[3] paloaltonetworks.com

grande fatia do preço da solução. A gestão da camada de *networking* é também um ponto vantajoso, uma vez que também é garantida pelos responsáveis da solução *Cloud*, sendo cuidadosamente projetada para garantir redundância e resiliência perante diferentes cenários sem afetar os diferentes subscritores. Por último, é também importante reforçar que só é possível disponibilizar uma solução *Cloud* com a implementação de *hypervisors,* descrita na secção *Hypervisors*, sendo que a implementação e gestão dos mesmos pode ser por vezes complexa, tornando-se mais recomendado deixar a cabo de equipas especializadas na temática.

2.2 *Infrastructure as a Service*

Com o aparecimento de diferentes propostas de *Cloud Computing,* vários métodos de *leasing* de poder computacional e armazenamento foram aparecendo, entre eles o *Infrastructure as a Service* ou IaaS.

O IaaS é uma subcategoria de *Cloud Computing* que providencia ao subscritor os detalhes de sistema operativo em uso, quantidade de cores de CPU, quantidade de memória RAM, de disco disponível entre outros. Assim, o subscritor possui acesso explicito às VMs (*Virtual Machines*) do seu domínio (alocadas ao seu utilizador), criando assim uma bolha isolada, podendo montar o seu ambiente às suas necessidades e sendo responsável pela sua configuração após a camada de virtualização da plataforma IaaS [4].

A utilização de plataformas IaaS pode estar presente em *Cloud* privadas ou públicas, podendo também estar acoplada a diferentes métodos de *leasing* de recursos discutido na secção Métodos de *Leasing* e apresenta-se, à medida do atual, como um dos métodos mais populares entre *Cloud Providers* como o *Google Cloud*[4], *Azure*[5] ou *AWS*[6].

2.3 Automação

A definição de automação vai muito para além da temática de *Cloud Computing* estando presente em todos (ou quase todos) os processos que envolvem trabalho humano. Na sua forma mais pura e genérica, o ato de automação refere-se ao processo de introdução de mecanismos, máquinas ou processos a tarefas anteriormente

[4] cloud.google.com
[5] azure.microsoft.com
[6] aws.amazon.com

executadas por humanos com o objetivo de facilitar, acelerar ou mesmo tornar possível processos anteriormente limitados por fatores anatómicos [5].

Especificamente na temática de *Cloud Computing* existe uma necessidade intrínseca de automação a diferentes níveis. Grande parte das soluções de *Cloud* disponibilizam ao subscritor final uma plataforma de gestão de custos, máquinas e recursos sendo indispensável alterar o paradigma passado de gestão de ambientes virtuais. Assim, processos rotineiros de criação e eliminação de máquinas, adição de discos ou de placas de rede, que eram executados manualmente por técnicos, passam a ser programados e executados perante os pedidos de cada subscritor pela plataforma. Deste modo, é possível garantir a coerência dos processos, assim como uma rápida e eficaz resposta aos pedidos por parte da solução *Cloud* [6].

2.4 *Cloud Computing*, Organização Lógica e Arquiteturas

A *Cloud Computing*, como descrito anteriormente, ainda é, à medida do atual, um conceito bastante flexível pela diversidade de serviços que podem ser oferecidos e pelos diferentes tipos de modalidades de *Clouds*. É importante entender que com o conceito de *Cloud* são criadas premissas como a aparência de recursos computacionais ilimitados, devido à nebulosidade criada para os subscritores, a eliminação da necessidade de adquirir recursos computacionais em antecipado e da instalação dos mesmos e o *ad aeternum* dos serviços e recursos hospedados [2].

2.4.1 Ambientes de *Cloud Computing*

Como supracitado, o *Cloud Computing* pode pertencer a diferentes ambientes dependendo de quem possui o controlo sobre os recursos e a plataforma. Como referido em *Cloud Deployment Models* [7], existem quatro tipos de ambientes *Cloud* diferentes, cada um com o seu propósito e particularidade, de modo a satisfazer as necessidades de cada cliente e solução desejada.

Começando a análise dos ambientes mais fechados para os mais públicos e abertos, o modelo de negócio da *Cloud* privada apresenta-se com moldes de cariz mais reservado à instituição ou empresa que adquire ou gere a solução. Em norma, e por ideologia pré-definida, a solução é especialmente selecionada por quem procura uma maior segurança e privacidade sobre a localização e processamento dos dados da infraestrutura e dos serviços, uma vez que o modelo de negócio tem como princípio a utilização da solução por uma única entidade. No entanto, o modelo privado, apesar de muitas vezes ser associado a soluções *on-premise,* tal premissa não é de todo correta,

uma vez que podem ser tanto adotadas soluções *on-premise* como *off-premise*, continuando-se a tratar, ainda assim, do mesmo modelo privado [7]. Ademais, é ainda importante referir um dos fatores que define a não adoção, por parte de muitas entidades, de uma solução *Cloud* em modelo privado, que passa pelo valor monetário associado ao mesmo. É, contudo, relevante ter em mente que este modelo pode apresentar uma relação recurso/preço menos favorável em relação a modelos de *Cloud* pública, sendo nomeadamente o tamanho da infraestrutura o principal fator a ter em conta aquando da comparação [7].

Em seguida no espetro, o modelo de *Community Cloud* é uma fusão dos conceitos chave presentes nas diversas *Cloud* privadas, sendo estas utilizadas entre diferentes entidades que partilham as mesmas missões ou objetivos. É possível encontrar aplicações de *Community Cloud*, tanto em ambientes *on-premise* como *off-premise*, podendo esta também ser gerida pela organização ou por terceiros. A utilização deste modelo é particularmente útil em organizações e entidades que partilhem o mesmo modelo de negócio, permitindo organizar e desenhar de forma mais eficiente a plataforma *Cloud*, ajudando a mitigar a falácia *one-size-fits-all* [7].

A utilização de *Cloud* híbridas são caracterizadas pela utilização, em simbiose, de duas ou mais *Cloud* públicas e privadas ao mesmo tempo. Este modelo possibilita a união das vantagens de ambos os modelos público e privado, oferecendo uma solução mais flexível e resiliente sobre cada uma delas. É especialmente utilizado em entidades que necessitem de um crescimento rápido em momentos em que não é possível instalar e configurar *hardware* para suprimir a procura [7].

Por último, o modelo mais conhecido e possivelmente mais utilizado por subscritores e clientes finais, a *Cloud* pública. A *Cloud* pública apresenta-se no extremo do leque dos modelos, estando de forma indiscriminada, disponível ao público que deseje desenvolver soluções na mesma. O modelo é, em modo geral, considerado mais barato para o subscritor, não necessitando de capital inicial para utilizar a solução, ao contrário dos restantes modelos, estando disponível *on-the-go*, evitando esperar meses para o desenvolvimento e implementação de outros modelos. É também importante salientar que os modelos de *Cloud* pública apresentam preocupações no que toca ao armazenamento e processamento de dados dos clientes. Na essência do seu modo de funcionamento de negócio, o processo acaba por não apresentar a transparência necessária ao subscritor e ao cliente final para empregar dados, considerados sensíveis e de alto risco, acabando por afastar este modelo de solução a muitos subscritores [7].

2.4.2 *as-a-Service*

O conceito *as-a-Service* introduz a conceção da disponibilização de qualquer tipo de recurso ou serviço ao cliente final e/ou ao cliente que contrata o serviço. A definição é tão vasta que acaba por se espalhar a outras áreas que nada ou pouco têm a ver com a informática, sendo criados (para além dos três principais referidos em seguida no presente tópico) vários acrónimos para definir a sua função e a sua área de atuação. Os acrónimos seguem a nomenclatura de uma ou mais letras para definir o ramo de atuação seguido de aaS, existindo até, inconvenientemente, siglas com a mesma formação, mas significados diferentes, como é o caso de CaaS que pode ser usado para referir a *Code as a Service* ou *Container as a Service*.

Na temática atual, existem três tipos de serviços (*as-a-Service*) pertinentes de abordar devido à sua abrangência e conceção generalizada como os principais na área da disponibilização de recursos computacionais, sendo eles o *Infrastrute-as-a-Service* (IaaS), o *Plataform-as-a-Service* (PaaS) e o *Software-as-a-Service* (SaaS), Figura 2.1.

Diferentes clientes possuem diferentes tipos de requisitos de *leasing* de poder computacional. Equipas mais focadas na área de sistemas têm tendência a desejar maior controlo sobre as suas máquinas virtuais (VM), como, por exemplo, mudar o sistema operativo, adicionar ou remover discos ou alterar placas de rede, entre outros. Neste caso, uma plataforma IaaS permitirá, da melhor forma, suprimir as necessidades do cliente em questão [4]. Noutro espetro, um cliente que necessite, por exemplo, de uma plataforma de auxílio ao desenvolvimento de *software* para efetuar testes do seu código, necessita apenas de interagir com aplicações específicas, sem necessidade de configurar o sistema operativo ou sistema de virtualização, sendo mais apropriado e cómodo o uso de *Plataform as a Service* ou PaaS [4]. Por último, *Software as a Service* ou SaaS é mais orientado para prestar serviços diretamente ao utilizador final a partir de um *browser* ou de uma aplicação, ficando à responsabilidade do prestador de serviços toda a gestão da pilha de serviços necessários para o bom funcionamento do mesmo.

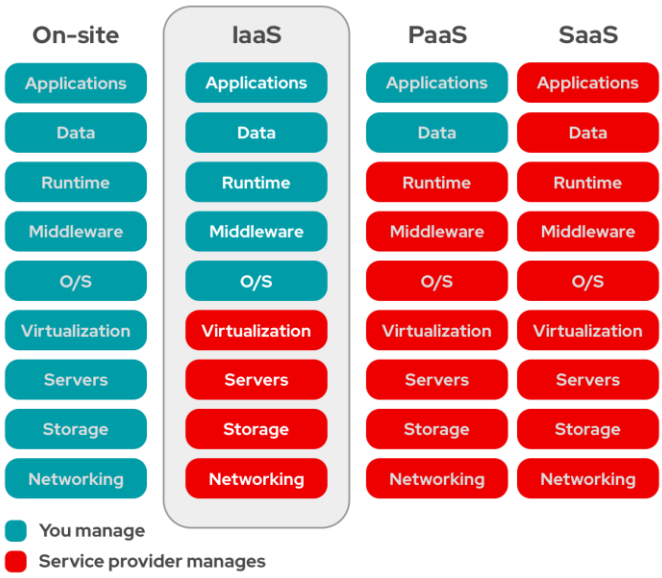


Figura 2.1 · Diferentes tipos de hospedagem de serviços [8]

2.4.3 Principais Vantagens e Desvantagens

Apesar de serem apresentadas como *one-size-fits-all*, as soluções de *Cloud* demonstram um vasto leque de vantagens difíceis de atingir em ambientes *on-premise*, no entanto, e quase numa situação de ação/reação, existem também algumas desvantagens e problemas que até à data do presente documento não foram solucionadas. Durante o restante subcapítulo serão apresentadas as principais vantagens e desvantagens do desenvolvimento em *Cloud*, assim como os principais beneficiados e prejudicados com cada particularidade apresentada.

Os benefícios apresentados pela *Cloud* são sem dúvida o seu ponto forte para ajudar à mudança de paradigma empregue pela maioria dos detentores de soluções que dependem do meio informático para o seu bom funcionamento.

Como descrito em *Study on advantages and disadvantages of Cloud Computing – the advantages of Telemetry Applications in the Cloud* [9], a *Cloud* destaca-se por possuir boa qualidade/preço perante os recursos que disponibiliza, por possuir planos de restauro e backups podendo estes ser transparentes ao subscritor da solução *Cloud*, ou propositadamente definidos pelos mesmos. Outro ponto atraente é o fácil requisito para a implementação de sistemas operativos e serviços de forma autónoma, assim como a gestão de licenciamento, caso se aplique, facilitando o processo de desenvolvimento do subscritor. É também possível, a qualquer momento, escalar os recursos da sua máquina, em contraste com os períodos de manutenção que se traduzem em horas de indisponibilidade de serviços, caso estes se encontrassem numa máquina física na posse do subscritor. Das vantagens descritas beneficiam em grande

parte subscritores que possuem uma necessidade de poder computacional e de armazenamento considerado baixo ou médio, uma vez que os descarta de toda a instalação e manutenção associada aos equipamentos físicos, da rede e dos *softwares* que necessitam para suportar o seu negócio.

Por outro lado, opções como armazenamento (quase) ilimitado e processos de replicação de dados representam para o subscritor menos uma preocupação, sendo só necessário "requisitar mais". O fácil acesso à informação armazenada é também um processo simples, podendo ser consumida por diferentes dispositivos em qualquer local geográfico. Em adenda, perante um portefólio muito diverso de subscritores é sempre necessário desenvolver ambientes à medida para novas aplicações ou conceitos, possibilitando depois disponibilizar aos restantes subscritores esses mesmos ambientes prontos a serem implementados. De notar, nos pontos descritos no presente parágrafo, beneficiam especialmente subscritores com necessidades computacionais e de armazenamento mais robustas, de modo a dar resposta ao rápido crescimento aplicacional.

No entanto, existem algumas preocupações e problemas associados à *Cloud* que devem ser considerados quando se implementa uma nova solução aplicacional. Apesar das SLAs disponibilizadas pelos grandes provedores de serviços *Cloud* excederem os noventa e nove virgula nove porcento de disponibilidade, como, por exemplo, o *Azure* que garante noventa e nove virgula noventa e nove porcento (previsão de 8 segundos de indisponibilidade por dia), existe o risco em associar todas as soluções informáticas de um subscritor a uma só *Cloud*, que em caso de indisponibilidade não é possível ao subscritor garantir a rápida correção e recuperação dos serviços, assim como a recuperação dos dados. Para além disso, a segurança em *Cloud* é um ponto bastante sensível, visto que o utilizador não tem garantias da localização física e lógica dos seus dados, assim como dos métodos de encriptação e segurança associados aos mesmos, carecendo de especial atenção o armazenamento de dados sensíveis, como dados pessoais, médicos ou judiciais. Seguindo o tema da segurança, é importante ter em mente que *Clouds*, especialmente grandes provedores de serviços, estão 24/7 em constante ataque e são grandes prémios para atacantes, tanto pelas elevadas recompensas associadas em termos monetários, como em termos político-sociais. Por outro lado, o valor qualidade/preço começa a deteriorar-se com a subscrição de mais recursos. Podem parecer contraditórias as afirmações antes efetuadas, no entanto, para subscritores com muitos serviços hospedados em *Cloud,* o valor percentual poupado, em comparação a subscritores com menos recursos associados, vai reduzindo significativamente sendo até mais benéfico o investimento em equipa, *hardware* e *software* para implementação *on-premise*, tornando-se, no entanto, necessária uma avaliação dos prós e contras a priori. Por último, é também importante referir que, por

vezes, é necessário implementar serviços com requisitos muito particulares pelas diferentes razões, acabando por se constatar uma inflexibilidade de algumas funções das plataformas *Cloud*, como, por exemplo, serviços desenvolvidos em linguagem já não suportada por *hardware* e *softwares* atuais, mas que ainda são utilizadas e não são passíveis de serem virtualizadas. A falta de suporte e a complexidade das plataformas *Cloud* obriga a que seja necessário prestar cursos e *workshops* dos portais aos utilizadores e subscritores [9].

Assim, é possível constatar que existem inúmeros problemas associados às plataformas *Cloud*, tanto para subscritores com poucos, como para aqueles com muitos recursos, sendo necessário uma avaliação sensata e pragmática perante cada subscritor se é ou não vantajoso implementar as suas soluções *on-premise* ou em ambientes de *Cloud* pública.

2.5 *Hypervisors*

Os *hypervisors* possuem um papel fundamental na existência, do modo como conhecemos, das soluções *Cloud*. No entanto, antes de aprofundar nos detalhes de cada solução, é necessário entender o que são *hypervisors* e quais os conceitos chave dos mesmos.

O aparecimento dos *hypervisors* alterou o paradigma de desenvolvimento e de gestão de máquinas. Assim, antes da existência dos *hypervisors* sempre que fosse necessário implementar uma nova máquina, o processo passava por comprar um novo servidor, instalar fisicamente no *datacenter*, instalar os *firmwares,* para só depois instalar o sistema operativo desejado [10]. Este processo, para além de lento, é extremamente ineficaz no que diz respeito ao uso de recursos computacionais, espaço físico ocupado e a nível energético. Como a utilização da máquina física depende exclusivamente do sistema operativo e dos *softwares* instalados, por vezes, verifica-se que o uso de recursos é ineficiente, consequentemente a atualização dos recursos da máquina (CPU, RAM, placa de rede, *etc*...) requer desligar a mesma e intervir fisicamente nela, acabando por aumentar o *downtime* [10].

Por outro lado, o espaço físico é também um fator importante para a alteração de paradigmas. Sempre que é necessária a instalação de uma máquina nova é necessário ocupar um espaço físico no *datacenter*, assim como a necessidade de ligações elétricas e de ligações de rede, sendo necessários espaços físicos maiores para hospedar o mesmo conjunto de máquinas (com sistemas operativos independentes), aumentando por consequência o custo de manutenção e gestão [10].

Outro fator de diferenciação é a eficiência energética obtida dependendo da solução utilizada. Com o uso do paradigma antigo, existe uma relação de uma máquina física para um sistema operativo, obtendo-se um rendimento significativamente inferior devido à energia necessária para manter a máquina em funcionamento. No caso dos *hypervisors* (para uma mesma máquina física), também se verifica o uso dessa energia, sendo, no entanto, dividida pelos diferentes sistemas operativos implementados na solução, oferendo um rácio energia/sistema operativo ou energia/serviços comparativamente mais favoráveis[10] (Figura 2.2).

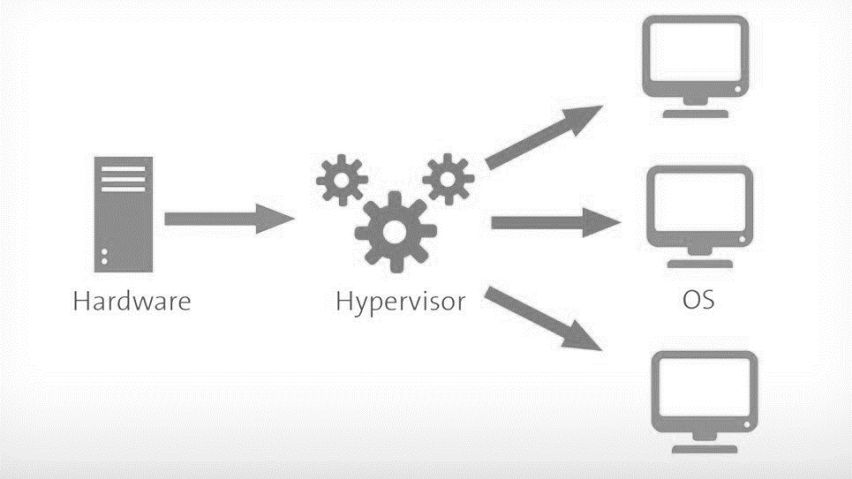


Figura 2.2 - Representação simplista da hierarquia de um *hypervisor* [11]

É também importante referir a existência de dois tipos de arquiteturas de *hypervisors*, a de tipo um e a de tipo dois, como ilustrado na Figura 2.3.

A arquitetura de tipo um destaca-se pela utilização, diretamente no *hardware*, da solução *hypervisor*. Com a utilização de soluções do tipo um é possível tirar mais partido do *hardware*, uma vez que o mesmo se encontra gerido exclusivamente pelo *hypervisor* [12].

Por outro lado, na arquitetura do tipo dois verifica-se a utilização da solução de virtualização sobre um sistema operativo nativo, como uma distribuição *Linux*, *Windows* ou *MacOS*. Esta solução, é notoriamente menos eficaz devido à adição de mais uma camada lógica que acaba por retirar parte da flexibilidade e gestão sobre os recursos físicos do dispositivo físico, perante o tipo um, assim como os recursos usados pelo sistema operativo que hospeda a solução de virtualização. No entanto, a solução de virtualização tipo dois é abertamente usada, não só a nível empresarial ou em contexto de produção, mas também nas máquinas pessoais uma vez que facilita a criação de ambientes de testes, sem afetar o sistema operativo principal, e especialmente porque permite usar diferentes distribuições de sistemas operativos. Opções como *VirtualBox*,

ou *VMware WorkStation* permitem hospedar máquinas virtuais, dando ao utilizador o controlo sobre parâmetros como capacidade de disco, número de *cores,* capacidade de memória RAM, entre outros. Desta forma é possível, na mesma máquina física, por exemplo, sobre um sistema *Windows* possuir uma ou várias distribuições *Linux e/ou Windows*, sem que exista influência das máquinas virtuais sobre o sistema operativo do hospedeiro [12].

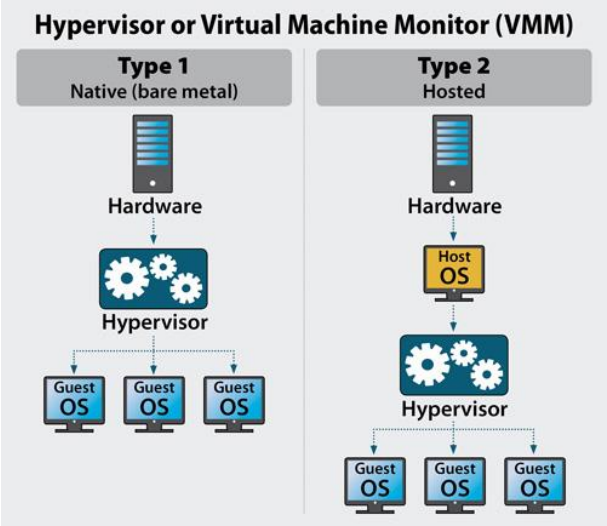


Figura 2.3 - Representação simplista dos dois tipos de *hypervisors* [13]

2.6 Métodos de *Leasing*

Como discutido anteriormente, o contexto de *Cloud* apresenta-se como uma nova página em branco para a tecnologia e o mundo empresarial, assim como para os métodos de negócio sobre os recursos subscritos.

O desenvolvimento de uma *Cloud* pode ser descrito como um processo complexo e extenso de integração de um conjunto de funções e casos de uso requisitados pelos diferentes subscritores, obtendo-se uma solução, idealmente, flexível para todos. E é a flexibilidade que coage para o complexo processo de cálculo de valores de subscrição e também para o aparecimento de diferentes métodos de subscrição.

Em modo de contextualização, o subscritor, dependendo claramente das soluções, possui acesso e liberdade para escolher aspetos como a velocidade do disco (dependendo da aplicação desejada pode ser necessário o uso de discos do tipo SSD sobre os tradicionais HDD, possuindo diferentes preços por gigabyte), a frequência de *backups* da sua infraestrutura ou máquina, assim como o período de retenção desses mesmos *backups*, ou aspetos como apoio técnico para a configuração da infraestrutura,

máquinas ou até mesmos aplicações e ambientes aplicacionais. É, assim, possível entender que o cálculo de custos pode diferir muito entre aplicações, ambientes e equipas, sendo necessário métodos de avaliação de custos que sejam inclusivos e apresentem um saldo neutro ou positivo ao provedor de serviços, de modo à solução ser viável.

2.6.1 *Pay-As-You-Go*

Uma das modalidades mais populares por entre os diferentes provedores de serviços *Cloud,* e também bastante optada pelos subscritores, é a modalidade *pay-as-you-go* ou *pay-as-you-consome*. Esta, permite ao subscritor só pagar pelos recursos que consome durante um intervalo de tempo, ou seja, no caso de ser necessário a criação de uma máquina durante um período reduzido, ou for necessária a alteração de recursos de uma já existente, o subscritor pagará apenas pelo que usa durante o tempo que usa (podendo ser cobrado ao dia, hora ou mesmo minuto) [14].

Este método permite ao subscritor ter uma noção do valor que será cobrado, uma vez que o próprio define os recursos da máquina em questão. Caso o subscritor efetue a alteração de uma máquina, o valor é recalculado e este será somado ao valor gasto até ao momento da alteração.

2.6.2 *On-Demand*

A modalidade de subscrição *on-demand* possibilita ao serviço uma adaptação autónoma dos recursos das máquinas, oferecendo estabilidade para aplicações que possuem requisitos rígidos no que toca a recursos computacionais, possibilita o crescimento autónomo da infraestrutura, eliminando a necessidade de monitorização assim como gestão humana por máquina e oferecendo uma adaptabilidade perante picos de poder computacional espalhados no tempo. Algumas derivações desta modalidade permitem definir valores mínimos e máximos dos diferentes recursos, de modo a mitigar alguns problemas inerentes à modalidade.

Apesar da utilização da modalidade *on-demand* se apresentar tentadora, esta pode apresentar riscos a nível de segurança e a nível de custos.

No que toca a questões de segurança, esta modalidade como possui adaptabilidade autónoma de recursos, no caso de um ataque, como, por exemplo, *distributed denied of service* (DDOS), a plataforma tendencialmente adaptará a máquina em questão, atribuindo mais recursos computacionais. A ação de aumentar os recursos perante um ataque do tipo DDOS pode ser uma ação legítima para evitar que os serviços

fiquem sobrecarregados com os pedidos ilegítimos. Contudo, no caso das plataformas de *Cloud* em que este processo é feito de modo autónomo, pode resultar nelas uma rutura e esgotamento dos recursos. A título de exemplo, um servidor que disponibilize um certo serviço que em condições de normal funcionamento utilize uma pequena fração dos recursos computacionais disponibilizados, na eventualidade de um ataque, este pode aumentar significativamente, podendo até atingir o valor máximo atribuído à máquina. Com este exemplo em mente, no caso de um ataque do tipo DDOS a um *datacenter* inteiro que esteja em modo *on-demand* pode significar que, toda a infraestrutura do *datacenter* use cem porcento dos recursos computacionais, o que potencialmente resulta em comportamentos inesperados pela plataforma de virtualização de *cloud*, entre outros.

Por outro lado, a nível de custo existem duas grandes preocupações. Em primeiro lugar o facto de ser difícil efetuar uma previsão de gastos estimados para um período de tempo futuro. Existe a possibilidade de, recorrendo a dados de gastos passados para uma determinada máquina, efetuar uma previsão mais ou menos aproximada da realidade, todavia não é possível garantir o valor calculado. Em segundo, e referente ao item abordado no parágrafo anterior, em caso de ataques que perturbem a utilização normal média dos recursos computacionais de uma máquina, poderá traduzir-se num custo elevado para o subscritor, que pode só descobrir o ataque horas ou dias depois do mesmo inicializar.

2.6.3 *Tokens*

O conceito de *tokens* ou créditos é especialmente importante para grandes instituições que possuem diferentes equipas de desenvolvimento. Apesar de não ser considerado um método de *leasing*, como sugere o presente subcapítulo, os *tokens* apresentam-se como uma moeda virtual usada na plataforma de *Cloud*. Para quem desenvolve aplicações, o valor monetário real gasto por uma máquina virtual ou projeto é irrelevante e alheio aos mesmos, pelo que a utilização de *tokens* apresenta-se como um método viável e simples de manter controlo sobre os gastos gerais da infraestrutura computacional de uma instituição, dando, porém, liberdade controlada aos utilizadores para possuir recursos computacionais.

Tokens podem ser equiparados a uma moeda de troca sem valor de circulação, sendo assim possível atribuir valores aos diferentes recursos em forma de *tokens*. Para tal, é também necessário atribuir por projeto ou utilizador, uma carteira onde seja possível carregar *tokens*. É também possível criar carteiras de "diferentes" *tokens*, ou

seja, com *tokens* para diferentes recursos, permitindo aos gestores da plataforma *Cloud* ter uma visão mais clara e concreta da distribuição de recursos.

A gestão dos *tokens* é coordenada pela organização responsável pela plataforma *Cloud*, podendo a qualquer momento alterar o valor de cada recurso.

2.7 Plataformas IaaS

Apesar do conceito por detrás das conceções como *Cloud Computing* ou IaaS terem um lugar e um juízo de valores bem fundamentado a nível teórico, a transação para um contexto tangível por vezes não é tão linear como aparenta, levantando-se um conjunto de pequenos (ou não) pormenores, que no desenvolvimento da plataforma *IaaS* acabam por criar empasses ou limitações não calculadas nos contextos teóricos. Ademais, e com os olhos postos nos ambientes empresariais, encarece às soluções um aspeto *enterprise* que, inevitavelmente, introduz uma série de pormenores no amplo espetro dos processos que estão associados a estes ambientes dinâmicos.

Existe, assim, uma necessidade fulcral de responder aos objetivos primordiais propostos pela ideia de IaaS, como criar máquinas/ambientes de forma autónoma e transparente para o subscritor, encarecendo à volta dessa máxima aspetos como segurança, automação de processos inerentes ao ciclo básico da plataforma, gestão de recursos, gestão monetária, monitorização da plataforma, entre muitos mais. Assim, e com os principais pilares da ideologia da solução *IaaS* em mente é possível iniciar uma revisão compreensiva das diversas soluções *on-premise* que atualmente o mercado disponibiliza, considerando os aspetos positivos e negativos de cada solução, assim como os valores monetários adstritos a questões de aquisição, manutenção, suporte e uso das plataformas (*open-source* vs. *closed-source*). É de salientar também que o processo de comparação direta é, por vezes, falível e ineficiente, tanto pela disparidade de funcionalidades ou pelo falta de métricas compatíveis ,e/ou palpáveis, passando a ser necessário uma análise mais passível de opiniões de carater pessoal do autor.

2.7.1 *vRealize* (*VMware*)

Sendo só apresentado com o nome de *vRealize* em 2014 o projeto *vCenter Operations Management Suite* mostra-se inicialmente como um gestor, estando uma camada aplicacional lógica acima, do orquestrador de máquinas virtuais proprietário da *VMware*, o *vCenter*. Ambos, em simbiose, comunicam entre si via interfaces *REST API* presentes em ambas as soluções, possibilitando a rápida e fácil integração entre ambas as soluções, tanto como com plataformas terceiras.

Atualmente a solução *vRealize* divide-se em quatro componentes distintas disponibilizando diferentes tipos de serviços tanto ao subscritor da solução, como aos administradores, possibilitando uma maior visibilidade sobre a infraestrutura dispensando, ao mesmo tempo, a necessidade de um conhecimento aprofundado sobre tanto a solução *vRealize*, o *vCenter* e o histórico de configurações das máquinas virtuais adjacentes à solução.

O componente *Automation* destaca-se pelo seu papel central na implementação do elemento *IaaS* na solução *vRealize*, permitindo ao mesmo tempo ao gestor da solução customizar, implementar e gerir os diferentes aspetos que farão parte do produto final que o subscritor terá acesso. Este divide-se ainda em quatro grandes grupos com funções bem delimitadas, o *Cloud Assembly*, o *Code Stream*, o *Orchestrator* e o *Service Broker*.

O *Cloud Assembly* apresenta-se como o ponto central de gestão de ativos, tanto físicos como lógicos a nível de limites e restrições de uso. Neste, é possível criar diferentes "*Project*", nome dado ao ambiente de ação e de responsabilidade de uma equipa, sendo também possível associar diferentes utilizadores, zonas de *Cloud* (origem dos recursos computacionais) e apresentar *Design*, como ilustrado na Figura 2.4. Como nos *Design* só é possível definir os recursos a serem usados na sua execução, quando é necessário um processo mais específico ou algorítmico, são usadas *Subscriptions*, que interagem com o *Orchestator* (discutido mais à frente neste capítulo) possibilitando um tratamento de dados mais granular, assim como a comunicação e recolha de informação alheia ao *vRealize* em plataformas terceiras a partir de APIs. O *Design* apresenta-se como um conjunto de atributos, formas de ação e questionários que permitem ao subscritor final executar, resultando na apresentação de um *Deployment* com as caraterísticas tanto configuradas pelo utilizador como pelo administrador da solução *Cloud*. Um *Deployment* representa, normalmente, uma máquina virtual, sendo possível ter acesso a aspetos como as características selecionadas anteriormente ou gerados de modo autónomo (como a atribuição de IPs ou seleção do armazenamento físico), assim como controlar ativamente aspetos administrativos da máquina virtual como número de cores de processador, memória RAM ou memória de disco. É também nos *Deployment* possível aceder à consola das máquinas hospedadas em *vCenter*. No caso de necessidade de apresentar opções de ação personalizadas aos *Deployments* é sempre possível utilizar *Resource Actions* personalizadas, que em norma comunicam com a plataforma *Orchestrator* fazendo uso de *scripting* nativo ou desenvolvido por terceiros. Na tentativa de interligar o *vCenter* com o *vRealize* são também introduzidos componentes como *Storage Profile*, *Image Mapping* ou *Network Profile* que permitem fazer a transcrição e o elo dos elementos entre as plataformas. Por último, o *Cloud*

Assembly permite entender a ponte existente entre a plataforma *vRealize* e a *vCenter* e configurá-la de modo que exista uma fusão entre soluções uniforme.

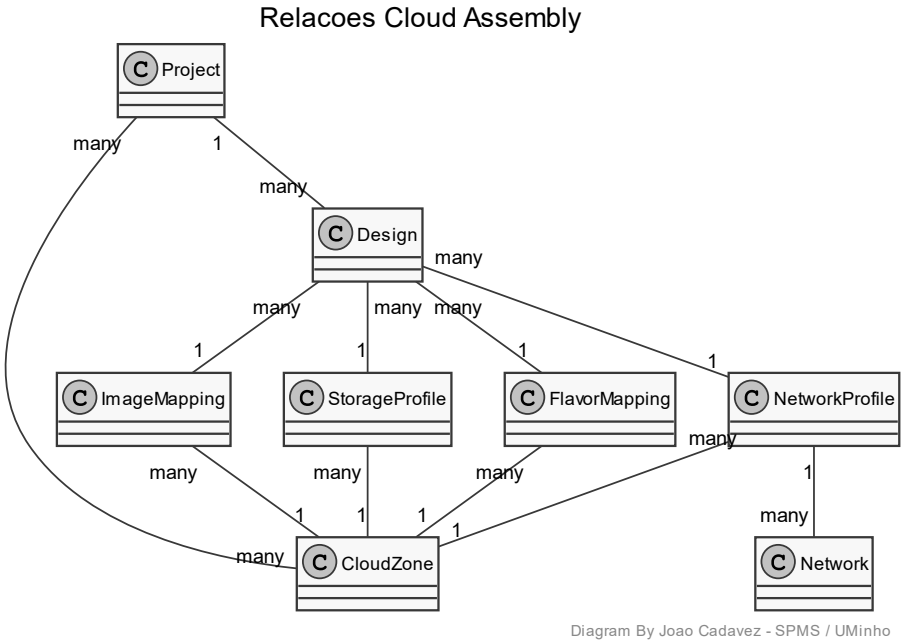


Figura 2.4 · Diagrama de relações *Cloud Assembly – vRealize*

Por outro lado, o componente *Code Stream* direciona-se mais para a programação sequencial, com recurso a caixas de instruções, aproximando a proposta a uma linguagem de programação declarativa [15] do tipo *block programming* [16], tal como ilustrado na Figura 2.5. Este componente diferencia-se especialmente dos restantes da solução *vRealize* pela sua orientação para as técnicas de *continuous integration / continuous delivery* (CI/CD) [17], presente nos processos de desenvolvimento aplicacional, de modo a facilitar a entrega rápida e eficaz de *software* com o mínimo de *overhead* possível. Este processo utiliza maioritariamente *containers* [18] (como *Docker*[7] ou *Kubernetes*[8]), que pelo facto de utilizarem o *kernel* do sistema operativo do hospedeiro, é possível criar, compilar, testar, e lançar uma nova versão de um componente aplicacional de modo rápido e principalmente seguro, uma vez que a publicação não acontecerá caso alguma das etapas falhe.

[7] docker.com
[8] kubernetes.io

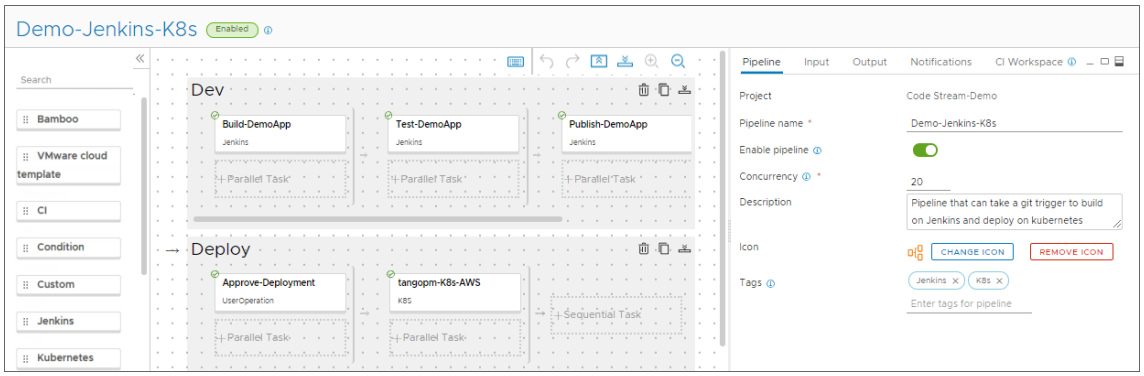


Figura 2.5 - Exemplo de fluxo no *Code Stream – vRealize* [19]

A componente *Service Broker* é o *frontend* utilizado pelo subscritor. Como administrador da solução é possível configurar parâmetros como os grupos de utilizadores que têm acesso à solução, quais os itens que cada grupo de utilizadores tem no seu catálogo ou aprovar os requerimentos dos subscritores (caso tenham). Com o papel de subscritor é a única interface a que tem acesso na solução, podendo efetuar toda a gestão, criação e manutenção dos seus recursos pela mesma. É de grande importância a correta configuração da interface *Service Broker*, de modo a evitar acessos indevidos na solução, assim como a boa gestão administrativa dos recursos computacionais.

Por último, o *Orchestrator* é o componente de *scripting* da solução. Este serve como ponte entre a solução *vRelize* e plataformas terceiras a partir de várias interfaces. Assim, é possível invocar *scripts* (ou *workflow* na terminologia da solução) do *Orchestrator* nos restantes componentes da solução e, utilizando diferentes linguagens como *javascript*, *python* ou *powershell*, é possível pré-processar a informação, fazer pedidos externos e pós-processar a informação conforme for necessário. A recolha de informação externa dá-se maioritariamente a partir de pedidos do tipo REST API, no entanto outras opções como acesso a bases de dados SQL, pedidos *PowerShell* ou SNMP são algumas das opções nativas disponíveis pela plataforma. Ademais, a plataforma também oferece opções de controlo de versões a partir de *Git*, assim como agendamento de execuções, *logs*, *debug* de *workflow*, entre outros.

Com o intuito de oferecer uma solução mais completa e complexa, como analíticas e recolha de *logs* por partes dos vários sistemas que suportam a solução *vRealize* (tanto nativos como terceiros), a solução ainda oferece a plataforma *Log Insight* que é responsável pela recolha de *logs* do tipo *syslog*, efetuando a correlação dos mesmos de modo a gerar alarmísticas complexas e a manter histórico do comportamento da solução, como também das ações tomadas pelos utilizadores dos diferentes serviços. A gestão macro dos recursos e tabulação de preços por recursos

computacional é efetuada no componente *Operations*, sendo também possível testar o *hardware* para falhas ou alertas de configurações, assim como criar gráficos de comportamento e evolução dos diferentes sistemas inerentes à solução. Por último, o componente *Lifecycle Manager* oferece a possibilidade de gestão da plataforma a nível de atualizações tanto da mesma, bem como dos *hosts* físicos e da plataforma de *hypervisor*.

Em suma a solução *vRealize* da *VMware* apresenta-se com uma segmentação de responsabilidades e organização de recursos muito bem definidos, ao mesmo tempo que disponibiliza uma panóplia de recursos necessários e convenientes à fácil gestão da plataforma *Cloud*, facilitando ao subscritor a navegação e promovendo a fácil adaptação à plataforma e ao seu modo de funcionamento como um todo. Para o administrador da solução, a plataforma pode apresentar uma curva de aprendizagem acentuada especialmente para quem nunca contactou com soluções *Cloud* ou com outras soluções da *VMware*, no entanto, bastante intuitiva com balões de ajuda ou explicação em grande parte das operações. O desenvolvimento de soluções ou casos de uso personalizados é possível, no entanto tentar alterar o fluxo desenhado pela solução pode implicar bastante trabalho de tentativa e erro, uma vez que a *VMware* peca na documentação apresentada aos administradores, pela falta de clareza de algumas relações entre componentes ou pela inexistência de menções em algumas operações. É também importante ter em mente que a solução só é suportada em ambientes *VMware*, apresentando *plug-ins* para aplicações terceiras aos quais o suporte pode ser questionável, ou seja, a integração com aplicações terceiras à família *VMware* não possui suporte, podendo ser um ponto de caracter importante na aquisição da solução. De notar que a solução é *closed-source* possuindo planos de aquisição, instalação, gestão/suporte e de licenciamento.

2.7.2 *Nutanix Cloud Manager*

Apresentam-se ao público como um *software* integrador de várias plataformas *Cloud* e de *hypervisors on-premise*, com a premissa de que se tem verificado um movimento, por parte das empresas, para distribuir os seus serviços por diferentes plataformas de computação, na procura de diminuir o risco de perdas de serviços, enquanto garantem a capacidade de crescimento instantâneo caso se justifique. Deste modo, a solução *Nutanix Cloud Manager* promete, de uma forma central e uniforme, oferecer um controlo sobre os diferentes serviços de computação, ao mesmo tempo que dispõe de ferramentas e mecanismos genéricos que permitem a personalização de ações perante a realidade de cada cliente. Deste modo, garante uma elevada versatilidade dos recursos e simplificação das implementações. Em conjunção, são adicionados

mecanismos de contabilização de custos que permite a visibilidade sobre os investimentos efetuados enquanto recomenda a otimização de custos pela adaptação dos recursos alocados a cada serviço/máquina, serviços de segurança e de gestão de serviços/máquinas [20].

