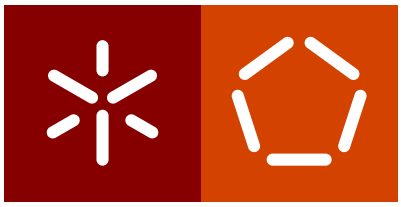


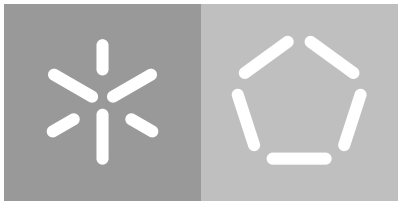
Universidade do Minho
Escola de Engenharia
Departamento de Informática

Nuno Silvino Santos Dionísio

Agilizar o Deployment de Aplicações Modernas

Soluções Cloud para a Indústria Automóvel

Abril 2017



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Nuno Silvino Santos Dionísio

Agilizar o Deployment de Aplicações Modernas

Soluções Cloud para a Indústria Automóvel

Mestrado em Engenharia Informática

Supervisão da dissertação por
António Luís Pinto Ferreira de Sousa
André Leite Ferreira

Abril 2017

AGRADECIMENTOS

Este documento representa o culminar de um longo trabalho que teve impacto quer na minha vida, quer na vida das pessoas que me são mais próximas, não posso deixar de dar um agradecimento especial a toda a minha família pelo apoio incondicional.

De forma direta ou indireta, muitas pessoas contribuíram para a minha formação académica, mas é normal que se destaquem os meus colegas e os meus professores. Todo o conhecimento que adquiri tem sido fundamental no meu percurso profissional, e foi igualmente importante para desempenhar as tarefas para a realização deste trabalho. Desta forma, um muito obrigado a todos os meus colegas e professores que se disponibilizaram para me apoiar, aconselhar e ensinar ao longo do meu percurso académico.

Entre os professores destaco o meu orientador António Luís Sousa, que me acompanha desde o meu primeiro ano de mestrado, e o meu supervisor André Leite Ferreira pela confiança e compreensão.

Trabalhar em equipa não é fácil, por isso não posso de deixar de reconhecer e agradecer aos meus colegas do projeto P10, pela sua disponibilidade e principalmente pela sua paciência.

Por fim, o meu obrigado ao departamento de engenharia da BOSCH Car Multimedia Portugal S.A., pela oportunidade de me envolver num projeto estimulante e me proporcionar todas as condições para a conclusão deste trabalho.

ABSTRACT

To achieve an agile development, teams must be provided with tools to facilitate the implementation and automation of the processes throughout the software development life cycle. However the technologies aren't itself the unique element that make teams and organizations agile.

Agile methodologies contribute to ensuring the success of software projects, as they allow, at an early stage, the participation of stakeholders in the process, allowing quick detection of problems in the requirements and at the product to be developed.

To enhance all investment placed in the technologies, it is crucial to change the way that teams operate, and adopt practices and processes that allow to maximize all available resources.

This thesis elaborates an approach which creates a harmony between the development and operational processes, taking advantage of state-of-the-art technologies and tools.

RESUMO

Para se alcançar um desenvolvimento ágil, as equipas de desenvolvimento devem estar munidas de ferramentas que facilitem a execução e a automatização dos processos ao longo do ciclo de vida de um produto de software. Porém não são as tecnologias por si só que tornam as equipas e as organizações ágeis.

As metodologias ágeis são essenciais para garantir o sucesso dos projetos de software, pois permite, numa fase embrionária, a participação dos *stakeholders* no processo, viabilizando a rápida deteção de problemas nos requisitos e no produto que se pretende desenvolver.

Para potenciar todo o investimento colocado nas tecnologias é necessário alterar o modo como as organizações operam, e adotar práticas e processos que permitam maximizar todos os recursos existentes.

Este documento propõe uma abordagem que consiste em criar uma harmonia entre o processo de desenvolvimento e operacional, recorrendo a tecnologias de última geração.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CONFIGURAÇÕES

LISTA DE TERMOS

A

API *Application Programming Interface*. 31, 32, 34, 50

AWS *Amazon Web Services*. 2, 20

B

BIC *Bosch IoT Cloud*. 26, 29, 30, 49

BOOTP *Bootstrap Protocol*. 50

BOSH *Bosh Outer Shell*. 24, 30, 49–53

C

CLI *Command Line Interface*. 31, 49, 50, 55

CPI *Cloud Provider Interface*. 52

D

DC *Data Center*. 31, 36

DEA *Droplet Execution Agent*. 30

DFL *Data Format Language*. 14

DHCP *Dynamic Host Configuration Protocol*. 37–39, 50

DNS *Domain Name Server*. 38, 39, 52, 54

DSL *Domain-Specific Language*. 14

I

I&D Inovação e Desenvolvimento. 2, 22

IAAS *Infrastructure as a Service*. 2, 19, 20, 26, 31, 49, 52

IAC *Infrastructure as Code*. 14

IOT *Internet of Things*. 29

ISP *Internet Service Proviver*. 26

L

LVM *Logical Volume Manager*. 42

N

NIST *National Institute of Standards and Technology*. 18

P

PAAS *Platform as a Service*. 20, 22, 26, 30, 31, 57

PXE *Preboot Execution Environment*. 38

S

SAAS *Software as a Service*. 20

SO Sistema Operativo. 12, 38, 40–42, 44, 46, 47, 56, 57

SSH *Secure Shell*. 43, 44, 50

SVN *Subversion*. 27–29

T

TFTP *Trivial File Transfer Protocol*. 38

U

UM Universidade do Minho. 26, 36, 37, 39, 54

UPS *Uninterruptible Power Supply*. 25, 36

V

VM Máquina Virtual. 17, 24, 26, 32, 34, 37, 39, 46, 50–54, 57

VXLAN *Virtual Extensible LAN*. 26

1

INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

O termo ”crise de software” [1] surgiu nos anos 70, consequência da imaturidade do processo de Engenharia de Software. Essa crise decorre dos inúmeros problemas que, volvidas décadas, permanecem até aos dias de hoje [2]. São disso exemplo, a dificuldade de medir o progresso à medida que o software é desenvolvido, criando imprecisões nas estimativas de prazos e custos; permanece a reduzida capacidade de produção, porquanto o volume de profissionais é relativamente baixo face à procura existente; os projetos de desenvolvimento de software são alicerçados em requisitos vagos do cliente, nem sempre atingindo a pretensão final do pedido.

Algumas das causas destes problemas estão relacionadas com as metodologias utilizadas, é o caso do modelo em cascata, onde o processo é sequencial e linear, ou seja, as diferentes etapas do desenvolvimento do produto dependem das suas antecessoras. Por sua vez, estas etapas tendem a ser estáticas, uma vez que à medida que se avança no projeto, o custo da mudança aumenta exponencialmente [3].

Como resposta a alguns dos problemas acima referenciados surgem as metodologias ágeis com o intuito de restaurar a credibilidade no processo de desenvolvimento de software. Os fundamentos genéricos dessas metodologias estão indicados no *The Agile Manifesto* [4], publicado em 2001.

Através dos métodos ágeis, as equipas têm capacidade de entregar código completo em ciclos de vida curtos, permitindo assim: a celeridade no *feedback*, melhoria contínua e uma rápida adaptação à mudança [5]. Porém ”*o software só proporciona valor se estiver a ser executado num ambiente de produção*” [6], por conseguinte, é necessário encontrar mecanismos que permitam agilizar o processo de disponibilização do produto de software aos utilizadores finais, em suma, código completo nem sempre é sinónimo de código em produção [7].

Paralelamente à evolução do processo de desenvolvimento, também a forma e as tecnologias de como o software é interposto em ambiente de produção, têm sofrido avanços significativos.

No ano 2000, o *deploy* das aplicações era feito em servidores físicos não virtualizados, ou seja, um maior desempenho pressupunha a obtenção de mais servidores.

Em 2001, a VMware [8] criou o primeiro produto de virtualização para servidores x86, consentindo que diferentes aplicações pudessem ser executadas isoladamente através de ambientes virtuais, no mesmo servidor físico.

Até essa altura, a disponibilização de aplicações obrigava à compra de dispositivos físicos. Desde então, mais concretamente em 2006, a Amazon mudou o paradigma, através do seu serviço de *cloud*, *Amazon Web Services* (AWS) [9], a Amazon aluga os seus recursos computacionais às organizações, evitando estas de realizar a compra de recursos computacionais, até ao momento em que realmente necessitem delas. Este serviço foi posteriormente apelidado de *Infrastructure as a Service* (IaaS).

A partir de 2009, a Heroku [10] revoluciona a forma como as aplicações são disponibilizadas, no sentido em que, elimina a necessidade de recorrer à equipa operacional para efetuar o *deploy* das aplicações. Desta forma, os programadores têm a liberdade de disponibilizar as suas aplicações num ambiente propício para o efeito, sem terem de se preocupar com o sistema operativo ou o *runtime* onde as aplicações são executadas [11]. A par das tecnologias proprietárias enunciadas, surgiram em 2010 e 2011, as plataformas *open source* OpenStack [12] e Cloud Foundry [13], respetivamente. Devido à sua essência, muitas organizações têm adotado estas tecnologias para suprir as suas necessidades mediante a implementação das suas soluções privadas baseadas na *cloud*. Exemplo disso têm sido as organizações do sector da indústria automóvel, que para responderem a necessidades particulares (a privacidade ou falta de funcionalidades específicas para determinados casos de uso) têm optado por esta solução. Verifica-se assim, uma tendência na aposta no desenvolvimento de aplicações nativas para a *cloud* por parte das organizações ligadas ao sector automóvel, com o intuito de incrementar a *driving experience* [14, 15].

1.2 MOTIVAÇÃO

O factor motivacional por detrás da elaboração da presente dissertação, pode dizer-se, com rigor, que é a minha integração num projeto de inovação e desenvolvimento Inovação e Desenvolvimento (I&D) P10 - *Cloud Applications for Smart Cars* nas instalações da BOSCH Car Multimedia, em Braga.

O projeto supra referenciado, pretende validar a capacidade sensorial dos smartphones de forma a aferir a qualidade do pavimento, bem como, do estilo de condução de um condutor de automóvel.

O método para atingir o pretendido consiste na realização de várias etapas. Numa primeira fase, um conjunto de sensores envia dados pela rede que são guardados numa base de dados através de um serviço; numa segunda fase, os dados são processados, de forma

a detetar padrões dos quais se pretende extrair informação que possa indicar o estado do pavimento e características de condução do automobilista.

Na génese da presente dissertação, encontra-se igualmente o ímpeto na redução do tempo em que os protótipos são disponibilizados aos clientes/utilizadores finais, para obter a agilização do *feedback*, de maneira a limar as vicissitudes do protótipo – desafio 1.

Fornecer uma infraestrutura tecnológica de alta disponibilidade para os serviços a desenvolver no âmbito do projeto – desafio 2.

1.3 OBJETIVOS

Os objetivos a que este trabalho se propõe são:

- Efetuar um estudo das práticas utilizadas para a redução do tempo de *deployment* das aplicações.
- Identificar as ferramentas com maior relevância;
- Definir uma arquitetura para a infraestrutura de suporte aos serviços a desenvolver no âmbito do projeto;
- Definir uma arquitetura e ferramentas para a *delivery pipeline*;
- Instalação de uma *cloud* privada;
- Efetuar o *deploy* da aplicação desenvolvida no projeto P10 - *Cloud Applications for Smart Cars* no ambiente *cloud*.

1.4 ESTRUTURA DO DOCUMENTO

Nesta secção são apresentados, de uma forma muito sucinta, os tópicos abordados ao longo dos 5 (cinco) capítulos que fazem parte deste documento.

1 - INTRODUÇÃO É feito um enquadramento do projeto em questão e expostos os objetivos para o trabalho a realizar.

2 - ESTADO DA ARTE São apresentados os conceitos fulcrais para uma melhor compreensão do contexto em que este trabalho se envolve.

3 - PROBLEMA Neste capítulo é descrito o problema e quais os desafios que surgem no decorrer da análise efetuada.

4 - CONCEÇÃO E IMPLEMENTAÇÃO Neste capítulo são expostas as decisões tecnológicas e técnicas com as devidas justificações. É apresentada a arquitetura que dá suporte à infraestrutura que é implementada, assim como a aplicação prática deste trabalho.

5 - CONCLUSÃO E TRABALHO FUTURO É apresentado um sumário da execução do trabalho realizado. Ademais, são elencadas as dificuldades que foram encontradas no decorrer do projeto e identificados aspetos a melhorar num trabalho futuro.

2

ESTADO DA ARTE

2.1 APLICAÇÕES MODERNAS

Entende-se por aplicações modernas [16], aplicações que utilizam tecnologias *state of the art* tal como: *containers* de micro serviços e serviços de *big data*. Em paralelo à utilização destas tecnologias, é ainda acrescida a capacidade de mover essas aplicações de ambientes de desenvolvimento para produção com o máximo de segurança e eficiência possível.

Os alicerces da abordagem em que este trabalho é baseado, estão expostos na pirâmide de desenvolvimento de aplicações empresariais modernas (Figura 1) definida por Markus Eisele [17].

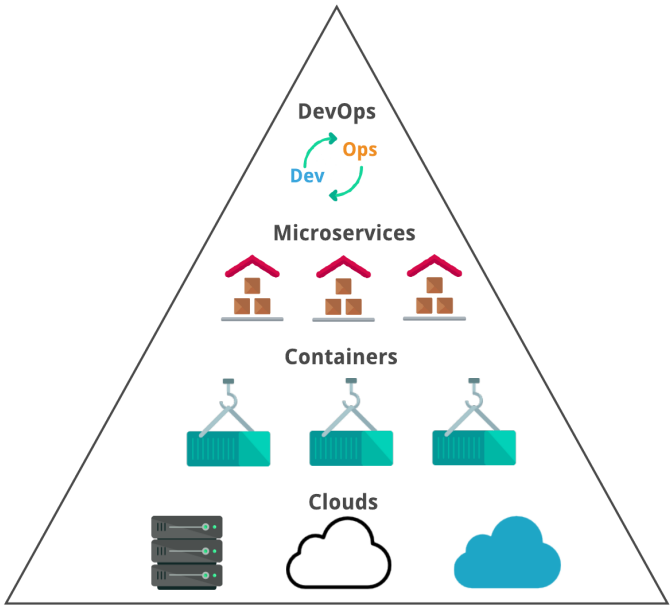


Figura 1: A pirâmide do desenvolvimento de aplicações empresariais modernas (baseado em [17], icons de Freepik [18], Roundicons [19]).

Markus Eisele [17] apresentou no seu livro *"Modern Java EE Design Patterns - Building scalable architecture for sustainable enterprise development"*, o que define como a pirâmide

do desenvolvimento de aplicações empresariais modernas (Figura 1), visando um crescimento sustentado de sistemas, baseados em arquiteturas que promovem a escalabilidade de aplicações de uma forma efetiva.

Elementos que compõem a Figura 1:

- DevOps - Refere-se ao método do processo de desenvolvimento de software e à interação com a componente de operações da infraestrutura.
- Microservices - Está relacionado com a componente arquitetural do software.
- Containers - A tecnologia onde as aplicações e as suas dependências vão estar instaladas.
- Clouds - Ambiente onde os *containers* são executados.

2.2 DEVOPS

Entre o término do código e a disponibilização deste, existe uma etapa denominada de *deployment* que se poderá revelar crítica e demorada em projetos em que a complexidade é acrescida devido à integração com sistemas, componentes ou ambientes heterogéneos [20].

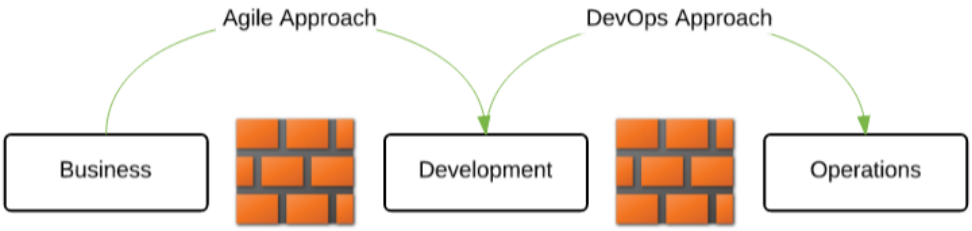


Figura 2: Business Process (Baseado em [21])

Se por um lado as equipas de desenvolvimento pretendem disponibilizar a versão mais recente do software, por outro, temos o lado operacional que promove a estabilidade do sistema. É neste último ponto que surge um conflito de interesses (Figura 2) entre as diferentes equipas que Lee Thomson e Andrew Shafer apelidaram de *Wall of Confusion* [22].

SILOS ORGANIZACIONAIS Apesar da aplicação dos métodos ágeis no desenvolvimento, os papeis das equipas das organizações continuam a seguir um modelo em cascata, tendo

como consequência a intensificação da existência de silos (pessoas ou equipas que não comunicam entre si) [23]. Além disso é frequente a existência de ambiguidades relativamente ao limite do âmbito do trabalho que cada equipa tem que realizar.

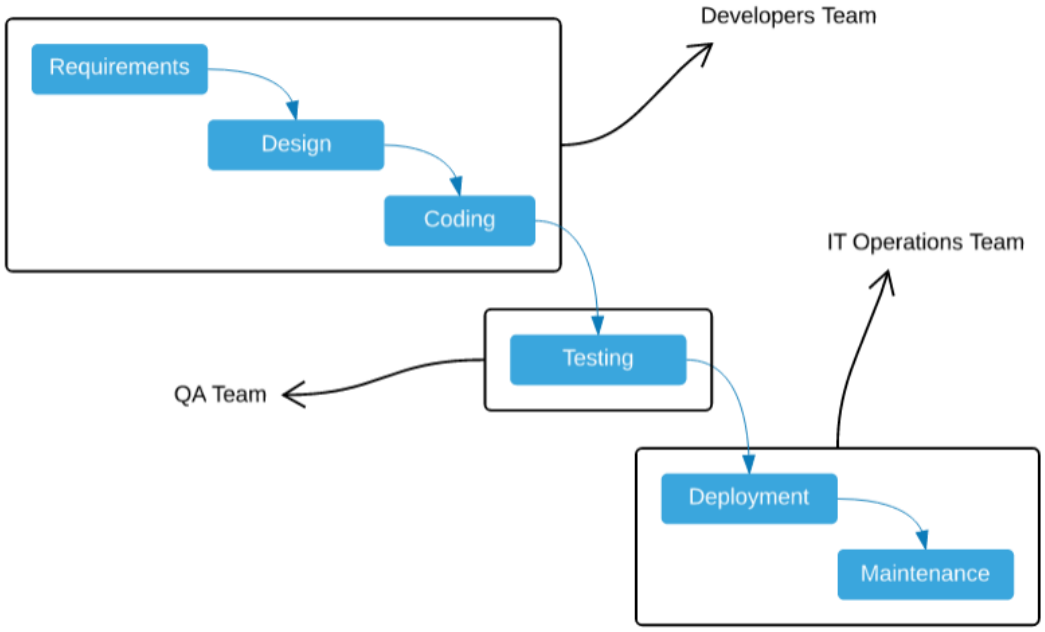


Figura 3: Silos Organizacionais (Baseado em [23])

Como podemos observar pela Figura 3, as diferentes equipas à medida que concluem as suas tarefas enviam para a equipa seguinte o artefacto gerado e assim sucessivamente, até que este ciclo termina no ambiente da equipa de operações quando é feito o *deployment* do software [24].

