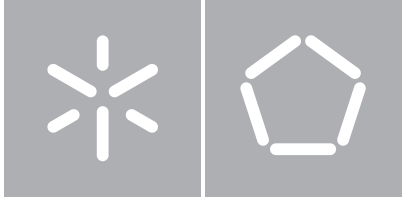
Universidade do Minho
Escola de Engenharia



Universidade do Minho
Escola de Engenharia

Dissertação de Mestrado

Interoperabilidade e Portabilidade em Ambientes PaaS

David Gonçalves da Cunha

Dissertação apresentada à Universidade do Minho para obtenção do grau de Mestre em Redes e Serviços de Comunicações sob a supervisão científica do Professor Doutor Pedro Nuno Miranda de Sousa, e com coordenação na empresa pelo Engenheiro Pedro Miguel Naia Neves.

Universidade do Minho

Escola de Engenharia

Departamento de Informática

Outubro, 2012

Agradecimentos

Esta dissertação resultou de um longo trabalho e dedicação. Contudo, não seria possível sem a ajuda e conselhos que me foram dados durante todo o trabalho.

Em primeiro lugar, quero agradecer a disponibilidade e orientação académica do professor Doutor Pedro Nuno Sousa, e também agradecer ao orientador da empresa, o Eng. Pedro Neves, pelo apoio e disponibilidade.

Um agradecimento a todos os colaboradores da PT Inovação e ao Instituto de Telecomunicações de Aveiro, que direta ou indiretamente contribuíram para o sucesso deste trabalho.

Por último, quero agradecer à minha família e a minha namorada, Francisca, por todo o apoio e compreensão durante a elaboração desta dissertação.

Resumo

O *Cloud Computing* tem emergido como sendo um novo paradigma para entrega de serviços através da Internet. Neste mercado em expansão, o serviço de PaaS (Platform-as-a-Service) tem sido objeto de grande interesse por parte das mais variadas organizações permitindo o fácil *deployment* de aplicações sem necessidade de uma infraestrutura dedicada, instalação de dependências ou configuração de servidores. No entanto, cada fornecedor de solução PaaS acaba por gerar um *lock-in* do utilizador às suas características proprietárias, tecnologias ou APIs (Application Programming Interfaces). Além disso, dando como garantida a conectividade até aos clientes, a rede de operadores como seja o caso da Portugal Telecom (PT) acaba por servir apenas de *dumb-pipe* entre o fornecedor e os seus clientes.

Este projeto foca-se na especificação, desenvolvimento e avaliação de uma camada de abstração que visa unificar os processos de gestão e aquisição de informação sobre aplicações e bases de dados criadas através de diversos PaaS, de modo a combater o *lock-in* existente no mercado. Neste sentido, um utilizador de PaaS pode selecionar a plataforma mais adequada para uma aplicação interagindo de forma idêntica com qualquer fornecedor suportado, tendo também a possibilidade de migrar aplicações entre fornecedores distintos. Assim sendo, um operador como a PT tem agora a possibilidade de agir como um mediador entre os utilizadores e os fornecedores de PaaS.

Abstract

Cloud Computing has emerged as a new paradigm for services delivering over the Internet. In this growing market, PaaS (Platform-as-a-Service) service model has been the subject of great interest from several organizations allowing a straightforward deployment of applications without the need for a dedicated infrastructure, dependencies installation or server configuration. However, each PaaS provider generates a lock-in to their proprietary features, technologies or APIs (Application Programming Interfaces). Moreover, giving connectivity to their customers as guaranteed, the network of operators such as Portugal Telecom (PT), acts only as dumb-pipe between the provider and its customers.

This project focuses on the specification, development and test of an abstraction layer that aims to unify the management and information processes of applications and databases created in PaaS environments, in order to struggle the existing lock-in in the market. As result, a PaaS user will have the opportunity for selecting the most appropriate platform for an application, interacting seamlessly with any supported vendor, as also having the opportunity to attain the portability of applications among distinct PaaS providers. Thus, an operator such as PT is now able to act as a mediator between the customers and the PaaS providers.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

ADE	Application Development Environment
API	Application Programming Interface
APM	Application Performance Management
ARPANET	Advanced Research Projects Agency Network
AWS	Amazon Web Services
CAMP	Cloud Application Management for Platforms
CLI	Command-Line Interface
CPU	Central Processing Unit
CRM	Customer Relationship Management
CSB	Cloud Service Broker
CSE	Content Selection Enabler
CtxE	Context Enabler
CxB	Context Broker
DNS	Domain Name System
EC2	Elastic Compute Cloud
EJB	Enterprise Java Bean
FP7	Seventh Framework Programme
GAE	Google App Engine
GB	Gigabyte
GME	Group Manager Enabler
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDC	International Data Corporation
IaaS	Infrastructure-as-a-Service
JAXB	Java Architecture for XML Binding
JSON	JavaScript Object Notation

JVM	Java Virtual Machine
JavaEE6	Java Enterprise Edition 6
LXC	Linux Containers
MB	Megabyte
NIST	National Institute of Standards and Tecnology
OASIS	Organization for the Advancement of Structured Information Standards
OCCI-WG	Open Cloud Computing Interface Working Group
OGF	Open Grid Forum
PDP	Platform Deployment Package
PT	Portugal Telecom
PTIN	Portugal Telecom Inovação
PaaS	Platform-as-a-Service
QoS	Quality-of-Service
RAM	Random-Access Memory
RDS	Relational Database Service
REST	Representational State Transfer
ROI	Return on Investment
S3	Simple Storage service
SDK	Software Development Kit
SLA	Service Level Agreement
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SaaS	Software-as-a-Service
TI	Tecnologias de Informação
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WSDL	Web Service Description Language
XEP	XMPP Extension Protocol
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

Capítulo 1

Introdução

1.1 Enquadramento e Motivação

Com o crescimento exponencial da Internet e das tecnologias de processamento e armazenamento, os preços dos recursos computacionais foram reduzidos tornando-se acessíveis às grandes massas. Isso fez com que as aplicações entregues pela *web*, e agora acessíveis por múltiplos terminais, despoletassem um novo mercado atrativo para a indústria das TI (Tecnologias de Informação). Além de uma *buzzword* como o próprio termo *web* o é, *cloud computing* é uma evolução de vários paradigmas tecnológicos das últimas décadas transformando o sonho do *computing as a utility* numa realidade [1]. O potencial de tal modelo preconiza um grande impacto na indústria das TIs onde os recursos computacionais e o *software* são entregues aos utilizadores finais através de um paradigma *pay-per-use* [2]. A analogia do *cloud computing* aos serviços tradicionais como eletricidade, gás ou água verifica-se sempre presente. Os utilizadores podem aceder aos serviços quando o pretenderem, independentemente das suas localizações e sendo cobrados pelos fornecedores relativamente aos níveis de utilização [3].

Nos últimos anos, sensivelmente desde de 2009, têm surgido diversos fornecedores e *start-ups*[1] de soluções PaaS que competem entre si arduamente focando-se nos aspetos inovadores que os possam diferenciar face ao utilizador. Portanto, cada fornecedor tem intrinsecamente associado diferentes linguagens de programação, *frameworks*, *Database-as-a-Service*, taxonomias, modelos de negócio, ferramentas de desenvolvimento e APIs

[1] organizações recém-criadas que estão em fase de desenvolvimento e pesquisa de mercados.

para interação por terceiros. Por um lado, esta mescla de fornecedores favorece o processo de seleção por parte do utilizador que tem a oportunidade de avaliar as diferentes ofertas sem instalar, configurar ou manter alguma dependência. Porém, em contrapartida, surge a possibilidade de existir um *lock-in* do utilizador a um fornecedor específico e às suas características proprietárias. Até ao momento, e após um utilizador formar uma relação de negócio com um fornecedor, o mesmo fica vulnerável aos processos de interação, à API disponibilizada e aos modelos de negócios que podem ser sujeitos a alteração repentina. A interoperabilidade entre fornecedores de serviços *cloud* tem surgido na comunidade como sendo um tópico de preocupação geral associado à temática do *inter-cloud* [4]. A nível empresarial se este conceito se aplicasse, faria com que mais organizações diminuíssem a sua reticência na adesão ao *cloud computing*, neste caso soluções PaaS, de forma a gerirem e migrarem aplicações transparentemente entre diferentes fornecedores.

A Portugal Telecom Inovação (PTIN), empresa do grupo PT onde foi desenvolvido este projeto, poderá fornecer os alicerces para a entrada no mercado de *cloud* de uma abordagem mediadora entre os utilizadores e os fornecedores de PaaS. Consequentemente será oferecida aos clientes a capacidade de controlar várias plataformas de forma centralizada abstraindo as diferenças intrínsecas na interação com cada fornecedor suportado.

1.2 Objetivos

Este trabalho tem como principal objetivo definir e desenvolver uma camada de abstração que visa unificar os processos de gestão e aquisição de informação de aplicações criadas através de diversos PaaS, de modo a combater o *lock-in* existente no mercado. Inicialmente deverá ser realizada uma análise dos conceitos relacionados com o *cloud computing* como também de algumas iniciativas de interoperabilidade entre fornecedores de soluções de *cloud* já existentes na comunidade. Depois deste estudo, será necessário investigar algumas das principais plataformas que surgem no mercado com o intuito de implementar uma solução que agregue as diferentes ofertas de forma transparente para o utilizador. Essa solução verificar-se-á útil na unificação e centralização da criação, gestão e monitorização de aplicações em diversos PaaS. Além disso, a informação de estado devolvida aos utilizadores dar-lhes-á a oportunidade para migrar aplicações para outros fornecedores pertencentes ao ecossistema.

Desta forma, são enumerados os seguintes objetivos parcelares delineados para este

trabalho:

- Investigar conceitos associados ao *cloud computing*, *Platform-as-a-Service* e conceitos SOA (Service-Oriented Architecture).
- Analisar iniciativas já existentes que visam unificar a gestão de recursos efetuada através de diversos fornecedores de soluções *cloud*.
- Explorar os vários fornecedores de PaaS existentes no mercado bem como as tecnologias, ferramentas e modos de operação suportados.
- Especificar e implementar uma solução que agregue as varias soluções de PaaS selecionadas, centralizando todos os processos de criação, gestão e monitorização, dando também ao utilizador a possibilidade de migrar aplicações entre fornecedores.
- Definir casos de estudo para testar a solução em cenários reais, avaliando os resultados obtidos como o seu desempenho.

1.3 Sumário das Principais Contribuições

A principal contribuição deste trabalho é o desenvolvimento de uma solução que permita combater o *lock-in* existente no mercado de fornecedores de plataformas. Consequentemente foi desenvolvida uma plataforma denominada por *PaaSManager* que permite aos utilizadores destes ambientes operarem as suas aplicações de forma unificada e centralizada qualquer que seja o PaaS onde a aplicação está hospedada. Como fruto de uma parte significativa deste trabalho, foi publicado o artigo [5], apresentado na conferência internacional CLOSER 2012 [6]. Adicionalmente, foi publicado um outro artigo [7], que abrange um dos casos de estudo propostos, na conferência CRC 2012 [8]. E por fim, foi submetido um artigo, que incide exclusivamente na solução *PaaSManager*, na conferência SAC 2013 [9]. De momento o artigo encontra-se em processo de revisão.

1.4 Organização da Dissertação

Este documento está organizado em 5 capítulos. A descrição de cada um dos capítulos é a seguinte:

- Introdução: este capítulo fez o enquadramento e a contextualização do trabalho, ao apresentar o tema geral e os objetivos do trabalho a desenvolver.

- Estado da Arte: aborda a base teórica que irá permitir a especificação e a implementação de uma camada de abstração que agregue as ofertas de diferentes fornecedores de PaaS. São apresentados os conceitos associados ao *cloud computing*, os três principais modelos de serviços, as iniciativas de interoperabilidade existentes e a temática relacionada com o desenvolvimento de aplicações segundo conceitos SOA.

- Especificação e Desenvolvimento do *PaaSManager*: aborda todo o trabalho desenvolvido, desde da análise efetuada às diferentes plataformas, passando pela a especificação da arquitetura e o próprio desenvolvimento dos diversos módulos da solução *PaaSManager*. Igualmente será detalhado o funcionamento de uma *framework* de serviços baseados em informação de contexto desenvolvida na PTIN que será utilizada em um dos casos de estudo.

- Ensaios e Avaliação: demonstra os vários casos de estudos convencionados, a metodologia de testes utilizada e a avaliação dos resultados obtidos. O intuito é o de validar a solução em cenários reais.

- Conclusões: neste último capítulo são apresentadas as principais conclusões obtidas de todo o trabalho. É realizada uma análise de todos os capítulos e as principais contribuições desta dissertação. Por fim, são expostas algumas das futuras melhorias que poderão ser efetuadas na solução proposta.

- Apêndice A: no apêndice A encontra-se documentada a API do *PaaSManager* detalhando os vários métodos suportados, como os parâmetros de entrada necessários para cada operação e alguns exemplos de pedidos e respostas.

Capítulo 2

Estado da Arte

Neste capítulo serão destacados os conceitos associados ao *cloud computing* e respetivos modelos, IaaS (Infrastructure-as-a-Service), PaaS e SaaS (Software-as-a-Service). Posteriormente serão apresentadas algumas iniciativas de interoperabilidade entre fornecedores de *cloud* que visam unificar o aprovisionamento e a gestão de serviços. Por fim, os princípios SOA e sinergias existentes com o *cloud computing* serão também abordados.

2.1 Definir *Cloud Computing* e a sua Evolução

O conceito que deu origem ao *cloud computing* não é de todo moderno. O termo data da década de 60 quando John McCarthy, um cientista americano reconhecido pelos seus estudos na área da inteligência artificial, afirmou que um dia a computação seria organizada e entregue como uma utilidade pública [10]. Por sua vez em 1969, Leonard Kleinrock, um dos cientistas responsáveis pela ARPANET (Advanced Research Projects Agency Network), declarou que no futuro as redes de computadores se tornariam muito mais sofisticadas servindo utilizadores em todo o mundo nas suas próprias habitações ou escritórios [3]. Ambos estavam corretos, mas no entanto apenas no século XXI, e após a explosão da *web* como hoje a conhecemos, o termo *cloud* ganhou popularidade para caracterizar os modelos de negócio da entrega de serviços através da Internet [11]. A falta de uma definição normalizada gerou ceticismo e confusão sendo que ainda hoje não existe uma definição única e consensual aceite pela comunidade. Porém o NIST (National Institute of Standards and Tecnology) definiu o *cloud computing* como sendo "*um modelo que permite o acesso ubíquo,*

conveniente e on-demand através da Internet, a um conjunto partilhado e configurável de recursos computacionais (por exemplo, redes, servidores, armazenamento e aplicações) que podem ser rapidamente aprovisionados e libertados com um esforço mínimo de gestão ou de interação com o fornecedor de serviço" [12].

Não sendo um conceito completamente novo, diversas tecnologias como *grid computing*, *cluster computing*, *utility computing* ou a virtualização fazem parte dos princípios fundamentais do *cloud computing* [13]. A Figura 2.1 apresenta a evolução ocorrida ao longo das últimas décadas desde da afirmação de John McCarthy, passando pela origem da virtualização e do *grid computing* até ao lançamento dos primeiros serviços comerciais baseados na *cloud* pela Amazon. Nos dias de hoje, as *buzzwords* mais debatidas abordam o modelo *cloud* 2.0, que descreve a integração de serviços *web* 2.0 com aplicações baseadas na *cloud*, e a *inter-cloud* que fomenta a interoperabilidade de serviços de *cloud* entre diversos fornecedores [11],[14],[15].

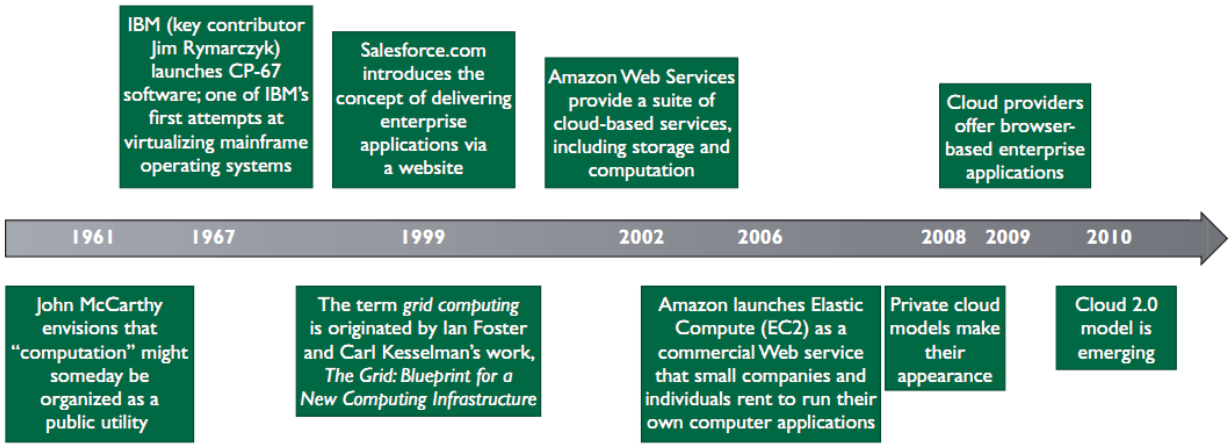


Figura 2.1: Evolução do *cloud computing* [16]

O crescimento e a maturação da *cloud* continuará a seguir esta tendência de evolução constante prevendo-se que durante esta década a maior parte dos utilizadores da *Internet* sejam cada vez mais consumidores ativos de soluções hospedadas na *cloud*. Como prova disso, a Market Research Media realizou um estudo a pedido do governo norte-americano, prevendo que até 2015 o mercado de *cloud computing* cresça cerca de 40% [17]. Por sua vez, O IDC (International Data Corporation), analisou o número de novos empregos gerados direta e indiretamente com este crescimento obtendo resultados de 13,8 milhões de novos postos de trabalho em 2015, como se pode verificar na Figura 2.2.

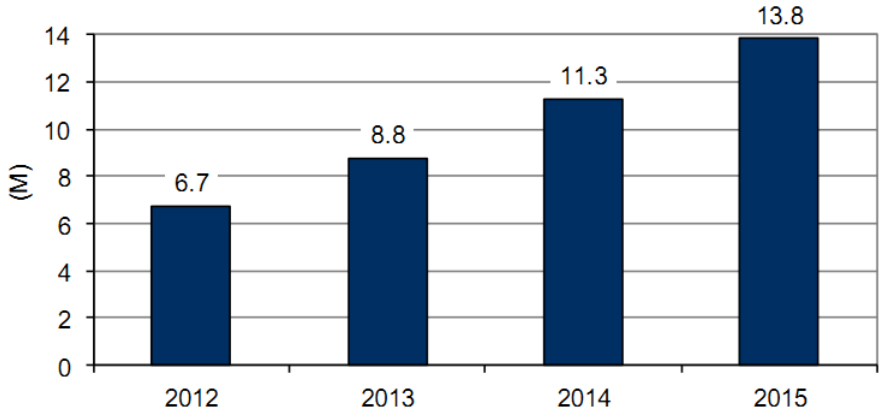


Figura 2.2: Previsão de empregabilidade devido ao *cloud computing* [18]

2.2 Características Fundamentais

Normalmente, qualquer planeamento de capacidade de recursos computacionais, configuração de sistemas, atualizações de *software*, problemas na rede elétrica ou desastres naturais, é suportado pelas próprias organizações e respetivos departamentos técnicos. Em certos casos, todas estas responsabilidades podem se verificar penosas e insustentáveis se por exemplo ocorrer alguma falha técnica na infraestrutura ou se, por um certo motivo, os servidores não estiverem disponíveis para atenderem toda a procura dos clientes. No entanto, estes comprometimentos podem ser transferidos para terceiros, nomeadamente fornecedores comerciais de soluções de *cloud*, que normalmente possuem *data centers* em diversos pontos geográficos com equipas técnicas especializadas 24 horas por dia. Com esta migração, são reduzidos os custos associados à manutenção de infraestrutura e à contratação e treino de equipas especializadas que forneçam a resiliência necessária em situações adversas. Por outro lado, o aprovisionamento de recursos computacionais torna-se automatizado de forma a cobrir a procura necessária pelos clientes sem existir investimento em novos elementos de *hardware* (CapEX[1]).

A Figura 2.3 descreve sucintamente como o modelo tradicional e o *cloud computing* reagem à variação da capacidade necessária para uma atender a procura dos utilizadores a um certo serviço. A curva à laranja representa como essa própria procura se comporta ao longo tempo. A curva é rigorosamente acompanhada pela oferta *cloud* devido a sua característica de rápida elasticidade que permite escalar facilmente os recursos computaci-

[1] capital investido na aquisição de bens.

onais. No entanto, como se observa na linha verde, o modelo tradicional não acompanha a ocorrência de picos de procura. Em certas situações de carga, a infraestrutura pode revelar sobrecapacidade, ou em casos opostos, subcapacidade que consequentemente resulta na degradação da experiência de utilização de um serviço.

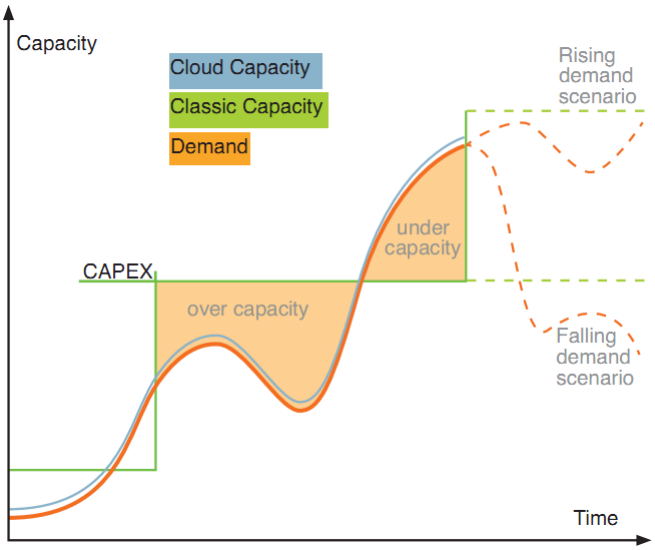


Figura 2.3: *Cloud Computing* vs. Modelo Tradicional [19]

As vantagens de utilizar soluções *cloud* são diversas cobrindo desde aspetos técnicos a económicos. Em suma, as características fundamentais do *cloud computing* [10],[20],[21] são enumeradas seguidamente:

***On-Demand*:** Esta capacidade possibilita o aprovisionamento de recursos computacionais por parte do utilizador conforme as suas necessidades sem precisar de existir uma interação direta com o fornecedor do serviço.

Rápida Elasticidade: É um dos principais alicerces que suporta o *cloud computing*. Para o utilizador existe a ilusão que os recursos fornecidos são infinitos, os mesmos podem ser elasticamente aprovisionados e libertos em qualquer período de forma a cobrirem a procura dos clientes.

***Pay-per-use*:** Os utilizadores são cobrados pelo fornecedor consoante a utilização dos recursos disponibilizados. Esta modalidade proporciona uma melhor gestão e otimização económica do serviço contratado quando existem picos de consumos.

Investimento Inicial Reduzido: Tradicionalmente as organizações das mais diversas áreas, realizam um grande investimento inicial (CapEX) para adquirir *hardware*, licenças de *software* ou contratar funcionários especializados nas TI (se for o caso). Com o paradigma da *cloud*, existe um reduzido investimento inicial sendo que os recursos contratados são pagos relativamente à utilização. Portanto o CapEx é transformado em OpEx[2] o que se verifica benéfico e flexível para o balanço económico de qualquer organização.

***Pool* de Recursos Partilhada:** Os vários servidores físicos e virtuais presentes em um determinado *data center* de um fornecedor, podem ser partilhados por diferentes utilizadores. Esta propriedade de *multi-tenancy* reduz drasticamente os custos associados, suportando vários sistemas operativos e aplicações de diversos utilizadores em um menor número de componentes de hardware. Além disso, torna mais eficiente e rápido os processos de atualização ou *patching* do *software* disponibilizado.

Acesso Ubíquo: Um acesso simples e heterogéneo é normalmente o modelo utilizado para aceder aos serviços de *cloud* hospedados em diversos *data centers* localizados pelo globo. Um terminal com acesso à Internet, tal como um *smartphone* ou um computador portátil, pode facilmente consumir esses serviços.

2.3 Tipos de Cloud

Diferentes utilizadores exigem diferentes requisitos desde de custos, QoS (Quality-of-Service) ou segurança. Enquanto certos utilizadores podem-se encontrar mais empenhados em reduzir custos de investimento inicial, outros podem requerer um certo nível de segurança no armazenamento de dados ou até estarem interessados em agregar os vários requisitos de forma a possuírem a solução que reconhecem mais interessante para o seu negócio [10]. Assim sendo, o *cloud computing* abraçou várias formas de organizar os recursos em diversos ambientes tendo cada um as suas características mais ou menos vantajosas para os utilizadores.

[2] capital investido na manutenção e operação de bens.

