Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Manuel Augusto Tarouca Martins

# Infrastructure as Code: Automatização do aprovisionamento e configuração de Infraestrutura e Serviços no Ensino Superior



Fevereiro de 2022

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Manuel Augusto Tarouca Martins

# Infrastructure as Code: Automatização do aprovisionamento e configuração de Infraestrutura e Serviços no Ensino Superior

**Tese de Mestrado**
Sistemas e Tecnologias da Informação para as Organizações

Professor Doutor Filipe Manuel Simões Caldeira



Fevereiro de 2022

*“The calamity of the information age is that the toxicity of data increases much faster than its benefits.”*

**Nassim Taleb**

# RESUMO

Atualmente, a grande maioria dos Sistemas de Tecnologias de Informação (STI) existentes nas instituições de ensino superior estão implementados segundo paradigmas tradicionais. O uso dessas abordagens dificulta a implementação de sistemas de alta capacidade que garantam escalabilidade e maior disponibilidade. Tendo por exemplos, a disponibilidade de plataformas de apoio ao ensino, e.g: Moodle ou, à flexibilidade de acesso remoto a laboratórios informáticos, os sistemas atualmente implementados precisam de dar resposta às necessidades atuais de forma versátil e eficiente.

Os avanços nos últimos anos no domínio de *Cloud Computing* e *Infrastructure as Code*, possibilitam uma fácil adoção ao paradigma das tecnologias de *Cloud Computing*. Estes avanços apresentam diversos benefícios para as instituições de ensino superior, através da simplificação da configuração dos sistemas, automatização de processos, redução de custos de infraestrutura em conjunto com a otimização da utilização dos recursos preexistentes assim como uma fácil adaptação aos diferentes requisitos que surgem da rápida e constante evolução dos STI.

Associados aos benefícios para as instituições de ensino superior, existe também um benefício transversal de melhoria do processo de ensino, principalmente no ensino de disciplinas de Engenharia Informática. A adoção deste paradigma oferece a este nicho, ferramentas e plataformas que contribuem para a melhoria da eficiência do ensino mediante a disponibilidade de laboratório informáticos virtuais, acesso remoto e de aprovisionamento e configuração de infraestrutura *on-demand*.

Contudo, a implementação e configuração destes processos é um problema complexo e que produz elevada entropia na configuração de vários sistemas heterogéneos. Neste âmbito, o presente trabalho visa propor e implementar uma arquitetura de *Private Cloud* recorrendo à plataforma Openstack no contexto do ensino superior e ao uso de ferramentas de aprovisionamento e configuração como o Ansible, oferecendo uma implementação estandardizada e automatizada. De forma a ilustrar a sua aplicabilidade são também apresentados e discutidos casos de uso que veiculam os processos com as plataformas propostas para serem utilizadas pela comunidade da instituição de ensino superior.

# ABSTRACT

Currently, most of the Information Technology Systems (ITS) existing in higher education institutions are implemented according to traditional paradigms. The use of these approaches makes it difficult to implement high-capacity systems that ensure scalability and greater availability. With examples of the availability of education support platforms, e.g., Moodle or the flexibility of remote access to computer labs, the systems currently implemented need to meet current needs in a versatile and efficient way.

The advances in recent years in the field of Cloud Computing and Infrastructure as Code, allow an easy adoption to the paradigm of Cloud Computing technologies. These advances present several benefits for higher education institutions, through the simplification of systems configuration, process automation, reduction of infrastructure costs in conjunction with the optimization of the use of pre-existing resources as well as an easy adaptation to the different requirements that arise from the rapid and constant evolution of ITS.

Alongside the advantages introduced for higher education institutions, there is also a cross-cutting benefit of improving the teaching process, especially in the teaching of Computer Engineering disciplines. The adoption of this paradigm offers this niche tools and platforms that contribute to the improvement of the efficiency of teaching through the availability of virtual computer labs, remote access and provisioning and configuration of on-demand infrastructure.

However, the implementation and configuration of these processes is a complex problem and produces high entropy in the configuration of various heterogenous systems. In this context, the present work aims to propose and implement a Private Cloud architecture using the Openstack platform in the context of higher education and the use of provisioning and configuration tools such as Ansible, offering a standardized and automated implementation. To illustrate its applicability, use cases that convey the processes with the proposed platforms to be used by the community of the higher education institution are also presented and discussed.

## PALAVRAS-CHAVE

ansible
openstack
automatização
infraestrutura
virtualização

## KEY WORDS

ansible
openstack
automation
infrastructure
virtualization

# AGRADECIMENTOS

Quero aproveitar este espaço para deixar umas palavras de agradecimento às pessoas que contribuíram de algum modo na produção deste documento.

Em primeiro lugar quero agradecer à minha mãe, Ermelinda Martins Tarouca, por todo o apoio e dedicação que me tem dado ao longo da minha vida sem nunca duvidar das minhas capacidades. Em segundo lugar quero agradecer à minha família e amigos pela motivação dada ao longo deste percurso. Menção especial para o professor e orientador Filipe Manuel Simões Caldeira por tudo o que me ensinou desde a licenciatura até agora e pela disponibilidade que ofereceu na orientação do desenvolvimento deste trabalho.

Quero também agradecer a todas as pessoas com que interagi ao longo da minha vida, pois o seu contributo permitiu formar a pessoa que atualmente sou.

# ÍNDICE GERAL

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABELAS

# ABREVIATURAS E SIGLAS

| | |
|---|---|
| IPV | Instituto Politécnico de Viseu |
| ESTGV | Escola Superior de Tecnologia e Gestão de Viseu |
| CC | *Cloud Computing* |
| TI | Tecnologias de Informação |
| API | *Application Programming Interface* |
| JSON | *JavaScript Object Notation* |
| YAML | *YAML Ain't Markup Language / Yet Another Markup Language* |
| REST | *Representational State Transfer* |
| OS | *Operating System* |
| VPN | *Virtual Private Network* |
| IP | *Internet Protocol* |
| DN | *Domain Name* |
| DNS | *Domain Name System* |
| CLI | *Command Line Interface* |

# 1. Introdução

Atualmente, a grande maioria dos Sistemas de Tecnologias de Informação (STI) existentes nas instituições de ensino superior estão implementados segundo paradigmas tradicionais. O uso dessas abordagens dificulta a implementação de sistemas de alta capacidade que garantam escalabilidade e maior disponibilidade. Tendo por exemplos, a disponibilidade de plataformas de apoio ao ensino, e.g: Moodle ou, à flexibilidade de acesso remoto a laboratórios informáticos, os sistemas atualmente implementados precisam de dar resposta às necessidades atuais de forma versátil e eficiente.

Os avanços nos últimos anos no domínio de *Cloud Computing* e *Infrastructure as Code*, possibilitam uma fácil adoção ao paradigma das tecnologias de *Cloud Computing*. Estes avanços apresentam diversos benefícios para as instituições de ensino superior, através da simplificação da configuração dos sistemas, automatização de processos, redução de custos de infraestrutura em conjunto com a otimização da utilização dos recursos preexistentes assim como uma fácil adaptação aos diferentes requisitos que surgem da rápida e constante evolução dos STI.

A necessidade de soluções mais ágeis no ensino das disciplinas na área da Engenharia Informática abre lugar à integração de tecnologias de Infraestrutura *as a Service* [1]. Este tipo de integração no processo de ensino, com a adoção de ferramentas e plataformas que contribuem para a melhoria da eficiência do ensino mediante a disponibilização de laboratório informáticos virtuais, acesso remoto e de aprovisionamento e configuração de infraestrutura *on-demand*, que são características dos ambientes nos quais os alunos realizam atividades práticas.

Com uma mudança de paradigma, as características que definem a *Cloud Computing* (CC) tais como elasticidade, flexibilidade, eficiência, confiabilidade e disponibilidade [2], podem ficar mais facilmente ao alcance dos alunos comparativamente ao método tradicional de

laboratório no local dos estabelecimentos de ensino, através do aprovisionamento e acesso remoto de infraestrutura virtual. Em situações em que o acesso físico aos estabelecimentos de ensino é limitado, por exemplo, devido às contingências de uma pandemia, que mobilize o ensino presencial para o virtual, este paradigma torna-se cada vez mais premente.

Em paralelo, a implementação do modelo de *Private Cloud* nos estabelecimentos de ensino oferece diversas vantagens para a instituição e os administradores de sistemas da mesma, exemplos disso são a redução de custos devido ao reaproveitamento de hardware não utilizado e otimização do seu uso, consequência da virtualização das suas funções (deixa de ser dedicado a um final específico e permite ser anfitrião a diversos tipos de recursos). Salienta-se também que pode ser libertada a carga horária empregue à manutenção dos recursos, visto que estes processos podem ser automatizados com o uso de ferramentas de configuração [3].

É importante referir que a implementação e configuração das plataformas que formam uma *Private Cloud* é um processo complexo e que produz elevada entropia na configuração dos sistemas. Apesar do fato de o conceito de *Cloud Computing* possuir diversas vantagens, este está associado a novas ferramentas e plataformas nas quais, muitas organizações podem ainda não possuir conhecimento especializado. No entanto este paradigma permite e promove o uso de tecnologias definidas por software, minimizando assim problemas como erros humanos de configuração, inconsistência da infraestrutura e a necessidade de conhecimento específico em determinadas áreas (sistemas, redes, armazenamento, etc.). As tecnologias definidas por software permitem descrever qualquer parte do STI que pode ser definida com software, desde a própria infraestrutura, até ao processo de implementação de STI [4].

Este trabalho, propõe e implementa uma solução de infraestrutura virtual, recorrendo à plataforma Openstack para a criação de uma *Private Cloud* no contexto do ensino superior. Para a solução proposta utiliza-se a ferramenta de aprovisionamento e configuração, Ansible, para oferecer uma implementação estandardizada e automatizada. Também são desenvolvidos casos de uso que veiculam os processos com as plataformas propostas para serem utilizadas pela comunidade da instituição de ensino superior, sendo apresentado como exemplo o processo de automatização da implementação e configuração, na *Private Cloud*, da plataforma de *e-Learning* Moodle.

O documento do projeto encontra-se estruturado da seguinte forma: no capítulo 2 é apresentado um estado de arte, onde conceitos, tecnologias e alternativas relevantes à proposta são discutidos. No capítulo 3, apresenta-se a metodologia de investigação e execução do projeto. No capítulo 4, a arquitetura do projeto é apresentada e examinada em conjunto com a implementação e configuração das diversas componentes da arquitetura proposta. No capítulo 5 é apresentado o ambiente de desenvolvimento e a metodologia utilizada para o desenvolvimento das Ansible *roles* e a sua estrutura, assim como uma explicação dos artefactos desenvolvidos que automatizam o processo de implementação da arquitetura. No capítulo 6, são propostos e debatidos os casos de uso que veiculam os processos da instituição de ensino superior com a arquitetura concebida. Por fim, no capítulo 7 são discutidas as

principais conclusões do trabalho realizado, apresentam-se as considerações finais e é delineado o trabalho futuro.

# 2. Estado de arte

Neste capítulo é apresentada um estado de arte sobre os conceitos e tecnologias relevantes para o desenvolvimento do projeto, o qual discute e fundamenta as soluções escolhidas, assim como as principais alternativas existentes.

## 2.1 Virtualização

O conceito de virtualização refere-se, geralmente, à capacidade de executar diversos sistemas operativos no mesmo hardware. Isto é a virtualização atua como uma camada entre o hardware físico e os sistemas que executam dentro deste, tipicamente chamados de máquinas virtuais, podendo-se referir que uma máquina virtual é a abstração de um computador físico. As máquinas virtuais, no entanto, por si só não conseguem comunicar com o hardware físico, deste modo, o *hypervisor* é a componente que atua como intermediária entre a camada física e as máquinas virtuais. O *hypervisor* é essencial para a virtualização sendo responsável por gerir os recursos alocados a cada máquina virtual [5].

Os *hypervisors* dividem-se em dois tipos [6]: os de tipos 1, que são sistemas operativos mais leves e instalados diretamente sobre o hardware físico. Como principais *hypervisors* tipo 1 destacam-se o VMWare ESXi [7], KVM [5], Proxmox [8]. Os *hypervisors* de tipo 2 são executados dentro de um sistema operativo em execução no hardware físico. São exemplos de *hypervisor* do tipo 2 [9] o Oracle VirtualBox e VMware Workstation. Comparativamente, podemos observar que os *hypervisors* de tipo 1 fornecem melhor performance relativamente aos *hypervisors* de tipo 2, principalmente pelo fato de executarem diretamente sobre o hardware reduzindo assim o *overhead* do *hypervisor*. A melhor performance faz com que em ambiente empresarial, o tipo 1 seja o *hypervisor* mais comum.

Um aspeto importante da virtualização [10] é a capacidade de permitir reduzir custos inerentes à infraestrutura de Sistema e Tecnologias de Informação (STI), isto é, em vez de alocar recursos em excesso para apenas servir um serviço, é possível utilizar o mesmo hardware para hospedar diversos tipos de serviços, e cada um de forma isolada e independente, melhorando também a eficiência de gestão dos recursos de hardware. Outro aspeto relevante é que ao existir independência entre as componentes também permite aumentar a disponibilidade destes, visto que problemas que possam ocorrer em uma componente virtual não irão afetar as restantes.

Devido aos benefícios apresentados relativamente à virtualização, as plataformas de *Cloud Computing* são hoje construídas sobre tecnologias de virtualização.

Recentemente, na indústria do desenvolvimento de STI, surgiu a necessidade de isolar as dependências que um determinado software executa. A criação de diversas máquinas virtuais, à partida parece a solução, mas a instalação de um sistema operativo para apenas separar dependências de software é exagerado e introduz atrito e reduz agilidade no desenvolvimento de software. De forma a aumentar a agilidade e facilitar a gestão surgiu a tecnologia de *containers* [11], que de forma semelhante às máquinas virtuais, são executadas em um contexto isolado. No entanto cada ambiente é executado sobre o mesmo sistema operativo, no qual cada *container* partilha os mesmos recursos com os demais. Isto permite com que esta abordagem seja ainda mais económica ao nível de otimização dos recursos. Alguns exemplos de software de *containers* são Docker [11][12] e LXC [12].

A administração, gestão e orquestração avançada de *containers* [13] pode ser efetuada através do software Kubernetes que permite automatizar diversas componentes de implementação e manutenção do ciclo de vida dos *containers*.

## 2.2 *Cloud Computing*

*Cloud Computing* (CC) [14] é a tecnologia que pretende abstrair o conhecimento da localização física do hardware e da configuração do sistema, do utilizador que usufruí de diversos tipos serviços tecnológicos normalmente fornecidos através da Internet. Desta forma, os custos indiretos da administração e configuração dos SI, são colocados no lado dos fornecedores destes serviços.

Comparativamente aos sistemas tradicionais, a CC oferece às organizações diversas vantagens que são caracterizadas por [15]:

- Disponibilizar dados, aplicações e outros serviços independentemente do dispositivo ou localização do utilizador.
- Necessitar de um menor conhecimento técnico para implementar os serviços.

- Disponibiliza serviços de alta disponibilidade, nomeadamente pelo fato de ser implementada em diversas localizações físicas.
- Nos casos no qual existe partilha de recursos entre utilizadores, o custo também irá ser partilhado o que permite a eficiência da utilização da infraestrutura.
- Manutenção fácil dos serviços, pois estes são independentes do utilizador.
- O utilizador apenas paga aquilo que usa através da disponibilização de métricas associadas aos seus consumos.
- A segurança pode ser tão boa ou melhor que que nos sistemas tradicionais, devido à alocação de recursos especializados em resolver problemas de segurança, por parte do fornecedor de serviços, o que muitas vezes em sistemas tradicionais ser ignorado. No entanto este tema é o um dos maiores obstáculos que impede a adoção das tecnologias *Cloud*, devido ao fator confiança que os clientes necessitam de depositar nos fornecedores dos serviços e por normalmente os serviços serem disponibilizados pela Internet, abre possibilidades de novos vetores de ataque.

### 2.2.1 Arquitetura dos serviços *Cloud*

A arquitetura dos serviços *Cloud* pode ser dividida em diferentes camadas, de acordo com as necessidades do consumidor, das quais as principais são *Software as a Service*, *Platform as a Service* e *Infrastructure as a Service*.

A primeira camada, *Software as a Service* corresponde ao fornecimento de aplicações através da Internet sem necessidade de instalar nos sistemas dos utilizadores (ex.: Microsoft Office 365, SalesForce, Dropbox, Slack, Gmail, etc.). Os utilizadores usufruem destes serviços de forma semelhante e com baixa capacidade de customização.