Sendo assim, DevOps pretende suprimir silos através da criação de equipas transfuncionais, ou mediante a promoção da comunicação entre estes de forma a aumentar a velocidade de *deployment*. DevOps é normalmente representado por um diagrama de *venn* que apresenta as equipas de desenvolvimento, operações e de qualidade. A zona de interceção de todas as equipas é denominada de DevOps. O diagrama é mostrado na Figura 4.

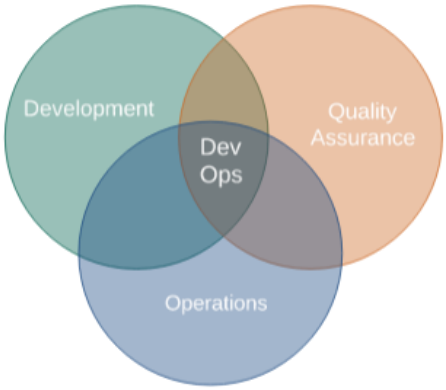


Figura 4: Diagrama de *Venn* - DevOps

DEFINIÇÃO O nome DevOps resulta de uma mistura de ***dev****elopment* + ***op****erations*, e podemos considera-lo como a resposta à problemática relacionada com o conflito de interesses das equipas de desenvolvimento e operações. Porém, não existe uma definição universal de DevOps e por esse motivo existem equívocos devido ao âmbito deste conceito. É recorrente a existência de mal entendidos, nomeadamente quando é referido DevOps como um papel que alguém desempenha, como um conjunto de ferramentas, entre outros [25].

Neste documento, DevOps consiste numa *framework* conceptual que é definida como um conjunto de práticas que pretende reduzir o tempo desde que é efetuada uma alteração num sistema e essa alteração é disponibiliza num ambiente de produção [7, 26].

Esta definição reúne de forma sintética os traços comuns das várias definições existentes na bibliografia atual.

VALORES E FILOSOFIA A importância de colocar novos requisitos em produção é fundamental, pois só desta forma sabemos se o que está a ser desenvolvido tem valor para o mercado, através do *feedback* recebido pelos utilizadores/clientes [27, 28].

A implementação de uma filosofia DevOps assenta em 3 (três) aspetos fundamentais [28]:

1. Mudança de mentalidade nas equipas através de um conjunto de valores estabelecidos;
2. Conhecer as técnicas e identificar quais as que melhor se ajustam à empresa/projeto;
3. Possuir e saber utilizar as ferramentas adequadas para executar as técnicas em questão;

Em 2010, na Conferência DevOpsDays em Mountain View (California), Damon Edwards e John Willis definiram 4 (quatro) valores base para a *framework* DevOps: Cultura, Automatização, Medição e Partilha, também conhecidos como CAMS [29].

Cultura A cultura DevOps pretende eliminar a "*Wall of Confusion*", a demolição desse muro é feita através da enfatização da componente humana e a forma como as pessoas se envolvem enquanto equipa, pois estas são os principais responsáveis pelo sucesso. As ferramentas, apesar de serem uma mais valia são relegadas para segundo plano, uma vez que "*se não existir uma cultura, todas as tentativas de automatização serão infrutíferas*" [21].

Automatização "*DevOps não é um problema tecnológico. DevOps é um problema de negócio*" [30]. Neste sentido, quando se adota *"DevOps como um estimulador de negócio, são fornecidas as ferramentas e cultura necessárias para facilitar o planeamento eficiente da release, previsibilidade e sucesso"* [28]. A automatização ao longo das diferentes etapas de teste, pré-produção e produção, tem como resultado ganhos diretos na produtividade, quer pela rápida execução das tarefas, quer pela prevenção do erro humano intrínseco à realização das tarefas executadas manualmente [28, 31].

Medição Segundo Willis [21] "Se não conseguimos medir, não podemos melhorar", assim sendo, é necessário um mecanismo rigoroso de monitorização que permita medir através de um conjunto de métricas o estado das tarefas já realizadas ou a realizar.

Estas métricas podem ser divididas em 5 (cinco) dimensões, são elas [32]:

- Eliminação de desperdício: métricas indicam a quantidade de tempo despendido nos diversos processos;
- Melhoria Contínua: métricas obtidas através da opinião dada pelos diversos colaboradores do projeto;
- Fluxo contínuo: quantidade de requisitos que as equipas conseguem adicionar aos *sprints*, que serão posteriormente disponibilizados sincronizadamente com produto final. *Sprint* é um intervalo de tempo em que a equipa de desenvolvimento trabalha com o objetivo de obter um conjunto de requisitos implementados no produto.
- Equipas multi funcionais: métricas relacionadas com a versatilidade da equipa, isto é, a capacidade que vários membros de uma equipa têm para executar tarefas de outra com a qualidade pretendida.
- Sistemas de informação: facilidade com que as diversas equipas têm acesso a documentação ou procedimentos para a realização de determinada tarefa.

Se por um lado é importante medir o tempo que uma nova versão do software demora a entrar em produção, é de igual modo importante medir o tempo que os erros demoram a ser corrigidos.

Sistemas de alarmística e recolha/análise de métricas estão em constante desenvolvimento de forma a encontrar padrões e tendências com o objetivo de minimizar o tempo de

deteção de falhas. O objetivo primordial é criar mecanismos automáticos de correção, e por conseguinte diminuir o tempo de recuperação da falha [31].

Partilha Os fundamentos DevOps têm como principal pilar o conhecimento empírico. A partilha de ideias, problemas ou soluções são fundamentais para a constante melhoria dos processos. Implementar este conceito poderá ser desafiante em empresas que possuem equipas com processos bem definidos há alguns anos, onde é comum a existência de silos. Silos ocorrem quando as equipas de desenvolvimento não partilham o seu conhecimento com as equipas de operações, no entanto esta problemática também pode ocorrer dentro da própria equipa de operações, que é normalmente referida como "*Ops-Ops problem*"[6, 33].

2.2.1 *Continuous Delivery*

Em Setembro de 2012 no âmbito de um evento organizado pela Agile Quebec City e Elapse Technologies, Robert C. Martin fez uma apresentação de nome "Exigir Profissionalismo em Desenvolvimento de Software" [34], onde foram abordados problemas que ocorrem no desenvolvimento de software, motivando o atraso no *deployment* de software.

Segundo Robert C. Martin, "*I want you ready all the time, find some way, i dont care how, to deploy all the time*".

Com esta afirmação o autor defende que o código que é colocado na base do projeto onde é feito o desenvolvimento (conhecida como *mainline* ou *trunk*) deve ser código funcional, isto é, o código antes de ser adicionado ao repositório deve ser compilável e passar por todos os testes especificados.

Surge desta forma o conceito de *Continuous Delivery*, que é a capacidade que uma equipa tem de fazer *deploy* a qualquer momento. Um modelo de desenvolvimento orientado a testes pode ser um elemento frutífero para alcançar o pretendido.

Esta capacidade permite aumentar a qualidade, reduzir custos e incrementar o número de entregas.

Com o incremento do número de entregas (débito), aumenta a capacidade de obter mais rapidamente um *feedback* pelos utilizadores.

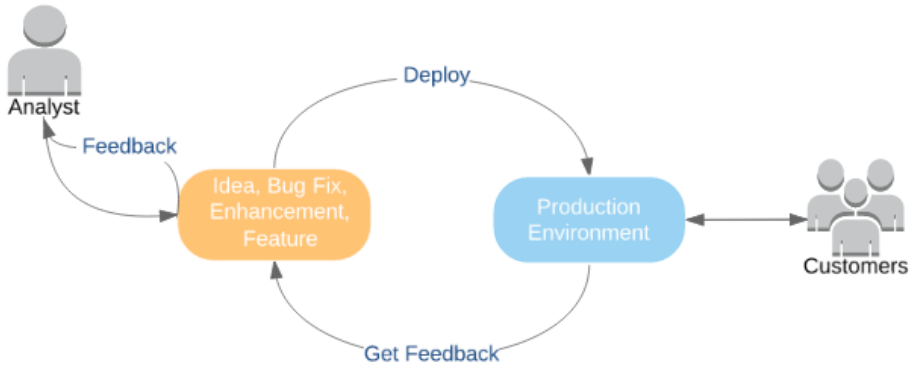


Figura 5: Feedback Contínuo (Baseado em [35])

Através de um rápido *feedback* dos utilizadores é possível [35]:

- Melhorar a qualidade do software a entregar.
- Aperfeiçoar o ambiente (infraestrutura) em que o software está a ser disponibilizado.
- Lapidar o processo de entrega de software.

Para obtermos esta sequência iterativa de *feedbacks* deve-se ter em conta três aspetos primordiais: gestão de configurações, integração contínua e testes automáticos.

Delivery Pipeline

Delivery Pipeline pode ser considerada como a base do *Continuous Deliver*. Esta permite o fluxo constante das mudanças entre as várias etapas por si compostas através de sistemas automatizados. Através da *Delivery Pipeline* é possível identificar obstáculos que ocorrem nas transições das etapas e avaliar o progresso do projeto, aumentando assim a capacidade da equipa tomar decisões assertivas.

A Figura 6 representa a *Delivery Pipeline* composta pelas etapas que serão usadas como referência para este trabalho.

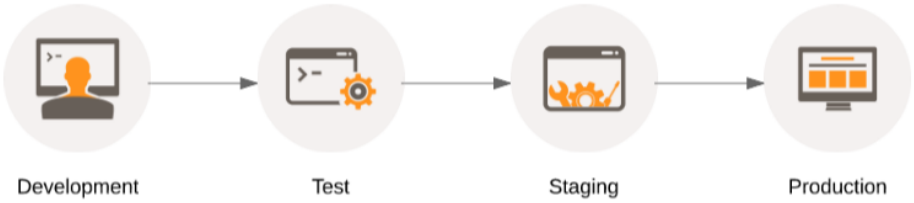


Figura 6: Delivery Pipeline

A pipeline inicia com o desenvolvimento, onde o programador desenvolve o código e testes do código desenvolvido. Antes de proceder ao *commit* das alterações para o repositório o código deve passar em todos os testes localmente. Quando o *commit* é efetuado, é despoletado um evento que procede à realização dos testes de integração num ambiente de testes. Caso não ocorram erros nos testes o código é integrado e enviado para o ambiente de pré-produção. Este ambiente deve ser idêntico ao ambiente ao de produção, e devem proceder-se a novos testes, como por exemplo: testes de stress ou testes manuais relacionados com *user experience*. Após a validação de todos os requisitos de qualidade, o código é enviado para o ambiente de produção [7].

Uma prática DevOps consiste em garantir que os ambientes sejam homogéneos em todas as etapas da pipeline. Para tornar esse processo eficaz utilizam-se ferramentas de provisionamento, orquestração ou sistemas de gestão de configurações que aceleram e gerem esses mesmos ambientes com o mínimo de esforço.

Deliver Paradigms

Nesta secção são apresentados alguns modelos que são utilizados para o processo de gestão das configurações da infraestrutura a vários níveis, tais como: configurações de rede, Sistema Operativo (SO), serviços etc [36].

- **Ad-hoc** é uma abordagem em que a configuração dos servidores são feitas manualmente o que origina uma constante incoerência no estado destes. Os servidores estão recorrentemente num estado desconhecido, na medida em que apenas são feitas alterações à medida (*Ad-hoc*) quando necessário. A utilização de ferramentas de automatização revela-se quase inadequada uma vez que o estado atual de cada sistema é desconhecido, e é praticamente impossível efetuar a replicação das configurações deste tipo de sistemas. Este tipo de sistemas é vulgarmente apelidado de *Snowflake Servers* [36, 37].
- **Configuration synchronization** [36] ou **Runtime Configuration** [38] consistem em agendar regularmente alterações das configurações aos servidores para garantir que estes estão num estado coerente.
- **Immutable infrastructure** [36] consiste em criar novos servidores em detrimento de fazer alterações nos já existentes. Este padrão foi apelidado por Martin Fowler [37] de *PhoenixServers*. Nesta situação é possível assegurar com total certeza que o servidor se encontra num estado coerente, uma vez que eventuais problemas relacionados com configurações manuais são eliminados. Aspetos que são necessários ter consideração são os que estão relacionados com o estado do servidor em tempo de execução e os dados das aplicações que estão a ser executadas, uma vez que estes

não são testados.

2.2.2 *Gestão de Configurações*

Segundo a definição do *Swebok* [39]: *"Gestão de configurações é a disciplina de identificar a configuração de um sistema em diferentes pontos distintos no tempo com o propósito de sistematicamente controlar mudanças das configurações e manter a integridade e a rastreabilidade das configurações ao longo do ciclo de vida do sistema"*.

Gestão de configurações remete para um processo composto por varias atividades entre as quais se destacam: a identificação, armazenamento, controlo de mudança, informação do estado.

Os itens resultantes deste processo devem ser rastreáveis e para tal recorre-se a metadados onde são guardadas várias informações como por exemplo o nome do autor do artefacto, data, item predecessor, ou outros itens relacionados com este. Cada item existente no sistema de gestão de configurações tem um identificador único que o distingue dos demais existentes. Assim que este é adicionado ao sistema deve estar guardado de forma a que seja possível encontra-lo a qualquer altura.

Neste documento, controlo de versões refere-se à gestão de versões de ficheiros de código, documentação etc. Por outro lado, gestão de configurações é visto como um conceito mais abrangente, que incide essencialmente na alteração do estado dos sistemas.

As técnicas utilizadas pelas organizações para efetuar a gestão das configurações de software podem variar, no entanto existem pressupostos que indicam se os métodos utilizados são adequados.

A capacidade de reproduzir integralmente um determinado sistema num determinado período, saber quem fez determinadas alterações ou a facilidade com que as alterações são efetuadas e disponibilizadas, são algumas das exigências fulcrais para um sistema de gestão de configurações [27, 40, 41].

Gestão de Configurações Automáticas

Se através de sistemas de gestão de configurações temos a capacidade de alterar um sistema de um determinado estado atual para um outro qualquer estado definido anteriormente, é de extrema importância manter a consistência e uniformidade nesses mesmos ambientes onde o software vai ser instalado. Recorrendo à automatização de processos e procedimentos, em detrimento de configurações manuais, garante-se a consistência nos ambientes e um incremento na produtividade. A automatização é feita através de ferramentas que nos permitem fazer a gestão de configurações automáticas, sejam elas *scripts* ou ferramentas avançadas de aprovisionamento e/ou orquestração [42].

2.2.3 *Infrastructure as Code*

Infrastructure as Code (IaC) [43], por vezes apelidada de "*programable infrastructure*", é normalmente citado como um princípio DevOps que consiste em automatizar tarefas repetitivas através de código. Contrariamente à programação de *scripts*, ferramentas de IaC recorrem normalmente a *Domain-Specific Language (DSL)*, *Data Format Language (DFL)* ou outras linguagens de alto nível que se apresentam como uma opção mais adequada e voltada para atividades de aprovisionamento e orquestração. Uma vez que estamos a utilizar código para programar a infraestrutura temos a possibilidade de aplicar conceitos de engenharia de software, que resulta uma maior credibilidade nos processos e a desejada consistência nos sistemas, uma vez que podemos executar testes sobre o código programado e assim aferir o bom funcionamento deste [36].

2.2.4 *Integração Contínua*

Integração Contínua [44], é uma prática que consiste em incorporar o código que está a ser desenvolvido com frequência junto do código já existente.

Através desta prática, as equipas de desenvolvimento sabem que a versão do último código disponível no repositório será sempre uma versão estável que passou com sucesso em todos os testes e na etapa de compilação (*Build*).

A diminuição do intervalo de tempo em que são feitas as integrações de software, tem como consequência direta a deteção precoce de erros, devendo estes ser corrigidos com prioridade máxima para que se evite a acumulação destes. Ao se evitar a acumulação de erros, o esforço necessário para a sua correção é reduzido.