2.3.1 *Cloud* Pública

Este tipo de *cloud* tem como ponto-chave a comercialização por parte de um fornecedor de uma oferta de serviços ao público em geral ou a organizações que operam em diversas áreas. Este paradigma é o mais popular no *cloud computing*, permitindo a requisição de recursos como servidores, armazenamento, processamento e aplicações *on-demand* através da Internet. Os benefícios são diversos para quem utiliza este género de ambientes, já que não é necessário efetuar algum investimento inicial em infraestrutura ou um extenso planeamento prévio de capacidade [22]. Sendo *multi-tenant*, ou seja, com diversos utilizadores a operarem recursos no mesmo ambiente, os preços praticados pelos fornecedores verificam-se mais reduzidos do que em outro tipo de soluções. No entanto, algumas organizações e respetivos administradores de sistemas podem revelar reticência na colocação de dados críticos e serviços fora do perímetro das *firewalls* corporativas. Nestas situações, outras soluções baseadas nas próprias premissas podem-se verificar mais atrativas para estes tipos de negócios [14]. A Figura 2.4 realça a interação dos utilizadores, sejam particulares ou empresariais, com os fornecedores de serviços de *cloud* pública.

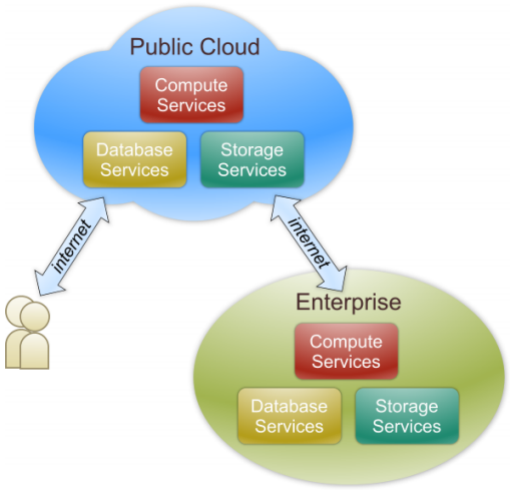


Figura 2.4: *Cloud* pública [23]

2.3.2 *Cloud* Privada

Esta abordagem, nomeadamente conhecida por *cloud* privada, expõe as potencialidades da construção e operação de recursos dedicados como *hardware* e *software* exclusivamente para consumo da organização ou arrendamento a terceiros. Este paradigma oferece um

grande controlo no desempenho, eficiência, fiabilidade e segurança, contudo, tem vindo a ser criticado fortemente por ir contra os princípios fundamentais do *cloud computing* sendo considerada uma solução menos ótima para os utilizadores. As organizações que implementam esta abordagem de *cloud* privada necessitam de adquirir e operar os recursos computacionais nas suas premissas como já ocorre no paradigma tradicional das TIs [14],[20]. Como vantagem, a infraestrutura previamente adquirida poderá ser reaproveitada maximizando o ROI[3] (Return on Investment) da organização detentora, sendo que em alguns casos, a mesma suporta requisitos específicos que uma *cloud* pública poderá não fornecer a partida. A Figura 2.5 apresenta o paradigma de *cloud* privada dentro de uma organização.

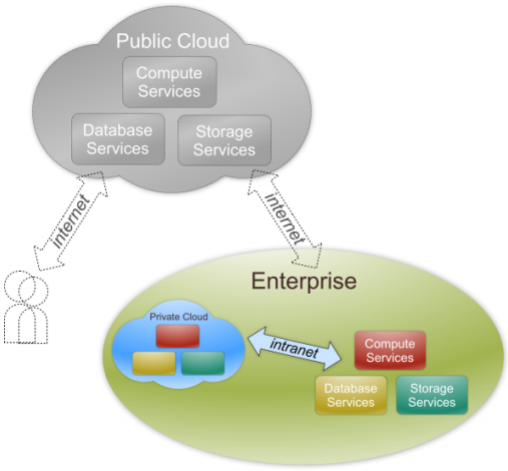


Figura 2.5: *Cloud* privada [23]

2.3.3 *Cloud* Híbrida

Este tipo de *cloud* é um híbrido entre a *cloud* pública e a *cloud* privada que visa a atenuar as limitações inerentes em cada um dos paradigmas. A *cloud* híbrida é uma composição de recursos contratados na *cloud* pública e recursos dedicados oferecendo às organizações a possibilidade de operar entre diferentes *clouds*. Esta abordagem é caracterizada pelo maior controlo e segurança comparado ao serviço público mas mantendo uma rápida elasticidade e aprovisionamento *on-demand* [14]. Através de interfaces normalizadas ou proprietárias é possível migrar os recursos entre os ambientes públicos e privados. Este processo, conhecido por *cloud bursting*, verifica-se essencial na ocorrência de picos de tráfego ou situações

[3] estimativa dos benefícios obtidos em relação ao capital investido.

adversas no qual a transferência de recursos proporciona uma cópia de segurança e um aumento do desempenho e fiabilidade nos serviços disponíveis para os clientes. Para alguns elementos da comunidade, este paradigma híbrido é considerado o ideal para qualquer utilizador que pretenda adotar o *cloud computing* maximizando o ROI e mantendo a flexibilidade e a segurança exigida. A Figura 2.6 destaca a associação de serviços de *cloud* pública com serviços de *cloud* privada formando uma *cloud* híbrida.

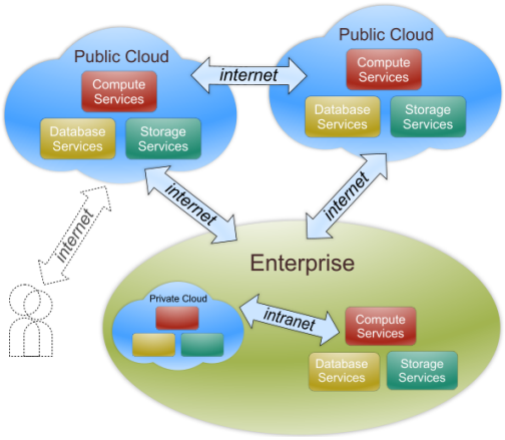


Figura 2.6: *Cloud* híbrida [23]

2.4 Modelo de Serviços

De modo a clarificar conceitos e os fornecedores definirem os seus níveis de atuação no mercado emergente de *cloud computing*, foi necessário estabelecer fronteiras entre os vários serviços que podem ser disponibilizados. A definição de um modelo por camadas, que igualmente pode ser encontrado em outras tecnologias como a pilha TCP-IP, fazia todo o sentido ser aplicado num ambiente de *cloud*. De momento ainda permanece um debate ativo entre elementos da comunidade sobre a totalidade das camadas existentes e o que cada uma engloba. Sendo possível encontrar na literatura modelos com 11 camadas [21], o modelo mais comum e mais utilizado é definido por apenas 3 camadas como é apresentado na Figura 2.7.

Os três elementos que perfazem este modelo são o *Infrastructure-as-a-Service* (IaaS), o *Platform-as-a-Service* (PaaS) e o *Software-as-a-Service* (SaaS). Uma analogia que permite entender o enquadramento de cada nível da pilha é a seguinte: por si só, a infraestrutura não é útil – ela apenas existe esperando que alguém a torna produtiva de forma a resolver

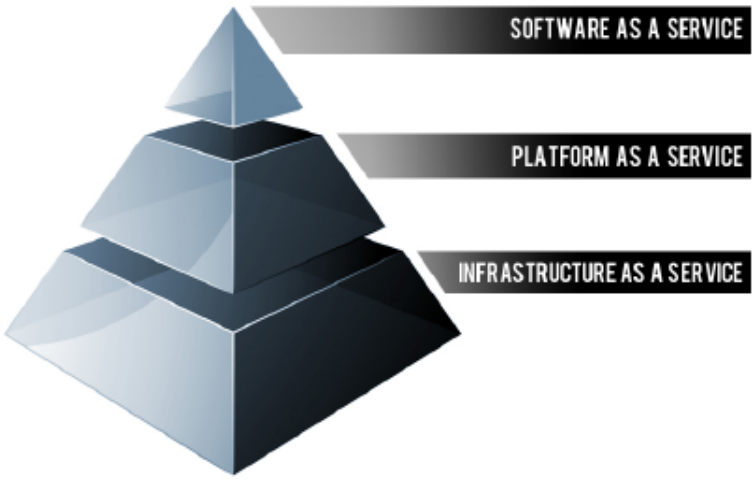


Figura 2.7: Modelo de serviços

um problema em particular. Imagine-se um sistema de transporte, mesmo com diversas estradas construídas as mesmas não seriam úteis sem a existência de automóveis que transportassem pessoas e bens. Podemos assim considerar que as estradas são a infraestrutura e os automóveis são a plataforma, que por sua vez assenta na infraestrutura e transporta pessoas e bens. Essas entidades transportadas podem ser consideradas as aplicações que são entregues pela Internet [24].

Este modelo é orientado ao fornecimento de serviços onde cada camada sustenta a camada superior. Assim sendo, cada camada pode ser vista como um cliente da camada abaixo. Este conceito abre oportunidades de negócio para fornecedores especializados numa camada, contratando o serviço ao fornecedor da camada inferior, ou então para fornecedores que ambicionam atuar transversalmente com o objetivo de cobrir um grande número de utilizadores [25]. Na Figura 2.8, observa-se os principais fornecedores existentes nas três ofertas de serviços de *cloud* onde se verifica um grande domínio da Amazon, Google e Microsoft.

Figura 2.8: Principais fornecedores de soluções *cloud*

2.4.1 *Infrastructure-as-a-Service*

Infrastructure-as-a-service é a forma de entregar remotamente recursos de infraestrutura como servidores, armazenamento ou rede, *on-demand*. Com este serviço é possível solicitar, por exemplo, uma nova instância de um servidor ficando a mesma operacional logo após esse pedido. Quando já não for necessário, o servidor poderá ser suprimido libertando os recursos alocados. Este modelo é normalmente utilizado através dos três tipos de *cloud* detalhados na secção 2.3. No caso da *cloud* pública, a infraestrutura é partilhada com outros utilizadores através da Internet e contratada num modelo *pay-per-use*. Os recursos são elásticos podendo ser facilmente escalados acompanhando a exigência dos utilizadores. Para assegurar a disponibilidade e fiabilidade, a noção da localização geográfica da infraestrutura contratada é uma das valências deste nível da pilha. Um fornecedor com *data centers* localizados em diversos pontos geográficos assegura a redundância e balanceamento de carga no caso de falhas possibilitando ao utilizador selecionar a zona mais vantajosa para o seu negócio. Em todo este serviço existe um compromisso por parte dos fornecedores na entrega dos recursos com o intuito de garantir e respeitar os acordos estabelecidos através de um SLA (Service Level Agreement). Alguns fornecedores como a Amazon garantem a disponibilidade de serviço acima dos 99,95% afirmando que se esse valor não for cumprido, o utilizador poderá receber uma nota de crédito igual a 10% da sua fatura [26]. O acesso remoto aos recursos contratados é realizado através de interfaces bem definidas variando de fornecedor para fornecedor. Certos fornecedores disponibilizam

interfaces gráficos *web*, CLIs (Command-Line Interfaces) ou APIs. Os interfaces gráficos podem ser preferidos por alguns utilizadores que em alguns passos simples desejam aprovisionar e monitorizar os recursos contratados. Por sua vez, a CLI oferece a oportunidade de executar *scripts* que realizem tarefas programadas. As APIs facultam o desenvolvimento de módulos de *software* que recorrem a pedidos HTTP (Hypertext Transfer Protocol) para interagir diretamente com a infraestrutura através das diversas funcionalidades de gestão expostas pelo fornecedor [13]. Através da abordagem da *cloud* pública, este serviço é de facto interessante para as organizações ou *start-ups* de diversas áreas que não possuam capital suficiente para investir em infraestrutura (CapEX) ou que apenas necessitam de recursos num curto espaço de tempo devido a necessidades internas. Estes utilizadores têm a possibilidade de usufruir deste modelo transformando o CapEX em despesas operacionais (OpEX) e acompanhando facilmente a procura dos clientes ao longo do tempo. No entanto, se um utilizador já possuir uma infraestrutura dedicada em rede privada que cumpra os níveis de desempenho exigidos, ou se existirem requisitos de segurança ou de legalidade inerente às suas atividades, outras soluções de *cloud* privada ou híbrida poderão verificar-se mais adequadas [27].

Os grandes fornecedores de IaaS são sem surpresa grandes nomes do mundo das TIs. A Amazon, Rackspace, GoGrid, Microsoft, IBM, e a Google mais recentemente, são alguns dos titãs da oferta de infraestrutura. Dessa forma será feita uma síntese do principal fornecedor, a Amazon.

Figura 2.10: AWS

A Amazon com o AWS (Amazon Web Services) [28] é um pioneiro e um dos maiores fornecedores no mercado de *cloud computing* introduzindo o serviço de IaaS em 2006. A sua oferta varia entre as diversas áreas que perfazem o modelo: EC2 (servidores com sistema operativo Windows ou Linux), S3 (armazenamento de dados), RDS (base de dados relacionais), Elastic MapReduce (processamento de dados), entre outros. Com estes serviços a Amazon oferece aos utilizadores a possibilidade de usufruírem deste vasto catálogo de recursos computacionais para estabelecerem o seu negócio baseado num modelo *pay-per-use*. As diversas APIs RESTful disponibilizadas são bastante completas sendo consideradas uma normalização entre a comunidade. As funcionalidades disponibilizadas pelas APIs abrangem o aprovisionamento e toda a gestão necessária dos recursos nos diversos serviços. A segurança e a geolocalização são igualmente uma forte aposta da Amazon. A AWS detém *data centers* em localizações estratégicas com perímetros de segurança físicos e espalhados pelos vários continentes de

forma a garantir a redundância e a proteção contra desastres ou outras situações adversas.

2.4.2 *Platform-as-a-Service*

Até este momento, o IaaS continua a ser o modelo de serviço *cloud* mais utilizado e com mais sucesso na área. Todavia o *Platform-as-a-Service*, sendo orientado ao desenvolvimento de aplicações, possui o potencial para abstrair as organizações de todos os processos de manutenção e configuração de servidores, bem como de instalação de dependências [29]. Um fornecedor de PaaS oferece um ambiente integrado simples e intuitivo para desenvolver, testar, monitorizar e hospedar aplicações *web* e respetivas bases de dados. Estas plataformas têm o intuito de diminuir a complexidade e o tempo alocado em todos os processos do ciclo de vida de uma aplicação, suportando as características inerentes ao *cloud computing*. Apesar do mercado de PaaS ainda se encontrar numa fase precoce, o modelo de negócio suportado poderá se tornar num componente importantíssimo na cadeia de valor da indústria de *software*. A Forrester estimou que em 2016 o volume gerado devido ao mercado de PaaS poderia atingir os 15,2 mil milhões de dólares [30], e por sua vez a Gartner, anunciou que 2013 será o ano do *Platform-as-a-Service* no qual irão surgir diversos avanços significativos. No seio de uma organização, o impacto destes ambientes, tanto a nível técnico e económico, é enorme. Os vários profissionais das TIs, nomeadamente, programadores, administradores de sistemas e até gestores empresariais, são abrangidos por esta revolução. Para o programador, o ambiente fornecido por um PaaS permite que ele se foque mais no desenvolvimento de código em vez de na configuração e manutenção das dependências subjacentes. Cada plataforma suporta diferentes ambientes de execução existindo alguns fornecedores especializados na oferta de linguagens de programação ou ferramentas específicas. No mercado é possível encontrar plataformas que suportam ambientes Java, Python, Ruby ou apenas .NET, através de servidores *web* e aplicacionais populares. As *frameworks* disponibilizadas variam desde de Ruby on Rails, Spring ou Java EE e a persistência desde de SQL, MySQL ou NoSQL. Consequentemente, os programadores desfrutam da oportunidade de testar facilmente diversas plataformas com intuito de encontrar o fornecedor de PaaS que cumpra os requisitos exigidos. Porém, é sempre indispensável que os envolvidos avaliem quais aplicações fazem mais sentido migrar para a *cloud* e qual o impacto que esse processo irá ter na organização. Essa temática irá ser discutida na secção 2.4.3. Um PaaS é bastante útil para projetos de desenvolvimento que envolvam diversas equipas que pretendem automatizar todo o ciclo de vida de uma aplicação de

forma interativa e incremental baseando-se em metodologias de desenvolvimento ágil de *software*. O fluxo de desenvolvimento que é evidenciado num ambiente *cloud* não é em nada discrepante do normalmente utilizado no modelo tradicional:

- Especificar
- Desenvolver
- Testar e Integrar
- Declarar versão

Após estas etapas, a aplicação poderá ser otimizada e colocada em produção se for o caso. Como as equipas de desenvolvimento e testes partilham a mesma plataforma, é aumentada a eficiência dos processos e são reduzidos os conflitos e erros que possam surgir durante todo o ciclo de vida aplicacional. Na maioria dos PaaS, todas as particularidades associadas a infraestrutura são de certa forma ocultadas sendo que o aprovisionamento automático de recursos computacionais faz parte das responsabilidades dos fornecedores contratados. Além disso, são também geridos todos os mecanismos de recuperação em caso de falhas, balanceamento de carga e segurança, como autenticação e autorização [31]. Os administradores de sistemas, que incluem como tarefas técnicas, a análise de *logs*, a atualização de *software* ou até a manutenção de discos rígidos, podem agora se focar em manter os sistemas complexos operacionais em vez de servidores e sistemas operativos. Por outro lado, os gestores também apreciam a flexibilidade que o PaaS pode trazer para as suas organizações. Em poucos dias um simples protótipo de um serviço pode ser lançado para produção sem existir um grande investimento inicial por parte da organização detentora. Subitamente, a inovação cessa de estar no horizonte de quem possui avultados recursos económicos para transformá-la num produto comercializável.

No mercado surgem diversos fornecedores de PaaS cada um oferecendo um serviço diferenciado para o utilizador. Além dos titãs Amazon, Google e Windows, emergem ofertas especializadas em linguagens de programação específicas, caso do CloudBees, e igualmente soluções *open-source*, nomeadamente o OpenShift da Red Hat e o CloudFoundry da VMware. O Heroku da Salesforce.com surge como um dos principais fornecedores de PaaS especializado em linguagens *open-source* tais como Ruby e Python e suportando Git como ferramenta de controlo de revisão. Dessa forma será feita uma síntese desses

fornecedores.

Figura 2.12: CloudBees

O CloudBees [32] foi fundado em 2010 e é um PaaS inteiramente direcionado ao desenvolvimento de aplicações Java. Portanto, suporta qualquer linguagem de programação e *framework* que executem sobre uma JVM (Java Virtual Machine), nomeadamente, Java, JRuby, Scala, Play, Grails, Spring, etc. O CloudBees fornece dois serviços, sendo um deles orientado ao desenvolvimento e teste de aplicações, e outro orientado apenas ao *deployment* e execução das mesmas. Através do *DEV@Cloud*, o utilizador tem acesso a ferramentas como Jenkins, Maven, Sonar e repositórios Git e SVN de modo a controlar todo o ciclo de vida da aplicação. Por sua vez através do *RUN@Cloud*, poderá efetuar rapidamente o *deployment* de uma aplicação, já criada anteriormente, adquirindo acesso a todas as funcionalidades de gestão, monitorização e *logging*. Para interação por parte de utilizadores o CloudBees fornece diversas interfaces, como o próprio interface gráfico *web*, um *plugin* para Eclipse, um SDK (Software Development Kit) e uma API RESTFul. No entanto, a API apenas permite controlar as funcionalidades inerentes ao modelo de execução *RUN@Cloud*. O modelo de negócio suportado fornece 2 planos de subscrição *pay-per-use*. O plano *Free* é ótimo para os utilizadores que desejam testar a plataforma de forma gratuita mas estando limitado a 5 aplicações, 5 base de dados MySQL e uma instância de servidor por aplicação. Por outro lado, o plano *Enterprise* oferece as funcionalidades de recuperação em caso de falhas, balanceamento de carga, segurança e escalabilidade tanto horizontal através de instâncias como vertical através de *app-cells*. No contexto do CloudBees, um *app-cell* é uma porção de um servidor e possui 128MB de memória RAM. Por sua vez, uma instância como o próprio nome indica, é uma instância de um servidor. Existe uma relação entre ambos os conceitos, já que uma instância poderá suportar no máximo 10 *app-cells*. A nível de infraestrutura, o CloudBees assenta por omissão no serviço público fornecido pela AWS, todavia para comodidade dos utilizadores, possibilita outras opções como a HP, a OVH.com ou uma qualquer infraestrutura privada que o utilizador detenha.

Figura 2.14: CloudFoundry

O CloudFoundry [33] é um PaaS que possui uma abordagem dissemelhante em comparação a quase todas as restantes plataformas do mercado por ser completamente *open-source* com licença Apache v2. Lançado em 2011 pela VMware, o CloudFoundry prima por suportar diversos ambientes de execução: Java, Scala, Ruby, Node.js, etc.; *frameworks*: Spring, Sinatra, Rails etc.; base de dados: MySQL, MongoDB, Redis, PostgreSQL; sem estar fixo a uma única

infraestrutura. O utilizador tem a oportunidade de alterar o código fonte do próprio PaaS e assentá-lo sobre qualquer serviço de infraestrutura ao seu dispor, seja público ou privado. Com este paradigma *multi-cloud* e *open-source*, o CloudFoundry tem recebido um exaustivo reconhecimento no mundo das plataformas sendo apelidado de Linux do *cloud computing*. Por detrás deste sucesso, existe uma forte comunidade de utilizadores e organizações como a ActiveState ou a Joyent que têm participado arduamente no desenvolvimento do CloudFoundry. Além disso, o código fonte da plataforma foi utilizado como base para algumas soluções comerciais como o AppFog, Stackato e outras soluções gratuitas como o IronFoundry. Para a interação por parte de utilizadores, existem diversas interfaces como uma CLI, um *plugin* para Eclipse e uma API RESTFul. O código fonte e documentação da própria plataforma como também de todas as interfaces de utilizador encontram-se disponíveis em repositórios públicos do GitHub. De momento, o CloudFoundry não possui nenhum modelo de negócio devido a última versão ainda ser beta, porém é fornecida uma solução pública limitada a 16 bases de dados e 20 aplicações. A nível de recursos, a soma de memória RAM alocada por todas as aplicações está limitada a 2GB sendo que o utilizador pode escalar horizontalmente a aplicação até ao máximo de 16 instâncias por detrás de um balanceador de carga. O CloudFoundry promete mais desenvolvimentos e novidades num futuro breve que poderá passar por uma solução comercial.

Figura 2.16: Heroku

O Heroku [34] surgiu em 2007 como sendo um PaaS orientado exclusivamente ao desenvolvimento de aplicações em Ruby. Após ser adquirido pela Salesforce.com em 2010, o Heroku evoluiu suportando mais tecnologias e tornando-se um dos PaaS mais utilizados e célebres do mercado. De momento, é estimado que suporte mais de 1,5 milhão de aplicações de todo o tipo de linguagens de programação desde de Java, Scala, Python, Ruby, PHP ou Node.js. com base de dados PostgreSQL ou Redis. O Heroku possui um amplo catálogo de *addons* que permite aos utilizadores adicionarem diversos tipos de base de dados SQL e NoSQL, serviços de monitorização, *logging*, faturação, teste, etc. Para interação por parte de utilizadores, o Heroku fornece diversas interfaces como o próprio interface gráfico *web*, uma CLI, um *plugin* para Eclipse, Git e uma API RESTFul. O Git tem-se tornado um *de facto* no mercado de PaaS sendo cada vez mais utilizado como ferramenta de *deployment* e controlo de revisão de código fonte em diversos PaaS, não sendo exceção no Heroku. O modelo de negócio suportado é *pay-per-use* onde o utilizador tem acesso às funcionalidades de recuperação em caso de falhas, balanceamento de carga, segurança e escalabilidade de processos. Ao contrário de outros fornecedores, o Heroku tem uma abordagem diferente em relação a escalabilidade.

Em vez de serem alocadas novas instâncias de servidores de forma a suportar elasticamente os pedidos à aplicação, o Heroku foca-se em processos denominados de *dynos*. Um *dyno* possui 512MB de memória RAM estando completamente isolado através de LXC (Linux Containers) de forma a não afetar o correto funcionamento de outros *dynos* existentes. Um *dyno* pode ser de tipo *web*, de modo a aumentar o desempenho e o atendimento de pedidos HTTP, e de tipo *worker*, para executar *background jobs* e fornecer mais capacidade de processamento a aplicação. A infraestrutura que suporta o Heroku é fornecida inteiramente pela AWS não existindo nenhuma informação sobre a possibilidade de utilizar outros IaaS ou uma abordagem de *cloud* privada.