A segunda, *Platform as a Service*, corresponde à plataforma de computação onde os requisitos de software e hardware correspondem às necessidades do cliente. Por exemplo, neste tipo de plataforma pode ser oferecido aos desenvolvedores de software um ambiente onde é possível gerir o ciclo de vida do software, desde o desenvolvimento, teste, implementação e alojamento do software. Alguns exemplos de *Platform as a Service* são: AWS, Microsoft Azure, Red Hat OpenShift, IBM Cloud Foundry, Google App Engine, Oracle Cloud Platform. Esta camada permite maior customização dos serviços disponíveis nos catálogos dos fornecedores, que por sua vez irá aumentar a flexibilidade nos custos, no entanto a implementação e configuração irá ser mais complexa. É necessário salientar que o fornecedor da plataforma ainda possui diversas responsabilidades, sobre a gestão e manutenção dos sistemas operativos, assim como das redes e armazenamento da plataforma. Relativamente aos consumidores da plataforma, a sua responsabilidade apenas recai sobre o ciclo de vida das aplicações e dados.

Por último, a *Infrastructure as a Service*, permite ao utilizador usufruir de uma infraestrutura sem que este tenha de efetivamente investir na compra, instalação e configuração do hardware

necessário sendo apenas cobrado pelos recursos utilizados e tempo de utilização. Como consequência, os custos e o tempo de entrega dos serviços são reduzidos [15]. Alguns exemplos chave são DigitalOcean, Linode, Rackspace, AWS EC2, Azure IaaS, OpenStack, CloudStack, OpenNebula, Google Cloud Infrastructure, Oracle Cloud Infrastructure.

### 2.2.2 Métodos de implementação

Estão definidas quatro formas principais de implementação de soluções *Cloud* [14][16], dependendo essencialmente dos requisitos e necessidades dos clientes, dado que diferentes tipos de clientes possuem diferentes tipos de requisitos, quer ao nível do detalhe da gestão e configuração dos serviços, privacidade, segurança e inclusive escalabilidade. Em seguida são discutidos os quatro diferentes tipos de implementação das plataformas *Cloud*.

Em uma implementação *Public Cloud*, os recursos de hardware são geridos pelo fornecedor de serviços *Cloud*, e partilhados entre os seus clientes e estes apenas pagam pelos serviços que utilizam. Esta configuração permite reduzir os custos de TI das organizações assim como podem dispor de recursos de infraestrutura sem investimento prévio que permite deste modo capacidade de escalabilidade virtualmente ilimitada.

Importa referir que pelo fato de estar pública e consequentemente mais exposta, a *Public Cloud* poderá estar mais suscetível a ataques maliciosos. Este tipo de implementação é bastante adequado a organizações que pretendem uma configuração flexível e pré-configurada dos serviços *Cloud* sem necessidade de avultados investimentos em equipamentos e manutenção.

A *Private Cloud* é a contraparte da *Public Cloud*, e nesta os recursos de hardware estão presentes nos *datacenters* internos das organizações. Esta abordagem permite a gestão com mais controlo sobre a segurança dos sistemas e a sua manutenção e configuração. É de salientar que os recursos disponíveis na *Private Cloud* apenas são partilhados entre os utilizadores da organização.

A abordagem *Hybrid Cloud* é a combinação entre o modelo de *Public* e *Private Cloud*. É utilizado quando uma organização necessita de expor a sua *Private Cloud* a serviços externos, quer para aceder a partir da Internet ou para adicionar recursos da *Cloud* pública quando é necessário realizar tarefas que necessitam de mais recursos do que aqueles disponíveis nos *datacenters* da organização.

Por fim a *Community Cloud* é utilizada quando várias organizações constroem em conjunto uma infraestrutura *Cloud*, quer em um fornecedor externo ou nos *datacenters* das organizações, de acordo com requisitos e políticas partilhadas entre as partes [15].

A escolha acertada no tipo de implementação é de extrema importância por parte das organizações quando participam em um processo de migração dos sistemas tradicionais para este tipo de plataformas, pois esta escolha vai determinar a quantidade de esforço por parte dos administradores de sistemas e o nível de dificuldade e manutenção dos novos SI. Isto é se

a organização necessitar de reduzir a quantidade de administradores de sistemas, pode optar por uma transição para a *Public Cloud*, pois o fornecedor de serviços *Cloud* assumirá este papel.

### 2.2.3 Relevância da *Cloud Computing* no ensino superior

A par do exponencial crescimento mundial da adoção de tecnologias de *Cloud Computing*, estas também têm vindo a ganhar popularidade no meio académico (ensino superior) por oferecer inúmeros benefícios, mediante a disponibilidade de ferramentas de apoio ao ensino através da Internet, de laboratório informáticos virtuais, acesso remoto e de aprovisionamento e configuração de infraestrutura *on-demand*, e permitindo capacitar as instituições de ensino com ferramentas que permitem melhorar o processo de ensino, devido à inerente flexibilidade e redução de custos [17]. Também oferece às instituições localizadas em países em desenvolvimento, caracterizadas pela falta de recursos monetários, acesso a ferramentas mais económicas e a possibilidade de efetuarem uma redução de custos da infraestrutura [18].

Desta forma os principais benefícios da utilização de tecnologias de CC em instituições de ensino são:

- Oferecer um maior foco em atividades de ensino e investigação, através da redução do tempo dispensado e dificuldade do aprovisionamento e configuração manual de infraestrutura, serviços e aplicações.
- Redução a dificuldade de configuração de sistemas complexos de TI [19], através da estandardização dos processos de configuração dos sistemas de TI mediante o uso de ferramentas de configuração que reforçam o uso de boas práticas de configuração e minimizam o erro provocado pelas configurações manuais
- Possibilitar uma rápida implementação de sistemas de TI [20], visto que processos automatizados pelas ferramentas de configuração não necessitam de ser repetidos manualmente e podem ser reusados e integrados em casos de uso semelhantes.
- Aumentar o contacto dos estudantes com novas tecnologias, através da disponibilidade de novas plataformas e métodos de aprovisionamento, configuração e interação com infraestrutura virtual através de laboratórios virtuais em comparação com os laboratórios físicos normalmente imutáveis.

### 2.2.4 Plataformas de *Cloud Computing*

Apesar de os grandes fornecedores de serviços *Cloud*, entre os quais o Azure, AWS, Google, oferecerem plataformas para gerir os serviços e recursos *Cloud*, estes são proprietários e não permitem implementar e gerir uma infraestrutura *Cloud* privada de forma flexível, sem dependência dos fornecedores, com o risco de *vendor lock-in* [21]. Para colmatar este problema, com o objetivo de implementar uma plataforma *Cloud* para gerir a infraestrutura Privada de uma organização, existem projetos como por exemplo Openstack que oferecem

ferramentas para a implementação de uma plataforma *Cloud* que permite gerir a Infraestrutura privada *as a Service*. Outras alternativas mais conhecidas são [22] CloudStack e OpenNebula, no entanto o desenvolvimento ativo, dimensão da sua comunidade e a quantidade de ferramentas de configuração e automatização que suportam estas plataformas não se encontram ao nível do Openstack, que se evidencia como standard de facto.

A plataforma OpenStack é um projeto *open-source*, com desenvolvimento ativo, que reúne várias componentes de software, originalmente desenvolvidas pela NASA e Rackspace. Devido à sua robustez e alta disponibilidade, a plataforma Openstack é atualmente utilizada em soluções de *Public e Private Cloud*. Em consequência da alta modularidade e capacidade de configuração, permite ser implementada de acordo com os requisitos e necessidades da organização [23].

De entre as inúmeras componentes existentes na pilha de ferramentas que compõem a plataforma Openstack, podem-se destacar as seguintes como sendo as principais : Nova, oferece a capacidade de aprovisionar instâncias virtuais; Glance, permite gerir o armazenamento de imagens dos sistemas operativos da componente de virtualização; Neutron, gere a componente de rede (*Network as a Service*); Cinder, é a componente que oferece serviços de *Block Storage* para as instâncias da componente Nova. Por fim, a componente Keystone oferece o serviço de autenticação e autorização. Também existe a componente de interface para o utilizador chamada Horizon que oferece uma interface web que permite gerir as restantes componentes através de um *dashboard* intuitivo.

Devido à elevada interoperabilidade entre os diferentes componentes de software, à existência de APIs e ferramentas compatíveis com diversas linguagens de programação que a permitem usar, é possível automatizar e configurar a administração de uma solução de OpenStack, com ferramentas de automatização como por exemplo Ansible [24] e Terraform, e a ferramenta Openstack Heat que faz parte do leque das ferramentas suportadas de forma oficial pelo projeto Openstack.

## 2.3 *Infrastructure as Code*

*Infrastructure as Code* (IaC) é um novo paradigma de gestão e configuração da infraestrutura dos SI de modo descritivo que de forma semelhante ao desenvolvimento de software permite controlar as versões da infraestrutura através do versionamento dos ficheiros de código que descrevem a infraestrutura e configuração dos SI.

As tecnologias de IaC permitem transformar processos manuais realizados por administradores de sistemas em processos automatizados através de ferramentas de aprovisionamento e configuração.

As ferramentas de aprovisionamento e configuração permitem implementar uma infraestrutura de raiz e configurar os sistemas de informação através apenas de scripts e de ficheiros que definem a infraestrutura através de código.

Estes tipos de ferramentas trazem diversos benefícios para as organizações como por exemplo [4]:

- Sistemas descartáveis, consistência da configuração da infraestrutura.
- Tal como no desenvolvimento do software, uma vez que a infraestrutura e a sua configuração se encontram definida em ficheiros, é possível realizar o controlo de versões e implementação de sistema de integração continua.
- Aumento da eficiência no desenvolvimento de software.
- Orquestração de implementação avançada de sistemas de informação, através da interoperabilidade entre ferramentas, APIs e virtualização de infraestrutura.

No âmbito de automatização do aprovisionamento, administração e configuração de Infraestrutura, é possível afirmar que existem dois tipos de ferramentas de automatização, sendo que estas não são exclusivas a cada tipo [4]:

- As ferramentas de aprovisionamento, utilizadas especificamente para alocar os recursos nas plataformas de infraestrutura dinâmica (Infraestrutura as a Sevice), exemplos deste tipo de ferramentas: Terraform, OpenStack Heat, Ansible.
- As ferramentas de configuração, utilizadas para gerir as dependências e configurações dos recursos aprovisionados. Destacam-se nesta categoria, as ferramentas Chef, Puppet, SaltStack e Ansible.

Na Tabela 1 apresenta-se uma comparação de diferentes ferramentas de aprovisionamento e configuração, para diferentes métricas.

| **Métrica** | **Ansible** | **Chef** | **Puppet** | **Saltstack** | **Terraform** | **Openstack Heat** |
|---|---|---|---|---|---|---|
| **Configuração** | Sim | Sim | Sim | Sim | Não | Não |
| **Aprovisionamento** | Sim | Não | Não | Não | Sim | Sim |
| **Linguagens de configuração** | YAML | Ruby | DSL | YAML | HCL | YAML |
| **Arquitetura** | Sem agente | Com agente | Com agente | Com agente | Sem agente | Sem agente |

Tabela 1 – Comparação entre as ferramentas de aprovisionamento e configuração

### 2.3.1 Ansible

O Ansible é uma ferramenta *open-source* de aprovisionamento e configuração. Foi desenvolvida com o intuito de ser simples, segura e de fácil aprendizagem. A sua principal vantagem em relação às ferramentas da mesma categoria é que não é necessário instalar

nenhum agente nos sistemas geridos, isto é o Ansible acede remotamente desde um controlador master, a sistemas UNIX/LINUX pelo protocolo de comunicação SSH e em Windows por WinRM, reduzindo o *overhead* dos sistemas geridos, uma vez que não é necessário um agente com recursos constantemente alocados. Esta ferramenta também tem capacidade de comunicar com outras plataformas através de REST APIs. Adicionalmente pode-se dizer que é uma ferramenta idempotente, i.e., a ferramenta apenas modifica os sistemas geridos caso seja necessário. [4][25]

O Ansible utiliza ficheiros escritos na linguagem *Yet Another Markup Language* (YAML) chamados *playbooks* os quais descrevem o que será realizado em cada componente da infraestrutura. A estrutura dos *playbooks* consiste em uma lista de *plays* que descreve o conjunto de operações a realizar num certo conjunto de sistemas geridos, estes definidos em inventários. Cada play é constituída por um conjunto de tarefas (*tasks*). Cada *task* invoca um módulo de Ansible, que corresponde a ficheiros de código Python que vão ser executados nos sistemas geridos. Por último, os *playbooks* podem ser encapsulados em roles, as roles são utilizadas para aplicar configurações comuns em diferentes cenários de forma rápida e simples.

É necessário salientar que como o Ansible é uma ferramenta *open-source*, desenvolvida para ser extensível, permite desenvolver novos módulos, em qualquer linguagem de programação, desde que aceite input e retorne output no formato agnóstico JSON.

### 2.3.2 Ansible Tower (AWX)

O Ansible é um projeto open-source promovido pela empresa de software Red Hat. Esta empresa oferece uma solução comercial chamada Ansible Tower. O Ansible Tower é uma plataforma web utilizada para controlar e centralizar a infraestrutura TI através de um dashboard e possui controlo de acesso, agendamento de tarefas de execução de Ansible playbooks (armazenados em sistemas de controlo de versão), notificações integradas e gestão visual de inventário [26]. A versão upstream do Ansible Tower, chamada AWX, é gratuita, open-source e possuí mais funcionalidades que a versão estável Ansible Tower oferece, contudo, a Red Hat não oferece suporte para a versão gratuita.

Estas plataformas, permitem aos utilizadores com baixo conhecimento de aprovisionamento de infraestrutura e configuração de SI, lançar ou agendar processos de automatização que são aplicados sobre os sistemas e infraestrutura.

## 2.4 Sistemas de controlo de versões

Os sistemas de controlo de versões [27] são maioritariamente utilizados no desenvolvimento de software para rastrear e acompanhar as diferentes versões de código em desenvolvimento. Igualmente, permitem rastrear cada alteração e contribuição de cada desenvolvedor. Os

sistemas de controlo de versões permitem aos desenvolvedores trabalhar em diferentes versões do código simultaneamente. As principais vantagens de utilização de sistemas de controlo de versões é a capacidade de cada desenvolvedor trabalhar de forma independente, isto é, o desenvolvedor pode trabalhar no código e contribuir para o código principal a qualquer altura e da mesma forma, outro colaborador pode começar a contribuir em qualquer altura.

Existem diversos sistemas de controlo de versões, os principais são: Git, Team Foundation Server, Subversion, e Mercurial [28].

O Git é um sistema de controlo de versões *open-source* caracterizado pela sua fácil e leve capacidade de ramificação. Cada ramificação do código principal (ramo *master*) corresponde a uma cópia integral e única do ramo principal. Este ramo pode ser modificado e uma vez que o desenvolvedor se encontre satisfeito com o estado do ramo, pode fundi-lo com o ramo principal ou descartá-lo.

No contexto do uso de Infrastructure as Code [29], os artefactos de código desenvolvido para as ferramentas de configuração em conjunto com os sistemas de controlo de versão, permite criar um versionamento da infraestrutura de forma semelhante ao desenvolvimento do software. Desta forma é possível através da análise das diferenças de versões, detetar problemas inseridos com a atualização do código que aprovisiona e configura a infraestrutura. Este modelo permite também criar uma infraestrutura mais transparente pois facilita a documentação e auditoria desta.

### 2.4.1 Plataformas de controlo de versão

O Gitea [30], é uma plataforma web *open-source*, utilizada para armazenar e gerir repositórios Git remotamente, e também possuí outras funcionalidades de colaboração como rastreamento de bugs, bases de conhecimento (*wikis*) e suporte a *code reviews* (*pull requests*). As principais vantagens desta plataforma são: requisitos mínimos de recursos, capacidade de ser auto hospedado, ou seja, é possível controlar uma instância privada desta plataforma, configuração e implementação simples (através de Docker containers ou através de um binário condensado em um ficheiro), integração em sistemas de autenticação federados e alta capacidade de customização da plataforma, como por exemplo na possibilidade da escolha do sistema de gestão de base de dados utilizado pela plataforma.

Algumas alternativas a esta plataforma são Gitlab [31], Bitbucket [32] e Github [33], no entanto a plataforma Bitbuket e Github são proprietárias pelo que são pagas, não permitem gerir uma instância privada na própria infraestrutura privada da organização. Comparativamente ao Gitea, o Gitlab também permite ser gerido de forma autónoma, no entanto e é uma plataforma completa, isto é, rica em funcionalidades *out of the box* de apoio a processos de DevOps, e.g: *pipelines* de implementação/desenvolvimento contínuo (CI/CD), que por consequência necessita maior quantidade de recursos.

# 3. Metodologia

Com o objetivo de implementar com sucesso o projeto proposto foram seguidas um conjunto de etapas flexíveis e iterativas.

A etapa inicial é demarcada por um estudo do Estado de Arte, onde se investiga sobre trabalho já realizado por outros investigadores assim como sobre os conceitos, tecnologias e casos de uso relevantes para este projeto. Uma vez reunida uma base de conhecimento suficiente, começa-se por configurar e implementar o conjunto das componentes de software que dão suporte ao OpenStack, usando para isso uma máquina virtual dentro do *hypervisor* ESXi instalado no servidor físico.