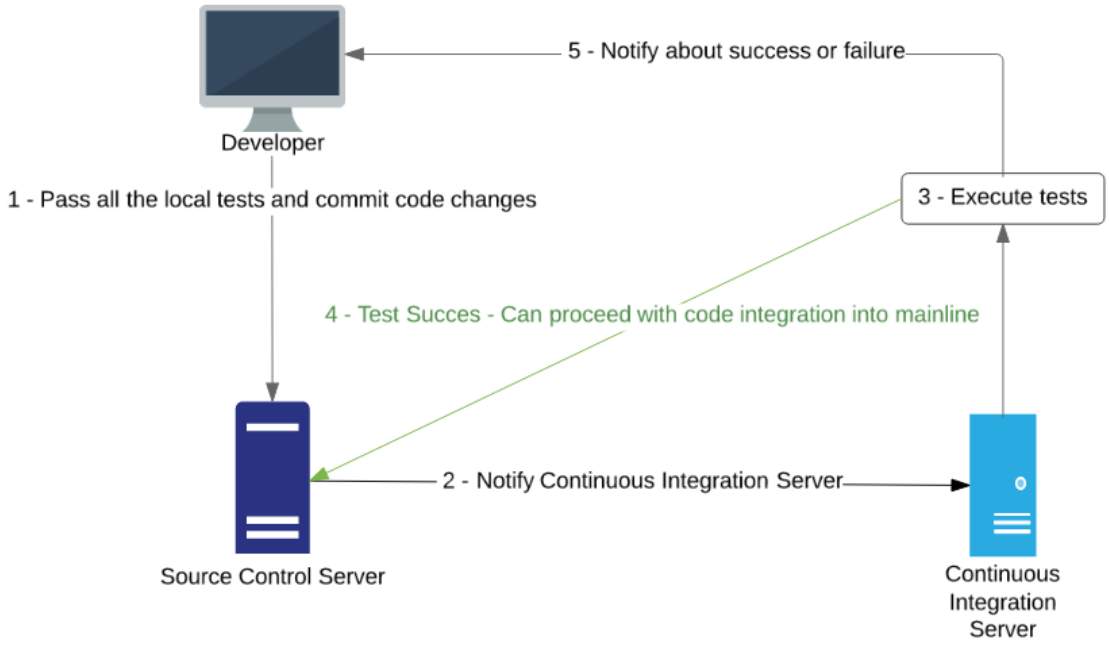


Figura 7: Integração Contínua - Fluxo genérico

O funcionamento da integração contínua (Figura 7) tem várias fases. Numa fase inicial, o programador possui uma cópia da versão do código estável na sua máquina de desenvolvimento. Findo o trabalho e antes deste ser enviado para o repositório, o programador deve verificar que o código passa em todos os testes localmente. No intervalo de tempo em que o programador esteve a efetuar alterações no código, a versão estável atual do repositório pode ter sofrido alterações. Deste modo, deve ser verificado localmente se não existem conflitos entre a versão local e a versão atual do código do servidor. Caso não existam conflitos, deve-se proceder à integração, caso contrário deverão ser feitas as devidas correções. Uma das formas para agilizar este processo consiste em recorrer a ferramentas de integração contínua [27, 45].

2.2.5 *Testes Automáticos*

Testes automáticos [46] é um processo que usa ferramentas para executar *scripts* que contêm um conjunto de verificações que devem ser validadas numa aplicação, componente ou módulo de *software* antes destes serem colocados em produção.

Existem diferentes tipos de testes que podem ser aplicados nas diversas etapas do ciclo de vida do software para garantir a conformidade do software desenvolvido com os requisitos especificados.

Testes funcionais, testes unitários, testes de segurança, testes de stress, testes de desempenho, são alguns exemplos dos vários tipos de testes que podem ser realizados [47].

Podemos beneficiar da utilização de ferramentas de testes de software em várias situações [33]:

- Testes que são realizados de forma manual, quando o programador pretende garantir que o software não contém erros para posteriormente fazer o *commit* do código para o sistema de controlo de versões.
- Testes que são agendados, normalmente demorados e que podem ser realizados a horas em que os programadores não estejam a trabalhar no projeto.
- Testes que são despoletados por eventos, por exemplo, quando um *commit* é efetuado para o sistema de controlo de versões. Este é um caso típico quando aplicamos a prática de integração contínua.

Testes de software consomem muitos recursos computacionais, são demorados, árduos e propícios a erros. Apesar dos constantes esforços para assegurar a qualidade do software, é comum a entrega de software com defeitos [48].

Para colmatar alguns dos problemas intrínsecos à realização de testes, recorre-se a soluções de testes automáticos para tornar o processo de testes mais eficiente.

2.3 MICRO SERVIÇOS

O estilo arquitetural de micro serviços é um conceito que começou a ganhar maior visibilidade a partir do ano 2014.

O estilo arquitetural de micro serviços é um conceito ajustado para a *cloud* (Secção 2.5) e tem o intuito de promover o desacoplamento de aplicações através da criação de pequenos serviços.

Se numa abordagem tradicional a solução apresentada é normalmente composta por um grande bloco monolítico, numa arquitetura baseada em micro serviços, as aplicações são divididas em múltiplas e distintas partes.

Na arquitetura de micro serviços o desacoplamento é feito através da separação de funcionalidades por pequenos serviços, em que cada um é executado num único processo, e o não funcionamento de um serviço, não tem, obrigatoriamente de significar a falha completa da aplicação.

Idealmente os micro serviços utilizam protocolos leves para comunicar, como por exemplo: o protocolo HTTP, MQTT [49] ou NATS [50]; através de interfaces bem definidas.

A migração de aplicações para micro serviços introduz algumas vantagens como a redução do *time-to-market*, uma melhor organização no processo de desenvolvimento ou uma maior flexibilidade nas tecnologias a utilizar.

Aplicações que seguem uma arquitetura baseada em *microservices* são normalmente instaladas e executadas em *containers* (Secção 2.4) [51–53].

2.4 CONTAINERS

O conceito de *container* é normalmente apelidado de *lightweight virtual machine*, uma vez que é uma tecnologia de virtualização que opera a nível do sistema operativo, contrariamente à abordagem de máquinas virtuais (VMs) que recorrem à virtualização ao nível de hardware.

Um *container* é construido para encapsular e executar a aplicação com todas as suas dependências (*Runtime*, bibliotecas, etc), idealmente com um único processo. Apesar de partilharem recursos do sistema operativo onde estão a ser executados, estes recursos estão completamente isolados entre si. Ou seja, um *container* não consegue identificar que processos estão a ser executados no sistema operativo anfitrião, ou noutro *container*.

Esta tecnologia permite um arranque muito rápido, e um *scale-up/scale-out* de aplicações muito eficiente comparativamente às VMs. Tal deve-se ao maior *overhead* causado pelo maior número de camadas tecnológicas de abstração que existe na *stack* de VM como se pode observar na Figura 8 [53].

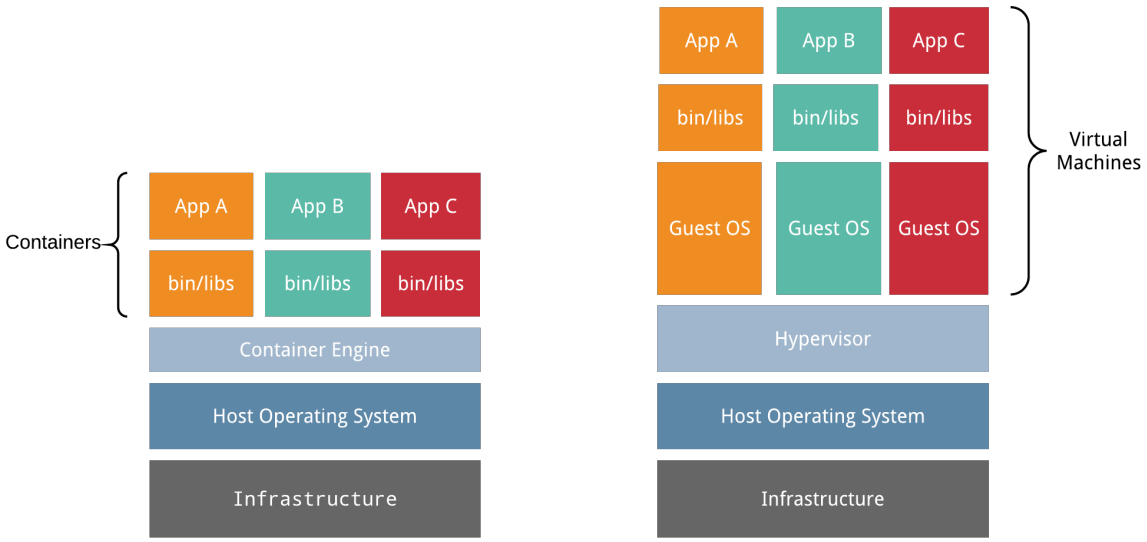


Figura 8: Comparação da stack das tecnologias Containers e Máquinas Virtuais

Algumas das tecnologias mais populares são o Docker [54] que está disponível para os sistemas operativos: Windows, Linux e MacOS; LXC [55], Rocket [56] e OpenVZ [57] para distribuições Linux; e ainda o Jails [58] para o sistema operativo FreeBSD.

2.5 COMPUTAÇÃO CLOUD

2.5.1 *Definição*

The National Institute of Standards and Technology (NIST) Definition of Cloud Computing escrito por Mell and Grance [59] é provavelmente a definição mais consensual de computação *cloud*, que é definida como um modelo que permite o acesso *on-demand* a um conjunto de recursos como por exemplo: redes, servidores, armazenamento. As aplicações e serviços, são fornecidos e disponibilizados com um esforço reduzido de gestão. Esta definição é composta por 5 (cinco) características essenciais, 3 (três) modelos de serviço, e 4 (quatro) modelos de implementação" [59].

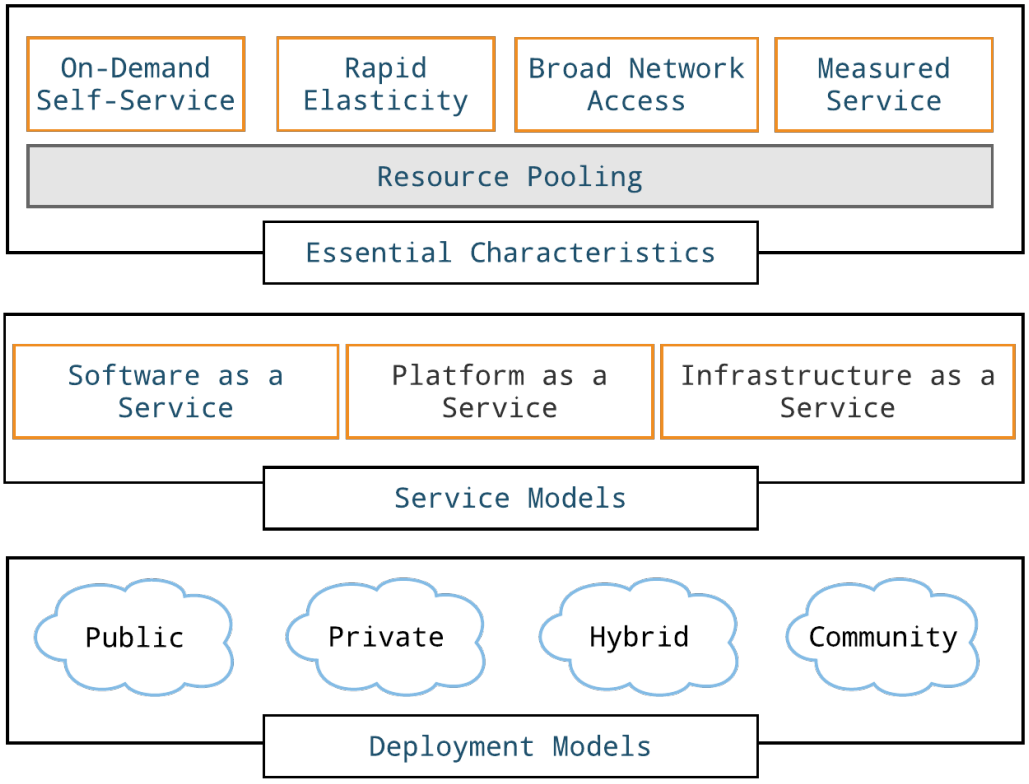


Figura 9: NIST - Modelo visual da definição de Computação *Cloud* [59]

2.5.2 *Características*

Segundo a definição de NIST [59], a computação *cloud* é caracterizada por:

- permitir que o cliente possa gerir os recursos que precisa de forma autónoma (*On-demand self-service*);

- estar disponível através da Internet de forma a ser acedida pelos mais variados dispositivos, como por exemplo: *smartphones*, *tablets* ou computadores (*Broad network access*);
- possuir uma *pool* de recursos que podem ser distribuídos dinamicamente pelos diversos clientes (*Resource pooling*).
- ser capaz de escalar e libertar recursos de uma forma eficiente, e se possível de forma automática (*Rapid elasticity*).
- estar capacitada para controlar e otimizar recursos tendo por base a monitorização dos serviços que disponibiliza (*Measured service*).

2.5.3 *Níveis de Serviços*

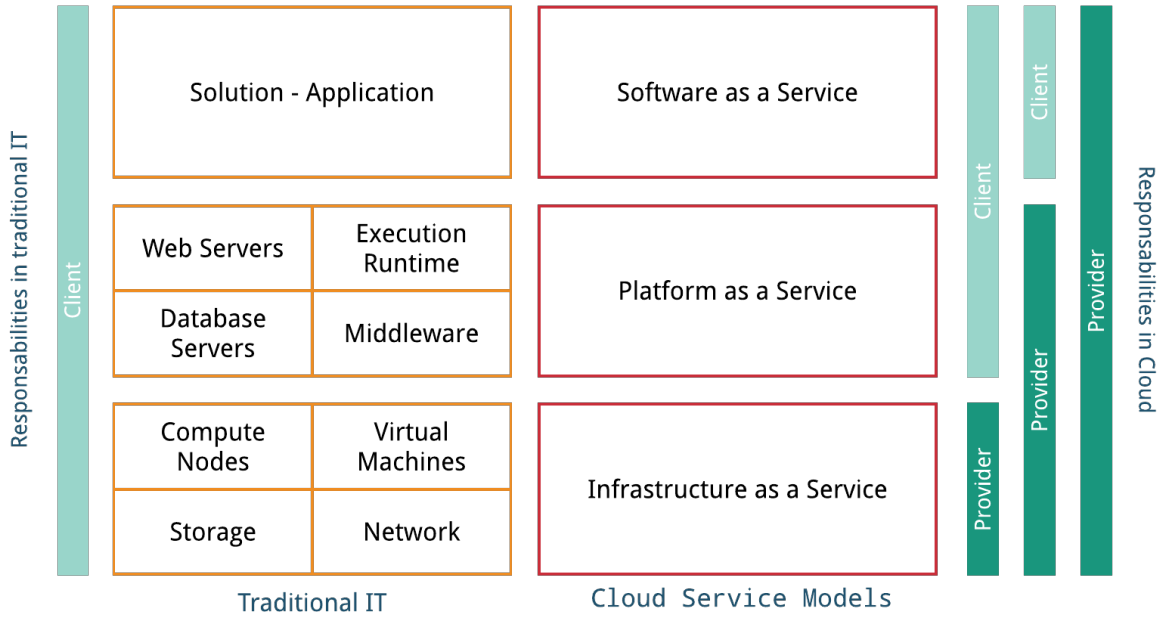


Figura 10: Traditional IT vs Cloud Service Models

No modelo tradicional o cliente é responsável por gerir toda a *stack* da infraestrutura (Figura 10). Na tecnologia *cloud* é permitido que o cliente delegue a responsabilidade de cada uma das três camadas para um fornecedor.

Infrastructure as a Service

A camada IaaS fornece serviços a nível da infraestruturas, permite provisionar recursos virtuais de computação, armazenamento ou componentes de rede. O cliente fica sem a ne-

cessidade de se preocupar com a gestão dos componentes físicos (*Hardware*) integrantes da solução que pretende [59]. Geralmente este serviço recorre ao modelo de *pay-as-you-go*, onde o cliente só paga o que utiliza, evitando desta forma investimentos a longo prazo em dispositivos físicos, e consequentemente a redução de riscos, e um maior controlo de custos.

Alguns dos fornecedores de serviços da camada IaaS com mais relevância na atualidade são a *Amazon Web Services* (AWS) [9], Microsoft Azure [60], Google Cloud Platform [61], VMware vCloud Air [8], ou a Rackspace [62]. As tecnologias dos diversos prestadores de serviços supra mencionados podem variar entre soluções proprietárias ou código aberto. Algumas tecnologias que operam na camada de IaaS são: OpenStack [12], CloudStack [63], HPE Helion Eucalyptus [64], OpenNebula [65].

Plataform as a Service

A camada de *Platform as a Service* (PaaS) fornece ao utilizador um ambiente completo onde este pode fazer *deploy* da sua aplicação. O cliente/utilizador não tem a possibilidade de gerir os componentes da infraestrutura como nodos de computação, dispositivos de armazenamento de dados ou redes. Por outro lado, tem à sua disposição um *runtime* onde pode fazer *deploy* da sua aplicação com o mínimo de esforço possível, pois é-lhe fornecido um ambiente com todas as dependências para a execução da sua aplicação, independentemente da linguagem de programação ou tecnologia utilizada. Neste sentido o programador não tem que se preocupar com o ambiente que vai dar suporte à aplicação, pois o foco é direcionado para o rápido desenvolvimento e disponibilização do produto final [59].

Os produtos que fornecem serviços de PaaS com maior impacto na comunidade são o Microsoft Azure [60], Pivotal Cloud Foundry [66], IBM Bluemix [67], Bosch IoT Cloud [68], Heroku [10] ou o Red Hat Openshift [69]. Alguns dos produtos mencionados têm por base tecnologia *open source*, e que pode ser instalada em ambientes locais como é o caso do Openshift Origin [69], ou do Cloud Foundry [13].

Software as a Service

O serviço prestado ao nível de *Software as a Service* (SaaS) consiste em disponibilizar ao cliente final uma solução de software que está a ser executada na infraestrutura *cloud*, em que o responsável pelo ciclo de vida desse mesmo software é da inteira responsabilidade do fornecedor. As atualizações, a adição de novas funcionalidades ou correções de erros do software são da responsabilidade do fornecedor [59]. O cliente não tem acesso a nenhuma camada de serviços abaixo do SaaS, portanto não tem a possibilidade de controlar/gerir recursos computacionais. Exemplos práticos de SaaS são o Gmail [70] ou o Office 365 [71].

2.5.4 *Modelos de Implementação*

São 4 (quatro) os modelos de implementação definidos por Mell and Grance [59]:

Cloud Privada

Neste modelo a infraestrutura é dedicada exclusivamente para uso interno de uma organização, ou seja, apenas um grupo restrito de utilizadores tem acesso aos recursos disponibilizados. A gestão e instalação da infraestrutura pode ser efetuada na própria organização (*on-premises*), ou recorrendo a entidades externas (*off-permises*).

Cloud Pública

O modelo de *cloud* pública é aplicado quando se pretende fornecer serviços para o público em geral através da Internet.

Cloud Híbrida

A *cloud* híbrida é uma mistura de dois ou mais modelos de infraestruturas *cloud*, como por exemplo: *cloud* privada, comunitária ou publica, formando uma única entidade.

Cloud Comunitária

Na *cloud* comunitária os recursos de infraestrutura são partilhados por diversas organizações onde comunidades possuem interesses comuns relativamente a aspetos de segurança, jurisdição, etc. Consecutivamente permite a redução de custos face a uma implementação de uma *cloud* privada individual para cada organização.