Figura 2.18: OpenShift

O OpenShift [35] é um PaaS lançado pela Red Hat em 2011, no entanto em 2012, tornou-se uma solução *open-source* e *multi-cloud* de certa forma semelhante a plataforma CloudFoundry. Até ao momento, o OpenShift suporta um grande número de ambientes de execução: Java, Ruby, PHP, Perl, Python, Node.js, etc.; *frameworks*: CakePHP, Django, Spring, Sinatra, Rails etc.; base de dados: MySQL, Redis, PostgreSQL, SQLite; sendo o único PaaS que suporta totalmente Java EE6. O OpenShift possui uma grande comunidade de utilizadores bastante participativos utilizando diversos repositórios públicos no GitHub onde se encontra alojada toda a documentação e o código fonte da plataforma. Para a interação por parte de utilizadores, existem diversas interfaces como o próprio interface gráfico *web*, uma CLI, um *plugin* para Eclipse, Git e uma API RESTFul. Em semelhança ao Heroku, o OpenShift promove o Git como ferramenta para o *deployment* e controlo de revisão do código fonte das aplicações. O modelo de negócio disponibilizado pelo OpenShift é realizado através de planos de subscrição *pay-per-use*. O plano *FreeShift* é atraente para os utilizadores que pretendem testar a plataforma de forma gratuita mas estando limitado a 3 *small gears*. Por outro lado, o plano *MegaShift* oferece as funcionalidades de recuperação em caso de falhas, balanceamento de carga, segurança e escalabilidade através de um maior número de *small* ou *medium gears*. No contexto do OpenShift, um *gear* é uma porção limitada de memória e espaço em disco. Um *gear* poder ser de tipo *small gear*, limitado a 512MB de memória RAM e 1GB de espaço em disco, ou de tipo *medium gear*, limitado a 1GB de memória RAM e 1GB de espaço em disco. O utilizador tem a oportunidade para escalar horizontalmente a aplicação até ao número máximo de 16 *gears* por detrás de um balanceador de carga. A nível de infraestrutura, a oferta pública do OpenShift é arquitetada no topo de servidores da AWS com sistema operativo Red Hat Enterprise Linux (RHEL). Porém, sendo um PaaS *multi-cloud*, o OpenShift poderá ser assentado sobre

qualquer outra oferta de infraestrutura pública ou privada.

Figura 2.20: AWS Beanstalk

O AWS Elastic Beanstalk [36] surgiu em 2010 sendo um PaaS construído exclusivamente sobre o serviço de IaaS da Amazon: EC2, S3, SimpleDB, Elastic Load Balancing, etc. De momento, o Beanstalk ainda se encontra numa versão beta fornecendo aos utilizadores ambientes de execução Java, PHP, .NET, base de dados NoSQL, através do Amazon DynamoDB, e base de dados MySQL, SQL Server ou Oracle, através do Amazon RDS. Para interação por parte de utilizadores, existem diversas interfaces como o próprio interface gráfico *web*, uma CLI, um *plugin* para Eclipse e Visual Studio, um SDK (Software Development Kit), Git e uma API RESTFul bastante documentada. O modelo de negócio suportado pelo Beanstalk é puramente *pay-per-use* onde o utilizador tem acesso às funcionalidades de recuperação em caso de falhas, balanceamento de carga, segurança e escalabilidade. Analogamente aos restantes modelos de negócio da AWS o utilizador é cobrado em relação ao tempo alocado por cada instância, balanceador de carga, armazenamento para a aplicação e a largura de banda de saída dos servidores, sendo que a largura de banda de entrada é gratuita. Uma característica particular do Beanstalk é o de permitir selecionar a localização geográfica do servidor onde será hospedada a aplicação, desde dos Estados Unidos, Irlanda ou Japão. Assim sendo, a linha que separa a oferta de PaaS da oferta de IaaS da Amazon é bastante ténue.

Figura 2.22: Google App Engine

O GAE (Google App Engine) [37] foi lançado em 2008 pela Google como uma versão de *preview* sendo que apenas em 2011 surgiu como uma versão estável e final. O GAE suporta como ambientes de execução: Java, Python, Go; *frameworks*: Spring, Django, Struts; e base de dados proprietárias: Google Cloud SQL e Google Datastore; sendo que a última opera com uma sintaxe proprietária denominada de GQL. Este tipo de abordagem pode trazer obstáculos para portabilidade das aplicações devido às tecnologias proprietárias que a plataforma da Google suporta. Em contrapartida, o GAE possui uma excelente integração com várias APIs Google nomeadamente de autenticação OAuth, correio eletrónico, *instant messaging*, URL (Uniform Resource Locator) Fetch, memcache, Blobstore, Mapreduce, etc. Para a interação por parte de utilizadores, o GAE disponibiliza diversas interfaces como o próprio interface gráfico *web*, uma CLI, um SDK (Software Development Kit), um *plugin* para Eclipse e uma API RESTFul. A documentação disponibilizada é extensa cobrindo todo o tipo de funcionalidades que um utilizador poderá

usufruir da plataforma. O modelo de negócio utilizado pelo GAE é *pay-per-use*, no entanto, revela-se bastante complexo devido a contabilizar todo o tipo de recursos e operações que as aplicações hospedadas poderão efetuar. Desta forma, o utilizador é cobrado em relação ao tempo alocado por instância, armazenamento da aplicação, correio eletrónico enviado, espaço utilizado por *logs*, mensagens XMPP (Extensible Messaging and Presence Protocol), largura de banda de saída dos servidores e todas as operações efetuadas às bases de dados. Além disso, existem limites por minuto ou por hora para todos os pedidos efetuados às APIs nativas dos vários serviços disponibilizados pela Google. Com este género de modelo, o utilizador necessita de ter em atenção a qualidade e o desempenho das suas aplicações *web* no processo de desenvolvimento de forma a ser o mais benéfico possível a nível de faturação. Analogamente ao Beanstalk, o GAE subsiste sobre os recursos de infraestrutura da própria organização, neste caso a Google.

As Tabelas 2.1 e 2.2, resumem as principais características de cada plataforma aqui detalhadas.

Tabela 2.1: Resumo das características dos PaaS analisados

Plataformas	**Linguagens**	**Frameworks**	**Base de Dados**	**Interfaces de Utilizador**	**IaaS**
CloudBees	Java JRuby Scala	Grails Java Web Play! Spring Rails	MySQL	Web GUI SDK API *Plugin* Eclipse	AWS HP OVH.com IaaS privado
CloudFoundry	Java Node.js Ruby	Grails Java Web Spring Rails Sinatra Node	MongoDB MySQL PostgreSQL Redis	CLI API *Plugin* Eclipse	*Multi-Cloud*
Heroku	Java Node.js Ruby Python	Grails Django Spring Rails Sinatra	*Addons:* PostgreSQL MySQL Redis etc.	Web GUI CLI Git API *Plugin* Eclipse	AWS
OpenShift	Java Ruby PHP	JavaEE6 Spring Django	MongoDB MySQL PostgreSQL	Web GUI CLI Git	*Multi-Cloud*

	Python Perl	Rails Sinatra Zend	SQLite	API *Plugin* Eclipse	
AWS Beanstalk	Java .NET PHP	-	DynamoDB RDS	Web GUI CLI SDK Git API *Plugin* Eclipse *Plugin* Visual Studio	AWS
Google App Engine	Java Python Go	Spring Django Struts	Cloud SQL Datastore	Web GUI SDK API *Plugin* Eclipse	Google

Tabela 2.2: Resumo das características dos PaaS analisados (continuação)

Plataformas	**Taxonomia**	**Monitorização**	**SLA**	**Modelos de Negócio**
CloudBees	Instância *App-Cell*	Web GUI New Relic AppDynamics *Logs*	Estado do Serviço: http://status. cloudbees.com/	Planos: *Free* *Enterprise*
CloudFoundry	Instância	API *Logs*	N/A	N/A
Heroku	*Web Dyno* *Worker Dyno*	New Relic *Logs*	Estado do Serviço: https://status. heroku.com/	*Pay-per-use*
OpenShift	*Gear*	Web GUI *Logs*	N/A	Planos: *FreeShift* *MegaShift*
AWS Beanstalk	Instância	Web GUI CloudWatch *Logs*	AWS SLA[4] 99.95%	*Pay-per-use*
Google App Engine	Instância	Web GUI *Logs*	GAE SLA[5] 99.95%	*Pay-per-use*

Da mesma forma, foram investigados outros fornecedores de PaaS existentes no mercado que oferecem ambientes de execução para linguagens de programação específicas. A Tabela 2.3 agrupa as plataformas estudadas resumindo as suas posições no mercado.

Tabela 2.3: Outras plataformas existentes

Linguagens	**Plataformas**
Java	AppFog, AppHarbor, Cumulogic, DotCloud, IronFoundry, Microsoft Azure, Playapps, WSO2 Stratos
Node.js	AppFog, DotCloud, Joyent, Nodejitsu Nodester
PHP	AppHarbor, DotCloud, EngineYard, IronFoundry PHPFog, Microsoft Azure
Python	AppFog, DotCloud, IronFoundry
.NET	AppHarbor, IronFoundry , Microsoft Azure
Ruby	AppFog, AppHarbor, DotCloud, EngineYard IronFoundry, Microsoft Azure
Erlang	IronFoundry

2.4.3 *Software-as-a-Service*

No nível mais alto da pilha de *cloud computing* surge o *Software-as-a-Service* que, em termos elementares, é definido por aplicações que são entregues através da Internet. Como cliente de SaaS, o mesmo não necessita de instalar nenhum *software* no seu terminal, nem de se preocupar onde o mesmo se encontra hospedado ou que linguagem de programação foi utilizada no seu desenvolvimento. Este conceito de entrega de serviços não é novo remontando à década de 60 e 70. Porém, apenas na década de 90 com a explosão verificada devido ao surgimento da *world wide web*, este paradigma chegou às grandes massas. Algumas das aplicações variavam desde do correio eletrónico, processadores de texto ou simples calendários [38]. Com a introdução do *cloud computing* nos anos 2000, brotaram diversos fornecedores como a Salesforce.com, com a sua aplicação de CRM (Customer Relationship Management), a Google, com os seus diversos serviços, Gmail ou Google Maps, o Twitter, com a sua rede social e a Amazon, com a sua plataforma de comércio *online* operando sobre infraestruturas de *cloud*. Em certos casos, os fornecedores de SaaS estão alheios a infraestrutura ou plataforma contratada colocando a sua equipa técnica apenas focada na melhoria e desenvolvimento de novos produtos para os seus clientes. A Figura 2.23 elucida

a cadeia de valor do mercado de SaaS desde da infraestrutura ou plataforma *cloud* até ao cliente final.

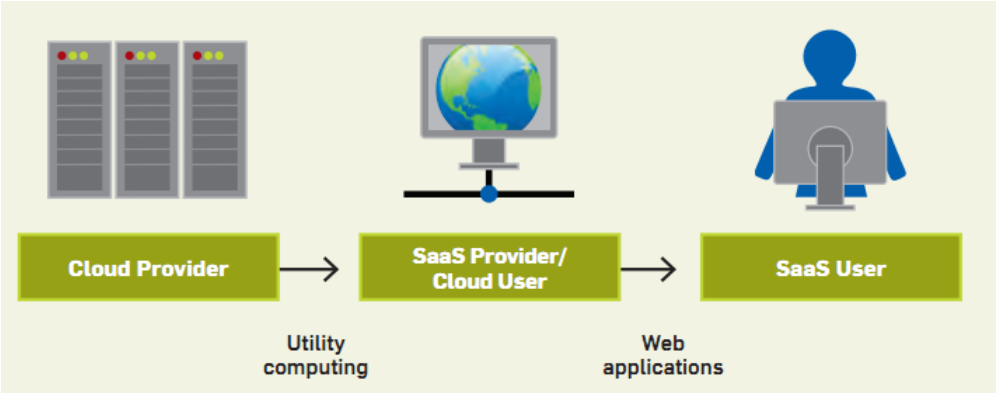


Figura 2.23: Cadeia de valor do mercado de SaaS [2]

A entrega de aplicações através da Internet aliciou igualmente organizações de diversas dimensões, que consideraram a migração ou desenvolvimento de aplicações para a *cloud* como sendo uma oportunidade de negócio ou de mera reestruturação de equipas e custos. Existem vantagens e desvantagens neste processo. Inicialmente é necessário eleger quais aplicações fazem mais sentido envolver e posteriormente quais fornecedores, de IaaS ou de PaaS, são mais adequados para suportar a aplicação em causa com o plano de negócio que a organização se depare como mais vantajoso. Para a eleição das aplicações é indispensável compreender as suas características, o âmbito ou a duração de tempo em que serão utilizadas. Portanto, aplicações com grande interação *web* ou móvel, aplicações com o propósito de serem utilizadas durante um curto espaço de tempo devido a um projeto, ou aplicações que poderão estar sujeitas a picos de procura bastante frequentes, serão as principais candidatas a serem migradas ou desenvolvidas para um ambiente de *cloud*. No entanto, existem situações em que este processo não será o mais adequado ou não irá trazer vantagens a nível económico para a organização detentora do *software*. É caso de aplicações que transacionam dados críticos sujeitos a regulações ou barreiras legais, ou então aplicações que podem ser facilmente hospedadas nas premissas da organização devido a infraestrutura privada já existente [13].

Como é óbvio neste nível da pilha, a quantidade de fornecedores é imensa desde do Yahoo, Google, Salesforce.com, SAP, Microsoft entre outros. Dessa forma será feita uma

síntese dos principais.

Figura 2.25: Google Apps

A gigante Google [39], uma líder no mercado de SaaS, fornece diversas aplicações orientadas às grandes massas, organizações e área da educação. Em 2006 foi lançado o seu serviço mais popular, o Gmail, um cliente de correio eletrónico com suporte inicial de 2 GB de armazenamento. Ao longo dos últimos anos, surgiram um imenso número de serviços que reforçaram o catálogo Google Apps suportando as mais diversas atividades dos clientes. Desses serviços alguns foram evoluindo, outros permaneceram ativos e outros foram eliminados ou absorvidos. O vasto catálogo fornece aplicações desde do Google Maps, uma aplicação com suporte a mapas mundiais, o Google Docs, para partilha de documentos através de um editor de texto, o Google Talk, um *instant messaging*, o Google Drive, um serviço de armazenamento e de sincronização de arquivos *online*, entre outros. Todos estes serviços possuem APIs especificadas onde uma grande comunidade de utilizadores participa. Através destas interfaces, é possível integrar facilmente os serviços Google com outras aplicações existentes.

Figura 2.27: Salesforce.com

A Salesforce.com [40] surgiu no mercado do *cloud computing* por volta de 2006 possuindo uma infraestrutura *multi-tenant* semelhante ao que a AWS e a Google suportam. Nos dias de hoje todos os rendimentos da Salesforce.com são baseados no mercado de *cloud* mantendo um crescimento anual bastante razoável que permitiu a aquisição de diversas organizações das TIs, exemplo da célebre plataforma Heroku. No entanto, a solução que proporcionou esta expansão e reconhecimento foi de facto a aplicação de CRM que é líder no mercado de SaaS empresariais. A Salesforce.com CRM consiste em 2 elementos: The Sales Cloud, uma aplicação para as organizações gerirem os seus recursos humanos e processos mais eficazmente, e The Service Cloud uma plataforma para colaboração em tempo real, partilha de informação empresarial, contactos e processos entre departamentos ou parceiros. Todos estes serviços podem ser completamente integrados com o Google Apps, Facebook ou LinkedIn através de uma composição de APIs de cada fornecedor.

2.5 Iniciativas de Interoperabilidade

Como mencionado, o principal objetivo deste trabalho emerge da preocupação atual da comunidade sobre os obstáculos da interoperabilidade entre fornecedores de soluções *cloud*,

nomeadamente de *Platform-as-a-Service*. Existem dois principais fatores que originaram este efeito: as soluções de *cloud* não terem sido desenhadas para uma possível agregação, bem como a atual competição que permanece entre os grandes fornecedores devido aos seus interesses na disponibilização de serviços diferenciados [4]. A temática da *inter-cloud* tem progredido nos últimos anos envolvendo vários projetos sem fins lucrativos e organismos de normalização com o objetivo de atingir a interoperabilidade em ambientes *cloud*. Com esta solução, os utilizadores deixariam de estar tão vulneráveis a um fornecedor específico e às suas características proprietárias obtendo a oportunidade de comparar, gerir e migrar informação entre fornecedores heterogéneos [41]. Por outro lado, *start-ups* de soluções *cloud* avistariam a sua posição no mercado reforçada podendo adquirir mais facilmente uma razoável quota de mercado.

A maior parte do esforço que tem vindo a ser efetuado nesta área envolve apenas a camada de *Infrastructure-as-a-Service*. No entanto, mais recentemente, surgiram alguns projetos que subiram na pilha do modelo de serviços investigando como seria exequível agregar a oferta de fornecedores de PaaS. A interoperabilidade entre fornecedores é possível ser atingida através de variadas formas as quais foram e encontram-se a ser exploradas. Tipicamente os estudos realizados realçam 3 tipos de abordagem:

- Gestão unificada/API normalizada
- Modelo de dados comum
- *Cloud Broker*

Como este trabalho se foca na especificação e desenvolvimento de uma camada de abstração que permita uma gestão unificada de vários fornecedores, apenas os estudos que possuem essa abordagem serão destacados. Para a camada de infraestrutura, o OCCI-WG [42] (Open Cloud Computing Interface Working Group) foi formado em 2009 pela OGF (Open Grid Forum) com o objetivo de eliminar o *lock-in* existente com o desenvolvimento de uma API que permita gerir e escalar recursos computacionais de vários fornecedores de IaaS. Como resultado deste estudo, surgiu uma interface RESTful denominada de OCCI que torna possível ao utilizador desenvolver ferramentas que o possam auxiliar na gestão de recursos computacionais, deixando como trabalho futuro o desenvolvimento das funcionalidades de monitorização e controlo de faturação. As principais implementações do OCCI suportam até ao momento o OpenStack, OpenNebula, sendo que o grupo afirma também

existir uma implementação para a API do EC2 da AWS. A nível de PaaS e SaaS, a OGF assegura que o OCCI suporta igualmente estes modelos, contudo cada nível possui diferentes requisitos que não podem ser generalizados. Porém, a documentação fornecida pela OGF abrange somente a gestão de recursos de infraestrutura como rede, armazenamento e servidores virtuais. Outro projeto interessante que se foca na área de IaaS é conhecido por Apache DeltaCloud [43]. Esta iniciativa da Red Hat e da Apache Software Foundation surgiu em 2010 com o objetivo de criar uma API *open-source* capaz de interagir com as diversas APIs nativas dos fornecedores de IaaS tanto públicos como privados. Como resultado foi implementada uma API RESTFul em Ruby que abstrai todas as diferenças dos vários fornecedores. Esta interface possibilita ao utilizador criar, gerir e obter informação sobre instâncias de servidores e armazenamento em qualquer um dos fornecedores suportados de forma centralizada. Até ao momento, o DeltaCloud suporta um elevado número de APIs desde EC2 e S3 da AWS, Eucalyptus, IBM SBC, GoGrid, OpenNebula, Rackspace, RHEV-M, RimuHosting, Terremark, VMware vSphere, OpenStack, Microsoft Azure e Google Storage. Sendo um projeto licenciado pela Apache, qualquer indivíduo poderá contribuir na implementação como também no desenvolvimento de clientes HTTP além do cliente Ruby fornecido e disponível para *download*. A Figura 2.28 elucida o funcionamento geral do DeltaCloud interagindo com as diversas APIs dos fornecedores e expondo as funcionalidades suportadas através das interfaces de cliente.

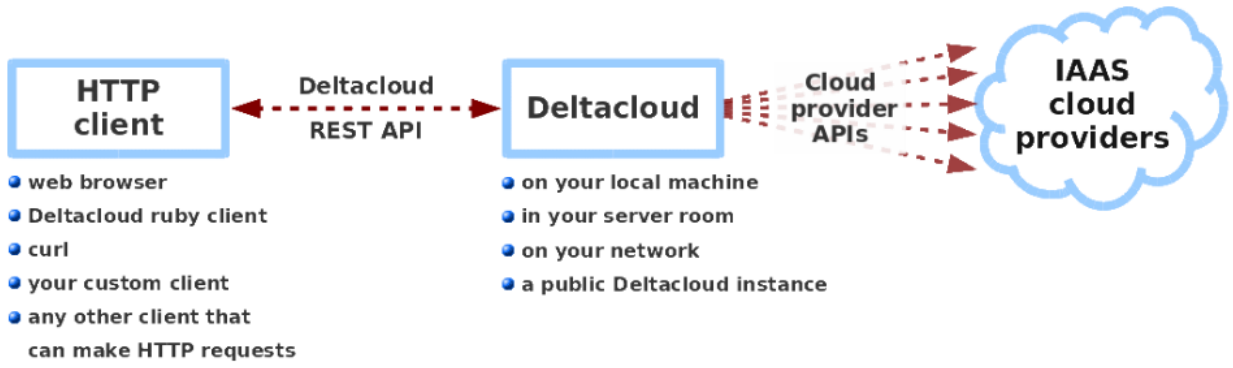


Figura 2.28: Arquitetura Apache DeltaCloud [43]

Subindo um nível no modelo de serviços emergem algumas iniciativas que visam permitir igualmente a interoperabilidade mas entre fornecedores de PaaS. Do lado da comunidade europeia surge o projeto Cloud4SOA [44] do FP7 (Seventh Framework Programme). O projeto Cloud4SOA iniciou-se em 2010 e tem como visão resolver os problemas de interoperabilidade semântica existentes entre fornecedores de PaaS. Dessa forma espera reduzir o

lock-in dos utilizadores facilitando os processos de criação e gestão das aplicações hospedadas em diferentes plataformas. Para isso foi definida uma arquitetura que utiliza modelos e técnicas de semântica de modo a recolher informação sobre cada fornecedor e abstrair as diversas APIs nativas. Com esta *framework*, o utilizador terá acesso às diversas funcionalidades do ciclo de vida de uma aplicação em um ambiente PaaS de forma centralizada. Esta iniciativa apenas finalizará em meados do ano de 2013 sendo que até ao momento possui alguns resultados encorajadores e importantes para esta área de investigação. Em 2010 é igualmente iniciada uma outra iniciativa, no entanto, envolvendo alguns dos titãs do mercado como a Oracle, Rackspace, RedHat, CloudBees, Huawei, Cloudsoft Corporation e Software AG. Porém apenas em finais de Agosto de 2012 este grupo foi anunciado publicamente. Os seus principais objetivos são especificar e desenvolver uma API *open-source* intitulada de CAMP (Cloud Application Management for Platforms) [45], que permite criar, gerir e monitorizar aplicações e bases de dados através de vários fornecedores, de forma centralizada e independente da plataforma. Apesar de existirem diversos fornecedores que oferecem ambientes com diferentes tecnologias, o CAMP aproveita as similaridades inerentes no ciclo de vida aplicacional com o intuito de criar uma solução genérica e unificada. A API especificada [46] é RESTFul suportando JSON (JavaScript Object Notation) como modelo de dados para interagir com os diversos recursos, nomeadamente, plataformas, aplicações, bases de dados, etc. O CAMP igualmente definiu uma taxionomia para as ferramentas de desenvolvimento (Netbeans, Eclipse, Vim, etc.) designando-as por ADE (Application Development Environment). Através de *plugins* que poderão ser implementados, os utilizadores terão a possibilidade de desenvolver o código de fonte de uma aplicação no ADE realizando o respetivo *deployment* para o fornecedor desejado. O CAMP é definido por ser uma solução completamente agnóstica às linguagens de programação e *frameworks*. Portanto, a portabilidade é uma forte aposta de forma a permitir a migração de aplicações entre diferentes fornecedores de PaaS. Para garantir essa portabilidade, foi definido o PDP (Platform Deployment Package) que é um arquivo que reside juntamente com o código fonte de uma aplicação contendo a descrição de dependências, meta-dados, configurações necessárias para a aplicação e certificados de segurança. Com esta iniciativa os diversos parceiros pretendem revolucionar o mercado de PaaS, tendo submetido o CAMP a OASIS (Organization for the Advancement of Structured Information Standards) esperando que daqui por 18 meses seja adotado como a primeira normalização de ambientes PaaS. A reação da comunidade a esta especificação tem sido mista, tecendo alguns elogios mas também algumas dúvidas de como o mercado poderá se adaptar e absorver tal solução. É

necessário esperar que esta especificação amadureça e que a primeira implementação da API esteja disponível para ser utilizada e integrada pelos utilizadores.

Como o serviço e a definição de que é realmente um PaaS ainda se encontra numa fase precoce, não tem existido muitos esforços ou estudos significativos que investigam esta área. Além disso, o mercado está em constante evolução sendo que cada fornecedor suporta diferentes tecnologias e ferramentas de desenvolvimento de forma a se poder diferenciar face aos utilizadores. Contudo, o impacto destas plataformas a nível empresarial poderá verificar-se importantíssimo e atraente para as organizações de diversos negócios. Portanto espera-se que surgem avanços expressivos neste nível de serviço de *cloud* nos próximos anos.

2.6 Service-Oriented Architecture

Os sistemas distribuídos são por base heterogéneos possuindo cada um deles diferentes propósitos, paradigmas, modelo de dados ou linguagens de programação. Hoje em dia, estes sistemas são cada vez maiores e mais complexos, portanto mais difíceis de monitorizar, operar e integrar com novas soluções. De forma a manter a escalabilidade necessária para prorrogar com novos módulos e processos, a flexibilidade é a única solução. O SOA surgiu nos finais da década de 90 sendo uma perspetiva que define como os serviços podem ser concebidos para suportar os requisitos dos utilizadores independentemente do sistema ou infraestrutura subjacente [47]. Alguns dos alicerces do *cloud computing* assentam fortemente em conceitos SOA, considerando-se que a utilização de ambos é a abordagem mais eficaz para alavancar as arquiteturas existentes no seio de qualquer organização.