A integridade da plataforma surge da simbiose entre diferentes plataformas já existentes na qual fazia sentido trabalharem em si. Assim, e como ilustrado na Figura 2.6, o módulo *AIOps* é responsável por possibilitar o automatismo associado à ferramenta baseado em alertas e eventos capturados a partir de mecanismos de *low-code*. É assim idealizado o conceito de criação de ações que possam corrigir erros encontrados na plataforma, espoletando ações de autorreparo ou notificação dos intervenientes [21].

Por outro lado, o módulo *Self Service IT & Governance* possibilita a integração com as diferentes entidades provedoras de recursos computacionais, aglomerando o controlo das máquinas e serviços que estão distribuídos entre os mesmos. Assim, são disponibilizados *templates* de criação de máquinas e/ou serviços dependendo das necessidades impostas, que, criando uma abstração lógica, possibilitam o controlo dos recursos, como, por exemplo, reiniciar, ligar, desligar ou aumentar recursos computacionais. A abstração e convergência para um modelo único de trabalho propõe uma melhoria na fluidez de criação e manutenção dos ambientes, ao mesmo tempo que afasta o administrador de configurações complexas e particulares de cada plataforma [22].

O módulo *Nutanix Cost Governance* é o ponto de controlo dos recursos usados e da despesa espectável para a infraestrutura. Com a integração dos diferentes módulos, é capaz de calcular valores espectáveis para a criação de uma nova máquina, ou a atualização dos recursos computacionais, tendo em conta os valores praticados em cada plataforma de *Cloud* ou com base nas configurações que o utilizador introduz sobre os seus recursos físicos, como preços de manutenção de equipamento, valores da equipa, gastos elétricos, entre muitos outros parâmetros, que reunidos permitem calcular o valor por recurso. Deste modo, com a ajuda da análise do consumo de recursos de cada ambiente implementados, é possível efetuar uma análise compreensiva para sugerir a adaptação dos recursos, permitindo a otimização dos custos, ao mesmo tempo que apresenta, contemplado num único local, os valores cobrados pelo ambiente [23].

Por último, e como a segurança necessita sempre de andar de mão dada com as soluções *Cloud*, a utilização do módulo *Nutanix Security Compliance* ajuda a controlar eventos de intrusão e de propagação de *malware* entre máquinas. Com recurso a métodos autónomos, é garantida análise de rede, assim como a nível aplicacional, efetuando a defesa com a implementação de políticas *Zero Trust*. Em culminar é

possível a partir de um quadro de análise a validação e monitorização da segurança e das ameaças inerentes tanto à infraestrutura, como aos serviços em ambientes de desenvolvimento [22].

Figura 2.6 - Representação dos elementos integradores do NCM [22]

Em síntese, a solução apresenta pontos fortes nas possibilidades de integração com diferentes soluções *Cloud* e soluções *on-premise*, bem como na disponibilização de mecanismos de automação e de *templates* de máquinas ou aplicações para garantir rapidez nas implementações. O mecanismo de controlo de custos apresenta particular utilidade para impedir o crescimento anormal oculto dos custos, e a integração com o módulo de segurança disponibiliza visibilidade e confiança, facilitando a monitorização e alertando para comportamentos anómalos. No entanto, a solução apresenta, em comparação com as restantes, uma baixa popularidade e fraca comunidade de utilizadores, podendo ser considerado o principal ponto fraco.

2.7.3 *Red Hat Cloud Suite*

Com o rápido crescimento da concorrência nas soluções *Cloud*, a *Red Hat* sentiu a necessidade de apostar também na área apresentando a sua solução *Red Hat Cloud Suite* no final de 2012 [24], possuindo atualmente no mercado um posicionamento forte perante os restantes. Para tal, a *Red Hat* apostou na utilização de várias plataformas *open-source* da própria empresa, criando um ecossistema *Cloud* capaz de satisfazer os requisitos dos clientes [25].

De modo a dar resposta às exigências do mercado, o *Red Hat Cloud Suite* faz uso de quatro plataformas (Figura 2.7) desenvolvidas e geridas pela própria companhia, cada uma com o propósito de completar e enriquecer a solução como um todo, agregando mais o *Red Hat Cloud Forms,* responsável por gerir e apresentar ao subscritor final o ambiente *Cloud* no qual efetua as suas interações [25].

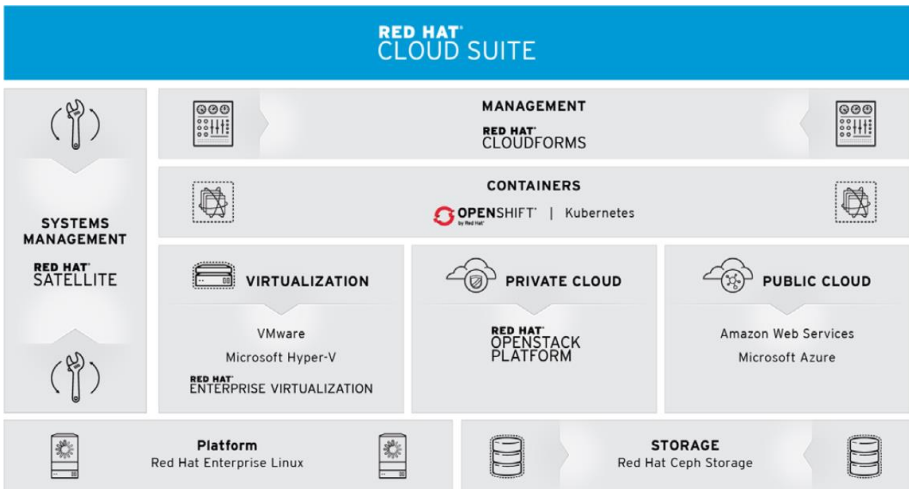


Figura 2.7 - Camadas e organização lógica da plataforma *Red Hat Cloud Suite* [25]

Em primeiro lugar, o uso da plataforma *open-source Red Hat OpenStack Plataform* permite aprovisionar à solução a intercomunicação e gestão de recursos computacionais necessários para elaborar uma solução *Cloud*. Ademais, a utilização do *OpenStack* permite a integração da solução *Cloud on-premise* com *Clouds* públicas, possibilitando a criação de *Clouds* híbridas, ideais para clientes com requisitos mais restritivos ou para quem já possui soluções em *Cloud* públicas e pretende unificar tudo numa só plataforma. Para além disso, o *OpenStack* é também responsável pela conectividade com os servidores físicos (*baremetal*) e controlo dos mesmos, estando estes na base das soluções *Cloud on-premise*, permitindo a virtualização dos recursos computacionais como a RAM e os CPUs, mas também a integração da rede com os recursos virtualizados e a integração com soluções terceiras de armazenamento. *OpenStack* é desenhado sobre os padrões de indústria, facilitando a interação com a solução, como, por exemplo, aceder e ser acedido por terceiros via interfaces API [26].

Com a rápida adoção dos serviços hospedados em *containers* a aumentar, a *Red Hat* sentiu também a necessidade de juntar à sua panóplia de ferramentas mais um dos seus softwares *open-source*, o *Red Hat OpenShift*. Nesta, é possível de forma estratégica garantir uma gestão coerente e eficaz dos recursos dedicados aos *containers*, assim como proporcionar versatilidade numa área em crescimento considerável. Também é possível o aprovisionamento de serviços altamente requisitados, como bases de dados (*Mongo DB*[9], *Oracle SQL*[10], etc) ou serviços generalizados (*Jenkins*[11], *Ansible Tower*[12],

[9] mongodb.com

[10] docs.oracle.com/cd/B19306_01/server.102/b14200/intro.htm

[11] jenkins.io

etc) possuir *templates* previamente determinados, muitos deles nativos, sendo facilmente atualizados e acima de tudo proporcionando um aprovisionamento mais rápido, consistente e coerente sobre técnicas clássicas que envolvem interação humana em passos sequenciais (suscetível a erro e incoerências). A plataforma permite também encapsular máquinas virtuais em *containers* de modo a possibilitar a migração e posterior facilidade de gestão a aplicações *legacy* que de outro modo não poderiam ser tratadas como *containers*. Ademais, a plataforma, como se integra num dos componentes da solução *Red Hat Cloud*, é também especialmente adaptada para apresentar o melhor desempenho perante a solução, assim como a sua melhor integração e interação [27].

A utilização da plataforma *Red Hat Virtualization*, pertencente ao *pack Red Hat Cloud*, permite oferecer a plataforma de virtualização necessária para suportar e gerir as máquinas virtuais da solução *Cloud*. Baseado na solução *open-source* KVM [28], permite a virtualização de ambientes *Windows* e *Linux* sobre um único *hardware* com o intuito de poupar recursos e energia, mas também de forma a fornecer o dinamismo necessário na criação e gestão de máquinas em ambientes *Cloud*, enquanto garante a segurança e estabilidade exigida pela criticidade da solução. Deste modo, a solução proporciona a base progenitora para o sucesso da solução, representando a plataforma *hypervisor* adotada pela *Red Hat Cloud* [29].

Por fim, em modo de aglomeração das plataformas e componentes anteriormente a referenciados, a plataforma *Red Hat Satellite* promete oferecer a ponte entre todas as plataformas *Red Hat*, assentando-se no meio das mesmas, como ilustrado na Figura 2.8, de modo a promover a boa gestão e interligação da infraestrutura, ao mesmo tempo que procura promover a eficiência da solução e sua segurança. Para tal são implementados sistemas que possibilitam a uniformização das operações, promovendo a utilização de padrões de conformidade, enquanto proporciona a otimização do tempo de configuração e gestão por parte dos gestores da solução. Da mesma forma, são tidos em conta aspetos como o cálculo da escalabilidade do sistema e do *hardware* alocado à solução associando-se também à redução de custos afiliados tanto à monitorização da infraestrutura física como lógica e à deteção de módulos não otimizados, proporcionando assim sugestões de melhoria ativa e/ou passiva [30].

[12] docs.ansible.com/ansible/2.5/reference_appendices/tower.html

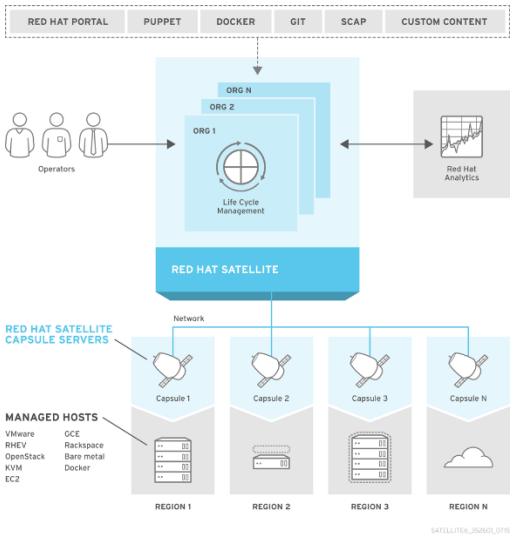


Figura 2.8 - Representação Lógica da Plataforma *Red Hat Satellite* [31]

2.7.4 *OpenStack*

O *OpenStack* apresenta-se como uma das soluções mais promissoras e ambiciosas de *Cloud* privada apresentada ao público. Iniciando a sua vida no início de 2010, o *OpenStack* começou como um artigo de nome "*Cloud Computing Fabric Controller*" pela necessidade de código de gestão para servidores em infraestrutura *Cloud* por parte de duas entidades, que acabaram por unir esforços e em julho do mesmo ano apresentar oficialmente a ideia ao público.

Com uma proposta sagaz, o *OpenStack* promete "produzir uma plataforma de computação em nuvem de código aberto onipresente que seja fácil de usar, simples de implementar, interoperável entre implantações, funcione bem em todas as escalas e atenda às necessidades de usuários e operadores de nuvens públicas e privadas", e tudo com recurso exclusivamente a *software open-source* [26].

Como verificado nas soluções concorrentes, o *OpenStack* distribui a responsabilidade das diferentes ações por diferentes módulos, como ilustrado na Figura 2.9, delegando a cada um deles uma função especifica e bem definida no contexto da plataforma [32].

Durante o decorrer da presente secção serão abordados os módulos NOVA, SWIFT e GLANCE por se apresentarem como cérebro da operação na solução *OpenStack*.

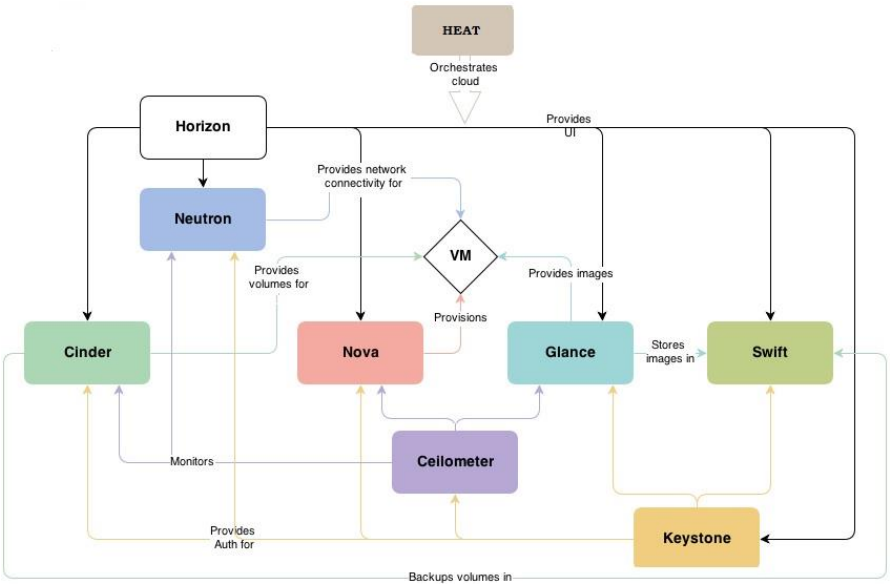


Figura 2.9 - Representação dos componentes da solução *openStack* [33]

O módulo NOVA apresenta um papel de gestão de *Cloud* na solução *OpenStack*, sendo da sua responsabilidade o controlo e monitorização das máquinas virtuais da *Cloud*, dos recursos computacionais disponíveis e em uso e dos recursos de rede, culminando num só sítio, e de uma forma abrangente, o estado de cada componente essencial para o bom funcionamento da solução *Cloud* como plataforma de gestão e como provedor de serviços [34].

Tal como representado na Figura 2.10, o módulo NOVA por si só é dividido em vários *deamons*[13], responsáveis por processos como disponibilizar uma interface API do módulo NOVA, interação com as diferentes soluções integrantes do *OpenStack*, interação com as máquinas virtuais, interação com a rede e reserva de endereços IPs para as máquinas virtuais, controlo das unidades de armazenamento, entre muitos mais processo inerentes ao bom e correto funcionamento do módulo como um todo [34].

[13] Processo computacional executado em plano de fundo

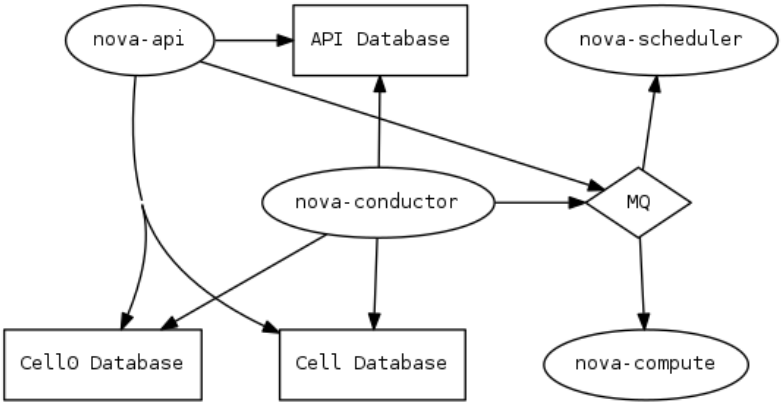


Figura 2.10 - Representação lógica da NOVA [34]

O módulo SWIFT é responsável pelo armazenamento em larga escala de informação de inúmeros objetos inerentes à solução *OpenStack*. A sua função passa por disponibilizar redundância de dados, armazenamento seguro da informação, assim como possibilidade de armazenamento de grandes ficheiros e *streaming* de vídeo e áudio sem esquecer da escalabilidade necessária para uma solução desta natureza [35].

Tal como os restantes módulos, o SWIFT recorre ao uso de diferentes *deamons* para segmentar as suas responsabilidades, tendo em mente a redundância e potencial de crescimento, como ilustrado na Figura 2.11. O SWIFT *Object Server*, é responsável por gerir o armazenamento e os processos que a si estão associados. Por outro lado, o SWIFT *Proxy Server*, é usado como ponte entre o *Object Server* e o balanceador e a API do módulo. O SWIFT *Container Server* e o *Account Server*, têm como principal função gerir as listagens de objetos presentes nos repositórios, assim como gerir os acessos aos mesmos [35].

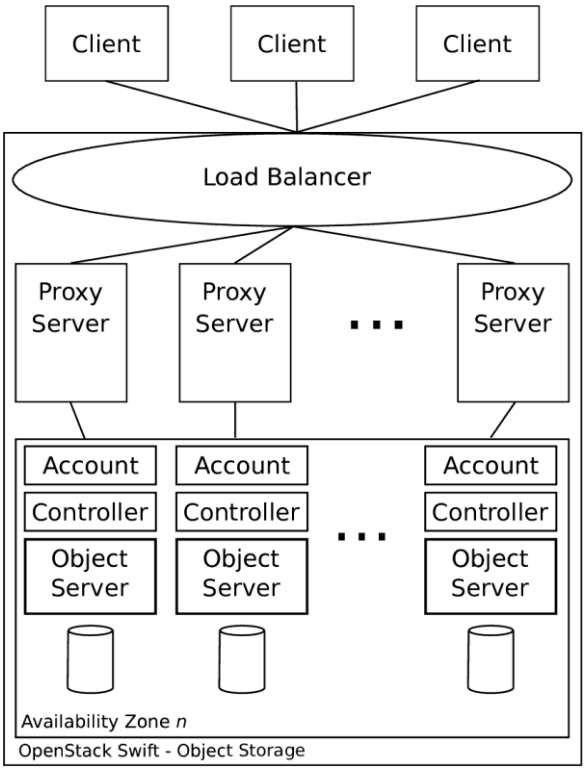


Figura 2.11 - Representação lógica SWIFT [35]

O módulo GLANCE, apesar de aparentemente disponibilizar uma função muito simples, representa uma enorme flexibilidade no que toca à interação do subscritor com a solução *Cloud*. Assim, este oferece a possibilidade de carregar para a plataforma imagens ISO[14] à vontade do subscritor. Estas podem ser originais de uma versão especifica requisitada pelo mesmo, ou uma versão personalizada, oferecendo grande flexibilidade a quem gere uma conta como subscritor na plataforma [36].

2.8 Métodos de Automação

A automação, como já discutida anteriormente neste documento, procede de uma definição vasta e muito abrangente, levando a que seja necessário avaliar em aspetos como a robustez, a complexidade, a autonomia e a prioridade o "nível de automação" de um processo ou função autónoma. Em tese, um processo autónomo, para reter as suas propriedades de autónomo, necessita de um estímulo externo e diferente do seu estado de repouso normal, como por exemplo o espoletar de uma ação

[14] Imagem de CD ou DVD de ficheiros (ISO 9660)

por um utilizador externo, uma ação programada para um dado dia ou hora ou um culminar de diferentes aspetos que espontânea inicialize uma cadeia de processo(s).

No entanto, para permitir a reprodução e ação dos processo autónomos computacionais, é necessário a utilização de interfaces de comunicação entre o Humano e a máquina. Linguagens como o *Assembly* foram fulcrais para o início da interação Humano/máquina, sendo, no entanto, muito mais próximas da máquina do que do Humano, ou seja, são mais dificilmente interpretadas por Humanos. Com a evolução da programação e a sua popularização, a existência destas linguagens deixa de ter sentido, a um nível generalizado de aplicações práticas, o uso de linguagens de baixo nível, como o exemplo anterior, o *Assembly*, começando a ganhar grande popularidade linguagens consideradas de mais alto nível, começando pela linguagem *C* e evoluindo para *Java*, *Python*, *C#* entre outras, que se apresentam muito mais próximas da linguagem Humana com sintaxes fáceis de relacionar, processo de procura de erros simples (*Debug*), facilitado ainda pelo aparecimento de IDEs[15] (*Integrated Development Environment*) que permitem o auxilio com inúmeras ferramentas que facilitam ao programador o rápido desenvolvimento, a sugestão de utilização de vários padrões arquiteturais e a sugestão e explicação de diversos métodos nativos à linguagem ou dos diferentes pacotes de código (*framework*[16]) desenvolvidos pela comunidade [37], [38].

2.8.1 *Block Based Coding*

O *Block Based Coding* (BBC) é um conceito tornado popular na última década, inclusive com fortes apoios de grandes empresas como o *Google* e meios de educação de diferentes países, uma vez que é considerado um dos mais simples e intuitivos meios de introdução à programação declarativa, tendo assim um forte papel nas novas competências digitais básicas para os mais novos [39]. O BBC é um tipo de programação que representa tipos de ações e funções, normalmente, em formas geométricas que possuem uma interface única, ou seja, somente as outras formas que respeitem a interface é que podem ser usadas imediatamente antes ou depois da forma seleciona, como demonstrado na Figura 2.12.

[15] Interfaces de desenvolvimento de código que facilitam os processos rotineiros de correr, compilar ou debug que têm o intuito facilitar o processo de programação

[16] Conjunto de métodos de código comuns a diferentes projetos, servindo o intuito de camuflar e simplificar o desenvolvimento de software, especialmente em processos comuns.

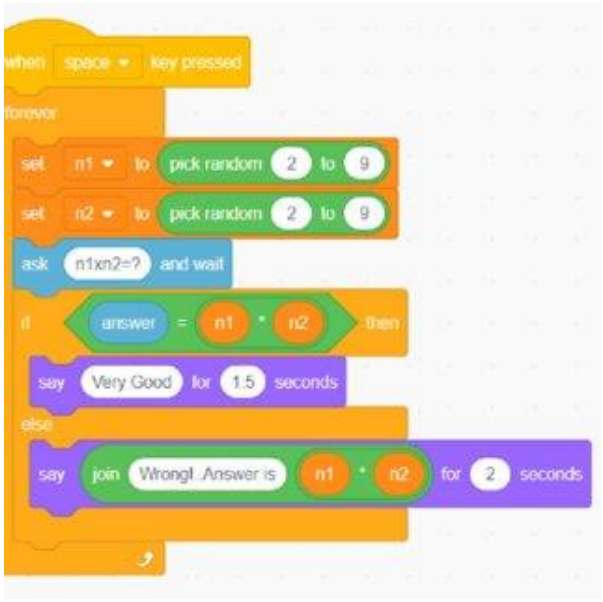


Figura 2.12 - Excerto de código de *scratch* [40]

Apesar de considerado rudimentar e introdutório, a utilização de paradigmas de programação BBC introduz os conceitos chaves da programação como os *loop*, *while*, *if*, ou os *else* possibilitando, ainda que de forma limitada, facilitar processo considerados repetitivos para o Ser Humano, podendo-se assim considerar um tipo ou forma de automação de processos [40].

2.8.2 *Low-Code*

A utilização de linguagens *Low-Code*, nos últimos anos, tem sido alvo de uma grande procura por parte de quem necessita de um desenvolvimento aplicacional rápido e pouco personalizado em relação a aplicações padrão no mercado. Desenvolvimento de, por exemplo, lojas de venda *online* ou portais de afiliação de clientes são, de uma forma geral dentro da sua categoria individual, plataformas que pouco diferenciam de solução para solução. O objetivo principal do *Low-Code* é suprimir a elevada procura de plataformas genéricas, possibilitando ao desenvolvedor utilizar grandes peças de código genérico já desenvolvido para facilitar a criação destas mesmas plataformas genéricas, poupando recursos humanos e horas de trabalho, evitando assim a necessidade de recriar código. Deste modo, casos de uso como possibilitar *login* de um utilizador, ou adicionar um cesto de compras à conta do utilizador ou até mesmo notificar o utilizador de um evento na plataforma, pode ser efetuado com recursos a pouco ou nenhum código assentando principalmente numa plataforma gráfica de *drag-and-drop* de itens representativos das ações desejadas e da criação de fluxos de trabalho a partir de fluxogramas, como ilustrado na Figura 2.13 e na Figura 2.14.

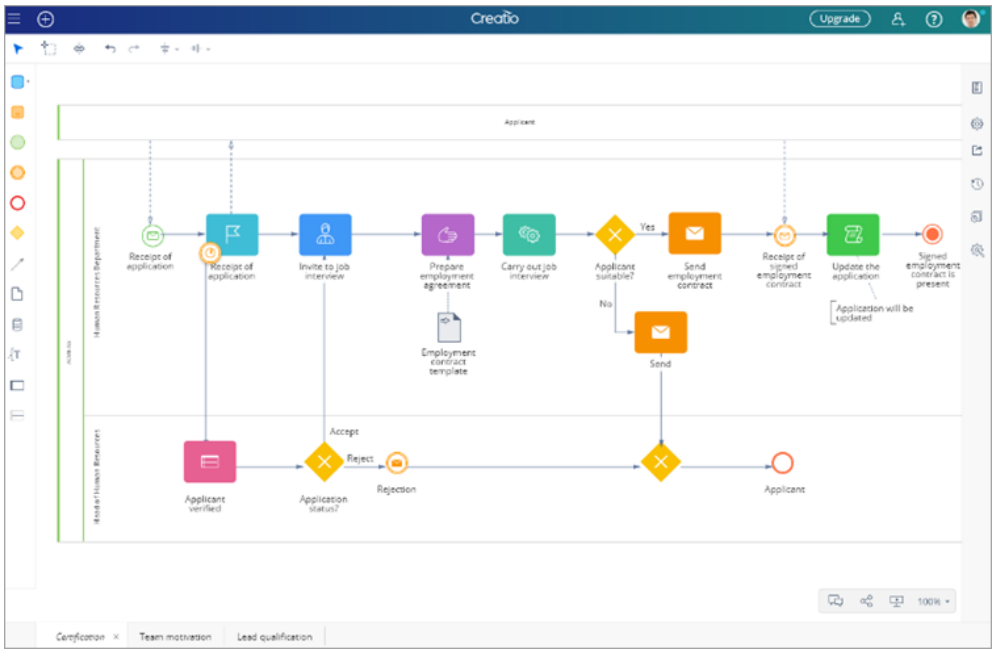


Figura 2.13 - Imagem representativa de um fluxograma em plataforma *Low-Code* [41]

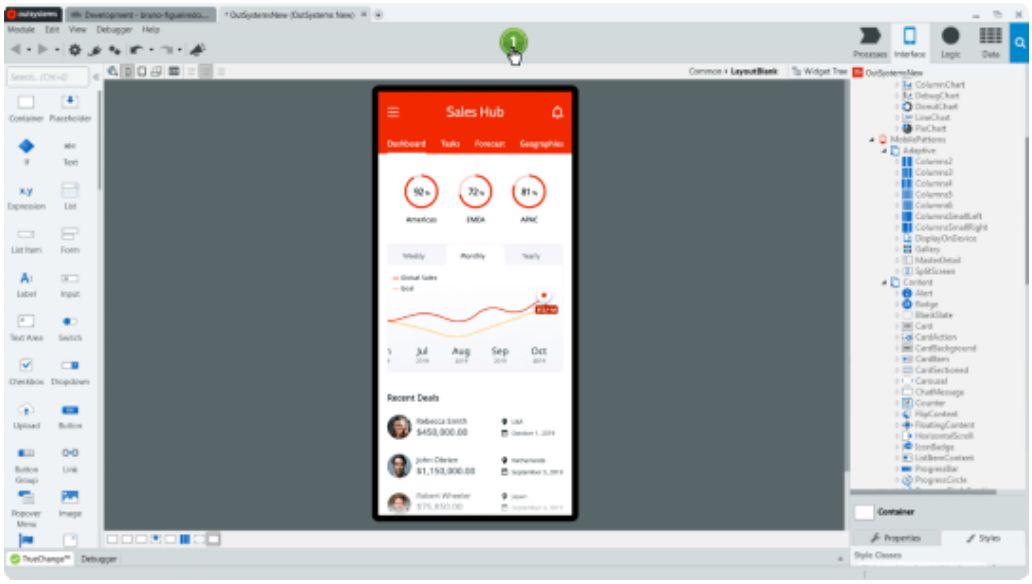


Figura 2.14 - Imagem representativa de uma plataforma de *Low-Code* [42]

2.8.3 *Python*

Ganhando terreno nos últimos anos como umas das principais linguagens de *scripting* usadas em *Dev-Ops*[17] nas áreas de redes, sistemas e *Cloud*, apresentando uma vasta versatilidade nas diferentes áreas da programação, assim como a fácil interação e integração com diferentes serviços terceiros. O *Python* é considerado uma linguagem que programação de alto nível, que pode ser utilizada para grandes projetos, mas é especialmente útil e utilizada para pequenos e rápidos *scripts*, tanto pela sua sintaxe simples, como pelo processo claro de instalação de *framework* e bibliotecas e pelo vasto leque de sistemas operativos suportados pela linguagem [43].

2.8.4 *Javascript*

Derivado da linguagem de programação *Java*, o *Javascript* ganhou um forte reconhecimento no desenvolvimento de *website* no tripleto HTML, CSS e *Javascript*, ocupando o lugar de *scripting*, de recolha e execução de ações. É especialmente usado em ações do lado do navegador do cliente (*client-based*), no entanto já possui uma vertente focada no desenvolvimento de *backend* (*server-based*) conhecida como *NodeJS*. A forte comunidade, a elevada parecença com a sua linguagem mãe, o *Java*, possibilitou a transação de muitos programadores que já conheciam a sintaxe *Java* e o seu modo de operação para o ambiente *Javascript*, que em junção com a curva de aprendizagem suave, provocou um grande crescimento na popularidade desta linguagem, apresentando-se no top dez das linguagens mais usadas durantes os últimos anos [44].

2.9 Interfaces de Automação

O processo de automação passa por vezes pela integração e interação entre a máquina que corre o código dos processos de automação e serviços externos das mais diversas naturezas e motivações. Com o aumento da necessidade de centralização da gestão dos processos referentes a diferentes plataformas ou ferramentas, mais do que nunca existe o carecimento da interação ubíqua e de simbiose entre os diferentes programas, de modo a proporcionar ao utilizador final, e até mesmo aos gestores das plataformas, uma experiência fluida e imersiva com o intuito de melhorar os processos, diminuindo recursos humanos e reforçando a coerência de configurações e dos processos iterativos associados aos mesmos.

[17] O culminar dos processos de desenvolvimento e a operação em um processo sequencial e sem disrupção

Existem, assim, uma panóplia de modelos de interfaces de interação que permitem a troca segura e padronizada de informação necessário ao bom funcionamento dos processos de automação.

2.9.1 SOAP

Com as primeiras aparições em meados de 1998, o *Simple Object Access Protocol* (SOAP) faz uso de uma camada aplicacional já bem conhecida na altura do seu lançamento o HTTP[18], para enviar informação em formato XML[19] sobre padrões de ação pré-definidos. O pacote SOAP divide-se me duas partes o corpo, onde existe a informação em XML e o cabeçalho onde se encontram especificados aspetos importantes sobre o tipo de ação, o conteúdo, entre outros, estando tanto o cabeçalho como o corpo SOAP encapsulados num pacote HTTP. Como o SOAP depende intrinsecamente do HTTP, a sua segurança depende também da segurança do pedido HTTP, ou seja, se o pedido HTTP for encriptado, ou HTTPS[20], o pacote SOAP pode-se considerar seguro, pois encontra-se no corpo do pedido HTTPS [45].

2.9.2 REST API

Em semelhança à interface de troca de informação SOAP, a interface REST API (RESTful API) faz uso da camada aplicacional HTTP para transportar a informação e o cabeçalho HTTP para informar parâmetros fulcrais, como o tipo de pedido (GET, POST, DELETE, etc.) ou o objeto requerido (pelo endereço URL), possibilitando a apresentação de informação em diferentes formatos, sendo o mais usual o JSON (*Javascript Object Notation*), que facilita a interpretação por humanos, sendo no entanto fácil de delimitar pela máquina. A utilização de pedidos REST seguros é também possível, bastando garantir a segurança da camada HTTP com a sua encriptação [46].

2.9.3 *Telnet*/SSH

O uso de protocolos como o *telnet*, ou a sua versão segura SSH, permite aceder à consola das máquinas possibilitando a gestão e interação com o seu leque de funcionalidades disponibilizado.

[18] *Hypertext Transfer Protocol*

[19] *Extensible Markup Language*

[20] *Hypertext Transfer Protocol Secure*

Apesar de não ser considerado de todo um método ortodoxo por quebrar o princípio da responsabilidade única, a recolha de dados via *telnet*/SSH é por vezes a única opção em equipamentos e máquinas, que por um lado não disponibilizam outras interfaces mais oportunas para a recolha dos dados, ou não disponibilizam os dados necessários nessas mesmas interfaces. A utilização de bibliotecas terceiras, como por exemplo o *Paramiko*[21] em *Python*, possibilitam entrar via SSH na máquina e recolher dados para poderem mais tarde serem processados. No entanto existem algumas preocupações relacionadas com esta abordagem, como o armazenamento de palavras-passe nos *scripts* e o constante estabelecimento de sessões com o servidor podendo provocar rutura de serviços.

2.9.4 SNMP

Introduzido em 1988, o *Simple Network Management Protocol* (SNMP), foi introduzido com o intuito de facilitar a gestão e monitorizar os equipamentos de rede de forma simples e remota. Para tal o SNMP faz uso de uma árvore de diferentes itens, sendo possível mapear estes valores a partir das MIBs [47].

Com a generalização da interface SNMP, outros dispositivos e sistemas operativos começaram a adotar este método acabando por criar elementos na árvore MIB pré-definida, possuindo atualmente um valor identificador da companhia ou instituição [48].

Assim, é possível por uma plataforma de recolha, ou também conhecida como *manager* efetuar pedidos aos dispositivos a monitorizar, conhecidos com o *agents*, de uma OID[22] específico. O *agent* ao receber a informação, procede à recolha do elemento solicitada e responde ao *manager* com a mesma, completando o processo de requisição de informação, como é possível verificar na Figura 2.15.

[21] paramiko.org

[22] SNMP Object Identifier

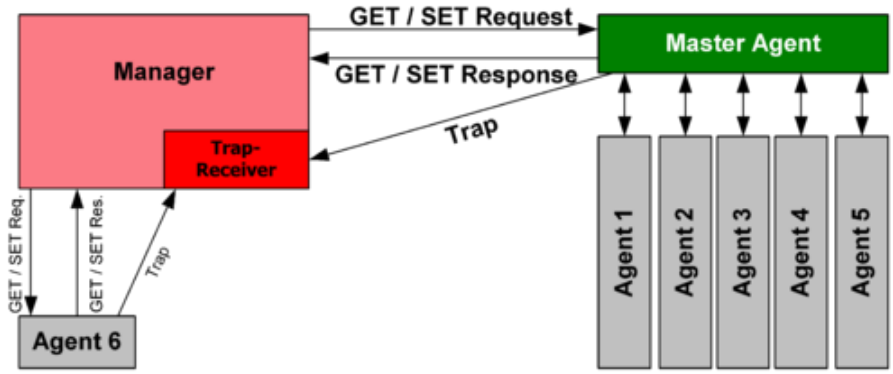


Figura 2.15 - Representação da iteração das interfaces SNMP [49]

2.9.5 Comunicação com Base de Dados

Por vezes a recolha de dados passa por verificar o passado e o que já foi processado pelo próprio sistema ou por um terceiro. Assim, a utilização de bases de dados possibilita o armazenamento de informação que pode ser consultada a qualquer momento, sendo possível pesquisar elementos específicos ou momentos temporais.

A integração de bases de dados com interfaces de automação passa, na sua maioria, por implementar uma biblioteca que efetua o processo de comunicação e estabelecimento de sessão com a mesma, que normalmente ocorre utilizando um protocolo de comunicação pré-definido pela solução de base de dados via rede. A partir desse momento é possível efetuar qualquer tipo de pesquisa na base de dados com o recurso a *queries*.

2.10 Trabalhos Relacionados

Em nota de conclusão de capítulo, é importante, juntamente com os conhecimentos consolidados anteriormente, efetuar uma análise aos trabalhos relacionados, com o intuito de perceber que aspetos já foram abordados nesta temática, o modo e as soluções encontradas que possam beneficiar o presente trabalho.

A maior especificidade do trabalho é, normalmente, inversamente proporcional aos trabalhos relacionados já desenvolvidos na temática, pelo que o presente projeto não saiu do padrão espectável. Assim, a pesquisa por trabalhos relacionados passou pelo uso das palavras-chaves previamente definidas para a construção de uma *query*, sendo de seguida usada num motor de buscas académico. A partir dos resultados obtidos, foi possível selecionar os artigos e/ou estudos que melhor contribuem na presente temática abordada.

O artigo *Automated Provisioning of Application in IAAS Cloud using Ansible Configuration Management* [50], procura explicar a elevada procura não só por recursos computacionais, mas sobretudo nos tempos de resposta e a pressão que é colocada sobre os administradores de sistemas para dar vazão aos requisitos cada vez mais complexos e à falta de profissionais na área. O artigo defende que tendo em conta muitas vezes os ambientes de *DevOps* complexos necessários para os projetos são criadas muitas dependências, fazendo que os processo de lançamento de atualizações ou correção de erros se torne mais moroso. A ferramenta de escolha para a automação dos processos foi o *Ansible*[23], que possibilitou a configuração das ações declarativas e sequenciais, oferecendo uma integração simples e direta com a plataforma de *Cloud* selecionada.

Em adenda, o artigo *Guidelines for Building a Private Cloud Infrastructure* [51] faz uma introdução aos conceitos teóricos inerentes à *Cloud*, assim como análise compreensiva sobre que tipo de solução se adapta mais a cada ambiente. Deste modo, durante o documento é exemplificada a implementação de uma *Cloud* privada com recurso a *software Open-Source*, enquanto são discutidos aspetos e pormenores a ter em conta perante a realidade de cada ambiente de implementação de destino. Na nota de conclusão do trabalho, os autores concluíram que o uso de *Cloud* privadas implicam um grande investimento a priori e a dedicação de recursos para a implementação da solução, possuindo, a curto prazo, um maior valor de implementação e menos flexibilidade em termos de crescimento instantâneo e constante monitorização e manutenção da solução. No entanto permitem uma maior personalização das ações, implementação e uso em geral da solução, sendo, em grande parte dos casos, possível a implementação com *Clouds* públicas quando é necessário dar resposta a crescimentos abruptos da infraestrutura.

Em conclusão, é possível constatar que existe um consenso nas vantagens de implementação e uso de *Cloud* privadas dependendo da realidade de cada entidade. Estas oferecem essencialmente transparência de informação, poder de escolha da infraestrutura física, assim como a possibilidade de desenho perante as necessidades especificas que cada ambiente impõe por natureza. A possibilidade de personalização da infraestrutura facilita, inevitavelmente, a criação de processos autónomos, sendo estes em termos gerais, a reflexão das necessidades e tarefas sequencias sem caracter intelectual imediato reportadas pelas equipas de administração de infraestruturas e gestores de projetos.

[23] ansible.com

Capítulo 3 – Proposta Arquitetural e Especificações da Plataforma

No presente capítulo são apresentadas as propostas arquiteturais lógicas para dar resposta aos casos de uso propostos inicialmente. Ao mesmo tempo, são discutidas as opções disponíveis e justificados os caminhos escolhidos, procurando equilibrar as decisões de engenharia tomadas, com as necessidades e exigências da entidade SPMS. É também contextualizado no projeto as ferramentas e atributos da plataforma, especificando e demonstrando o projeto base para possibilitar a implementação dos casos de uso propostos.

3.1 Contextualização

Como discutido inicialmente nos objetivos do projeto, o propósito final da solução de *Cloud* utilizada passa, em parte, pela integração de diferentes sistemas de recolha e tratamento de dados, de modo a proporcionar uma maior autonomia de processos face ao cenário transato. Para tal, e durante o presente capítulo, serão apresentados e discutidos vários modos de interpretação, execução e implementação dos casos de uso propostos inicialmente pela SPMS para a integração na solução *Cloud*.

Tendo em conta que a decisão da solução *Cloud* a ser utilizada no projeto não diz respeito ao autor do presente documento, uma vez que foi adquirida a priori no início da proposta de dissertação, a responsabilidade de julgamento e escolha da solução não dizem, por consequência, respeito ao mesmo. Assim, com a obrigatoriedade da utilização da solução *vRealize* da *VMware* como base do projeto, o primeiro ponto passa por entender o seu funcionamento e componentes para que, de forma coerente e ajustada, seja possível desenhar e implementar os casos de uso propostos à medida tanto da SPMS como da plataforma *vRealize*, tendo sempre em mente o modo de operação e a melhor interação do utilizador final com a plataforma.

Com o intuito de a acoplar todos os requisitos e casos de uso inicialmente propostos, a projeção inicial de integração com os diferentes componentes, de modo a tornar intuitivo o fluxo de trabalho, passou pela divisão em dois eixos o sentido da integração (Figura 3.1):

Norte-Sul que se destina à interação entre o subscritor ou utilizador e a máquina virtual (o principal motivo da interação com a plataforma). Assim, como ilustrado na Figura 3.1, a linha de interesse de interação do subscritor, apesar de oblíquo ao mesmo, é em direção aos *hypervisors*, representando em parte o núcleo do sistema.

Este-Oeste destina-se à integração entre a solução base, o *vRealize*, e as diferentes plataformas, de terceiros em maior parte dos casos, de modo a possibilitar maior autonomia fase às funções oferecidas nativamente pelo vRealize.