3

PROBLEMA E DESAFIOS

O problema que se pretende solucionar é intrínseco ao projeto P10. O facto deste se inserir num projeto de I&D, traduz-se numa grande volatilidade nos requisitos. Novas versões de software surgem constantemente e é necessário valida-las. Deste modo, é importante que estas sejam disponibilizadas para serem testadas pelo cliente e/ou utilizadores com o menor esforço e brevidade possível.

Nesse sentido pretende-se dotar a equipa de ferramentas e recursos computacionais disponibilizados por uma infraestrutura capaz de responder à necessidade de lidar com a grande volatilidade nos requisitos.

3.1 DESAFIOS

Para resolver o problema pretende-se implementar uma infraestrutura capaz de dar resposta às necessidades de uma equipa que se rege por um desenvolvimento moderno de aplicações. Para tal será definida uma arquitetura com base nos requisitos do sistema e restrições a nível do projeto.

Pretende-se criar uma *Delivery Pipeline* e avaliar de que forma os ambientes de testes/pré-produção podem manter a homogeneidade ao nível do serviço de PaaS, com a solução comercial da *Robert Bosch BMG* denominada de *Bosch IoT Cloud*. Existem ainda desafios tecnológicos que advêm da imprevisibilidade do número de clientes que a aplicação pode ter, assim como a carga no sistema daí resultante.

A resolução dos problemas que surgem no decorrer do projeto tendem a ser mais complexos e difíceis de gerir à medida que as equipas crescem, não só pelos problemas técnicos, mas também pelo défice na eficácia na comunicação. Desta forma, além de se capacitar as equipas com ferramentas e tecnologias para ajudar na resolução dos problemas, pretende-se utilizar um processo eficaz em que os riscos relacionados com o atraso do *deployment* de software sejam reduzidos.

A escolha da arquitetura, a automatização das tarefas e a comunicação entre os vários componentes da arquitetura, são provavelmente os maiores desafios. Como tal, a escolha das ferramentas revela-se crucial. Aspetos como: curva de aprendizagem, projetos *open*

source, funcionalidades e documentação abrangente, serão os mais relevantes na escolha final [24].

Depois de identificados quais as causas para o atraso da passagem dos artefactos entre as diferentes etapas da *Delivery Pipeline*, surgem uma série de desafios para mitigar a ocorrência desses desvios.

3.2 REQUISITOS

Nesta secção são descritos os requisitos referentes à infraestrutura para o projeto em questão, no contexto da sua operacionalidade e resposta às expectativas de qualidade do serviços a disponibilizar.

3.2.1 *Requisitos Funcionais*

Descrição
O administrador do sistema aprovisiona VMs, redes virtuais, dispositivos de armazenamento ou *runtimes* a pedido.
Componente de Suporte
- Serviço de computação do OpenStack (Nova). - Serviço de block storage do OpenStack (Cinder). - Serviço de rede do OpenStack (Neutron). - Componente DEA ou Diego do CloudFoundry.

Descrição
O administrador gere os recursos através de uma dashboard.
Componente de Suporte
- Dashboard do OpenStack que comunica com o componente Horizon.

Descrição
O administrador atribui recursos a diferentes grupos de utilizadores.
Componente de Suporte
- Capacidade embutida de multi projetos no OpenStack. - Capacidade embutida no Cloud Foundry, através de Spaces e Orgs.

Descrição
O sistema recolhe métricas para analisar o estado e o desempenho dos serviços.
Componente de Suporte
- Agentes Metron do Cloud Foundry. - Agentes NRPE do Nagios.

Descrição
O administrador atribui IPs dinamicamente aos *hosts* do sistema.
Componente de Suporte
- Serviço de DHCP.

3.2.2 *Requisitos Não Funcionais*

A categorização dos requisitos não funcionais é feita de acordo com a norma ISO 25010 [72] - *Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE)*.

Categoria
Disponibilidade
Descrição
O sistema escala os recursos computacionais e de armazenamento sem a interrupção dos serviços em execução.
Componente de Suporte
- *Bosh Outer Shell* (BOSH)

Categoria
Disponibilidade, Desempenho
Descrição
O sistema distribui a carga computacional pelos nodos de computação.
Componente de Suporte
- *Scheduler* do OpenStack que efetua o balanceamento dos recursos computacionais e armazenamento, mediante a carga dos dispositivos físicos. - *Diego Auction* faz o balanceamento dos processos de aplicações sobre as várias VMs disponibilizadas pela instalação do Cloud Foundry.

Categoria
Confiabilidade, Tolerância a Falhas & Disponibilidade
Descrição
O sistema opera após uma falha de um componente sem a indisponibilidade total do sistema.
Componente de Suporte
- Controlador RAID (Configurado em RAID 1). - *Uninterruptible Power Supply* (UPS).

Categoria
Confiabilidade, Instalabilidade
Descrição
Os componentes do sistema são replicados através de soluções automatizadas.
Componente de Suporte
- Packstack. - Scripts automatização. - Kickstart.

Categoria
Manutenibilidade, Operabilidade, Instalabilidade
Descrição
O sistema utiliza tecnologias que possuem suporte/documentação abrangente e/ou oficial.

Categoria
Desempenho, Segurança
Descrição
O sistema deve adaptar os seus recursos computacionais para assegurar um nível aceitável de prestação de serviços.
Componente de Suporte
- Serviço de escalabilidade horizontal e vertical do Cloud Foundry.

Categoria
Confiabilidade, Tolerância a Falhas
Descrição
O sistema deve ser resiliente a falhas de software permitindo várias instâncias da mesma aplicação.
Componente de Suporte
- Componente *cloud controller* do Cloud Foundry.

Categoria
Segurança
Descrição
O sistema fornece acesso aos recursos apenas aos utilizadores/dispositivos autorizados.
Componente de Suporte
- Serviço Keystone do OpenStack. - Serviço UAA do Cloud Foundry. - Serviço SSH, permite acesso às VMs através de chaves públicas. - Serviço Neutron do Openstack, que utiliza a tecnologia *Virtual Extensible LAN* (VxLAN) nas redes virtuais criadas. - Switch físico que isola as várias redes recorrendo a VLANs.

Categoria
Desempenho, Usabilidade & Tempo de Resposta
Descrição
O sistema deve estar ligado a um *Internet Service Proviver* (ISP), que permita uma ligação de rede de alto débito.
Componente de Suporte
- Router ligado à rede da Universidade do Minho (UM).

3.2.3 *Restrições*

1. Homogeneização da camada de PaaS com o *Bosch IoT Cloud* (BIC), através da plataforma Cloud Foundry.
2. Instalação de uma solução IaaS que seja oficialmente suportada pela plataforma Cloud Foundry.

4

CONCEÇÃO E IMPLEMENTAÇÃO

4.1 DECISÕES TECNOLÓGICAS

Com base nos requisitos e restrições descritos na Secção 3.2 são explicadas as decisões tomadas a nível tecnológico no decorrer do projeto.

4.1.1 *Sistemas de Controlo de Versão*

No que diz respeito à escolha da tecnologia para efetuar o versionamento de ficheiros, existem restrições do projeto, que limitam essa escolha às tecnológicas *Subversion* (SVN) [73] ou Git [74]. Por esse motivo foi necessário efetuar um estudo e uma escolha consciente sobre que decisão tomar relativamente a este contexto.

Em 2015 o site *Stackoverflow* [75], que ocupa a posição 56 do *Ranking Alexa* [76], fez um inquérito para identificar o perfil dos seus visitantes. Uma das questões era qual o sistema de controlo de versão que os seus utilizadores utilizavam. Participaram neste inquérito mais de 26 mil pessoas de 157 países.

Com um total de 16.694 respostas a esta pergunta, verifica-se pela Figura 11 que as soluções mais utilizadas são o Git com uma larga vantagem sobre o SVN, e em terceiro lugar uma opção proprietária, o Team Foundation Server [77].

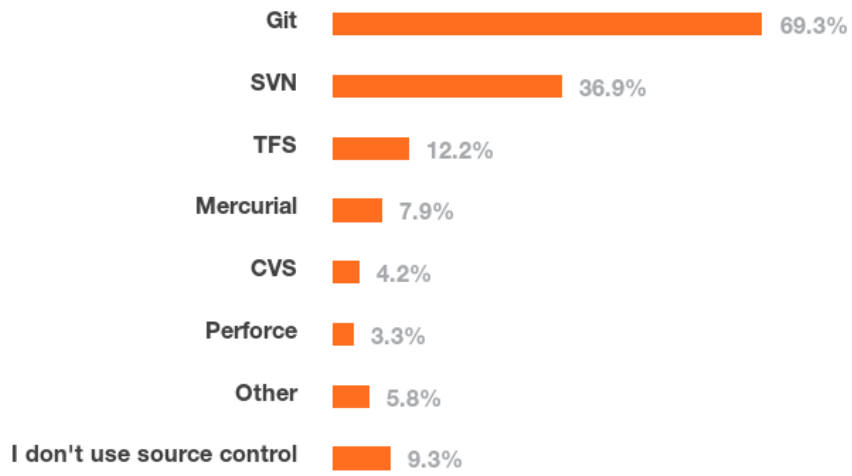


Figura 11: Ferramentas de Controlo de Versão - Developer Survey 2015 [78]

O Git foi criado por *Linus Torvalds* para dar suporte à gestão de controlo de versão de ficheiros do *kernel* do *linux*. Ao longo dos anos muitas organizações têm adotado e migrado os seus projetos para o Git como por exemplo: a Google, Microsoft, Facebook, Netflix ou KDE [74].

O Subversion [73] está em desenvolvimento desde 2000 e apresenta-se como uma solução estável e também utilizada por grandes empresas, no entanto, nos últimos anos tem-se assistido à migração de projetos com alguma notoriedade de SVN para Git. Analisando a Figura 12 verifica-se uma tendência (Fonte: Google Trends 2016) de crescimento do Git sustentando, em contraste com o decréscimo do SVN.

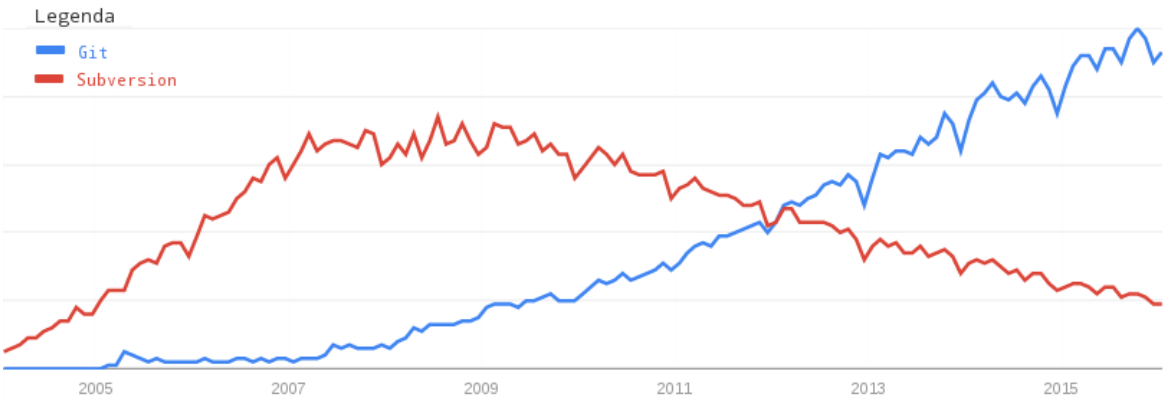


Figura 12: Git vs Subversion - Interesse ao longo do tempo (Fonte: Google Trends, 2016)

A característica mais evidente que distingue as duas ferramentas em questão são o modelo que estas adotam. No sistema Git é utilizado um modelo de controlo de versão distribuído. Este modelo permite que os clientes efetuem cópias completas do repositório para as suas máquinas locais. No caso de uma falha no servidor, qualquer repositório dos clientes pode servir como base para recriar o repositório no servidor. Deste ponto de vista, pode-se considerar uma vantagem face ao modelo centralizado que é utilizado no SVN.

No modelo centralizado o servidor é um ponto único de falha, isto significa que numa eventual indisponibilidade do servidor, a equipa fica impossibilitada de realizar operações no repositório. Caso se verifiquem problemas no repositório, este não pode ser reconstruido (como acontece nos modelos distribuídos).

Ambas as soluções são gratuitas e *open source*, os respetivos sites oficiais contemplam documentação extensiva e detalhada. Pela sua maturidade o Git e o SVN dão garantias de serem opções confiáveis para projetos de qualquer dimensão [74, 79, 80].

Face ao exposto, optou-se por utilizar a tecnologia Git através da plataforma Bitbucket.

4.1.2 *Servidor de Integração Contínua*

Existem várias opções que preenchem os requisitos pretendidos para serem executadas no servidor de integração contínua, entre as quais o Jenkins [81] e o Bamboo [82].

No projeto para além do Bitbucket [83] também é utilizado o JIRA [84] para gestão e planeamento do projeto, ambos os produtos fazem parte da suite da Atlassian [85]. Para se tirar todo o potencial que é dado pela integração das várias ferramentas é escolhido o Bamboo.

O Bamboo tem integração embutida com os serviços Bitbucket e JIRA, desta forma a equipa fica com a capacidade de rastrear um artefacto de software compilado no Bamboo, saber quem fez a última alteração no software, e qual a tarefa que deu origem a essa alteração.

Algumas das funcionalidades mais importantes que o Bamboo fornece para o projeto, são as *builds* e testes automáticos que são acionados após um *commit*, e o *deploy* automático do artefacto gerado para um ambiente especifico.

4.1.3 *Tecnologias PaaS*

Bosch IoT Cloud

Face ao desenvolvimento de vários produtos capazes de se ligarem à Internet, a tecnologia *Internet of Things* (IoT) transformou-se numa área primordial e transversal a todas as divisões de negócio da *Robert Bosch GmbH*, foi nesse contexto que a a mesma lançou a sua própria solução *Cloud* denominada de BIC.

Os pilares tecnológicos da camada de PaaS do BIC são baseados na solução comercial da plataforma *Cloud Foundry* da Pivotal. Como já explanado durante este documento, é fundamental manter a homogeneidade nos ambientes ao longo do ciclo de vida do software para mitigar o risco de problemas exististes no processo de *deployment* do software, mantendo assim o controlo sobre o comportamento do software instalado nos diferentes ambientes. Desta forma, surge assim uma restrição a nível tecnológico.

Cloud Foundry

O Cloud Foundry é uma plataforma *open-source* que opera na camada de PaaS, que fornece aos programadores um ambiente de execução para containers preparado para contextos de produção. A sua escolha resulta de uma restrição tecnológica implícita dos requisitos como descrito na Secção 3.2.

Uma das características do Cloud Foundry é que este faz a distinção entre aplicações e serviços. As aplicações são executadas em *containers*, idealmente devem seguir as indicações da metodologia "*The Twelve-Factors App*" [86]. Os serviços, como por exemplo sistema geral de base de dados, serviços de email, ou qualquer outro tipo de componente necessário para a aplicação são executados em máquinas virtuais. Esses serviços são geridos através da ferramenta BOSH.

BOSH é uma ferramente *open source* para fazer a gestão do ciclo de vida, *deployment* e monitorização de sistemas distribuídos. Um desses sistemas é o próprio Cloud Foundry.

O Cloud Foundry suporta aplicações desenvolvidas nas mais diversas linguagens de programação, *frameworks* e tecnologias, como por exemplo: Go [87], Node.js [88], .NET [89], Ruby [90] etc.

Quando é feito o envio (*push*) da aplicação para a plataforma Cloud Foundry, este deteta automaticamente que *buildpack* deve ser aplicado.

O *buildpack* é uma combinação de *scripts* para instalar todas as dependências necessárias para a execução da aplicação. Opções adicionais de *deployment* da aplicação podem ser adicionadas através de um *manifest file*. Como resultado deste processo obtêm-se um *droplet* que é um ficheiro que contém a aplicação e as suas dependências pronta para ser executada.

Atualmente existem duas tecnologias associadas ao *Cloud Foundry*, o *Droplet Execution Agent* (DEA) e a mais recente arquitetura *Diego*, que pretende substituir o DEA.

A escalabilidade automática da aplicação é possível, porém, é necessário a associação de um serviço de *auto-scaling* que está disponível apenas na versão comercial. A escalabilidade manual pode ser feita até ao máximo de recursos que não ultrapassem a *quota* atribuída.

Para garantir a alta disponibilidade o *Cloud Foundry* utiliza o HM9000 (*Health Manager*), que é responsável por detetar através do sistema de mensagens *NATS* o estado atual do

droplet. Se existir alguma inconformidade, é enviado um pedido ao componente *cloud controller* para destruir a instância que está a ser executada e iniciar uma nova.

A documentação oficial oferece suporte para a instalação do *Cloud Foundry* nas seguintes plataformas:

1. *Amazon Web Services*
2. *Microsoft Azure*
3. *OpenStack*
4. *vSphere*
5. *vCloud Air or vCloud Director*

4.1.4 *Tecnologias IaaS*

Devido ao uso do *Cloud Foundry* na camada de PaaS, o número total de escolhas para a tecnologia de IaaS fica reduzido a 5 (cinco) opções.

A *Amazon Web Services* tal como o *Microsoft Azure* são fornecedores públicos, neste contexto, e sendo um dos requisitos a instalação de uma cloud privada *on premise*, com total controlo sobre a infraestrutura, estas opções são descartadas.

As opções da VMWare (vSphere, VCLoud Air e vCloud Director) devido ao seu elevado custo interferem com as restrições financeiras do projeto, desse modo não são opções para a escolha final.

O *OpenStack* torna-se assim a opção adequada, não apenas pela exclusão de partes descrita, mas também por ser uma tecnologia que tem tido uma grande adoção e crescimento a nível de maturidade ao longo dos anos recentes. De notar que há relatos de existirem opções (Ex: *CloudStack*) em que foi verificado o *deploy* com sucesso do Cloud Foundry, porém, o fator relacionado com o suporte oficial foi tido em consideração, e desta forma essas opções foram descartadas.

OpenStack

O *OpenStack* é uma tecnologia *open source* para a criação de *clouds* publicas e privadas. Esta opera na camada IaaS, que tem como finalidade controlar um conjunto de recursos de computação, *storage* e rede de um *Data Center* (DC). Os administradores têm assim o controlo da gestão dos recursos através de uma *Application Programming Interface* (API), que pode ser acedida por uma *Command Line Interface* (CLI) ou através de uma *dashboard*.

A plataforma *OpenStack* é composta por vários serviços que são identificados e descritos a seguir.