O SOA não é uma *framework* nem uma tecnologia, não é algo que se possa adquirir ou instalar, apenas visa flexibilizar a manutenção de grandes sistemas e processos sendo orientado em 3 conceitos:

Serviços: Para uma organização, os serviços representam o valor de negócio através das TIs. Eles podem ser simples ou mais complexos através das diversas operações especificadas durante o processo de desenvolvimento. Normalmente os serviços são expostos através de interfaces ou contratos que abstraem os seus detalhes técnicos, por exemplo um WSDL (Web Service Description Language), promovendo a reutilização e a interoperabilidade com outras aplicações. A Figura 2.29 caracteriza um serviço num contexto SOA.

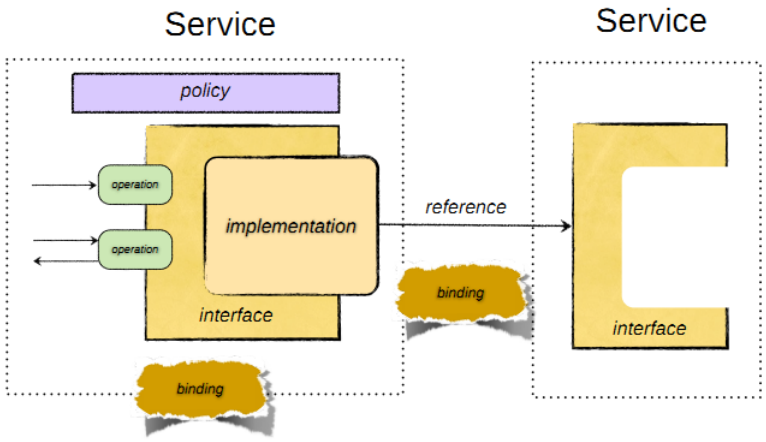


Figura 2.29: Representação de um serviço em SOA [48]

Interoperabilidade: A harmonização é um fator importante para o funcionamento eficaz de qualquer sistema. Cada sistema pode utilizar diferentes paradigmas, protocolos, linguagens de programação ou operar em diversos departamentos e organizações. Por sua vez, o SOA não combate esta heterogeneidade mas sim agiliza-a. Para manter os serviços interoperáveis, o SOA recorre a normalizações como o HTTP que por sua vez é utilizado pelo SOAP (Simple Object Access Protocol) e REST (Representational State Transfer).

Loose Coupling: Para que um sistema seja flexível, escalável e tolerável a falhas, é imperativo minimizar todas as suas dependências desenhando-o de forma modular. Assim sendo, qualquer modificação que seja fundamental efetuar terá um impacto mínimo na arquitetura. Além disso, é possível obter a flexibilidade e a escalabilidade exigida com o intuito de evitar *bottlenecks* ou a expansão caótica do sistema que dificulta a sua manutenção.

Como já foi amplamente discutido, o *cloud computing* tem como princípio fundamental dar aos utilizadores a flexibilidade para contratarem recursos computacionais *on-demand* num paradigma *pay-per-use*. Através da infraestrutura ou plataforma contratada, os utilizadores podem expor aplicações para consumo por parte de clientes e APIs para desenvolvimento por parte de terceiros. A este nível, o SOA e o *cloud computing* partilham o mesmo fundamento ao serem ambos orientados ao serviço, ou seja, ao valor de negócio [49]. No entanto, o SOA tem muito mais para oferecer. Além de promover a agilidade e a reutilização, o SOA também coadjuva eficientemente na eleição dos serviços que sejam

bons candidatos a serem migrados para a *cloud* pública ou a manterem-se nas premissas da organização [21]. A flexibilidade obtida através do SOA aliada às características fundamentais do *cloud computing*, poderá ser a chave para a eficiência das arquiteturas distribuídas a curto e médio prazo.

2.7 Sumário

Neste capítulo foram abordados os conceitos associados ao *cloud computing* bem como o modelo de serviços, focando-se essencialmente no serviço de PaaS e nos principais fornecedores existentes no mercado. As iniciativas de interoperabilidade e as suas considerações servirão de base para a especificação e implementação de uma camada de abstração que agrega as APIs nativas de diferentes ofertas de PaaS. Como será abordado no próximo capítulo, essa camada verificar-se-á útil na unificação e centralização da criação, gestão e monitorização de aplicações em diversas plataformas. Igualmente apresentou-se as vantagens do desenvolvimento de aplicações para a *cloud* orientado aos princípios SOA.

Capítulo 3

Especificação e Desenvolvimento do *PaaSManager*

Neste capítulo, após uma descrição geral das funcionalidades a desenvolver, será descrita a análise efetuada às diversas APIs dos fornecedores de PaaS selecionados, que serviu de base para a especificação e desenvolvimento da solução agregadora *PaaSManager*. Um enquadramento da arquitetura do *PaaSManager* e do funcionamento interno dos vários módulos que constituem o sistema será realizado a partir de diagramas ilustrativos. Por fim, será detalhado o funcionamento de alguns dos serviços baseados em informação de contexto que constituem a *framework* de contexto desenvolvida na PTIN. Alguns destes serviços serão portados para a *cloud* recorrendo a solução desenvolvida.

3.1 Funcionalidades Gerais da Solução

Nesta secção pretende-se enquadrar os requisitos estabelecidos no âmbito deste trabalho, especificando as principais funcionalidades que foram propostas para o desenvolvimento do *PaaSManager*. O sistema teria de abstrair as diferenças entre plataformas heterógenas suportando as operações implementadas pelas APIs nativas dos fornecedores. Assim sendo, um utilizador teria acesso de forma centralizada a um ecossistema de PaaS através de interfaces de cliente desenvolvidas, ou então, através de um qualquer módulo que o próprio poderia integrar. As funcionalidades propostas reúnem:

- Criar e Gerir aplicações e bases de dados

- Obter informação sobre aplicações e bases de dados
- Monitorizar aplicações em tempo-real
- Migrar aplicações entre fornecedores que suportem as tecnologias necessárias

Dessa forma, foram estudadas as ofertas públicas de alguns dos fornecedores de PaaS com o objetivo de entender as disparidades inerentes, nomeadamente, as tecnologias e ferramentas suportadas, as APIs nativas, etc. Como resultado dessa análise, alguma da informação obtida foi já mencionada na secção 2.4.2, sendo que a descrição sobre as APIs das plataformas suportadas será agora descrita na secção 3.2.

3.2 Análise das APIs dos PaaS Suportados

Cada fornecedor de PaaS disponibiliza uma API para os utilizadores interagirem com a plataforma. Através dessa interface são expostas diversas funcionalidades que permitem criar, gerir e obter informação sobre aplicações e bases de dados. Porém, do leque de métodos disponibilizados foi fundamental eleger as funcionalidades idênticas ou semelhantes que são partilhadas entre as várias APIs analisadas. Apenas dessa forma seria possível obter a interoperabilidade necessária para o utilizador interagir de igual forma com diferentes PaaS que possuem diferentes APIs.

Das plataformas investigadas na secção 2.4.2, foram selecionadas para o ecossistema 4 fornecedores: CloudBees, CloudFoundry, IronFoundry e Heroku. Todos os fornecedores possuem diferentes APIs, ferramentas de monitorização e de *deployment*, em exceção do CloudFoundry e do IronFoundry que partilham uma implementação de API idêntica, porém tecnologias discrepantes. Seguidamente serão detalhadas as interfaces e ferramentas suportadas por cada fornecedor para que fosse possível proceder à especificação técnica dos requisitos propostos.

3.2.1 CloudBees

O CloudBees é um PaaS orientado exclusivamente ao desenvolvimento de aplicações que executem sobre uma JVM. Portanto o CloudBees suporta todos os ambientes de execução e *frameworks* fundamentais para os programadores que necessitem de desenvolver uma

aplicação baseada em Java. Esta plataforma fornece dois serviços, o *DEV@Cloud* e o *RUN@Cloud*. Todavia, a API disponibilizada (*api.cloudbees.com/api*) suporta somente o modelo de execução *RUN@Cloud* que permite realizar o *deployment* de uma aplicação para o CloudBees, após a mesma já ter sido criada e testada no ambiente de trabalho do utilizador. Esse processo é sempre efetuado através do *deployment* do ficheiro *.war* de uma aplicação. A API RESTFul disponibilizada não se encontra extensamente documentada no portal do CloudBees. No entanto, através de um cliente disponibilizado no GitHub[1], é possível analisar os diversos métodos suportados. A Tabela 3.1 resume as principais funcionalidades expostas pela API mencionada.

Tabela 3.1: Métodos suportados pela API do CloudBees

Métodos Suportados	***Descrição***
applicationDeployArchive	*deploy* do ficheiro *.war* de uma aplicação
applicationStart	*start* de uma aplicação
applicationStop	*stop* de uma aplicação
applicationRestart	*restart* de uma aplicação
applicationDelete	elimina uma aplicação
applicationScale	*scaling* manual de uma aplicação até ao respetivo número de instâncias
applicationGetSourceUrl	devolve URL ativo de uma aplicação
getApplicationConfiguration	devolve parâmetros de configuração de acesso a uma base de dados
configurationParametersUpdate	insere parâmetros de configuração de acesso a uma base de dados
configurationParametersDelete	elimina parâmetros de configuração de acesso a uma base de dados
configurationParameters	devolve lista de parâmetros de configuração efetuada em uma aplicação
applicationInfo	devolve informação útil sobre uma aplicação
applicationList	devolve lista de aplicações criadas
tailLog	devolve *logs* de uma aplicação
databaseCreate	cria base de dados
databaseDelete	elimina uma base de dados
databaseList	devolve lista das bases de dados criadas
databasePassword	altera a palavra-chave de uma base de dados
databaseSnapshotCreate	cria um *snapshot* dos dados atuais de uma base de dados

[1]https://github.com/cloudbees/cloudbees-api-client

databaseSnapshotDelete	elimina um *snapshot* de uma base de dados
databaseSnapshotDeploy	*deploy* de um *snapshot* de uma base de dados
databaseSnapshotList	devolve lista dos *snapshots* criados de uma base de dados

Os métodos detalhados dão ao utilizador a oportunidade para criar, gerir e obter informação sobre aplicações e bases de dados. No caso da monitorização, apesar do CloudBees disponibilizar na sua interface *web* as estatísticas de monitoria sobre uma aplicação, não existe nenhum método da API que permita recolher este tipo de métricas. Porém, o CloudBees detém um ecossistema de SaaS vindo de parcerias efetuadas com diversas organizações permitindo serem adicionados novos módulos para o processo de desenvolvimento ou serem obtidas diversas informações sobre as aplicações criadas. Neste caso, surgem uma parceria com 2 APMs (Application Performance Management), o NewRelic e o AppDynamics. O NewRelic é de fato o APM mais célebre e completo do mercado fornecendo várias métricas sobre aplicações e bases de dados *on-demand*. Em cada instância de servidor onde executa a aplicação, é instalado um agente que recolhe estatísticas em tempo-real sobre o ambiente que suporta uma aplicação. Essa informação é útil para o utilizador melhorar ou solucionar eventuais problemas relacionados com as suas aplicações. O NewRelic possui uma API RESTFul disponibilizada no GitHub[2] com toda a documentação necessária para compreender o funcionamento e as métricas suportadas. A Tabela 3.2 resume as métricas devolvidas pela API.

3.2.2 CloudFoundry

Por sua vez, o CloudFoundry suporta diversas tecnologias desde de Java, Ruby, Node.js e respetivas *frameworks*. A API RESTFul disponibilizada (*api.cloudfoundry.com*) permite realizar o *deployment* de uma aplicação para o CloudFoundry, após a mesma já ter sido criada e testada no ambiente de trabalho do utilizador. Ao executar esse processo é fundamental declarar a *framework* que é usada pela aplicação para que o CloudFoundry prepare o respetivo ambiente de execução. A interface não se encontra documentada,

[2]https://github.com/newrelic/newrelic_api

Tabela 3.2: Métricas de monitorização suportadas pela API do NewRelic

Métricas Suportadas	***Descrição***
Apdex	Mede a experiência do utilizador numa escala de 0 a 1 baseando-se no Response Time do acesso a uma aplicação
CPU	Percentagem de utilização de CPU (%)
Memory	Memória RAM utilizada (MB)
Response Time	Tempo de resposta de acesso (ms)
Throughput	Atividade das transações *web* (rpm)
Database	Percentagem de utilização de uma base de dados (%)

porém, através de um cliente disponibilizado no GitHub[3], é possível analisar os diversos métodos suportados. A Tabela 3.3 resume as principais funcionalidades expostas pela API.

Tabela 3.3: Métodos suportados pela API do CloudFoundry

Métodos Suportados	***Descrição***
createApplication	cria aplicação
uploadApplication	*deploy* do código fonte de uma aplicação
startApplication	*start* de uma aplicação
stopApplication	*stop* de uma aplicação
restartApplication	*restart* de uma aplicação
deleteApplication	elimina uma aplicação
deleteAllApplications	elimina todas as aplicações criadas
updateApplicationInstances	*scaling* manual de uma aplicação até ao respetivo número de instâncias
updateApplicationEnv	atualiza as variáveis de ambientes de uma aplicação
updateApplicationMemory	atualiza a memória RAM de uma aplicação
getDefaultApplicationMemory	devolve valor de omissão de memória RAM necessária para uma *framework*
getApplicationInstances	devolve informação útil sobre as instâncias de uma aplicação
getApplication	devolve informação útil sobre uma aplicação
getApplications	devolve lista das aplicações criadas
getApplicationStats	devolve estatísticas em tempo-real de uma aplicação

[3]https://github.com/cloudfoundry/vcap-java-client

getCrashes	devolve erros ocorridos nas instâncias de uma aplicação
createService	cria base de dados
deleteService	elimina uma base de dados
deleteAllServices	elimina todas as bases de dados criadas
bindService	realiza a associação de uma base de dados a uma aplicação
unbindService	elimina a associação de uma base de dados a uma aplicação
getService	devolve informação útil sobre uma base de dados
getServices	devolve lista das bases de dados criadas
updateApplicationServices	atualiza as bases de dados associadas a uma aplicação
getServiceConfigurations	devolve parâmetros de configuração de acesso a uma base de dados

Os métodos detalhados dão igualmente ao utilizador a capacidade para criar, gerir e obter informação sobre aplicações e bases de dados. Ao contrário do CloudBees, a API nativa do CloudFoundry disponibiliza estatísticas em tempo-real sobre cada instância de servidor onde executa a aplicação. Portanto, essa informação pode ser invocada de modo a se observar o comportamento atual da aplicação como também para se solucionar eventuais problemas relacionados com a mesma. A Tabela 3.4 resume as métricas devolvidas pela API.

Tabela 3.4: Métricas de monitorização suportadas pela API do CloudFoundry

Métricas Suportadas	***Descrição***
Disk Quota	Espaço em disco total alocado para uma aplicação (GB)
Disk	Espaço em disco utilizado (MB)
Usage CPU	Percentagem de utilização de CPU (%)
Memory Quota	Memória RAM total alocada para uma aplicação (MB)
Memory	Memória RAM utilizada (MB)
Uptime	Tempo de atividade da instância (horas:min:segs)

3.2.3 IronFoundry

Em comparação ao CloudFoundry, o IronFoundry adiciona o suporte de um maior número de tecnologias desde de .NET, PHP, Python, Erlang, e respetivas *frameworks*. A API RESTFul disponibilizada é idêntica à fornecida pelo CloudFoundry, todavia, com um *end-point* diferente (*api.ironfoundry.me*). Através dos métodos expostos o utilizador tem a possibilidade para criar, gerir, obter informação sobre aplicações e bases de dados, além das estatísticas em tempo-real das instâncias onde executa uma aplicação.

3.2.4 Heroku

O Heroku é um PaaS orientado a diversas linguagens como Java, Python, Ruby, PHP ou Node.js. Ao contrário das outras plataformas, a API disponibilizada (*api.heroku.com*) não suporta nenhum método que permita realizar o *deployment* de uma aplicação. Como alternativa é imprescindível o utilizador possuir alguma experiência na ferramenta Git de forma a executar as diversas ações de gestão de código fonte. Ao criar uma aplicação, o Heroku fornece um repositório remoto para que seja realizado o *deployment* através de um comando *git-push*. A API RESTFul[4] encontra-se extensivamente documentada sendo que os métodos podem ser testados *online*. Por sua vez, é igualmente disponibilizado um cliente no GitHub[5] para testes e desenvolvimento. A Tabela 3.5 resume as principais funcionalidades expostas pela API.

Tabela 3.5: Métodos suportados pela API do Heroku

Métodos Suportados	***Descrição***
createApp	cria aplicação
setMaintenanceMode	se o valor colocado for "1" é efetuado o *stop* de uma aplicação, caso o valor seja "0" é efetuado o *start* da mesma
restart	*restart* de uma aplicação
destroyApp	elimina uma aplicação
appExists	devolve se uma aplicação em concreto já foi criada
renameApp	altera o nome de uma aplicação
addConfig	insere variáveis de ambiente em uma aplicação
removeConfig	elimina variáveis de ambiente de uma aplicação

[4]https://api-docs.heroku.com/
[5]https://github.com/heroku/heroku.jar

listConfig	devolve lista de variáveis de ambiente de uma aplicação
addCollaborator	adiciona um utilizador responsável por uma aplicação
removeCollaborator	elimina um utilizador responsável por uma aplicação
listCollaborators	devolve lista de utilizadores responsáveis por uma aplicação
getApp	devolve informação útil sobre uma aplicação
listApps	devolve lista das aplicações criadas
getReleaseInfo	devolve informação sobre uma *release* de uma aplicação
listReleases	devolve lista de *releases* de uma aplicação
rollback	reverte uma aplicação a uma *release* anterior
getLogs	devolve *logs* de uma aplicação
addAddon	associa um *addon* a uma aplicação
removeAddon	elimina a associação de um *addon* a uma aplicação
listAppAddons	devolve lista de *addons* associados a uma aplicação
restartProcessByName	*restart* de um processo pelo nome configurado
restartProcessByType	*restart* de um processo pelo tipo estipulado (*web* ou *worker*)
scaleProcess	*scaling* manual de uma aplicação até ao respetivo número de processos
listProcesses	devolve lista de processos de uma aplicação

Os métodos detalhados dão ao utilizador a possibilidade para criar, gerir e obter informação sobre aplicações e bases de dados. Analogamente ao CloudBees e através do amplo de catálogo de *addons*, é possível recolher estatísticas em tempo-real sobre uma aplicação através do NewRelic. As métricas suportadas são idênticas ao que foi resumido anteriormente na Tabela 3.2.

3.3 Especificação da Solução

Nesta secção procede-se à descrição da arquitetura proposta bem como dos respetivos módulos que foram especificados para o desenvolvimento do *PaaSManager*. Após terem sido analisadas as APIs dos diversos fornecedores, serão igualmente descritos os vários métodos escolhidos e implementados que dão ao utilizador uma experiência completamente transparente e independente da plataforma suportada.

3.3.1 Arquitetura Lógica

No desenho da arquitetura do *PaaSManager* teve-se em atenção a modularidade preponderante que permite que todo o sistema continue em completo funcionamento mesmo que alguma API de um fornecedor ou uma API de monitorização não esteja a operar corretamente. Consequentemente, cada API foi implementada por diferentes módulos de *software* e gerida por entidades únicas orientadas aos grupos de serviços de gestão e de informação, respeitando os princípios IoC/DI (Inversion of Control/Dependency Injection). Por fim, uma interface RESTFul expõe as várias operações especificadas para serem invocadas. Na Figura 3.1 observa-se os diversos módulos que integram o *PaaSManager* como as iterações existentes entre os mesmos.

Toda a implementação foi realizada em Java recorrendo à plataforma JavaEE6 (Java Enterprise Edition 6) devido à ótima integração com EJBs (Enterprise Java Beans), *web services* e camadas de persistência. O servidor aplicacional JBoss 7.1.1 foi o servidor selecionado para suportar o *PaaSManager* no ambiente de desenvolvimento e teste. Para clareza e organização do grande número de dependências, a ferramenta Maven foi utilizada para a gestão do projeto. Por fim, todo o código fonte do *PaaSManager* reside em diversos repositórios remotos Git que possuem *branches* de versões estáveis e de versões de teste. Na secção 3.4 os vários componentes que constituem a arquitetura especificada na Figura 3.1 serão detalhados.

3.3.2 Métodos Suportados pelo *PaaSManager*

O *PaaSManager* foi desenhado com o propósito de centralizar as diferentes ofertas de fornecedores de PaaS existentes no mercado. Para que isso fosse exequível foram estudadas as APIs expostas pelas plataformas selecionadas. Como resultado, especificaram-se alguns

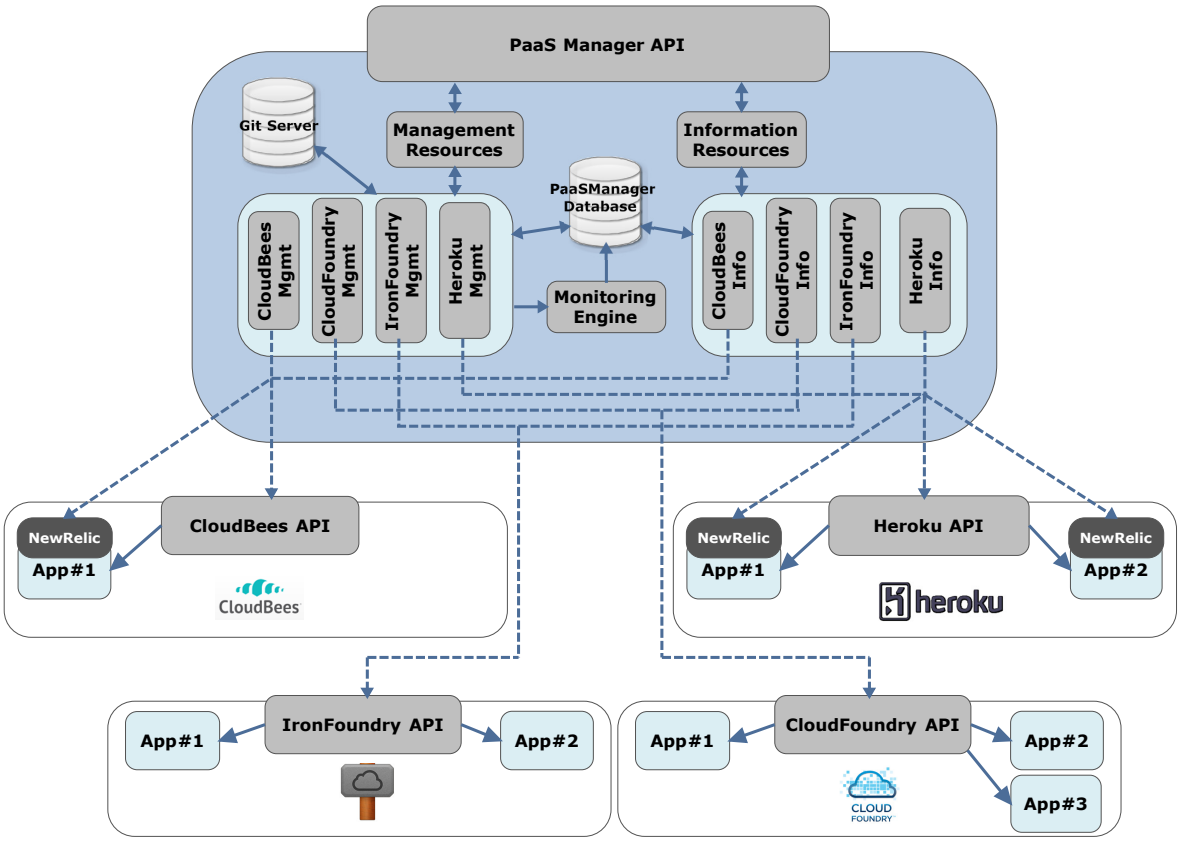


Figura 3.1: Arquitetura do *PaaSManager*

métodos essenciais com base nas semelhanças presentes entre as várias APIs. Nos casos em que a ordem de semelhança fosse reduzida, foi fundamental proceder a desenvolvimentos orientados a dar transparência total ao utilizador. Nos restantes casos, os métodos expostos que não eram partilhados pelos remanescentes PaaS do ecossistema, não foram implementados. Desta forma, a introdução do suporte de uma nova plataforma diminuiria as modificações que teriam de ser realizadas nos diversos módulos facilitando todo o processo de integração. Analogamente foi esta a abordagem utilizada no desenvolvimento do Apache DeltaCloud com o intuito de suportar apenas as operações mais essenciais partilhada entre os vários fornecedores de IaaS. Além da ordem de semelhança, a seleção das operações suportadas pelo *PaaSManager* teve em consideração fornecer ao utilizador os processos fundamentais do ciclo de vida de uma aplicação, como é observado na Figura 3.2.