O servidor físico disponibilizado inicialmente foi um equipamento Lenovo ThinkStation P310 (com CPU Xeon E3-1225 V5 3.3GHz, 64GB de memória RAM e 2TB de armazenamento em um disco rígido). Numa segunda fase, foi disponibilizado um equipamento Supermicro Super Server (com 2x Intel Xenon CPU E5-2640 v3 @ 2.60GHz, 128GB de memória RAM e 6TB de armazenamento em quatro discos rígidos). Ambos os servidores foram instalados em um laboratório informático da ESTGV e ligados à rede interna da escola. A implementação em dois sistemas distintos permite demarcar os requisitos mínimos para a implementação da arquitetura proposta.

Assim que o OpenStack se encontre implementado, configurado e testado no servidor físico, prossegue-se para a etapa seguinte na qual, recorrendo ao *hypervisor* ESXi, se configuraram as plataformas de automatização e os sistemas de controlo de versões para o código a desenvolver. Paralelamente, também se preparam e reúnem tarefas e casos de uso que pretende automatizar, utilizando como contexto, as plataformas implementadas.

A próxima etapa é demarcada pelo desenvolvimento dos artefactos de automatização, desde Ansible *roles* e *playbooks*, a *scripts* escritos na linguagem Python ou Bash, caso sejam necessários. O código gerado nesta etapa será armazenado e documentado dentro do sistema

de controlo de versões implementado anteriormente. Uma vez desenvolvidos estes artefactos, dá-se início à integração com a ferramenta de automatização. Nesta etapa é possível obter resultados, métricas e observações que mais tarde podem desencadear alterações e ajustes ao plano inicial. Se possível poderá ser implementado uma *pipeline* de integração e melhoria contínua no desenvolvimento dos artefactos de automatização, através de plataformas de integração contínua.

Os resultados obtidos de todas as etapas, são *input* essencial para o desenvolvimento deste documento, onde se pode refletir o trabalho e descobertas realizadas, que por sua vez também será exposto na apresentação do Projeto.

Este método de trabalho pode ser caraterizado por ser flexível, ágil, e entregar resultados em um curto intervalo de tempo, por isso é possível ser comparado a uma metodologia *Agile* [34]. Ao escrever código para gerir e configurar infraestrutura e serviços, ao utilizar ferramentas de automatização e controlo de versões, existe a possibilidade de integrar todo um processo em uma *pipeline* de *DevOps* [35], [36], [37].

# 4. Arquitetura e implementação

Neste capítulo será apresentada a arquitetura do sistema proposto bem como a fundamentação de cada componente deste sistema, incluindo as funções que cada componente desempenha, assim como algumas considerações adicionais e opções disponíveis para a sua implementação. Adicionalmente, apresenta-se a implementação das componentes da arquitetura do sistema proposto, acompanhadas de informação sobre a sua configuração juntamente com algumas considerações adicionais.

## 4.1 Arquitetura

A Figura 1 apresenta a arquitetura definida para o sistema proposto, a disposição das componentes do sistema proposto e a interação funcional entre estas. Os principais componentes são infraestrutura privada, *Virtual Private Network* (VPN), Plataforma de automatização, plataforma de *Infrastruture as a Service* e o agregado de serviços partilhados, os quais serão apresentados detalhadamente neste capítulo.

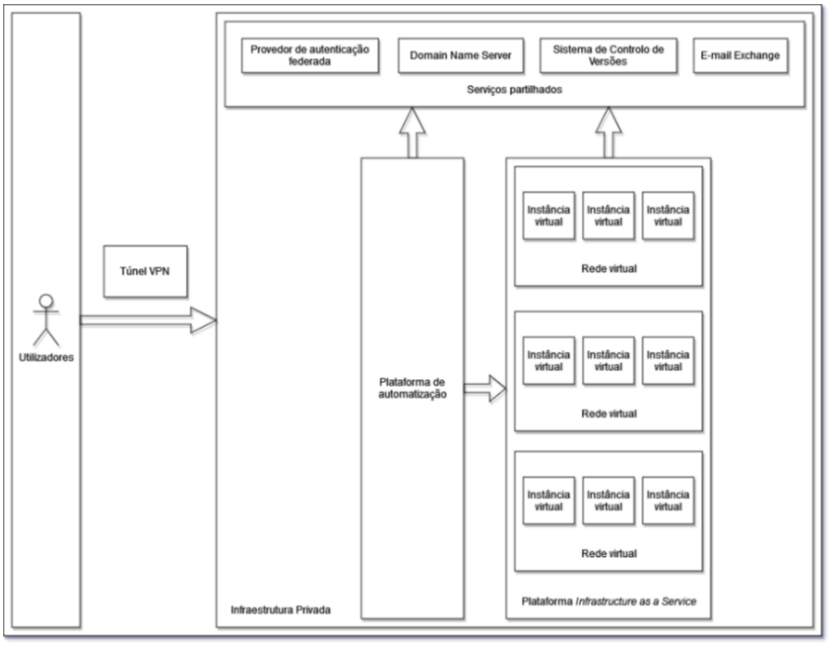


Figura 1 – Diagrama da arquitetura do sistema proposto

## 4.2 Componentes da arquitetura

Esta seção descreve as diversas componentes do sistema proposto no diagrama anterior e é acompanhada com a fundamentação de cada componente, que funções desempenha, em conjunto com algumas considerações adicionais e opções disponíveis para a sua implementação.

### 4.2.1 Infraestrutura Privada

Esta componente abrange o hardware que suporta a implementação das componentes da arquitetura proposta.

De modo a otimizar o processo de implementação das restantes componentes, a virtualização dos recursos físicos é recomendada, portanto surge a necessidade de utilização de *hypervisors*. Os *hypervisors* permitem deste modo interligar as diferentes componentes da infraestrutura física (servidores, redes e armazenamento) em uma plataforma de gestão de recursos virtuais. Existente, portanto, a possibilidade de integrar a infraestrutura privada da instituição com outras, tanto de infraestrutura de *cloud* pública como de infraestrutura privada de outras instituições. Apesar de esta proposta abordar uma implementação de uma *cloud* privada é também possível a integração em um modo híbrido com capacidade de alta disponibilidade e escalabilidade. As opções, de *hypervisores*, exploradas são VMWare ESXi e Proxmox. Nestas plataformas serão aprovisionadas as máquinas virtuais que implementam as restantes componentes do sistema proposto. Devido às elevadas necessidades de recursos físicos é recomendada a utilização de equipamentos de classe empresarial com capacidade de execução de *workloads* de elevado processamento.

### 4.2.2 *Virtual Private Network*

Esta componente é introduzida na arquitetura de forma a possibilitar acesso remoto através da internet à infraestrutura privada do sistema, com o objetivo de permitir aos utilizadores interagir com as componentes do sistema de forma segura tanto para a organização como para o utilizador. Esta abordagem, em contraparte com uma abordagem na qual as plataformas propostas estão expostas para a Internet, permite minimizar a superfície de ataque, possibilitando e autorizando apenas o contacto com as plataformas a quem se conectou à rede privada através de um túnel de VPN.

A introdução da componente *Virtual Private Network* é essencial em situações em que os utilizadores das plataformas não tenham acesso físico às instalações da instituição, por exemplo quando os alunos têm aulas através ferramentas de ensino online.

As opções de plataformas de software disponíveis e exploradas de implementação para a *Virtual Private Network* são OpenVPN e Wireguard [38].

### 4.2.3 Plataforma de automatização

A plataforma de automatização tem como objetivo permitir que os utilizadores possam iniciar, gerir e agendar os processos de automatização. Estes processos variam desde o aprovisionamento, gestão e configuração de recursos virtuais ou físicos, realizar tarefas de manutenção de sistemas, implementação e integração de serviços da organização, e orquestração dos anteriores de modo prático e compreensível. A plataforma por si só não possibilita a realização destes processos pelo que são necessários administradores de sistemas capazes de escrever os artefactos de automatização que realizem este tipo de tarefas. A plataforma fornece a possibilidade de agregar estes processos e permitir uma gestão mais centralizada com algumas funcionalidades que facilitam a iteração dos utilizadores com a infraestrutura e plataformas da arquitetura proposta, sem necessidade de interagir com uma *Command Line Interface* (CLI) para executar os artefactos de automatização.

As opções exploradas para as plataformas de automatização são o Ansible Tower/AWX que utiliza a ferramenta de gestão, configuração e aprovisionamento Ansible.

### 4.2.4 Plataforma de *Infrastructure as a Service*

A componente “plataforma de *Infrastructure as a Service*”, é o conjunto de software e ferramentas que permite a implementação de uma plataforma de *cloud* privada, permitindo proporcionar o aprovisionamento de redes, armazenamento e máquinas virtuais de modo flexível, elástico e *on-demand*. No contexto desta arquitetura, é sine qua non que estas plataformas tenham capacidade de interoperabilidade com ferramentas de aprovisionamento, por exemplo o Ansible, de forma a permitir que seja possível implementar um processo de automatização do aprovisionamento e configuração dos recursos geridos pela plataforma.

### 4.2.5 Serviços Partilhados

Os serviços partilhados são o agregado dos serviços que permitem o bom funcionamento e interoperabilidade das componentes do sistema, estes serviços caracterizam-se por fornecer suporte funcional e centralização de processos importantes para a comunicação entre os utilizadores e as plataformas, e vice-versa, assim como autenticação federada, envio de e-mails, e um sistema de gestão de domínios, de modo a possibilitar uma utilização simplificada das plataformas.

***Provedor de autenticação federada.***
Esta componente, provedor de autenticação federada, oferece a capacidade de fornecer identidade e autenticação centralizada na instituição. Esta componente permite interligar diversas plataformas do sistema utilizando a mesma identidade, removendo a necessidade de criação de contas de utilizador para cada plataforma. Deste modo, a gestão de utilizadores é realizada pela instituição e apenas são fornecidas as credenciais aos utilizadores autorizados. Assume-se nesta arquitetura que a componente "Provedor de autenticação federada" está já implementada na instituição, sendo apenas necessário configurar a sua integração nas plataformas implementadas.

***Domain name system***
O *Domain Name System* (DNS), nesta arquitetura assume o papel de centralizar o mapeamento entre endereços de IP de plataformas e domínios. Este mapeamento centralizado permite ao utilizador aceder facilmente às plataformas do sistema proposto através de endereços fáceis de ser interpretados (nomes) em comparação com endereços IP (números). Do ponto de vista técnico também permite uma fácil manutenção das componentes do sistema, pois quando é necessário modificar um endereço IP de uma componente, as restantes componentes dependentes desse sistema não vão necessitar de manutenção adicional, pois estas irão consultar o sistema centralizado de domínios pelo mapeamento mais recente.

As opções de plataformas de software disponíveis e exploradas de implementação de um sistema de DNS foram CoreDNS [39] e Dnsmasq [40].

***Sistema de controlo de versões (VCS)***
Durante o desenvolvimento de artefactos de automação, é recomendado armazenar o código desenvolvido em sistemas centralizados de controlo de versões de forma a armazenar e rastrear as diversas modificações ao longo do ciclo de vida do desenvolvimento. Os VCS executam estas funções e possuem interoperabilidade com as plataformas de automatização, pois, estas últimas necessitam de executar código localizado em repositórios. É comum também utilizar os VCS para armazenar ficheiros de texto contendo documentação detalhada sobre ao que cada repositório se refere, bem como outras considerações que o desenvolvedor considerar relevante.

O VCS explorado e utilizado durante o desenvolvimento e implementação do projeto foi o Git, com a utilização de plataformas como Gitea e Github.

***Email Exchange***

De modo a permitir notificar assincronamente os utilizadores dos resultados de processos de automatização, entre outros tipos de notificações relevantes, o e-mail é um método prático para concretizar este tipo de atividade. Geralmente, os sistemas de troca de e-mail, já estão previamente implementados nas instituições, pelo que, a geração de notificações relativas às plataformas de automatização poderá ser incorporada nos processos de comunicações já estabelecidos. Este tipo de notificações também permite informar os utilizadores através de mensagens sucintas e personalizadas sem a sobrecarga de informação que os resultados dos processos de automatização geram.

## 4.3 Implementação

A Figura 2 representa o diagrama de implementação das diversas componentes da arquitetura proposta assim como a sua interligação. Nesta secção será abordada em detalhe a implementação de cada uma destas componentes.

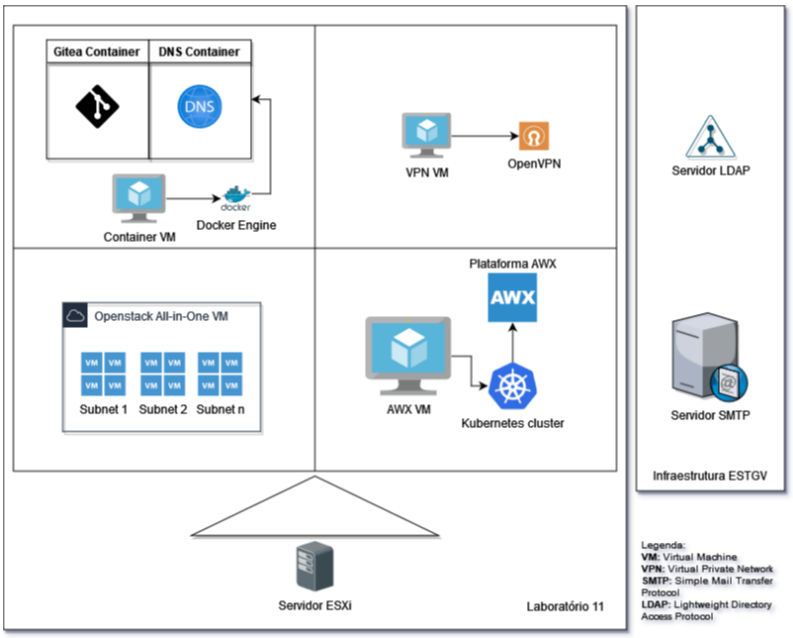


Figura 2 - Diagrama de implementação da arquitetura

## 4.4 Hardware

A implementação da arquitetura do sistema, decorreu inicialmente no Laboratório 11 da ESTGV, onde o hardware ficou instalado. Começou por ser implementada em um servidor de menores dimensões, que devido a limitações da sua capacidade de processamento foi substituído por um apto a atender aos requisitos.

Após a implementação da VPN (*Virtual Private Network*), em uma máquina virtual, foi possível continuar a implementação da restante arquitetura remotamente.

### 4.4.1 Especificações técnicas

Em seguida são apresentadas as especificações técnicas dos servidores utilizados no decorrer da implementação do sistema.

***Primeiro Servidor:***

- Lenovo ThinkStation
- Intel Xeon CPU E3-1225 v5 @ 3.30GHz
- 64GB RAM
- 2 TB de armazenamento

***Segundo Servidor:***

- Supermicro Super Server
- 2x Intel Xenon CPU E5-2640 v3 @ 2.60GHz
- 128GB RAM
- 6 TB armazenamento

## 4.5 Networking

No que diz respeito à topologia de rede, o servidor possui duas interfaces de rede, a primeira ligada à rede do laboratório e segunda exposta à internet através de um endereço IP público.

### 4.5.1 Rede do laboratório

A rede do laboratório atende às seguintes características:

Subnet: 192.168.80.0/24
Gateway: 192.168.80.254

***Configuração das interfaces de rede do Servidor***
As interfaces de rede do servidor foram configuradas com os parâmetros enunciados na Tabela 2.

| Interface | IPv4 | Gateway | DNS |
|---|---|---|---|
| Interface 1 | 192.168.80.111 | 192.168.80.254 | 192.168.0.2<br>192.168.0.7 |
| Interface 2 | confidencial | confidencial | confidencial<br>confidencial |

Tabela 2 - Configuração das interfaces de rede do servidor

### 4.5.2 Outras considerações

A arquitetura contou apenas com um servidor principal, no entanto existe a oportunidade de junção de nós para distribuir a carga de trabalhos e simultaneamente garantir redundância e maior disponibilidade. Por exemplo, visto que o primeiro servidor onde foram realizados testes iniciais não ter capacidade de atuar como um nó principal, este poderia atuar como um nó secundário quer de computação, *storage* ou de *networking* do serviço Openstack, ou ainda servir como um nó isolado que disponibiliza um serviço da arquitetura do sistema.

## 4.6 Componentes

Esta secção tem como objetivo introduzir cada componente do sistema juntamente com as suas características de configuração A configurações das máquinas após o sistema operativo estar completamente instalado foi realizado com a aplicação das Ansible *roles*, pelo que nenhuma configuração manual foi realizada sobre elas, exceto quando foi necessário proceder à execução de scripts *ad hoc*.

### 4.6.1 ESXi Host (Hypervisor)

O servidor físico executa o *Hypervisor* de tipo 1 ESXi v7.0.0 da VMware, que inicialmente foi escolhido por possuir as características mais práticas e de fácil configuração e compatibilidade com o hardware disponível.

No decorrer dos trabalhos surgiram algumas limitações, as quais, para uma prova de conceito não se consideram relevantes, contudo para uma implementação de qualidade de produção seriam fatores desqualificadores da escolha como a melhor opção.