SERVIÇOS NUCLEARES DO OPENSTACK

Nova gere os recursos computacionais das instâncias que estão a correr na infraestrutura. Este componente utiliza a API *libvirt* para interagir com os diversos *hypervisors*, uma vez que não possui capacidade própria de virtualização.

Swift é um sistema de armazenamento distribuído para o armazenamento de objetos. Dada a especificidade da tecnologia, esta, tem a capacidade de lidar com o armazenamento de elevadas quantidades de objetos, objetos de grande tamanho e fornecer redundância de dados.

Keystone é o componente responsável pelo serviço de autenticação e autorização para os serviços do *OpenStack*. Este possui o conhecimento de todos os *endpoints* dos serviços do OpenStack.

Neutron é responsável pela a componente de rede do *OpenStack*. Fornece uma API que permite que os utilizadores criem redes, as atribuam a determinados projetos etc.

Cinder é um serviço de armazenamento persistente de dados que utiliza o tipo de armazenamento de dados *Block Storage*. O *Cinder* disponibiliza recursos de armazenamento ao serviço *Nova*.

Glance é responsável por gerir as imagens das VMs. Estas são utilizadas pela componente de computação do *OpenStack* para criar novas instâncias.

4.2 DIAGRAMA DA DELIVERY PIPELINE

Após serem apresentadas as escolhas tecnológicas é exposta a proposta da *delivery pipeline* para o projeto através da Figura 13.

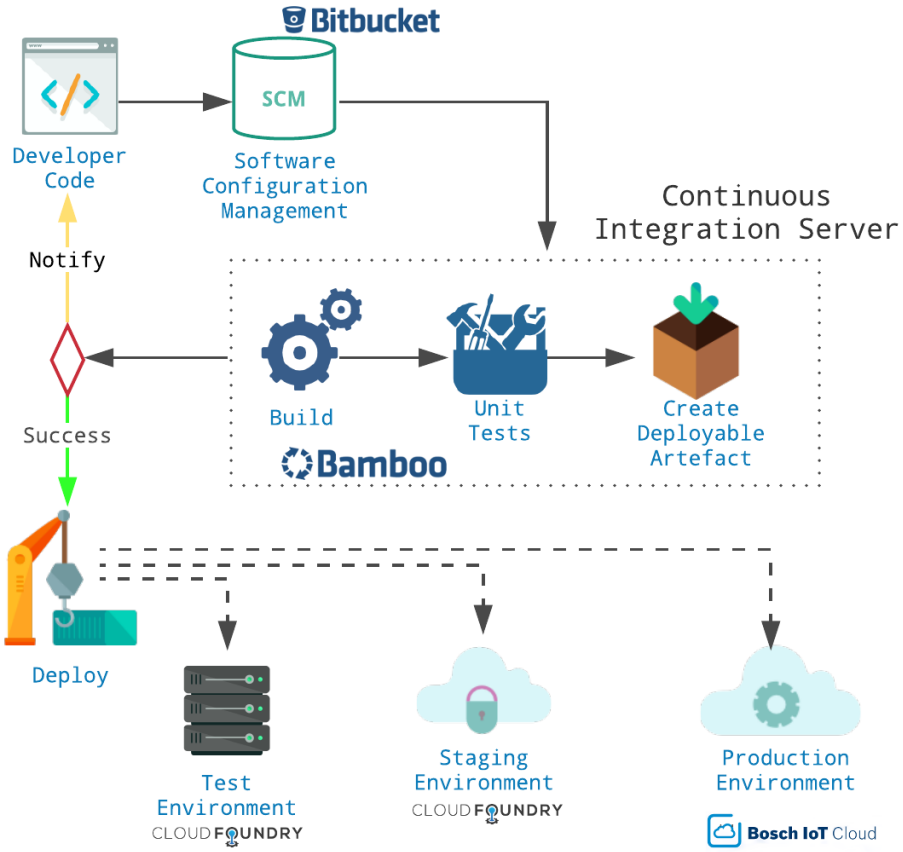


Figura 13: Delivery Pipeline do projeto

Quando o programador efetua um *commit* do código para o repositório Git que é suportado pela plataforma Bitbucket, é lançado um evento automático que informa a instância do Bamboo que compila o código e executa os testes definidos. Quando terminadas as tarefas o programador é informado sobre o resultado dessas tarefas. Em caso de sucesso, o programador pode escolher em que ambiente quer efetuar o *deploy* do artefacto gerado. Neste caso particular todos os *deploys* são efetuados para a plataforma Cloud Foundry. No caso particular de *staging* e *production* não é utilizada a opção de auto *deploy* disponibilizada pelo Bamboo, uma vez que existe um maior controlo e necessidade de mais permissões para efetuar alterações nos ambientes em questão. Essa tarefa é portanto executada por uma pessoa responsável para o efeito.

4.3 ARQUITETURA DA INFRAESTRUTURA

Numa primeira fase identificaram-se os serviços do OpenStack necessários para o *setup* da infraestrutura. Esses serviços foram agrupados em 3 (três) componentes lógicos: Componente de rede, computação e controlo, como exposto na Figura 14.

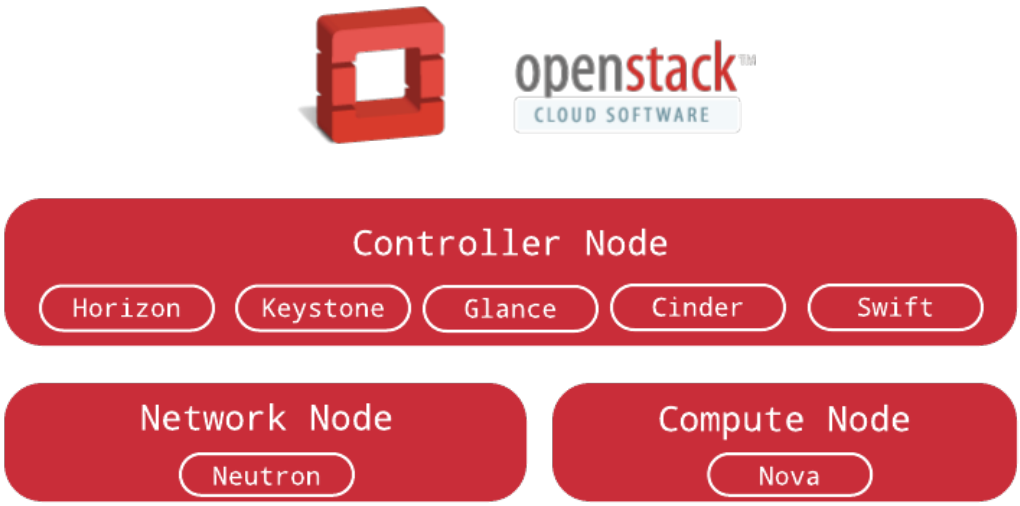


Figura 14: Componentes Lógicos do OpenStack

Controlo O componente de controlo corresponde à API dos serviços, web interface de gestão, base de dados e sistema de mensagens. Os serviços *horizon*, *keystone*, *glance*, *cinder*, *swift* compõem este componente.

Rede O componente de rede, é responsável pela execução dos serviços de rede. Esta componente é constituída pelo serviço *neutron*.

Computação O componente de computação possui os serviços necessários para a execução de VMs. O serviço *nova* faz parte desta componente.

Através da Figura 15 pode-se observar a infraestrutura lógica que dá suporte ao projeto. Esta é composta por 2 (dois) componentes de computação, 1 (um) componente de controlo e 1 (um) componente de rede.

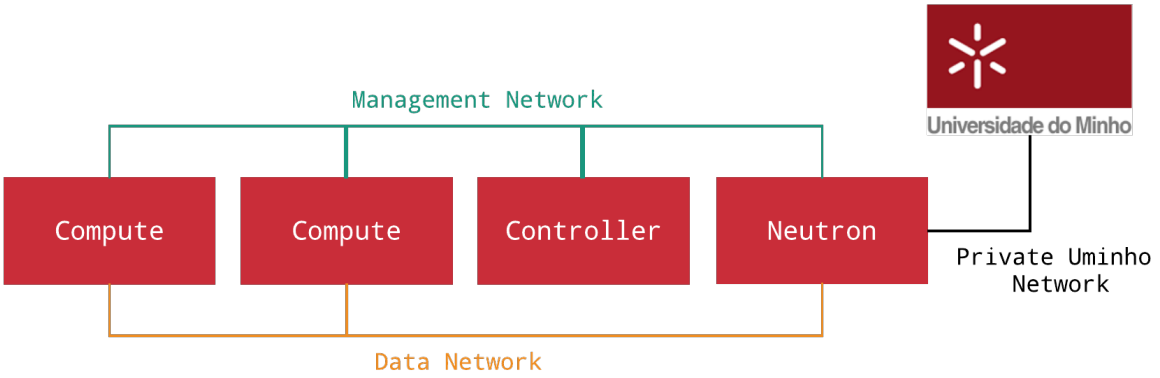


Figura 15: Arquitetura Lógica da Infraestrutura

A conceção arquitetural lógica deve ser independente da implementação física, porém, esta é influenciada quer pelos requisitos não funcionais, quer pelas restrições impostas no projeto.

A próxima etapa consiste em efetuar o mapeamento da arquitetura lógica com a arquitetura física. Os recursos computacionais para o efeito estão listados na Tabela 1.

Tabela 1: Dispositivos da infraestrutura

Nome	**Quantidade**	**Tipo de Dispositivo**	**Descrição**
Node01/02	2	Servidor	16 cores, 48 GB RAM, 900 GB Storage.
Node00	1	Servidor	16 cores, 24 GB RAM, 900 GB Storage.
Gateway	1	Servidor	8 cores, 16 GB RAM, 200 GB Storage.
Switch	1	*Switch*	18 portas 1Gb, com suporte para VLans.

Após identificados os dispositivos disponíveis, procede-se ao mapeamento entre os componentes lógicos e dispositivos físicos como se verifica na Tabela 2.

Tabela 2: Mapemaneto d dos componentes lógicos com dispositivos físicos

Componente Lógico	**Componente Físico**
Computação	node01 e node02
Rede	node00
Controlo	node00

Os servidores *node01/02* funcionam exclusivamente com a componente de computação do OpenStack. O servidor *node00* executa os serviços que fazem parte dos componentes de controlo e rede. Opta-se por agregar estes dois componentes num único servidor, uma vez que a maior parte do peso computacional vai estar centrado nos nodos de computação. Idealmente e sem as restrições no número de servidores físicos presentes, esses componentes estariam em dispositivos distintos.

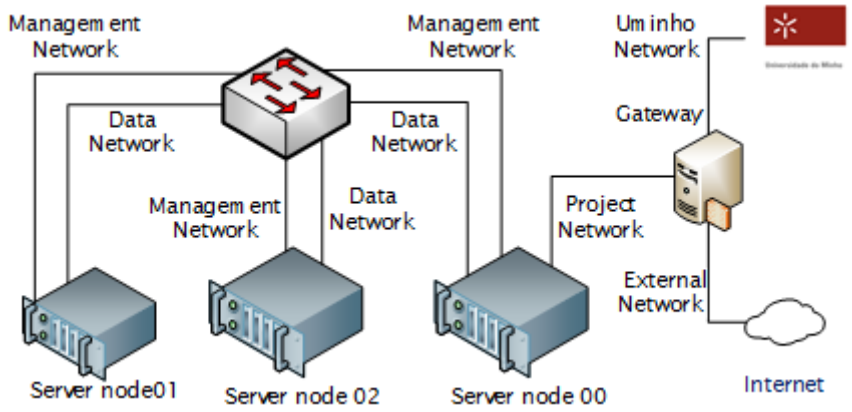


Figura 16: Arquitetura Física da Infraestrutura

ARQUITETURA FÍSICA Como se verifica pela Figura 16, o *Gateway* atua como ponto de acesso à rede interna do projeto. A sua ligação pode ser feita através de dois canais distintos uma vez que possui um IP público, e um IP da rede interna da UM que também tem acesso para a Internet. As suas principais funções consistem em atuar como *router* e *firewall*, fornecendo uma camada de segurança adicional ao fazer a separação da rede pública da rede do projeto.

Para mitigar a ocorrência de falhas (físicas) do sistema foram adotados alguns mecanismos de *fail-over*, entre os quais:

- Ligação de todos os dispositivos a uma UPS, para garantir redundância a nível de energia.
- Todos os servidores possuem dois discos a atuar em modo de RAID 1, isto significa que o sistema tolera a falha de um disco por servidor sem *downtime* dos serviços.

DESCRIÇÃO DAS REDES O facto da infraestrutura do projeto coabitar com outros projetos dentro do DC da rede da UM, carece de cuidados adicionais relativamente ao endereçamento dado aos dispositivos.

Através da Figura 16 verifica-se que todos os dispositivos computacionais estão interligados por um *switch*, por meio de uma ou mais redes.

Para diminuir número de colisões de datagramas no *switch*, são criadas 2 (duas) VLans (Tabela 3) com base na sub-rede IP. Com esta ação é aumentada a segurança e um maior desempenho na rede.

Tabela 3: VLans baseadas na Sub-Rede

Nome da Rede	**VLan**
Management Network	20
Data Network	30
Restantes Redes	1

DESCRIÇÃO DAS REDES

Project Network Os IPs desta rede são fornecidos pelo servidor *Dynamic Host Configuration Protocol* (DHCP), são endereço de atribuição dinâmica. Os IPs públicos atribuídos às VMs no OpenStack pertencem a esta rede. O *Gateway* encaminha o tráfego da rede UM para esta rede. Ou seja dispositivos dentro da rede interna da UM conseguem aceder diretamente a esta rede.

Management Network Todos os nodos de que têm serviços do OpenStack instalados pertencem a esta rede. Esta rede é utilizada pelo OpenStack para gestão interna. Esta rede não é exposta ao exterior, porém todos os pedidos do exterior passam por esta rede para serem posteriormente encaminhados para os serviços pretendidos.

Data Network A Rede de *Data Network* é a rede utilizada para a comunicação entre os nodos de computação. Apenas os nodos com o serviço *nova* ou *neutron* instalados estão interligados por esta rede. Esta rede surge da necessidade de evitar que as operações dos nodos de computação sejam afetadas pelo congestionamento da *Management Network*.

External Network É a rede utilizada para os utilizadores acederem aos serviços e aplicações através da Internet.

Uminho network É a rede que permite o acesso aos recursos através da rede interna da UM.

4.4 GATEWAY - SERVIÇOS

Até se alcançar uma versão com uma configuração satisfatória da instalação do *OpenStack* é conveniente realizar várias iterações, nomeadamente através da instalação de várias versões do *OpenStack* com diferentes configurações. Devido ao facto de alguns processos se repetirem várias vezes, automatizar esses mesmos processos leva a uma maior produtividade na realização do trabalho.

O método *Preboot Execution Environment* (PXE) permite a automatização do processo de instalação do SO nas máquinas físicas. A descrição e configuração descritas têm em consideração a utilização o SO Linux por meio da distribuição CentOS 7.

4.4.1 *Serviço PXE*

O serviço de PXE permite que um computador cliente faça *boot*, execute ou instale um SO através de uma interface de rede. Este método é normalmente utilizado quando se pretende instalar um SO em vários computadores.

A instalação de um serviço PXE envolve a utilização de 2 (dois) protocolos essenciais: o *Trivial File Transfer Protocol* (TFTP) e DHCP.

Principais passos para a configuração do serviço PXE:

1. Configurar o servidor de ficheiros.
2. Configurar os ficheiros no servidor TFTP necessários para arranque do PXE.
3. Iniciar o serviço de TFTP
4. Configurar o serviço DHCP, nomeadamente o intervalo de IPs disponíveis para atribuição e qual o ficheiro que deve ser utilizado para o *boot* do SO.
5. Iniciar o serviço de DHCP
6. Iniciar o dispositivo cliente e efetuar a instalação do SO.

4.4.2 *Servidor TFTP*

O protocolo TFTP [91] é o protocolo padrão utilizado para o *boostrap* de SOs através de rede. Este protocolo é identificado como ajustado para esta tarefa uma vez que é baseado no protocolo padrão da Internet (IP) . Para este trabalho recorre-se ao servidor *vsftp*.

4.4.3 *Servidor DHCP*

O DHCP é um protocolo que atribui automaticamente endereços IP a dispositivos. Inicialmente existe uma *pool* de IPs que posteriormente são disponibilizados assim que um dispositivo efetua um DHCP *Request*. Depois desse pedido é atribuído ao dispositivo um IP assim como outras informações importantes como a mascara da sub-rede, endereço do *Gateway*, servidor *Domain Name Server* (DNS) etc.

O ficheiro de configuração utilizado para a instalação está exposto abaixo:

```
1  ddns-update-style interim;
2  ignore client-updates;
3  authoritative;
4  allow booting;
5  allow bootp;
6  allow unknown-clients;
7
8  subnet 192.168.187.128 netmask 255.255.255.128 {
9    range 192.168.187.129 192.168.187.253;
10   option domain-name-servers 192.168.187.2;
11   option domain-name "cloud.ia";
12   option routers 192.168.187.2;
13   default-lease-time 600;
14   max-lease-time 7200;
15
16   next-server 192.168.187.2; #  DHCP Server IP
17   filename "pxelinux.0";
18
19   # Fixed IPs for known machines.
20   host node00 { hardware ethernet 08:00:27:19:**:**; fixed-address
         192.168.187.10; next-server 192.168.187.2; filename "pxelinux.0"; }
21   host node01 { hardware ethernet 08:00:27:59:**:**; fixed-address
         192.168.187.11; next-server 192.168.187.2; filename "pxelinux.0"; }
22   host node02 { hardware ethernet 08:00:27:96:**:**; fixed-address
         192.168.187.12; next-server 192.168.187.2 ; filename "pxelinux.0"; }
23 }
```

Listing 4.1: Configuração do servidor DHCP

O serviço de DHCP fornece IPs na rede 192.168.187.0/24 no intervalo de IPs compreendido entre 192.168.187.129 e 192.168.187.249, a todas as máquinas físicas da infraestrutura do projeto, mas é também utilizado para atribuir IPs da rede interna da UM às VMs instanciadas no *OpenStack*. Quando o cliente efetua o DHCP *request* ao servidor, este envia também o endereço do serviço DNS que se encontra instalado no mesmo servidor (192.168.187.2).

Na parte final da configuração é feita a atribuição de IP por afinidade ao *MAC address* das interfaces de rede. A utilidade em efetuar este tipo de configuração prende-se com o facto de no futuro ser necessário efetuar configurações personalizadas. Como cada dispositivo pode correr diferentes serviços, sabe-se à *priori* qual é o IP que corresponde a determinado dispositivo.