Portanto, o utilizador tem a possibilidade de controlar uma aplicação desde da sua criação até a sua eliminação de um dos PaaS suportados. Estas operações representadas

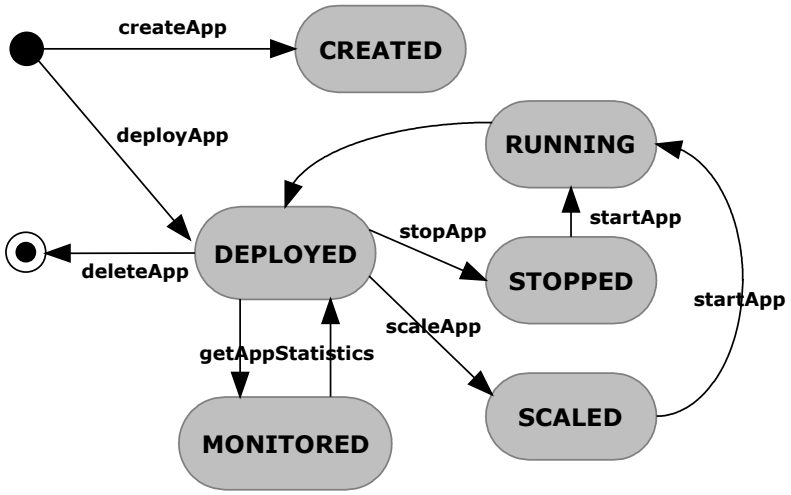


Figura 3.2: Ciclo de vida de uma aplicação

na Figura 3.2 fazem parte de um vasto leque de serviços que foram divididos em 2 grupos que reúnem as várias funcionalidades implementadas:

- Serviços de Gestão
- Serviços de Informação

Cada grupo agrega um conjunto de operações conforme os seus propósitos isolando as funcionalidades relacionadas com a gestão de aplicações e de bases de dados, das funcionalidades de obtenção de informação. Assim sendo, esta fragmentação potencia o desenvolvimento por terceiros de módulos que apenas se foquem nas funções realmente pretendidas. Na Tabela 3.6 observa-se os métodos suportados pelo *PaaSManager*, bem como o mapeamento com as respetivas operações invocadas em cada API de fornecedor que serviram de base para a construção desta camada de abstração.

Tabela 3.6: Métodos Suportados pela API do *PaaSManager*

CloudBees	***CloudFoundry IronFoundry***	***Heroku***	***Serviços de Gestão***	***Descrição***
-	create Application	createApp	**createApp**	cria aplicação em um determinado PaaS

application DeployArchive	upload Application	Git	**deployApp**	*deploy* do código fonte de uma aplicação em um determinado PaaS
application DeployArchive	upload Application	Git	**migrateApp**	efetua a migração de uma aplicação para um determinado PaaS
application Start	start Application	set Maintenance Mode = '0'	**startApp**	*start* de uma aplicação
application Stop	stop Application	set Maintenance Mode = '1'	**stopApp**	*stop* de uma aplicação
application Restart	restart Application	restart	**restartApp**	*restart* de uma aplicação
application Delete	delete Application	destroy Application	**deleteApp**	elimina uma aplicação
application Scale	update Application Instances	scale Processes	**scaleApp**	*scaling* manual de uma aplicação até ao número de instâncias
application DeployArchive	upload Application	Git	**updateApp**	atualiza o código fonte de uma aplicação
database Create	createService bindService	addAddon	**create Service**	associa uma base de dados a uma aplicação
database Delete	deleteService unbindService	removeAddon	**delete Service**	elimina uma base de dados
CloudBees	***CloudFoundry IronFoundry***	***Heroku***	***Serviços de Informação***	***Descrição***
applicationInfo	getApplication getCrashes	listProcesses	**getApp Status**	devolve o estado de uma aplicação
newRelic API	getApplication Stats	newRelic API	**getApp Statistics**	devolve estatísticas em em tempo-real de uma aplicação
applicationInfo	getApplication	listApps	**getApp**	devolve info.

	getCrashes	listProcesses	**Info**	sobre uma aplicação
applicationInfo	getApplication getCrashes	listApps listProcesses	**getApp ListInfo**	devolve lista e info. sobre as aplicações
databaseInfo	getService	listApp Addons	**getService Info**	devolve info. sobre uma base de dados (credenciais de acesso)
databaseInfo	getService	listApp Addons	**getService AppListInfo**	devolve lista e info. sobre as bases de dados associadas a uma aplicação
databaseInfo	getService	listApp Addons	**getService ListInfo**	devolve info. sobre as bases de dados
tailLog	-	getLogs	**getAppLogs**	devolve info. de *logs* de uma aplicação
-	-	-	**getPaaS Offering**	devolve lista de tecnologias e métricas de monitorização suportadas pelos fornecedores de PaaS

Os métodos especificados pelo *PaaSManager* dão ao utilizador a capacidade para controlar todo o ciclo de vida de uma aplicação e respetivas bases de dados. Em certas operações foi necessário proceder a transformações, nomeadamente, suportar Git para o *deployment* e atualização do código fonte devido ao Heroku, como de igual forma agregar diversos métodos da API dos fornecedores para se obter o resultado pretendido. A escalabilidade e respetiva taxonomia foram unificadas sendo agora possível escalar horizontalmente cada aplicação através de mais instâncias.

Em relação à monitorização, cada fornecedor suporta diferentes paradigmas, no caso

do CloudBees e Heroku através do NewRelic, e no caso do CloudFoundry e IronFoundry através da API nativa. Portanto, o método *getAppStatistics* foi projetado de forma a unificar a monitorização devolvendo uma mensagem com um formato único, porém, preenchida com as respetivas métricas suportadas. Além desse método, o *getAppStatus* e o *getAppLogs* devolvem informação valiosa para o utilizador ter acesso ao estado atual de uma aplicação ou às diversas situações ocorridas. Para cumprir a portabilidade transparente entre fornecedores de PaaS, foi implementado o método *migrateApp* que efetua a migração de uma aplicação para uma nova plataforma se a mesma suportar as tecnologias necessárias para a aplicação executar corretamente. Ainda surge o método *getPaaSOffering* que foi desenhado com o objetivo de devolver informação relacionada com as tecnologias e métricas de monitorização suportadas pelas várias plataformas, que será útil para outras operações como também para um dos casos de estudo discutidos no capítulo 4. O funcionamento dos diversos métodos será detalhado ao longo das próximas secções enquadrando-se no desenho e implementação da arquitetura proposta.

3.4 Módulos do *PaaSManager*

Nesta secção procede-se à análise do desenho e da implementação dos vários módulos que constituem o *PaaSManager* apresentado na Figura 3.1. Igualmente serão descritas as interação existentes entre os diversos componentes por forma a compreender o funcionamento global do sistema.

3.4.1 *PaaSManager API*

A API do *PaaSManager* é o elemento que expõe as várias funcionalidades suportadas de forma transparente para o utilizador. A lógica e transformações efetuadas são abstraídas pela interface permitindo uma fácil integração com outros módulos de *software* ou com qualquer cliente HTTP, seja CLI, *browser*, ou cURL. A seleção do desenvolvimento de uma API RESTFul resulta das inerentes vantagens de uma abordagem leve e orientada à *web*. O REST, que suporta o conjunto de princípios arquiteturais fundamentais dos sistemas RESTFul, permite realizar pedidos HTTP através dos vários verbos GET, POST, PUT, DELETE com o intuito de manipular os *resources* pretendidos. Um *resource* é um conceito fundamental do REST sendo considerado como uma entidade física ou abstrata.

Para expor as diversas entidades, são utilizados URIs (Uniform Resource Identifiers) de modo a serem associados endereços HTTP (URLs) de acesso aos *resources*. Por sua vez a manipulação e execução de operações aos próprios *resources* são efetuadas através de pedidos aos URIs definidos, sendo que as respostas devolvidas são tipicamente representadas através de HTML (HyperText Markup Language), XML (Extensible Markup Language) ou JSON [50]. A abordagem RESTful tem vindo a ser amplamente utilizada pelas diversas APIs que se pode encontrar na Internet, seja de fornecedores de soluções de *cloud*, redes sociais, serviços de vídeo *on-line*, etc. Portanto este paradigma foi igualmente adotado para o desenvolvimento da API do *PaaSManager* abraçando as boas práticas do REST. Cada método especificado anteriormente na Tabela 3.6 possui um URI de acesso que dá a possibilidade para manipular entidades através dos verbos GET, POST, PUT e DELETE. Por sua vez, foram definidas 2 entidades *root* **mgmt** e **info**, das quais têm associadas respetivamente mais 2 entidades: **apps** e **services**. Estas associações surgem por um lado para diferenciar as operações de gestão das operações obtenção de informação, e por outro para diferenciar as aplicações das bases de dados. Consequentemente os URIs base foram divididos por:

- /mgmt/apps
- /mgmt/services
- /info/apps
- /info/services

A partir destes endereços são construídos os restantes identificadores de cada operação suportada pelo *PaaSManager*. No desenho dos mesmos foram consideradas as boas práticas definidas por Roy T.Fielding [51] de manter os endereços o mais simples e intuitivo possível com o objetivo de ser pragmático para os utilizadores. As respostas devolvidas pela interface produzem representações de informação em XML e JSON permitindo assim a seleção do formato mais adequado. Pode-se observar na documentação da API (no Apêndice A) os diversos métodos e respetivos parâmetros de entrada, como igualmente alguns exemplos de pedidos e respostas. No âmbito da autenticação e autorização no sistema, em todos os pedidos à API é obrigatório o envio de uma *api-key* que identifica o utilizador registado no *PaaSManager*. Para a autenticação e autorização nos fornecedores de PaaS o modelo adotado, mediante possíveis questões de negócio da PT e comodidade para o utilizador,

foi o do *PaaSManager* atuar como uma entidade mediadora e independente, ou seja, para cada plataforma possuir uma conta e respetivas chaves de acesso à API nativa. Consequentemente, o utilizador final não necessita de se registar em cada um dos fornecedores para usufruir dos serviços. Porém, facilmente seria possível permutar este modelo para um paradigma comparável ao que foi adotado pela API do DeltaCloud, no qual cada utilizador possui previamente uma conta no fornecedor colocando as suas credenciais em todos os pedidos à API.

3.4.2 Serviços de Gestão

O *Management Resources* que integra a Figura 3.1, é o módulo que contém o algoritmo de decisão responsável por executar o método pretendido implementado pelo respetivo *adapter*[6] de PaaS. Ao nível de operações de gestão, surgem 4 *adapters*: CloudBees Mgmt, CloudFoundry Mgmt, IronFoundry Mgmt e Heroku Mgmt. Esses *adapters* são EJBs *stateless* com toda a lógica que implementa os métodos relacionados com os serviços de gestão que são expostos pela API dos respetivos PaaS. As respostas devolvidas pelos *adapters* de PaaS são unificadas, tendo sido definidas classes Java mapeadas para respostas XML através de JAXB (Java Architecture for XML Binding) e para respostas JSON através de Jettison. Além das APIs dos fornecedores de plataformas, os vários *adapters* interagem diretamente com outros elementos fundamentais de forma a cumprir as diversas atividades associadas aos serviços de gestão. Por exemplo, a base de dados central MySQL mantém estado das aplicações criadas, para que não seja necessário enviar em todos os pedidos à API do *PaaSManager* o identificador do fornecedor de PaaS onde a aplicação se encontra hospedada. E por outro lado, um servidor Git, onde é criado um repositório por aplicação possibilitando ao utilizador manter remotamente várias versões do código fonte. Os repositórios Git são essenciais para algumas das operações suportadas pelo *PaaSManager*, nomeadamente, *createApp*, *deployApp*, *udpateApp* e *migrateApp*.

Processo de Criação de uma Aplicação

A Figura 3.3 ilustra o processo ocorrido durante o método *createApp* que é a primeira operação invocada quando um utilizador pretende criar uma aplicação numa determinada plataforma. O pedido recebido pela API do *PaaSManager* é instantaneamente enviado

[6]Padrão utilizado para adaptar as interfaces de cada fornecedor de plataforma.

ao módulo de gestão, o *Management Resources*, que direciona a mensagem para o *adapter* do PaaS pretendido. Por sua vez, a própria lógica do *adapter* cria um repositório Git para a aplicação e invoca o método da API do fornecedor da plataforma para preparar o ambiente de execução. Se o processo concluir com sucesso, alguma informação de estado da aplicação, como o identificador do PaaS, da aplicação e do utilizador responsável, etc., é armazenada na base de dados central sendo devolvida ao utilizador uma resposta em XML ou JSON.

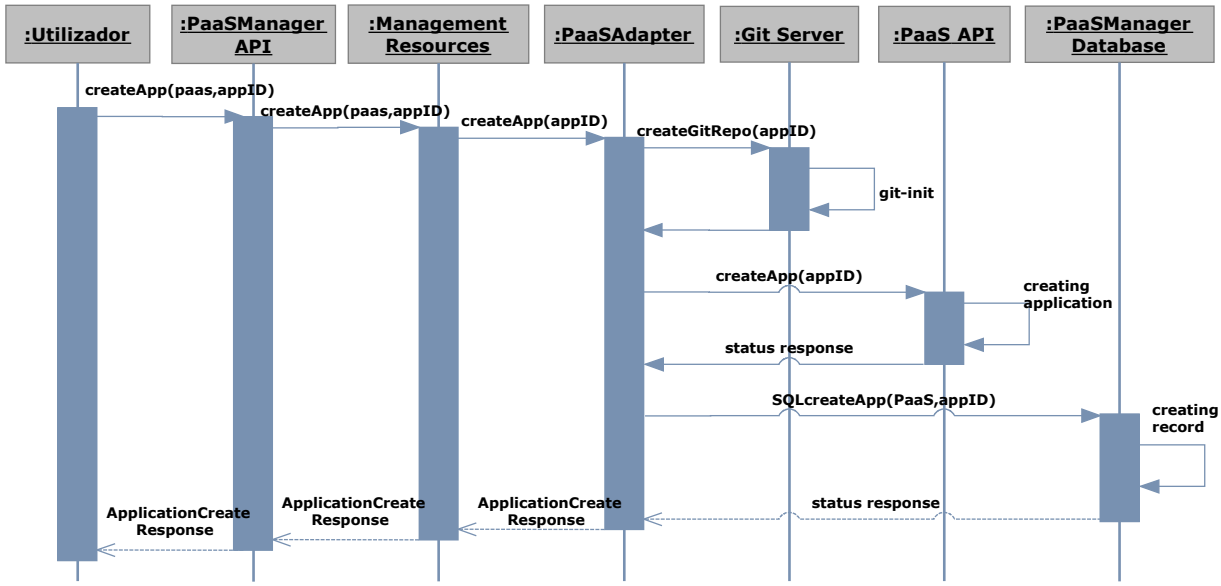


Figura 3.3: Diagrama de sequência no processo de criação de uma aplicação

Processo de *Deployment* de Código Fonte

O *deployment* do código fonte foi unificado para que o utilizador não necessite de se preocupar com as diferentes ferramentas oferecidas pelos fornecedores, nem se terá de enviar para a plataforma um ficheiro *.war* ou todo o código fonte da aplicação. A Figura 3.4 ilustra o processo ocorrido durante o método *deployApp*. O pedido recebido pela API é instantaneamente enviado ao módulo de gestão, *Management Resources*, que realiza uma busca a base de dados central com o objetivo de adquirir a identificação da plataforma onde se encontra criada a aplicação. Através do resultado obtido é invocado o *adapter* do PaaS pretendido, que por sua vez, executa os comandos *git-add* e *git-commit* no repositório da aplicação. Após este processo, é efetuado o *deployment* na plataforma conforme o paradigma associado. No caso do CloudBees, é feita uma pesquisa no repositório até ser

encontrado o ficheiro *.war* da aplicação. Para o CloudFoundry e IronFoundry é realizado o mesmo processo para aplicações baseadas em Java, porém, para as restantes é enviado um ficheiro *.zip* que contém todo o código fonte. Por fim para Heroku é executado um comando *git-push* para o repositório remoto criado exclusivamente para a aplicação. Numa situação de sucesso, é iniciado o motor de monitorização com o objetivo de recolher estatísticas em tempo-real sobre a aplicação, armazenando-as na base de dados central. Finalmente é devolvida uma resposta em XML ou JSON ao utilizador. A estratégia de funcionamento adotada para o motor de monitorização será posteriormente analisada com mais detalhe na secção 3.4.4.

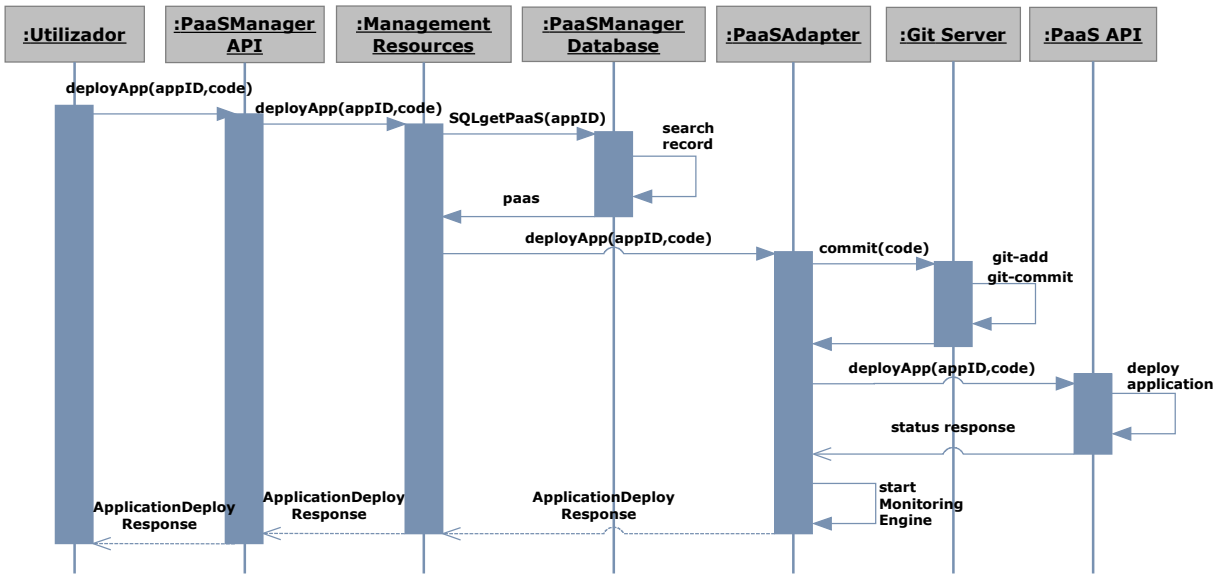


Figura 3.4: Diagrama de sequência no processo do *deployment* do código fonte de uma aplicação

Processo de Migração de uma Aplicação

A portabilidade de uma aplicação para um novo fornecedor torna-se fulcral para os utilizadores que, após analisarem o comportamento da mesma, pretendem testar ou averiguar se é possível obter um melhor desempenho em outro ambiente PaaS. Por outro lado, as questões de negócio também poderão pesar na decisão de migrar para uma plataforma que lhe ofereça outro modelo mais vantajoso para a atividade pretendida. No âmbito deste trabalho, este processo de migração não visa definir nenhum modelo normalizado que possa ser partilhado pelos fornecedores heterogéneos, mas sim adaptar a camada de abstração

ao que já é exposto pelas APIs dos fornecedores. Portanto, a abordagem utilizada obriga a uma prévia análise se a plataforma para onde o utilizador quer migrar a aplicação, suporta as tecnologias necessárias para a mesma executar, por exemplo Java, PHP, etc. Dos fornecedores implementados, apenas o Heroku permite que se efetue a recolha do código fonte através de comandos Git, particularmente, o comando *git-pull*. Dos restantes PaaS do ecossistema, não é permissível aceder ao código da aplicação após o processo de *deployment*. Consequentemente surgiu a necessidade da instalação de um servidor Git central que armazenasse o código fonte das aplicações e respetivas versões. Esta abordagem poderá questionar a escalabilidade da arquitetura, porém, a eliminação de aplicações pelos utilizadores responsáveis irá sempre implicar a supressão dos repositórios associados de forma a manter o sistema limpo de código desnecessário. Por outro lado, de forma a manter o processo de migração minimamente transparente para o cliente, será necessário que o URL da aplicação migrada seja o mesmo podendo ser efetuado através da atualização de DNS (Domain Name System).

A Figura 3.5 resume os passos principais que são executados durante um processo de migração de uma aplicação. Inicialmente é consumado o pedido à API sendo encaminhado ao *Management Resources*. Por sua vez, a base de dados central é inquirida de forma a se obter a identificação da plataforma onde a aplicação está hospedada como também da *framework* de suporte. Após o *adapter* possuir essa informação, é analisado se o PaaS para onde a aplicação será migrada suporta a *framework* necessária. Na circunstância de existir conformidade, o código da aplicação mantido no repositório é portado para a nova plataforma. O estado físico da aplicação no novo PaaS é analisado sendo que, numa situação de sucesso, a aplicação existente na plataforma anterior é eliminada. Para concluir o processo, a informação de estado na base de dados central é atualizada sendo iniciado o motor de monitorização associado ao novo fornecedor. Porém, se o estado físico da aplicação não corresponder ao esperado, a aplicação que já operava no PaaS inicial é mantida em funcionamento sendo devolvida uma resposta de insucesso para o utilizador. O *PaaS Manager* até ao momento apenas suporta migração de aplicações que não possuem persistência devido às limitações que esse caso poderia colocar. Essas limitações englobariam efetuar uma cópia de toda a base de dados da aplicação para um local central sem inserir inconsistências, bem como realizar igualmente uma auto-reconfiguração no código fonte da aplicação para a mesma aceder a nova base de dados criada. Porém, através do método *getServiceInfo* é devolvido ao utilizador as credenciais de acesso a uma base de dados para o mesmo importar ou exportar os dados manualmente

através de ferramentas já existentes.

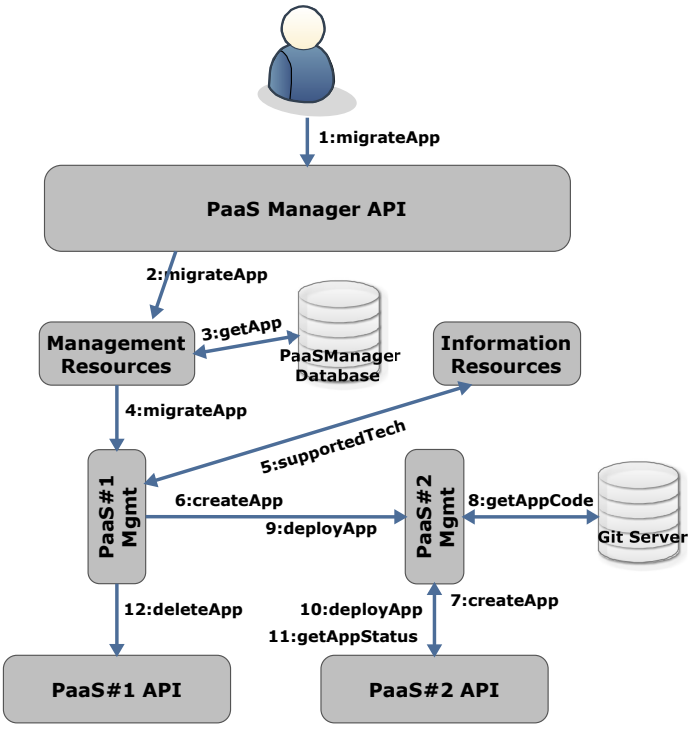


Figura 3.5: Processo da migração de uma aplicação para um novo fornecedor de PaaS

3.4.3 Serviços de Informação

O *Information Resources* presente na Figura 3.1, é o módulo responsável por executar o método pretendido implementado pelo *adapter* do PaaS desejado para as operações de aquisição de informação. Analogamente ao *Management Resources*, surgem 4 *adapters*: CloudBees Info, CloudFoundry Info, IronFoundry Info e Heroku Info. Esses *adapters* são EJBs *stateless* com toda a lógica que implementa os métodos relacionados com os serviços de informação que são expostos pela API dos respetivos PaaS. As respostas devolvidas pelos *adapters* de PaaS são igualmente unificadas sendo definidas em XML e JSON. As operações especificadas dão ao utilizador toda a informação essencial sobre o comportamento das aplicações criadas nas diversas plataformas, podendo ser utilizada para ativação de processos de migração. Além disso, foi desenhado um módulo interno que disponibiliza todas as tecnologias e métricas de monitorização suportadas pelos vários fornecedores de PaaS de forma estática. Essa informação devolvida é utilizada na operação de migração

de aplicações podendo ser igualmente utilizada como forma de informar o utilizador sobre o que cada plataforma lhe poderá oferecer a nível tecnológico. Para a mensagem unificada das tecnologias e métricas suportadas, foi especificado um modelo com 4 tipos de dados: *runtimes*, *frameworks*, *services* e *monitoringMetrics*. Esta especificação permitiu harmonizar toda a informação sobre o que cada plataforma suporta com o intuito da operação ser aplicada em cenários reais, sendo um deles analisado num dos casos de estudo apresentado no capítulo 4. No modelo definido, os 3 primeiros tipos de dados representam as tecnologias suportadas, sendo o último as métricas de monitorização que são disponibilizadas pelos vários fornecedores. Os *runtimes* apresentam as linguagens de programação que cada PaaS suporta, as *frameworks*, como o próprio nome indica, as *frameworks* disponibilizadas, e os *services* as bases de dados. O nome *services* foi selecionado por representar um maior nível de abstração de modo a se poder incluir futuramente no catálogo ferramentas e outros serviços que possam ser associados às aplicações. Por fim, o *monitoringMetrics* representa a lista de métricas suportadas por cada PaaS como foi analisado na secção 3.2.