É deste modo importante contextualizar desde já algumas das plataformas usadas no projeto. No que diz respeito à infraestrutura de rede, foi utilizada uma plataforma em contexto de SDN, com a solução ACI da Cisco. Sem entrar em pormenores do funcionamento da solução ACI, que não é o âmbito do presente projeto, é possível, para uma mesma rede, isolar em diferentes domínios de *broadcast* as diferentes máquinas da rede em questão. Assim, o objetivo é cada âmbito possuir um domínio de *broadcast* único, sem acessos por defeito, de modo a garantir independência total entre as máquinas de diferentes projetos, por consequência aumentando a segurança limitando a propagação de *malware* por *layer* dois, conferindo ao mesmo tempo uma sensação de *greenfield* ao subscritor.

A ferramenta PHPIPAM, ou apenas IPAM, também descrito no decorrer inúmeras vezes no decorrer do documento, possibilitam a integração via API e a documentação dos endereços de rede atribuídos a cada ambiente e máquina virtual de forma autónoma. A ferramenta, que é usada maioritariamente para controlo de ambientes e gestão de endereçamentos de rede, disponibiliza ainda uma panóplia de funções indispensáveis às ações diárias dos administradores.

Por último, a integração com servidor de email, SMTP, proporciona a recolha de informação e de ações que são do interesse dos utilizadores da plataforma, notificando sobre erros ou procedimentos necessários. A integração com plataformas terceiras limita-se unicamente aos métodos de comunicação máquina-máquina disponibilizada pelas mesmas, à qual são utilizadas técnicas discutidas na seção 2.9 Interfaces de Automação para garantir a intercomunicação.

A integração com plataformas terceiras limita-se unicamente aos métodos de comunicação máquina-máquina disponibilizada pelas mesmas, à qual são utilizadas técnicas discutidas na seção 2.9 Interfaces de Automação para garantir a intercomunicação. Assim, podemos tomar como premissa que quanto melhor e mais extensa a integração entre plataformas, maior a independência da solução de interações humanas.

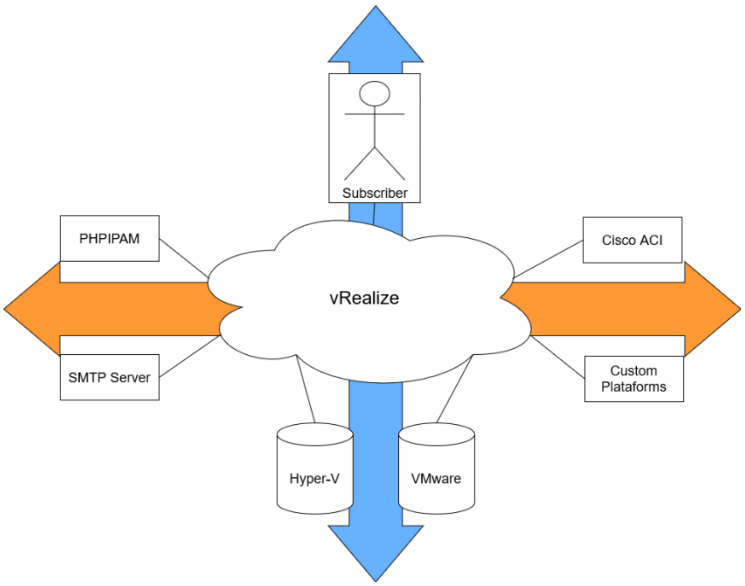


Figura 3.1 - Relações entre os diferentes componentes na arquitetura projetada

Assim, os casos de uso propostos são divididos em dois grupos, os que possuem interação com a plataforma de virtualização *VMware* e os que possuem interação com a plataforma *Hyper-V*. É de salientar que a utilização de *Hyper-V*, uma solução não suportada nativamente pela solução *vRealize*, vem responder à necessidade de resposta heterogéneas, a nível de fabricantes, de *hypervisors* de modo a introduzir redundância e a evitar pontos de falha únicos na arquitetura. O método de implementação deve garantir que exista coerência de informação e de recursos entre a plataforma de gestão e os *hypervisors*. Idealmente, para além da possibilidade de interação entre os subscritores e a plataforma de gestão, a implementação de mecanismos de autoavaliação de infraestrutura, deteção de erros e problemas aparece como um ponto importante para a gestão e garantia de bom funcionamento por parte da equipa de manutenção da plataforma.

3.2 Discussão dos Casos de Uso e Das Ferramentas Disponíveis

3.2.1 Criação de máquinas virtuais

Como apresentado, a entidade SPMS necessita de criar máquinas para três sistemas operativos, *Windows*, *Linux* e Outros (englobando sistemas operativos *legacy* ou que não se encaixem nos descritos anteriormente), sendo que máquinas *Windows*, por questões de licenciamento, deverão ser todas criadas em ambientes *Hyper-V*, enquanto as restantes serão criadas em ambiente *VMware*. A nova máquina deverá ser apresentada no grupo de rede do projeto, que caso não existe deverá ser criado. Ao momento da

criação deverá ser reservado, na rede do âmbito da máquina, um IP no IPAM e injetadas as configurações de rede, como por exemplo, IP reservado, máscara de rede, *default gateway*, DNS, entre outros, apresentando ao utilizador o IP reservado. Por fim, os recursos utilizados pela máquina deverão ser descontados à cota reservada ao projeto no qual foi criada a máquina. Nativamente as máquinas devem possuir gestão remota (via SSH[24] ou RDP[25]), assim como possibilidade de acesso à consola caso a máquina perca acesso à rede.

A integração do algoritmo de criação visa ser o mais transparente e eficiente possível, visando a fácil interação entre o utilizador e a plataforma e evitando a criação de exceções que comprometam a futura atualização das políticas.

3.2.2 Eliminação de máquinas virtuais

A necessidade de exclusão de máquinas virtuais não passa só pela libertação dos recursos computacionais alocados à mesma. No paradigma de *Cloud*, mais especificamente tendo em conta as necessidades de negócio da entidade, a eliminação de máquinas terá que passar, numa primeira fase, pela verificação e eliminação de reserva de IPs na rede da mesma, libertação do valor equivalente aos recursos ocupados pela máquina na cota do projeto, a eliminação (caso seja a única máquina do projeto) do grupo de rede do projeto na plataforma ACI, para só depois localizar a máquina no seu *Hypervisor* e proceder à eliminação, notificando no final o utilizador da eliminação via email e eliminar todos as relações com a mesma na plataforma *vRealize*.

A implementação do algoritmo deverá ter em conta a necessidade de *rollback* caso algum dos passos possua um problema, assim como imitir um alerta via email à equipa responsável pela gestão da plataforma.

3.2.3 Importação de máquinas

As necessidades da entidade SPMS ditam que o projeto não é um *green field*, sendo necessário complementar o paradigma pré-*Cloud* ao projeto. Com isto, será necessário importar máquinas oriundas das plataformas *Hyper-V* e *VMware*, disponibilizando todas as opções de customização apresentadas às máquinas criadas nativamente na plataforma. As máquinas deverão possuir os campos de parâmetros fundamentais, como por exemplo, o ambiente aplicacional, categoria aplicacional, o nome da máquina virtual e o *template* usado para a sua criação.

[24] Protocolo de acesso remoto via interface de comando (CLI)
[25] Protocolo de acesso remoto com interface gráfica usado maioritariamente em sistemas *Windows*.

3.2.4 Ligar/Desligar máquinas (em *Hyper-V*)

Como referido anteriormente, o *Hyper-V* não é suportado nativamente pelo *vRealize*, pelo que é necessário possibilitar ao subscritor uma opção de alterar o estado da máquina entre ligado e desligado, dando assim a autonomia necessária que a plataforma requer. A integração deverá ser coerente com os métodos nativos para *VMware*, de modo a não causar disrupção na experiência de uso do utilizador, ao mesmo tempo que interaja diretamente com o *Hyper-V*. A integração deve integrar princípios de segmentação de responsabilidades e de baixo acoplamento de modo a reduzir as dependências e a facilitar novas integrações e rápidas atualizações.

3.2.5 Alteração de parâmetros das máquinas (em *Hyper-V*)

Deve ser disponibilizado ao subscritor opções (das requeridas pelas primeiras abordagens dos subscritores à plataforma) de alteração de parâmetros referentes às máquinas virtuais em *Hyper-V*. O processo deverá ser dividido em duas fases, a apresentação dos parâmetros atuais (exemplo CPU: 4 cores) e a possibilidade de alteração dos valores. A implementação de cada funcionalidade deverá ter em conta os requisitos bases do *Hyper-V*, pelo que, por exemplo, não será possível reduzir espaço nos discos, mas apenas aumentar, ou não será possível alterar o número de CPU com a máquina ligada. A implementação deverá avisar o subscritor com uma mensagem informativa, o erro ou as limitações encontradas caso existam.

3.2.6 Notificação de Erros e Alertas de Operação

Em modo de prevenção, será necessário a adoção de mecanismos de verificação e alarmística de diferentes parâmetros da operação rotineira da plataforma. As alarmísticas deverão ser canalizadas para as equipas de gestão da plataforma, monitorizando aspetos como incoerências entre os *hypervisors* e a plataforma IaaS, máquinas passiveis de otimizações de recursos e máquinas com erros durante longos períodos. A integração das alarmísticas deverá disponibilizar imediatamente os resultados obtidos aos gestores da plataforma ou serem executados, de modo autónomo, com a frequência que assim necessitar.

3.3 Base de trabalho

De modo a introduzir a temática prática do trabalho proposto, o primeiro aspeto a responder foi a estruturação da base do projeto, tendo em mente as recomendações do fabricante. Para tal, e com o software previamente instalado e configurado, inicializou-se a personalização da solução visando segmentar as responsabilidades e criar ambientes de intervenção seccionados para as diferentes equipas e projeto. Em modo de inicialização, é necessário conectar a solução ao servidor LDAP da entidade SPMS, para tal acede-se ao componente *Identify Manager*, plataforma de acesso exclusivo ao utilizador *Administrator* da solução, e na aba *Identify & Access Management* efetuamos o registo do servidor (Figura 3.2).

Figura 3.2 - Registo do servidor LDAP no *vRealize*

Após a configuração dos parâmetros requisitados, é necessário aceder à nova linha de registo de *LDAP* e selecionar *Sync Settings -> Groups* e adicionar o *Distinguished Names* [52] do grupo do *Active Directory* (AD) que é necessário ter acesso à plataforma e que ficará associado a um ou mais projetos. No final é necessário guardar e forçar a sincronização entre as plataformas na opção *Save & Sync* (Figura 3.3). (Nota: O *vRealize* necessita de parâmetros específicos de cada utilizador da AD, pelo que caso não estejam preenchidos não será possível importar o utilizador em questão). Após a sincronização será possível ver os utilizadores importados na aba *Users & Groups*.

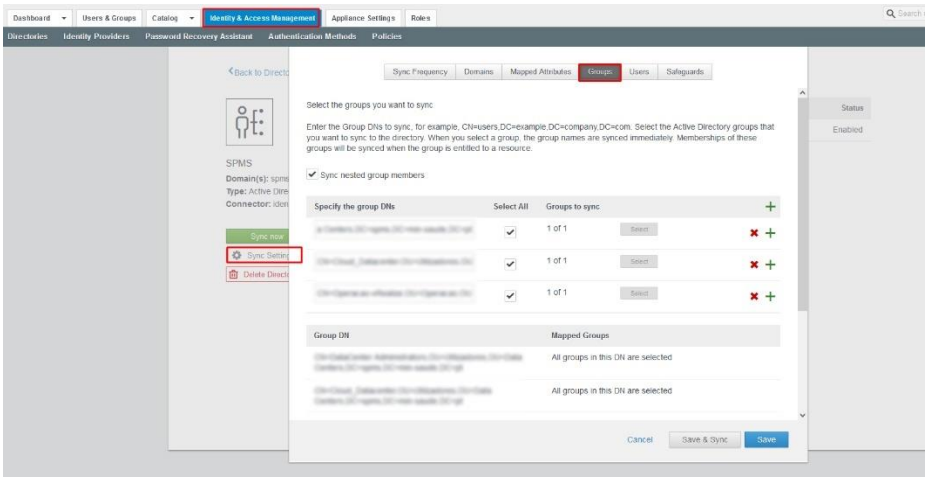


Figura 3.3 - Adição dos grupos de AD ao *vRealize*

De seguida, já na componente *Cloud Assembly* da plataforma *Automation* procedemos ao registo dos componentes de computação da *VMware*. Para tal devemos começar por registar o acesso ao *hypervisor* na plataforma, na aba *Infrastructure*, *Connections*, *Cloud Accounts*. Nesta aba, para além da possibilidade de registar o *hypervisor*, também é possível efetuar o registo a *Clouds* públicas como *AWS*, *Google Cloud* ou *Azure*. No presente exemplo, efetuamos a configuração do acesso ao *hypervisor VMware*, sendo necessário especificar aspetos como o nome da conta, endereço IP, *username* da conta de serviço (com permissões de administração) e a respetiva palavra-passe. Após validar as configurações, é possível selecionar o novo objeto da *Cloud Account* e proceder à sincronização das imagens e *deployments* nos respetivos botões (esta opção não é obrigatoriamente necessária). Após a adição da *Cloud Account* é possível na aba lateral *Resources* verificar os recursos computacionais, placas de rede, armazenamento, entre outros aspetos relativos às máquinas virtuais e ao *hypervisor*.

A partir deste ponto, existem condições básicas reunidas para criar um projeto e associar recursos computacionais. Cada projeto pode ser populado por vários grupos de utilizadores ou vários utilizadores com diferentes permissões dependendo do nível de administração desejado (explorado em detalhe no decorrer do documento) (Figura 3.4).

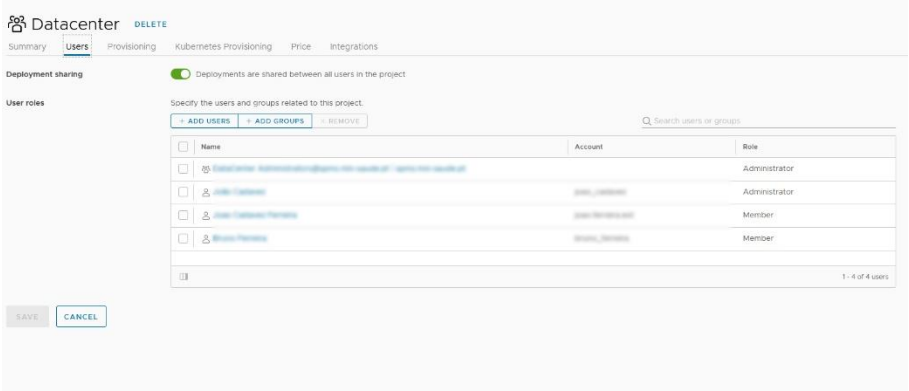


Figura 3.4 - Definição dos níveis de administração do projeto

Nas *Cloud Zones* é possível limitar os recursos usados e/ou definir prioridades entre *Cloud Zones* (Figura 3.5). É importante definir limites nos projetos de modo a impedir crescimentos inesperados do uso de recursos. Em adenda, é possível definir parâmetros personalizados que podem ser especificados a priori de modo a abrangerem todo o projeto em *deployments* futuros.

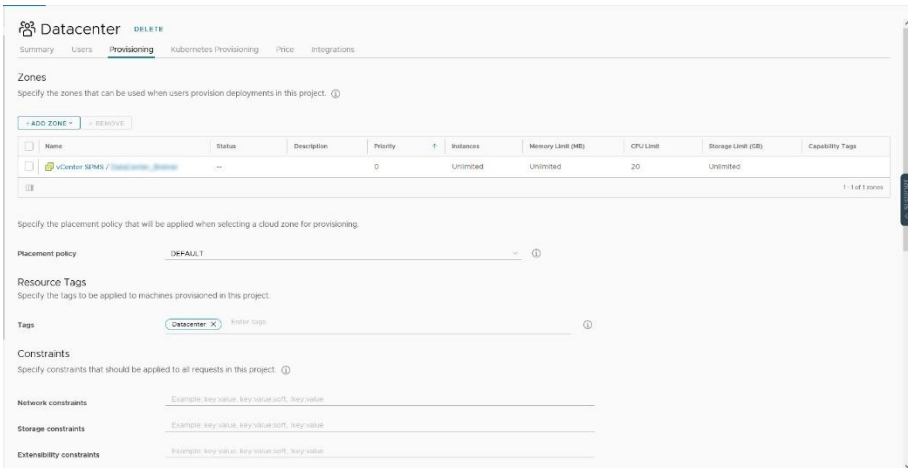


Figura 3.5 - Exemplo da configuração dos projetos no *vRealize*

A base do projeto está concluída podendo passar assim para a integração com outras plataformas, sendo que, por defeito, os componentes do *vRealize* encontram-se segmentados entre eles. Assim, de modo a permitir a integração, e como descrito anteriormente, devemos aceder à aba *Integrations* em *Cloud Assembly*, *Infrastructure*, *Connections* e selecionar a integração desejada. Opções como *vRealize Operations Manager*, *vRealize Orchestrator* ou *My VMware* são opções consideradas "essenciais" uma vez que possuem relação direta com alguns dos componentes do *vRealize*. As integrações de outras funcionalidades podem oferecer opções interessantes e úteis, como a possibilidade de *backups* incrementais para uma plataforma *Git* como o *GitLab*, *GitHub* ou *BitBucket*, ou a interação com plataformas de documentação como o *IPAM* (Figura 3.6).

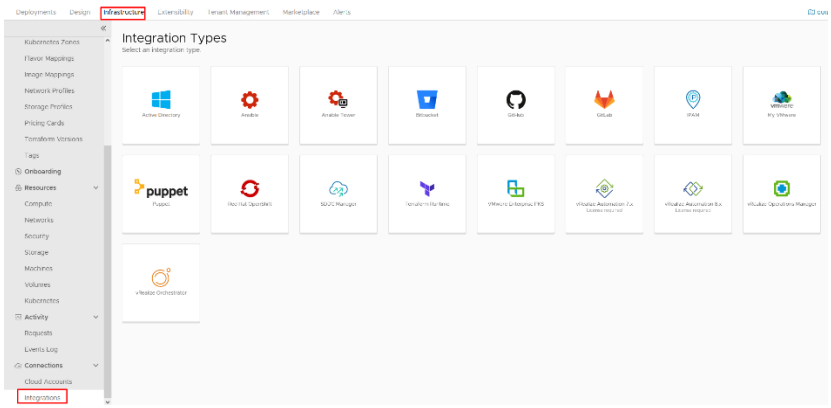


Figura 3.6 - Integrações nativas *vRealize*

3.3.1 Criação de Projeto Base

De modo a permitir a inicialização da integração do *Hyper-V*, foi necessário a criação de um projeto básico capaz de fornecer um ambiente de testes e integração da solução.

Como ilustrado na Figura 3.7, na aba *Provisioning*, foi necessário associar o projeto a uma *Cloud Zone* e definir os recursos disponíveis à mesma (que para a finalidade foi deixado com as configurações por defeito, ou seja, sem limites de processamento, memória ou armazenamento). É também possível a configuração de restrições de modo a possibilitar associações automáticas de *tags* tendo em conta o projeto, assim como criar nomes customizados das máquinas e dos *deploys* criados com os parâmetros do projeto e da mesma.

Na aba *Users* é possível especificar os grupos ou utilizadores que possuem acessos ao projeto e o grau hierárquico no mesmo. Estes podem ser *administrator* e ter acesso a todos os recursos do projeto, *member* que dependendo da configuração que o *administrator* proporcionar, possui permissões a determinadas ações e configurações ou o *viewer* apenas possui acesso a visualizar os recursos e as configurações do projeto.

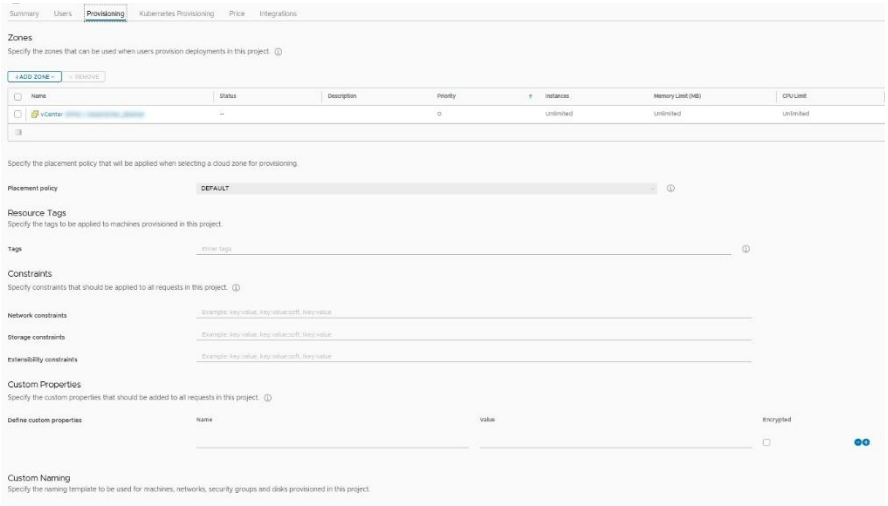


Figura 3.7 - Configuração do projeto

Pela natureza complexa da plataforma existem ainda uma panóplia de opções e mecanismos que enriquecem e possibilitam a customização que não são explorados no presente trabalho.

3.4 Integração Com a Plataforma *VMware*

A integração do *vRealize* com a plataforma de virtualização da *VMware* apresenta-se como um processo transparente pelo simples facto da solução ter sido desenhada para funcionar em volta da *VMware*, como já referido anteriormente. Após a interligação das duas plataformas, como descrito no subcapítulo Base de trabalho, é necessário mapear os recursos no *vRealize*.

O mapeamento de recursos permite definir, de um modo abrangente e ao mesmo tempo volátil, fazendo assim a ponte entre valores fixos de recursos e uma conceção lógica menos técnica.

3.4.1 Mapeamento de *Flavors*

Como ilustrado na Figura 3.8, o mapeamento tem como base a seleção da região de computação (como apresentado anteriormente pode variar entre soluções *on-premise* ou em *Cloud* pública) e os recursos físicos/lógicos destinados desejados.

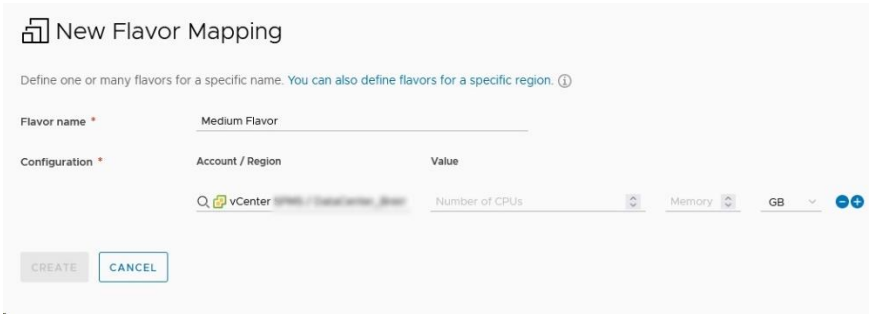


Figura 3.8 - Definição de novo mapeamento de recursos

Tomemos como exemplo aplicações com diferentes requisitos computacionais. O gestor da plataforma cria diferentes "*flavors*" (terminologia utilizada pela *VMware* para identificar diferentes patamares), cada um com características diferentes. Podemos assim assumir, hipoteticamente, que o *flavor* "pequeno" seja destinado a aplicações com baixos requisitos de computação e assim possam estar numa região de máquinas mais antigas ou com menos recursos físicos e que compartilham um patamar base de 2 cores e 2 gigabytes de RAM por máquina virtual. Por outro lado, máquinas com requisitos mais robustos destinam-se a um *flavor* diferentes, como seria o exemplo do "grande", que se pode encontram numa região de máquinas físicas mais vigorosa e com, por exemplo, 8 cores e 16 gigabytes de RAM.

O benefício do uso de *flavors* é a flexibilidade de alterar valores consoante o crescimento da plataforma e dos requisitos, assim como fornecer ao subscritor uma base de trabalho caso não tenha em mente a priori as necessidades requeridas.

3.4.2 Mapeamento de Imagens

Tal como acontece nos *flavors*, o mapeamento de imagens permite importar para o *vRealize* imagens presentes nas diferentes regiões de computação. Deste modo é possível introduzir imagens do tipo ISO, ou *templates* criados a priori com as configurações e opções nativas desejadas pela instituição em questão. Em adenda, como ilustrado na Figura 3.9, é também possível a utilização de *constraints* que, como abordado no decorrer do documento, permitem a utilização dinâmica da imagem por parte do *deployment*, assim como a opção de adição de comandos no campo *Cloud Configuration* que se refletem em comandos, pelo *Cloud-Init* (discutidos em profundidade no decorrer do documento) diretamente na máquina criada (como alterar o nome da máquina, palavra-passe, endereço IP, entre outros), facilitando a personalização das máquinas e reforçando a automação e autonomia da plataforma.

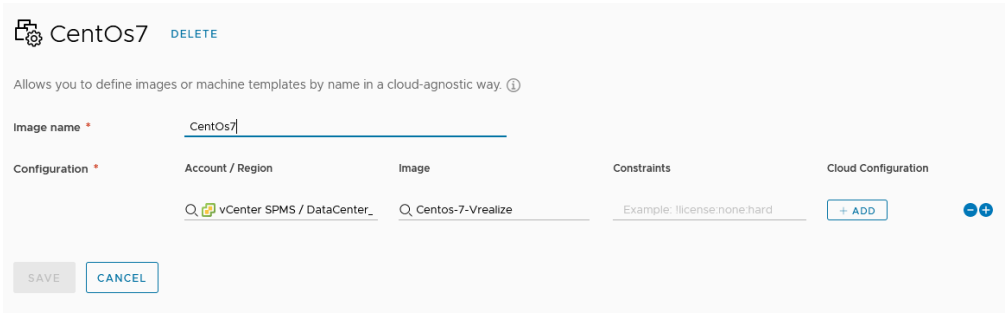


Figura 3.9 - Mapeamento de imagens

Em termo de nota, caso uma das imagens do *VMware* não esteja a aparecer nas imagens disponíveis para o mapeamento, é necessário ir a *Infrastructure* -> *Connections* -> *Cloud Accounts*, aceder à conta e sincronizar as imagens da plataforma [53].

3.4.3 Mapeamento de Rede

O mapeamento de rede permite ao gestor da plataforma apresentar, restringir e controlar o modo como diferentes máquinas são apresentadas à rede, podendo passar por um método puramente arbitrário do utilizador, ou por mecanismos mais complexos que podem variar desde a pura gestão de endereçamento disponível, até a questões de caracter de segurança e isolamento.

Um dos cenários relatados como base de conceito é a divisão das redes em tipos de projeto, como por exemplo, desenvolvimento, qualidade e produção. Com a utilização do mapeamento de rede é possível definir *tags*, como ilustrado na Figura 3.10, que, no momento do *deployment*, permitem a seleção automática da rede baseada nas regras aplicadas à seleção de opções.

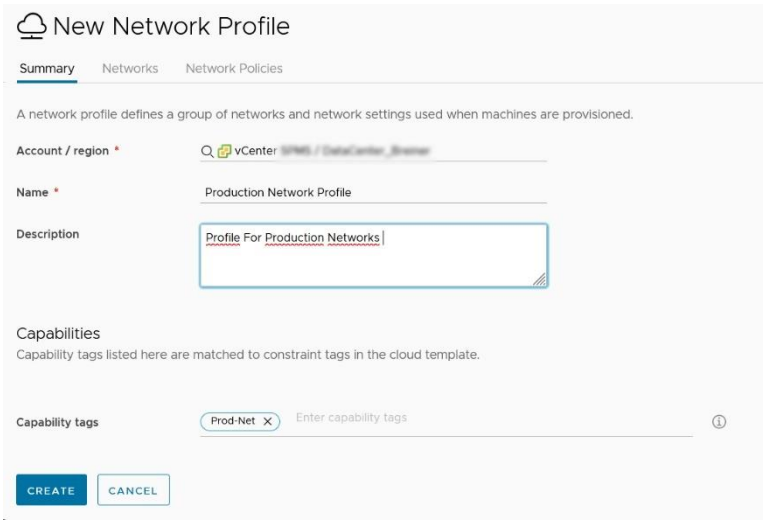


Figura 3.10 - Criação de mapeamento de rede

Na aba *Networks* é possível, a partir do *vCenter*, selecionar as placas de redes que fazem parte do perfil em questão. A seleção da placa de rede vem associada à opção de seleção de uma gama de IPs a serem disponibilizados e, a partir de uma ferramenta de terceiros, o IPAM (analisada a integração em detalhe no decorrer do documento), é possível efetuar a gestão automática dos endereços aquando da criação das máquinas, assim como a gestão da documentação referente aos mesmos na plataforma, acabando por funcionar como um servidor de DHCP sem *lease*, mantendo assim o IP permanentemente reservado. É também possível, para cada placa de rede a seleção dos parâmetros inerentes à rede, permitindo assim um elevado nível de granularidade [54].

É de salientar também a existência de opções política de isolamento que se encontram apenas destinadas à utilização em ambientes *NSX* ou *VMware Cloud*.

3.4.4 Mapeamento de Armazenamento

Nos mesmos moldes dos descritos anteriormente, o mapeamento de armazenamento permite, às medidas da plataforma, um reconhecimento dos recursos

do *hypervisor*, assim como a gestão (reserva, libertação e uso) dos recursos da infraestrutura lógica e/ou física.

Para o projeto, e tendo em mente a configuração mínima de funcionamento, o armazenamento foi configurado em modo *Standard Disk*, uma vez que não foi garantido do lado do *hypervisor* a possibilidade de utilização da opção *First Class Disk*. Na opção das *storage policy* é necessário selecionar a opção de mapeamento do tipo de disco criado no *hypervisor*, sendo no caso do tipo *Production Datastore*, Figura 3.11. De entre as demais opções não foi necessária nenhuma configuração extra para possibilitar o desempenho desejado no projeto. No entanto, é de notar que o campo de armazenamento é altamente configurável e personalizável, possibilitando a divisão clara e coerente dos recursos.

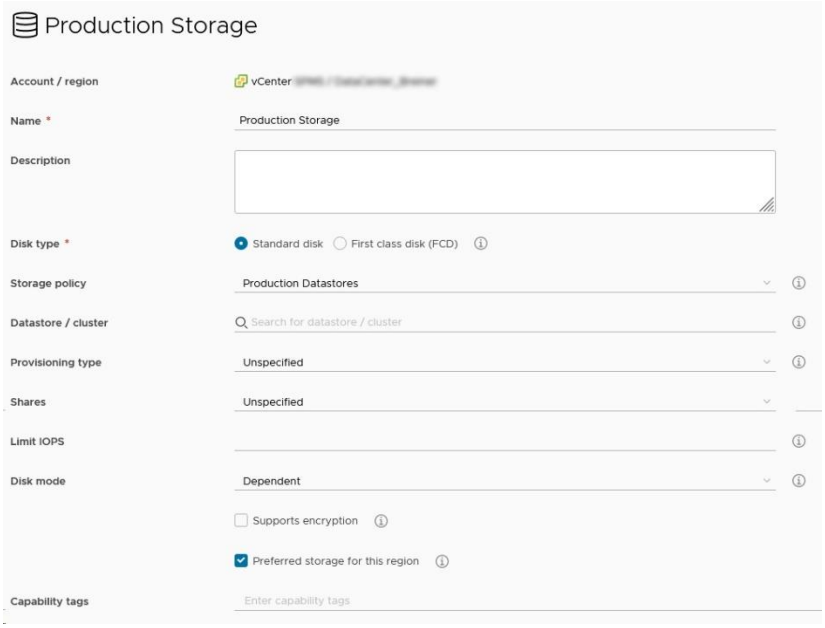


Figura 3.11 - Criação de mapeamento de armazenamento

3.4.5 Integração entre módulos *vRealize*

Tal como referenciado anteriormente, a plataforma *vRealize* é, por natureza, modular o que, ao culminar com a possibilidade de aquisição de licenciamento por módulos individuais resulta na necessidade da, apesar de simples, integração manual entre os módulos nativos de terceiros, como presente na Figura 3.12.

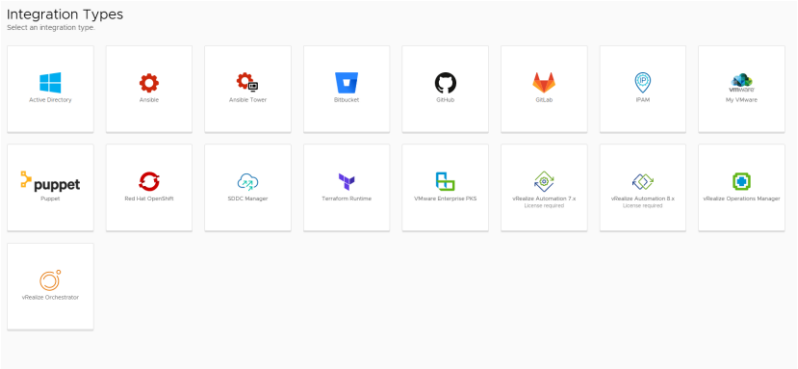


Figura 3.12 - Tipos de integração *Cloud Assembly*

Durante o presente projeto, foi necessária a integração entre os módulos, *service broker* e *code stream* que pode ser feita diretamente no módulo *service broker* na opção *Content Sources* na aba *Content & Policies*, sendo necessário a especificação do projeto em causa e do nome da integração, como ilustrado na Figura 3.13.

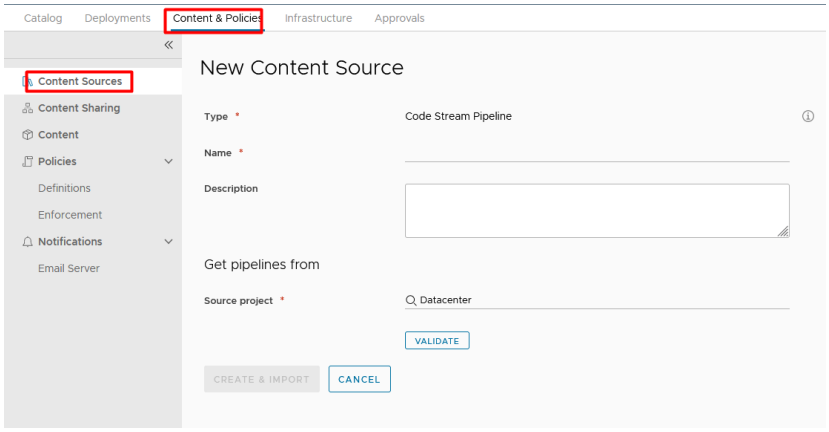


Figura 3.13 - Integração *Service Broker - Code Stream*

Da mesma forma, foi necessário a integração entre o *Cloud Assembly* e o *Orchestrator*, sendo adicionado no módulo *Cloud Assembly* em *Infrastructure -> Connections -> Integrations*, Figura 3.14.

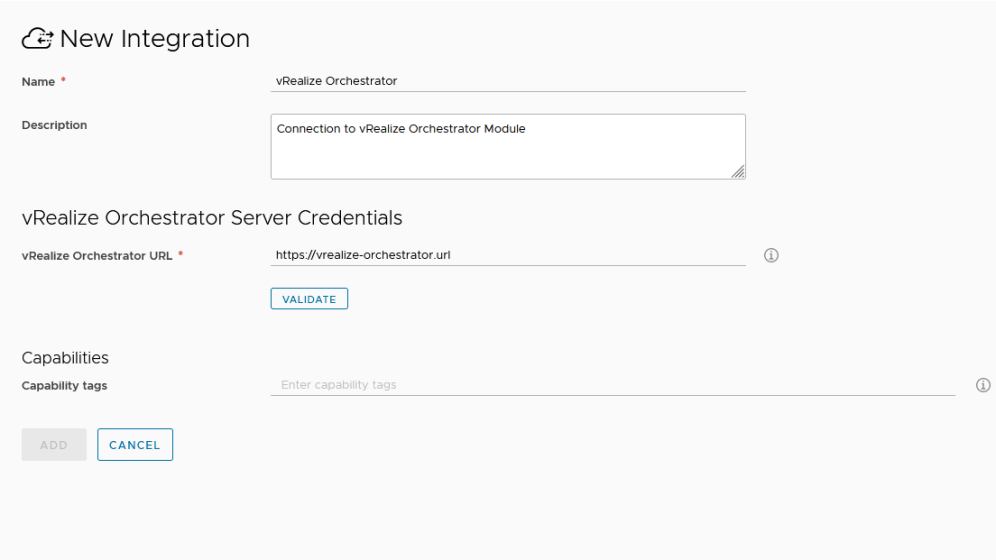


Figura 3.14 - Integração *Cloud Assembly - Orchestrator*

3.5 Introdução à Plataforma de *Scripting*

A versatilidade da plataforma *vRealize* e a simples integração e *deployment* de infraestrutura são alguns dos pontos fortes da mesma. De modo a conseguir ir de encontro a todos (ou quase todos) os requisitos únicos de cada cliente, a solução passou por desenvolver uma plataforma dedicada ao *scripting* e automação, que integra com múltiplas plataformas e disponibiliza uma panóplia de linguagens de programação, criando um *middleware* entre as aplicações terceiras e a plataforma *Cloud*, ou criar tarefas de rotina internas à plataforma.

O primeiro impacto com a ferramenta, e como ilustrado na Figura 3.15, é o menu de monitorização, que disponibiliza alguns dos dados do desempenho e contadores referente à máquina dedicada à plataforma, assim como estatísticas dos *scripts* que foram executados, tempos de resposta e/ou execução e sucesso ou insucesso dos mesmos.

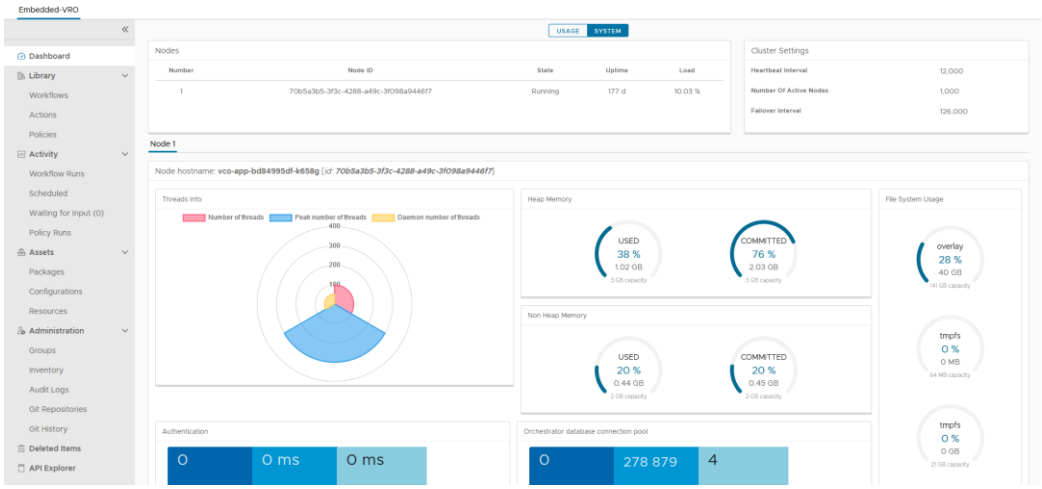


Figura 3.15 - Página inicial *Orchestrator*

De modo a desenvolver e integrar os *scripts* desenvolvidos recorre-se aos *workflows.* Os *workflows* apresentam-se como um objeto de integração que faz a ponte entre a personalização e os recursos da plataforma, permitindo o desenvolvimento em código ou pseudocódigo de funções com baixa/média complexidade capazes de espoletar, internamente ou externamente, ações de alerta, interação com o utilizador ou com o gestor, entre outras.

3.5.1 *Workflow*

Cada *workflow* possui um identificador único interno, que é utilizado pela plataforma e em alguns casos raros pelo gestor da plataforma, assim como um nome único que identifica o seu comportamento, que podem ser encontrados na aba *summary*. Ainda na mesma aba é possível encontrar alguns campos destinados à organização dos *workflows*, como o campo da versão, que ajuda a identificar as diferentes iterações do *workflow*, o campo *tags* que permite a organização por elementos-chave, um campo descrição para anotações do comportamento do *workflow*, grupos de acesso e a pasta de localização do *workflow*. Apesar da extensa lista de ações disponíveis na aba *summary* dos *workflow* a utilização regular destes campos permite uma organização importante nas plataformas, especialmente com o crescimento do número de *scripts* nos diferentes âmbitos de cada projeto.

A aba *variables* e *inputs/output* são abas de criação de variáveis globais ao *workflow*. Existem, à criação do presente documento, um pouco mais de setecentos e cinquenta tipos diferentes de variáveis, podendo ser simples (como *strings*, *number*, *boolean*, etc.) ou complexos (*Active Directory Group*, *Powershell Object*, etc.), podendo

ser inicializadas e preenchidas a priori ou no decorrer do *workflow*. Existe também a possibilidade de criação de variáveis personalizáveis que podem ser utilizados tais como os objetos em linguagens como o *Java* ou *C#*, e permite o mapeamento num único objeto de vários valores de diferentes tipos. A adenda ao anteriormente descrito, a aba *Input/Output* possui ainda a possibilidade de criação de variáveis do tipo *Input* e *Output* que vão ser apresentadas como objetos de introdução de variáveis e obtenção de resultados no final do *workflow*.