As limitações referidas foram relativamente ao licenciamento, no qual a versão orientada para testes laboratoriais e escolares, apenas permite utilizar um número limitado de vCPUs (8) por máquina virtual, e a impossibilidade de utilização da API REST do *hypervisor*, que por sua vez impede a utilização de ferramentas de configuração que viriam a complementar a capacidade de automatização da implementação da arquitetura.

Sendo o ESXi um *hypervisor* de tipo 1, as restantes componentes da arquitetura foram implementadas como máquinas virtuais acomodadas dentro do *hypervisor*, e este assume toda a capacidade referida nas especificações técnicas.

***Configuração do sistema***

OS (*Operating system*): VMWare ESXi 7.0.0
Hostname: localhost
IP (Internet Protocol address): 192.168.80.111
DN (*Domain name*): esxi.iac.estgv

A Tabela 3 assinala as portas abertas deste sistema.

| Porta | Serviço |
|---|---|
| 443 | https |
| 22 | ssh |

Tabela 3 – Lista de portas abertas do ESXi Host

### 4.6.2 Servidor de VPN

Neste servidor é executado o serviço OpenVPN que permite estabelecer uma conexão VPN desde o exterior para a rede interna do laboratório da ESTGV, onde está localizada a infraestrutura física da arquitetura. Com este serviço foi possível realizar a investigação de forma remota, assumindo-se também como uma componente essencial do sistema, pois permite os utilizadores acederem aos recursos remotamente.

O serviço de VPN para além de criar um túnel virtual para a rede do laboratório também distribui para os clientes conectados o endereço do servidor de DNS (*Domain Name Server*).

Foi escolhido o software OpenVPN devido à facilidade de configuração e consequente implementação. Por possuir um painel de administração Web, tratar-se de software de código aberto e possuir integração com serviços de autenticação federada foram as razões adicionais para a escolha desta opção.

Embora o projeto OpenVPN seja gratuito e de código aberto a *appliance* utilizada para implementar o servidor de OpenVPN, oferece o OpenVPN Access Server que necessita de licenciamento adicional. Na versão *out of the box* é imposta a limitação na quantidade de utilizadores que se podem conectar simultaneamente. Face às limitações do OpenVPN Access Server e caso o serviço de VPN seja implementado de raiz, existe alternativas tais como o Wireguard, que também possui fácil configuração/implementação.

A *appliance* foi configurada com duas interfaces de rede, uma para ser exposta à internet e outra para conexão para a rede do laboratório.

***Configuração do sistema***
OS: Ubuntu 18.04 LTS (Virtual Appliance VMware ESXI)

Hostname: openvpn

IP: 192.168.80.113, confidencial

DN: vpn.iac.estgv

A Tabela 4 assinala as portas abertas deste sistema.

| Porta | Serviço |
|---|---|
| 443 | https |
| 22 | ssh |

Tabela 4 - Lista de portas abertas do servidor de VPN

### 4.6.3 Servidor de Docker Containers

Nesta máquina virtual, foi instalado o Docker Engine com o intuito de ser possível lançar diversos *containers* em apenas uma máquina virtual. Deste modo torna a arquitetura mais compacta e diminui o *overhead* de criação e configuração de máquinas no *hypervisor*.

Os dois *containers* que aqui atuam, respetivamente, o servidor de DNS e uma instância do sistema de controlo de versões de código Gitea, são serviços que partilham da mesma capacidade virtual, contudo, o serviço de DNS utiliza poucos recursos, pelo que o restante fica disponível para o serviço Gitea, que também conta como característica o baixo consumo de CPU e RAM.

A máquina foi configurada com 8 vCPUs, 8 RAM e 1 TB de armazenamento, será necessário realizar *benchmarks* em contexto de produção para inferir as reais necessidades técnicas e encontrar possíveis *bottlenecks* ou até mesmo verificar se existe capacidade de expandir a quantidade de *containers* aqui hospedados.

***Configuração do sistema***
OS: Ubuntu 20.04.02 LTS

Hostname: docker

IP: 192.168.80.200

DN: git.iac.estgv

A Tabela 5 assinala as portas abertas deste sistema.

| Porta | Serviço |
|---|---|
| 22 | git |
| 222 | ssh |
| 53 | dns |
| 3000 | https |

Tabela 5 - Lista de portas abertas do servidor de containers

***Considerações***
A porta do serviço SSH, que por omissão é a porta 22, foi remapeada para a porta 222, pois o servidor de *git* utiliza esta porta por omissão para clonar repositórios pelo protocolo SSH. Mudar a porta iria requerer configuração adicional desnecessária para os utilizadores do serviço Gitea.

***Serviço de DNS***
É neste *container* no qual é executado o serviço CoreDNS que permite correr o serviço de DNS utilizado para outros serviços da infraestrutura e também anunciado pelo servidor de VPN para que os seus clientes apontem para este servidor DNS. Desta forma é possível aceder aos serviços pelo domínio em vez de estar dependente de endereços IP que não são *user-friendly*. A gestão dos DN é realizada através da atualização do ficheiro de configuração.

O exemplo do ficheiro de configuração onde é mapeado o *domain name* para o endereço de IP correspondente pode ser consultado na Figura 3:

```
$ORIGIN iac.estgv.
@ 3600 IN SOA esxi.iac.estgv. mail.iac.estgv (
    2017042745
    7200
    3600
    1209600
    3600
    )
  3600 IN NS a.iana-servers.net.
  3600 IN NS b.iana-servers.net.

esxi IN A 192.168.80.111
git  IN A 192.168.80.200
vpn  IN A 193.137.7.19
awx  IN A 192.168.80.202
stack  IN A 192.168.80.204
devstack  IN A 192.168.80.206
```

Figura 3 – Ficheiro de configuração de mapeamento de *domain names*

O ficheiro que corresponde à definição das características do *container* pode ser consultado na Figura 4:

```
--- # docker-compose.yml
version: "3"
services:
  coredns:
    container_name: coredns
    image: coredns/coredns
    ports:
      - 53:53/udp
    volumes:
      - /root/data/coredns/:/root/
    command:
      - -conf
      - /root/Corefile
```

Figura 4 – Ficheiro de configuração do *container* de DNS

***Plataforma Gitea***

A plataforma Gitea atua como um sistema de controlo de versões de código no qual também é possível consultar documentação relativa ao código e fazer *issue tracking*.

Os utilizadores podem interagir com a plataforma Gitea através da aplicação Web e também existe a possibilidade de ser integrada com um sistema de autenticação federado.

Esta plataforma também permite aos utilizadores associarem as suas chaves públicas à sua identidade, que pode ser consultada publicamente através da REST API do Gitea e irá permitir realizar interoperabilidade entre o aprovisionamento de instâncias do Openstack, já com as chaves públicas autorizadas.

O *container* foi configurado para utilizar o Postgres [41] como motor de base de dados da plataforma. Esta base de dados também atua como um container separado, no entanto é encapsulado em um serviço que envolve o *container* Gitea e o container da base de dados.

O ficheiro que corresponde à definição das características dos *containers* pode ser consultado na Figura 5:

```
--- # docker-compose.yml
version: "2"
volumes:
  gitea-data:
    driver: local
  gitea-config:
    driver: local
services:
  server:
    image: gitea/gitea:latest-rootless
    environment:
      - GITEA__database__DB_TYPE=postgres
      - GITEA__database__HOST=db:5432
      - GITEA__database__NAME=gitea
      - GITEA__database__USER=gitea
      - GITEA__database__PASSWD=*censurado*
    restart: always
    volumes:
      - gitea-data:/var/lib/gitea
      - gitea-config:/etc/gitea
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
    ports:
      - "3000:3000"
      - "2222:2222"
    depends_on:
      - db

  db:
    image: postgres:13
    restart: always
    environment:
      - POSTGRES_USER=gitea
      - POSTGRES_PASSWORD=*censurado*
      - POSTGRES_DB=gitea
    volumes:
      - /root/postgres:/var/lib/postgresql/data
```

Figura 5 - Ficheiro de configuração dos *containers* da plataforma Gitea

### 4.6.4 Openstack

A plataforma Openstack é o sistema central desta arquitetura. É este sistema que permite aprovisionar e configurar pelos utilizadores a infraestrutura virtual através da ferramenta de configuração Ansible ou através da plataforma web AWX.

Foi utilizado como método de implementação o projeto Devstack - All-In-One Single VM, que permite através de um ficheiro de configuração inicial e um *script* de automatização implementar uma instância de testes do Openstack. Este método de implementação foi criado com o intuito de lançar rapidamente uma *test bed* de Openstack com configuração mínima e é bastante utilizado pelos desenvolvedores de componentes de Openstack para testar a implementação de novas componentes. Deste modo não será a melhor abordagem para implementar uma instância de nível de produção através deste método de implementação, mas neste contexto de investigação é o ideal, devido à complexidade de implementação de todas as suas componentes. O Devstack também permite implementar em outros tipos de topologias para além desta abordagem monolítica, como por exemplo em *Multi-Node Lab*, que distribui as componentes em diversos nós, no qual cada nó é configurado parar executar um ou mais componentes, através da configuração explicita no ficheiro de configuração inicial.

Existe outros métodos de *deployment* do Openstack mais robustos e customizáveis, que também permitem realizar operações de manutenção, alguns exemplos apresentados na Tabela 6:

| **TripleO** | **Kolla-ansible** | **Openstack-ansible** | **Openstack-charms** |
|---|---|---|---|
| Implementa o OpenStack utilizando o próprio OpenStack | Implementa o OpenStack em *containers* utilizando Ansible | Implementa atráves de Ansible *playbooks* e *roles* | Implementa em containers utilizando Charms e Juju da Canonical |

Tabela 6 – Métodos alternativos de implementação do Openstack

Informação adicional sobre os possíveis métodos de implementação Devstack podem ser consultados no endereço da referência [42].

Por omissão o método de *deployment* Devstack, enunciado previamente, instala apenas as componentes essenciais para implementar a instância de Openstack (*keystone, glance, nova, placement, cinder, neutron, and horizon*), mas permite adicionar *plugins* no ficheiro de configuração do *deployment* para realizar uma implementação, também automatizada, de componentes extras. Na página de documentação do projeto [43], pode ser consultado quais as componentes extras suportadas, do mesmo modo como desenvolver um *plugin* para implementação de uma componente customizada e configurar serviços extras como por exemplo, realizar a integração de um sistema de autenticação federada.

As versões do Openstack são lançadas em ciclos de 6 meses, sob um nome de código e possuem datas-limite a partir das quais deixam de ser mantidas. Assim, é relevante mencionar que a utilizada nesta implementação foi a versão *Wallaby*, com data-limite de manutenção a 2022-04-27, outras versões, *Victoria e Ussuri*, também foram testadas.

A configuração definida nesta implementação apenas customizou as *passwords* a serem utilizadas para os serviços, a restrição explicita da utilização de apenas endereços IPv4, e a gama de alocação de *floating* IPs para as instâncias criadas dentro do Openstack. Neste caso,

visto que a rede do laboratório se encontra na gama dos 192.168.80.0/24, foi atribuída a gama desde o endereço 192.168.80.230 ao endereço 192.168.80.250. As opções de configuração adicionais podem ser consultadas na Figura 6.


Figura 6 – Ficheiro de configuração do Devstack

Falta salientar que o servidor virtual onde foi implementado foi configurado com as seguintes características: 64 GB RAM, 8 vCPU, 2 TB de armazenamento e relativamente à configuração das interfaces de rede, duas foram ligadas à rede do laboratório, uma para ser gerida pelo componente de *networking* (*neutron*) do Openstack e outra interface para administração do sistema.

***Configuração do sistema***

OS: Ubuntu 20.04.02 LTS

Hostname: devstack

IP: 192.168.80.204, 192.168.80.206

DN: devstack.iac.estgv

A Tabela 7 assinala as portas abertas deste sistema.

| Porta | Serviço |
|---|---|
| 22 | ssh |
| 80 | http |
| 9696 | neutron |
| 5000 | keystone |
| 8774 | nova |
| 9292 | glance |
| 8776 | cinder |
| 8003 | placement |
| 3260 | iSCSI target |
| 3306 | MySQL |
| 8003 | Message Broker |

Tabela 7 - Lista de portas abertas do servidor de Openstack

***Considerações***

O método de *deployment* Devstack é testado continuamente pela comunidade que desenvolve o projeto e garante a implementação sem problemas na última versão *Long Term Support* (LTS) da distribuição Ubuntu e nas versões mais recentes da Fedora, CentOS/RHEL e OpenSUSE. No entanto durante a investigação, a implementação que correu melhor foi na distribuição Ubuntu, pelo que nas restantes surgiram sempre problemas inesperados e sem soluções evidentes.

Foram necessárias aplicar configurações adicionais na firewall e nas interfaces de rede da máquina virtual para conseguir estabelecer conectividade entre as instâncias criadas dentro do Openstack e o resto da rede externa.

Relativamente ao armazenamento, foi necessário configurar manualmente *volume groups* no *Logical Volume Manager* do sistema operativo, para a componente *Cinder* os conseguir administrar.

### 4.6.5 AWX

A plataforma AWX apresenta-se como a plataforma Web na qual os utilizadores vão interagir com o a gestão da infraestrutura do Openstack. Apesar de o Openstack possuir a sua própria interface Web de gestão, o AWX irá facilitar os utilizadores através do lançamento de fluxos de automatização sobre a configuração e gestão da infraestrutura no Openstack. Deste modo os utilizadores não necessitam de realizar longos processos de configuração na interface do Openstack e será possível realizar o agendamento de tarefas.

A plataforma também oferece, através da sua REST (*Representational State Transfer*) API, capacidade de automatização da criação e gestão de recursos da plataforma (organizações, projetos, trabalhos).

A plataforma AWX é executada na máquina virtual em um cluster de Kubernetes. Este cluster foi implementado com as ferramentas *awx-operator* e *minikube* que permitem realizar a instalação e manutenção da plataforma em um ambiente de Kubernetes local. O driver utilizado para o cluster de Kubernetes foi o Docker.

O AWX oferece vários métodos de autenticação, quer para contextos de automatização através de *tokens*, ou para utilizadores finais com autenticação tradicional nome de utilizador/password e autenticação federada.

***Configuração do sistema***
OS: CentOS 8.3

Hostname: awx

IP: 192.168.80.202

DN: awx.iac.estgv

A Tabela 8 assinala as portas abertas deste sistema.

| Porta | Serviço |
|---|---|
| 443 | https |
| 22 | ssh |

Tabela 8 - Lista de portas abertas do servidor de AWX

***Considerações***
O projeto AWX está em desenvolvimento contínuo pelo que as versões mais recentes nem sempre são as mais estáveis, em comparação com a alternativa paga, a versão Ansible Tower, é mais estável e conta com suporte oficial da Red Hat.

Relativamente ao sistema operativo da máquina virtual, foi utilizado a distribuição Linux CentOS 8.3, a qual deixou de ser mantida a partir do mês de dezembro de 2021. Alternativas a este sistema operativo surgiram, entretanto, como por exemplo Rocky Linux que continuam a executar a mesma função que o projeto CentOS inicialmente propunha.

### 4.6.6 Serviços internos da ESTGV

Entre os serviços que se caracterizam como serviços internos da ESTGV, destacam-se o serviço de autenticação federado LDAP, e um servidor e-mail.

É sugerido o sistema de autenticação LDAP, porque é suportado por todas as plataformas, OpenStack, AWX e Gitea, de forma a unificar o método de autenticação dos utilizadores do sistema.

O servidor de e-mail surge com a finalidade de distribuir notificações desde a plataforma AWX (execução de tarefas, resultado de tarefas) aos utilizadores relevantes.

A proposta desta componente é apenas teórica visto que estes serviços não foram implementados nem integrados no sistema, mas surgem como opções relevantes e essenciais para atingir um sistema com nível de produção.

# 5. Desenvolvimento de Ansible *roles*

A implementação desta arquitetura foi conseguida através do desenvolvimento de diversas roles de Ansible. O desenvolvimento destas roles possibilitou a criação de fluxo de trabalho que permite destruir e recriar novamente a arquitetura do sistema de acordo com as configurações definidas, por outro lado também é capaz de adicionar/remover configurações específicas sem destruir o progresso já alcançado.

Este capítulo apresenta o ambiente de desenvolvimento, a metodologia para o desenvolvimento das *roles* e a sua estrutura, assim como uma explicação dos artefactos desenvolvidos.

## 5.1 Ambiente de desenvolvimento das Ansible *roles*

O ambiente de desenvolvimento utilizado para a criação e implementação das Ansible *roles* possui as seguintes características:

- Visual Sudio Code como o IDE de desenvolvimento.
- Windows Subsystem for Linux (Ubuntu 20.04) como sistema operativo para desenvolver, através de conexão remota, e disputar os *playbooks* desde a linha de comandos, e.g: *ansible-playbook -i inventory <your-playbook.yml>*.
- Python *virtual enviroment*, para encapsular as dependências Python, e.g: *python -m venv iac-env*.
- Web *console* do *hypervisor* VMware ESXi para criação da infraestrutura virtual.
- Customização do ficheiro de configuração do Ansible e.g: */etc/ansible/ansible.cfg*.
- Inventário dos sistemas a serem configurados, e.g: *./iac.estgv/inventory.ini*.