A título de exemplo refira-se um dos métodos suportados que possibilita ao utilizador obter o estado físico de uma aplicação a executar em uma plataforma. A Figura 3.6 ilustra o processo ocorrido durante o método *getStatus*. De forma a dar um simples entendimento aos utilizadores, foram especificados 4 estados: *running*, *stopped*, *crashed* e *unknown*. No processo de obtenção de estado, o pedido efetuado a API do *PaaSManager* é instantaneamente enviado ao módulo de informação, Informação Resources, que direciona a mensagem para o *adapter* do PaaS pretendido. Por sua vez, a própria lógica do *adapter* invoca o método da API da plataforma associada. Conforme a informação de estado devolvida, a mesma é mapeada para um dos 4 estados definidos com o intuito de unificar a resposta XML ou JSON para o utilizador.

3.4.4 Motor de Monitorização

A monitorização em tempo-real de aplicações permite dar ao utilizador a informação necessária sobre o comportamento das mesmas nos diferentes PaaS suportados. Até ao momento não existe um consenso nem nenhum modelo normalizado de monitorização partilhado entre os maiores fornecedores. Nos últimos anos, os estudos efetuados nesta área tentam definir *frameworks* de monitorização como também um modelo de métricas que se verifique apropriado para uma gestão eficiente de aplicações na *cloud* [52],[53]. A lista de métricas ideais para a monitorização é bastante extensa, incluindo métricas de disponibi-

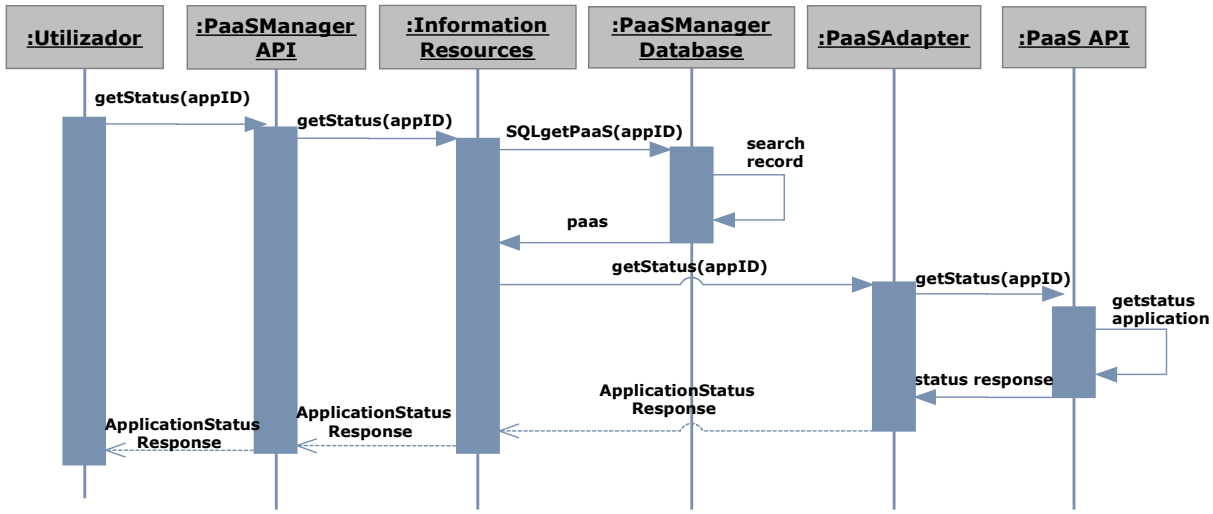


Figura 3.6: Diagrama de sequência no processo de obtenção de estado de uma aplicação

lidade, tempo de resposta, memória RAM, utilização de CPU, acessos a bases de dados, transações *web*, *threads*, sessões de utilizadores, entre outras. Desta lista, as métricas podem ser mensuráveis de forma direta, como por exemplo acessos a bases de dados, sessões de utilizadores, ou por outro lado mensuráveis de forma calculada a partir de 2 ou mais métricas, como é o caso do tempo de resposta, utilização de CPU, etc. [54]. Uma normalização do modelo de monitorização faria com que os SLAs definidos pelos fornecedores fossem comparáveis e o utilizador pudesse testar a mesma aplicação em 2 ambientes heterogéneos com o objetivo de obter informação sobre o comportamento dos ambientes disponibilizados. No entanto, ao nível do serviço de PaaS, cada fornecedor possui diferentes métricas e formas de monitorizar as aplicações sendo que grande parte da informação relacionada com os recursos de infraestrutura é abstraída para o utilizador. Como foi analisado na secção 3.2, o CloudBees e o Heroku possuem parceria com APMs, nomeadamente, o NewRelic que permite a monitorização de aplicações através da instalação de agentes nas instâncias onde as próprias executam. Por outro lado, o CloudFoundry e IronFoundry devolvem outro tipo de estatísticas diretamente através da API nativa.

Neste contexto, ao contrário do que foi realizado nas restantes funcionalidades do *PaaS-Manager*, o motivo do desenvolvimento de um motor de monitorização não foi o de unificar o modelo de métricas mas sim o de disponibilizar ao utilizador as diferentes estatísticas de forma centralizada. Assim sendo, cada fornecedor poderá oferecer mais ou menos valias na opinião do utilizador conforme a informação de monitorização que o sujeito pretende obter.

Desta forma foi especificado um módulo denominado por *Monitoring Engine*, apresentado na Figura 3.1, que efetua a recolha em tempo-real das métricas expostas pelas diferentes plataformas do ecossistema. A informação obtida é depois armazenada na base de dados central para ser devolvida ao utilizador através de pedidos à API do *PaaSManager*.

O processo de recolha e armazenamento na base de dados cabe a cada *adapter* relacionado com os serviços de gestão. Após a aplicação ter sido portada para uma plataforma, é ativada uma *thread* do motor de monitorização que se mantem ativa até que o estado da aplicação esteja em modo *stopped* ou então até que seja eliminada. O processo é definido por uma recolha síncrona efetuada todos os minutos à API do NewRelic ou à API nativa, conforme o PaaS onde a aplicação se encontra hospedada. Na Figura 3.7 observa-se os diversos passos que são executados durante o processo de monitorização.

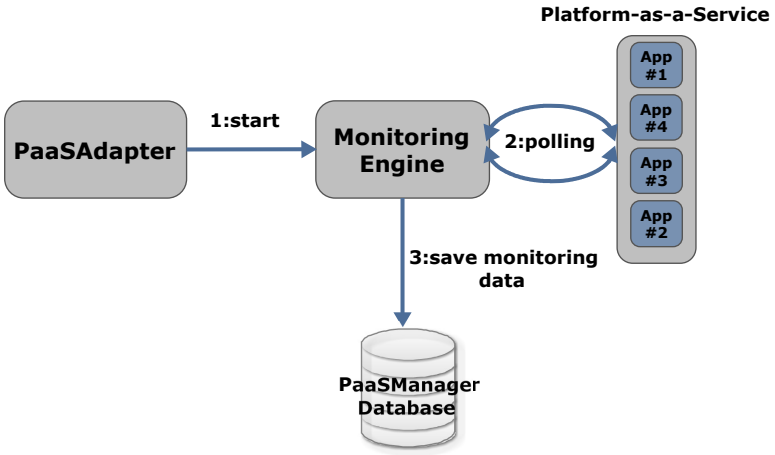


Figura 3.7: Processo de monitorização efetuado pelo *Monitoring Engine*

Enquanto as estatísticas são armazenadas na base de dados, o utilizador pode invocar o método *getStatistics* que faz com que os *adapters* de serviços de informação processem os valores estatísticos devolvendo uma resposta em XML ou JSON. No conteúdo da resposta, são assinaladas as instâncias onde a aplicação executa como as métricas recolhidas conforme o modelo de monitorização da plataforma. Com este tipo de informação, é possível criar diagramas ilustrativos, relatórios ou despoletar eventos para o utilizador obter o conhecimento pormenorizado sobre comportamento das aplicações que detém.

3.4.5 Modelo de Dados

Ao fim de se poder manter o estado das várias aplicações que poderão ser criadas através do *PaaSManager*, surgiu a necessidade de definir uma estrutura de suporte aos vários módulos que constituem o sistema. A construção do modelo teve em atenção a relação que existe entre um utilizador responsável e as aplicações criadas, com o intuito de poderem existir aplicações com identificações idênticas porém de diferentes utilizadores. Por outro lado, cada aplicação poderá estar associada a uma ou mais bases de dados, sendo essa informação útil para algumas das operações de gestão e informação do *PaaSManager*. Para a monitorização foi igualmente indispensável o armazenamento das estatísticas recolhidas para serem posteriormente gerados diagramas ilustrativos e relatórios para o utilizador.

A Figura 3.8 detalha o modelo constituído por 7 entidades. No entanto este modelo é facilmente escalável podendo ser estendido com o suporte a novos PaaS apenas através da criação de entidades orientadas à monitoria em cada plataforma. A entidade *developer* representa o detentor das aplicações registado no sistema sendo composta pelo atributo, *idDeveloper*, e pelo atributo da chave de acesso à API, *apiKey*. Como é natural, cada utilizador poderá deter responsabilidade sobre várias aplicações. Portanto surge a entidade *application* com o atributo *idApplication*, a identificação, *appID*, a representação da *framework* utilizada, *appFramework*, a plataforma onde está hospedada, *paasProvider*, e a chave estrangeira, *idDeveloper*, que associa a aplicação a um utilizador. O atributo *appFramework* é essencial no processo de migração para ser confirmado se o PaaS para onde o utilizador tenciona migrar a aplicação suporta as tecnologias necessárias para ela executar com sucesso. As bases de dados estão representadas pela entidade *service* que é composta pelo atributo *idService*, a identificação, *serviceID*, e a chave estrangeira, *idApplication*, que associa uma base de dados a uma aplicação. A monitorização orientada a cada plataforma fez com que fosse indispensável definir entidades distintas. Apesar de alguns dos PaaS partilharem o mesmo modelo de métricas foi preferível separar os dados recolhidos por plataforma com o intuito de manter a informação mais organizada para a pesquisa. Em cada entidade observa-se a identificação da instância de servidor de onde foram recolhidas as estatísticas, os atributos relacionados com as métricas, o *timestamp* da medição, e a chave estrangeira, *idApplication*, que associa os dados de monitorização a uma aplicação em específico.

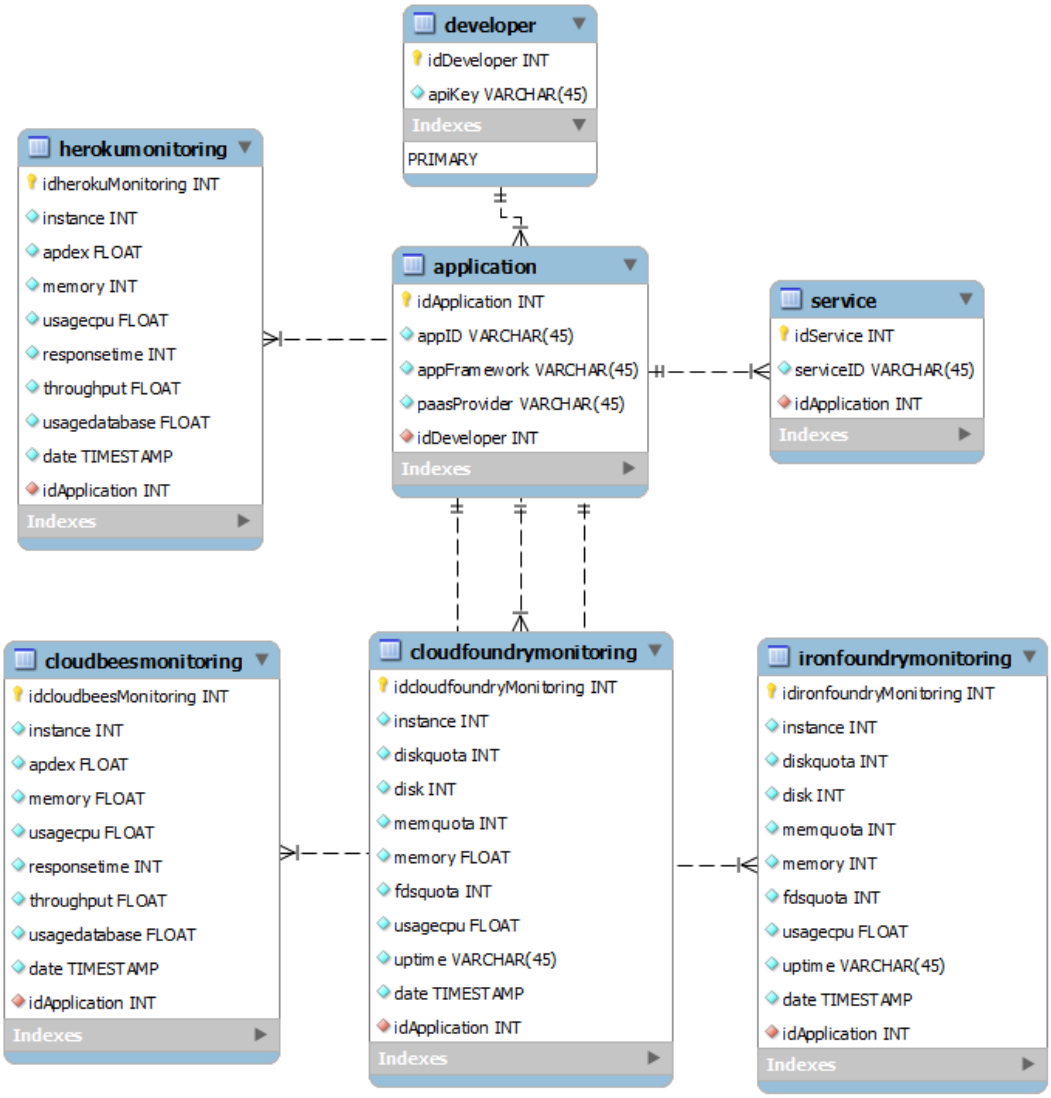


Figura 3.8: Modelo de dados de suporte à operação do *PaaSManager*

3.5 *Context-as-a-Service*

Após a apresentação da arquitetura e funcionamento interno do *PaaSManager*, esta secção analisa uma *framework* de serviços baseados em informação de contexto que foi desenvolvida na PTIN. Este protótipo será parcialmente portado para a *cloud* através do *PaaSManager* num caso de estudo definido no capítulo 4. Consequentemente serão destacados os diversos componentes que constituem o sistema bem como a interação existente entre os vários módulos.

3.5.1 Visão Geral

Context-as-a-Service traz a oportunidade de usar a informação gerada pelos clientes num ambiente móvel para selecionar e entregar conteúdo mais adequado para os mesmos (por exemplo, vídeos, fotos, etc.). Estes meta-dados podem caracterizar qualquer situação do cliente, tal como, a sua posição, gênero, como simultaneamente interesses de musicais ou sociais. Portanto, os sistemas sensíveis a informação de contexto provam ser o futuro dos serviços de Internet móvel para encontrar e agrupar utilizadores com interesses comuns entregando o conteúdo mais adequado [55]. Naturalmente, os espaços públicos poderão tornar-se espaços públicos inteligentes oferecendo uma variedade de serviços móveis que reagem tanto ao ambiente como às condições do cliente. No âmbito do projeto FP7 C-CAST [56] que finalizou em 2010, a PTIN desenvolveu uma *framework* composta de diversos serviços que reagem a informação de contexto de um utilizador. A proliferação de tais serviços móveis pode, em alguns casos, gerar uma sobrecarga na infraestrutura subjacente se não for escalável. A fim de preservar o desempenho adequado, o *PaaSManager* irá permitir portar para um ou mais fornecedores de PaaS alguns dos serviços de contexto que constituem o sistema, como igualmente monitorizar o comportamento dos mesmos de forma centralizada.

3.5.2 Arquitetura

A arquitetura, apresentada na Figura 3.9, é composta por vários módulos que permitem recolher e processar informação relativa ao terminal móvel do cliente. Como componente central e fundamental da *framework* surge um servidor XMPP denominado por *Context Broker*. Este servidor tem um papel ativo nos processos efetuados pelos 3 serviços de contexto: *Context Enabler*, *Group Manager Enabler* e *Content Selection Enabler*. Por fim existe um servidor de conteúdos multimédia (*Content Provider*) e vários serviços externos que podem ser inquiridos de modo a devolver informação de contexto complementar. Dos componentes que constituem a *framework* apenas alguns dos serviços baseados em informação de contexto serão modificados e portados para um ambiente *cloud* através do *PaaSManager*.

3.5.3 Módulos da *Framework* de Contexto

De seguida será apresentado o funcionamento e as características do servidor XMPP *Context Broker* como dos 3 serviços de contexto. Dos componentes que constituem a arquitetura, praticamente todos os módulos da *framework* foram desenvolvidos no âmbito do C-CAST. Portanto é natural que os vários detalhes que envolveram a especificação e o desenvolvimento dos módulos não sejam minuciosamente descritos.

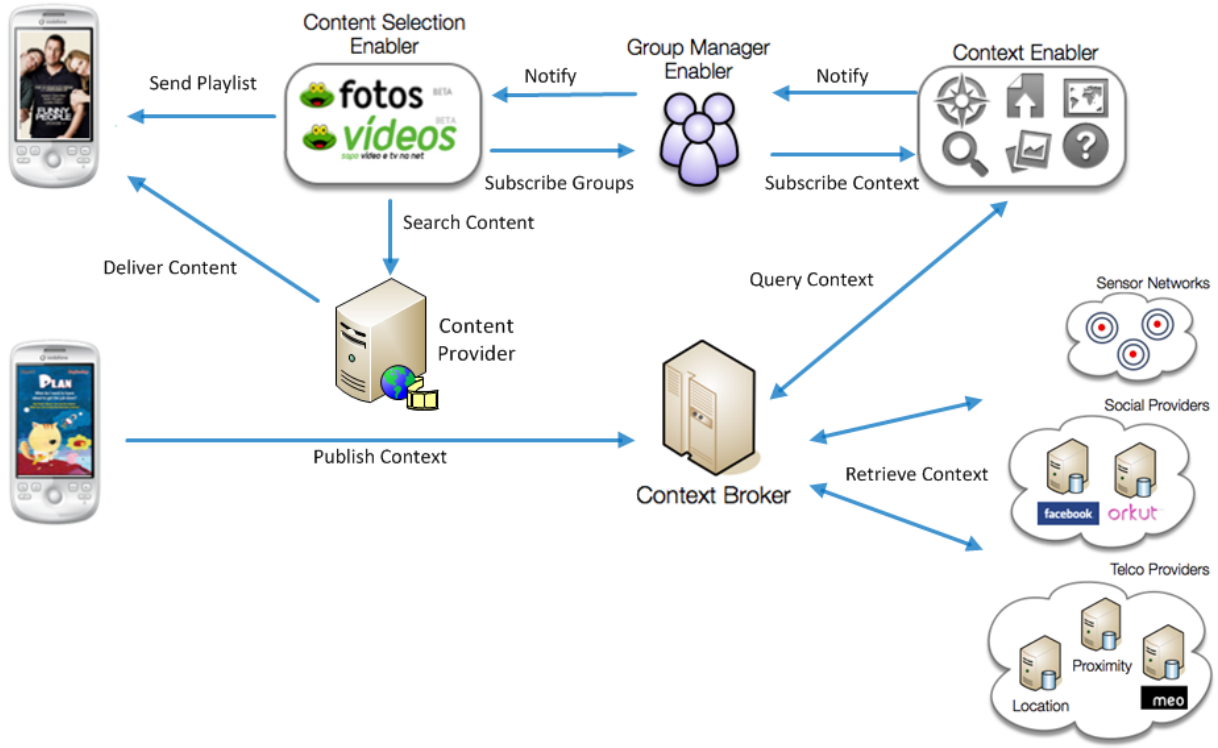


Figura 3.9: Arquitetura da *framework* de contexto

Context Broker

O CxB (Context Broker) é um servidor XMPP da Ignite Realtime [57] que foi selecionado por ser *open-source* e desenvolvido em Java. O Openfire inclui diversos XEPs (XMPP Extension Protocols) que fornecem as funcionalidades fundamentais para a *framework* de contexto como por exemplo o indispensável XEP-0060 *Publish-Subscribe*. O CxB é responsável por agregar toda a informação de contexto dos vários clientes registados controlando o fluxo de mensagens entre os diversos componentes da arquitetura. Os dados publicados por esses componentes são definidos por XML *schemas* que permitem um mais fácil processamento

e integração de novos módulos na *framework*. Toda a comunicação entre os vários módulos é realizada de forma assíncrona através do protocolo XMPP que executa no topo da pilha TCP/IP. O XMPP é largamente usado nos sistemas de *instant-messaging* por ser um protocolo aberto e preparado para as interações de *publish/subscribe*.

Context Enabler

O CtxE (Context Enabler) é o único componente desta arquitetura que foi desenhado e desenvolvido durante este trabalho. É um *web service* SOAP em Java com o propósito de subscrever e devolver informação de contexto sobre um utilizador em específico, interagindo diretamente com o CxB. A existência deste serviço permite o *loose coupling* e a reutilização segundo os conceitos SOA podendo o mesmo ser integrado com outros sistemas simultaneamente.

Group Manager Enabler

O GME (Group Manager Enabler) tem por base um motor de regras responsável por identificar, criar e gerir grupos de clientes a partir de informação de contexto. O GME subscreve a informação existente no CxB de forma a estabelecer grupos baseados nas várias condições pré-configuradas pelo administrador do serviço: localização, preferências sociais, informação capturada por sensores, presença, gênero, etc. [58]. O *web service* CtxE surge como um intermediário neste processo subscrevendo a informação de contexto presente no CxB e devolvendo-a ao GME através de notificações XMPP. Existem 3 estados de operação no GME:

- Formação de grupos
- Atualização de grupos
- Remoção de grupos

A primeira fase do GME é a de formação de grupos. Essa formação ocorre através da análise da informação de contexto dos clientes que se encontrem *online* de forma a colocar os utilizadores nos grupos correspondentes. As regras para a formação de grupos são pré-configuradas pelo administrador num ficheiro XML e armazenadas numa base de dados MySQL. Quando surge uma alteração na informação de contexto de um utilizador,

por exemplo quando um utilizador se move de local ou atualiza alguma informação no seu perfil de rede social, esta alteração de estado poderá despoletar uma atualização na formação grupos. Nesta circunstância o GME gera notificações para todos os componentes que tiverem subscrito a informação de grupos. Por fim, quando um ou mais grupos são removidos através da configuração do administrador, o CxB é notificado desta alteração como os restantes módulos. O GME foi definido no C-CAST como sendo uma aplicação Java (*.jar*) que executava em um servidor Jetty. Porém, procedeu-se a algumas alterações que permitiram a migração para um ambiente de *cloud*, nomeadamente, transformar a aplicação para um ambiente *web* de forma a executar num servidor mais encontrado na oferta dos fornecedores de PaaS, como por exemplo o Apache Tomcat.

A Figura 3.10 descreve os vários estados operacionais do GME como a interação XMPP com os módulos de suporte.

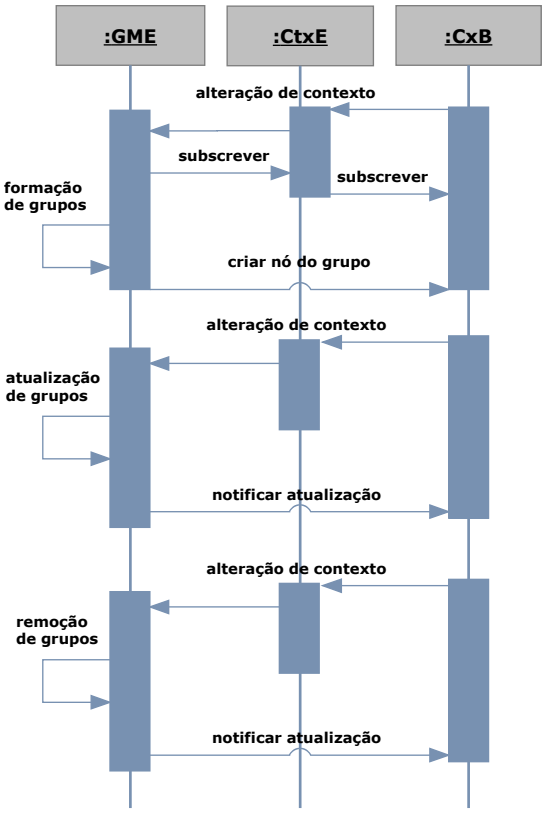


Figura 3.10: Diagrama de sequência do *Group Manager Enabler*

Content Selection Enabler

O CSE (Content Selection Enabler) tem por base um motor de regras responsável pela seleção dos conteúdos (por exemplo, vídeos, fotos, etc.) mais adequados para um cliente ou para grupos de clientes. Este módulo atua como um elemento essencial na decisão e entrega de conteúdos baseados em informação de contexto. O CSE subscreve a informação de grupos gerada pelo GME, selecionando do conteúdo alojado em um servidor (*Content Provider*) qual ou quais são os mais adequados para serem transmitidos aos utilizadores. Esse processo é efetuado através da análise dos meta-dados relacionados com o conteúdo que indicam as características do mesmo. Os modos de operação do CSE despoletam quando surgem notificações do GME e do *Content Provider* de forma a selecionar, atualizar ou remover conteúdos da lista de reprodução que é enviada aos terminais móveis dos clientes. Após o processo de seleção, que pode ser por base no dialeto, localização, interesses do utilizador, ou por exemplo a hora atual do dia, é realizada a entrega de uma lista de reprodução com um ou mais conteúdos selecionados. Por sua vez, o cliente tem a possibilidade de escolher da lista qual o conteúdo que deseja visualizar bem como a ordem em que os mesmos serão executados. Portanto, o CSE poderá ser mais ou menos complexo dependendo do algoritmo do motor de regras que é utilizado por base para as várias operações.