Para determinar o fluxo do *script* e definir as diferentes regras de negócio cada *workflow* disponibiliza uma tela branca onde é possível dispor diferentes elementos interativos (Figura 3.16). Cada fluxo necessita obrigatoriamente de um elemento de começo (*start workflow*) e de um ou mais elementos de fim (*end workflow*) que necessitam, entre o início e o final, de continuidade em todos os casos. Por defeito, caso seja obtido uma mensagem de erro ou falha, é lançado um aviso que pode ser tratado caso seja capturado no local onde foi lançado. O processo é sempre sequencial (exceto nos casos em que são necessários processos assíncronos), começando no elemento de começo e seguindo o fluxo pré-definido que, dependendo das regras de negócio, podem assumir diferentes fluxos no *workflow*.

Figura 3.16 - Esquema de ação do *Workflow*

Os elementos servem de base para todas as operações que podem ser criadas por cada *workflow*, e a combinação dos mesmos possibilitam replicar funções de programação complexas. À medida do presente, e respeitando *Single Responsability*

Principle, cada *workflow* deve possuir apenas uma função bem definida, sendo o mais genérico possível, permitindo que possa ser reutilizado noutros *workflows* sem necessidade de repetição de código e com o benefício de propagação de atualizações com maior facilidade.

O elemento *Scriptabke Task* apresenta-se como uma das opções mais utilizadas e mais úteis da ferramenta. Com a adição do mesmo, é possível definir variáveis a serem utilizadas, assim como na aba *Scripting* utilizar a caixa de texto para definir as políticas de negócio. Como se pode verificar na Figura 3.17 (canto superior esquerdo), é possível definir a linguagem de programação a ser utilizada. Entre as opções estão *Javascript*, *Python3.7*, *Node.JS* e *PowerCLI*, sendo por defeito utilizada a linguagem *Javascipt*, que é também a recomendada pela maior documentação na plataforma *vRealize* e também por ser a única em iterações anteriores da plataforma. Com recurso às variáveis importadas, é possível o tratamento das mesmas a partir de código e funções nativas de cada linguagem, oferecendo no *output* os diferentes resultados originados pelas regras de negócio definidas a priori. É importante salientar que, atualmente, apesar das linguagens referidas suportarem bibliotecas externas no seu modo nativo, na plataforma apenas é possível utilizar as funções nativas de cada linguagem.

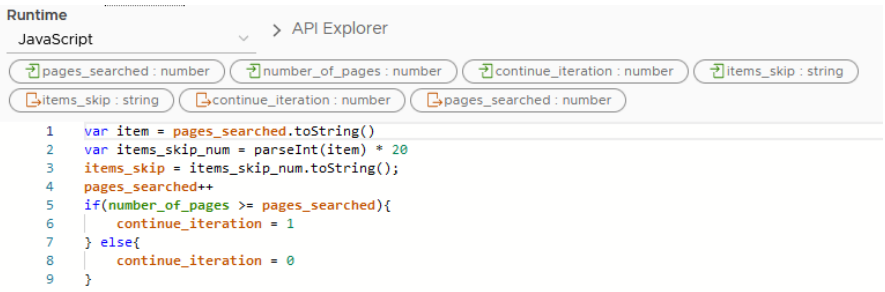


Figura 3.17 - Excerto de código do elemento *Scriptable Task*

Para permitir a reutilização de *workflow* previamente definidos, é possível selecionar um *workflow element* que permite dentro de um *workflow* evocar outro. Com isto é possível reutilizar código, efetuar pedidos recursivos (apesar de não ser muito recomendado dada a natureza da plataforma). Existem também derivações do elemento como o *Foreach Element* que itera uma lista e inicializa os *workflow*, os *Asynchronous Element* para inicializar o *workflow* e não aguardar que este acabe para passar para o próximo elemento, *Nested Workflow* vários *workflow* diferentes seguidos ou *Schedule Workflow* para calendarizar *workflow* para um ponto no tempo.

O elemento *Decision* apresenta-se também como fulcral para a fluidez dos casos de uso, permitindo a decisão tendo em conta uma ou mais variáveis. Com dois modos de operação, um simples e um normal, o seu funcionamento baseia-se em código e no retorno de valores verdadeiros e falso. Em alternativa, o uso do elemento *Switch* possui um fluxo de funcionamento semelhante ao *Decision*, permitindo a utilização de mais de duas decisões possíveis de saída para elementos diferentes. Este, utiliza um modo simplista de decisões e é baseado em *if then else*, possuindo também um valor por omissão.

Do mesmo modo que maior parte dos editores de código, também é possível utilizar funções de *debug* para facilitar o desenvolvimento e o *troubleshoot* do código, contribuindo para a facilidade de uso da plataforma. É possível também a integração com plataformas de controlo de versões como o *github*, *gitlab* ou *bitbucket* que para além de criar um histórico de versões dos *scripts* criados, oferece também um método de *backup* das configurações para um caso de *disaster recovery* da infraestrutura e/ou plataforma. É também importante salientar que a própria plataforma possui um método de controlo de versões para fácil *rollback* das configurações caso se justifique.

Em modo de nota, durante o estudo e exploração dos *workflows*, foi possível constatar que por vezes as linguagens de programação não apresentam o comportamento espectável em comparação com as versões mais recentes das mesmas, e nem todos os métodos nativos são suportados ou apresentam a sintaxe esperada. Tomemos por exemplo, em *javascript*, a opção de iteração *foreach*, que apesar de ser recomendada pelo editor de código da plataforma, apresenta erro na linha sempre que o *script* é executado, não se verificando o mesmo comportamento noutros ambientes. A utilização dos métodos nativos são, em *javascript*, evocados pelo caracter "." após a variável, seguido do método nome do método desejado e terminando com os argumentos dentro de parênteses (em muito dos casos vazio), no entanto na utilização de (apenas) alguns métodos nativos, como por exemplo "*length*", não é necessário, na plataforma *vRealize*, a utilização de parênteses, gerando inclusive erros, caso sejam utilizados.

3.5.2 *Actions*

Possuindo só *input* e *output* de dados, as *Actions* disponibilizam uma plataforma de desenvolvimento de código em que a função é apenas processar dados e retornar os mesmos. O seu uso é normalmente reservado para pequenos *scripts* ou validação recorrentes que não justificam a utilização de *workflow*.

Pela sua natureza mais simples e objetivas, podem ser usadas para formar e guardar listagens partilhadas que tanto podem ser usadas pelo *Orchestrator* ou pelo *Cloud Assembly* como listas externas. É também viável para a centralização de dados de acesso a plataformas terceiras, facilitando a atualização dos dados caso seja necessário de uma única vez.

Como descrito no decorrer do trabalho, é possível usar as *Actions* para invocar *workflow*, sendo, no entanto, anti padrão o seu uso. Sendo descritos como simples blocos de processamento de dados, a introdução de dependências com os *workflow* criam fluxos de informação que podem ser considerados mais complexos do que os originalmente arquitetados, podendo até criar ciclos recursivos se usados de forma leviana.

3.5.3 Monitorização e Cronograma

A utilidade de ferramentas de monitorização é por vezes menosprezada nas primeiras etapas de um novo projeto, por muitas vezes ser complexo avaliar o comportamento normal ou por não introduzir, numa primeira fase, informação relevante sobre o sistema ou os serviços no mesmo. É, no entanto, importante desde o início começar por entender os padrões de funcionamento e as informações possíveis de serem observadas para facilitar tanto o processo de desenvolvimento, como o processo de *troubleshoot* da plataforma, sendo possível diagnosticar de modo passivo problemas que se comecem a manifestar.

No *Orchestrator*, para além do painel apresentado inicialmente com os estados dos diferentes componentes, é também disponibilizada a opção de monitorização do processo associado a uma execução de um *workflow*. Este, apresenta os valores iniciais tomados como *inputs*, o caminho entre o início e o fim do *workflow* tendo em conta os *inputs* e os valores de *log* enviados para a consola, assim como uma aba de desempenho que apresenta tempos de execução dentro do *workflow* e recursos utilizados (Figura 3.18). Com isto, é possível ter um registo das ações executadas e dos resultados obtidos facilitando a monitorização e a evolução dos *scripts* com o aumento da utilização por parte dos diferentes utilizadores.

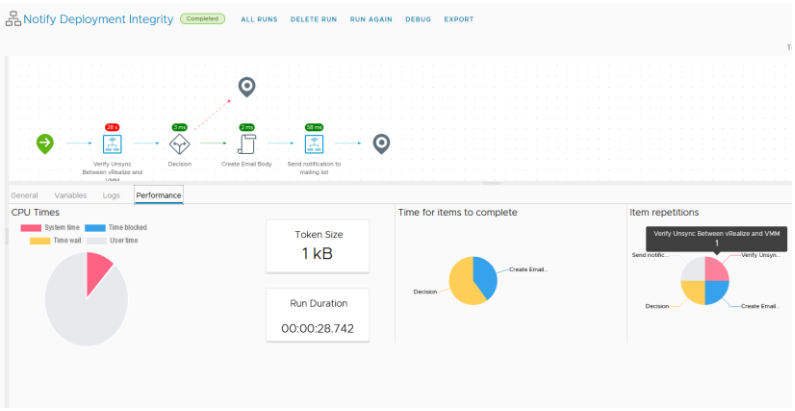


Figura 3.18 - Exemplo de informação de um *workflow* executado

A possibilidade de agendamento de ações é também um dos pontos importantes para a maior flexibilidade da solução. Assim, a plataforma disponibiliza a opção de executar ações sem nenhum estímulo externo, como normalmente acontece, possibilitando a especificação da frequência da execução, assim como a hora e as datas de início e de fim (Figura 3.19).

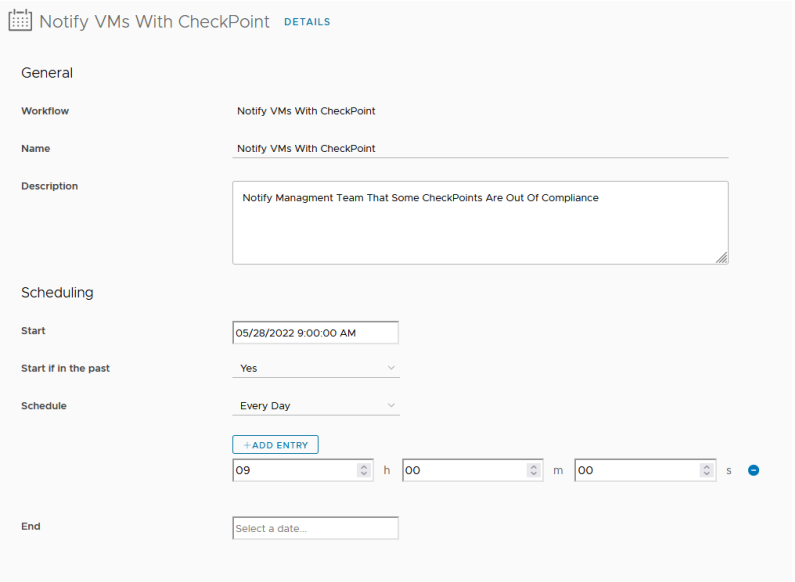


Figura 3.19 - Exemplo de agendamento de *workflow*

Capítulo 4 - Implementação da Solução

O presente capítulo destina-se à discussão sobre a implementação dos casos de uso propostos pela entidade SPMS e descritos anteriormente, sendo possível entender o processo lógico de implementação, assim como os passos necessários para a integração entre as diferentes plataformas e os métodos usados para as mesmas.

4.1 Implementação dos Casos de Uso

Após a apresentação dos casos de uso na secção (3.2 - Discussão dos Casos de Uso e Das Ferramentas Disponíveis), a posterior explicação da bancada de trabalho na secção (3.3 - Base de trabalho) e a explicação do modo de funcionamento de alguns elementos que constituem a plataforma *vRealize* (3.5 - Introdução à Plataforma de *Scripting*) é possível inicializar a apresentação técnica da implementação dos casos de uso. São ainda aprofundados alguns aspetos que ficaram por explicar nos capítulos anteriores, assim como foi possível interligar as ferramentas de modo que o subscritor da plataforma mantenha a ubiquidade da mesma.

Foi utilizado como linguagem de *scripting* base o *Javascript* pela simplicidade da mesma, assim como pela documentação mais vasta na comunidade *vRealize*. Em termos de interatividade com plataformas terceiras, foi necessário estringir aos métodos de interação oferecidos nativamente pelos mesmos, pelo que não foi então possível homogeneizar um único método.

4.2 Importação de máquinas

Como descrito no (3.2.3 - Importação de máquinas) e por se tratar de um projeto de caracter *brownfield* com as necessidades de coexistir com os anteriormente existentes, foi necessária a análise dos métodos que assim o permitiam.

A solução *vRealize*, e já como descrito por várias vezes no decorrer do presente documento, faz um bom trabalho na integração dos ecossistemas da própria companhia, tendo inclusive em conta a necessidade de importação de máquinas do *hypervisor vCenter* para a plataforma. Deste modo, o processo de importação torna-se simples e intuitivo sendo necessário como requisito a máquina que será importada estar no *hypervisor* que possui a ligação à plataforma, como ilustrado na Figura 4.1.

De seguida, e como ilustrado na Figura 4.1, é utilizada a opção nativa *Onboarding* (*Cloud Assembly -> Infrastructure -> OnBoarding*), selecionando o projeto e

de seguida a/as máquinas a serem importadas e o tipo de *deployment* que será apresentado. Por motivos de coerência e uniformidade é selecionado um *deployment* por máquina, respeitando a relação anteriormente definida.

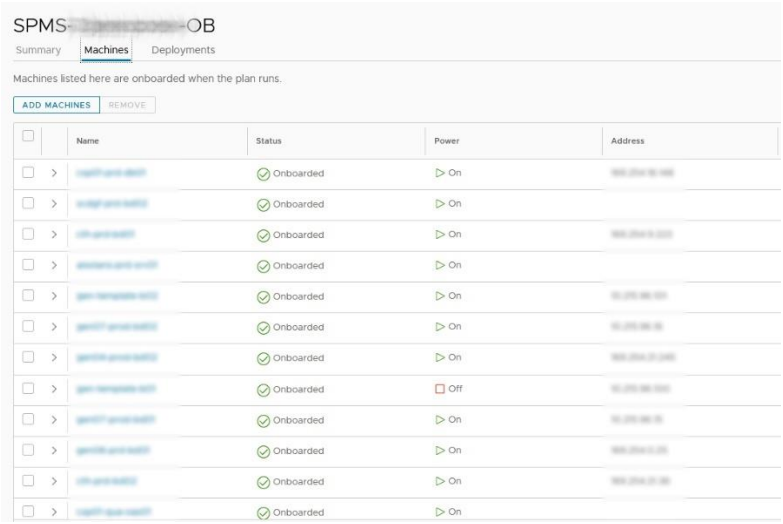


Figura 4.1 - Menu de importação de máquinas pré-existentes para o *vRealize*

Apesar de ser apresentado com um ícone diferente de um *deployment* criado originalmente no *vRealize*, e tendo em mente a importância de manter uma ubiquidade do tipo de *deployment* para a boa experiência de uso para o utilizador final, o ícone foi também convertido para a representação das máquinas em ambiente *VMware*.

No caso do *Hyper-V*, a implementação é um pouco mais complexa. Durante o estudo, e como muitas vezes referido anteriormente, um dos principais objetivos é facilitar a utilização da plataforma ao utilizador, pelo que foi evitado ao máximo sempre que possível a criação de exceções ao fluxo natural da plataforma. Com isto, a melhor opção para disponibilizar ao subscritor acesso às máquinas em *Hyper-V* será da mesma forma que acede às máquinas *VMware*, a partir dos *deployment* na *Service Broker*.

A possibilidade de apresentação de *deployments* no *Service Broker*, como ilustrado na Figura 4.2, passam pelas opções nativas, que não oferecem interatividade com o *Hyper-V*, o *Code Stream* que apesar de possuir alguns recursos interessantes para o desenvolvimento é mais focado nos processos de CI/CD, não possibilitando a flexibilidade necessária, ou a utilização de um *workflow* do *Orchestrator*. Pela maior versatilidade e capacidades de *scripting*, o *Orchestrator* foi a opção principal para a interatividade com as máquinas em *Hyper-V*.

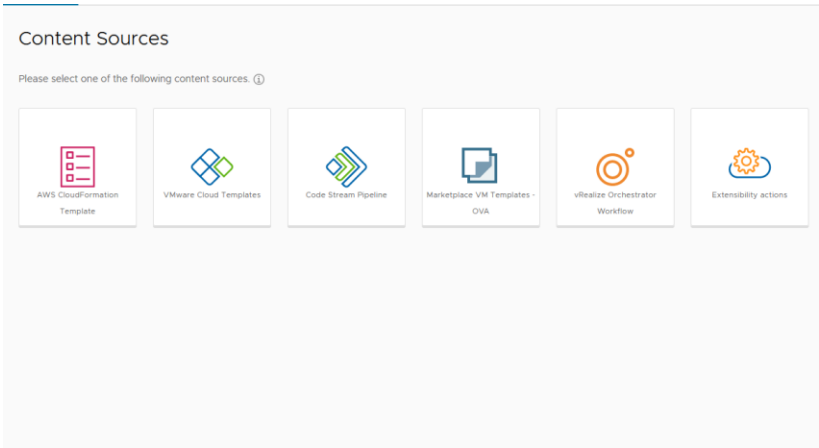


Figura 4.2 - Opções de criação de *deployments*

Para interagir com a máquina é então necessário um identificador único da mesma por parte do *Hyper-V*, sendo por conveniência, por facilidade de recolha e facilidade de interpretação selecionado o identificador "nome". Deste modo, todos os *scripts* serão desenvolvidos em torno do nome da máquina, limitando assim a mudança do nome das máquinas depois de criadas.

Com o identificador único definido, foi apenas necessário selecionar outros parâmetros que possam ser pertinentes no futuro. Para tal, e com a ajuda do *steakholder* do projeto, foi possível definir que para além do nome da máquina outros aspetos como o ambiente da máquina (DEV, QA, PRD[26]), a categoria aplicacional (APP, BD, WEB, SERVICES[27]), assim como o *template* de *Hyper-V* usado para criar a máquina. A alteração desta opção fica limitada ao momento da criação, pelo que não é possível efetuar qualquer alteração deste campo posteriormente. No entanto é importante salientar que o *deployment* apenas se trata de uma ilustração lógica da máquina virtual, pelo que a eliminação e posterior recriação com novos parâmetros é também possível, caso se justifique.

A criação do *deployment* não necessita então de espoletar nenhuma ação, uma vez que, como referido anteriormente, se trata apenas de uma abstração lógica. Com isto em mente, foi criado uma simples *pipeline* no *Code Stream* que recebe como parâmetro os elementos que queremos manter associados ao *deployment*. Assim, e como ilustrado na Figura 4.3, a pipeline possui os parâmetros anteriormente definidos e

[26] DEV -> Desenvolvimento: Ambiente inicial de uma aplicação, ambiente de testes; QA -> Qualidade: Ambiente intermédio, primeiros testes ponto-a-ponto; PRD -> Produção: Ambientes em produção, utilizado pelo cliente final

[27] APP -> Aplicacional: Serviços aplicacionais (*software* central); BD -> Base de Dados: Recolha e preservação de dados aplicacionais; WEB: Interface web, *frontend*; SERVICES -> Serviços: Serviços genéricos que não se enquadram nas restantes categorias;

apenas uma tarefa com uma validação sempre verdadeira. Esta validação, que no presente exemplo se trata de uma condição "1==1" permite o sucesso da operação e a representação lógica do objeto da máquina.

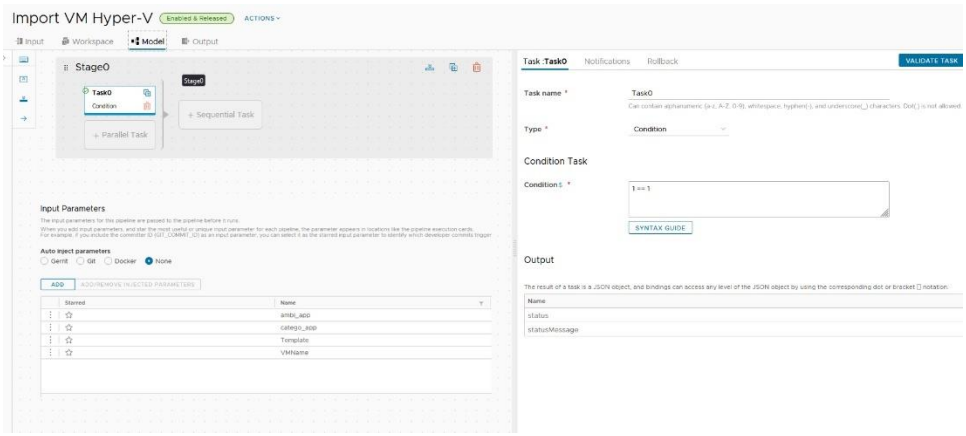


Figura 4.3 - Representação de uma *Pipeline* no *Code Stream*

Após a execução, é possível então importar a *pipeline* diretamente para o *Service Broker* para ser apresentado como uma ação no catálogo. Começamos então por aceder a *Content & Policies* e na aba *Content Sources*, especificando então o método de importação *Code Stream Pipeline*, o nome do *content source* e o projeto associado. De seguida, na aba abaixo *Content Sharing* após selecionar o projeto encontramos o *Content Source* criado anteriormente e é possível adicionar os itens que se encontram criados sobre o domínio do projeto selecionado, Figura 4.4.

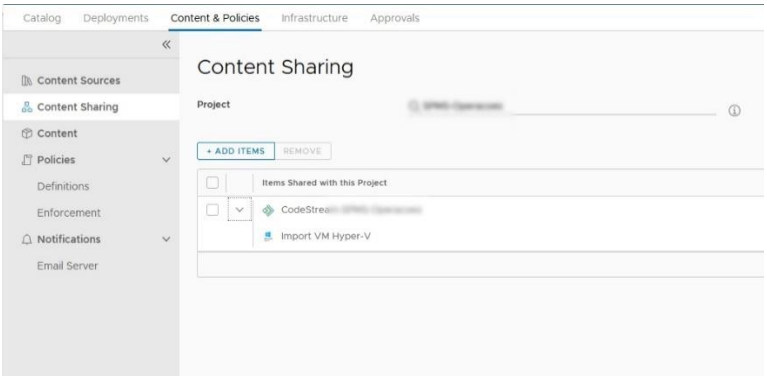


Figura 4.4 - Importação para o *Cloud Assembly* da *Pipeline*

A partir deste momento a *pipeline* já se encontra disponível no catálogo de ações podendo ser executado. Nele são apresentados, tal como se encontram identificados na *pipeline*, os parâmetros que o utilizador deve preencher em caixas de texto. Para melhorar o aspeto geral ao subscritor, a ação foi redesenhada no menu *Content &*

Policies, Content, onde se foi selecionada a opção *Customize Form*, permitindo assim a alteração do formulário. No presente caso, e como ilustrado na Figura 4.5, foi adicionado uma mensagem informativa relativamente aos utilizadores desta ação (na escrita do presente documento apenas se encontra disponível para os administradores da plataforma), assim como algumas regras de negócio que obrigam o nome do *deployment* (*deployment name*) a importar o texto preenchido no campo "VM Name", ou a seleção automática do projeto. Existem, no entanto, diferentes elementos que podem ser usados para especificar os diferentes parâmetros, como por exemplo caixas de texto, listas *dropdown* ou caixas de palavras-passe consoante os requisitos que o parâmetro exige.

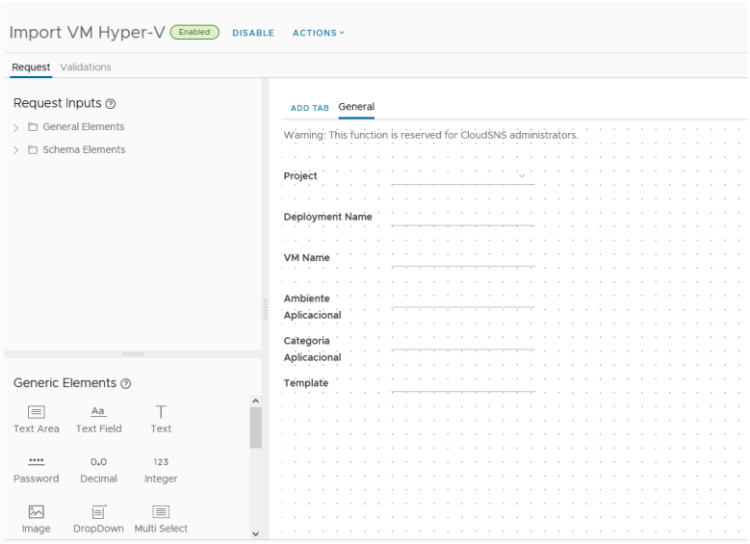


Figura 4.5 - Menu de customização de formulários

De seguida, e de modo a limitar as ações que os subscritores têm na plataforma, foi necessário adicionar regras de ações. É necessário ter em mente que os subscritores terão o título de *member* e os administradores terão o título de *Administrator*. É então necessário aceder ao *Service Broker*, *Content & Policies*, *Policies* e *Definitions* e criar três políticas. A primeira destina-se a controlar as ações a que os subscritores podem visualizar e interagir ou *Day 2 Actions*[28], especificando o projeto em questão, o tipo de utilizadores, as ações às quais possuem acessos e o tipo de execução, como ilustrado na Figura 4.6.

[28] Day 2 Actions ou ações de segundo dia, são ações inerentes a alterações que são efetuadas após a criação de uma máquina ou de uma representação lógica da mesma. Tem-se como ações relacionadas as alterações de recursos, ativação ou desativação de funções, etc.

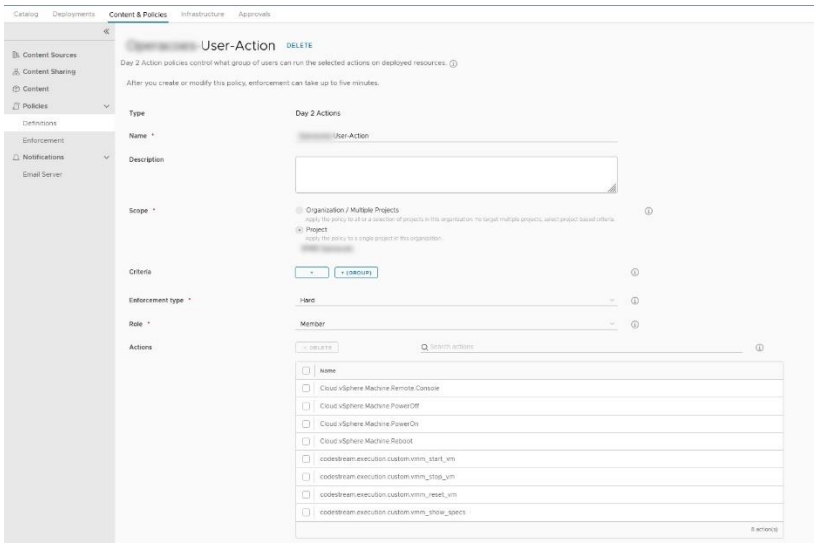


Figura 4.6 - Exemplo de política

A segunda política destina-se à criação de mecanismos de aprovação de funções disponibilizadas ou *Approval Action*. Do mesmo modo que foi ilustrado na Figura 4.6, é possível também especificar o projeto, o tipo de aprovação (se a nível do utilizador ou função), o modo de aprovação (se todos os administradores necessitam de aprovar ou se apenas um), os administradores que aprovam, a decisão por defeito (aprovar ou rejeitar), a data por defeito para a ativar a decisão por defeito (em dias) e as ações que estão sujeitas à aprovação. Assim, é garantido que o utilizador pode requerer certas ações, no entanto estão sempre salvaguardadas pela aprovação dos administradores, sendo esta política especialmente importante para garantir a gestão dos recursos da infraestrutura, evitando o sobredimensionamento.

Por último, é necessário criar uma política para permitir aos administradores terem acesso a todos os recursos do projeto. Note-se que por defeito, todos os intervenientes têm acesso aos recursos do projeto, no entanto com a implementação da primeira ação, todos os utilizadores passam a ter apenas acesso aos recursos selecionados. Para colmatar, é necessário criar uma política do tipo *Day 2 Action*, que se destine ao projeto e aos utilizadores que são administradores, especificando nas ações a opção "*" que permite selecionar toda e qualquer ação do projeto.

Assim, é possível garantir que os subscritores só possuem acesso às ações selecionadas e que certas ações necessitam de validação por parte dos administradores, enquanto garantem aos administradores ter acesso a todas as ações como anteriormente.

Por último, tendo em mente as necessidades de importar milhares de máquinas em *Hyper-V* para a plataforma e também como complemento ao trabalho desenvolvido

no caso de uso, foi desenvolvido um *script* externo à plataforma que permite importar a partir de um ficheiro do tipo CSV as máquinas e os seus parâmetros para a plataforma, assim como ao mesmo tempo aprovar as ações requisitadas (explicado anteriormente a sua implementação). A implementação da ferramenta de auxílio foi desenvolvida em *python*, recorrendo às funções disponíveis por API da plataforma *vRealize*, assim como funções não documentadas da mesma. Até ao presente momento o *script* permitiu a importação de mais de quinhentas máquinas, sendo agilizado o processo de importação, garantindo a coesão de configurações, assim como permitindo a documentação das máquinas importadas por iteração.

4.3 Ligar/Desligar Máquinas (*Hyper-V*)

Como descrito anteriormente, já é possível possuir uma precessão lógica sobre as máquinas que se encontram no *Hyper-V*, para além também de ser possível visualizar alguns parâmetros predefinidos e atribuídos aos *deployments*. No entanto, é clara a necessidade de possuir um modo de conectividade que permita a interatividade entre o subscritor do projeto e a máquina em questão. Tendo em conta que grande parte das ações tomadas nas máquinas virtuais passam por alterar o seu estado ou ajustar os seus recursos, é necessário oferecer essa mesma possibilidade ao subscritor final, sendo assim também possível a cedência dessa responsabilidade por parte da equipa técnica para o subscritor.

Com isto em mente, o primeiro passo passou por olhar para as possibilidades e para as dependências criadas. Assim, começando pela interação da plataforma com o *Hypervisor*, foi necessário no *Orchestrator* a criação de um *workflow* que interaja a máquina em questão, enviando um sinal para ligar ou desligar.

Tendo em conta que o elemento de identificação da máquina em *Hyper-V* é o nome da mesma, o *script powershell* foi desenvolvido com base no mesmo. Como ilustrado na Figura 4.7, os *scripts powershell* possuem uma linguagem simples e intuitiva, que são facilmente manipulados para conterem os campos desejados.

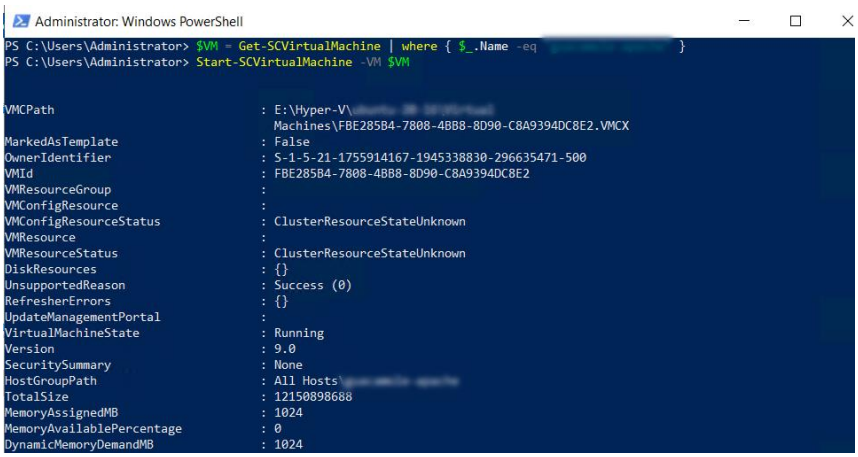


Figura 4.7 - Exemplo de *script powershell* para desligar máquinas em *Hyper-V*

Como descrito anteriormente, a possibilidade de *scripting* da plataforma *Orchestrator* em conjunto com a sua capacidade de conectar diretamente com via *powershell* a máquinas com sistema operativo *Windows*, possibilitam em primeiro lugar manipular às necessidades o *script powershell* que será executado e de seguida executar o mesmo. No caso do *Orchestrator* não lançar exceções podemos concluir com certeza que a execução decorreu como planeado e que a máquinas virtual se encontra ligada/desligada.

Assim, é possível, como ilustrado na Figura 4.8, a criação de um *workflow* que possui como *input* o nome da máquina a interagir, e que após validações e formulações de *script*, executar efetivamente uma ação espoletada pelo subscritor. No presente caso, a única validação relevante foi a verificação se a máquina já se encontra ligada ou desligada, uma vez que a máquina não pode ser regida a um estado em que já se encontra, sendo, no entanto, de notar que esta validação deriva diretamente de um processo identifico de execução, manipulação e interpretação descrito no presente *workflow*.

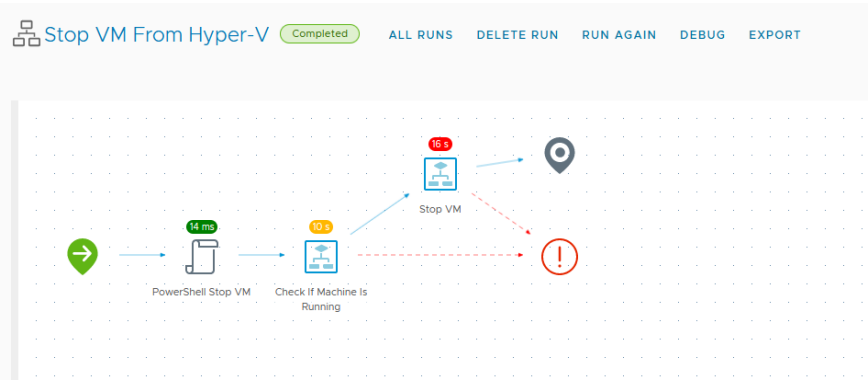


Figura 4.8 - *Workflow* de desligar máquinas em *Hyper-V*

Após a criação do *workflow* é necessário transpor para o subscritor a opção de mudança de estado das máquinas, sendo assim necessário intervir no *Cloud Assembly*. Tal como nas funções nativas para interagir com o *VMware*, o objetivo é também disponibilizar funções idênticas, pelo que foram selecionadas as *Resource Actions* para tal. Estas, disponibilizam ao subscritor a opção de interação personalizada com um nome ilustrativo da ação que, após ativo, espoleta a execução do *workflow* anteriormente criado no *Orchestrator*. Como presente na Figura 4.9, é possível selecionar o tipo de ação à qual vai ser disponibilizada a ação no campo *Resource Type* e selecionado o *workflow* que será executado. A customização das *Resource Actions* pode ser muito detalhada e inclusiva, sendo explorada em mais detalhe no decorrer do documento.

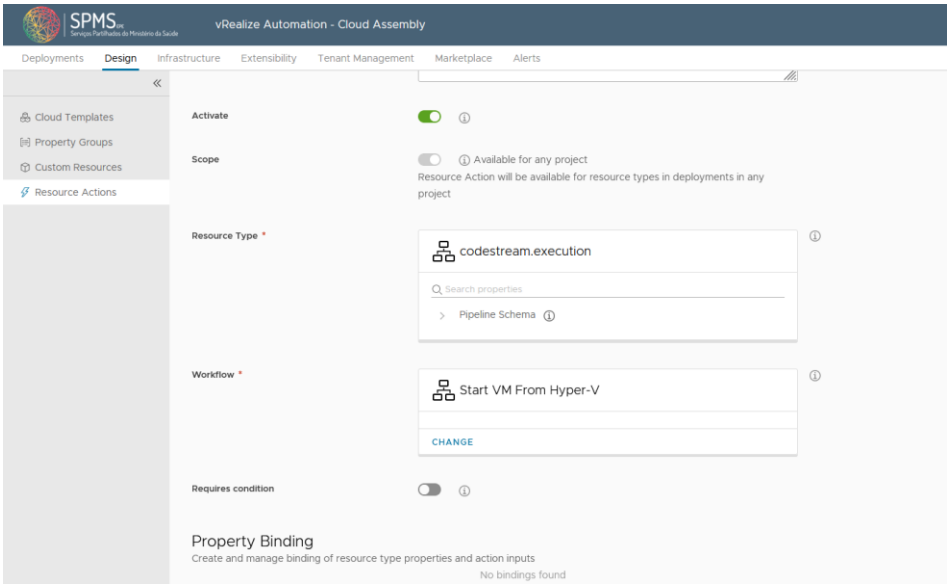


Figura 4.9 - Representação da configuração de uma *Resource Action*

Por fim, é importante salientar que as execuções espoletadas pelas *Resource Actions* integram em si parâmetros de meta dados úteis para a interligação dos módulos no *vRealize*, possibilitando, por exemplo, a recolha do nome da máquina do *deployment* no decorrer da execução do *workflow*. A apresentação final disponibilizada ao subscritor, como ilustrado na Figura 4.10, apresenta um *layout* idêntico às ações fornecidas nativamente, possibilitando assim a iteratividade direta entre o subscritor e a máquina, sem necessidade de intervenção de terceiros, enquanto garante a segurança nos acessos aos *hypervisors*.

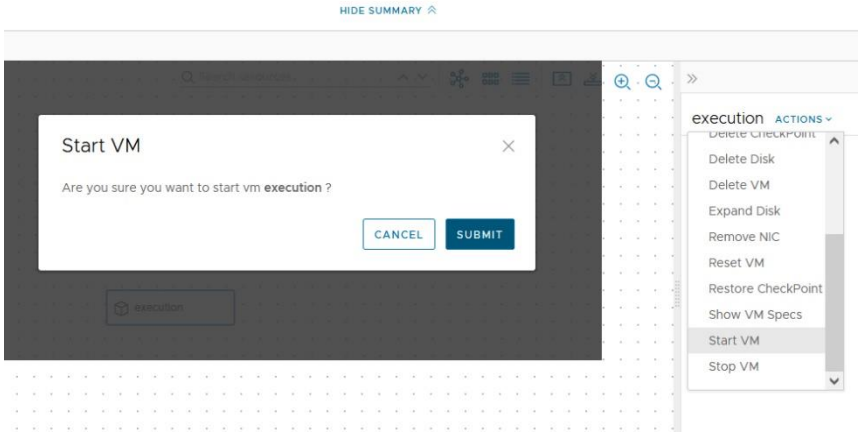


Figura 4.10 - Execução da ação desenvolvida

4.4 Eliminação de Máquina Virtual

O processo de eliminação de máquina é em tudo idêntico à criação de uma máquina virtual, estando no entanto, obviamente, os processos invertidos. É importante retroceder os processos de criação, de modo a garantir coerência no sistema, como também a confiança do administrador na fiabilidade da documentação criada e do estado de máquinas ou disponibilidade de recursos.

No caso das máquinas criadas diretamente em *VMware*, o processo é simplificado, sendo nativo o controlo de recursos, possibilitando o retorno dos mesmos ao saldo do projeto sem qualquer necessidade de intervenção por terceiros. No entanto, a reserva dos IPs no IPAM (3.1 - Contextualização), foi um processo personalizado, pelo que é necessária a implementação de mecanismos de reversão.

Do mesmo modo que foram usadas subscrições para capturar a ação de alocação de recursos, também é possível definir uma interação caso sejam capturadas ações de eliminação de recursos computacionais. Como ilustrado na Figura 4.11, aquando da captura do evento de remoção de recursos, é inicializado o *workflow* de eliminação de IP do IPAM.

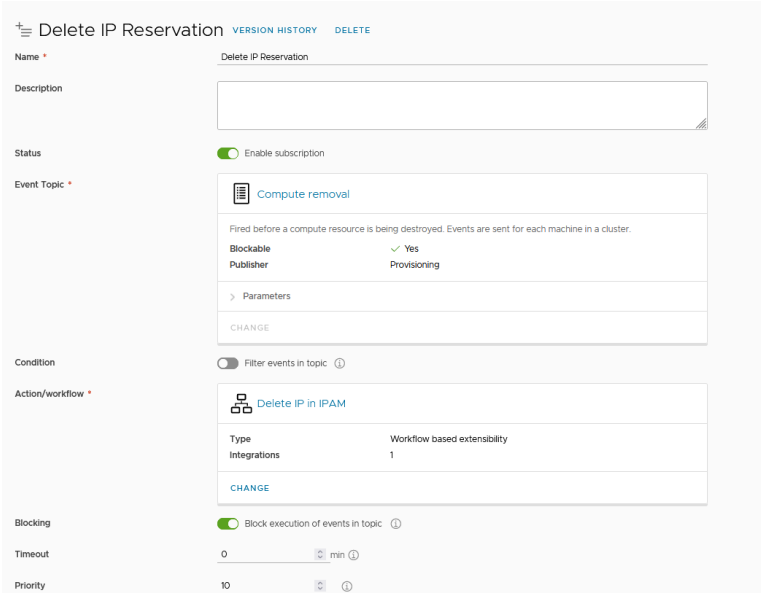


Figura 4.11 - Captura da ação de eliminação de recursos e eliminação da reserva de IP

Os parâmetros de *deployment* eliminados são enviados para o *workflow*, sendo possível recolher o ambiente e categoria aplicacional da máquina, que permite a partir do mapeamento deduzir a rede onde foi efetuada a reserva, e a partir do IP da máquina presente nos parâmetros, indicar qual a entrada que necessita de ser eliminada. O processo de eliminação é via API, no qual é necessário efetuar um pedido do tipo *DELETE* com a informação do */addresses/{ip}/{subnetId}/* preenchida com a informação recolhida.

Deste modo, é possível efetuar a eliminação na plataforma de reserva de IPs, mantendo a documentação atualizada a todo o momento. É importante salientar que a eliminação do domínio de *broadcast* criado no ACI (3.1 - Contextualização) não é, no entanto, eliminada, uma vez que pode possuir dependências cruzadas que necessitam de ser avaliadas com o devido cuidado, ao contrário do proposto inicialmente.