4.4.4 *Kickstart*

O *Kickstart* fornece um mecanismo de configuração personalizada utilizada na instalação do SO. Para evitar a configuração manual de todas as máquinas físicas, foi criado um ficheiro com as configurações base e comuns a todos os servidores físicos.

A configuração em questão pode ser consultada abaixo:

```
# System language
lang en_US

# Keyboard layouts
keyboard pt-latin1

# System timezone
timezone Europe/Lisbon --isUtc

# Network information
network  --bootproto=dhcp --device=eno1 --ipv6=off

# Root password
rootpw $1$pIHCtlQ3$d3oXABKXGwXLmyrIn82vR/ --iscrypted

#platform x86, AMD64, or Intel EM64T
reboot
text
url --url=ftp://192.168.187.2/pub
bootloader --location=mbr --append="rhgb quiet crashkernel=auto"

# Partition
# System bootloader configuration
bootloader --location=mbr --boot-drive=sda

# Partition clearing information
clearpart --initlabel --drives=sda --all

# Disk sda1 partitioning information
part /boot --fstype="xfs" --ondisk=sda --size=500

# Create sda2 LVM partition
part pv.155 --fstype="lvmpv" --ondisk=sda --size=74503
volgroup centos --pesize=4096 pv.155
logvol /home  --fstype="ext4" --size=20480 --name=home --vgname=centos
logvol swap  --fstype="swap" --size=2816 --name=swap --vgname=centos
logvol /  --fstype="xfs" --grow --maxsize=51200 --size=1024 --name=root --vgname=
    centos

%post
```

```
40 # Create sda3 partition
41 parted -a optimal /dev/sda mkpart primary 80GB 100%
42 %end
43
44 auth --passalgo=sha512 --useshadow
45 selinux --enforcing
46 firewall --enabled --ssh
47 skipx
48 firstboot --disable
49
50 # Packages to be installed
51 %packages
52 @base
53 @core
54 %end
```

Listing 4.2: Ficheiro de configuração do Kickstart

O ficheiro *kickstart.conf* especifica a informação necessária para a instalação do SO tal como: o idioma, o *layout* do teclado, o fuso horário, quais os pacotes de software a serem instalados, partições do disco etc.

Os sistemas de ficheiros são normalmente criados numa partição, quando todo o espaço do disco físico já se encontra particionado, neste tipo de situação, criar novas partições pode envolver um conjunto de procedimentos que tornam a tarefa complicada de realizar. Neste sentido, opta-se por efetuar o particionamento do disco no momento em que é efetuada a instalação do SO através do *kickstart*. Além das partições necessárias para a instalação do SO é criada uma partição adicional como podemos verificar através da Figura 17 (Linha 26 até à linha 37 do ficheiro de configuração do Kickstart). Esta partição adicional é utilizada no processo de instalação dos serviços do *OpenStack*, e é explicada em maior detalhe no procedimento de instalação do *OpenStack*.

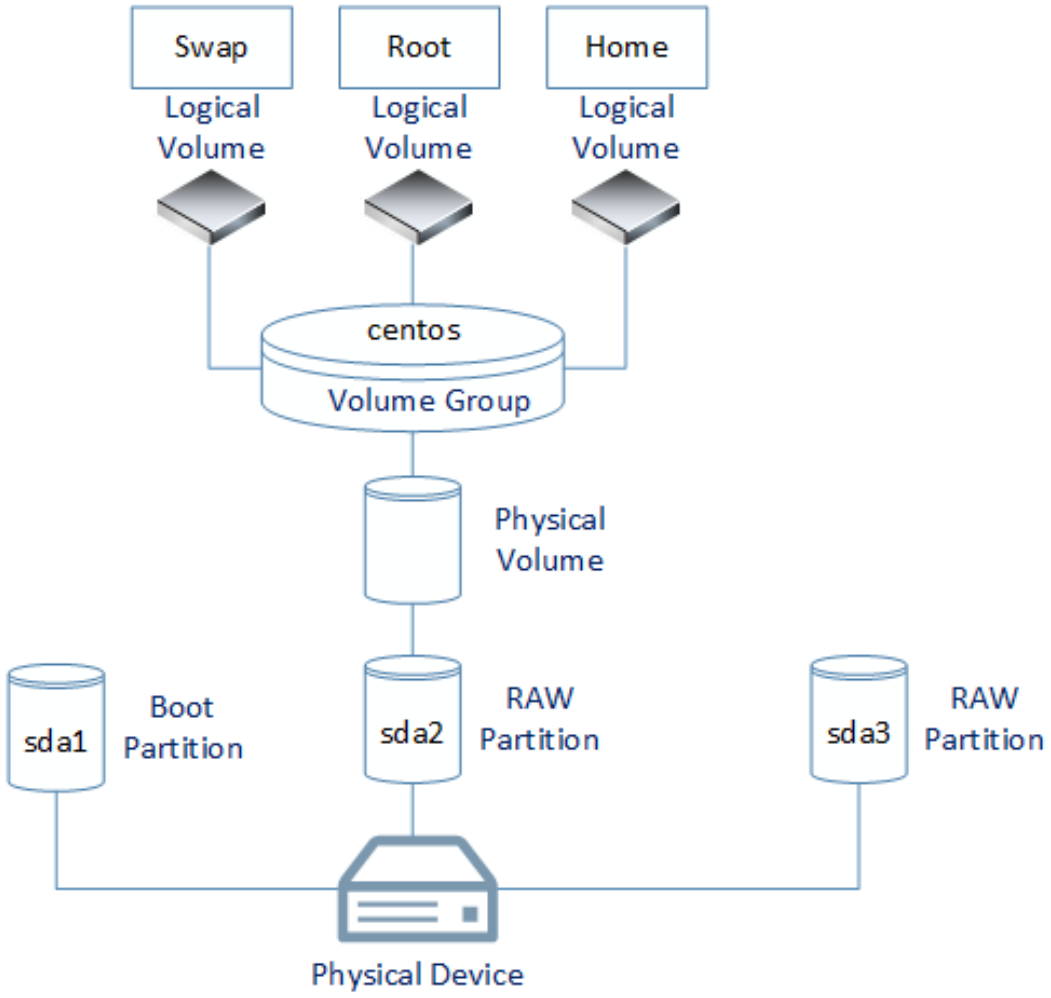


Figura 17: Estrutura de partições de ficheiros genérica

SDA1 A *boot partition* é a partição que contem os ficheiros necessários para o arranque do SO. Essa partição é montada em */boot/*.

SDA2 Utiliza uma partição *Logical Volume Manager* (LVM), em que agrupa os volumes lógicos:

Swap Partição utilizada para guarda dados de memória em disco.

Root O *Logical Volume Root* é montado na raiz do sistema "/"e é nessa partição que está instalado o sistema operativo e as aplicações.

Home O *Logical Volume Home* é montado em */home* e contém dados de todos os utilizadores do sistema operativo (à exceção do utilizador *root*).

SDA3 Esta partição é reservada para guardar dados dos serviços que vão ser utilizados no servidor em questão. Esta é criada com o objetivo de obter um melhor desempenho, organização e otimização de alguns componentes do *OpenStack*.

Para garantir uma maior segurança é mantido o *SELinux* e a *Firewall* ativos, sendo que neste último a porta 22 que corresponde ao serviço de *Secure Shell* (SSH) é aberta para permitir a ligação remota à máquina para efetuar configurações, quer manuais, quer através da ferramenta de gestão de configurações Ansible.

Por fim, para este mecanismo automático ser invocado é necessário especificar qual o ficheiro *kickstart* a ser utilizado no *Boot Menu (/var/lib/tftpboot/pxelinux.cfg/default)*

Para tal adiciona-se um argumento no arranque da imagem de instalação com o argumento KS e localização do ficheiro *kickstart*.
Exemplo: ks=ftp://192.168.186.2/pub/kickstart.conf

```
1 menu title ########## PXE Boot Menu ##########
2 label 1
3 menu label ^1) Install CentOS 7 x64 with Local Repo - Kickstart automated process
4 kernel centos7/vmlinuz
5 append initrd=centos7/initrd.img method=ftp://192.168.187.2/pub network ks=ftp
    ://192.168.187.2/pub/kickstart.conf devfs=nomount
```

Listing 4.3: Configuração do Boot Menu

4.5 INSTALAÇÃO OPENSTACK

A descrição presente neste processo de instalação refere-se à mais recente versão do *OpenStack*, a versão *Newton*, instalada dos repositórios oficiais da distribuição *CentOS 7*, que são mantidos pela comunidade RDO (projeto patrocinado pela RedHat).

Face à experiência adquirida ao longo deste projeto, verifica-se que o *OpenStack* é uma plataforma com um desenvolvimento muito veloz, sendo notórios os avanços no crescimento da maturidade dos serviços já existentes, mas também com a apresentação de novos serviços à medida que novas versões aparecem. Sendo estes os motivos que preponderantes pela opção da última versão do *OpenStack*.

A instalação do OpenStack pode ser complexa caso se opte por efetuar a instalação componente a componente, porém existem outros mecanismos que facilitam a instalação através de *scripts* automatizados. O RDO Disponibiliza dois métodos:

TRIPLEO Este método consiste em utilizar uma instância do *OpenStack* denominada de *undercloud* para efetuar a instalação e gestão da solução *OpenStack* final, denominada de *overcloud*. Este método é mais apropriado quando se pretende uma solução final para um ambiente de produção, com alta disponibilidade a nível de hardware e serviços.

PACKSTACK Packstack é uma ferramenta que recorre a módulos do Puppet (Ferramenta de gestão de configurações automáticas) para fazer *deploy* dos vários componentes do OpenStack via SSH, em servidores com um SO previamente instalado. Este método é adequado quando se pretende ter uma instância do OpenStack a correr em apenas um servidor ou para criar uma *cloud* para prova de conceito. Este método permite a adição de novos componentes ou adicionar instâncias de serviços na infraestrutura à *posteriori*.

Proceder à instalação manual além de complexo e demorado, é bastante suscetível a erros. Dessa forma é uma abordagem que não é colocada em equação. Face aos dois métodos disponibilizados pelo RDO, a abordagem Packstack mostra-se a mais adequada devido ao reduzido número de servidores. Esta abordagem não requer dispositivos adicionais, como se verifica na abordagem TripleO, onde é necessário um servidor dedicado para o *undercloud*.

Como apresentado na Figura 16, existem 3 servidores que dão suporte à execução dos serviços do *OpenStack*. Opta-se por instalar os componentes controlo e rede num servidor comum (Node00), e efetuar a instalação do componente de computação em dois nodos de computação (Node01 e Node02). Os dois nodos de computação escolhidos para essa instalação referem-se aos servidores com maior capacidade computacional. É essencial maximizar os recursos dedicado à componente de computação, uma vez que a instalação do Cloud Foundry possui requisitos elevados quer a nível de memória RAM quer a nível de número de cores de CPU.

4.5.1 *Configuração Packstack*

Para fazer uma instalação do Packstack personalizada é necessário gerar um ficheiro com as configurações executando a instrução:

```
1 packstack --gen-answer-file=openstack.conf
```

Listing 4.4: Gerar ficheiro de configuração do packstack

O ficheiro *openstack.conf* contem uma série de opções que permitem a personalização da instalação a efetuar. Algumas das principais alterações das configurações originais são apresentadas em baixo:

```
1 [general]
```

```
2
3 # Installed OpenStack Services
4 CONFIG_GLANCE_INSTALL=y
5 CONFIG_CINDER_INSTALL=y
6 CONFIG_NOVA_INSTALL=y
7 CONFIG_NEUTRON_INSTALL=y
8 CONFIG_HORIZON_INSTALL=y
9 CONFIG_SWIFT_INSTALL=y
10
11 # Hosts where the OpenStack Components are installed
12 CONFIG_CONTROLLER_HOST=192.168.186.33
13 CONFIG_NETWORK_HOSTS=192.168.186.33
14 CONFIG_COMPUTE_HOSTS=192.168.186.34,192.168.186.35
15
16 # Install and configure the nagios in all nodes
17 CONFIG_NAGIOS_INSTALL=y
18
19 # Cinder Volume Configuration is made after OpenStack deploy
20 CONFIG_CINDER_VOLUMES_CREATE=n
21
22 # Virtualization driver
23 CONFIG_NOVA_LIBVIRT_VIRT_TYPE=kvm
24
25 # Network Iterface for "Data Network" for node computes
26 CONFIG_NOVA_COMPUTE_PRIVIF=eno3.30
27
28 # Network Interface for "Management Network" in nova computes
29 CONFIG_NOVA_NETWORK_PUBIF=eno2.20
30
31 # Public IP needs to be assign manualy to the instances created in OpenStack.
32 CONFIG_NOVA_NETWORK_AUTOASSIGNFLOATINGIP=n
33
34 # Network Interface which connects OpenStack to Internet
35 CONFIG_NEUTRON_L3_EXT_BRIDGE=br-ex
```

Listing 4.5: Configurações parciais do Packstack

4.5.2 *Armazenamento de dados*

Existem dois tipos de estados de armazenamento de dados no OpenStack: armazenamento persistente e armazenamento efémero .

ARMAZENAMENTO EFÉMERO é parte integrante de uma instância de computação. O tempo de vida deste tipo de *armazenamento* é intrínseca ao tempo de vida dessa mesma

instância. Nesta instalação o armazenamento efémero de cada VM é feito no mesmo nodo de computação onde a instância está a ser executada.

ARMAZENAMENTO PERSISTENTE é um tipo de armazenamento que é independente da duração do tempo de vida das instâncias de computação, ou seja, é mantida até que o utilizador decida apaga-la.

Através do serviço de *block storage cinder*, os utilizadores podem criar volumes persistentes que podem ser conectados/desconectados de instâncias de computação.

O *OpenStack* não necessita de configurações adicionais depois de instalado. Porém, aconselha-se que sejam criadas partições adicionais para os diferentes tipos de armazenamento do OpenStack. Por omissão, através do processo de instalação *packstack*, as VMs e volumes são guardados na partição principal do SO. Com a criação destes, a partição pode encher muito rapidamente sendo o resultado dessa situação inesperado, podendo ocorrer falhas no sistema, perda de dados etc.

No contexto desta instalação a partição *sda3* é utilizada de duas formas no componente controlador, onde se encontra o *cinder* instalado, e no componente de computação.

NODOS DE COMPUTAÇÃO Por omissão, o *nova* guarda as instâncias criadas em: */var/lib/nova/*, que por norma é uma diretoria que pertence ao volume lógico onde está instalado o SO. Por questões de otimização, organização e pelos problemas que podem ocorrer já mencionadas nesta secção, opta-se por criar um volume lógico na partição *sda3* que será montado em: */var/lib/nova/*.

Podemos visualizar a estrutura das partições através da Figura 18.

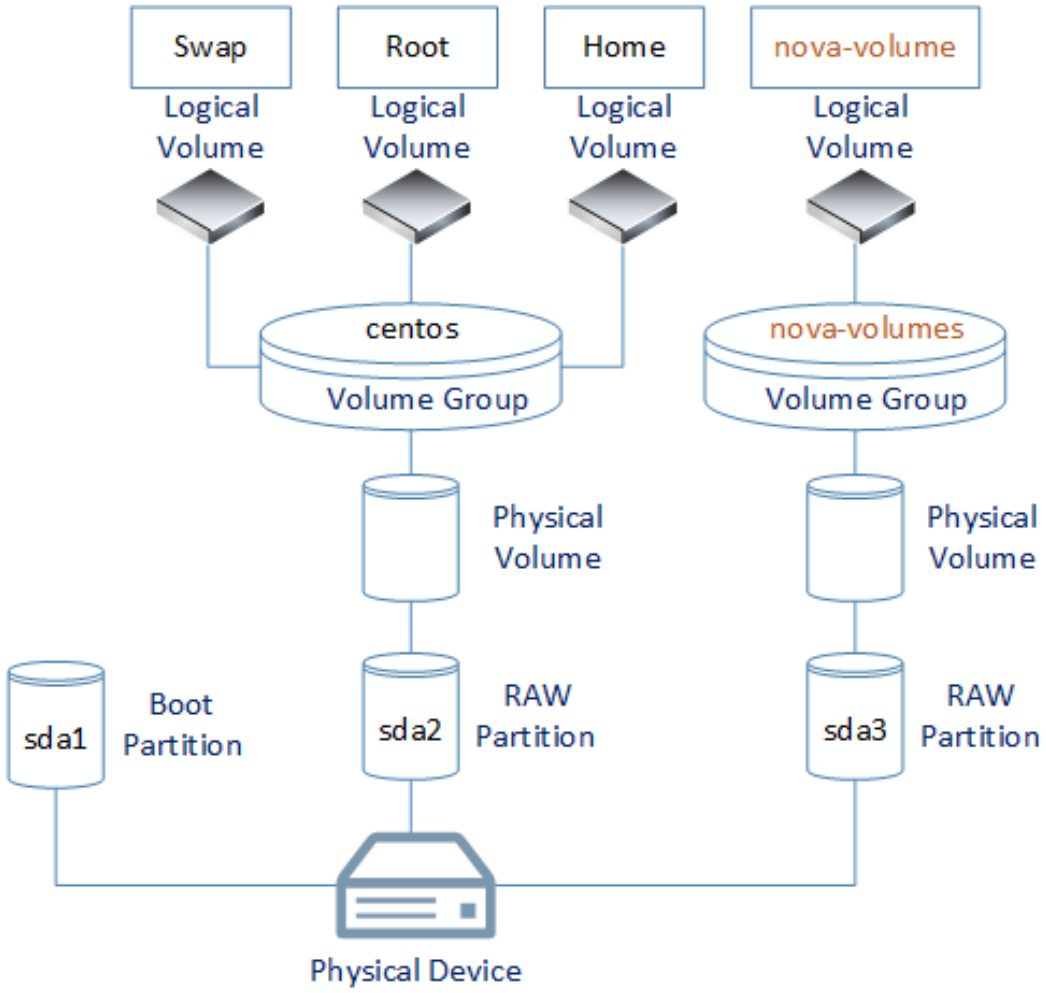


Figura 18: Estrutura de partição dos nodos de computação

Para obter a estrutura da partição *sda3* exposta na Figura 18 é necessário:

1. Criar o volume físico.
2. Criar o grupo lógico.
3. Criar o volume lógico.
4. Criar o sistema de ficheiros.
5. Efetuar o *auto-mount* da partição no arranque do SO, para que não seja necessário efetuar o *mount* da partição todas as vezes que o SO é reiniciado.
6. *Umount* da partição em */var/lib/nova*.
7. *Mount* do *Logical Volume* em */var/lib/nova*.