A Figura 3.11 descreve os vários estados operacionais do CSE e a interação existente com os módulos de suporte e com o cliente.

3.6 Sumário

Neste capítulo foi detalhada a especificação e desenvolvimento da camada agregadora de fornecedores de PaaS, denominada por *PaaSManager*. Inicialmente foram analisadas as APIs dos vários fornecedores de PaaS suportados com o intuito de definir as operações que seriam implementadas pela solução. De seguida, foi ilustrada a arquitetura e o funcionamento dos vários módulos que constituem o *PaaSManager*. Por último, foi introduzida a *framework* da PTIN de serviços baseados em informação de contexto que será parcialmente portada para um ambiente *cloud* através da solução desenvolvida.

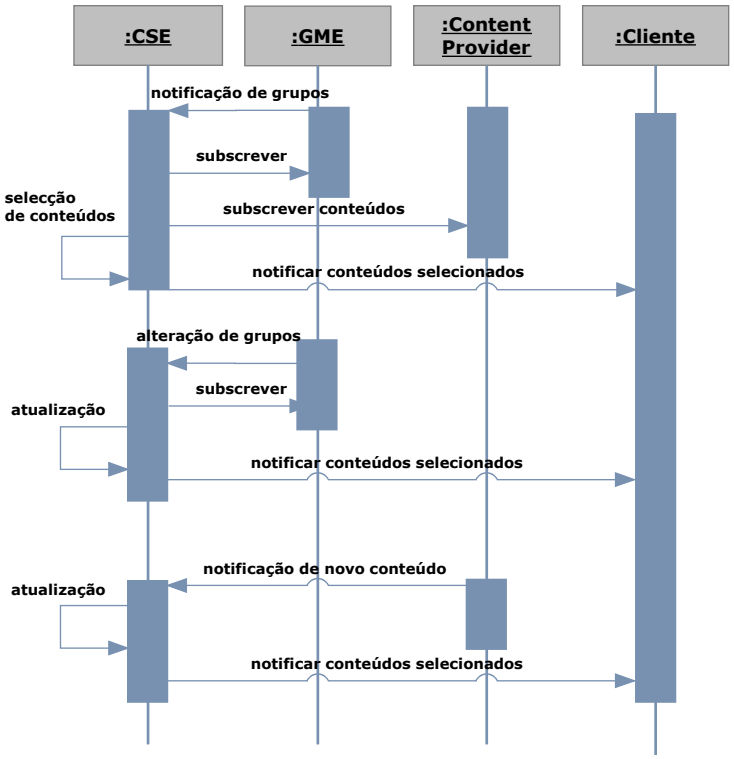


Figura 3.11: Diagrama de sequência do *Content Selection Enabler*

Capítulo 4

Ensaios e Avaliação

Neste capítulo são apresentados alguns dos casos de estudos delineados de modo a validar a solução *PaaSManager* em cenários reais. Estes casos de estudo abrangem as diversas funcionalidades suportadas sendo possível verificar a especificação efetuada dos vários módulos que constituem o sistema. São ainda apresentados os resultados provenientes de uma análise de desempenho, de forma a avaliar o comportamento da arquitetura com diversos utilizadores em simultâneo.

4.1 Cenários de Estudo

Esta secção tem como objetivo descrever os casos de estudos definidos para a validação do *PaaSManager*. Cada cenário retrata a utilidade da solução tanto no ponto de vista de utilizadores comuns como empresariais. Através da camada de abstração desenvolvida, as várias funcionalidades suportadas fornecem a oportunidade para os utilizadores criarem, gerirem e monitorizarem aplicações e base de dados hospedadas em diversos fornecedores de PaaS. A partir das operações expostas, novos módulos poderão ser construídos para interagir com a API disponibilizada e utilizar a informação devolvida para diferentes fins. Os cenários concebidos envolvem a integração de um sistema que utilize o *PaaSManager* de modo a recomendar ao utilizador qual o PaaS mais adequado, por exemplo, perante o perfil de dependências técnicas de uma aplicação. No caso de estudo apresentado na secção 4.2, serão portadas algumas das aplicações que constituem a *framework* de serviços baseados em informação de contexto da PTIN previamente descrita na secção 3.5. Após

proceder a criação da aplicação no ambiente recomendado, será possível gerir e monitorizar os recursos através de um único interface que agrega todas as operações suportadas e informação relacionada.

Adicionalmente, na secção 4.3 será apresentado um outro caso de estudo onde são apresentados e visualizados diversos resultados obtidos provenientes dos testes de desempenho efetuados ao *PaaSManager*. Para testar a solução foram selecionados alguns dos métodos fundamentais de forma a serem avaliados diferentes aspetos e depurado onde seria possível proceder a otimizações.

4.2 Recomendador de *Platform-as-a-Service*

Cada aplicação possui dependências específicas que podem ser tão distintas como a mesma ser desenvolvida em Java ou Python até necessitar de uma base de dados SQL ou NoSQL. Para um utilizador que pretenda portar as suas aplicações para um ambiente de *cloud*, ele precisará de examinar as diversas ofertas existentes no mercado, conhecer as tecnologias suportadas, estudar e testar os interfaces cliente, as APIs fornecidas, etc. Logo, torna-se essencial facilitar esse processo tornando-o mais cómodo e atraente para os utilizadores de *Platform-as-a-Service*. Dessa forma surgiu a plataforma CSB (Cloud Service Broker) que foi desenvolvida no âmbito de um projeto do Instituto de Telecomunicações de Aveiro suportado pela PTIN. O CSB visa efetuar a recomendação baseada em regras definidas e analisadas pelo seu motor de regras. As suas características principais são dar assistência no processo de recomendação e fornecer todas as operações suportadas pelo *PaaSManager* através de um único interface. Portanto, o utilizador tem assim acesso a todo o ciclo de vida de uma aplicação hospedada no PaaS recomendado pelo CSB. Além dos requisitos técnicos de uma aplicação, o algoritmo de recomendação poderá ser estendido suportando diversas exigências. São alguns exemplos a geração de notificações quando certos patamares nos valores de monitorização forem ultrapassados, de forma a respeitar os contratos definidos pelo próprio utilizador; ou então a recomendação por base nos modelos de negócio que se verifiquem mais adequados para o próprio detentor de uma aplicação.

A arquitetura deste cenário inclui além do *PaaSManager*, a plataforma CSB e as interfaces cliente: *web*, CLI e Git. Através da interface *web* e CLI o utilizador terá acesso às operações dos serviços de gestão e de informação suportados pelo *PaaSManager*. No momento da criação de uma aplicação, será preenchido o formulário que constitui o per-

fil tecnológico da aplicação, nomeadamente, os *runtimes*, *frameworks* e *services* (base de dados). Consequentemente o CSB invoca o método *getPaaSOffering* que retorna as tecnologias suportadas por cada plataforma do ecossistema, comparando com a informação inserida no perfil. Ao utilizador é devolvido uma lista ordenada com os fornecedores recomendados. Em relação ao Git, e analogamente ao Heroku e OpenShift, esta ferramenta foi definida como a interface de *deployment* do código fonte de uma aplicação para a plataforma pretendida. Portanto teve-se em atenção que do ponto de vista do utilizador, interagir com esta arquitetura não se verifique mais complexo do que interagir diretamente com um fornecedor em específico. A Figura 4.1 apresenta a arquitetura especificada para este cenário de utilização do *PaaSManager*.

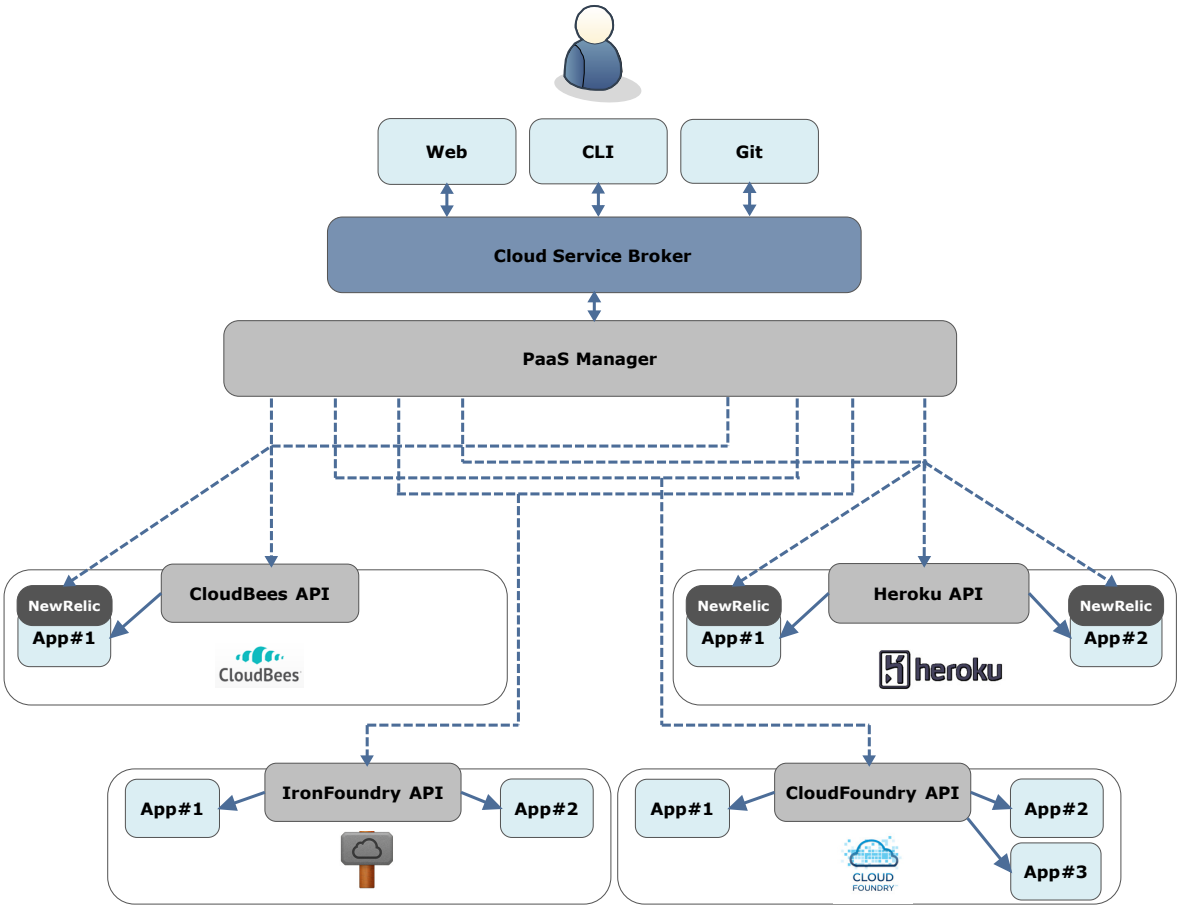


Figura 4.1: Arquitetura do caso de estudo recomendador de *Platform-as-a-Service*

Através das interfaces disponibilizadas serão portadas algumas das aplicações que constituem a *framework* de serviços baseados em informação de contexto da PTIN. A plata-

forma CSB irá recomendar qual ou quais os PaaS mais adequados para suportar estes serviços conforme o perfil tecnológico inserido. Após este processo de recomendação, as aplicações serão criadas e consequentemente será realizado o *deployment* do código fonte associado. Finalmente, em caso de sucesso, a interface permitirá a monitorização em tempo-real devolvendo informação sobre o comportamento dos serviços de contexto num ambiente de *cloud*. Para este cenário foram selecionados, da *framework* de contexto, o *web service* CtxE e a aplicação *web* GME previamente descritos na secção 3.5. No caso do CtxE a decisão foi estabelecida devido a ser um serviço simples que poderá ser reutilizado por outros sistemas em simultâneo. Por sua vez o GME foi selecionado por possuir um interface *web* e estar sujeito a diversas notificações vindas de outros módulos da arquitetura.

4.2.1 Processo de Criação de Aplicações

A primeira operação invocada pretende criar uma aplicação na plataforma recomendada. Os passos para completar essa tarefa são simples e pragmáticos. Inicialmente o utilizador não possui nenhuma aplicação registada, portanto é necessário definir uma identificação para a aplicação e preencher o perfil tecnológico para o CSB analisar qual o PaaS mais adequado. Para o CtxE foi selecionado o *runtime Java 1.6*, *framework Java EE6 Web Profile* e nenhuma base de dados. Consequentemente o CSB devolve uma lista ordenada indicando o ou os PaaS mais adequados para suportar esta aplicação. Neste caso, o CloudFoundry foi anunciado como sendo o ideal e portanto foi o PaaS selecionado para receber o CtxE, como é apresentado na Figura 4.2. Ao selecionar o CloudFoundry é automaticamente criado um repositório Git no *PaaSManager* no qual será colocado o código fonte para posterior *deployment*. Igualmente é armazenada na base de dados central a informação de estado da aplicação (identificador da plataforma, identificador da *framework* utilizada, etc.). Do lado do fornecedor de plataforma, são registados os dados aplicacionais e é preparado o ambiente de execução. Por fim, a Figura 4.3 ilustra a informação que é devolvida após o processo de criação do CtxE na plataforma selecionada.

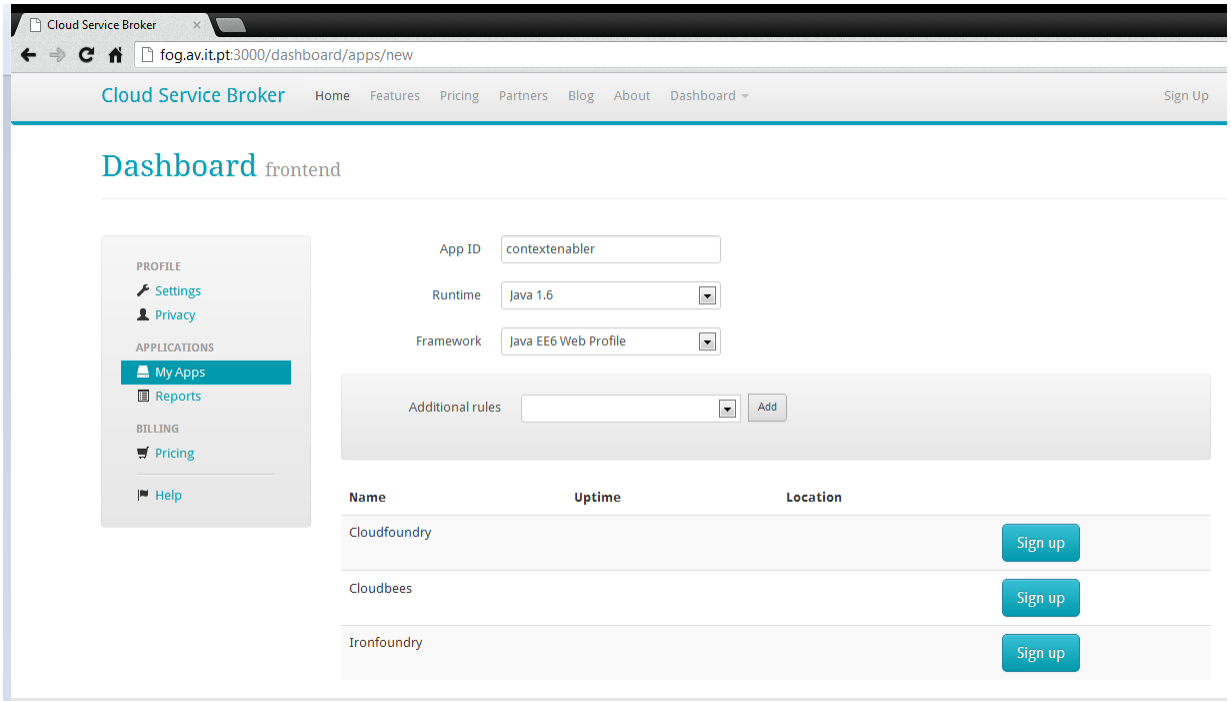


Figura 4.2: Definir identificação e perfil tecnológico do *Context Enabler*

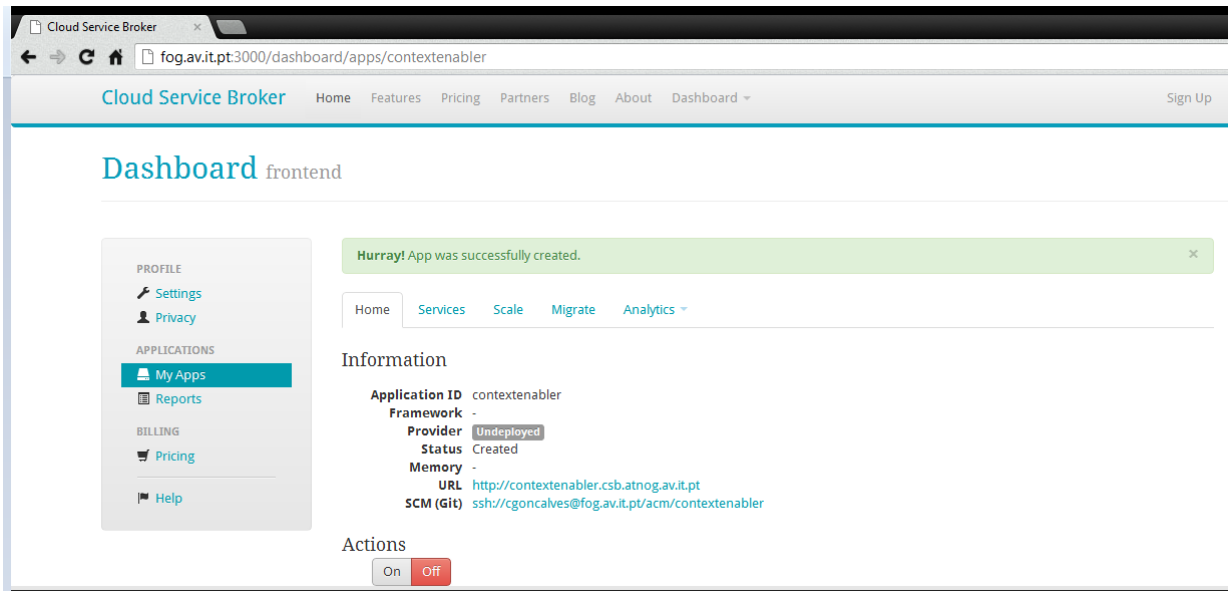


Figura 4.3: Informação devolvida sobre o *Context Enabler*

Para o GME os mesmos passos terão de ser repetidos sendo inicialmente registada a aplicação. Ao preencher o perfil tecnológico, é selecionada desta vez uma base de dados MySQL e respetiva versão. Conforme a informação inserida no perfil GME, o CSB anun-

cia que o CloudBees surge como sendo o PaaS ideal para suportar a aplicação, como é apresentado na Figura 4.4. Ao selecionar o CloudBees, é apresentada a informação sobre o estado atual do GME como se pode observar na Figura 4.5. Por fim, na Figura 4.6 é apresentada a lista de aplicações que foram criadas até ao momento.

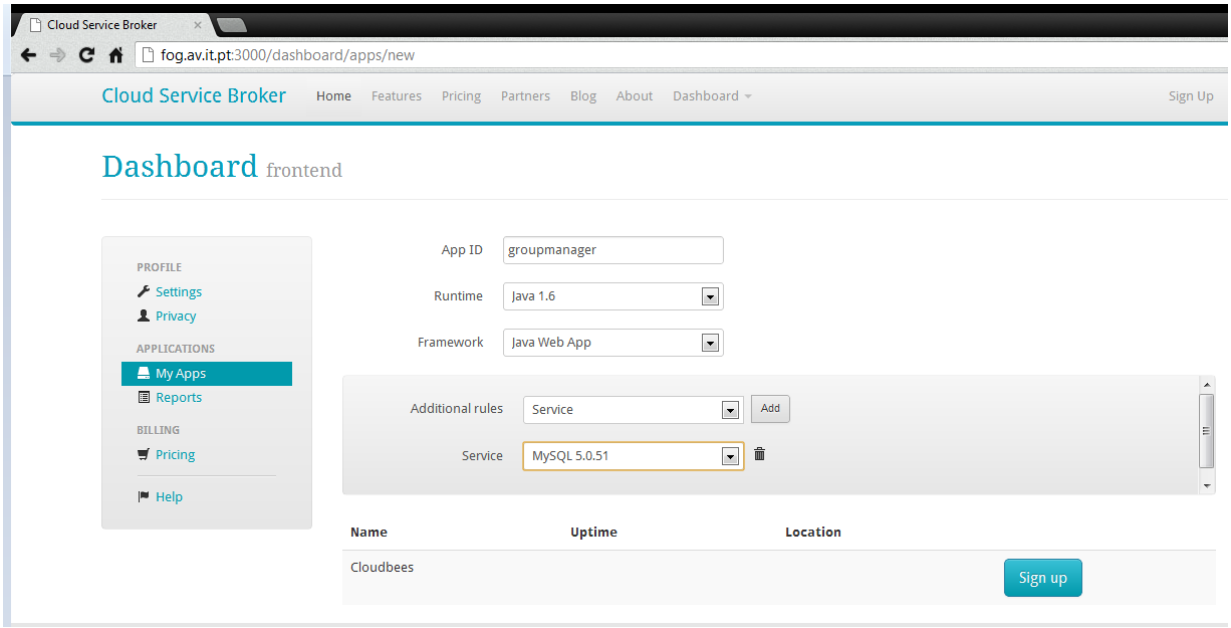


Figura 4.4: Definir identificação e perfil tecnológico do *Group Manager Enabler*

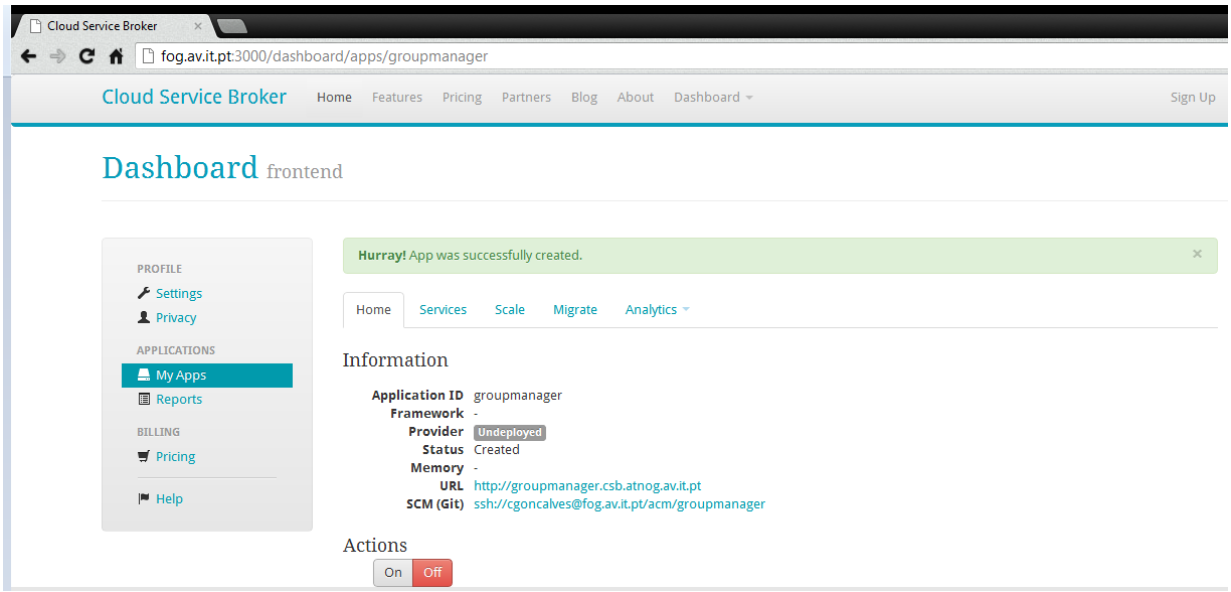


Figura 4.5: Informação devolvida sobre o *Group Manager Enabler*

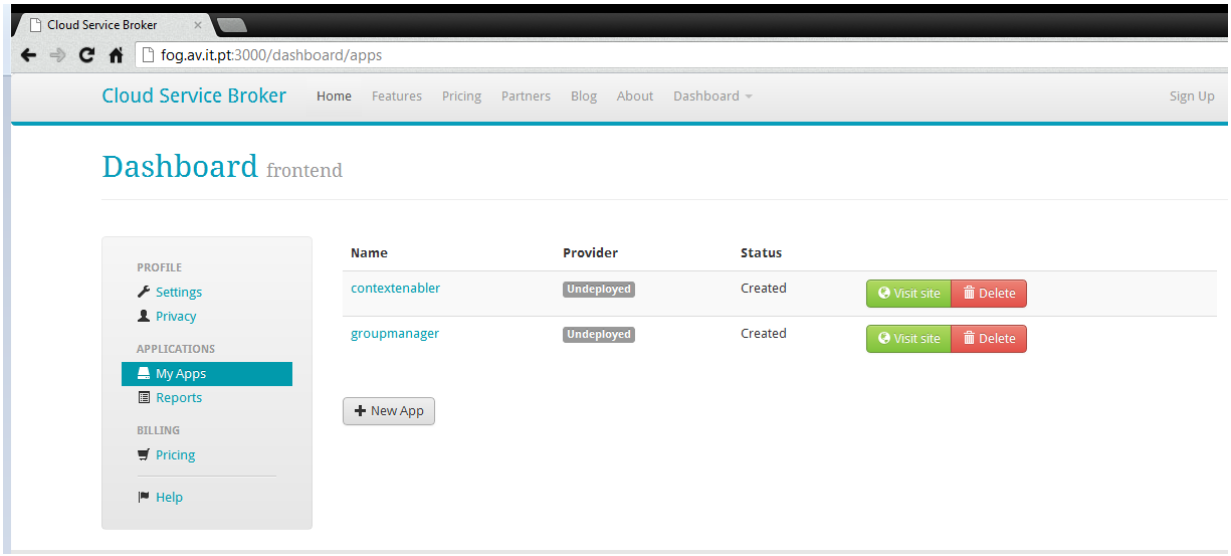


Figura 4.6: Lista de aplicações criadas

4.2.2 Processo de *Deployment* de Código Fonte

Após o processo anteriormente descrito, o código fonte de cada serviço poderá ser enviado, através do comando *git-push*, para os respetivos repositórios Git criados anteriormente. No caso do *Context Enabler* para o *ssh://cgoncalves@fog.av.it.pt/acm/contextenabler* e no caso do *Group Manager Enabler* para o *ssh://cgoncalves@fog.av.it.pt/acm/groupmanager*. Desta forma o código fonte do CtxE é *deployed* na respetiva plataforma encontrando-se em modo *running* como é apresentado na Figura 4.7. Sendo um *web service* SOAP é possível observar o contrato WSDL em *contextenabler.cloudfoundry.com/CtxEnabler?wsdl* na Figura 4.8.