Por outro lado, nos *deployment* de máquinas *Hyper-V* é necessário a criação de uma *Day 2 Action*, idênticas à de Ligar/Desligar máquinas virtuais. A ação apresentada ao subscritor está associada a um *workflow* dedicado à eliminação de máquinas.

O processo de eliminação passa por quatro fases distintas. Como ilustrado na Figura 4.12, o primeiro processo passa por eliminar a máquina do IPAM, seguindo os mesmos princípios descritos anteriormente. De seguida, é calculado o valor da máquina em questão e devolvido para o saldo do projeto em forma de *tokens*. A próxima fase é de eliminação da máquina virtual do *hypervisor*, sendo invocada uma ação de desligar a máquina e de seguida de eliminação por completo da mesma. Por último, para

possibilitar a eliminação do objeto lógico do *deployment*, é necessário aceder via API ao *vRealize* e proceder à eliminação do mesmo de modo definitivo.

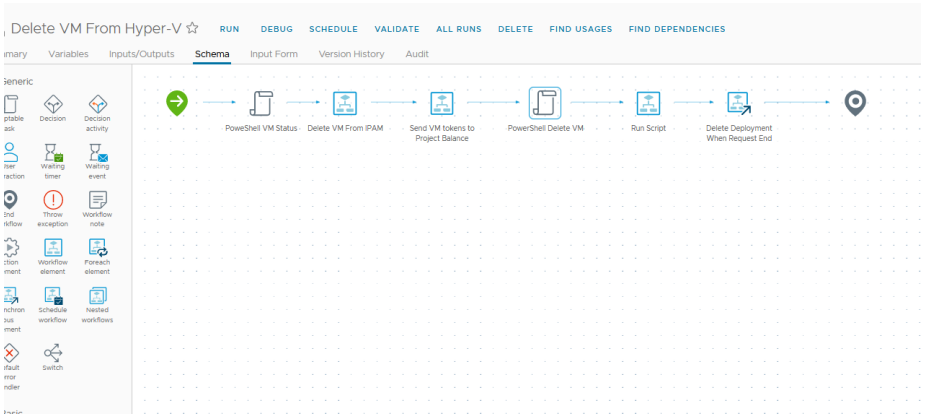


Figura 4.12 - *Workflow* de eliminação de máquina *Hyper-V*

Após espoletar a ação o subscritor poderá acompanhar a barra de processo da ação, sendo, no entanto, reportado um erro quando o *deployment* é eliminado, sendo este originado pela eliminação e consequente inexistência do mesmo.

4.5 Alteração de parâmetros das máquinas (em *Hyper-V*)

Ao contrário dos casos de uso abordados até ao momento, a alteração de parâmetros numa máquina virtual obriga à troca de informação dinamicamente entre o subscritor e o sistema. A disposição de informação sobre o sistema operativo e o modo como está a operar é um critério fundamental para prevenir falhas preventivamente e a possibilidade de alteração dos parâmetros das máquinas virtuais por parte dos subscritores aproximam a gestão das máquinas cada vez mais dos mesmos, libertando a equipa técnica.

Tendo em conta a descrição genérica do caso de uso, foi selecionado apena um como exemplo para demonstrar o processo de implementação dos mesmos, sendo possível encontrar na subseção Texto da secção Capítulo 8 - Apêndice a listagem das ações implementadas e uma breve descrição das mesmas.

De modo idêntico ao descrito nas implementações anteriores, inicializou-se o processo de implementação do caso de uso de alteração do número de CPUs pelo desenvolvimento de um *script* em *powershell* que permita executar a operação. Como ilustrado na Figura 4.13, o *script* integra dois processos, o primeiro de verificação do estado da máquina uma vez que só é possível alterar o número de cores caso a máquina se encontra desligada. O segundo processo é a parametrização do *script* para agir na

máquina em questão, validando de antemão o número de CPUs selecionados pelo subscritor, e caso este não respeite os limites estabelecidos, uma mensagem de erro será espoletada.

Após recolher o estado da máquina, no objeto *Decision* é validado o estado em que a máquina virtual se encontra, e caso não esteja desligada será enviado um aviso ao subscritor para desligar primeiro a máquina antes de proceder à atualização das suas características.

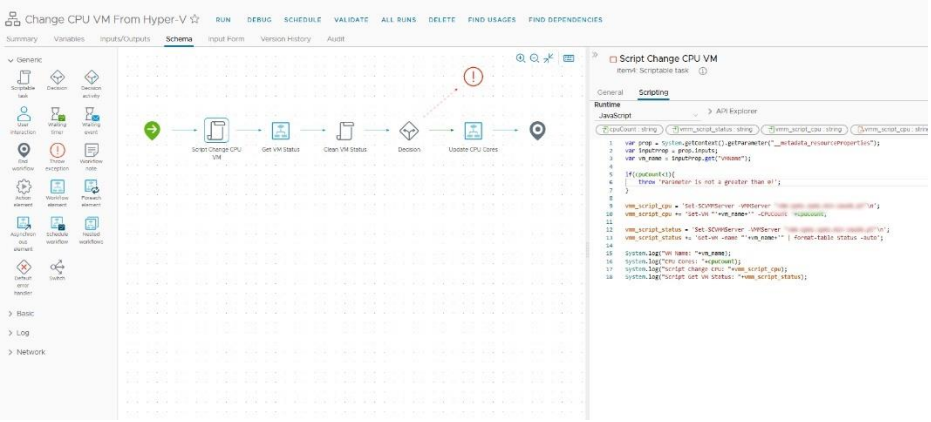


Figura 4.13 - Implementação de código para alteração de cores

Após a validação do estado da máquina o pedido é entregue no objeto "*Update CPU Cores*" que para além de espoletar a execução do *script powershell* previamente estruturado, altera também as cotas no módulo de *tokens* que será explorado no decorrer do documento.

Após a finalização do *workflow* para alterar os cores de uma máquina virtual, foi necessário explorar o melhor método para apresentar ao subscritor o caso de uso. A simples apresentação de uma caixa de texto ou um seletor de número ao subscritor, apesar de já representar um avanço no controlo das máquinas, não se torna espontâneo, uma vez que nem sempre existe a ideia do número atual de cores da máquina. Assim, para melhorar a experiência de uso, antes de ser espoletado o *workflow* descrito anteriormente, é necessário apresentar uma contextualização dos valores atuais.

Para visualizar primeiro os valores originais e só depois alterar para os valores selecionados pelo subscritor, foi necessário implementar um novo *workflow* só de recolha de dados da máquina virtual. A implementação deriva de outras recolhas de dados do *Hyper-V*, recebendo como *input* o nome da máquina e retornando o número de cores da mesma. O processo de recolha de dados (que ainda não foi referenciado no presente documento), passa pela conversão do output *powershell* em formato JSON a partir do método nativo *getAsJson()*, convertido de seguida para um objeto JSON, como

ilustrado na Figura 4.14. Durante a implementação do método de captura de resultados, foi constatada uma lacuna na documentação do *vRealize* quanto à utilização e funcionamento dos métodos, pelo que foi necessário (e recomendado para novas implementações) o desenvolvimento sequencial dos *workflows*. No caso da recolha de dados é necessário extrair toda a variável JSON e explorar em que chave se encontra o objeto que desejamos para assim efetuar o mapeamento necessário.

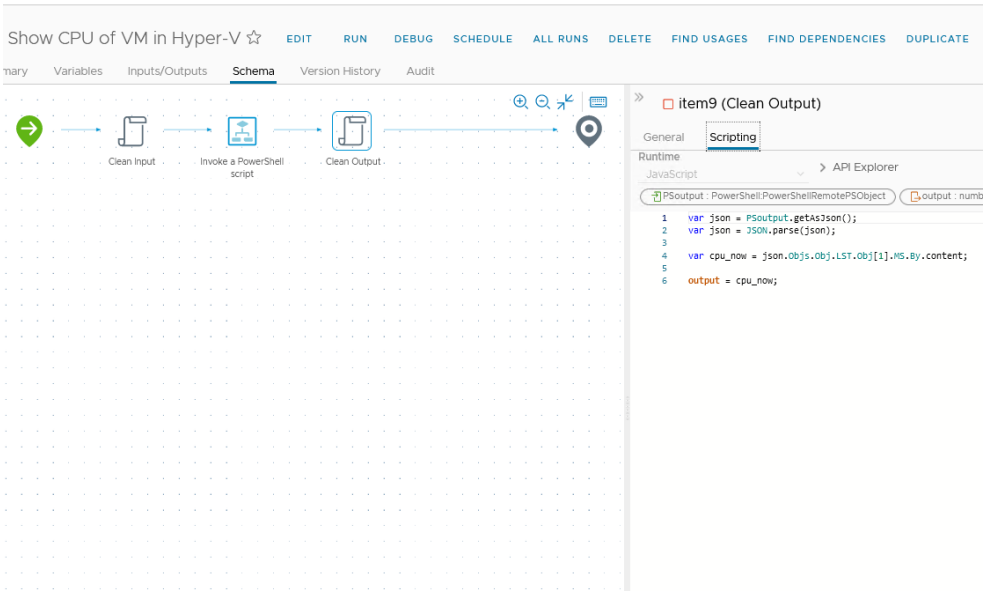


Figura 4.14 - Recolha e tratamento de output *powershell*

Com os dois *workflows* concluídos é assim possível inicializar o processo de integração final. Como ilustrado na Figura 4.15, o processo deriva de funções e ferramentas já exploradas anteriormente, no entanto mais complexo e separado por duas etapas. É também introduzido o uso das *Actions* que representam um importante elemento no funcionamento do caso de uso.

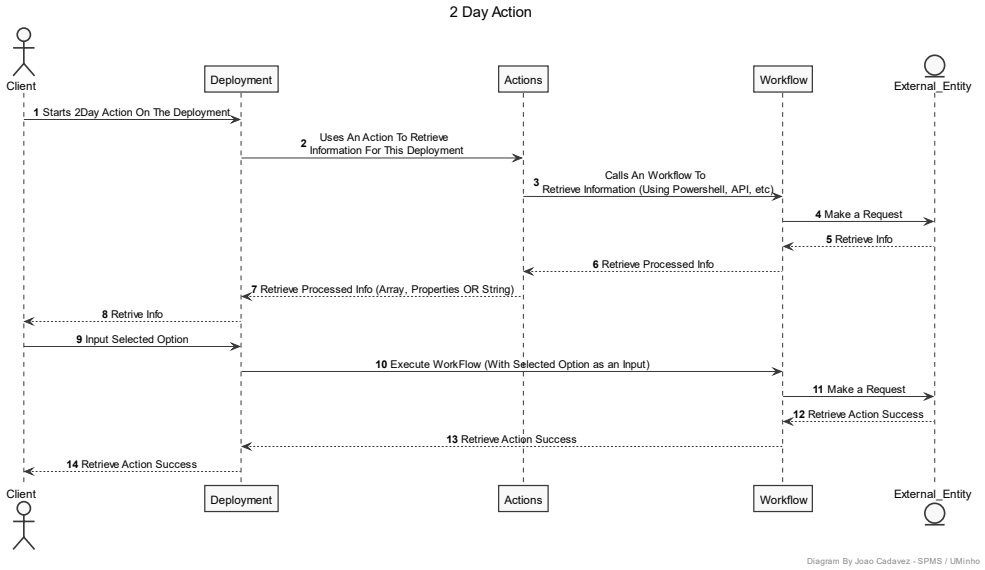


Figura 4.15 - Diagrama de sequência genérico de ações com valores dinâmicos

Associado ao *workflow* de recolha e apresentação de dados, foi necessário na aba de personalização das *Resource Actions* encontrar uma opção que possa executar o *workflow*, recolher a informação e apresentá-la de forma facilmente percetível ao subscritor. Assim, como apresentado na Figura 4.16, a utilização de recursos do tipo *External Source*, permite a evocação de apenas *Actions* do *Orchestrator*, que por si normalmente são apenas utilizadas para simples *scripts* ou para guardar variáveis gerais a vários *workflows*.

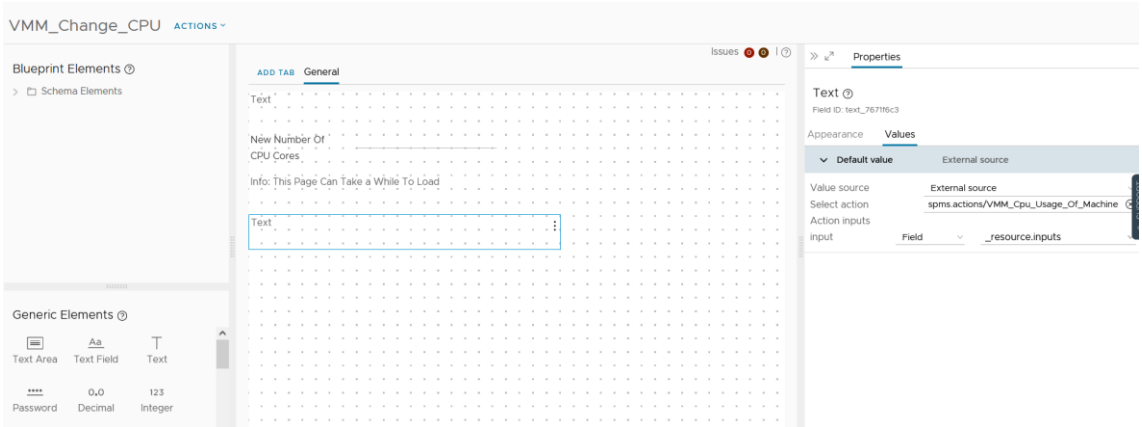


Figura 4.16 - Exemplo de evocação de recursos *External Source*

É assim necessário configurar a *Actions* a executar, processar e retornar os dados da execução de um *workflow* para deste modo possibilitar a apresentação dos dados ao subscritor. No entanto, como já referido na secção *Actions*, a utilização das

mesmas para executar *workflows* não é recomendada pela comunidade da plataforma, uma vez que foge do seu propósito mais simplista e introduz mais entropia no sistema assim como dependências que se podem revelar um problema no futuro da solução.

A implementação invulgar requer a utilização do código genérico representado na Figura 4.17, e que a partir do identificador do *workflow* (que é possível obter na aba *summary* do *workflow*) e do *array wfInput* com as variáveis de invocação do *workflow*, é possível executar o *workflow* e retornar na variável *output*.

É, no entanto, importante salientar que durante a implementação deste modelo foi identificado um erro de programação no código do *vRealize*. Aquando da seleção das variáveis de *input* da *Action* é necessário selecionar do tipo *properties* que corresponde ao tipo de variável que o *deployment* possui, sendo necessário de seguida na *Resource Action* selecionar os dados. Caso o tipo de *input* da *Action* se encontre como *properties* a plataforma não consegue interpretar o mesmo, sendo necessário, já depois de selecionados os dados na *Resource Action*, alterar o tipo de *input* da *Action* para *String* e interpretar como texto em formato JSON.

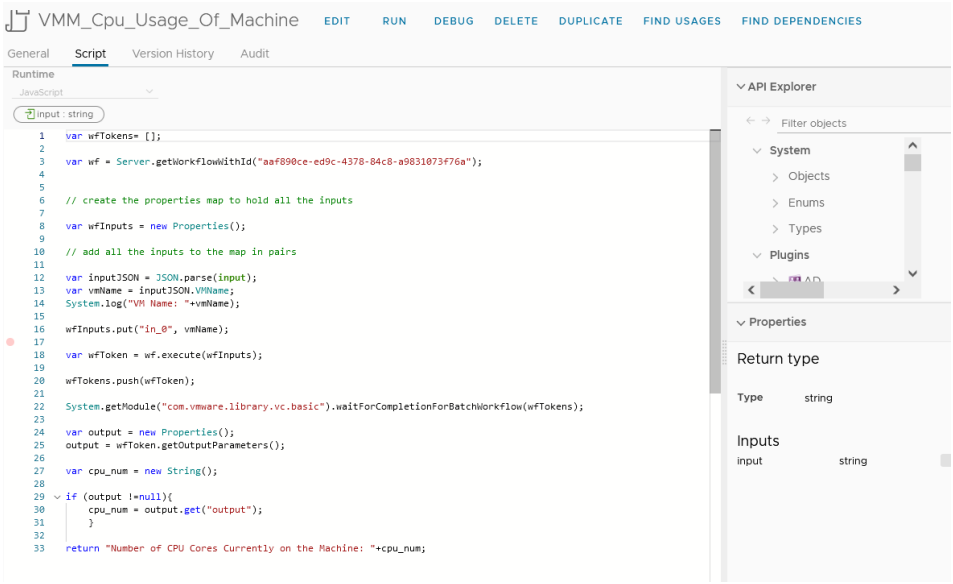


Figura 4.17 - Execução de *workflows* com *Actions*

Com a *Action* implementada e a executar o *workflow* de recolha dos dados informativos da máquina, é possível concluir o desenho da *Resourse Action* bastando selecionar o *workflow* de alteração de cores. Assim, como ilustrado na Figura 4.18, a *Resource Action* já apresenta o número atual de cores da máquina antes de possibilitar a alteração dos mesmos, respondendo assim às exigências do caso de uso.

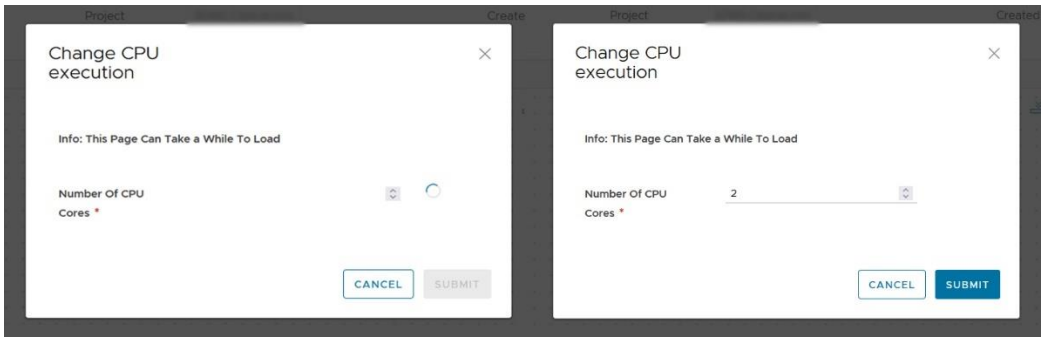


Figura 4.18 - Processo de alteração dos cores da máquina virtual

4.6 Criação de máquinas

A criação de máquinas foi desde o início do projeto um dos casos de uso mais ressaltados pela poupança de tempo de operação estimada. Na solução *vRealize*, e tendo em conta a necessidade de integração com o *Hyper-V* várias vezes reforçada anteriormente, obrigando assim à divisão do caso de uso em duas partes, sendo primeiramente necessário usar as ferramentas nativas para possibilitar a criação de máquinas com sistemas operativos do tipo *Linux* em ambientes *VMware*, e máquinas com sistema operativo *Windows* no *Hyper-V*.

Começado pela implementação das máquinas do tipo *Linux*, usou-se as funções nativas da ferramenta. Como ilustrado na Figura 4.19, o primeiro passo passa pela criação de um novo *Design* no *Cloud Assembly*, podendo ser dedicado a um único projeto, ou partilhado por vários, consoante o âmbito do mesmo.

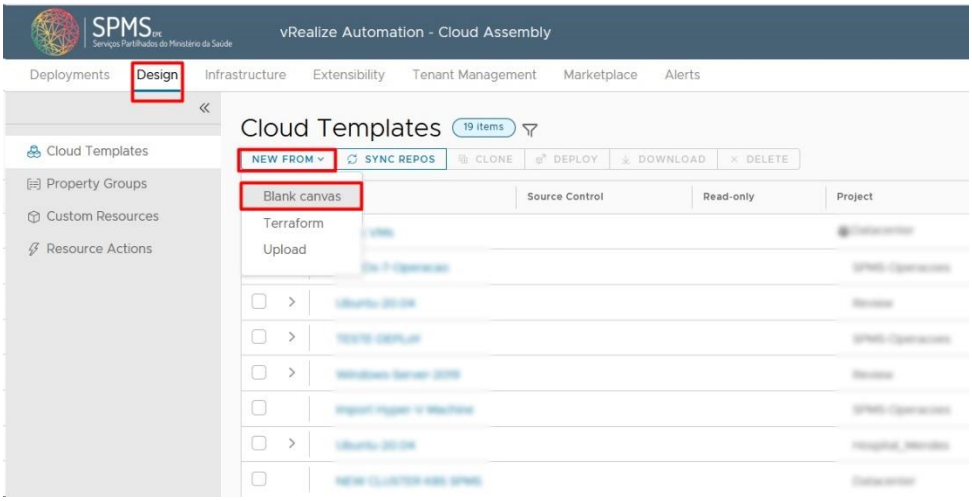


Figura 4.19 - Criação de um novo *Design*

Após a escolha dos parâmetros do *Design* é apresentada uma tela branca para definir os elementos e as suas relações, sendo possível encontrar inúmeros objetos diferentes na aba lateral direita, como ilustrado em Figura 4.20, que referenciam um objeto das diferentes plataformas integradoras como *AWS*, *Terraform* ou *vSphere* especificando, em cada objeto, os parâmetros que necessitam.

Do lado direito na aba *Code*, em formato YAML, é possível encontrar o código que a plataforma vai criando com a adição dos diferentes elementos. É também nesta aba que é possível alterar alguns parâmetros pré-definidos e introduzir opções de escolha ao subscritor para um ou mais parâmetros.

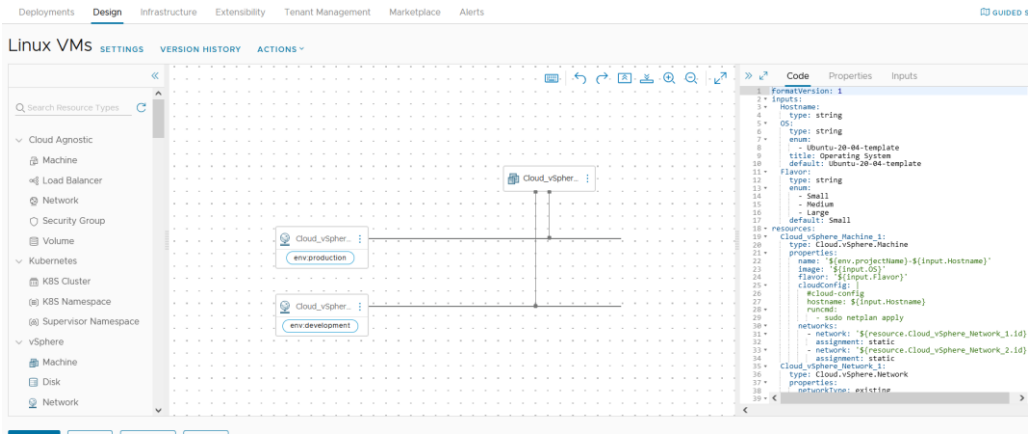


Figura 4.20 - Bancada de trabalho da opção *Design*

Pela diversidade de opções possíveis no presente caso de uso, foi necessário definir os principais e indispensáveis parâmetros a requisitar ao subscritor. Em primeiro o subscritor deverá especificar o nome da máquina que no código segue o padrão *${env.projectName}-${input.Hostname}-${input.Environment}-${inputCategory}* possibilitando mesmo que em projetos diferentes sejam selecionados nomes iguais, com o prefixo do nome do projeto no *hypervisor* a máquina seja única, em conjunto com o sufixo que ajuda a identificar o âmbito da máquina e, futuramente, permitir selecionar computação mais ou menos prioritária perante os parâmetros. Em segundo, o sistema operativo permite selecionar de uma listagem pré-definida, o *template* usado na criação da máquina que se encontra em antemão na biblioteca do *vSphere*. O campo *Flavor* permite selecionar um tamanho para a máquina em questão, ajudando subscritores com menos experiência em requisitos de *hardware* a terem a máquina virtual a funcionar sem conhecimento prévio ajudando inclusive a controlar o aprovisionamento excessivo de recursos a uma só máquina. Por último, o *Environment* e o *Ambient* possibilitam a contextualização da máquina no projeto e nos recursos, para além de possibilitar a atribuição de um IP estático na rede especifica para o âmbito da máquina.

Deste modo é possível criar uma máquina virtual com a utilização de um simples painel, ilustrado na Figura 4.21, de fácil interpretação para o subscritor. A personalização do painel é elevada, sendo possível adicionar regras de negócio aos *inputs*, diferentes tipos de painéis ou sequenciais dependendo dos *inputs* anteriormente inseridos.

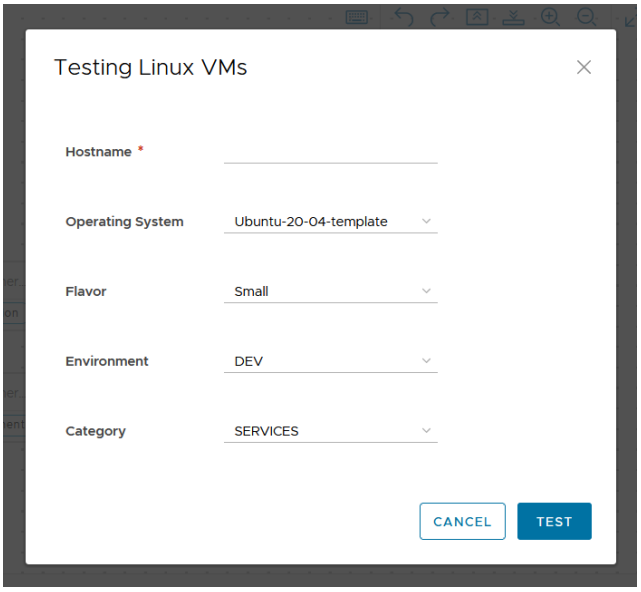


Figura 4.21 - Menu de criação de máquina virtual *Linux*

Para possibilitar a personalização da máquina virtual com o IP estático definido, assim como os parâmetros inerentes aos mesmo, ou a alteração do nome da máquina no sistema operativo, foi usado o mecanismo *Cloud-Init*. Este, definido na criação de cada um dos *templates* e personalizado com os parâmetros do subscritor que, ao momento de inicialização da máquina, são injetados surtindo efeito imediato dos mesmos. Como representado na Figura 4.22, por defeito é inserido o nome que o subscritor definiu como *hostname* e impressa uma mensagem de início de processo. No entanto, como o processo de decisão do IP depende do ambiente e da categoria aplicacional, é necessário completar o presente código com essas informações, sendo para tal usado os *workflows* do *Orchestrator*.

```
39      flavor: '${input.Flavor}'
40      cloudConfig: |
41        #cloud-config
42        hostname: ${input.Hostname}
43        runcmd:
44          - echo "Clod-Init Process Beginning"
45      networks:
```

Figura 4.22 - Exemplo de código *Cloud-Init*

Para capturar os eventos de execução inerentes à criação de máquinas em *vSphere*, foram utilizadas as *Subscritption*, como ilustrado na Figura 4.23, que estando associadas a um tipo de evento uma *Action* ou um *workflow* no *Orchestrator* ao mesmo tempo que são enviados os parâmetros do *deployment* que espoletou o mesmo. Para o presente caso de uso é importante garantir que a opção *Block execution of events in topic* se encontre ativa para bloquear a conclusão do *deployment* enquanto o *workflow* se encontra em execução.

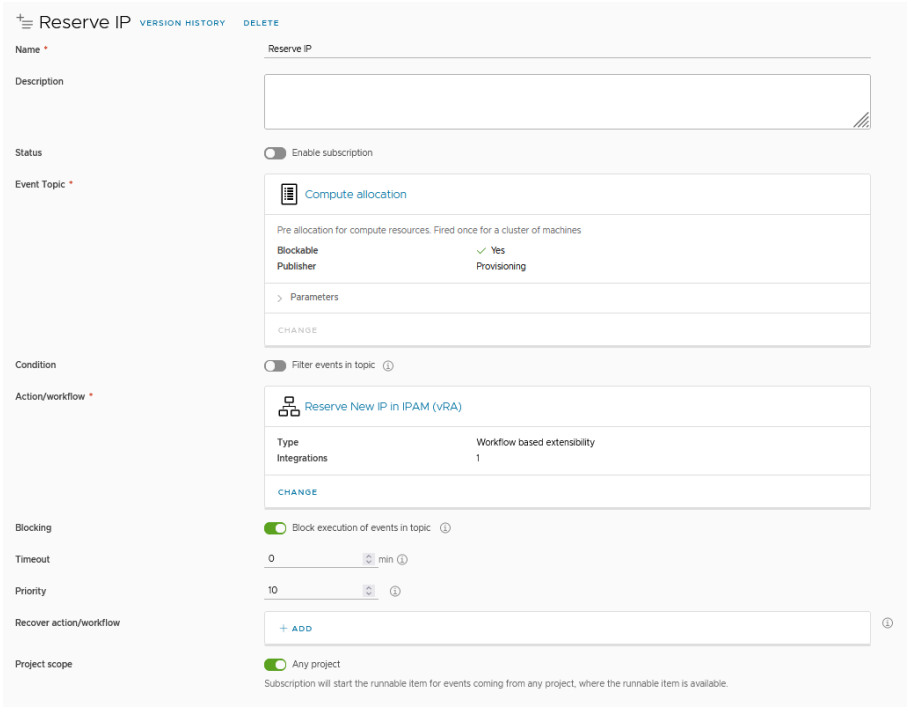


Figura 4.23 - Criação de uma *Subscription*

Com o *design* e a *subscription* criada, o próximo passo é manipular o código *Cloud-Init* definido inicialmente no *design*. Garantindo a entrega das variáveis num *workflow no Orchestrator*, a manipulação dos dados de *input* é garantida pela importação da variável *inputProperties* do *design* original, que sendo tratada como um objeto do tipo *Properties*, é possível extrair o valor original do *design*. De seguida, e como ilustrado na Figura 4.24, é também extraído do conjunto de configurações originais o tipo de imagem que o subscritor selecionou antes de espoletar a ação, permitindo assim com o objeto *switch* redirecionar para um *workflow* que garanta elaborar o conjunto de comandos necessários para alterar as opções de rede para cada distribuição de *Linux* disponibilizada.

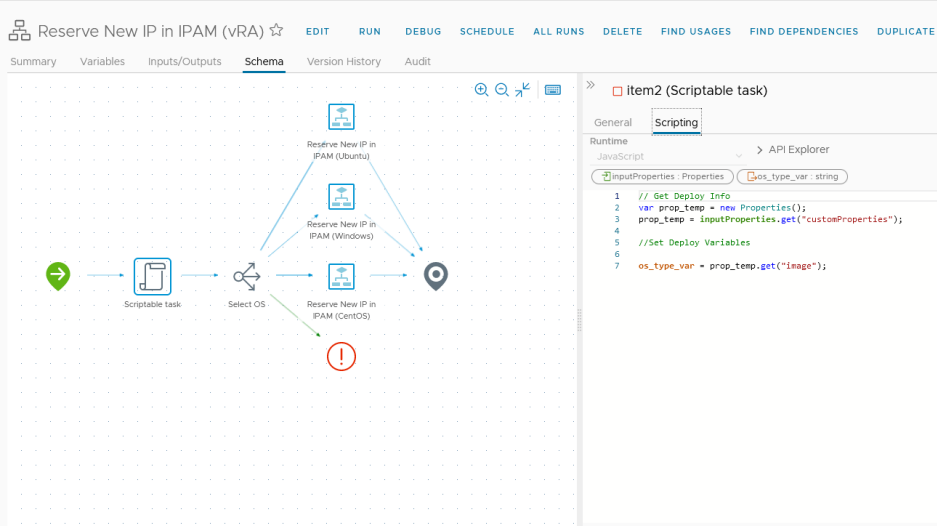


Figura 4.24 - *Workflow* de reserva de IPs

No entanto, apesar de se diferenciarem no modo como os comandos são constituídos, o conjunto final de comandos derivam todos do conjunto de parâmetros retornados pelo IPAM. Como referido anteriormente, o subscritor necessita de selecionar o ambiente e a categoria da máquina, sendo esses parâmetros usados a priori do pedido ao IPAM para selecionar a rede em que a máquina irá ser registada. Para tal, foi criada uma *Action* com o mapeamento para cada par de possibilidades ambiente/categoria e associado a um identificador da rede do IPAM, possibilitando a seleção da rede tendo em conta o âmbito da máquina. Em seguida, e como representado na Figura 4.25, e tendo em mente que é necessário o registo da API do IPAM e das operações do mesmo, sendo de seguida iniciada a comunicação entre o *Orchestrator* e o IPAM. Primeiramente é necessário recolher os dados da rede como os *DNS* e a *default gateway*, em segundo recolher a máscara de rede, e em último registar os dados da máquina virtual criada, assim como a data de criação da mesma no IPAM. Deste modo, o último passo é o processamento dos dados recolhidos e consequente criação do código associado ao *Cloud-Init* (este difere consoante o sistema operativo de cada máquina virtual), que é retornado no final do *workflow* para o *deployment* no campo *customProperties* do mesmo.

Como anteriormente referido, forçou-se a espera da *subscription* pela conclusão do *workflow* sendo assim possível, ainda antes de ter sido inicializado a implementação no *vSphere*, a injeção do *Cloud-Init* no novo *deployment* do *template* previamente definido.

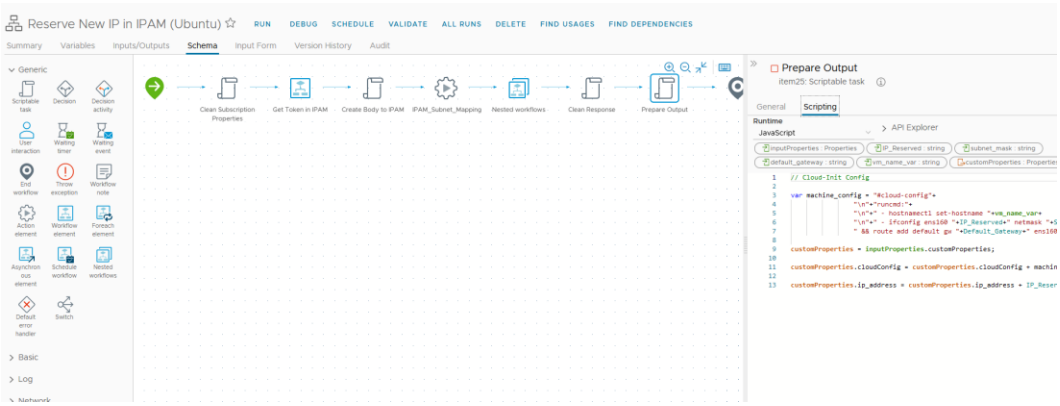


Figura 4.25 - *Workflow* de registo da máquina no IPAM

Do mesmo modo que foi possível reservar e documentar o IP para a máquina virtual, é também possível criar uma placa de rede para a máquina, caso não exista. Tomemos o exemplo um subscritor que cria uma máquina com o ambiente DEV e categoria APP no projeto Y, antes do momento de criação da máquina no *vSphere*, o *workflow* garante a criação, caso já não tenha sido criado para outra máquina do mesmo âmbito e do mesmo projeto, do novo domínio de *broadcast* no ACI com o nome Y-DEV-APP, sendo apenas necessária a integração da nova máquina com a placa Y-DEV-APP.

Para possibilitar a integração, foi criado um *workflow* no *Orchestrator* capaz de validar a existência do domínio de *broadcast* no ACI e gerá-lo caso o mesmo não exista. É associado de seguida o domínio de *broadcast* à máquina requerida ainda antes da mesma ser criada. Como ilustrado na Figura 4.26, o processo de validação da existência do domínio de *broadcast* no ACI passa, em primeiro plano, por integrar a API no *vRealize*, em seguida, é necessário recolher todos os domínios de *broadcast* do ACI e verificar se existe algum com as mesmas propriedades do âmbito em questão. Caso se verifique passamos ao passo de associação da placa de rede com a máquina virtual solicitada.

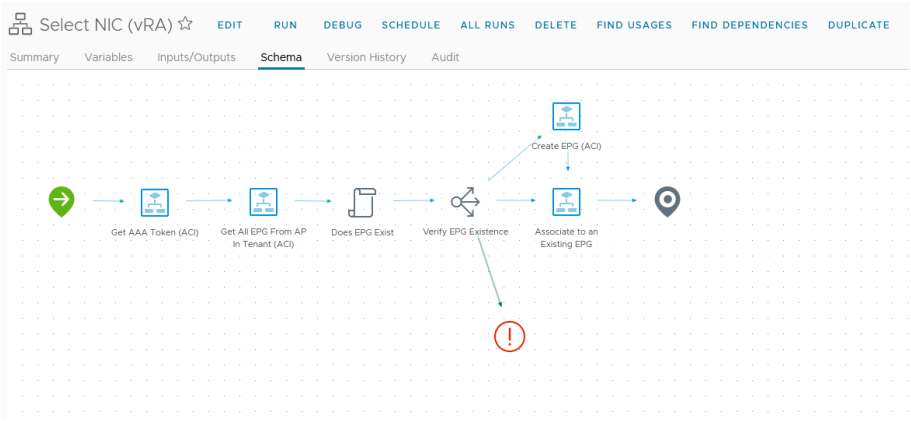


Figura 4.26 - Seleção da placa de rede / criação

Após a criação do domínio de *broadcast* é então necessário associá-lo ao *Network Profile* do projeto no *Cloud Assembly*. Para possibilitar a autonomia deste processo, foi criado o *workflow* presente na Figura 4.27, um conjunto de interações entre o *Orchestrator* e o *Cloud Assembly* que possibilitam num primeiro instante a integração com o *vSphere*, seguido da importação da nova placa de rede para o *Network Mapping* do projeto. Aquando da importação, na nova placa é especificada a marcação com a sequência idêntica ao nome do domínio de *broadcast* criado (Y-DEV-APP). Assim, e como ilustrado na Figura 4.28, é possível definir no *design* o tipo de etiqueta a utilizar aquando da seleção da placa e no momento de criação da máquina que, neste caso, ocorre após a conclusão dos *workflows*.

Figura 4.27 - Associação de um novo domínio de *broadcast* a um *Network Profile*

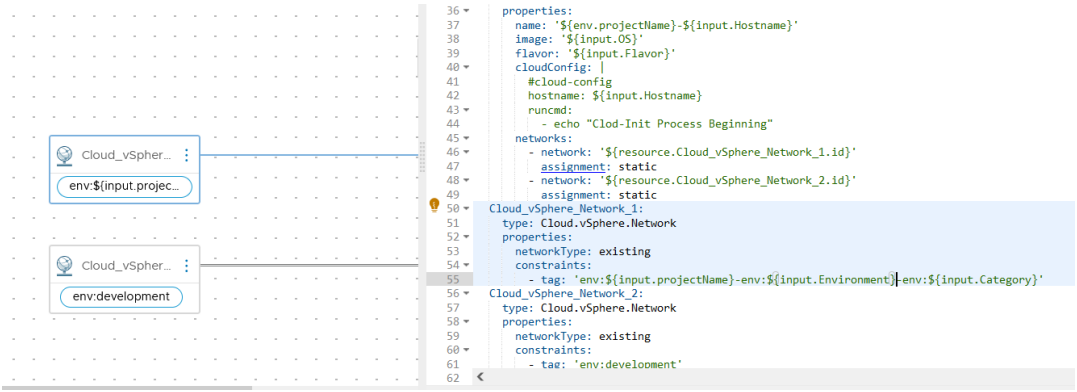


Figura 4.28 - Especificação da etiqueta de rede

Em modo de conclusão, com a utilização das *subscriptions* para garantirem o processamento e aprovisionamento dos recursos antes da criação da máquina virtual. Aquando da execução do *design*, é possível garantir não só a reserva e documentação da

máquina virtual na ferramenta IPAM (Figura 4.29), assim como garantir a retenção do domínio de *broadcast* com recurso à ferramenta de SDN da *Cisco*.

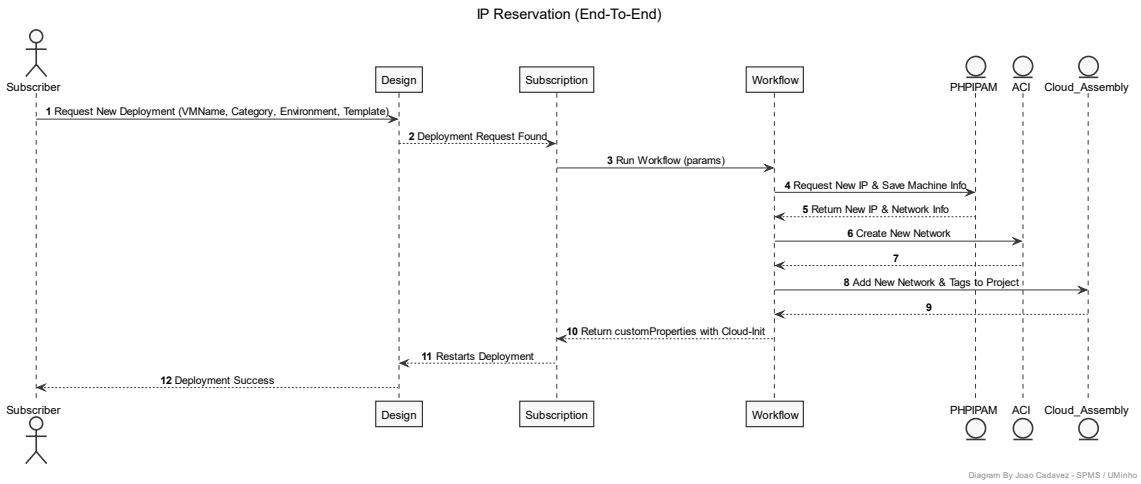


Figura 4.29 - Diagrama de sequência do processo de reserva e atribuição de IPs

No entanto, este processo só se aplica a máquinas que se encontrem em ambientes *VMware*, pois para máquinas *Windows* em ambiente *Hyper-V* são necessárias algumas alterações ao paradigma anteriormente explorado. Como ilustrado na Figura 4.30, o processo apresenta-se menos complexo do que o anteriormente explorado, fazendo uso principalmente da *Orchestrator* para executar o *workflow* de manipulação de *scripts powershell* e interação com os diferentes módulos dependentes para a criação da máquina sobre as regras de negócio estabelecidas inicialmente.