CINDER NODE O *cinder* quando instalado utiliza um grupo lógico através de um ficheiro (*loop device*). Porém este método não é eficiente, nem aconselhado a utilizar em ambientes de produção. Por conseguinte é criado grupo lógico com o nome ”cinder-volumes” na partição *sda3* do node00, servidor onde o serviço *cinder* está a ser executado.

Podemos visualizar a estrutura das partições que se pretende configurar através da Figura 19.

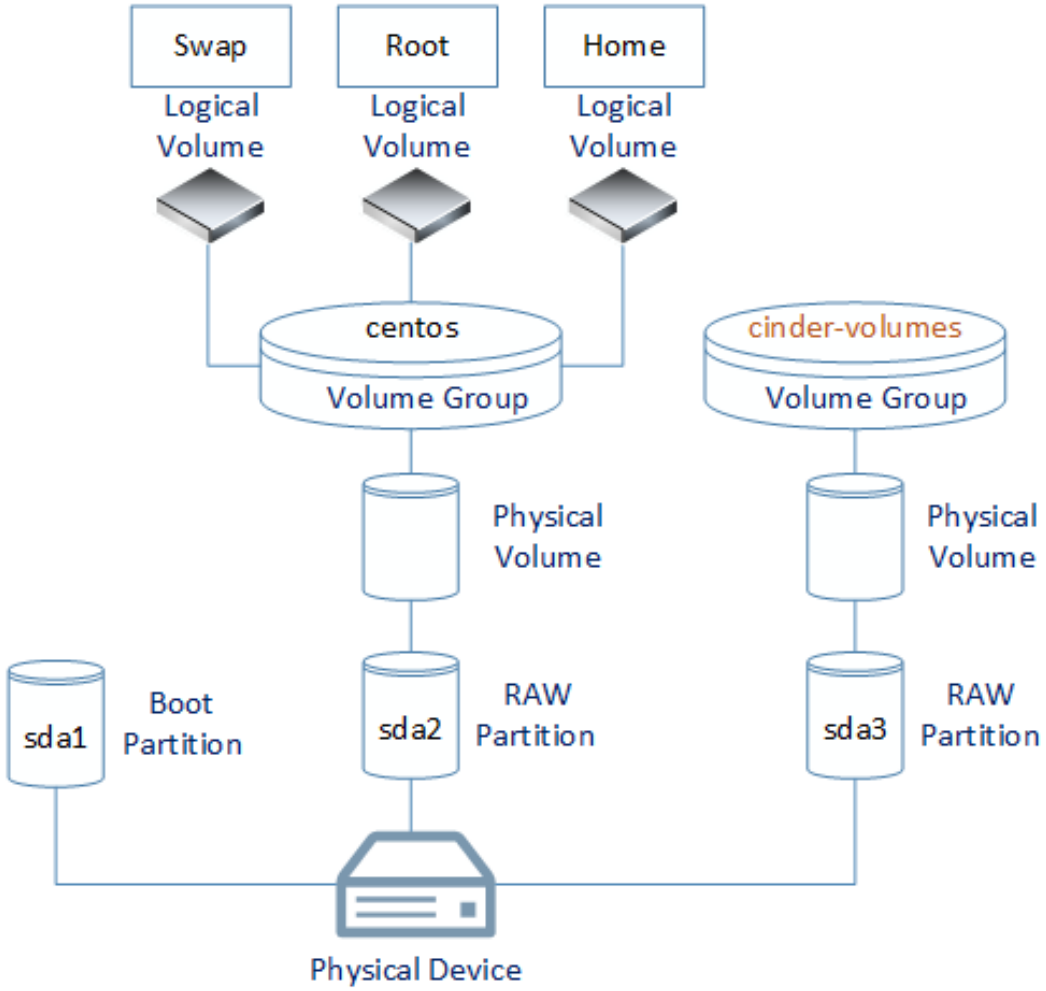


Figura 19: Estrutura de partições no nodo00

De seguida é necessário configurar o *cinder* para utilizar o volume criado: */etc/cinder/cinder.conf*

```
1 [lvm]
2 iscsi_helper=lioadm
3 iscsi_ip_address=192.168.186.33
4 volume_driver=cinder.volume.drivers.lvm.LVMVolumeDriver
5 volume_group=cinder-volumes
```

```
6 volume_backend_name=lvm
```

Listing 4.6: Configuração Cinder

Após a realização das configurações reinicia-se o serviço *openstack-cinder-volumes*.

4.6 INSTALAÇÃO CLOUD FOUNDRY

4.6.1 *Introdução*

A instalação do Cloud Foundry permite a homogeneidade do ambiente de pré-produção com o ambiente de produção (BIC), que é o culminar do principal objetivo deste trabalho.

Através desta instalação os programadores da equipa têm uma maior facilidade e autonomia em fazer *deploy* das suas aplicações, e a possibilidade de testar os serviços que estão a desenvolver.

A versão utilizada nesta instalação é a disponibilizada pela *Cloud Foundry Foundation*, a versão *open source* do Cloud Foundry.

Alguns dos procedimentos necessários para a instalação do Cloud Foundry estão relacionados com configurações do OpenStack, alguns destes são efetuados pela *dashboard* do OpenStack, e em casos mais específicos opta-se por utilizar a CLI de forma a facilitar a criação de *scripts* para a automatização de tarefas, numa eventual necessidade de repetir o processo.

Podem-se estabelecer 3 (três) etapas para a instalação do Cloud Foundry:

1. Validação da camada de suporte de IaaS (OpenStack).
2. Deploy da ferramenta BOSH.
3. Deploy do Cloud Foundry.

4.6.2 *Configurações e validações*

Apesar de não obrigatório, opta-se por criar um novo projeto no OpenStack para a instalação do Cloud Foundry. Assim é possível efetuar a gestão de utilizadores, segurança e recursos alocados ao Cloud Foundry com uma maior organização.

Criação de grupo se segurança

Os grupos de segurança são um mecanismo para restringir o tráfego das instâncias criadas no OpenStack. Ambas as instalações (BOSH e Cloud Foundry) requerem um grupo de segurança específico no OpenStack.

Para o *deploy* e funcionamento do BOSH é criado o grupo de segurança ”bosh” no OpenStack, em que este necessita de permitir a entrada de tráfego nas portas listadas na Tabela 4.

Tabela 4: Lista de portas do grupo de segurança ”bosh”

Protocolo	Porta	Origem	Descrição
TCP	22	0.0.0.0/0	Acesso SSH à VM do BOSH Director(Requisito da ferramenta *bosh-init*).
TCP	6868	0.0.0.0/0	Porta onde o Agente BOSH está a ser executado (Requisito da ferramenta *bosh-init*).
TCP	25555	0.0.0.0/0	Porta para interação com a API através da CLI.
TCP	1-65535	Grupo bosh	Apenas hosts que também pertençam ao grupo de segurança ”bosh”podem aceder a todas as portas sem restrições, para gestão e acesso a dados.

As portas 22, 6868, 25555 estão abertas para todos os remetentes. As portas 1-65535 apenas estão abertas para os *hosts* que pertençam ao mesmo ao grupo de segurança ”bosh”.

As instruções para a execução da tarefa são realizadas através da CLI no node00:

```
1 nova --os-project-name cloudia secgroup-create bosh "BOSH Security Group"
2 nova --os-project-name cloudia secgroup-add-rule bosh tcp 22 22 0.0.0.0/0
3 nova --os-project-name cloudia secgroup-add-rule bosh tcp 6868 6868 0.0.0.0/0
4 nova --os-project-name cloudia secgroup-add-rule bosh tcp 25555 25555 0.0.0.0/0
5 nova --os-project-name cloudia secgroup-add-group-rule bosh bosh tcp 1 65535
```

Listing 4.7: Criar Grupo de segurança ”bosh”no OpenStack

De seguida, procede-se à criação do grupo de segurança ”cf” com as regras de acesso necessárias para o correto funcionamento do mesmo do Cloud Foundry. O tráfego deve ser permitido nas portas presentes na Tabela 5.

Tabela 5: Lista de portas do grupo de segurança ”cf”

Protocolo	Porta	Origem	Descrição
UDP	68	0.0.0.0/0	Serviço de *Bootstrap Protocol* (BOOTP)/DHCP
UDP	3457	0.0.0.0/0	Serviço de logs *Doppler*.
TCP	22	0.0.0.0/0	Acesso SSH.
TCP	80	0.0.0.0/0	Acesso ao serviço HTTP
TCP	443	0.0.0.0/0	Acesso ao serviço HTTPS
TCP	4443	0.0.0.0/0	Serviço de storage Blobstore (Repositório de ficheiros binários).
TCP	1-65535	Grupo cf	Apenas *hosts* com origem do grupo ”cf”podem aceder a todas as portas sem restrições, para gestão e acesso a dados.

As instruções para a criação do grupo de segurança ”cf”são as seguintes:

```
1 nova --os-project-name cloudia secgroup-create cf "Cloud Foundry Security Group"
2 nova --os-project-name cloudia secgroup-add-rule cf tcp 22 22 0.0.0.0/0
3 nova --os-project-name cloudia secgroup-add-rule cf tcp 80 80 0.0.0.0/0
4 nova --os-project-name cloudia secgroup-add-rule cf tcp 443 443 0.0.0.0/0
5 nova --os-project-name cloudia secgroup-add-rule cf tcp 4443 4443 0.0.0.0/0
6 nova --os-project-name cloudia secgroup-add-rule cf udp 68 68 0.0.0.0/0
7 nova --os-project-name cloudia secgroup-add-rule cf icmp -1 -1 0.0.0.0/0
8 nova --os-project-name cloudia secgroup-add-group-rule cf cf tcp 1 65535
```

Listing 4.8: Criar grupo de segurança ”cf”no OpenStack

Criação de Flavors

Flavors são *templates* com a indicação dos recursos computacionais que as instâncias do OpenStack podem usufruir. Estes *templates* são escolhidos no momento em que é efetuada a instanciação das VMs. A variedade de *flavors* surge dos diferentes requisitos computacionais que cada componente do Cloud Foundry tem. Cada componente do Cloud Foundry é instalado numa VM exclusiva para o efeito. Para esta instalação opta-se por utilizar os valores recomendados para uma instalação padrão do Cloud Foundry.

Tabela 6: Lista de *flavors* necessários para o Cloud Foundry

Nome do Flavor	RAM	Storage	vCPUS	Ephemeral Storage
m1.small	2 GB	10 GB	1	20 GB
m1.medium	4 GB	10 GB	2	40 GB
m1.large	8 GB	10 GB	4	80 GB
m1.xlarge	16 GB	10 GB	8	160 GB

Para a criação dos *flavors* são executadas as seguintes instruções:

```
1 nova flavor-create m1.small  auto 2048  10 1 --ephemeral 20
2 nova flavor-create m1.medium auto 4096  10 2 --ephemeral 40
3 nova flavor-create m1.large  auto 8192  10 4 --ephemeral 80
4 nova flavor-create m1.xlarge auto 16384 10 8 --ephemeral 160
```

Listing 4.9: Criar Flavors no OpenStack

4.6.3 *Instalação do BOSH Director*

O BOSH *Director* é responsável pela gestão de todo o ciclo de vida do Cloud Foundry. Através dos seus componentes tem a capacidade de pedir novas instâncias à camada

de IaaS através da *Cloud Provider Interface* (CPI) quando são necessários novos recursos computacionais, monitorizar a utilização de recursos e disponibilidade dos serviços das instâncias, etc. Quer o BOSH *Director*, quer as instâncias que este cria têm por base uma imagem minimalista baseadas na distribuição Ubuntu ou CentOS. A esta imagem é dada o nome de *Stemcell*. Além dos serviços base esta imagem contempla um conjunto de componentes pré-instalados, a que são acrescidos ficheiros de configurações especificas do BOSH, assim como um agente BOSH.

O *deploy* do BOSH *Director* é efetuado recorrendo à ferramenta *bosh-init*. Esta ferramenta lê os atributos de um ficheiro de configuração (*manifest file*) e faz o *deploy*, consoante as configurações mencionadas.

O *manifest file* contêm informação sobre qual o CPI utilizar, qual a *stemcell* que é utilizada para a instanciação da VM onde o BOSH *Director* vai ser executado, o IP público e privado, o *gateway*, o servidor de DNS e a gama de endereços privados que este pode utilizar para efetuar o pedido de novas instâncias.

4.6.4 *Deploy do Cloud Foundry*

A instalação do Cloud Foundry consiste em duas fases: numa primeira fase é configurado o *manifest file* de instalação do Cloud Foundry, e numa segunda fase a instalação do Cloud Foundry através do BOSH.

O código fonte da versão mais recente do Cloud Foundry pode ser obtido através da instrução:

```
1 git clone https://github.com/cloudfoundry/cf-release.git
```

De seguida é necessário efetuar o download de todos os sub-módulos:

```
1 .cf-release/scripts/update
```

Sub-módulos são projetos externos ao Cloud Foundry, mas dos quais este tem dependências.

O deploy do Cloud Foundry é feito recorrendo ao BOSH *Director*. Numa primeira fase é feita a autenticação no BOSH *Director* através da instrução:

```
1 bosh target https://192.168.187.134
```

De seguida é necessário indicar qual o *manifest file* que contem as configurações de instalação do Cloud Foundry:

```
1 bosh deployment cf-manifest.yml
```

Por fim é efetuado o deploy do Cloud Foundry através das seguintes instruções:

```
1 bosh create release
2 bosh upload release
```

```
3 bosh deploy
```

1. É criada a *release* através da instrução ” *bosh create release*”, que consiste em compilar os vários componentes do Cloud Foundry.
2. É feito o upload da *release* para o BOSH *Director*, para que posteriormente possa fazer o *deploy* do Cloud Foundry.
3. É efetuado a instalação do Cloud Foundry. Esta instalação consiste em criar as instâncias com os vários serviços do Cloud Foundry.

Como se pode verificar pela Figura 20 o estado ”*running*”está presente nas 14 VMs dos componentes do Cloud Foundry.

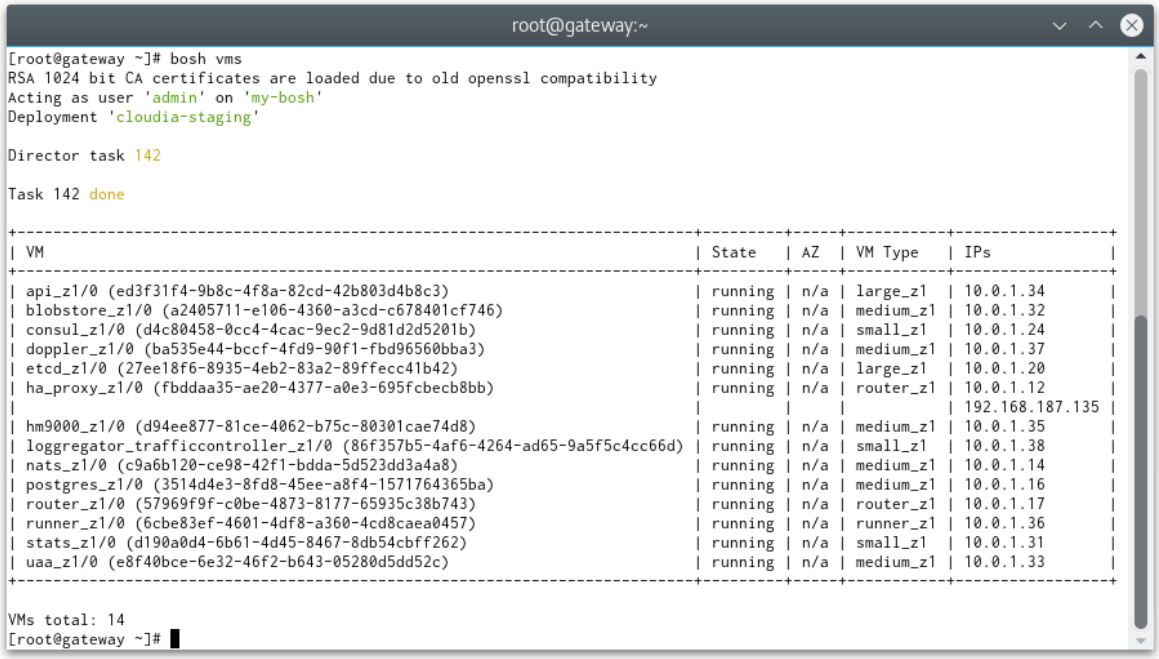


Figura 20: Lista do estado das VMs a executar componentes do Cloud Foundry

No total, para esta instalação do Cloud Foundry foram necessárias:

- 15 Instâncias - 14 (catorze) para o Cloud Foundry e 1 (uma) para o BOSH *Director*
- 39 VCPUs
- 78 GB de RAM
- 5 Volumes *cinder*.
- 135 GB utilizados em armazenamento em armazenamento *cinder*.

4.6.5 *Configuração DNS*

Para finalizar o processo de instalação do Cloud Foundry é necessário adicionar ao servidor de DNS as entradas do domínio referentes às propriedades:

- system_domain: cloudia.uminho.pt
- system_domain_organization: cloudia.uminho.pt
- app_domains: apps.cloudia.uminho.pt

estes domínios devem retornar o endereço onde o *Cloud Controller* do Cloud Foundry que está a ser executado, neste caso particular, no IP 192.168.187.135 como verificado na Figura 20. O *Cloud Controller* fornece pontos de acesso à API REST para que os clientes acedam ao sistema.

As entradas são adicionadas no gestor de DNS público para permitir que os *hosts* que acedem da rede UM não necessitem de alterar as suas configurações relacionadas com o servidor DNS que já utilizam.

Para evitar rutura do serviço numa eventual falha de Internet, os registos também estão na configurações do servidor de DNS privado, que é utilizado pelos dispositivos da infraestrutura criada.

4.7 DEPLOY DA APLICAÇÃO

A aplicação que vai ser executada na infraestrutura para este trabalho, fornece uma API ao exterior com a qual o cliente (utilizador) comunica.