- Ansible Galaxy para criar um esqueleto básico das *roles* de Ansible: e.g: *ansible-galaxy role init <namespace.nome-da-role>*.
- Software Git como sistema de controlo de versão de código desenvolvido juntamente com as plataformas Gitea e Github como repositórios remotos.
- Mono repositório privado na plataforma Github para controlo de versão do código e também como *fallback* o Gitea [44].
- Software KeePassXC para criar e guardar credenciais de forma encriptada.
- Software Joplin para documentar configurações, notas e outras considerações sobre a implementação da arquitetura.

## 5.2 Metodologia de desenvolvimento das Ansible *roles*

A metodologia utilizada para o desenvolvimento integra um conjunto de passos que foram seguidos de forma a desenvolver Ansible *roles* seguindo as melhores práticas. Sucintamente os passos podem ser descritos do seguinte modo:

Definição de requisitos - nesta etapa são reunidos os objetivos da role de forma genérica, eventualmente o nome da role traduz-se em uma abreviação do objetivo final. Por exemplo instalar o software *docker* em um servidor, irá traduzir-se em "*setup-docker*".

Dividir os requisitos em tarefas atómicas - para chegar a um objetivo é necessário definir um conjunto de tarefas que são necessárias executar para esse objetivo se concretizar, no entanto esta definição de tarefa terá se ser o mais granular possível.

As tarefas atómicas deverão ser possíveis de ser aplicadas segundo o princípio de idem potência de forma a minimizar as alterações aos sistemas alvo. Isto é, as tarefas podem ser executadas várias vezes sem que o resultado da operação não modifique após uma primeira aplicação inicial.

Detetar padrões repetitivos (e.g: portas, nomes de serviços) e colocar em variáveis de forma a *refatorar* o código.

Encriptar credenciais ou informação confidencial, caso necessário, através do Ansible Vault.

É necessário salientar que a documentação de pontos específicos a ter em conta dentro do âmbito de atuação da role é essencial para que as roles sejam facilmente utilizadas por outros utilizadores. Essa documentação deve ser escrita no ficheiro *README.md* de cada role.

## 5.3 Implementação da arquitetura com Ansible *roles*

Esta secção apresenta a estrutura das componentes dos artefactos de automatização em conjunto com a descrição dos artefactos desenvolvidos.
Foi criado um repositório *git* para o desenvolvimento das *roles* que vão dar suporte à implementação da arquitetura, chamado *iac.estgv*. Aquando da geração do esqueleto das *roles* foi determinado um *namespace* seguido de um ponto e nome da *role*, pelo que se traduz no seguinte exemplo *iac.nome-da-role*. A utilidade desta nomenclatura ajudará quando são utilizadas em conjunto com *roles* de terceiros.

A utilização de *roles* de terceiros permite tornar os *playbooks* mais compactos e reduz a necessidade de criar roles adicionais. Na plataforma Ansible Galaxy existe diversas *roles* criadas pela comunidade, por exemplo para instalar uma *role* de terceiros que tem como objetivo instalar o software *php*, executamos o comando *ansible-galaxy install geerlingguy.php* para ser possível usar nos *playbooks* desenvolvidos.

### 5.3.1 Inventário

Os inventários são utilizados como uma fonte de verdade pelo qual o Ansible irá conectar-se a cada máquina enunciada. Por omissão, caso não seja especificado nenhum inventário, o Ansible utiliza o ficheiro na pasta */etc/ansible/hosts* como inventário.

A Figura 7 apresenta o conteúdo de um inventário. Neste ficheiro é possível enumerar para cada *host* diversas variáveis como por exemplo o endereço pelo qual se irá conectar, que protocolo usar, exemplo SSH e que credenciais usar. Também existe a possibilidade de realizar manipulações como por exemplo agrupamento de *hosts*.

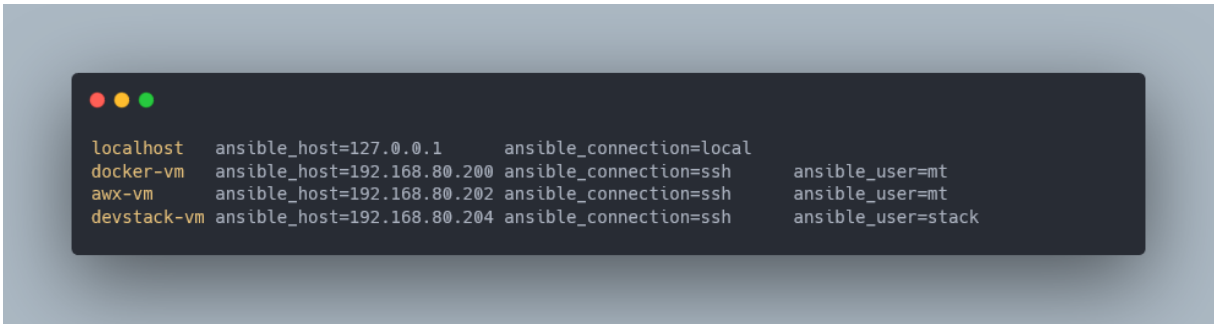


Figura 7 – Estrutura exemplo de um ficheiro de inventário do Ansible

### 5.3.2 *Playbooks*

Os *playbooks* são escritos na linguagem de serialização YAML e executam as *roles* desenvolvidas sobre um determinado inventário, para além disso podem também conter tarefas específicas sem estarem encapsuladas em *roles*.

A Figura 8 apresenta um exemplo de um *playbook*. Nos *playbooks* é permitido enunciar os alvos deste, definir ou sobrescrever variáveis, definir tarefas, e indicar que *collections* usar e que *roles* executar.

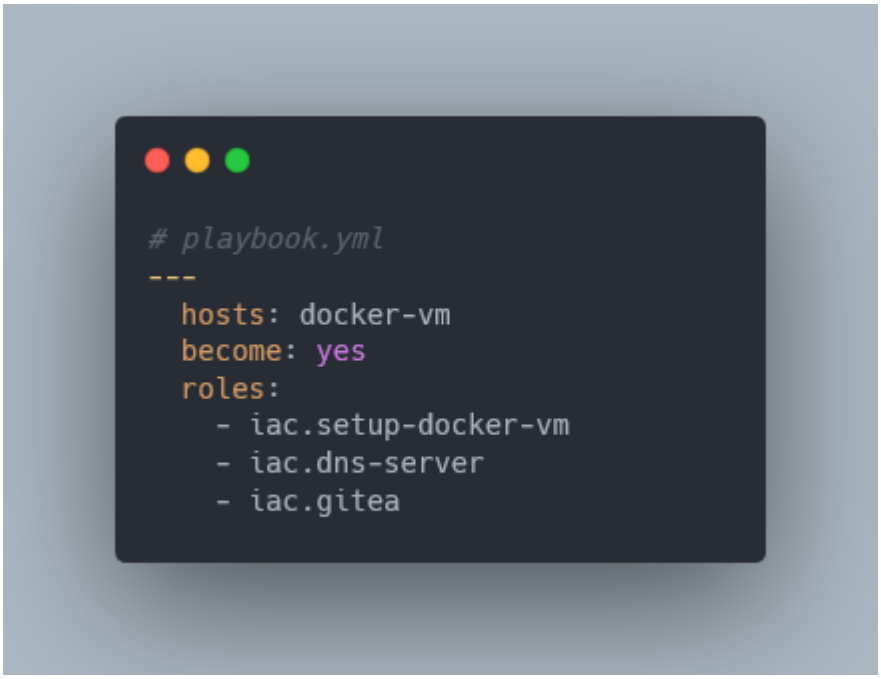


Figura 8 – Estrutura exemplo de um *playbook* do Ansible

A Figura 9 exibe a estrutura básica de uma tarefa. Neste contexto é dado um determinado nome para ser naturalmente compreensível e é chamado um módulo de Ansible que executa operações sobre os alvos. Neste nível ainda é possível definir e sobrescrever variáveis, delegar a tarefa a outros alvos, indicar condições de execução e ainda registar o resultado da tarefa em variáveis.

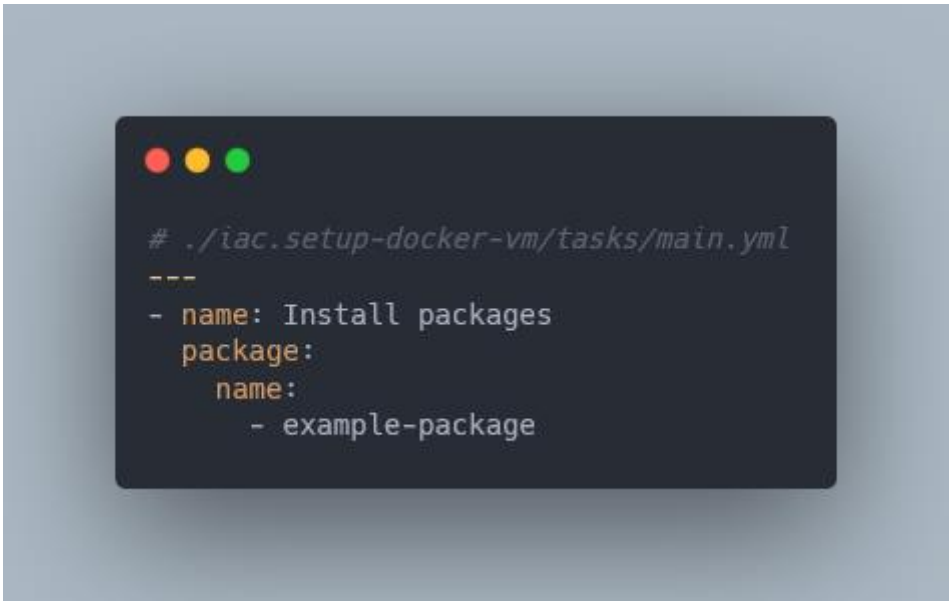


Figura 9 – Estrutura exemplo de uma task do Ansible

### 5.3.3 *Roles*

A Figura 10 exibe o esqueleto de uma role. Cada pasta organiza ficheiros consoante o seu contexto, tipicamente chamados *main.yml*, por vezes é necessário dar outros nomes em *roles* mais complexas, mas é sempre necessário importar estes novos ficheiros no *main.yml* da pasta. Em outros casos, como por exemplo na pasta *files* e *templates*, existem habitualmente ficheiros de configuração. Começando pela pasta *tasks*, esta contém as tarefas que a *role* irá executar, a *handlers*, contém tarefas que são invocadas *ad-hoc* após a execução de tarefas do fluxo normal de execução, na pasta *defaults* são declaradas variáveis padrão e na pasta *vars*, todos os tipos de variáveis.

A documentação da *role* é descrita no ficheiro *README.md*, no qual é utilizada a linguagem de marcação Markdown.

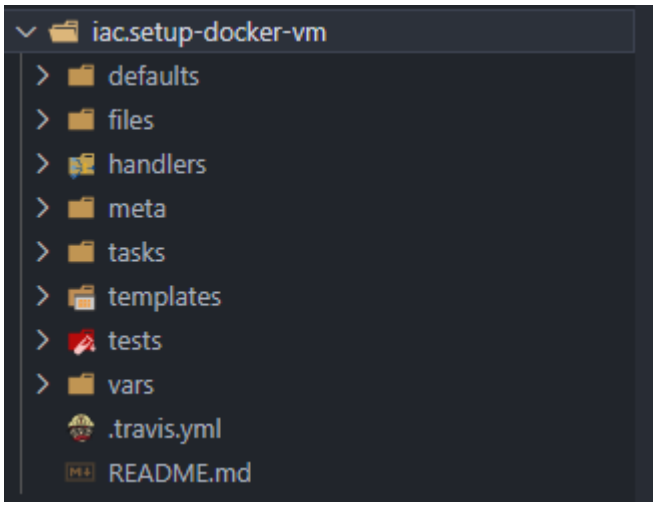


Figura 10 - Estrutura exemplo de pastas e ficheiros de uma Ansible *role*

### 5.3.4 Outros ficheiros

Outros ficheiros relevantes no contexto de execução do Ansible são por exemplo os seguintes:

O ficheiro de configuração "*ansible.cfg"*, no qual é possível alterar a configuração padrão do Ansible.

No contexto da arquitetura deste sistema, no qual algumas *roles* interagem com a REST API da plataforma AWX, é necessário enunciar algumas credenciais, i.e: *tokens* de autenticação, verificação de certificados, endereço do AWX, no ficheiro "*.tower-cli.cfg"*.

Para interagir com a REST API da plataforma Openstack, o Ansible irá detetar o ficheiro *clouds.yaml* caso ele exista. Na instalação do Devstack, este ficheiro encontra-se automaticamente criado no servidor onde foi instalado. Para além disso também é possível realizar o *download* deste diretamente da interface Web, caso o utilizador seja administrador da plataforma.

### 5.3.5 Roles desenvolvidas

As Ansible *roles* podem ser traduzidas em diagramas com a ferramenta *ansible-playbook-grapher* para facilmente visualizar o fluxo de execução. Nos seguintes tópicos serão discutidas as *roles* desenvolvidas juntamente com o diagrama da *role* correspondente.

***Implementação do servidor de containers***

A Figura 11 representa a *role* responsável para instalar o Docker Engine na máquina virtual de *containers*. Inicia por instalar dependências e atualizar todo o software do servidor para a última versão.

Em seguida instala o Docker Engine e algumas dependências de Python necessárias para o Ansible conseguir interagir com a API do Docker.

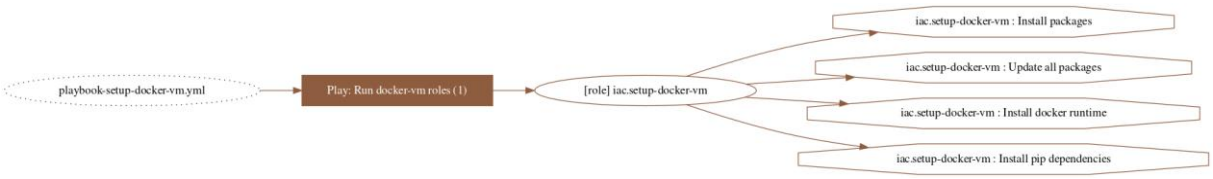


Figura 11 – Grafo do fluxo de execução da role "*iac.setup-docker-vm*"

***Implementação do serviço de DNS***

Para a instalação do serviço de DNS, como presente na Figura 12 garante a presença de ficheiros de configuração do serviço CoreDNS no servidor de *containers*. Em seguida carrega a imagem do container CoreDNS do registo oficial de imagens Docker, caso esta ainda não esteja presente. Os passos seguintes garantem que o serviço de DNS do sistema operativo não interfira com o *container*, através de alteração do ficheiro de configuração e reinício do serviço. Por fim garante que o *container* de CoreDNS está em execução com as características descritas no ficheiro *docker-compose.yml*.

É importante realçar que sempre que seja necessário atualizar a configuração de DNS, esta configuração necessita de ser realizada nos ficheiros *db.192.168.80* e *db.iac.estgv* presentes na pasta *./iac.dns-server/files/* da *role* e voltar a lançá-la.

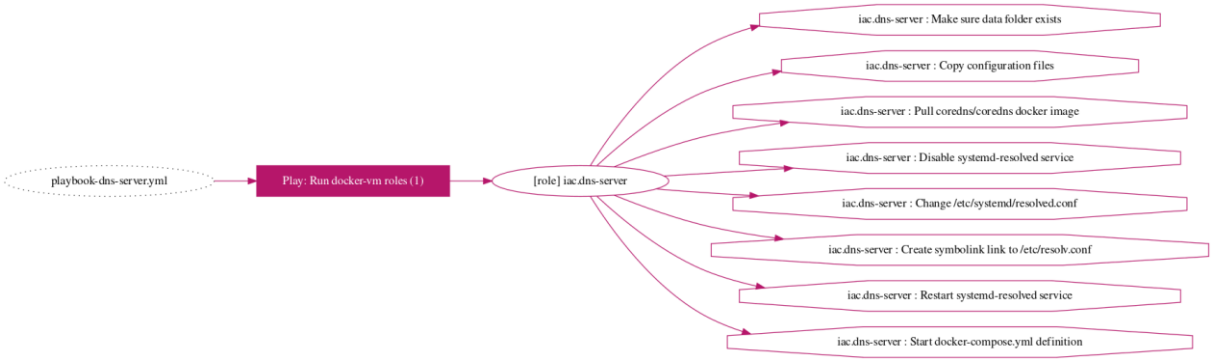


Figura 12 - Grafo do fluxo de execução da role "*iac.dns-server*"

***Implementação da plataforma Gitea***

A role apresentada na Figura 13 apenas garante que o *container* descrito com as características definidas no "*docker-compose.yml*" se encontra em execução.


Figura 13 - Grafo do fluxo de execução da role "*iac.gitea*"

É necessário salientar que as *roles* anteriores devem ser lançadas em conjunto pois as duas últimas estão dependentes da primeira, e as 3 garantem a correta implementação do servidor de *containers* que fornece serviço de DNS e aloja a plataforma Gitea.