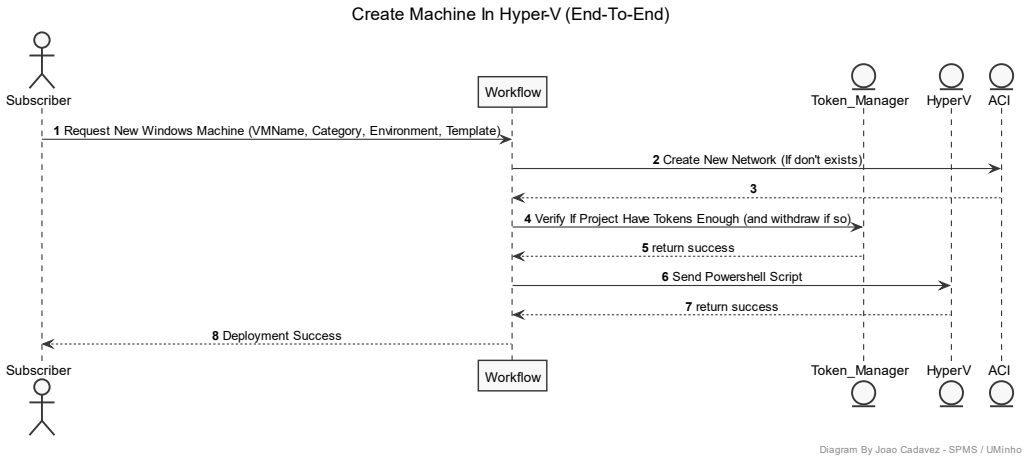


Figura 4.30 - Criação de máquina virtual em *Hyper-V*

Como anteriormente exemplificado, começamos o processo de implementação pelo *Orchestrator* criando um *workflow* responsável pela criação de uma máquina virtual. Definindo no *input* os campos indispensáveis para a criação de uma máquina virtual, que por coerência são em tudo semelhantes aos campos requeridos no caso das máquinas em *vSphere*, é necessário escrever um *script* em *powershell* para a criação da máquina virtual. Como ilustrado na Figura 4.31 o *workflow* principal é responsável por receber o pedido e garantir, antes de executar o *powershell* no *Hyper-V*, a existência da rede para a máquina virtual, para além da verificação se a equipa possui *tokens* suficientes para o pedido efetuado (discutido em pormenor no decorrer do documento).

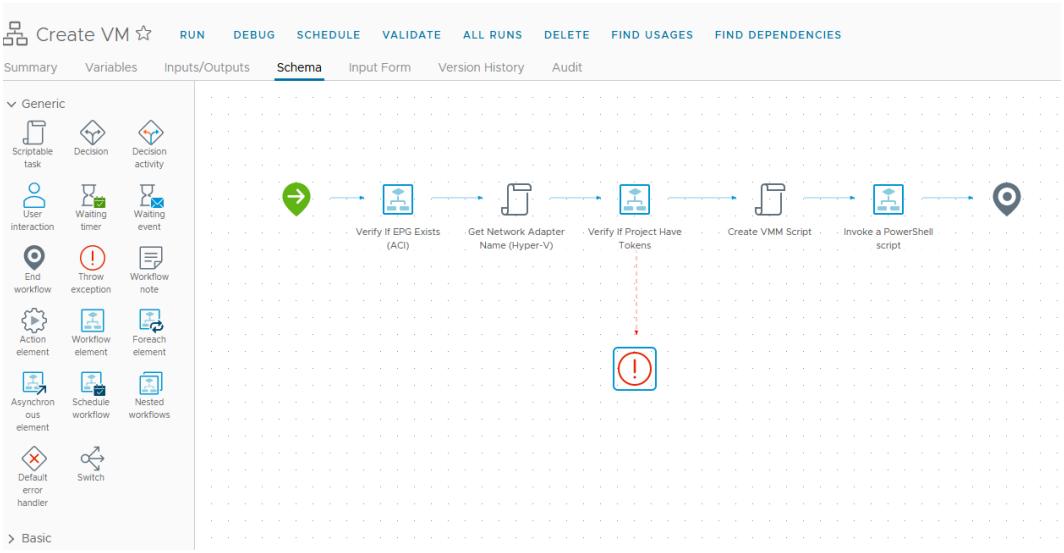


Figura 4.31 - *Workflow* de criação de máquinas em *Hyper-V*

Para verificar a rede, como explicado no caso das máquinas em *vSphere*, é executado o *workflow* que acede ao ACI e verifica se já existe uma rede para as máquinas do âmbito em questão e caso não exista efetua a criação de uma. No entanto, ao contrário das máquinas em *vSphere*, após a criação no ACI do novo domínio de *broadcast*, é possível associar diretamente à máquina no *Hyper-V*. Assim, após a conclusão do *workflow* é retornado o sucesso da ação, sendo de seguida, a partir do *template* utilizado para criação de máquinas virtuais, especificado o nome da placa rede que foi criada. Pelas parametrizações pré-definidas, o nome das placas de rede deriva sempre de um prefixo pré-conhecido, seguido do nome do projeto, ambiente e categoria, todos separados por um hífen, seguido de um sufixo também pré-conhecido, sendo assim possível conhecer o nome da nova placa de rede criada, assim como associá-la à máquina requerida.

Com o nome da placa de rede definida e as variáveis pré-conhecidas, o próximo passo é a definição do *powershell* necessário para a criação da máquina.

Como ilustrado na Figura 4.32, o processo de criação da nova máquina virtual passa por várias etapas. O primeiro passo é a seleção do objeto *template* a partir do nome do *template* selecionado pelo subscritor e de seguida requerer a avaliação dos membros do *cluster* computacional para selecionar o que possui melhor avaliação para hospedar a máquina. Este processo permite que as máquinas físicas do *Hyper-V* fiquem equilibradas em termos de novas criações de máquinas virtuais. Do mesmo modo que é necessário analisar e selecionar o melhor nó computacional, é também necessário efetuar o mesmo processo para o armazenamento, sendo requerido uma listagem decrescente dos volumes de armazenamento, por avaliação do *Hyper-V*, e selecionado o primeiro elemento da lista, ou melhor elemento. Deste modo é então possível criar uma máquina virtual, sendo apenas necessário, após a sua criação, a associação da placa de rede anteriormente definida à nova máquina, sendo efetuado nas últimas linhas da Figura 4.32.

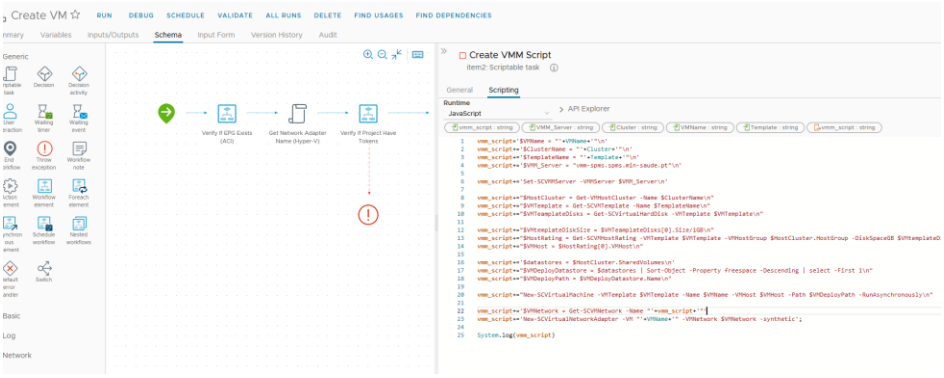


Figura 4.32 - Script *Powershell* de criação de máquina virtual em *Hyper-V*

Deste modo, o *workflow* pode ser concluído, retornando um sinal de sucesso para o *Service Broker*. Este processo espoleta o aparecimento de um novo *deployment* com o nome da máquina que, no caso do *Hyper-V*, é ainda necessário ligar a máquina (ação efetuada pelo subscritor) e selecionar a ação "*Show VM Specs*", como ilustrado na Figura 4.33, permitindo assim descobrir o endereço IP atribuído pelo DHCP e aceder via RDP à nova máquina criada.

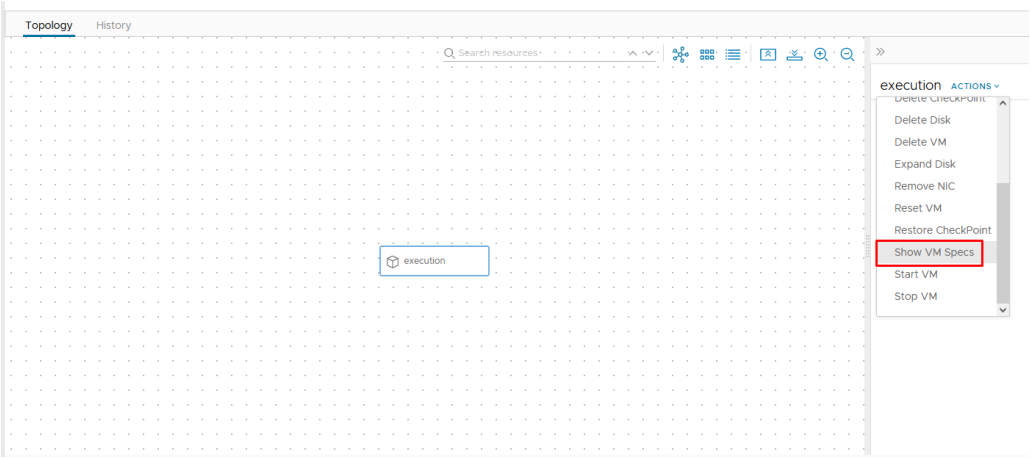


Figura 4.33 - Opção de visualização dos parâmetros da máquina (inclusive endereço IP)

É importante salientar que a priori da criação das máquinas virtuais, é efetuado um controlo de *deployment*, passando pela equipa de administração da plataforma para limitar o crescimento anómalo enquanto a solução se encontra na primeira fase de desenvolvimento.

Em síntese, o processo de criação de máquinas apresenta-se como um dos mais complexos e abrangentes no que toca aos módulos e ferramentas usados, no entanto sendo capaz de responder aos requisitos inicialmente propostos, enquanto disponibiliza um método de inter-relação como o subscritor análogo entre os dois *deployment* concebidos.

4.7 Notificação de Erros e Alertas de Operação

De forma a possibilitar a fácil manutenção dos serviços por parte dos administradores, enquanto se garante a preservação da coerência entre as plataformas de *hypervisors* e de IaaS, a implementação de módulos de notificação acabou por se demonstrar, com o amadurecimento da solução e consequente passagem a produção da mesma, como indispensável para uma boa gestão. Em modo de nota, pela natureza de integração débil entre a plataforma de IaaS e o *Hyper-V*, tendenciosamente as implementações descritas neste subcapítulo recaem maioritariamente sobre a plataforma *Hyper-V*.

No presente subcapítulo é descrita a implementação e o processo de engenharia por detrás da realização dos casos de uso de validar a coerência entre as máquinas no *Hyper-V* e no IaaS, tanto como a validação e notificação de *checkpoints* que ultrapassa os limites estipulados pela SPMS.

No primeiro caso de uso, para validar a coerência da solução, foi necessário recolher os dados de ambas as plataformas, *Hyper-V* e dos *deployments* do *vRealize*. Como ilustrado na Figura 4.34, o primeiro passo é recolher, tanto no *vRealize* como o *Hyper-V*, a informação dos *deployment* e máquinas virtuais em cada plataforma. No caso do *vRealize*, a recolha é feita via API diretamente à listagem fornecida pela plataforma, no entanto no caso do *Hyper-V* foi necessário efetuar um *script powershell* que recolha o nome de todas as máquinas dos *clusters* de computação selecionados.

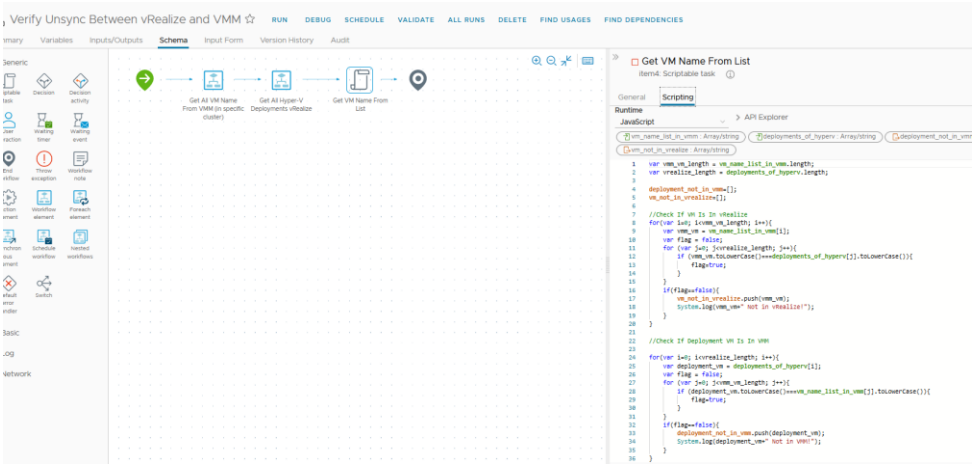


Figura 4.34 - *Workflow* validação dos *deployment*

No caso da recolha de dados do *Hyper-V*, o desenvolvimento do *script* inicialmente efetuava vários pedidos *powershell* ao *hypervisor*, o que rapidamente se demonstrou ineficiente demorando perto de uma hora para recolher toda a informação necessário. Assim, e como ilustrado na Figura 4.35, para otimizar o processo e reduzir o número de pedidos efetuados, os processos de *loop* foram passados para o lado do *hypervisor* com o envio dessas mesmas operações pelo *powershell*. Deste modo, o *script* executa várias chamadas a métodos nativos, reduzindo o tempo de execução para entre vinte a trinta segundos para recolher todos os nomes. De seguida, no campo "*Clean Output*" na Figura 4.35, é efetuada a interpretação dos valores retornados pelo *powershell* e organizado num objeto do tipo lista para possibilitar interpretação por parte da plataforma.

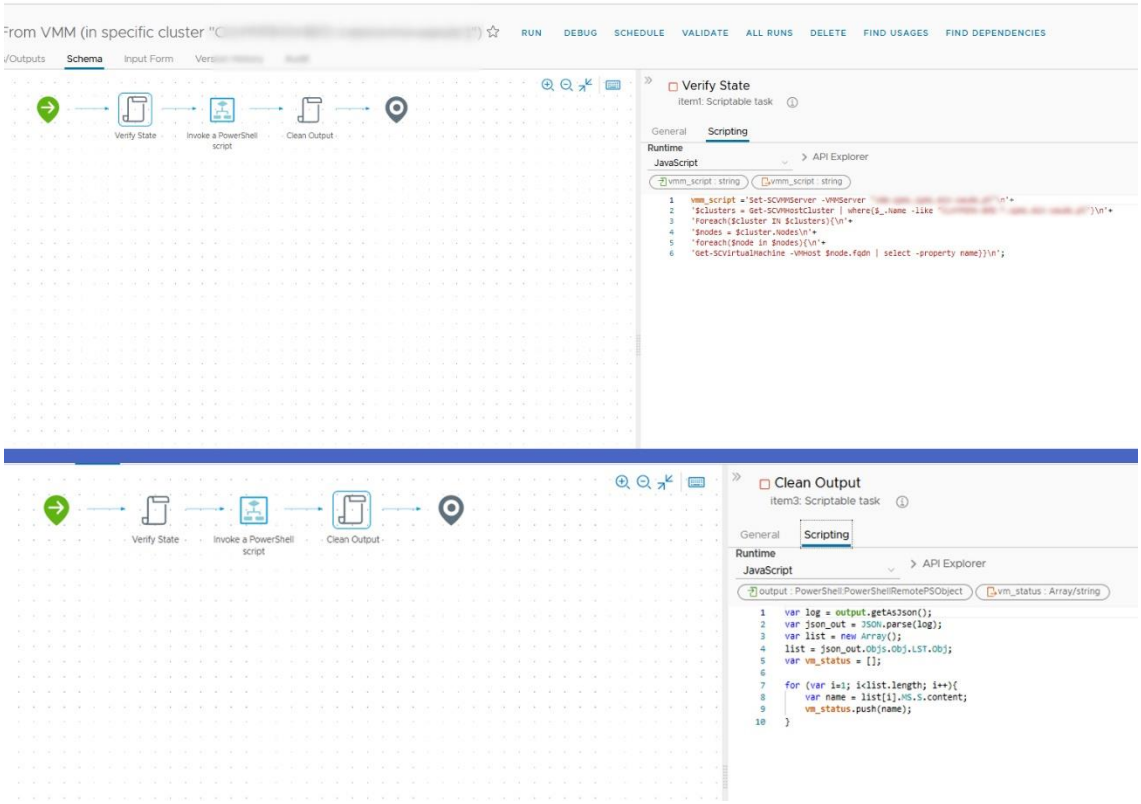


Figura 4.35 - *Script Powershell* de recolha dos nomes das máquinas

Após a recolha dos dados de ambas as plataformas, é necessário comparar ambas as listagens para obter as diferenças entre as mesmas. Tendo em conta que a execução dos *scripts* em *powershell* não são *case sensitive*, essa verificação foi ignorada, extraindo apenas duas listagens originadas da verificação. Na listagem "*present_in_VMM_not_in_vRealize*" é possível encontrar todas os nomes das máquinas que se encontram no *hypervisor* mas que ainda não foram importadas para o *vRealize*, que normalmente representam máquinas que ainda não foram importadas para a plataforma. Na mesma nota de ideias, a listagem "*present_in_vRealize_not_in_VMM*" agrega todas as máquinas que apesar de estarem no *vRealize* não se encontram no *hypervisor*, ocorrem normalmente quando uma máquina é importada com um nome errado ou quando a máquina é eliminada diretamente no *hypervisor*. Assim, é concluído o *workflow* e retornado para o *workflow* "pai" (que o chamou) as duas listagens obtidas.

De seguida, com as listagens preenchidas, o *workflow* "pai" "*Notify Deployment Integrity"* ilustrado na Figura 4.36, valida se alguma das listagens retornadas anteriormente é uma lista não vazia (pois pode não existir divergências, retornando listagens vazias), iniciando de seguida, caso haja pelo menos uma listagem com um ou mais elementos, a formulação do corpo do email. O corpo do email é formulado na tarefa "*Create Email Body*" no qual é adicionado um prefixo genérico previamente

definido, seguido de uma tabela com as duas listagens anteriormente retornadas, acabando com um sufixo também predefinido. O caso de uso acaba com o envio do email com o uso do *workflow* nativo via SMTP. É importante salientar que o registo de servidor de SMTP é idêntico à integração das APIs, e que os servidores utilizados fazem parte da infraestrutura da SPMS.

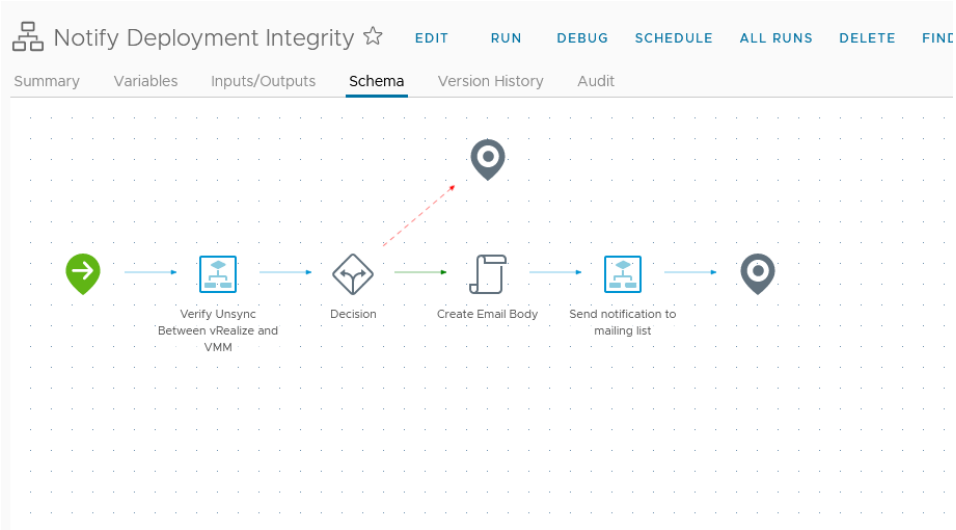


Figura 4.36 - Envio de notificação de inconsistência

Na implementação do caso de uso de notificação dos *checkpoints*, o *workflow* "pai" segue o mesmo princípio do descrito anteriormente, sendo reportado para a equipa de administração da plataforma via SMTP o relatório das máquinas que possuem *checkpoints* (pontos de restauro) que superem o limite estipulado de modo a evitar uso excessivo de armazenamento, assim como problemas associados à alta disponibilidade das máquinas e aos *backups* das mesmas.

É definido a priori um valor de dias de tolerância para o armazenamento de *checkpoints*, sendo no presente caso um valor originado da política de retenção em rigor na SPMS. Em seguida, como ilustrado na Figura 4.37, no *workflow* "*Show Checkpoints in VMM (VM & Date)*" é executado um *powershell* com o número de dias predefinidos que retorna uma listagem de nome da máquina e data de criação dos *checkpoints*.

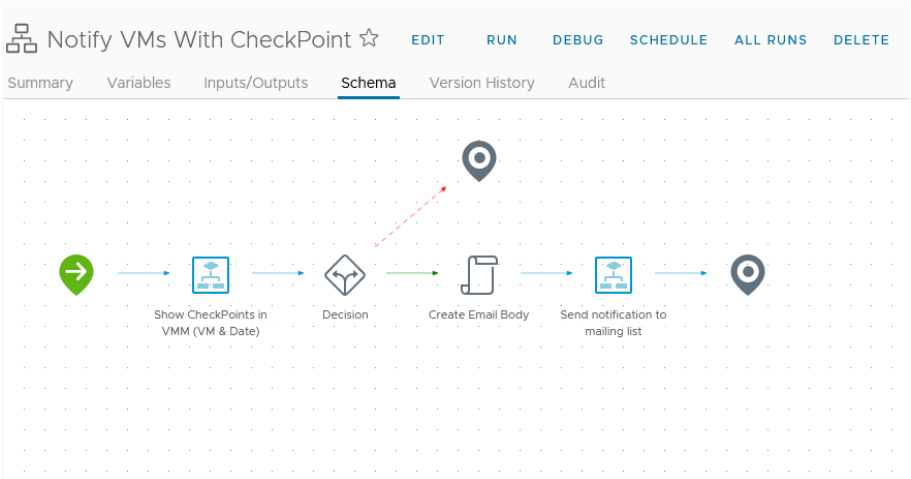


Figura 4.37 - Notificação de *Checkpoints* que não cumprem os requisitos

Em seguida, é validado se existe algum elemento na lista retornada e caso se verifique é criado o corpo do email e executado de seguida o *workflow* nativo de envio de emails para a *mailing list* dos administradores.

Em ambos os casos de uso, e na ausência de um estímulo externo que espolete a execução do *workflow* é necessário criar um cronograma responsável por desempenhar uma ou mais execuções num intervalo de tempo definido. Para tal, foi usada a função nativa do *vRealize* que, como ilustrado Figura 4.38, permite definir o *workflow* a executar, assim como o dia e a hora de execução e a recursividade da mesma.

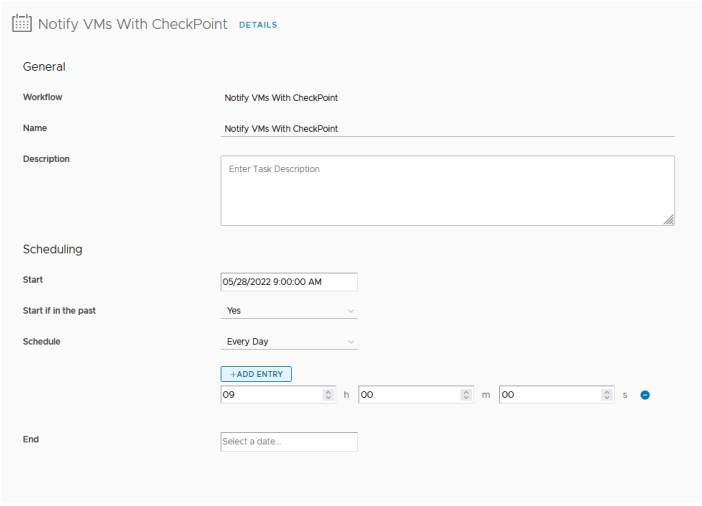


Figura 4.38 - Cronograma de espoleta de *workflow*

Deste modo é possível disponibilizar diariamente um relatório do estado da plataforma, assim como recomendar as ações necessárias para normalizar os itens que não cumpram os requisitos definidos inicialmente pela entidade SPMS. Foram também

disponibilizados os dinamismos necessários para permitir a flexibilidade na adaptação de requisitos futuros como por exemplo a alteração das *mailing list* ou dos parâmetros analisados e reportados.

4.8 Gestão de *Tokens* de Máquinas e Recursos em *Hyper-V*

Como descrito no decorrer do documento é retratada a integração da plataforma com um módulo personalizado desenhado e desenvolvido especialmente tendo em conta as necessidades impostas pela entidade SPMS. Apesar de um dos principais objetivos da plataforma ser automatizar e trocar os processos técnicos manuais dos administradores por mecanismos executados pelos subscritores, é também importante garantir a gestão dos recursos físicos e virtuais para garantir uma boa qualidade de serviço, mas também para dar tempo de projetar e adquirir recursos de computação físicos para a continuação do bom crescimento da plataforma.

Para tal, foi necessário entender junto da entidade SPMS os recursos a serem monitorizados e taxados, assim como qual a melhor divisão de responsabilidade para taxar as diferentes máquinas.

A divisão de responsabilidades foi clara desde a primeira discussão do assunto do novo modelo de controlo de recursos, ficando decidido que cada projeto possui uma cota de recursos que são identificados por *tokens*. Cada *token* é comparável a uma moeda e é subtraído na criação de novas máquinas ou atualização dos recursos de uma já existente.

Em seguida, foi decidido que o melhor método de taxação de recursos seria por componente computacional, sendo assim necessário definir o valor em *tokens* que será subtraído ou adicionado perante as ações dos subscritores. Foi também referida a necessidade de remover ou adicionar mais *tokens* aos projetos consoante a evolução dos mesmos.

Para definir os valores associado a cada recurso foi necessário, em modo de testes, identificar quais os recursos mais caros para a entidade SPMS. Em primeiro, e como representado na Tabela 4.1, a criação de cada máquina necessita de um valor base associado ao objeto lógico da máquina virtual, ficando definido um valor de dez *tokens*. Cada core de CPU virtual dedicado à máquina virtual foi cotado também em dez *tokens*, enquanto por cada *gigabyte* de RAM associado, foi associado um custo de cinco *tokens*. Para cotar o espaço em disco associado a cada máquina, e tendo em conta que é um dos recursos mais dispendiosos, foi associado um valor de dez *tokens* por cada cem *gigabytes* intrínsecos à máquina, evitando também o uso de valores decimais.

Tabela 4.1 - Valor Por Recurso

Recurso	Tarifa por n Recursos
Máquina	10 *tokens* / máquina
CPU	10 *tokens* / *core* CPU
RAM	5 *tokens* / *gigabyte*
Disco	10 *tokens* / 100 *gigabyte*

A plataforma de gestão de *tokens* é constituída por dois elementos, na base de dados foi selecionada uma não relacional, neste caso a *mongodb*, pela simplicidade de configuração inicial e pelo número reduzido de elementos esperados nas tabelas. No *backend*, o uso de *Python 3* apresenta-se como muito versátil para o pequeno projeto e o elevado número de documentação é também um ponto fulcral.

Na implementação do *backend* foi disponibilizado um serviço de API via HTTP com pedidos do tipo GET, POST, DELETE e UPDATE[29] de modo a permitir a manipulação dos dados. Como ilustrado na Figura 4.39, são exemplificados alguns dos métodos disponibilizados pela plataforma de *tokens*, assim como a integração com a base de dados, permitindo de forma simples disponibilizar ó método de controlo de requisitos requerido.

[29] Métodos de pedido HTTP

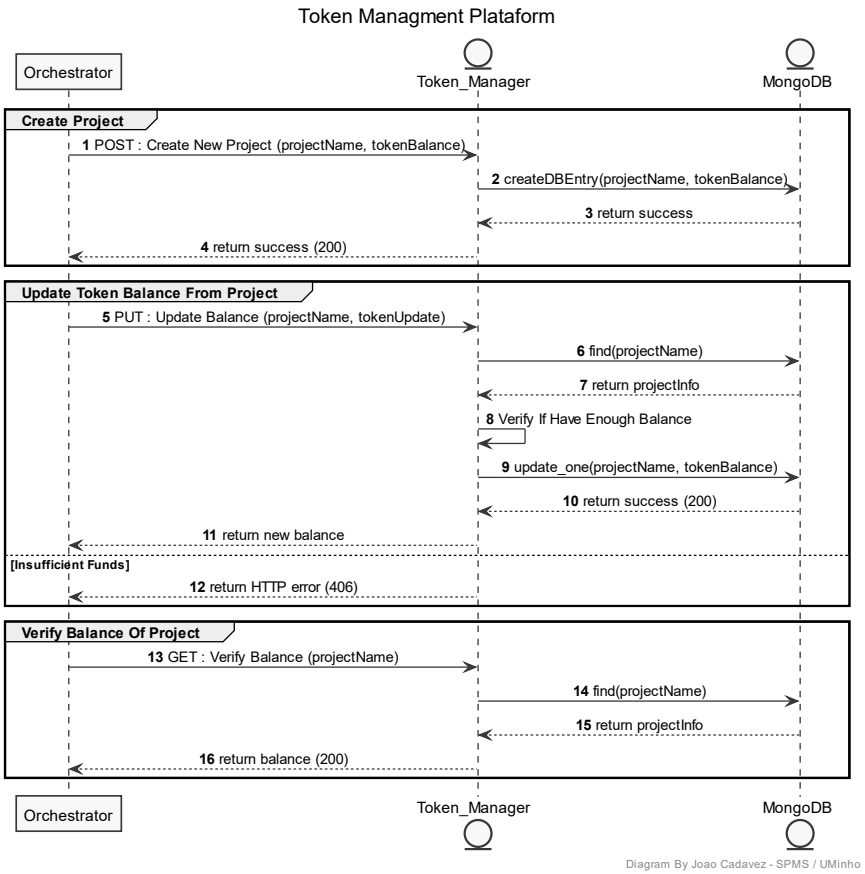


Figura 4.39 - Exemplo de funções da plataforma de gestão de *Tokens*

Como os valores atribuídos aos recursos podem sofrer mudanças devido à variação dos valores dos diferentes componentes computacionais, foi necessário disponibilizar uma forma simples de alterar esses mesmos valores. A possibilidade de alterar variáveis de ambiente ou os valores diretamente na máquina da plataforma de *tokens* pode ser um pouco complexo e levar a falhas por não se demonstrar um procedimento diário, pelo que, e de modo a evitar alterações desnecessárias no código ou na integração, foram criadas *Actions* no *Orchestrator* com os valores definidos para os recursos, como ilustrado na Figura 4.40. Como explicado anteriormente, as *Actions* permitem a criação de pequenos *scripts* ou retorno de valores que são usados por diferentes *workflows*, fazendo com que a alteração do mesmo seja replicada pelas várias instâncias que o usam.

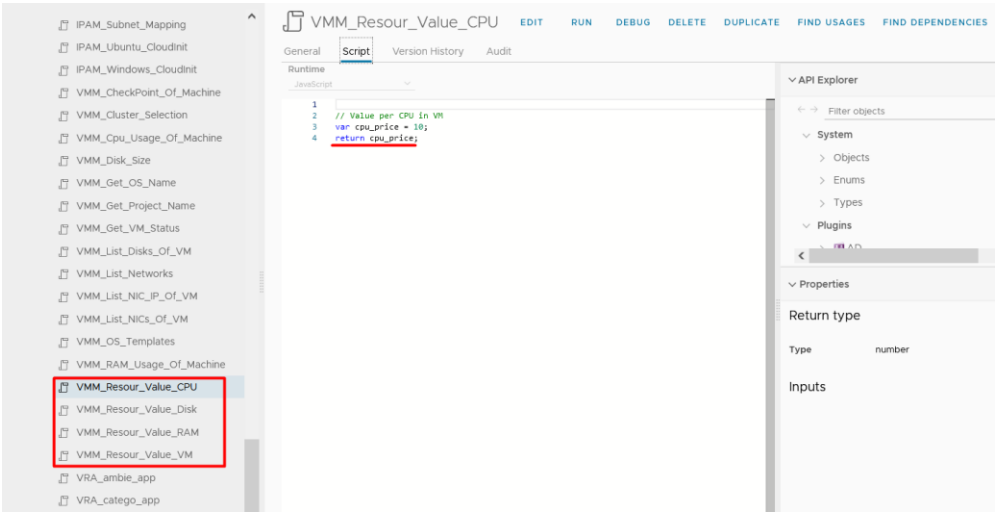


Figura 4.40 - *Actions* com os Valores gerais

De seguida, foi necessário criar um *workflow*, ilustrado na Figura 4.41 que fizesse uso das *Actions* ilustradas na Figura 4.40, sendo responsável por calcular e atualizar os *tokens* a serem removidos ou adicionados tendo em conta os *inputs* fornecidos, que por consequência estão associados aos *workflows* de criação/eliminação/atualização de recursos das máquinas em *Hyper-V*. É também importante salientar que a integração entre o *vRealize* e a plataforma segue os mesmos princípios da integração explorada e exemplificada no IPAM, pela criação de vários objetos associados aos diferentes métodos HTTP disponibilizados.

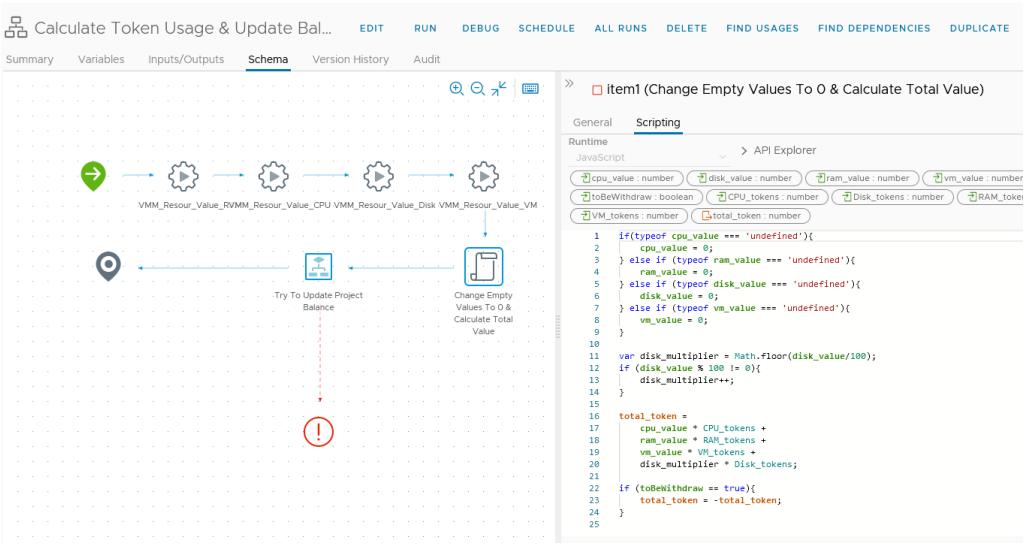


Figura 4.41 - *Workflow* de calculo de novo balanço de *tokens*

Em modo de conclusão, é assim possível garantir o controlo de recursos necessários para o bom funcionamento da plataforma sem sacrificar os automatismos anteriormente

criados, tendo em conta a premissa de uma integração flexível e escalável para a inclusão de novos requisitos futuros.

Capítulo 5 - Testes e Exame de Resultados

Durante o presente capítulo serão discutidos os testes ponto-a-ponto que garantem a integridade da solução, assim como os testes de desempenho da implementação comparativamente com o paradigma anterior. Consequentemente são também demonstrados os passos necessários para o uso da plataforma e dos recursos.

5.1 Importação de máquinas

O caso de uso de importação de máquinas apresentou-se como consequência da já existência de máquinas em *Hyper-V* na infraestrutura antiga, pelo que o exame de desempenho não traduz a utilidade, não sendo linear nem clara o sujeito em comparação.

Em demonstração dos resultados obtidos pela implementação do caso de uso é possível validar na aba partilhada entre os subscritores e os administradores o item de catálogo, ilustrado na Figura 5.1, podendo requerer a criação do item e consequente criação do objeto lógico. A função é, no entanto, reservada a administradores da plataforma pelo que os mesmos são responsáveis pela transferência das máquinas virtuais entre o paradigma antigo e o novo do *vRealize*.

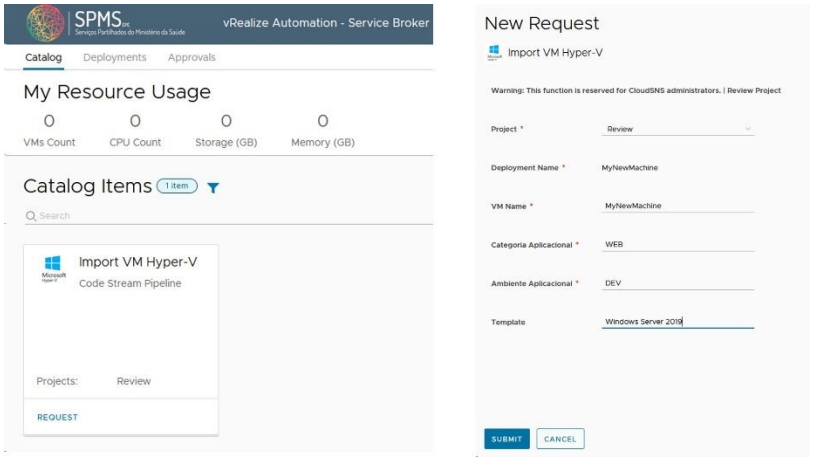


Figura 5.1 - Item de importação de máquinas virtuais provenientes do *Hyper-V*

Após o requisito ser submetido, passa pelos administradores aprovarem a ação anteriormente submetida. Apesar deste passo se demonstrar redundante é indispensável para evitar que os subscritores executem os itens sem autorização, sendo que estes mesmos itens não podem estar somente apresentados aos administradores.

Com o *deployment* aprovado com sucesso, é possível verificar a sua existência na listagem de *deployments* e, no caso de o selecionarmos, é também possível verificar o histórico e as ações pela qual passou até finalizar o processo (Figura 5.2).

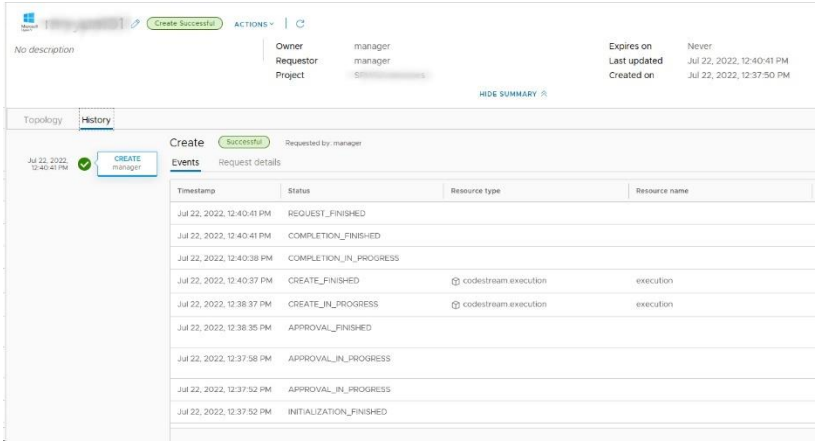


Figura 5.2 - Demonstração de sucesso do *deployment*

Para validar o bom funcionamento da importação, assim como validar se existe algum erro no nome da máquina ou na plataforma que impeça o bom funcionamento da mesma, é utilizada a funcionalidade "*Show VM Specs*" desenvolvida no presente projeto. Caso a máquina se encontre corretamente configurada na plataforma, é possível verificar os atributos da mesma, concluindo assim o teste de ponto-a-ponto. Como ilustrado na Figura 5.3, é possível verificar que já são exequíveis ações que envolvam a informação importada da máquina. A representação do processo ponto-a-ponto pode ser encontrada na secção Capítulo 8 - Apêndice - Imagens na Figura 8.4.

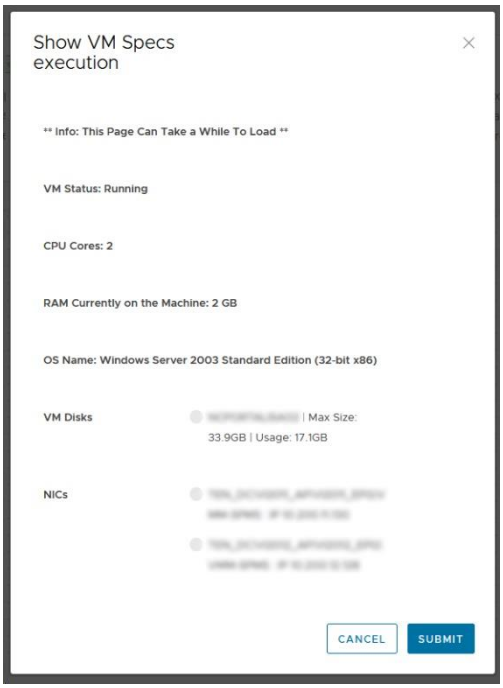


Figura 5.3 · Verificação da correta importação

5.2 Ligar/Desligar Máquinas (*Hyper-V*)

A possibilidade de interação com as máquinas virtuais em *Hyper-V* diretamente a partir da plataforma *vRealize* possibilita aos subscritores terem acesso a funções das mesmas que de outra forma não seria possível.