GRUPO DE SEGURANÇA Por omissão o Cloud Foundry aplica um conjunto de regras de segurança às aplicações que estão em execução. Nessas regras não contemplam o contacto com o exterior através da porta do sistema geral de base de dados que é utilizado pela aplicação para guardar os dados. Nesse sentido, foi adicionada uma regra de segurança às aplicações que permite o acesso à porta 27017 (Serviço MongoDB) com o IP interno da VM que está a executar o serviço de base de dados.

Para o efeito é criado o ficheiro *mongo-security.json* com a seguinte informação:

```
1 [
2   {
3     "protocol": "tcp",
4     "destination": "10.20.0.39/32",
5     "ports": "27017",
6     "log": true,
7     "description": "Allow mongodb traffic from CF network."
8   }
```

```
]
```

Listing 4.10: Criar grupo de segurança no Cloud Foundry.

De seguida é criado o grupo de segurança baseada na informação contida no ficheiro json, e posteriormente é feita a associação dessa regra adicional ao grupo de aplicações que estão em execução.

```
cf create-security-group mongodb-internal mongo-security.json
cf bind-running-security-group mongodb-internal
```

MANIFEST FILE DA APLICAÇÃO A criação de um *manifest file* é utilizada quando se pretender ter um ficheiro com todas as configurações de *deploy* da aplicação, evitando assim passar todos parâmetros na CLI sempre que se pretende efetuar um *deploy*.

O *manifest file* utilizado nesta instalação é o seguinte:

```
---
applications:
- name: sfde
  host: sfde
  buildpack: https://github.com/nunodio/cf-maven-buildpack.git
  domain: apps.cloudia.uminho.pt
  memory: 512mb
  disk_quota: 512mb
  instances: 1
  stack: cflinuxfs2
```

Listing 4.11: Manifest file da aplicação.

name Nome atribuído à aplicação

host Nome do *hostname* da aplicação.

buildpack Explicita qual o *buildpack* que é utilizado para o deploy da aplicação.

domain Domínio associado à aplicação. O endereço final da aplicação é a junção dos atributos ”host + domain”.

memory Limite máximo de memória que pode ser utilizada por cada instância de aplicação.

disk_quota Limite máximo do disco utilizado por cada instância de aplicação.

instances Indica o número de instâncias da aplicação. Posteriormente esse número pode aumentar ou diminuir, utilizando a instrução ”cf scale”.

Como se verifica pelo atributo ”buildpack” (Linha 5 do *Manifest file* da aplicação) indica que o *buildpack* se encontra num repositório Git.

Uma vez que a aplicação desenvolvida no projeto tinha características especiais, foi necessário adaptar um *buildpack* ajustado às dependências dessa mesma aplicação. O código fonte encontra-se disponível nesse mesmo link.

A partir deste momento, para disponibilizar a aplicação, o programador apenas necessita de executar a instrução *cf push*.

4.8 SUMÁRIO

A implementação prática deste trabalho resulta num processo muito mais eficaz de *deployment* de aplicações.

Através da Tabela 7 podem-se comparar as várias etapas que ocorriam no método inicial e o método utilizado atualmente, resultante do trabalho elaborado.

Tabela 7: Método de *deploy* de uma aplicação – O antes e o agora.

Método Inicial	**Método Atual**
- Instalação e configurações do SO. - Instalação do *Runtime*/Servidor aplicacional e configuração dos mesmos. - Instalação das dependências. - Deploy da aplicação.	Execução da instrução: cf push

A mudança mais visível é a redução da complexidade na disponibilização de uma aplicação, sendo substituído um conjunto de fases por uma simples instrução ”*cf push*”.

5

CONCLUSÃO E TRABALHO FUTURO

Através das várias iterações e sucessivos testes de toda a componente prática desenvolvida no decorrer deste trabalho, há alguns indicadores positivos a nível do processo e outros que indiciam algumas limitações a nível de infraestrutura.

Através da homogeneização dos ambientes é garantido, não só uma maior compatibilidade do produto de software desenvolvido nos vários ambientes, mas também uma maior facilidade na disponibilização de aplicações. Com a plataforma de PaaS a equipa de desenvolvimento não depende da equipa operacional para executar e testar a aplicação nos diversos ambientes.

Por outro lado, a equipa operacional não necessita de despender tempo a provisionar VMs com determinadas configurações e bibliotecas, evitando problemas relacionadas com o *deployment* do software, configurações do sistema operativo e serviços (ex: permissões de ficheiros, permissões do *SELinux*, *Firewall*, etc).

Existe ainda o requisito de escalabilidade que era essencial para um sistema com as características para o projeto em questão. A facilidade com que é possível escalar a aplicação ou a infraestrutura de suporte, é realizada de uma forma consistente.

A instalação de uma *cloud* privada era o objetivo primordial deste trabalho, assim sendo, foi superado o desafio proposto. Todavia, existem aspetos a serem aprimorados futuramente.

Devido às restrições de dispositivos de *hardware* existe a necessidade de agregar componentes no mesmo servidor, como é o caso do servidor nodo00 (Componente de Controlo). Numa perspetiva de separar os vários serviços, e pelas recomendações da literatura, é importante manter o serviço de rede *neutron* separado dos restantes, permitindo, desta forma, uma maior facilidade na escalabilidade dos componentes físicos, e atenuar o risco de ocorrência de falhas na infraestrutura.

A nível de armazenamento opta-se por utilizar uma partição de um disco para responder às necessidades exigentes de uma infraestrutura desta dimensão, porém esta solução tem as suas limitações. O facto de se agregar no mesmo disco partições com o SO e os dados das instâncias e volumes de armazenamento, pode influenciar no desempenho da infraestrutura. Além disso, a utilização adotada não permite a funcionalidade de *live migration* do

OpenStack. Ou seja, a necessidade de substituir uma máquina física pode ser sinónimo de *downtime* de vários serviços.

Apesar das anotações mencionadas, e como se trata de um protótipo para um ambiente de pré-produção, não se revela crítico para o projeto em questão.

A maior limitação que se sente para as necessidades deste projeto é o serviço de *auto-scaling* do Cloud Foundry. A versão *open source* não incorpora o conjunto de serviços que são disponibilizados pela versão comercial e fundadora do Cloud Foundry, a Pivotal.

Um dos aspetos mencionados ao longo deste trabalho está relacionado com a automatização dos processos. Quer o Cloud Foundry, quer o OpenStack, apresentam um processo bastante automatizado de instalação. Contudo, a velocidade ganha na automatização esbarra na documentação errónea e desatualizada, culminando no aumento da dificuldade do processo de aprendizagem, configuração e diagnóstico de erros.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Edsger W. Dijkstra. The humble programmer, 1972.

[2] Renata Bastos Ferreira, Francisco De Paula, and Antunes Lima. Metodologias Ágeis : Um Novo Paradigma de Desenvolvimento de Software. *II Workshop Um Olhar Sociotécnico Sobre a Engenharia de Software*, (3):107–116, 2006.

[3] PMI. *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*, volume 1. Project Management Institute, Inc., 2008. ISBN 1-933890-51-7.

[4] Martin Fowler and Jim Highsmith. The agile manifesto. *Software Development*, 9 (August):28–35, 2001. ISSN 10708588.

[5] Scrum Alliance. https://www.scrumalliance.org/, 2015. URL https://www.scrumalliance.org/.

[6] Patrick Debois. DevOps from a Sysadmin Perspective. *Login - The Usenix Magazine*, 36(6):3, 2011.

[7] Len Bass, Ingo Weber, and Liming Zhu. *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 1 edition, 2015.

[8] VMware. vCloud Air Infrastructure as a Service (IaaS) Hybrid Cloud, 2017. URL http://www.vmware.com/cloud-services/infrastructure.html.

[9] Amazon. Amazon Web Services (AWS) - Cloud Computing Services, 2017. URL https://aws.amazon.com/.

[10] Heroku. Heroku, 2017. URL https://www.heroku.com/.

[11] Dan Kohn. A Brief History of the Cloud from Servers to VMs to Buildpacks to Cloud Native Containers., 2016. URL https://www.youtube.com/watch?v=PKUiBuEfJ08.

[12] OpenStack. Home » OpenStack Open Source Cloud Computing Software, 2017. URL https://www.openstack.org/.

[13] Cloud Foundry. Cloud Foundry — The Industry Standard for Cloud Applications, 2016. URL https://www.cloudfoundry.org/.

[14] Ron Miller. Mirantis scores huge OpenStack win with VW — TechCrunch, 2016. URL https://techcrunch.com/2016/04/05/mirantis-scores-huge-openstack-win-with-vw/.

[15] Adam Bloom. How Ford Is Revolutionizing The Auto Industry With Pivotal And Microsoft — Pivotal, 2016. URL https://blog.pivotal.io/pivotal-cloud-foundry/case-studies/how-ford-is-revolutionizing-the-auto-industry-with-pivotal-and-microsoft.

[16] DC/OS. The Definitive Platform for Modern Apps — DC/OS, 2016. URL https://dcos.io/.

[17] Markus Eisele. Modern Java EE design patterns - O'Reilly Media, 2016. URL https://www.oreilly.com/ideas/modern-java-ee-design-patterns.

[18] Freepik. Freepik, 2016. URL http://www.flaticon.com/authors/freepik.

[19] Roundicons. Roundicons, 2016. URL http://www.flaticon.com/authors/roundicons.

[20] Alan Dearle. Software deployment, past, present and future. *FoSE 2007: Future of Software Engineering*, pages 269–284, 2007. doi: 10.1109/FOSE.2007.20.

[21] John Willis. What Devops Means to Me, 2010. URL WhatDevopsMeanstoMe.

[22] Damon Edwards. What is DevOps?, 2010. URL http://dev2ops.org/2010/02/what-is-devops/.

[23] C. Aaron Cois. DevOps and Agile, 2014. URL https://insights.sei.cmu.edu/sei_blog/2014/11/devops-and-agile.html.

[24] Jez Humble. Não existe Equipa de DevOps, 2014. URL https://www.thoughtworks.com/pt/insights/blog/n%C3%A3o-existe-equipe-de-devops.

[25] James Roche. Adopting DevOps practices in quality assurance. *Communications of the ACM*, 56(11):38–43, 2013.

[26] Floris Erich, Chintan Amrit, and Maya Daneva. DevOps Literature Review. Technical report, University of Twente, 2014.

[27] Jezz Humble and David Farley. *Continuous Delivery*. 2010. ISBN 9780321601919.

[28] Sanjeev Sharma and Bernie Coyne. *DevOps For Dummies*. John Wiley & Sons, Inc., 2nd ibm edition, 2015. ISBN 978-1-119-04705-6.

[29] John Willis. DevOps Culture. URL http://itrevolution.com/devops-culture-part-1/.

[30] Damon Edwards. DevOps is not a technology problem. DevOps is a business problem., 2010. URL http://dev2ops.org/2010/11/devops-is-not-a-technology-problem-devops-is-a-business-problem/.

[31] Jim Bird. *DevOps for Finance*. O'Reilly Media, 2015. URL http://www.oreilly.com/webops-perf/free/devops-for-finance.csp.

[32] Sander Kruis. *Designing a metrics model for DevOps at Philips IT*. PhD thesis, Eindhoven University of Technology, 2014.

[33] Jennifer Davis and Daniels Katherine. *Effective DevOps*. O'Reilly Media, 2015. ISBN 978-1-4919-2626-0.

[34] Robert C. Martin. Demanding Professionalism in Software Development, 2012. URL https://www.youtube.com/watch?v=p0O1VVqRSK0.

[35] Sanjeev Sharma. IBM DevOps: Where to Start, 2014.

[36] Kief Morris. *Infrastructure as Code*. O'Reilly Media, Inc., 2015. ISBN 978-1491924358.

[37] Martin Fowler. PhoenixServer, 2012. URL http://martinfowler.com/bliki/PhoenixServer.html.

[38] Kevin Fishner. DevOps Landscape 2015: The Race to the Management Layer, 2015. URL http://thenewstack.io/a-brief-look-at-immutable-infrastructure-and-why-it-is-such-a-quest/.

[39] a Abran, James W Moore, R Dupuis, Rl Dupuis, and L L Tripp. *Guide to the software engineering body of knowledge (swebok)*. 2001. ISBN 0769523307. doi: 10.1234/12345678.

[40] Anne Mette and Jonassen Hass. What Is Configuration Management? In *Configuration Management Principles and Practice*, chapter Chapter 1, pages 1–28. Addison Wesley, 2002. ISBN 0-321-11766-2.

[41] Ronald Kirk Kandt. Configuration Management Principles and Practices, 2002.

[42] Aliza Earnshaw and Tim Zonca. Automated Configuration Management - Why it Matters and How to Get Started. *Puppet Labs*, page 12, 2014.

[43] Margaret Rouse and Stephen Bigelow. Infrastructure as Code (IAC) definition. 2015. URL http://searchcloudcomputing.techtarget.com/definition/Infrastructure-as-Code-IAC.

[44] Martin Fowler. Continuous Integration, 2006. URL http://martinfowler.com/articles/continuousIntegration.html.

[45] James Shore. Continuous Integration on a Dollar a Day, 2006. URL http://www.jamesshore.com/Blog/Continuous-Integration-on-a-Dollar-a-Day.html.

[46] Margaret Rouse. Automated software testing definition. 2014. URL http://searchsoftwarequality.techtarget.com/definition/automated-software-testing.

[47] Glenford J Myers, Todd M Thomas, and Corey Sandler. *The Art of Software Testing*, volume 1. John Wiley & Sons, 2004. ISBN 0471469122. doi: 10.1002/stvr.321.

[48] George Candea, Stefan Bucur, and Cristian Zamfir. Automated software testing as a service. *Proceedings of the 1st ACM symposium on Cloud computing SoCC 10*, 33(4): 155, 2010. doi: 10.1145/1807128.1807153.

[49] MQTT. MQTT, 2017. URL http://mqtt.org/.

[50] NATS. NATS - Cloud Native, Open Source, High Performance Messaging, 2017. URL https://nats.io/.

[51] Anil Karmel, Ramaswamy Chandramouli, and Michaela Iorga. NIST Definition of Microservices, Application Containers and System Virtual Machines. 2016.

[52] James Lewis and Martin Fowler. Microservices - a definition of this new architectural term, 2014. URL http://www.martinfowler.com/articles/microservices.html.

[53] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture, 2016. ISSN 07407459.

[54] Docker. Docker - Build, Ship, and Run Any App, Anywhere. URL https://www.docker.com/.

[55] LXC. Linux Containers. URL https://linuxcontainers.org/.

[56] Rocket. CoreOS is building a container runtime, rkt, 2016. URL https://coreos.com/blog/rocket/.

[57] OpenVZ. OpenVZ Virtuozzo Containers, 2016. URL https://openvz.org/.

[58] Jails. Jails - FreeBSD, 2016. URL https://wiki.freebsd.org/Jails.

[59] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. *Nist Special Publication*, 145:7, 2011. doi: 10.1136/emj.2010.096966.

[60] Microsoft. Microsoft Azure, 2017. URL https://azure.microsoft.com/en-us/.

[61] Google. Google Cloud Computing, Hosting Services, APIs — Google Cloud Platform, 2017. URL https://cloud.google.com/.

[62] Rackspace. Rackspace: Managed Dedicated & Cloud Computing Services, 2017. URL https://www.rackspace.com/.

[63] CloudStack. Apache CloudStack: Open Source Cloud Computing, 2017. URL http://cloudstack.apache.org/.

[64] HPE Helion Eucalyptus. HPE Helion Eucalyptus Open Source Hybrid and Private Cloud Software for AWS Users — Hewlett Packard Enterprise, 2016. URL http://www8.hp.com/us/en/cloud/helion-eucalyptus-overview.html.

[65] OpenNebula. OpenNebula – Flexible Enterprise Cloud Made Simple, 2016. URL http://opennebula.org/.

[66] Pivotal Software. Pivotal Cloud Foundry. 2017.

[67] IBM Bluemix. IBM Bluemix, 2017. URL https://www.ibm.com/cloud-computing/bluemix/.

[68] Bosch IoT Cloud. Bosch IoT Cloud, 2017. URL https://www.bosch-si.com/products/bosch-iot-suite/iot-cloud/bosch-iot-cloud-2.html.

[69] Openshift Origin. Openshift Origin, 2016. URL https://www.openshift.org/.

[70] Google. Google Mail, 2017. URL https://mail.google.com.

[71] Microsoft. Office 365 for Business, 2017. URL https://products.office.com/.

[72] ISO. *ISO/IEC 25010:2011 Preview Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. 1 edition, 2011.

[73] Apache Subversion. Subversion, 2016. URL https://subversion.apache.org/.

[74] Scott Chacon. Git Community Book. *The Git Community*, page 132, 2009.

[75] Stack Exchange Inc. Stackoverflow, 2016. URL http://stackoverflow.com/.

[76] Alexa and Amazon. Alexa, 2016. URL http://www.alexa.com/.

[77] Microsoft. Team Foundation Server, 2016. URL https://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx.

[78] Stackoverflow. Developer Survey, 2015. URL http://stackoverflow.com/research/developer-survey-2015.

[79] C. Michael Pilato, Ben Collins-Sussman, and Brian W. Fitzpatrick. *Version Control with Subversion*. O'Reilly Media, second edi edition, 2008. ISBN 978-0596510336.

[80] Bill Childers. Geek Guide - The DevOps Toolbox. *Linux Journal*, page 22, 2015.

[81] Jenkins. Jenkins - Build great things at any scale, 2017. URL https://jenkins.io/.

[82] Atlassian. Bamboo - Continuous integration, deployment & release management, 2017. URL https://www.atlassian.com/software/bamboo.

[83] Atlassian. Bitbucket - the Git solution for professional teams, 2017. URL https://www.atlassian.com/software/bitbucket.

[84] Atlassian. JIRA Software - Issue & Project Tracking for Software Teams, 2017. URL https://www.atlassian.com/software/jira.

[85] Atlassian. Atlassian - Software Development and Collaboration Tools, 2017. URL https://www.atlassian.com/.

[86] Adam Wiggins. The Twelve-Factor App, 2012. URL https://12factor.net/.

[87] Google. The Go Programming Language, 2017. URL https://golang.org/.

[88] Node.js Foundation. Node.js, 2017. URL https://nodejs.org/en/.

[89] Microsoft. .NET - Powerful Open Source Cross Platform Development, 2017. URL https://www.microsoft.com/net.

[90] Yukihiro Matsumoto. Ruby Programming Language, 2017. URL https://www.ruby-lang.org/en/.

[91] R. Finlayson. Bootstrap loading using TFTP. 1984.