***Implementação da plataforma AWX***

A *role* apresentada na Figura 14 inicia por garantir que algumas dependências estejam instaladas no servidor e em seguida assegura que o *cluster* de Kubernetes está em execução. A *role* apenas certifica que o servidor possui a configuração necessária para o lançamento da orquestração.

Visto que o resto da instalação da plataforma não pode ser alcançada de forma idempotente, foi criado um *script*, o qual se encontra descrito na seguinte referência [45], o lançamento final foi realizado manualmente, com o comando *k3s kubectl apply -k base*.

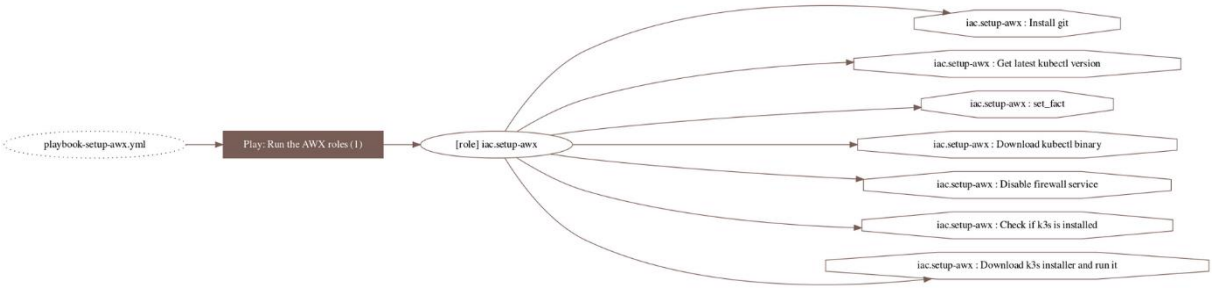


Figura 14 - Grafo do fluxo de execução da role "*iac.setup-awx*"

***Criação de recursos na plataforma AWX***

A *role* apresentada na Figura 15 tira partido da REST API da plataforma AWX para manipular os recursos disponíveis na plataforma. Desta forma os recursos são criados na plataforma de acordo com a definição das *tasks*.

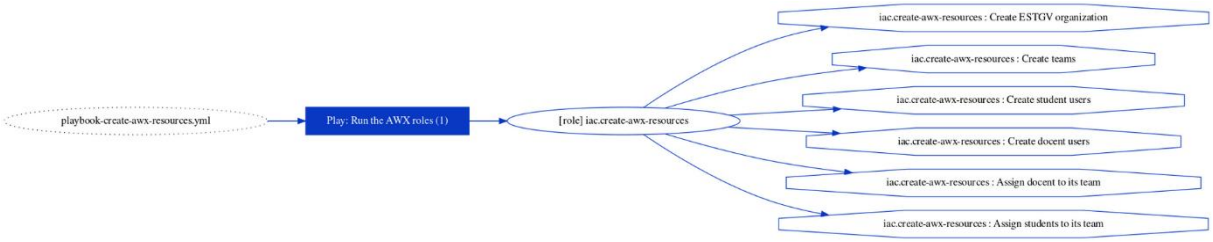


Figura 15 - Grafo do fluxo de execução da role "*iac.create-awx-resources*"

***Implementação do servidor de Openstack***

Na Figura 16 apresenta a *role* que apenas garante que o servidor possua a configuração necessária para o lançamento do *script* de instalação.

Devido ao comando que executa o script que realiza a instalação do Devstack não ser idempotente e realiza algumas alterações sobre as interfaces de rede, que por sua vez termina por instantes a conexões por SSH, o lançamento deste script foi realizado de forma manual através do comando, *./stack.sh* diretamente na consola de gestão do *hypervisor* ESXi.

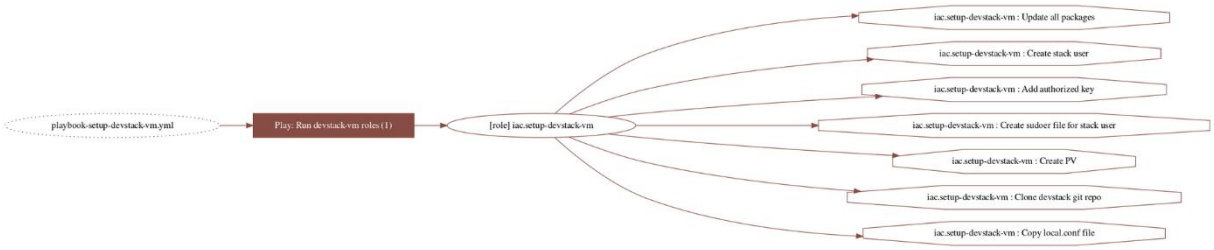


Figura 16 - Grafo do fluxo de execução da role "*iac.setup-devstack-vm*"

***Criação de recursos de Openstack***

Do mesmo modo que o Ansible permite interagir com a REST API da plataforma AWX, também permite interagir com a do Openstack. Assim é possível executar os módulos responsáveis pela criação de recursos da plataforma como também para consultar fatos relativos à infraestrutura. Neste tipo de operações, o Ansible consegue garantir um maior princípio de idem potência. A Figura 17 apresenta a role que aprovisiona estes recursos.


Figura 17 - Grafo do fluxo de execução da role "*iac.create-openstack-resources*"

# 6. Casos de uso das plataformas AWX e Openstack

Neste capítulo será discutido, utilizando diagramas de caso de uso, a forma como a plataforma AWX poderá ser utilizado por diferentes tipos de atores. Adicionalmente é também apresentada a implementação de um caso de uso com o recurso ao Ansible.

## 6.1 Descrição dos atores

Neste capítulo são apresentados os atores que interagem com a plataforma AWX. Os principais atores são os administradores de sistemas, docentes e alunos.

Os casos de uso que se enquadram no âmbito geral e não se especializam em apenas um tipo de ator podem se designar como casos de uso genéricos e destaca-se o seguinte:

Atualização da chave pública da plataforma Gitea;

### 6.1.1 Administradores de sistemas

Os casos de uso destacados para os administradores de sistema assentam sobretudo em operações de manutenção do sistema, configuração e implementação de serviços, neste caso o Moodle. Os casos de uso que se destacam são:

Implementação da plataforma Moodle;

Agendamento ou execução *ad-hoc*, tarefas de manutenção, e.g: destruir infraestrutura não usada;

### 6.1.2 Docentes

Os casos de uso projetados para serem usados pelos docentes, são contextualizados para dar suporte às aulas práticas laboratoriais. Destacam-se:

Implementação de um laboratório genérico de redes para as aulas, o objetivo deste caso de uso passa por aprovisionar uma rede com um determinado número de instâncias virtuais alocadas dentro dessa;

Possibilidade de reverter a criação do laboratório anterior;

### 6.1.3 Alunos

Os alunos dispõem dos seguintes casos de uso que os permite ajudar na implementação de projetos escolares, como também em contexto das aulas de laboratório, tal como podemos observar nos seguintes casos de uso:

Criação de máquinas virtuais *on demand*, com opção de diferentes imagens (e.g: imagens Ubuntu, Fedora, Rocky Linux, Arch linux, etc) e especificando em que rede alocar a instância a ser criada, desta forma, é possível realizar interoperabilidade com o caso de uso de implementação de um laboratório genérico, para quando um aluno necessitar mais do que uma máquina virtual;

Implementação de projetos (e.g: no *stack* de tecnologias NodeJS) dos alunos a partir de sistemas de controlo de versão (eg: Gitea, Github);

## 6.2 Casos de uso

Neste capítulo, os casos de uso serão descritos, acompanhados de um diagrama de forma a contribuir para uma melhor interpretação.

### 6.2.1 Atualização da chave pública

De forma a fornecer uma alternativa de atualização da chave pública do utilizador, a partir de dentro da plataforma AWX, sem necessidade de aceder à plataforma Gitea, foi criado o caso de uso apresentado na Figura 18.

O aluno ao executar o trabalho "Atualizar chave pública" na plataforma AWX, este necessita de fornecer a sua chave pública no formulário, em seguida o Ansible interage com a REST API do Gitea e realiza a operação de atualização da chave pública para a identidade do aluno.

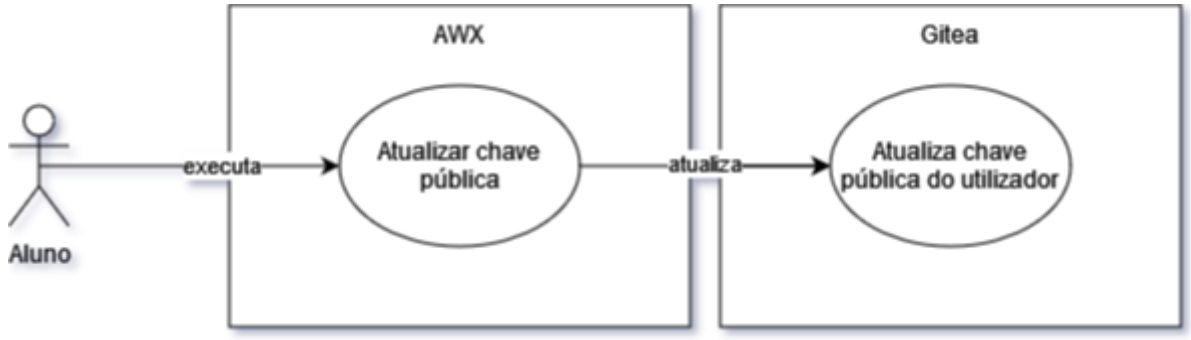


Figura 18 – Caso de uso "Atualização da chave pública"

### 6.2.2 Agendamento de tarefas de manutenção

Devido à limitação de recursos físicos, é recomendado existir um fluxo de manutenção da infraestrutura virtual do sistema Openstack. Quando existe possibilidade de criação *on-demand* de recursos, por vezes a sua remoção pode ser esquecida. A criação de recursos no Openstack, permite a associação de meta dados, as *tags*, que permite identificar se determinado recurso é prioritário ou não.

Neste contexto surge o caso de uso, apresentado na Figura 19, em que se permite aos administradores de sistemas executar tarefas de manutenção, tanto *ad-hoc* como realizar o agendamento da execução. Estas tarefas de manutenção têm por base consultar o inventário do Openstack sobre infraestrutura que não seja essencial, i.e recursos com *tags* não essenciais ou até mesmo recursos sem nenhuma *tag* para abranger todos os recursos. O agendamento das tarefas de manutenção (e.g: limpeza de infraestrutura) pode ser por exemplo, executar este trabalho todas as sextas-feiras durante a noite, para não interferir com as aulas.

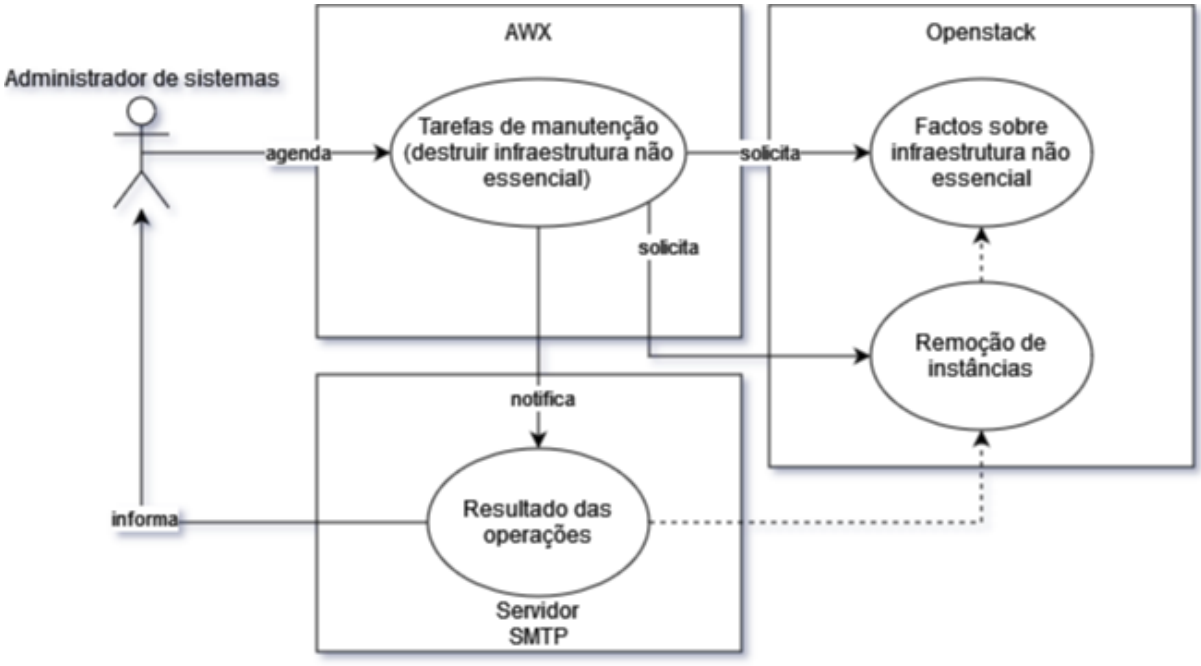


Figura 19 – Caso de uso "Agendamento de tarefas de manutenção"

### 6.2.3 Criação de um laboratório genérico

No caso de uso apresentado na Figura 20, o docente tem o objetivo de aprovisionar um laboratório para os seus N alunos, portanto executa o trabalho "Implementar um laboratório genérico para N alunos" na plataforma AWX, na qual preenche um formulário relativo à configuração da rede a ser aprovisionada e também uma lista do número mecanográfico dos alunos. Esta lista de números mecanográficos é importante pois permite que sejam enviados, por e-mail, os endereços IP para cada aluno no final de execução do trabalho. Por outro lado, também permite obter as chaves públicas dos alunos, para que quando estes acedam às instâncias pelo protocolo SSH, sejam automaticamente autenticados.

Após ser dada a ordem de início do trabalho, o Ansible irá aprovisionar a rede e as instâncias respetivas, com etiquetas relativas ao laboratório.

Após as tarefas finalizarem, o docente pode consultar o resultado do trabalho diretamente na plataforma AWX e aos alunos, é enviado individualmente um email com o *floating* IP da sua instância.

No final da aula, é esperado que o docente execute o trabalho que permite remover o laboratório criado antes.

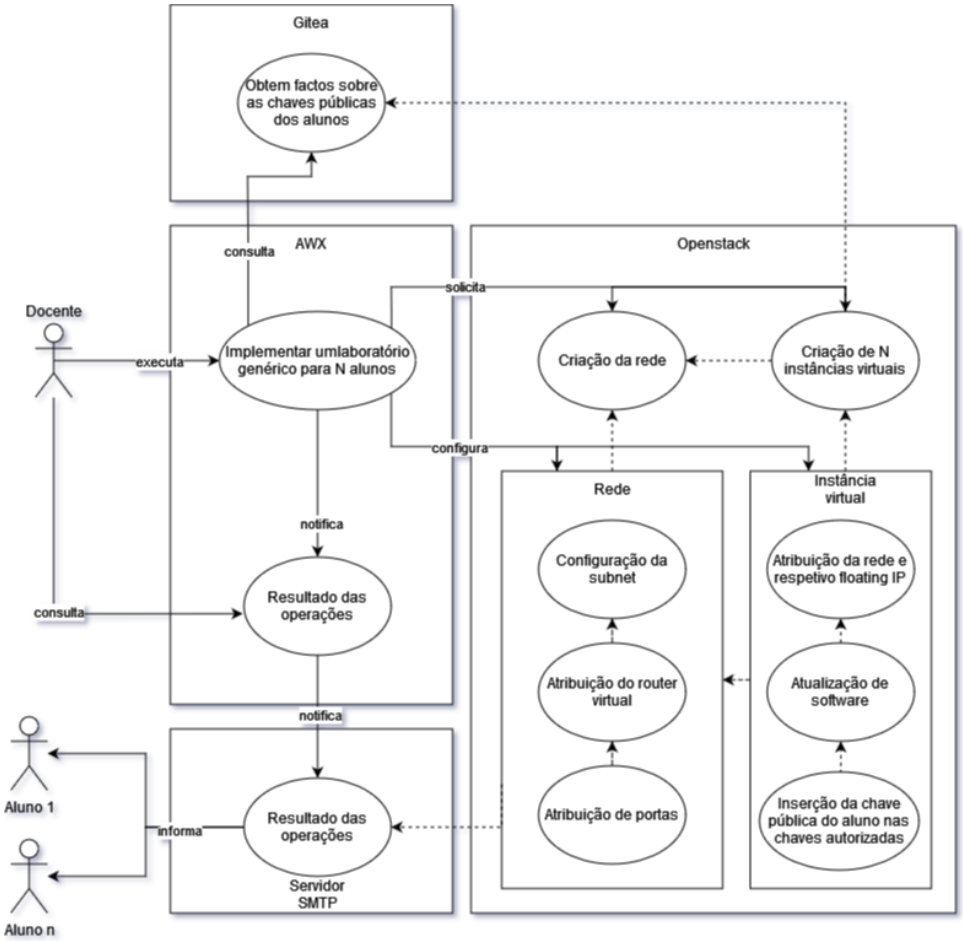


Figura 20 – Caso de uso “Criação de um laboratório genérico”

### 6.2.4 Remoção do laboratório genérico

Como referido anteriormente, é esperado que o ator docente execute o trabalho apresentado no caso de uso na Figura 21, fornecendo o nome do laboratório previamente criado. O Ansible irá consultar no Openstack os recursos de rede, de computação e de armazenamento com a etiqueta fornecida. Em seguida irá proceder à sua remoção pela ordem definida no diagrama.