A nível de desempenho absoluto, o tempo de execução dos processos variam consoante a complexidade das operações e principalmente da carga da plataforma e dos *hypervisor*. Para entender a influência de fatores externos no desempenho dos casos de uso, é necessária a recolha de três amostras (para evitar enviesamentos por recolhas atípicas) em cenários diferentes. O primeiro cenário de recolha é entre as sete e as oito da manhã, representando os horários com menor carga tanto na plataforma *vRealize*, como nos *hypervisors*. Em seguida, a recolha durante horário de pico de ações e de desempenho dos recursos centra-se por volta do meio-dia. Por último, são repetidos os dois passos anteriores durante o fim de semana, representando os cenários mais calmos. Assim, como presente na Tabela 5.1, existe uma média de tempo de execução de 26,12 segundos com um desvio padrão de 0,38, representando uma precisão significativa no que toca à consistência nos tempos de execução.

Tabela 5.1 · Valores de desempenho de ligar/desligar máquina *Hyper-V (em segundos)*

Recolha de Dados (seg)	1ª Amostra	2ª Amostra	3ª Amostra	Média
Semana, 8h00	25,78	25,11	26,79	25,89

Semana, 12h00	26,43	27,63	25,95	26,67
Fim-de-Semana, 8h00	27,06	24,91	25,48	25.81
Fim-de-Semana, 12h00	24,71	25,56	25,93	25,40
Média (Desvio Padrão) [55]	---	---	---	26,12 (0,38)

Como não foi apresentado mais nenhum método de implementação, não é possível comparar com o mesmo fluxo de ações subscritor/plataforma. No entanto, é possível estimar em média o tempo de espera equivalente ao paradigma antigo de modo a ter um termo de comparação que permita concluir a utilidade e melhoria da experiência de utilizador.

Quando um subscritor, no paradigma anterior, desejava ligar ou desligar a máquina, era necessário em primeiro lugar entrar em contacto por email ou telefone e só de seguida é que a ação seria executada. Como é espectável, o tempo de espera pode variar entre cinco minutos em casos urgentes e bastante lineares, desde o início da chamada, até ao administrador proceder à ação no *hypervisor*, até horas em casos de emails perdidos, indisponibilidade dos administradores, ou falta de informação sobre a máquina. Os diferentes passos, e os múltiplos intervenientes na sequência de ações aumentam, inevitavelmente, a probabilidade de falha e os tempos de resposta, que pela burocracia inerente a serviços críticos, penaliza o tempo total da ação.

Em suma, é possível verificar que os tempos de resposta das ações não são imediatos pela integração débil entre o *vRealize* e o *Hyper-V*, mas que tendo em conta os tempos de resposta típico do cenário anterior, assim como a elevada probabilidade de falha, a solução apresentada possibilita a interação direta subscritor/máquina, garantindo ao mesmo tempo um histórico constante das ações executadas e dos atores das mesmas, sendo espectável um melhor desempenho geral na execução das ações assim como uma melhor experiência de uso pelo subscritor.

5.3 Eliminar Máquinas Virtuais

O processo de eliminação de máquina é um processo importante para reforçar a autonomia do subscritor. Assim, o subscritor terá acesso a dois tipos de processos de eliminação. Como ilustrado na Figura 5.4 à esquerda, é possível verificar a opção que o subscritor necessita de selecionar, assim como os devidos avisos sobre a irreversibilidade da ação tomada. À direita da Figura 5.4, validamos o aviso nativo fornecido pelo *vRealize* momentos antes de eliminar a máquina.

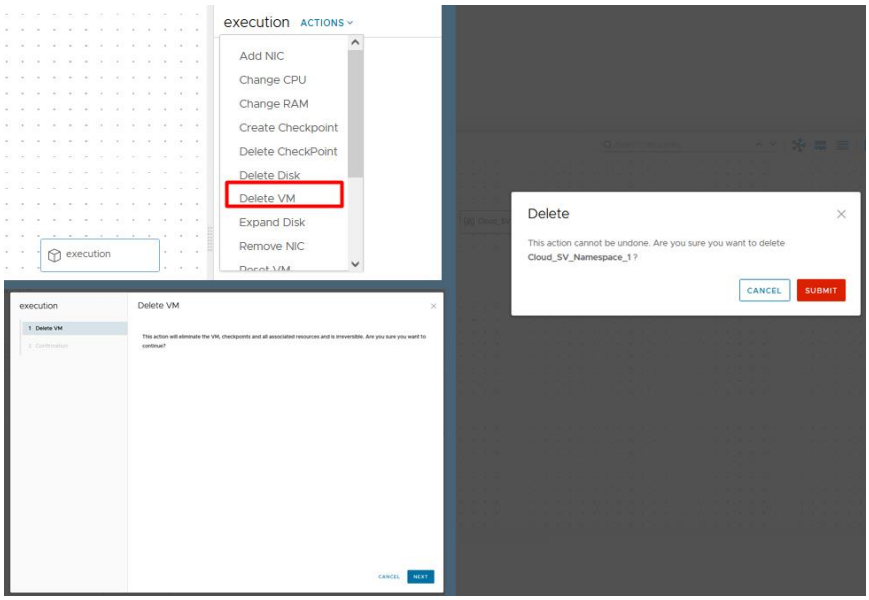


Figura 5.4 - Ilustração do processo de eliminação de máquina (esquerda *Hyper-V*, direita *VMware*)

Após o início do processo de eliminação de máquinas, para sistemas em *Hyper-V*, é espoletado um conjunto de ações, explicado em detalhe no subcapítulo 4.4 - Eliminação de Máquina Virtual, que possibilita o sucesso da operação. Como ilustrado na Figura 5.5 no canto superior esquerdo, os *workflows* são executados sequencialmente com sucesso, sendo possível confirmar o sucesso da operação a partir das ações reportadas pelas plataformas. No IPAM, é efetuado o acesso via API e requisitada a eliminação do IP reservado para a máquina, no *Hyper-V* a máquina é eliminada com sucesso, libertando os recursos utilizados e eliminando as dependências associadas à mesma (como por exemplo os *snapshots*). É de salientar que é efetuada a devolução dos *tokens* referentes à máquina, sendo o valor determinado ao momento do requisito da eliminação do recurso. Por último, é efetuado o pedido de eliminação do *deployment* do *vRealize* com recurso às capacidades de comunicação via API, sendo eliminado por completo o histórico do recurso.

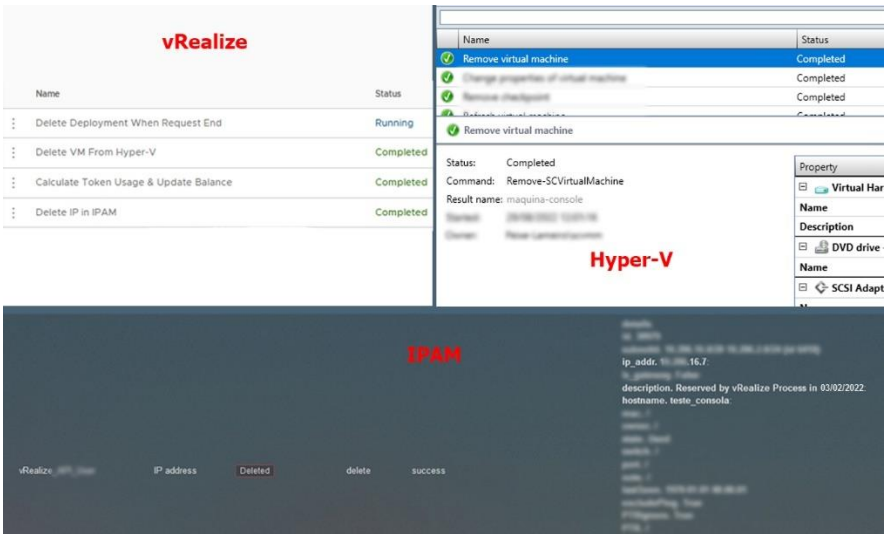


Figura 5.5 - Execução dos diferentes processos de eliminação de máquina *Hyper-V*

Pela integração nativa com o *VMware* o processo de eliminação torna-se mais simples, sendo apenas necessário a eliminação do registo do *IP* da plataforma *IPAM*. Quando o subscritor inicializa o processo, a *subscription* é espoletada, como ilustrado na Figura 5.6, dando origem à execução do *workflow* de eliminação do registo. Assim, como descrito anteriormente, o IPAM é acedido via API e é efetuada a remoção da informação solicitada, garantindo o sucesso da operação.

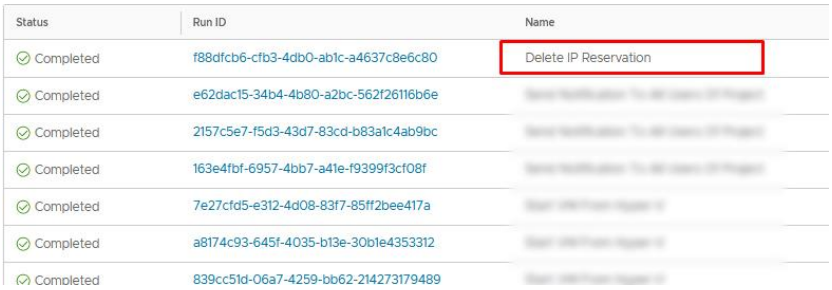


Figura 5.6 - Execução da *subscription* de eliminação de IP no IPAM

5.4 Alteração de parâmetros das máquinas (em *Hyper-V*)

A alteração de parâmetros nas máquinas virtuais em *Hyper-V* apresenta-se, no paradigma anterior, como uma tarefa especialmente morosa. A troca de parâmetros em máquina virtuais *Hyper-V* traduzem-se, em quase cem porcento dos casos, em atualizações aos recursos computacionais ou de armazenamento da máquina virtual, sendo necessário ter controlo dos mesmos devido a aquisição de componentes físico a priori da sua alocação total.

Com a implementação apresentada, o subscritor só necessita de ter acesso à plataforma *vRealize* e selecionar a máquina que deseja, como ilustrado na Figura 5.7, sendo possível, após a seleção do objeto "*execution*", selecionar o menu e escolher a opção de alteração de recursos. No exemplo da alteração do número de CPUs, representado na Figura 5.7, após aguardar alguns segundos enquanto os dados da máquina são recolhidos, é possível verificar o número atual de CPUs e é possível selecionar um novo valor, neste caso para quatro *cores*.

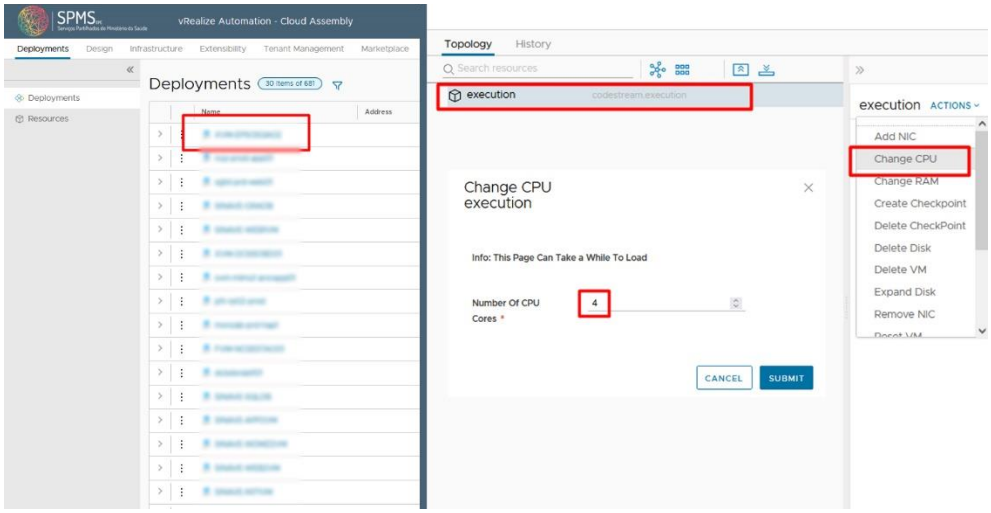


Figura 5.7 - Ilustração da seleção do objeto lógico máquina e ação associada

Após a confirmação da ação, o processo é inicializado e é possível verificar em tempo real o estado e a etapa do mesmo no *Orchestrator*, como ilustrado na Figura 5.8. É importante salientar que, neste momento, no decorrer do *workflow*, são validados alguns parâmetros entre eles o saldo do projeto para validar se é possível efetuar a operação.

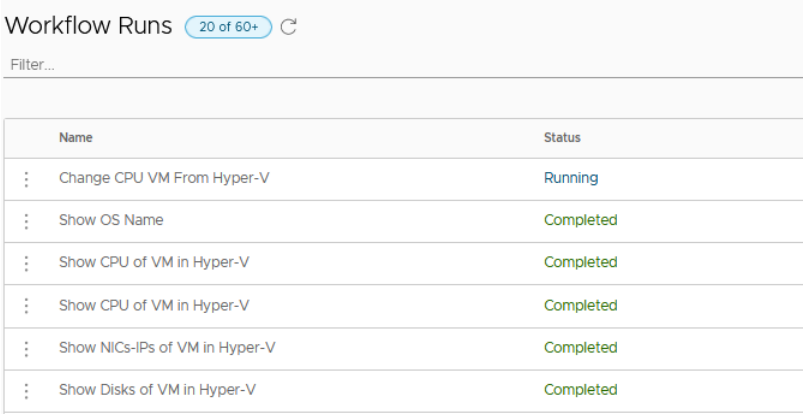


Figura 5.8 - *Workflow* de alteração de CPU em curso

Momentos depois, já no *hypervisor*, é possível constatar que alterações estão a ser efetuadas na máquina em questão, mais especificamente nos recursos da mesma. Como ilustrado na Figura 5.9, é possível observar que a alteração já foi efetuada com sucesso, e que o êxito do processo está a ser reportado para o *Orchestrator* e consequentemente para o subscritor.

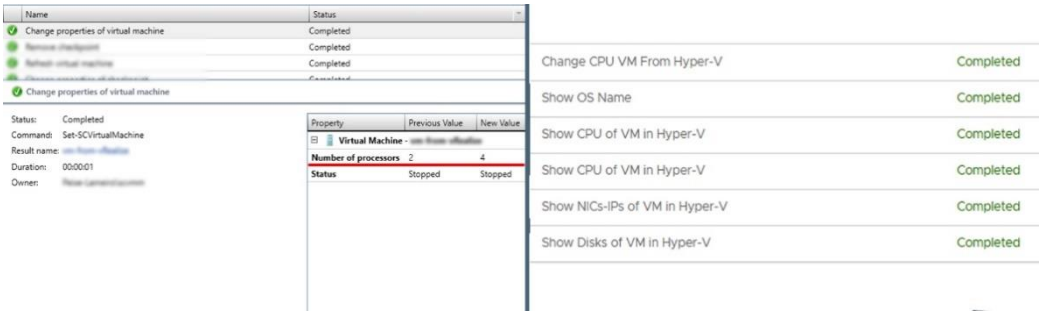


Figura 5.9 - Alteração do número de CPUs no *hypervisor* e conclusão em sucesso do *workflow*

Assim, é possível constatar o bom funcionamento de todo o processo ponto-a-ponto, desde o requisito de atualização de recursos, até ao aviso de sucesso de operação enviado para o subscritor.

No paradigma passado, para alterar os recursos de uma máquina, um subscritor necessitava de informar os administradores do *hypervisor* da necessidade de atualizar um recurso de uma máquina virtual. A maior parte das vezes a atualização de recursos é para aumentar os mesmos, o que faz com que a decisão de prosseguir com a atualização tenha de passar obrigatoriamente pelo responsável da infraestrutura.

Assim, de modo idêntico ao subcapítulo precedente e como ilustrado na Figura 5.10, na solução anterior o pedido de atualização de recursos é inicializado pelo subscritor via telemóvel ou email consoante a urgência, sendo atribuída a um administrador. Em seguida, o administrador valida a operação com o responsável, começando o administrador a operação diretamente no *hypervisor*, avisando do sucesso da operação mais tarde ao subscritor.

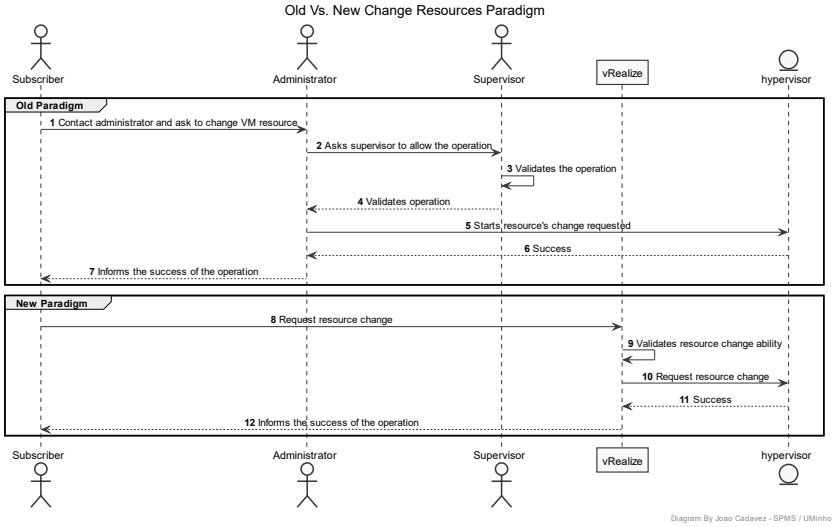


Figura 5.10 - Comparação entre o antigo e o novo paradigma de mudança de recursos

O número de etapas sujeitas para a realização do caso de uso e o número de sujeitos que possuem um papel importante no cenário é maior do que o desejável, comprometendo por vezes as SLAs e aumentando significativamente a probabilidade de erro humano.

A introdução do novo paradigma introduz uma maior versatilidade às ações desejadas pelos subscritores, possibilitando a diminuição significativa dos tempos de resposta. Para melhor entender os benefícios proporcionados pela implementação do novo mecanismo da gestão, foram recolhidos os tempos de resposta da plataforma *vRealize* desde o início da ação até à conclusão da mesma, permitindo assim a comparação com os valores anteriormente praticados.

Tabela 5.2 - Valores de Desempenho Alteração de Parâmetros De Máquina *Hyper-V (em segundos)*

Recolha de Dados (seg)	**1ª Amostra**	**2ª Amostra**	**3ª Amostra**	**Média**
Semana, 8h00	31,10	30,78	30,73	30,87
Semana, 12h00	30,96	31,32	31,90	31,39
Fim-de-Semana, 8h00	30,54	30,71	31,03	30,76
Fim-de-Semana, 12h00	31,27	31,49	30,89	31,21
Média (Desvio Padrão) [55]	---	---	---	31,00 (0,27)

Como espectável, e representado na Tabela 5.2, os tempos de execução rondam os 31 segundos, que em comparação com o paradigma anterior em que pode atingir

entre algumas dezenas de minutos a horas, é possível constatar uma melhoria considerável no que toca à experiência de uso e na redução de erros humanos.

5.5 Criação de máquinas

A implementação do caso de uso de criação de máquinas apresentou-se como um dos mais complexos, tanto pela interação com diferentes *hypervisors*, como pela relação com plataformas terceiras que permitem o bom funcionamento dos recursos na melhor simbiose possível.

Para inicializar os testes ponto-a-ponto é necessário dividir em duas fases a avaliação. Em primeiro lugar, e como referido no capítulo de implementação, é necessário avaliar as máquinas que são criadas em *VMware*, ou seja, as que usam as funcionalidades por defeito na plataforma *vRealize*. Em seguida, são avaliadas as máquinas criadas em *Hyper-V*, ou seja, todas as que possuem o sistema operativo *Windows*.

Durante os testes ponto-a-ponto serão demonstrados e avaliados a interação da plataforma *vRealize* com plataformas de terceiros de modo a demostrar a aplicabilidade prática descrita no capítulo de implementação.

Inicializando os testes com as máquinas criadas em *VMware*, a primeira etapa passa pelo requisito de uma nova máquina virtual por parte do subscritor. Como ilustrado na Figura 5.11, o subscritor necessita de selecionar a opção "CentOs-8" que inicializa o questionário dos atributos da máquina.

Como descrito na implementação, dependendo dos itens selecionados pelo subscritor no momento de criação da máquina, a mesma terá características diferentes que permitirá isolar os diferentes contextos e projetos.

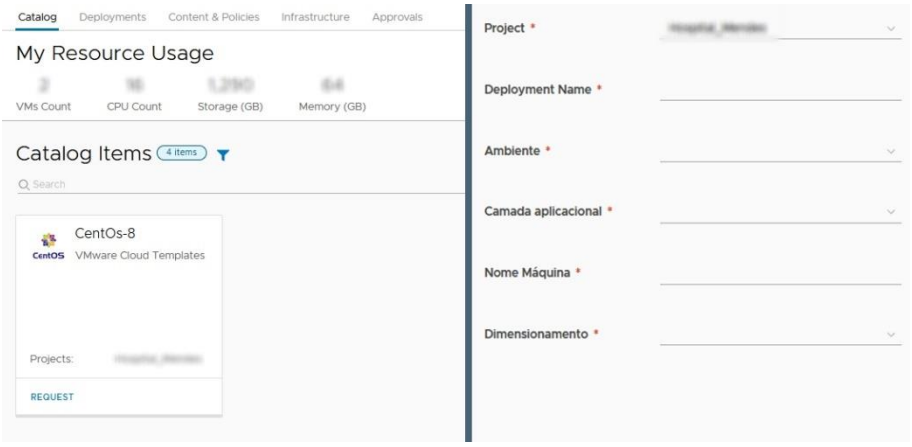


Figura 5.11 - Processo de criação de uma máquina virtual por parte do subscritor

Ao submeter a ação, o subscritor espoleta um conjunto de ações intrínsecas à plataforma *vRealize* e à integração com o *VMware*, que em boa parte são até ubíquas aos administradores e à plataforma. No entanto, antes da criação diretamente no *hypervisor* da máquina virtual, a interação é capturada e é executado um *workflow* previamente definido. Este, como anteriormente descrito, dependendo do sistema operativo selecionado, injeta uma configuração personalizada de *Cloud-Init* com o IP da máquina e o *hostname*, como ilustrado no parâmetro do *Cloud-Init* gerado na Figura 5.12.

Figura 5.12 - Output do *workflow* de alteração do *Cloud-Init*

Ainda durante a parametrização do *Cloud-Init* são efetuadas chamadas à plataforma externa IPAM, na qual é registada a nova máquina, como ilustrado na Figura 5.13, dependendo do seu contexto, e retornado o endereço IP atribuído à mesma e que é remetido nas parametrizações do *Cloud-Init.* A representação do fluxo de interações pode ser encontrado na Capítulo 8 - Apêndice - Imagens nas Figura 8.1, Figura 8.2 e Figura 8.3.

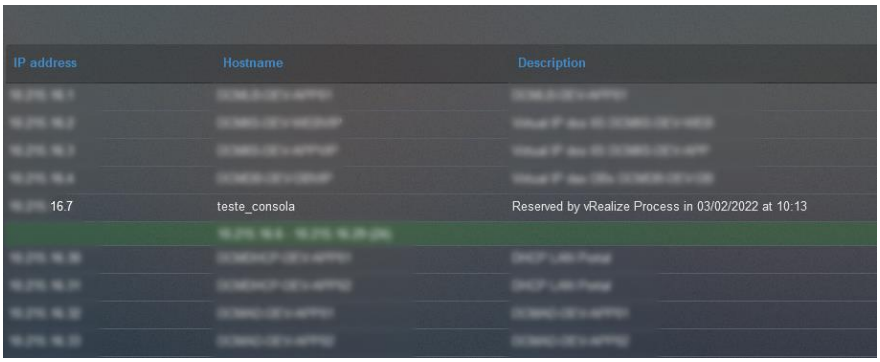


Figura 5.13 - Ilustração da reserva automática do IP no IPAM

Como também descrito durante a implementação do caso de uso, existe também a interação entre o *vRealize* e a plataforma de SDN ACI da *Cisco*, de modo a fornecer um domínio de *broadcast* isolado para as máquinas virtuais de um projeto. Assim, durante a criação da máquina é também efetuada a verificação se o domínio de *broadcast* já existe, e caso não exista é criado e adicionado à máquina via *tags*. Deste modo, e como ilustrado na Figura 5.14, é possível validar a criação do domínio que estará associado à máquina virtual na sua criação.

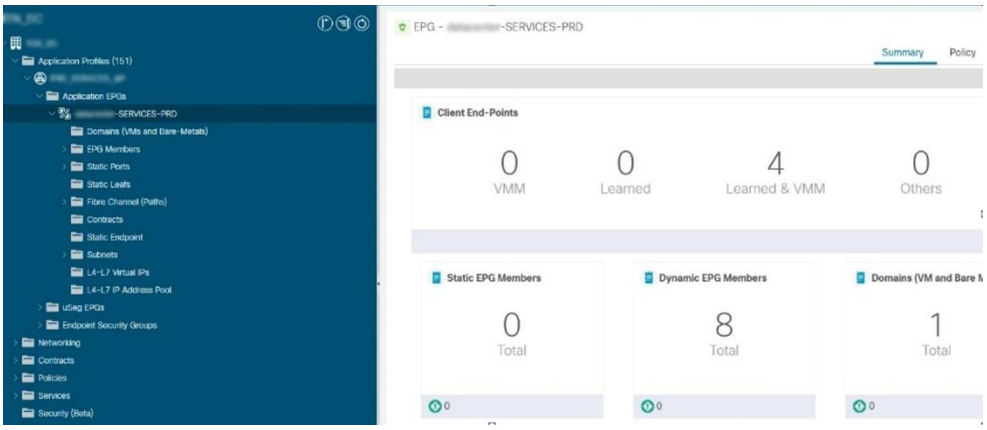


Figura 5.14 - Domínio de *broadcast* associado à máquina virtual

Durante o tempo de execução dos *workflows* o processo de criação da máquina virtual fica em pausa a aguardar o sucesso da operação do *workflow*. Após o receber continua o processo inerente à plataforma de modo usual, proporcionando a injeção dos parâmetros *Cloud-Init* e utilizando o novo domínio de *broadcast* que, como explicado, se traduz numa placa de rede no *hypervisor*. Deste modo, e como ilustrado na Figura 5.15, é possível constatar o sucesso da operação de criação da máquina virtual requerida.