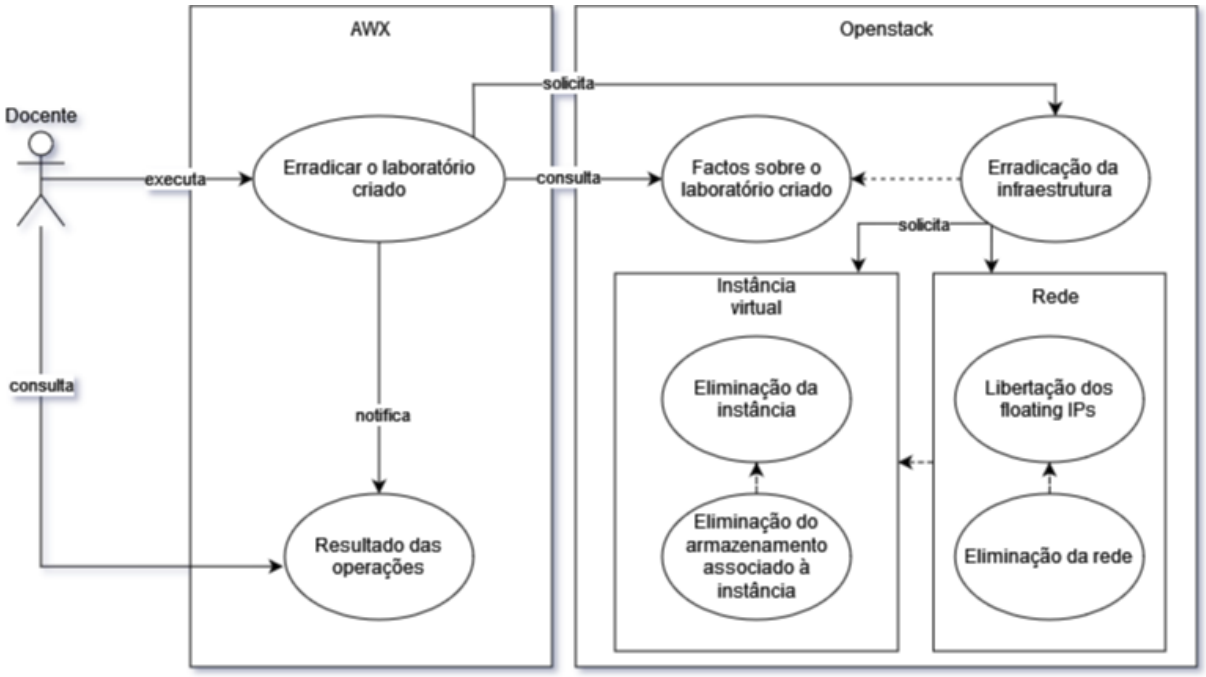


Figura 21 - Caso de uso “Remoção do laboratório genérico”

### 6.2.5 Criação de instâncias *on-demand*

Ocasionalmente os alunos necessitam de mais do que uma instância nas aulas práticas, por conseguinte o caso de uso apresentado na Figura 22 visa a complementar o laboratório genérico com mais uma instância por cada trabalho executado. O ator necessita de fornecer o nome do laboratório e selecionar entre uma lista de imagens do sistema operativo, a que deseja.

Similarmente ao caso de uso "Criação de um laboratório genérico", na instância do aluno irá dispor da sua chave pública e será enviado por e-mail o *floating* IP da instância.

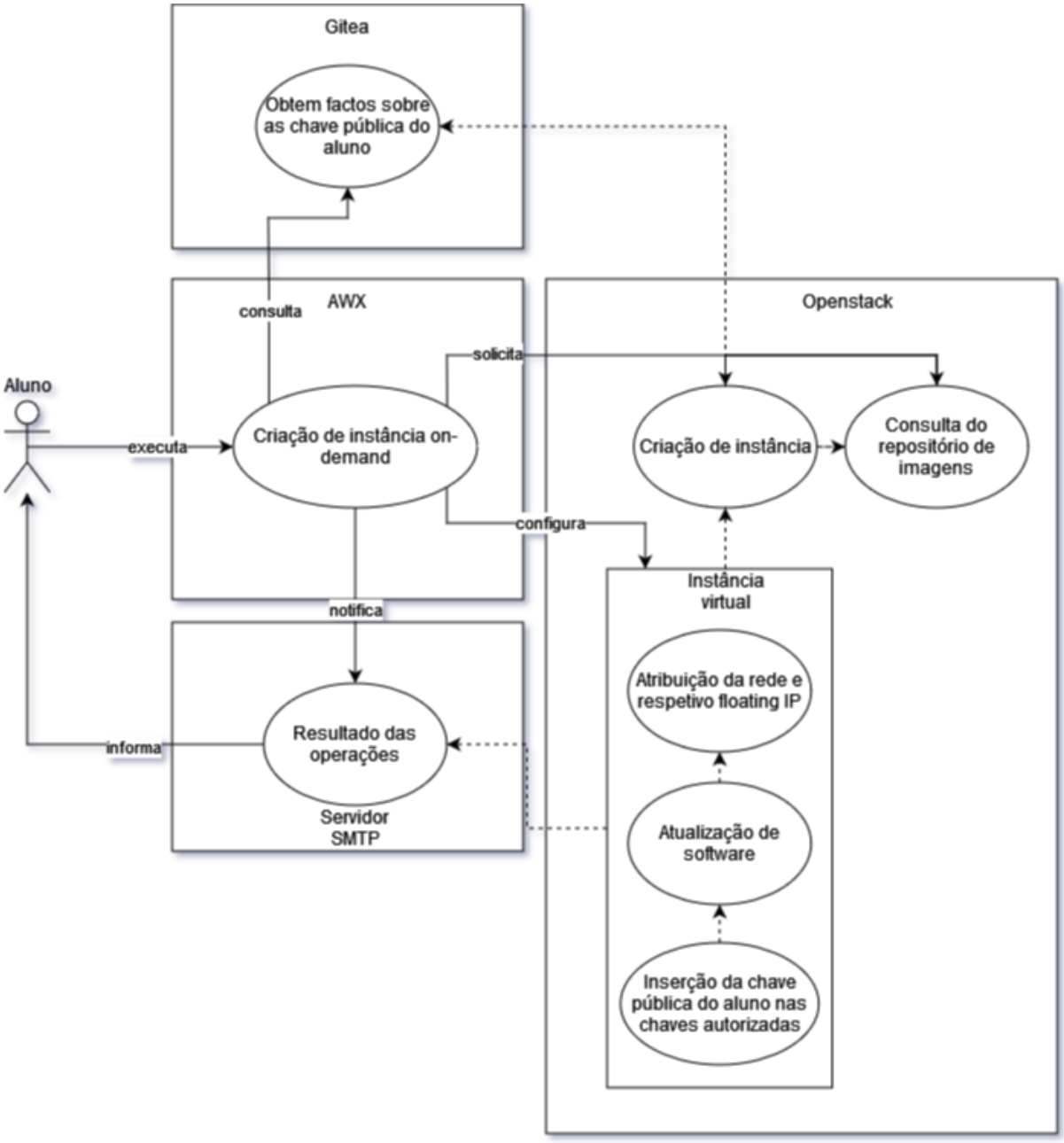


Figura 22 – Caso de uso “Criação de instâncias *on-demand*”

### 6.2.6 Implementação de projetos

O caso de uso apresentado na Figura 23 propõe-se a implementar um projeto guardado pelo ator em sistema de controlo de versão. Visto que existem vários tipos de tecnologias, este projeto terá de ser compatível com os métodos de implementação disponíveis na plataforma.

Similarmente ao caso de uso "Criação de VMs *on-demand*", este irá implementar da mesma forma, acrescendo a instalação de software necessário para executar o projeto e também irá clonar o código fonte a partir do sistema de controlo de versão onde esteja armazenado.

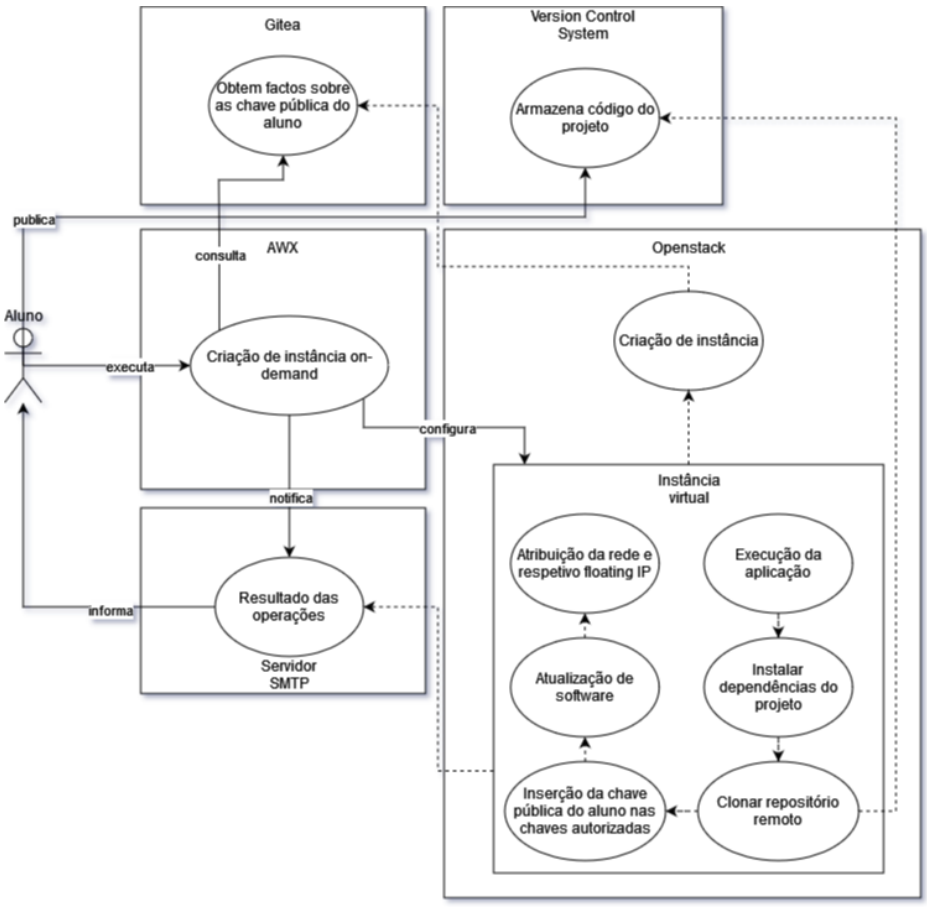


Figura 23 – Caso de uso “Implementação de projetos”

### 6.2.7 Implementação da plataforma Moodle

A implementação da plataforma Moodle representada na Figura 24 destina-se aos administradores de sistemas. Este caso de uso pretende automatizar o processo de implementação e configuração do Moodle da escola, em que o administrador apenas necessita de executar o trabalho no AWX. No final da execução, o administrador pode consultar informações relevantes no *output* da plataforma.

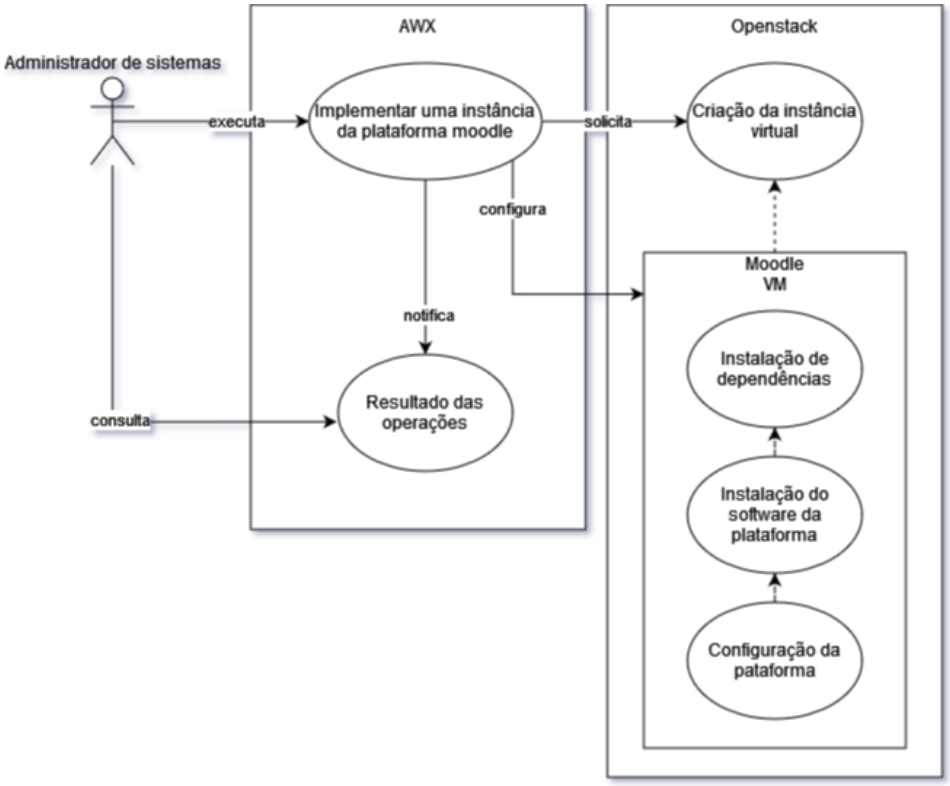


Figura 24 – Caso de uso “Implementação da plataforma Moodle”

As tarefas que são executadas na instância, e que permite implementar a plataforma Moodle, podem ser consultadas no diagrama representado na Figura 25.

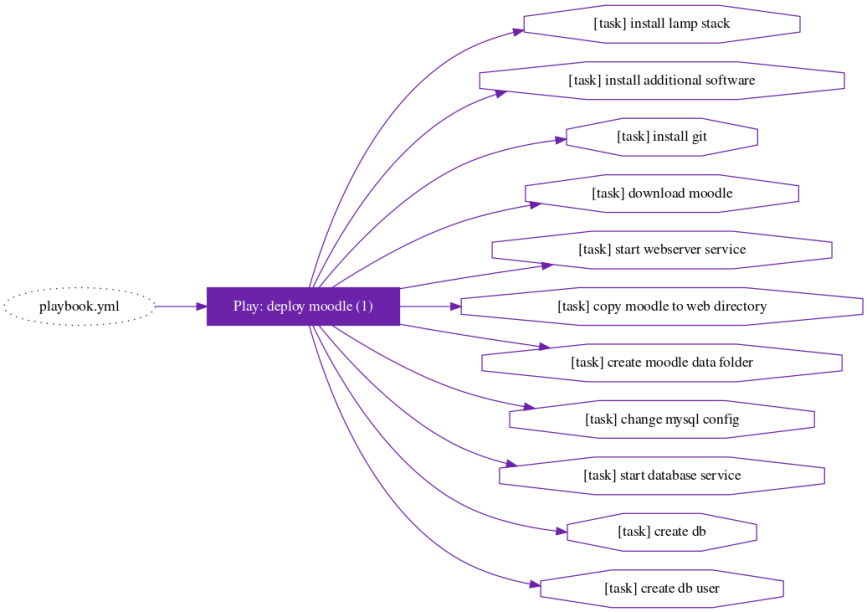


Figura 25 - Grafo do fluxo de execução das tarefas de implementação do Moodle

## 6.3 Considerações

É necessário salientar que em qualquer situação o administrador do sistema pode sempre utilizar a consola Web do Openstack para realizar tarefas específicas pelo que não fica limitado a executar apenas os trabalhos disponíveis no AWX. Além do mais, o administrador também pode desenvolver novos fluxos de automatização, guardando-os na plataforma Gitea e criando os recursos na plataforma AWX. O caso de uso seguinte demonstra como o ator poderá expandir o fluxo de automatizações na plataforma.

### 6.3.1 Desenvolvimento de novas automatizações

Para o desenvolvimento de novas automatizações, como apresentado na Figura 26, os administradores de sistemas necessitam de um ambiente de desenvolvimento semelhante ao apresentado no capítulo 5, Desenvolvimento de Ansible *roles*. Após o desenvolvimento da automatização, este terá de a publicar em um sistema de controlo de versão (e.g: Gitea, Github). Em seguida, na plataforma AWX, o ator terá de criar o projeto que aponta para o endereço do projeto no VCS.

Para terminar, esta nova automatização irá estar disponível na plataforma AWX, disponível para ser executada.

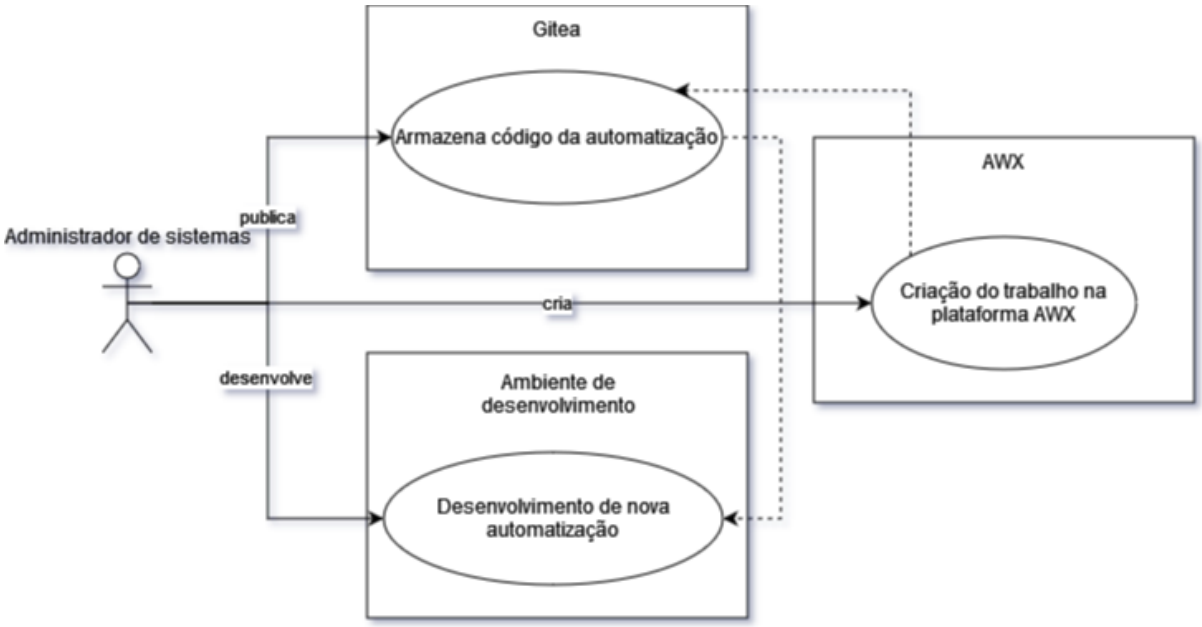


Figura 26 - Caso de uso “Desenvolvimento de novas automatizações”

# 7. Conclusões

Neste capítulo serão discutidas as principais conclusões relativas ao trabalho realizado, serão apresentadas algumas considerações finais e também é delineado um possível trabalho futuro.

A criação de *workflows* de automatização permite aos administradores de sistemas ter a capacidade de automatizar tarefas repetitivas, dando lugar à disponibilidade para se focarem em tarefas mais relevantes para a sua organização.

O uso de *Cloud* Privada no contexto do ensino superior em conjunto com ferramentas de automatização, permite gerir a infraestrutura de forma mais simples e eficiente, e com a garantia que a implementação de serviços seja realizada de forma mais estandardizada possível.

Para os alunos permite criar e aceder a recursos de infraestrutura que de outra forma não poderiam ter acesso, permitindo também aceder a infraestrutura de forma remota. Igualmente, aproxima os alunos, ao paradigma de infraestrutura *Cloud* e cria neles competências de gestão nesta área com a possibilidade de verificação dos seus benefícios.

Tanto para os docentes como para os alunos, permite dinamizar o teor das aulas práticas, porque para além da criação de infraestrutura *on demand*, que reduz o *overhead* temporal da sua criação e configuração, permite explorar outros contextos práticos.

No entanto o uso de ferramentas de configuração não é solução para todos os problemas e durante a implementação desta infraestrutura verificou-se, por vezes, que existem etapas de configuração na qual torna-se muito imprevisível conseguir conceptualizar um fluxo de automatização coerente e resiliente, especialmente na ferramenta utilizada (Ansible), na qual quando o contexto não permite seguir o princípio de idem potência, o benefício do seu uso revela-se reduzido.

## 7.1 Considerações finais

Nesta secção são apresentadas algumas considerações sobre os obstáculos enfrentados durante o desenvolvimento da investigação.

O obstáculo que mais prevaleceu durante a investigação revelou-se ser a falta de tempo, principalmente por ter sido desenvolvido durante uma altura em que o acesso ao estabelecimento de ensino foi bastante restrito. No entanto após a implementação do serviço de VPN, possibilitou trabalhar de forma remota. Contudo, por vezes o servidor que hospedava toda a infraestrutura, desligava-se, por consequência de quebras de luz e não voltava a reiniciar sozinho, mesmo estando configurado para tal. Este último obstáculo fazia muitas vezes parar o trabalho durante semanas, enquanto o problema não fosse resolvido presencialmente.

Um dos temas que mais poderia ter sido explorado, foi relativamente à funcionalidade de *wiki* dos repositórios da plataforma Gitea, que permite criar uma base de conhecimento expandida para além dos típicos ficheiros de Markdown *README.md* existentes nas Ansible *roles*.

O trabalho desenvolvido ao longo deste projeto também permitiu contribui para o estudo realizado na publicação "*An Overview on Cloud Services for Human Tracking*" [46].

## 7.2 Trabalho futuro

É sugerido como trabalho futuro aplicar a arquitetura descrita neste trabalho na Escola Superior de Tecnologia e Gestão de Viseu, transformar o conjunto dos casos de uso desenvolvidos em artefactos de automatização, e em conjunto com serviços informáticos, docentes e alunos do Departamento de Engenharia Informática, realizar um *benchmark* de forma a produzir resultados para posteriormente serem analisados de forma holística. A arquitetura conceptualizada e os casos de uso desenvolvidos, possuem ainda arestas por limar, no entanto, a análise de resultados iria permitir a melhoria do sistema para que no futuro realmente pudessem ser implementados para suportar tanto as aulas dos alunos do departamento de Engenharia Informática, bem como os administradores de sistemas dos serviços informáticos da ESTGV.

Relativamente a sugestões de pontos óbvios de melhoria imediata, podem destacar-se os seguintes:

Implementação do *hypervisor* Proxmox em vez do ESXi, devido às limitações da versão gratuita do VMWare ESXi e ao facto de ser *open-source*;

Implementação do *Virtual Private Network* com o software Wireguard, devido às limitações causadas pelo licenciamento do OpenVPN.

# REFERÊNCIAS


[1] S. Bonner, C. Pulley, I. Kureshi, V. Holmes, J. Brennan, e Y. James, «Using OpenStack to improve student experience in an H.E. environment», *2013 Sci. Inf. Conf.*, pp. 888–893, 2013, Acedido: Dez. 06, 2020. [Em linha]. Disponível em: https://ieeexplore.ieee.org/document/6661847.

[2] P. Kalagiakos e P. Karampelas, «Cloud Computing learning», em *2011 5th International Conference on Application of Information and Communication Technologies (AICT)*, Out. 2011, pp. 1–4, doi: 10.1109/ICAICT.2011.6110925.

[3] Nishant Kumar Singh, S. Thakur, H. Chaurasiya, e H. Nagdev, «Automated provisioning of application in IAAS cloud using Ansible configuration management», em *2015 1st International Conference on Next Generation Computing Technologies (NGCT)*, Set. 2015, pp. 81–85, doi: 10.1109/NGCT.2015.7375087.

[4] S. Naziris, «Infrastructure as code : towards dynamic and programmable IT systems», 2019.

[5] L. Abeni e D. Faggioli, «Using Xen and KVM as real-time hypervisors», *J. Syst. Archit.*, vol. 106, p. 101709, Jun. 2020, doi: 10.1016/J.SYSARC.2020.101709.

[6] A. D. Berzano *et al.*, «A review paper on hypervisor and virtual machine security», *J. Phys. Conf. Ser.*, vol. 1950, n. 1, p. 012027, Ago. 2021, doi: 10.1088/1742-6596/1950/1/012027.

[7] B. Dordevic, R. Furtula, e V. Timcenko, «VMware ESXi and Microsoft Hyper-V Hypervisor Performance Comparison», *2020 28th Telecommun. Forum, TELFOR 2020 - Proc.*, Nov. 2020, doi: 10.1109/TELFOR51502.2020.9306625.

[8] S. A. Algarni, M. R. Ikbal, R. Alroobaea, A. S. Ghiduk, e F. Nadeem, «Performance Evaluation of Xen, KVM, and Proxmox Hypervisors», *Int. J. Open Source Softw. Process.*, vol. 9, n. 2, pp. 39–54, Abr. 2018, doi: 10.4018/IJOSSP.2018040103.

[9] D. T. Vojnak, B. S. Eordevic, V. V. Timcenko, e S. M. Strbac, «Performance Comparison of the type-2 hypervisor VirtualBox and VMWare Workstation», *27th Telecommun. Forum, TELFOR 2019*, Nov. 2019, doi: 10.1109/TELFOR48224.2019.8971213.

[10] A. Rot e P. Chrobak, «Benefits, Limitations and Costs of IT Infrastructure Virtualization in the Academic Environment. Case Study using VDI Technology», pp. 738–745, Set. 2018, doi: 10.5220/0006934707380745.

[11] A. M. Potdar, D. G. Narayan, S. Kengond, e 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.

[12] A. R. Putri, R. Munadi, e R. M. Negara, «Performance analysis of multi services on container Docker, LXC, and LXD», *Bull. Electr. Eng. Informatics*, vol. 9, n. 5, pp. 2008–2011, Out. 2020, doi: 10.11591/EEI.V9I5.1953.

[13] E. Casalicchio, «Container Orchestration: A Survey», *EAI/Springer Innov. Commun. Comput.*, pp. 221–235, 2019, doi: 10.1007/978-3-319-92378-9_14.

[14] T. Alam, «Cloud Computing and Its Role in the Information Technology», *SSRN Electron. J.*, Abr. 2020, doi: 10.2139/SSRN.3639063.

[15] Y. Jadeja e K. Modi, «Cloud computing - concepts, architecture and challenges», em *2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*, Mar. 2012, pp. 877–880, doi: 10.1109/ICCEET.2012.6203873.

[16] G. Aryotejo, D. Y. Kristiyanto, e Mufadhol, «Hybrid cloud: Bridging of private and

public cloud computing», em *Journal of Physics: Conference Series*, Mai. 2018, vol. 1025, n. 1, p. 12091, doi: 10.1088/1742-6596/1025/1/012091.

[17] M. M. Alabbadi, «Cloud computing for education and learning: Education and Learning as a Service (ELaaS)», em *2011 14th International Conference on Interactive Collaborative Learning, ICL 2011 - 11th International Conference Virtual University, VU'11*, 2011, pp. 589–594, doi: 10.1109/ICL.2011.6059655.

[18] M. I. Tariq, S. Tayyaba, H. Rasheed, e M. W. Ashraf, «Factors influencing the Cloud Computing adoption in Higher Education Institutions of Punjab, Pakistan», em *Proceedings of 2017 International Conference on Communication, Computing and Digital Systems, C-CODE 2017*, Mai. 2017, pp. 179–184, doi: 10.1109/C-CODE.2017.7918925.

[19] Bridget McCrea, «IT on Demand: The Pros and Cons of Cloud Computing in Higher Education -- Campus Technology». https://campustechnology.com/articles/2009/08/20/it-on-demand-the-pros-and-cons-of-cloud-computing-in-higher-education.aspx (acedido Jan. 23, 2021).

[20] M. Mircea e A. Andreescu, «Using Cloud Computing in Higher Education: A Strategy to Improve Agility in the Current Financial Crisis», *Commun. IBIMA*, pp. 1–15, Jun. 2011, doi: 10.5171/2011.875547.

[21] G. Finta, S. P. Jukka, e K. Nurminen Advisor, «Mitigating the effects of vendor lock-in in edge cloud environments with open-source technologies», Out. 2019, Acedido: Fev. 20, 2022. [Em linha]. Disponível em: https://aaltodoc.aalto.fi:443/handle/123456789/40884.

[22] B. Mohammed, R. Kumar, P. Bedi, B. Deep, P. Kumar, e P. Sarna, «Analysis of Cloud Test Beds using OpenSource Solut ions Comparative Study of OpenNebula, CloudStack, Eucalyptus and OpenStack», *2015 3rd Int. Conf. Futur. Internet Things Cloud*, pp. 105–203, 2015, Acedido: Fev. 20, 2022. [Em linha]. Disponível em: http://www.publishingindia.com/ijdcc.

[23] T. Rosado e J. Bernardino, «An overview of openstack architecture», em *Proceedings of the 18th International Database Engineering & Applications Symposium on - IDEAS '14*, 2014, pp. 366–367, doi: 10.1145/2628194.2628195.

[24] Red Hat Inc., «Full Stack Automation with Ansible and OpenStack». https://www.redhat.com/en/blog/full-stack-automation-ansible-and-openstack (acedido Jan. 23, 2021).

[25] Red Hat Inc., «Ansible Documentation — Ansible Documentation». https://docs.ansible.com/ansible/latest/index.html (acedido Jan. 19, 2021).

[26] Red Hat Inc., «Ansible Tower | Ansible.com». https://www.ansible.com/products/tower (acedido Dez. 15, 2020).

[27] N. N. Zolkifli, A. Ngah, e A. Deraman, «Version Control System: A Review», *Procedia Comput. Sci.*, vol. 135, pp. 408–415, Jan. 2018, doi: 10.1016/J.PROCS.2018.08.191.

[28] F. F. Blauw, «The Use of Git as Version Control in the South African Software Engineering Classroom», *2018 IST-Africa Week Conf.*, 2018, Acedido: Jan. 11, 2021. [Em linha]. Disponível em: https://ieeexplore.ieee.org/document/8417196.

[29] T. A. Limoncelli, «GitOps», *Commun. ACM*, vol. 61, n. 9, pp. 38–42, Ago. 2018, doi: 10.1145/3233241.

[30] The Gitea Authors, «Documentation - Docs». https://docs.gitea.io/en-us/ (acedido Jan. 19, 2021).

[31] GitLab B.V., «GitLab Documentation». https://docs.gitlab.com/ (acedido Fev. 20, 2022).

[32] Atlassian, «Features | Bitbucket». https://bitbucket.org/product/features (acedido Fev. 20, 2022).
[33] GitHub Inc, «GitHub Documentation». https://docs.github.com/en (acedido Fev. 20, 2022).
[34] Kent Beck *et al.*, «Principles behind the Agile Manifesto», *2001*. http://agilemanifesto.org/principles.html (acedido Jan. 23, 2021).
[35] C. Ebert, G. Gallardo, J. Hernantes, e N. Serrano, «DevOps», *IEEE Softw.*, vol. 33, n. 3, pp. 94–100, Mai. 2016, doi: 10.1109/MS.2016.68.
[36] Jeff Geerling, *Ansible for DevOps*. 2020.
[37] S. A. I. B. S. Arachchi e I. Perera, «Continuous integration and continuous delivery pipeline automation for agile software project management», em *MERCon 2018 - 4th International Multidisciplinary Moratuwa Engineering Research Conference*, Jul. 2018, pp. 156–161, doi: 10.1109/MERCon.2018.8421965.
[38] S. Mackey, I. Mihov, A. Nosenko, F. Vega, e Y. Cheng, «A Performance Comparison of WireGuard and OpenVPN», *CODASPY 2020 - Proc. 10th ACM Conf. Data Appl. Secur. Priv.*, vol. 3, n. 20, pp. 162–164, Mar. 2020, doi: 10.1145/3374664.3379532.
[39] The CoreDNS Authors e The Linux Foundation, «CoreDNS: DNS and Service Discovery». https://coredns.io/manual/toc/ (acedido Fev. 20, 2022).
[40] A. Valkeinen, «Automated DNS installation and configuration».
[41] The PostgreSQL Global Development Group, «PostgreSQL: Documentation». https://www.postgresql.org/docs/ (acedido Fev. 20, 2022).
[42] OpenInfra Foundation, «Open Source Cloud Computing Platform Software - OpenStack». https://www.openstack.org/software/project-navigator/deployment-tools (acedido Fev. 20, 2022).
[43] OpenStack DevStack Team, «Plugins — DevStack documentation». https://docs.openstack.org/devstack/latest/plugins.html (acedido Fev. 20, 2022).
[44] Manuel Augusto Tarouca Martins, «GitHub - manueltarouca/iac.estgv: Repository which hosts the Infrastructure as Code artifacts»,. https://github.com/manueltarouca/iac.estgv (acedido Fev. 20, 2022).
[45] Dennis McCarthy, «AWX-operator_setup_on_K3s+https · GitHub». https://gist.github.com/dmccuk/098fccc488c8a5aeaa5f859855660018 (acedido Fev. 20, 2022).
[46] M. Martins, D. Mota, P. Martins, M. Abbasi, e F. Caldeira, «An Overview on Cloud Services for Human Tracking», pp. 3–12, Set. 2021, doi: 10.1007/978-3-030-87687-6_1.