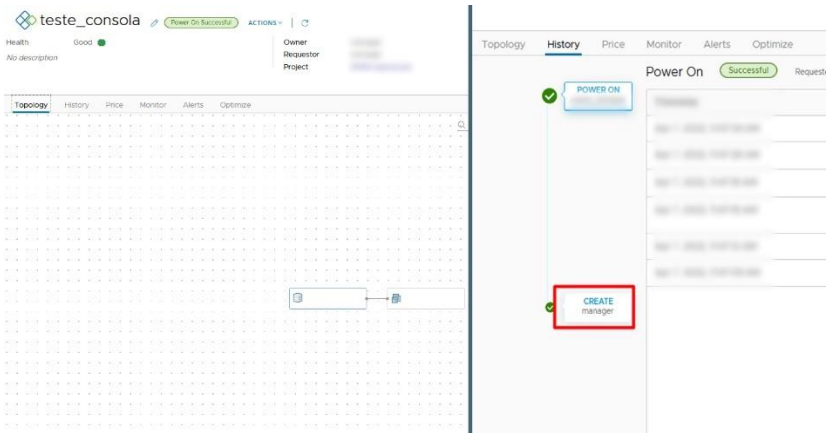


Figura 5.15 - Sucesso da criação da máquina virtual

É ainda importante demonstrar, como ilustrado na Figura 5.16 que foram aplicados os valores parametrizados nos *workflows* durante a execução da ação.

```
root@teste-consola:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0:  <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:ca:c9:c6 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet [redacted] 16.7/20 brd [redacted].255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::172:55af:96dd:7822/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Figura 5.16 - Injeção dos parâmetros selecionados na máquina virtual

Assim, é possível concluir o processo de criação de máquinas virtuais em ambientes *VMware* e o funcionamento do processo ponto-a-ponto da criação da máquina. É também importante salientar que apesar do processo complexo, para o subscritor o processo de criação de máquinas, após a introdução dos dados iniciais, é apenas uma barra de tarefas a carregar e que o mesmo só necessita de aguardar para o processo estar com o estado completo, sendo completamente ubíquo os processos e interações ocorridas em *background*.

Na mesma nota de testes, iniciamos a avaliação de integridade à criação de máquinas *Windows* em *Hyper-V*. O processo, de modo idêntico ao descrito para as máquinas *Linux*, começa pela criação de um estímulo externo, neste caso a espoleta do item do catálogo para criar máquinas *Windows* por parte do subscritor, como ilustrado na Figura 5.17.

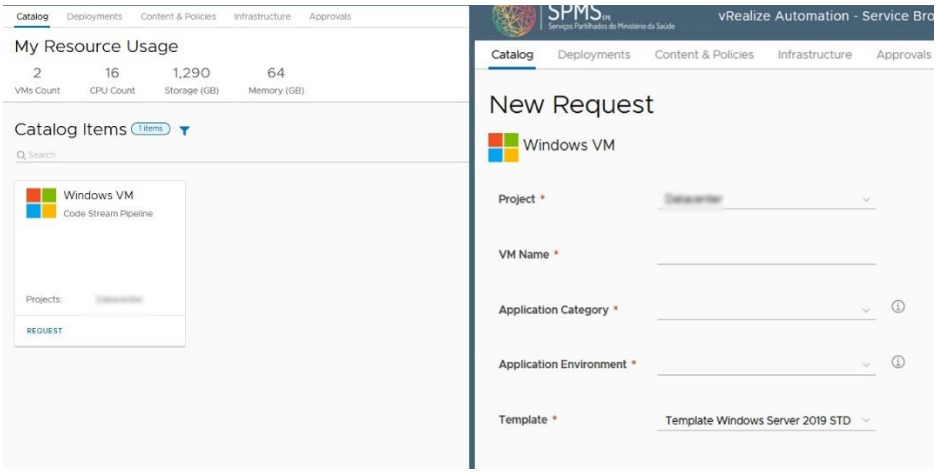


Figura 5.17 - Menu de criação de máquina virtual em *Hyper-V*

Em seguida, após o preenchimento dos campos e da submissão do pedido, o subscritor fica a aguardar um sinal de sucesso da plataforma enquanto em *background* os processos são executados. Pela natureza de *scripting*, o processo passa diretamente para o *Orchestrator* onde são inicializadas as validações e a criação da máquina.

Primeiramente é necessário avaliar diretamente na plataforma ACI se já existe rede para a nova máquina. Para tal, como já ilustrado anteriormente na Figura 4.32, a verificação da existência do domínio de *broadcast* garante, para além da verificação, também a sua criação caso não exista, sendo possível validar o sucesso da sua criação na Figura 5.14. Após a criação/verificação do domínio de *broadcast* é possível selecionar o nome do mesmo que será usado para atribuir à máquina virtual criada em seguida.

Em consequência, o *script powershell* é preenchido com a informação recolhida anteriormente e inicializado o processo de verificação de *tokens*. Como ilustrado na Figura 5.18, é possível verificar que o projeto possui os *tokens* necessários para a criação da nova máquina, assim como o balanço final do projeto após a validação da ação.

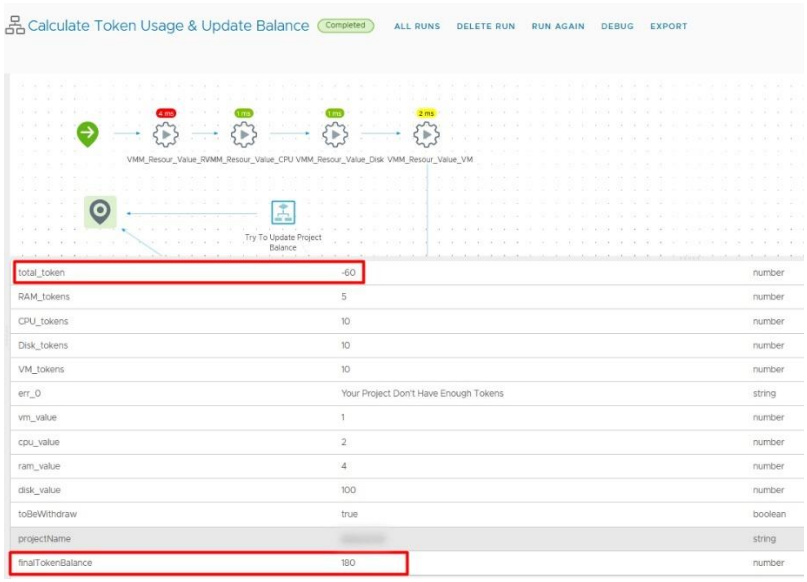


Figura 5.18 - Cálculo e desconto dos *tokens* referentes à nova máquina virtual

O próximo passo é a execução do *script powershell* preenchido anteriormente. A execução é feita diretamente na máquina que gere os *cluster Hyper-V*, o VMM, interagindo com os nós *Hyper-V* e resultando, como ilustrado na Figura 5.19, na criação da máquina com os parâmetros selecionados. Após a conclusão com sucesso do *workflow* o subscritor recebe uma mensagem de sucesso idêntica à Figura 5.15.

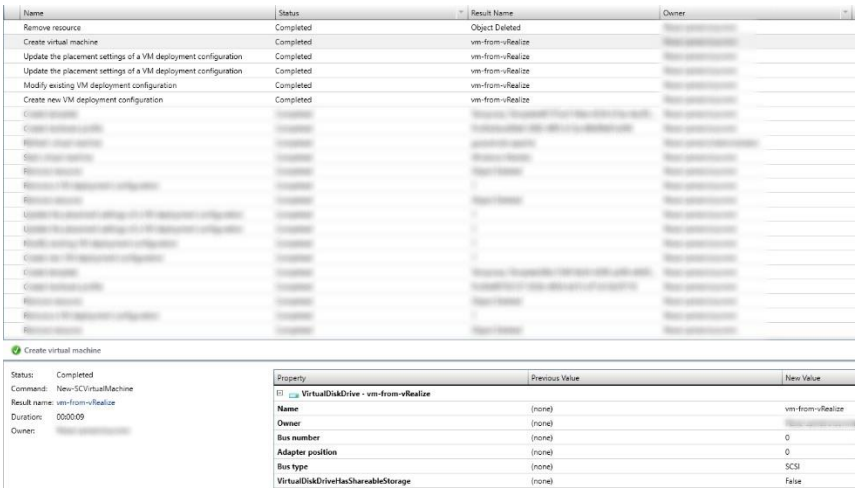


Figura 5.19 - *Job* de criação da máquina virtual no *Hyper-V*

A partir do sucesso da operação, o subscritor já pode aceder à máquina virtual em questão via RDP e utilizar as ações associadas à máquina para manipular parâmetros da mesma ou obter mais informações sobre o estado ou os recursos

atribuídos, concluindo assim o teste ponto-a-ponto da criação de máquinas *Hyper-V* com a demonstração das interações simbióticas entre as plataformas.

Por último, resta avaliar o desempenho das ações de criação de máquinas virtuais, tanto em *VMware* como em *Hyper-V*, comparando também com o paradigma anterior tanto em termos de tempo total de ações, como em probabilidade de erro humano nas configurações.

Como representado na Tabela 5.3, é possível verificar que o tempo de conclusão da criação da máquina virtual, e de todos os processos inerentes à ação, é muito próximo entre o *Hyper-V* e o *VMware*, o que em primeiro plano não seria espectável pela integração nativa da plataforma de virtualização da *VMware* com o *vRealize*. No entanto, e em modo de especulação, a diferença provavelmente deve-se aos processos de teste, verificações e validações que ocorrem intrinsecamente entre plataforma e ao *hypervisor* da *VMware*. No entanto, e destaca-se o desvio padrão mais elevado na criação de máquinas em *Hyper-V*, o que se pode traduzir numa integração menos estável e precisa, pelo menos no que toca aos tempos de resposta.

Tabela 5.3 - Valores de desempenho criação de máquina virtual (em segundos)

Recolha de Dados (seg)		**1ª Amostra**	**2ª Amostra**	**3ª Amostra**	**Média**
Hyper-V	**Semana, 8h00**	429	457	472	452,66
	Semana, 12h00	410	424	417	417,00
	Fim-de-Semana, 8h00	441	426	443	436,66
	Fim-de-Semana, 12h00	413	437	419	423,00
	Média (Desvio Padrão) [55]	**---**	**---**	**---**	**435,44 (14,59)**
VMware	**Semana, 8h00**	544	553	531	542,66
	Semana, 12h00	520	535	546	533,66
	Fim-de-Semana, 8h00	535	529	554	539,33
	Fim-de-Semana, 12h00	521	516	549	528,66
	Média (Desvio Padrão) [55]	**---**	**---**	**---**	**538,55 (3,72)**

Por fim, em modo de conclusão, é necessário relatar a realidade do paradigma anterior de modo a validar a viabilidade da solução proposta. De modo idêntico ao descrito no subcapítulo anterior e com uma ilustração simplificada na Figura 5.20, é possível validar que para a criação de uma máquina virtual no paradigma antigo passa pelo contacto via email ou telemóvel do subscritor para os administradores dos *hypervisors*, sendo validado em seguida a ação pelo responsável. Em seguida, todo o processo de criação de novos domínios de *broadcast*, documentação da informação da nova máquina no IPAM e por fim criação da máquina têm de ser executados pelo

administrador, envolvendo, inevitavelmente, uma grande probabilidade de erro humano na configuração ou na correta documentação das informações da máquina. No final o administrador informa o subscritor do sucesso da operação e comunica o endereço IP para aceder à máquina via RDP.

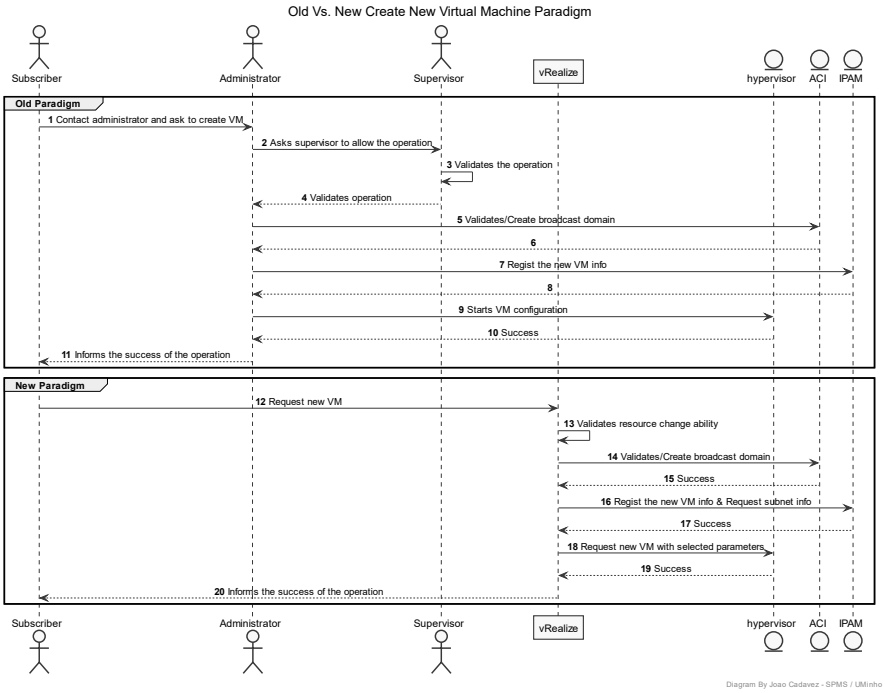


Figura 5.20 - Comparação entre o antigo e o novo paradigma de criação de máquinas

Tal como descrito previamente, pela complexidade do processo e pela dependência de pelo menos três elementos humanos, é difícil garantir SLAs estáveis e com tempos de resposta relativamente baixos (em comparação com as exigências criadas), sendo espectável em situações de urgência tempos de resposta entre os trinta minutos a uma hora, até várias horas de espera por parte do subscritor.

Deste modo, e tendo como base de argumentação a informação exposta durante os testes do presente caso de uso, é possível constatar com a implementação proposta uma melhoria no serviço fornecido ao subscritor, não só pela aprimoração do processo de criação de máquina, mas também pela garantia de documentação atualizada para futura consulta. Ademais, a diminuição do tempo de resposta fornece ao subscritor uma experiência de uso melhor, enquanto não necessita de depender da disponibilidade de elementos humanos, nem da sua viabilidade na execução das ações necessárias.

5.6 Notificação de Erros e Alertas de Operação

De modo a testar o desempenho e usabilidade da implementação proposta para o caso de uso de notificação de erros e alertas de operações, foi necessário analisar cada um dos passos executados pelas ações, de modo a validar o bom funcionamento ponto-a-ponto.

Inicialmente, e tendo em conta o caso de uso proposto no subcapítulo 4.7 - Notificação de Erros e Alertas de Operação, validamos a implementação da coerência entre as máquinas no *Hyper-V* e no *vRealize*. Iniciando pelo método de espoleta, foi necessário validar se a implementação de *scheduled* apresentou o resultado esperado, e como ilustrado na Figura 5.21, é possível verificar do histórico de execuções de *workflows* que está a ser executado todos os dias às nove da manhã.

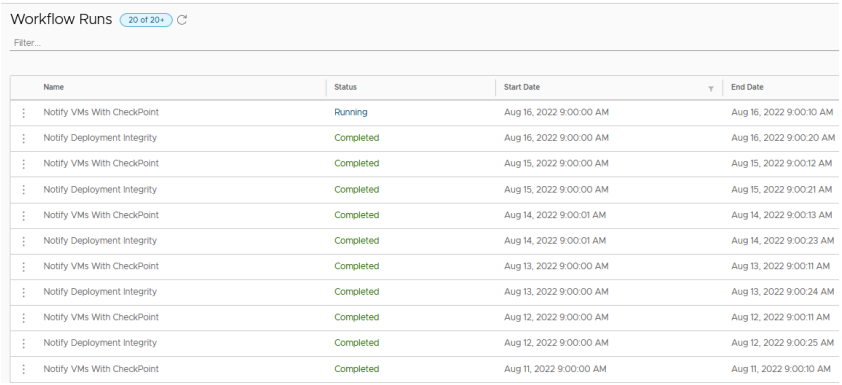


Figura 5.21 - Histórico de execuções de alertas e notificações

Em seguida, é necessário validar as consequentes execuções efetuadas internamente ao *workflow*. Começando pela recolha de dados do *hypervisor*, é possível validar no histórico de variáveis que foi capaz de encontrar as máquinas virtuais presentes no mesmo, ilustrado na Figura 5.22, e na execução seguinte do *workflow* "pai" é efetuada a recolha e tratamento das máquinas que estão no *vRealize*, também presente na Figura 5.22.

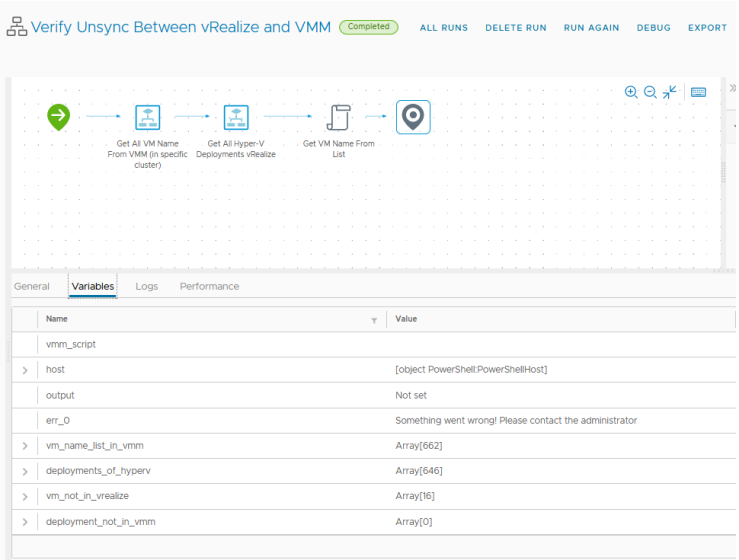


Figura 5.22 - Validação das máquinas virtuais *Hyper-V* na plataforma vs. *hypervisor*

Após a recolha e tratamento da informação, processo descrito no capítulo da implementação, são geradas duas listagens que necessitam de ser tratadas para um formato de mais fácil leitura e enviadas por email. O processo cria um texto no formato HTML que é automaticamente interpretado pelo motor de leitura de email.

Por fim, as informações de destinatário, assunto e corpo de email são submetidos no *workflow* nativo do *vRealize*, e utilizados os servidores *SMTP* da entidade SPMS, resultando, caso haja conteúdo em pelo menos uma das duas listagens originadas, num email como o ilustrado na Figura 5.23.

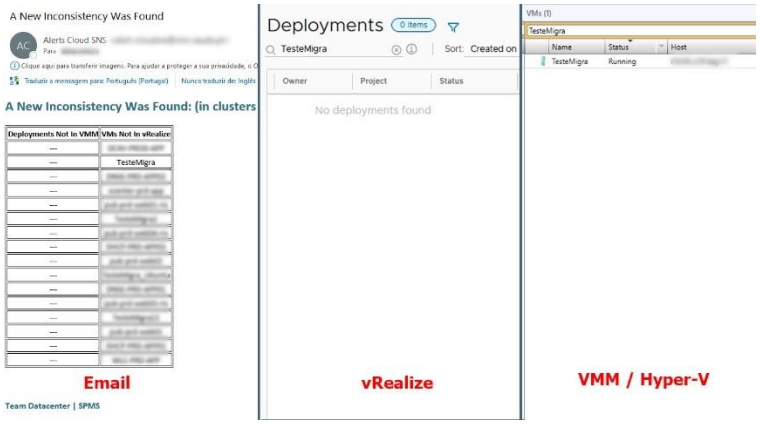


Figura 5.23 - Verificação do correto funcionamento da função de alerta

Para validar o correto funcionamento da ferramenta, e como presente Figura 5.23 é possível validar que a máquina "TesteMigra" que foi reportada como não estando

no *vRealize* mas estando no *hypervisor*, comprovando o funcionamento da implementação ponto-a-ponto.

De modo a verificar o bom funcionamento da função de validação e notificação de *checkpoints* que ultrapassa os limites estipulados, de um modo idêntico ao descrito anteriormente, começou-se por validar se o *scheduled* foi configurado corretamente, e como ilustrado Figura 5.24, é possível validar o funcionamento do mesmo todos os dias às nove horas da manhã.

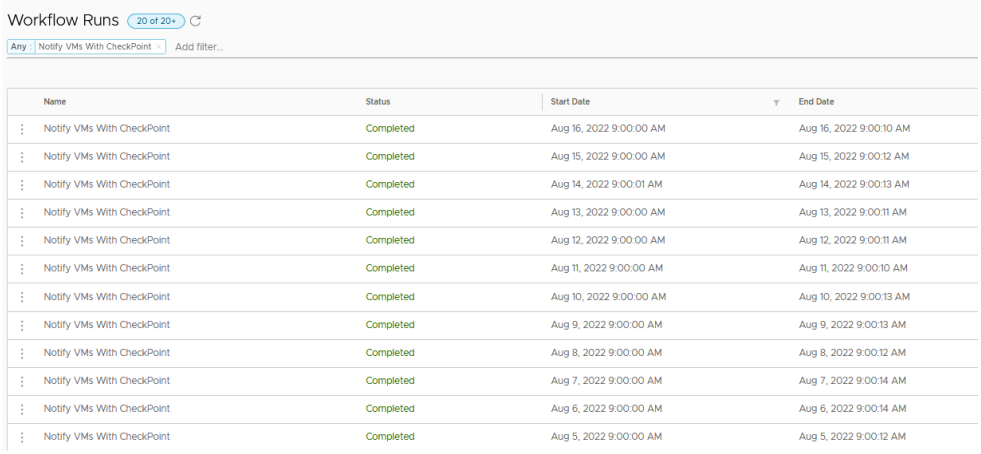


Figura 5.24 - Histórico das execuções do *workflow* de validação de *checkpoints*

Em seguida, e tendo em conta que o *script powershell* é preenchido com os dias de *threshold* definidos, é efetuado o pedido e processados os dados em formato de listagem. Por último, tal como descrito na implementação anterior, é efetuada a criação do corpo de email e de seguida o envio do mesmo para os destinatários selecionados. Conforme representado na Figura 5.25 é possível verificar o correto funcionamento da ferramenta implementada.

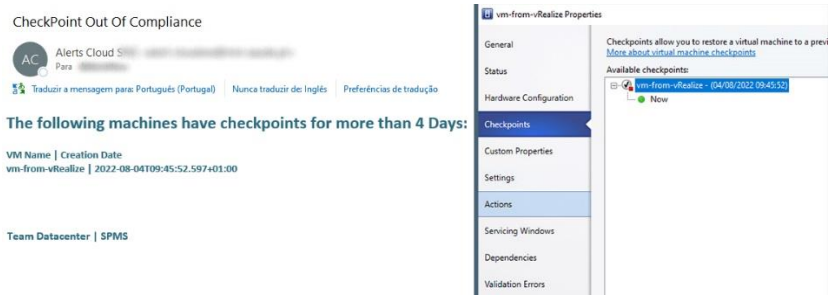


Figura 5.25 - Notificação de checkpoints fora do período autorizado

Em nota de conclusão, é possível apurar que o funcionamento das implementações cumprem com os requisitos propostos. Ademais, verificou-se que no

paradigma anterior o controlo destes recursos era efetuado descontinuamente num período inadequadamente definido, pelo que é também plausível de se concluir que a monitorização dos recursos se encontra mais organizada e simplificada.

5.7 Gestão de *Tokens* de Máquinas e Recursos em *Hyper-V*

O caráter ímpar do presente caso de uso torna complexa a comparação, uma vez que no paradigma anterior não existia mecanismo idêntico passível de fazer parte desta comparação. No entanto, é possível efetuar a análise de aplicabilidade da implementação com testes ponto-a-ponto que conferem o correto funcionamento da mesma.

Em primeiro lugar, e tomando como exemplo o caso de uso explorado nos testes do subcapítulo Criação de máquinas, verificamos as etapas e os resultados obtidos nas mesmas.

Antes ainda do processo de criação da máquina virtual, o *workflow* responsável pelo validação e cobrança dos *tokens* é invocado. Tendo em conta os *inputs*, é calculado o valor a cobrar e é chamada a opção *updateBalance* que envia um pedido de cobrança do valor representado na Figura 5.26.

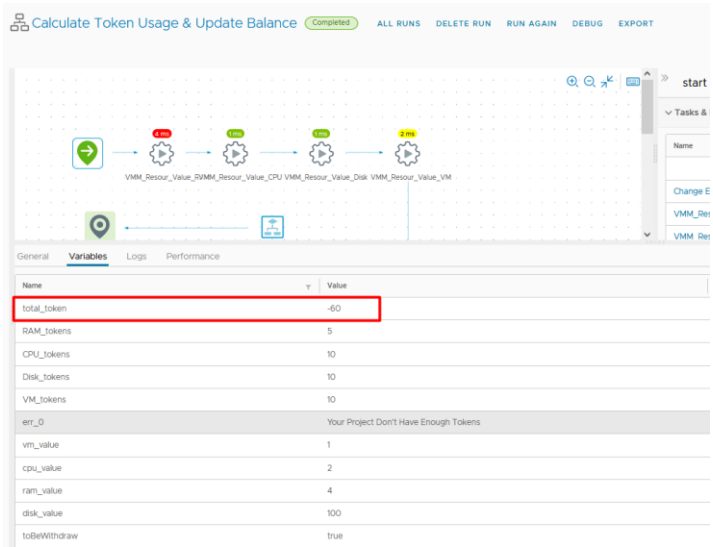


Figura 5.26 - Valor para adicionar ao saldo atual do projeto na plataforma

Na plataforma é possível verificar a receção de um pedido POST correspondente à cobrança em ação, ilustrado na Figura 5.27. Internamente, a plataforma efetua a validação se é plausível efetuar a cobrança do valor responde ao *vRealize* com o resultado.

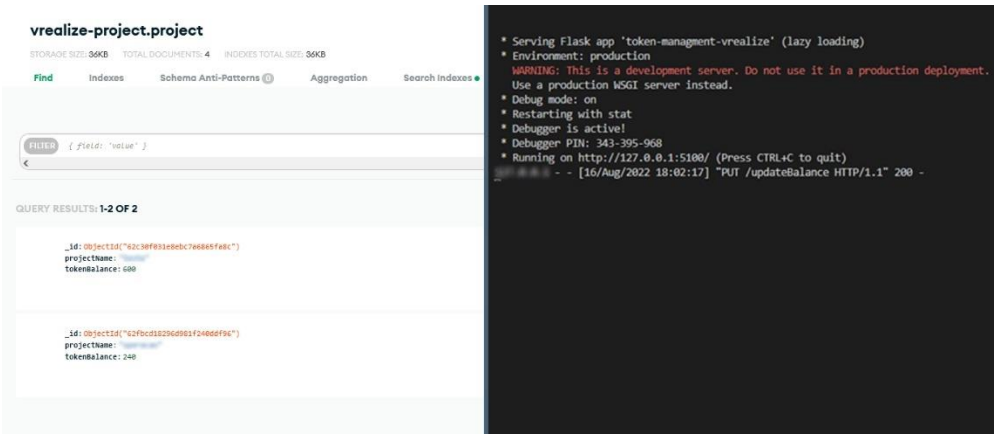


Figura 5.27 - Pedido de atualização de saldo recebido na plataforma & saldo original na base de dados (240 *tokens*)

Como ilustrado na Figura 5.28, é possível validar que a transação foi aprovada com sucesso e que o saldo do projeto foi atualizado para o 180 *tokens*, validando a integridade da solução.

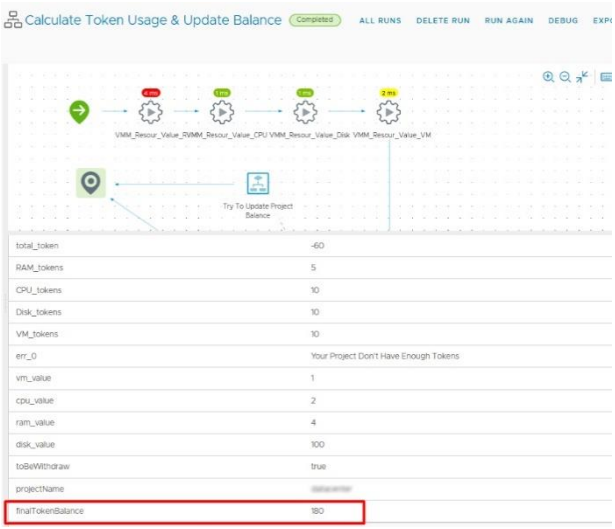


Figura 5.28 - Valor retornado do saldo atualizado do projeto

Deste modo, é possível concluir que o módulo cumpre com os requisitos apresentados, possibilitando o controlo dos recursos da infraestrutura. É também importante reforçar que a plataforma possui outras funções que não foram avaliadas no presente subcapítulo pela sua implementação idêntica, tornando redundante a avaliação do seu funcionamento. É de realçar ainda a utilidade da plataforma no controlo da infraestrutura, possibilitando oferecer ao subscritor cada vez mais autonomismo nas suas ações, enquanto disponibiliza um formato simples de controlo aos administradores.

Capítulo 6 - Conclusões

No decorrer do presente capítulo será apresentado o resumo e conclusão do trabalho desenvolvido, assim como as dificuldades sentidas no processo de implementação e desenho da solução. São também reforçados os objetivos cumpridos que foram inicialmente traçados, sendo discutidas as contribuições que os mesmos representam para a entidade SPMS em questão. Por último, são apresentados os pontos que não foram desenvolvidos devido a diversos constrangimentos na implementação e em adenda propostas as alterações futuras que agreguem valor à solução e à entidade SPMS.

6.1 Resumo do Trabalho Apresentado

De modo a realizar o presente trabalho de forma concisa, foi necessário, em primeiro lugar, entender como as plataformas de *Cloud* se organizam e operam de modo a oferecer aos seus subscritores, e consecutivamente clientes finais, uma plataforma que garanta qualidade de serviço elevada aliada à fácil configuração por parte dos subscritores. Deste modo, foi possível criar uma base de conhecimento coesa que possibilitou a análise dos casos de uso propostos de modo ponderado perante os requisitos gerais exigidos.

Para a implementação dos casos de uso foi imposta a necessidade de uso da plataforma *vRealize* da *VMware*, responsável por ser a solução central de controlo de recursos físicos e lógicos, ao mesmo tempo que disponibiliza uma página de gestão de recursos direcionada aos subscritores. O inevitável uso de plataformas terceiras está, em parte, contemplada na solução *vRealize*, no entanto muitas destas integrações necessitam de ser configuradas e, em grande parte do trabalho apresentado, personalizadas de modo a responder aos requisitos impostos inicialmente.

O processo de engenharia debruça-se então na criação de pontes de comunicações entre os diferentes ecossistemas necessários para o projeto, contemplando ao mesmo tempo a criação de mecanismos singulares que permitem o controlo e autogestão de processos inerentes aos casos de uso. Durante a implementação, foram explorados principalmente os recursos programáveis disponibilizados pela plataforma *vRealize*, o *scripting powershell* intrínseco à programabilidade do *Hyper-V*, assim como as capacidades de rápida e simples programação imanente à linguagem *Python*. Foram ainda utilizadas *REST APIs* que permitiram a comunicação e interligação com as diferentes plataformas, somando assim às tecnologias usadas no trabalho.

Assim, foi possível a integração de um *hypervisor* não suportado na solução, ao mesmo tempo que foram integrados métodos de monitorização e alarmísticas no sistema para auxiliar o controlo da plataforma, métodos de documentação de endereçamentos IP para manter o registo das máquinas em atividade em cada rede, assim como a integração com a plataforma de SDN da entidade SPMS, o que permite o isolamento das máquinas virtuais em domínios de *broadcast* diferentes.

Pela elevada ambição inicial do projeto, não foi possível garantir a implementação dos exemplos dados para o caso de uso de notificações, assim como a impossibilidade de *rollback* das configurações em caso erro no ato de eliminação ou criação de máquinas.

Por fim, foram apresentados os testes ponto-a-ponto de modo a comprovar a prova de usabilidade da solução apresentada, assim como a recolha dos tempos de resposta obtidos, de modo a assegurar a melhoria de serviços perante o paradigma anteriormente exercido.

6.2 Contribuições

As principais contribuições dos casos de uso implementados recaem, em primeiro lugar, pelos gestores dos inúmeros projetos relacionados com a entidade SPMS, e por último, ao consumidor final das plataformas desenvolvidas sobre as máquinas virtuais criadas e geridas.

A premissa da implementação da solução e dos casos de uso promovem também a melhoria nos tempos de resposta referentes a aspetos lógicos das máquinas virtuais, aliviando assim os administradores da infraestrutura de processos repetitivos e sequenciais, para se dedicarem a processos de engenharia.

Como contribuição adicional deste trabalho, pretende-se também estudar a possibilidade da escrita de um artigo científico que ilustre a solução desenvolvida e respetivos ganhos de eficiência obtidos.

6.3 Objetivos Por Cumprir

O sucesso das interações entre diferentes plataformas e *softwares* estão muitas das vezes dependentes do esforço e tempo dedicado para atingir uma determinada meta. O culminar de fatores à qual o presente trabalho foi submetido, como por exemplo

a meta temporal para entrada em produção das funções, a priorização de funcionalidades sobre outras ou ainda os prazos de conclusão do presente documento, originou, que alguns requisitos dos casos de uso não fossem concluídos ou que necessitem de revisões e melhorias em alguns pontos críticos.

Assim, e começando pelo principal requisito por cumprir, não foi possível disponibilizar ao subscritor um mecanismo viável de acesso à consola de sua máquina virtual em *Hyper-V*. O acesso à consola não é regularmente usado, no entanto em casos de problemas de placa de rede, é impossível ao subscritor aceder via RDP, pelo que neste cenário será necessário contactar os administradores da plataforma para validarem a situação. Este requisito, inerente ao caso de uso de importação e de criação de máquina, não foi possível concluir pela elevada complexidade de criação de uma plataforma que disponibilizasse a consola tendo em conta os acessos de cada utilizador do *vRealize*. Foi, no entanto, encontrado ao longo da pesquisa um software terceiro, o *Guacamole*[30], que possibilita a ligação à consola da máquina virtual diretamente de um serviço em HTML, possibilitando a melhor experiência de uso ao subscritor. A descoberta da plataforma foi tardia no processo de pesquisa, pelo que não foi suscetível de ser implementada.

O caso de uso de criação de máquina, especialmente criação de máquina em *Hyper-V* apresenta algumas lacunas de implementação pelo tempo despendido na mesma. Um dos casos é a incapacidade de registo no *IPAM* da máquina criada, uma vez que é o servidor DHCP a fornecer o IP disponível na rede. Esta implementação levanta um problema mais grave que é o facto de os servidores inicializarem com IP dinâmico, que caso o subscritor não altere de dinâmico para estático, preservando os dados atribuídos, que em caso de *reboot* existe a possibilidade de outra máquina assumir os valores originalmente atribuídos à mesma. Existem algumas formas de evitar este comportamento diretamente nos sistemas operativos, mas o mesmo não foi implementado. Nos trabalhos a desenvolver é explicado uma possível implementação para mitigar o problema.

6.4 Trabalho a Desenvolver

Apesar da solução, à medida do presente, já se apresentar em produção, existem sempre aspetos a melhorar na integração originalmente proposta e implementada.

Tendo em conta os 6.3 - Objetivos Por Cumprir, descritos anteriormente, a implementação da consola nas máquinas *Hyper-V* representa um ponto importante a ser

[30] guacamole.apache.org

estudado. Com a utilização da ferramenta *Guacamole* é possível disponibilizar uma plataforma terceira em que cada projeto e utilizador possui acesso às suas máquinas. A partir da API disponibilizada pelo *Guacamole* é possível importar novas conexões, sendo apenas necessário definir o nome da máquina virtual à qual será efetuada a conexão, assim como o nó de *hypervisor* de destino. Apesar da gestão a longo prazo se poder tornar desafiante, com recurso a *scripting* na plataforma *vRealize*, é possível gerir as máquinas e os subscritores perante as ações tomadas.

Outro aspeto que merece mais tempo de implementação é a atribuição de IPs a novas máquinas em *Hyper-V*. Uma possível interpretação do problema pode passar pela recolha de um IP disponível no IPAM e consequentemente reserva do mesmo diretamente no servidor de DHCP. Assim, apesar da máquina possuir um *IP* dinâmico, a sua alocação é estática estando garantida a sua reserva. Ao mesmo tempo, como existiria comunicação com o IPAM, também seria possível garantir a documentação das máquinas e dos seus IPs.

Apesar da integração com o *Hyper-V*, à medida do atual, apresentar as funcionalidades propostas, nem sempre é fácil trabalhar com o mesmo pelos *outputs* desformados e pelas execuções complexas, pelo que, numa integração futura que vise a melhoria de integração, recomenda-se ser criado um serviço que corra diretamente no *Hyper-V*. Este recebe diretamente via API os pedidos e os mesmos podem ser interpretados, executados e finalmente devolvida uma resposta ao requerente do pedido. Deste modo, é promovida a padronização dos métodos de comunicação, assim como a migração da complexidade para o *software* no *Hyper-V*, possuindo este melhores métodos de tratamento e manipulação de dados do que o *vRealize*.

Ao atual tratamento de máquinas eliminadas, verifica-se a eliminação total da informação da máquina, o que de um primeiro ponto de vista pode-se demonstrar benéfico, com o amadurecimento da solução pode-se tornar problemático. Assim, numa futura iteração, o desenvolvimento de métodos de gravação de histórico das máquinas, como por exemplo nome, data de criação/eliminação, histórico de ações, podem promover a um melhor fluxo de trabalho e documentação de ambientes passados.

Em modo de conclusão, o trabalho a desenvolver depende essencialmente do sucesso da solução e das opiniões tanto dos subscritores, como dos administradores. Deste modo, deve ser efetuada uma recolha de sugestões de melhoria da plataforma algum tempo após o seu lançamento e analisadas de forma a entender o que faz sentido implementar, se é possível, e qual o melhor método perante as circunstâncias exigidas.

Capítulo 7 - Bibliografia

[1] Cisco Systems, “Cisco Annual Internet Report,” San Francisco, 2018.

[2] C. Taurion, *Cloud Computing - Computação Em Núvem*, vol. 1. Rio de Janeiro: Brasport, 2009.

[3] Palo Alto Networks Inc., “Best Practice Assessment (BPA),” 2020. https://www.paloaltonetworks.com/services/bpa (accessed Nov. 16, 2021).

[4] S. Manvi and G. Shyam, “Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey,” *Journal of Network and Computer Applications*, vol. 41, pp. 424–440, May 2014.

[5] C. Jansseen, S. Donker, D. Brumby, and A. Kun, “History and future of human-automation interaction,” *Int J Hum Comput Stud*, pp. 99–107, Nov. 2019.

[6] K. Bekris, R. Shome, A. Krontiris, and A. Dobson, “Cloud automation: Precomputing roadmaps for flexible manipulation,” *IEEE Robot Autom Mag*, vol. 22, no. 2, pp. 41–50, 2015, doi: 10.1109/MRA.2015.2401291.

[7] D. Rountree and I. Castrillo, “Cloud Deployment Models,” *The Basics of Cloud Computing*, pp. 35–47, 2014, doi: 10.1016/B978-0-12-405932-0.00003-7.

[8] Red Hat Inc., “What is IaaS?,” *RedHat*. https://www.redhat.com/en/topics/cloud-computing/what-is-iaas (accessed Nov. 20, 2021).

[9] A. Apostu, F. Puican, G. Ularu, G. Suciu, and G. Todoran, “Study on advantages anddisadvantages of Cloud Computing – the advantages of TelemetryApplications in the Cloud,” *Recent Advances in Applied Computer Science and Digital Services*, 2013.

[10] A. Blenk, A. Basta, M. Reisslein, W. Kellerer, and S. Member, “Survey on Network Virtualization Hypervisors for Software Defined Networking,” *IEEE COMMUNICATIONS SURVEYS & TUTORIALS*, vol. 18, no. 1, 2016, doi: 10.1109/COMST.2015.2489183.

[11] VMware Inc., “What is a Hypervisor?” https://www.vmware.com/topics/glossary/content/hypervisor (accessed Dec. 10, 2021).

[12] P. Závacký, A. Eliáš, M. Strémy, I. Pavol Závacký, I. Andrej Eliáš, and D. Ing Maximilián Stémy, “HYPERVISOR FOR VIRTUALIZATION IN PRIVATE CLOUD,” vol. 23, 2015, doi: 10.1515/rput-2015-0030.

[13] C. Ayuya, “What is a Hypervisor Server?” https://www.serverwatch.com/virtualization/hypervisor-server/ (accessed Dec. 10, 2021).

[14] S. Ibrahim, B. He, and H. Jin, “Towards Pay-As-You-Consume Cloud Computing,” 2011. doi: 10.1109/SCC.2011.38.

[15] J. González-Moreno, M. Hortalá-González, F. López-Fraguas, and M. Rodríguez-Artalejo, “An approach to declarative programming based on a rewriting logic,” *The Journal of Logic Programming*, vol. 40, no. 1, pp. 47–87, Jul. 1999, doi: 10.1016/S0743-1066(98)10029-8.

[16] D. Weintrop and U. Wilensky, “To block or not to block, that is the question: Students’ perceptions of blocks-based programming,” *Proceedings of IDC 2015: The 14th International Conference on Interaction Design and Children*, pp. 199–208, Jun. 2015, doi: 10.1145/2771839.2771860.

[17] M. Virmani, “Understanding DevOps & bridging the gap from continuous integration to continuous delivery,” *5th International Conference on Innovative Computing Technology, INTECH 2015*, pp. 78–82, Jul. 2015, doi: 10.1109/INTECH.2015.7173368.

[18] A. M. Potdar, D. G. Narayan, S. Kengond, and M. M. Mulla, “Performance Evaluation of Docker Container and Virtual Machine,” *Procedia Comput Sci*, vol. 171, pp. 1419–1428, Jan. 2020, doi: 10.1016/J.PROCS.2020.04.152.

[19] VMware Inc., “What Is Code Stream.” https://docs.vmware.com/en/vRealize-Automation/services/Getting-Started-CodeStream/GUID-D137AB85-F66C-4A90-A710-66605FD0355B.html (accessed Jan. 28, 2022).

[20] Nutanix Inc., “Announcing Nutanix Cloud Manager.” https://www.nutanix.com/blog/announcing-nutanix-cloud-manager (accessed Aug. 18, 2022).

[21] Nutanix Inc., “AIOps & Automation Test Drive: Additional Playbook Ideas - Increase Memory of a VM Automatically | Nutanix Community.” https://next.nutanix.com/community-blog-154/aiops-automation-test-drive-additional-playbook-ideas-increase-memory-of-a-vm-automatically-39632 (accessed Aug. 18, 2022).

[22] Nutanix Inc., “Making a Mark with Nutanix Cloud Manager | Nutanix.” https://www.nutanix.com/blog/making-a-mark-with-nutanix-cloud-manager (accessed Aug. 18, 2022).

[23] Nutanix Inc., “Nutanix Cloud Manager Cost Governance.” https://www.nutanix.com/products/cloud-manager/cost-governance (accessed Oct. 15, 2022).

[24] Red Hat Inc., “Red Hat Cloud Solutions Life Cycle.” https://access.redhat.com/support/policy/updates/cloudsolutions/ (accessed Jan. 30, 2022).

[25] Red Hat Inc., “RED HAT CLOUD SUITE Integrated application development, cloud infrastructure, and management DATASHEET,” 2016.

[26] O. Sefraoui, M. Aissaoui, and M. Eleuldj, “OpenStack: Toward an Open-Source Solution for Cloud Computing,” *Int J Comput Appl*, vol. 55, no. 03, pp. 975–8887, 2012.

[27] Red Hat Inc., “Red Hat OpenShift makes container orchestration easier.” https://www.redhat.com/en/technologies/cloud-computing/openshift (accessed Jan. 31, 2022).

[28] Red Hat Inc., “What is KVM?” https://www.redhat.com/en/topics/virtualization/what-is-KVM (accessed Jan. 31, 2022).

[29] Red Hat Inc., “Understanding virtualization.” https://www.redhat.com/en/topics/virtualization (accessed Jan. 31, 2022).

[30] Red Hat, “Red Hat Satellite 6 (Datasheet),” 2018. [Online]. Available: https://www.redhat.com/rhdc/managed-files/ma-satellite-6-datasheet-f16450cs-201904-en_0.pdf

[31] Red Hat Inc., “Introduction to Red Hat Satellite 6.” https://access.redhat.com/documentation/en-us/red_hat_satellite/6.2/html/architecture_guide/chap-red_hat_satellite-architecture_guide-introduction_to_red_hat_satellite (accessed Feb. 05, 2022).

[32] K. Pepple, *Deploying OpenStack*. O’Reilly Media, Inc., 2011. Accessed: Jan. 30, 2022. [Online]. Available: https://books.google.com/books/about/Deploying_OpenStack.html?hl=pt-PT&id=gCK7z3PDXwMC

[33] R. Lingeswaran, “Openstack Architecture and components overview - UnixArena.” https://www.unixarena.com/2015/08/openstack-architecture-and-components-overview.html/ (accessed Feb. 15, 2022).

[34] OpenStack, “OpenStack Compute (nova) — nova 24.1.0.dev257 documentation.” https://docs.openstack.org/nova/latest/ (accessed Feb. 15, 2022).

[35] OpenStack, “Welcome to Swift’s documentation.” https://docs.openstack.org/swift/latest/ (accessed Feb. 15, 2022).

[36] OpenStack, “Welcome to Glance’s documentation.” https://docs.openstack.org/glance/latest/ (accessed Feb. 15, 2022).

[37] D. Padua, “THE FORTRAN I COMPILER,” *Comput Sci Eng*, vol. 2, no. 1, pp. 70–75, 2000, doi: 10.1109/5992.814661.

[38] R. Rojas, C. Göktekin, G. Friedland, and M. Krüger, “Plankalkül: The first high-level programming language and its implementation,” *Technical Report*, Feb. 2000.

[39] C. Gresse von Wangenheim, V. Rodrigues Nunes, and G. D. dos Santos, “Ensino de Computação com SCRATCH no Ensino Fundamental – Um Estudo de Caso,” *Revista Brasileira de Informática na Educação*, vol. 22, no. 03, p. 115, Nov. 2014, doi: 10.5753/rbie.2014.22.03.115.

[40] Y. H. Sung and Y. S. Jeong, “Development and application of programming education model based on visual thinking strategy for pre-service teachers,” *Universal Journal of Educational Research*, vol. 7, no. 5, pp. 42–53, 2019, doi: 10.13189/UJER.2019.071507.

[41] Creatio, “Low-Code and No-code.” https://www.creatio.com/our-technologies/low-code (accessed Feb. 15, 2022).

[42] OutSystems, “What is Low Code | Low Code Guide | OutSystems.” https://www.outsystems.com/guide/low-code/ (accessed Feb. 15, 2022).

[43] M. Lutz, *Programming Python*, Second., vol. 1. 2001. [Online]. Available: https://books.google.pt/books?hl=pt-PT&lr=&id=c8pV-TzyfBUC&oi=fnd&pg=PR11&dq=python+&ots=n57E5N_QPT&sig=b6JaoAPffH6KVDF065-xwFY8SAc&redir_esc=y#v=onepage&q=python&f=false

[44] A. Guha, C. Saftoiu, and S. Krishnamurthi, “The Essence of JavaScript,” *European Conference on Object-Oriented Programming*, pp. 126–150, 2010, doi: 10.1007/978-3-642-14107-2_7.

[45] E. O'Tuathail and M. Rose, "RFC 4227 - Using the Simple Object Access Protocol (SOAP) in Blocks Extensible Exchange Protocol (BEEP)," 2006 [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4227

[46] E. Rescorla, "RFC 2818: HTTP Over TLS," May 2000 doi: 10.17487/RFC2818.

[47] D. Mauro and K. Schmidt, *Essential SNMP: Help for System and Network Administrators*, vol. 1. 2005. [Online]. Available: https://books.google.pt/books?hl=pt-PT&lr=&id=65_0d25EpB4C&oi=fnd&pg=PT7&dq=SNMP&ots=H-sXumSjeQ&sig=5mxbO-rJkAoqivJXC6X3I5a8U60&redir_esc=y#v=onepage&q=SNMP&f=false

[48] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "RFC 1157 - Simple Network Management Protocol (SNMP)," 1990 [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1157

[49] "Simple Network Management Protocol." https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol (accessed Feb. 15, 2022).

[50] N. K. Singh, S. Thakur, H. Chaurasiya, and H. Nagdev, "Automated provisioning of application in IAAS cloud using Ansible configuration management," *International Conference on Next Generation Computing Technologies*, Sep. 2015, doi: 10.1109/NGCT.2015.7375087.

[51] Z. Pantić and C. Ho, "Guidelines for Building a Private Cloud Infrastructure-Technical Report Related papers Guidelines for Building a Privat e Cloud Infrast ruct ure," Universitetet i København, 2012.

[52] Microsoft Corporation, "Distinguished Names | Microsoft Docs." https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ldap/distinguished-names (accessed Apr. 20, 2022).

[53] VMware Inc., "Learn more about image mappings in vRealize Automation." https://docs.vmware.com/en/vRealize-Automation/8.4/Using-and-Managing-Cloud-Assembly/GUID-9CBAA91A-FAAD-4409-AFFC-ACC1810E4FA5.html (accessed Apr. 24, 2022).

[54] M. Parmar, "Create Network Profile in vRealize Automation | Mastering VMware." https://masteringvmware.com/vra-part-12-how-to-create-network-profile-in-vrealize-automation/ (accessed May 02, 2022).

[55] Jaykaran, "'Mean ± SEM' or 'Mean (SD)'?," *Indian J Pharmacol*, vol. 42, pp. 329–330, Oct. 2010, doi: 10.4103/0253-7613.70402.

Capítulo 8 - Apêndice

8.1 Texto

Adicionar placa de rede – Possibilita ao subscritor adicionar uma placa de rede das disponibilizadas pelos gestores da plataforma; Não efetua reserva direta no IPAM.

Alterar memoria RAM – Possibilita ao subscritor a alteração da memoria RAM de uma máquina virtual; Integrado com o módulo de *tokens*.

Eliminar/Criar/Restauro *Checkpoints* – Possibilita criar, eliminar ou restaurar *checkpoints* de uma máquina; À medida do atual encontra-se preparado para receber limite de *checkpoints* por máquina de modo a evitar má utilização de recursos computacionais.

Eliminar/Aumentar/Criar discos – Possibilita a gestão do armazenamento das máquinas; Integrado com o módulo de *tokens*.

Visualizar especificações da máquina – Oferece uma listagem detalhada do estado da máquina, do número de cores, memoria RAM instalada, tipo de sistema operativo, listagem de discos e listagem de placas de rede.

8.2 Imagens

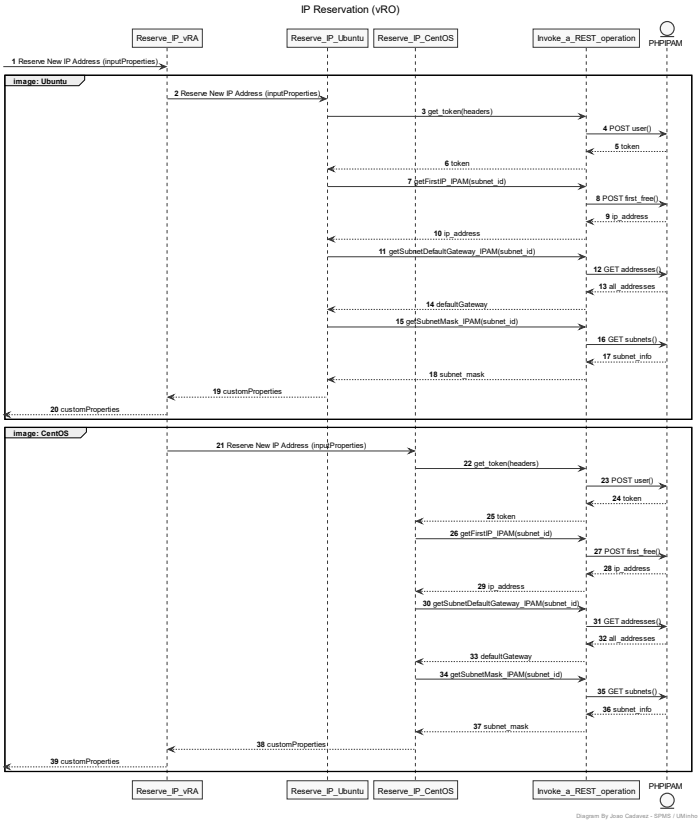


Figura 8.1 - Diagrama de sequência de reserva de IPs (*workflow*)

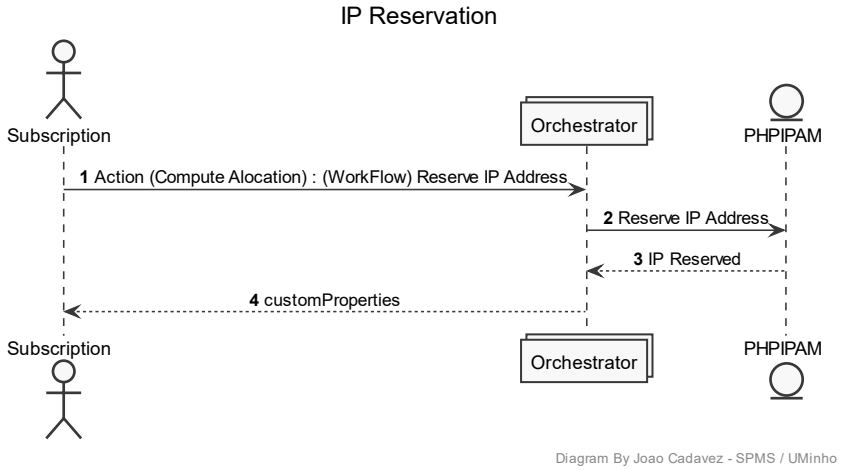


Figura 8.2 - Diagrama de sequência de reserva de IPs (Interno)

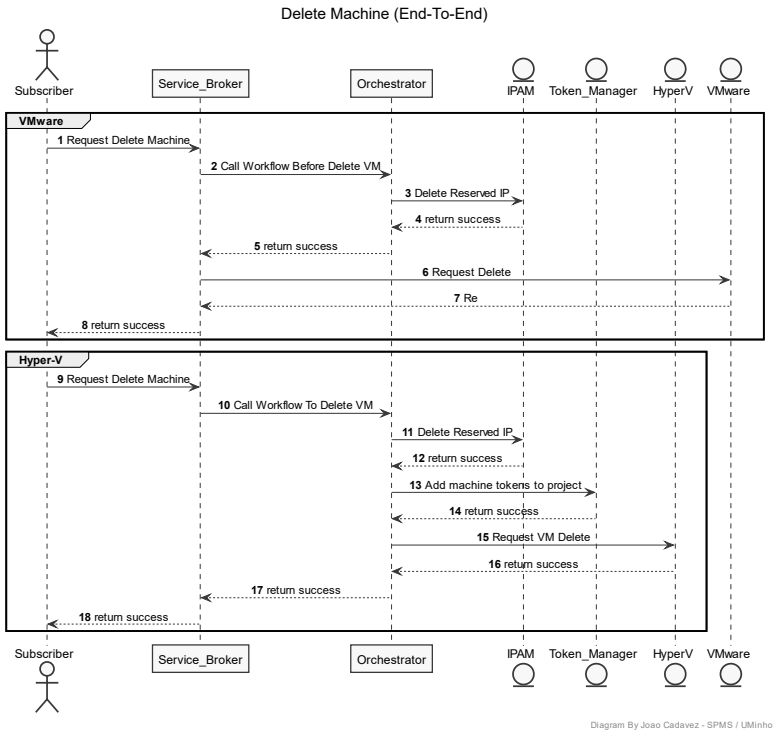


Figura 8.3 - Diagrama de Sequência de Reserva de IPs (*workflow*)

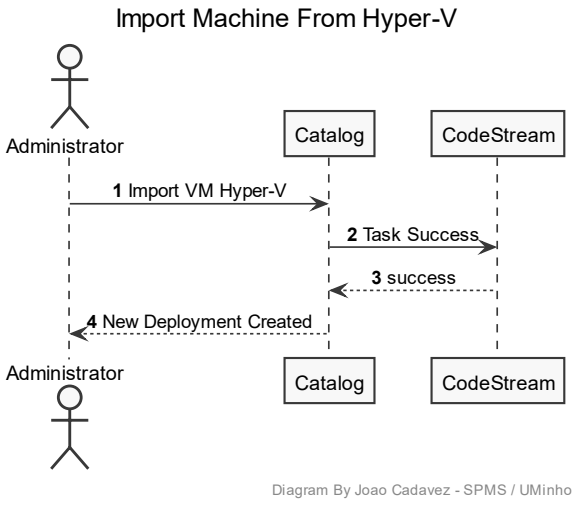


Figura 8.4 - Diagrama de sequência de importação de máquinas *Hyper-V*