Para o GME o processo é ligeiramente diferente devido a ter uma base de dados associada. Neste momento é necessário proceder a algumas alterações no código fonte do GME. As alterações efetuadas incluem implementar o WSDL do CtxE, que se encontra hospedado no CloudFoundry, e também colocar a identificação e credenciais de acesso da base de dados recém-criada. Após estas atualizações no código é possível proceder ao *deployment* através de comandos Git. Desta forma, o GME encontra-se em modo *running* como é apresentado na Figura 4.9. O interface *web* do serviço, ilustrado na Figura 4.10, já possui alguns grupos registados pelo administrador.

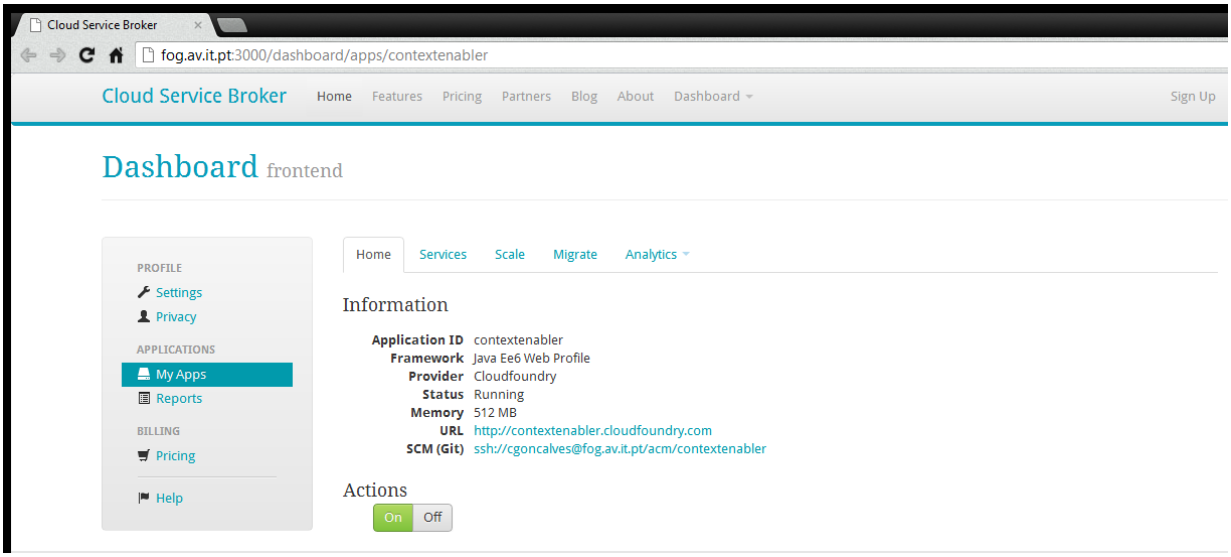


Figura 4.7: Processo de *deployment* do *Context Enabler* efetuado com sucesso

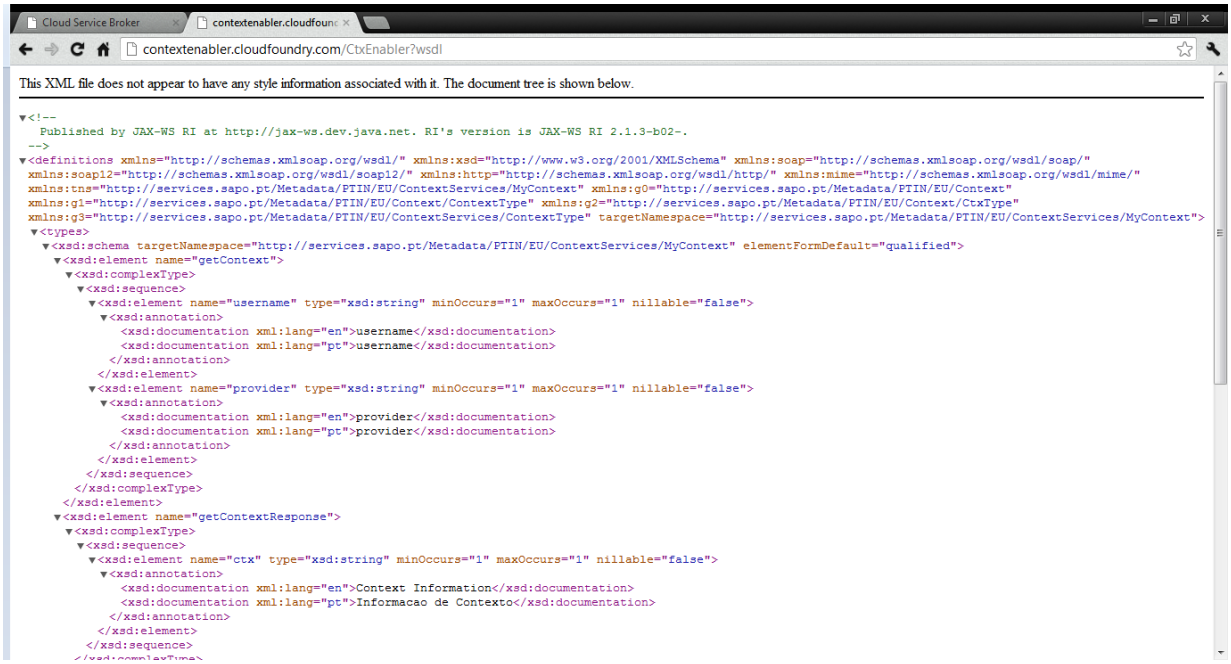


Figura 4.8: *Context Enabler* no CloudFoundry

Após as duas aplicações da *framework* de contexto estarem hospedadas num ambiente *cloud*, procedeu-se a um teste de funcionamento através de uma simples aplicação móvel de *instant-messaging* que partilha a posição GPS de um utilizador móvel. Neste caso foi partilhada a posição referente a Aveiro através de mensagens XMPP enviadas ao *Context Broker*. Essa informação é depois consumida e processada pelas 2 aplicações colocando o

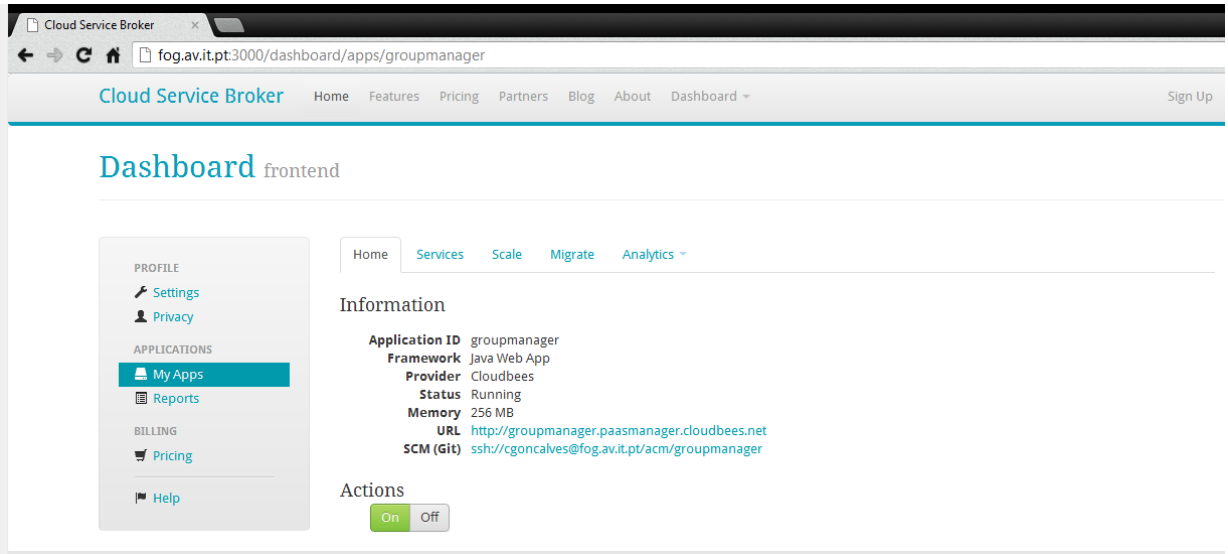


Figura 4.9: Processo de *deployment* do *Group Manager Enabler* efetuado com sucesso

Figura 4.10: *Group Manager Enabler* no CloudBees

cliente no grupo de utilizadores presentes em Aveiro como é ilustrado na Figura 4.11. Se mais utilizadores se encontrarem *on-line* e a suas informações de contexto coincidirem com os grupos criados no GME, os utilizadores serão agrupados e poderão, através do CSE,

receber uma lista de reprodução com os conteúdos mais adequados.

Figura 4.11: Utilizador *on-line* e pertencente ao grupo de utilizadores em Aveiro

4.2.3 Processo de Monitoria

Além de ser monitorizado o estado de uma aplicação, existem outras formas de obter informação sobre o seu comportamento num ambiente PaaS. Por um lado através dos *logs* do servidor aplicacional onde executa o CtxE, como é ilustrado na Figura 4.12. Por outro lado, através das estatísticas recolhidas diretamente através da API do CloudFoundry. Neste último, é possível aceder a um histórico de recolhas efetuadas nas últimas horas ou dias pelo motor de monitorização, sendo ilustrado na Figura 4.13.

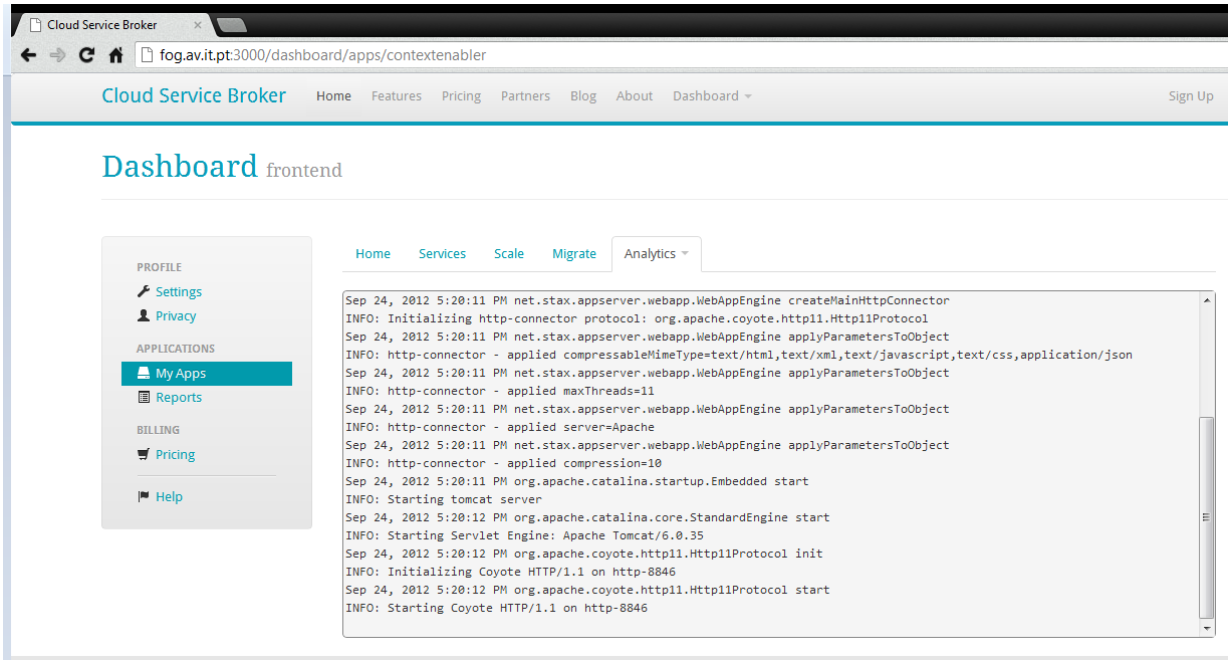


Figura 4.12: *Logs* do *Context Enabler*

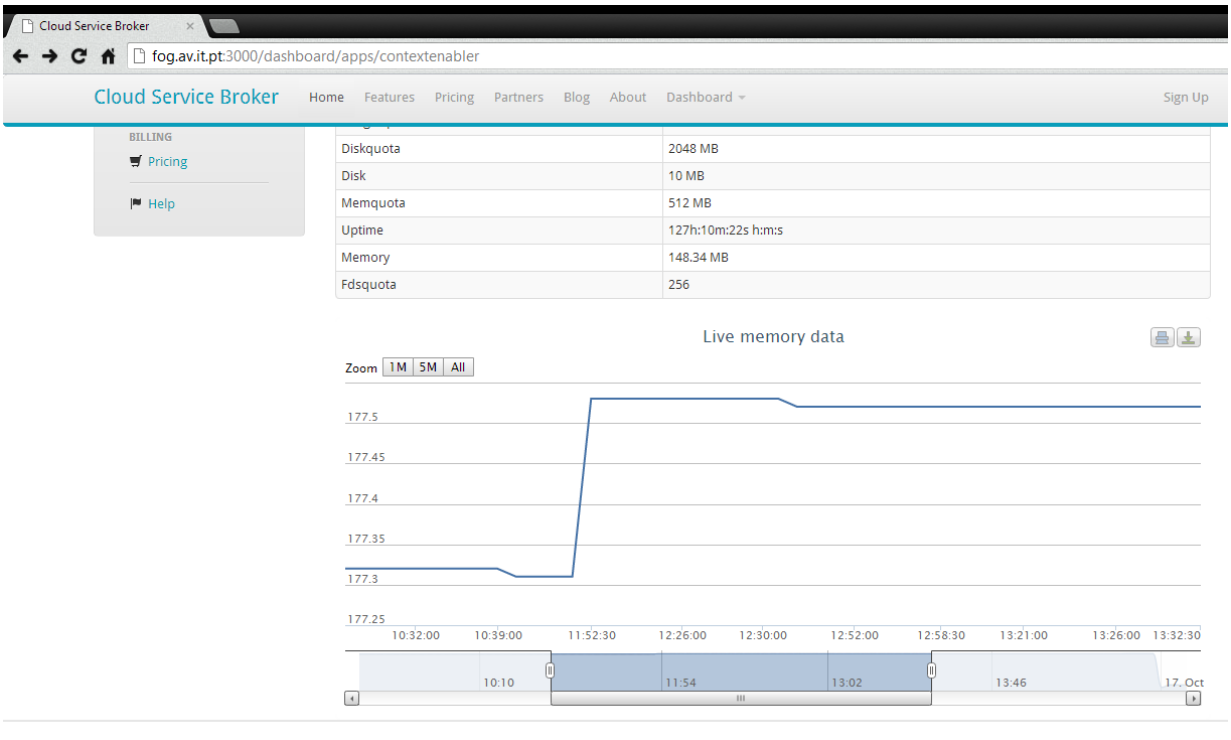


Figura 4.13: Monitorização em tempo-real do *Context Enabler*

4.3 Análise de Desempenho do *PaaSManager*

Esta secção tem como objetivo mostrar alguns resultados obtidos nos testes de desempenho efetuados ao *PaaSManager*. Consequentemente será analisado o comportamento do sistema durante algumas das várias operações suportadas em determinados cenários de carga.

4.3.1 Ambiente de Testes e Metodologia

Para testar a solução foram selecionados alguns dos métodos fundamentais de forma a serem avaliados diferentes aspetos e depurado onde seria possível proceder a otimizações. Os métodos selecionados foram os seguintes:

- *createApp*
- *deployApp*
- *migrateApp*
- *getAppStatus*

A ferramenta Apache JMeter [59] foi utilizada para efetuar os testes de carga ao *PaaSManager*. Com o JMeter é possível simular o acesso de diversos clientes em simultâneo aos recursos disponíveis medindo o desempenho de um serviço ou de um sistema. A principal métrica devolvida, e consequentemente analisada, foi o tempo de resposta. Como é óbvio esta métrica é bastante variável e depende de diversos fatores tais como da rede e respetiva atividade, a geolocalização física dos fornecedores de PaaS, a proximidade ao servidor aplicacional onde executa o *PaaSManager*, como igualmente dos recursos computacionais da própria máquina que o suporta. A máquina utilizada para albergar o *PaaSManager* possuía 3 GB de memória RAM, um processador Intel®DualCore™de 2 GHz e, como sistema operativo, o Ubuntu 12.04 LTS.

Os valores obtidos serviram de referência para compreender o comportamento que o sistema detém em operações essenciais. Por um lado, de forma a se observar o *overhead* que é acrescido pelo *PaaSManager* em cada pedido efetuado, e por outro, de forma a ser analisada a escalabilidade da solução com diferentes números de utilizadores. Para cada um dos métodos selecionados foram realizados ensaios com 10 e 30 utilizadores em

simultâneo, obtendo-se a média e o desvio padrão associado a cada operação efetuada nas diferentes plataformas que constituem o ecossistema. Esse número de utilizadores foi escolhido devido ao *PaasManager* utilizar apenas uma conta para cada PaaS. Assim sendo, um grande número de pedidos poderia se refletir num bloqueamento da conta ou num efeito de *throttling* por parte dos fornecedores. É necessário realçar que o objetivo desta análise não é comparar diretamente a eficiência de cada fornecedor mas sim o desempenho da arquitetura desenvolvida.

4.3.2 Métodos Avaliados

Seguidamente são apresentados os resultados obtidos nos diversos testes efetuados ao *PaaSManager*. Os dados apresentados nesta secção foram divididos em 2 séries: *PaaSManager* e *PaaSManager+PaaS API*, que respetivamente isolam o tempo consumido apenas no processamento interno dos pedidos no *PaaSManager*, do tempo consumido em todo o processo de uma operação incluindo a solicitação à API da plataforma em questão.

createApp

O processo do método *createApp*, detalhado na Figura 3.3 anteriormente apresentada, envolve a inicialização de um repositório Git para a aplicação, a preparação do ambiente de execução num PaaS específico e, finalmente, o armazenamento de informação de estado na base de dados central.

Com um total de 10 utilizadores em simultâneo a criarem respetivamente uma aplicação, foram obtidos os resultados apresentados na Figura 4.14. Em relação à série *PaaSManager*, o CloudBees e o Heroku possuem os resultados mais baixos sendo inferiores a 60 ms. Na ordem dos 500 ms encontra-se a média do CloudFoundry e IronFoundry. Esta discrepância surge devido a lógica interna da operação *createApp* ser diferente em cada um dos *adapters* de PaaS. Como é óbvio, na série *PaaSManager+PaaS API*, verifica-se que o tempo de resposta depende fundamentalmente da API nativa de cada fornecedor. Neste caso, o Heroku relevou o maior valor obtido no ecossistema, cerca de 2023 ms para completar o processo *createApp*. Por outro lado, o CloudFoundry e o IronFoundry apresentaram um valor de médio de 1200 ms enquanto o CloudBees, sendo o único PaaS que não suporta este método na sua API, não possuiu nenhum valor identificado.

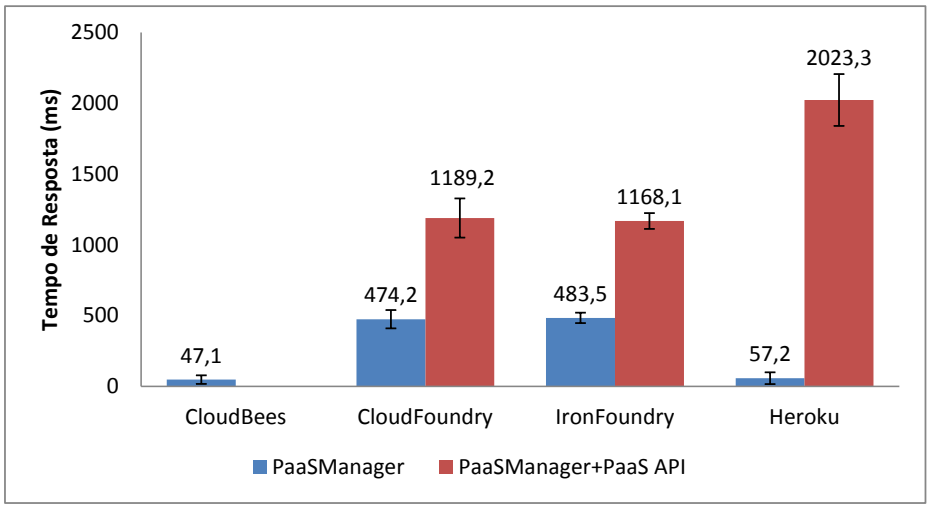


Figura 4.14: Tempo de resposta – *createApp* 10 utilizadores

Para a mesma operação foi testado também o acesso de 30 utilizadores em simultâneo. Os resultados, que são observados na Figura 4.15, não revelam grande discrepância relativamente aos tempos recolhidos no teste com 10 utilizadores, ilustrando um bom comportamento em termos de escalabilidade por parte da solução neste caso.

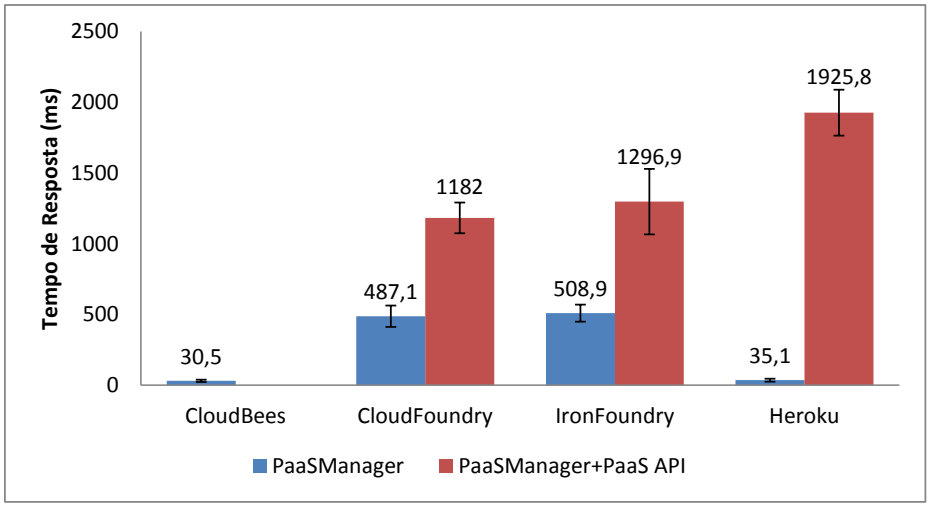


Figura 4.15: Tempo de resposta – *createApp* 30 utilizadores

deployApp

O método *deployApp*, especificado na Figura 3.4 anteriormente apresentada, abrange diversos processos incluindo o *upload* do código fonte da aplicação para o respetivo repositório Git, o *deployment* para o PaaS selecionado invocando o método da API nativa, e por fim a inicialização do motor de monitorização. Para esta avaliação, o *web service Context Enabler*, com 8 MB de dimensão, foi a aplicação escolhida de forma a serem obtidos dados que pudessem ser comparados independentemente da plataforma.

Com 10 utilizadores a realizarem o *deployment* em simultâneo, foram obtidos os resultados apresentados na Figura 4.16. A série *PaaSManager* revela valores de tempo de resposta entre os 600 e os 1200 ms que inclui o *upload* do código fonte e respetivo *commit* para o repositório Git. Porém na série *PaaSManager+PaaS API*, surge uma grande proeminência do CloudBees em relação aos restantes fornecedores. O pedido atingiu uma média de aproximada de 94655 ms, nos quais mais de 90% do tempo de resposta é utilizado apenas pelo processamento da API nativa do CloudBees. Por outro lado, o Heroku possuiu um tempo de *deployment* mais reduzido, cerca de 63000 ms, enquanto o CloudFoundry e IronFoundry revelaram tempos mais baixos, aproximadamente 6500 ms.

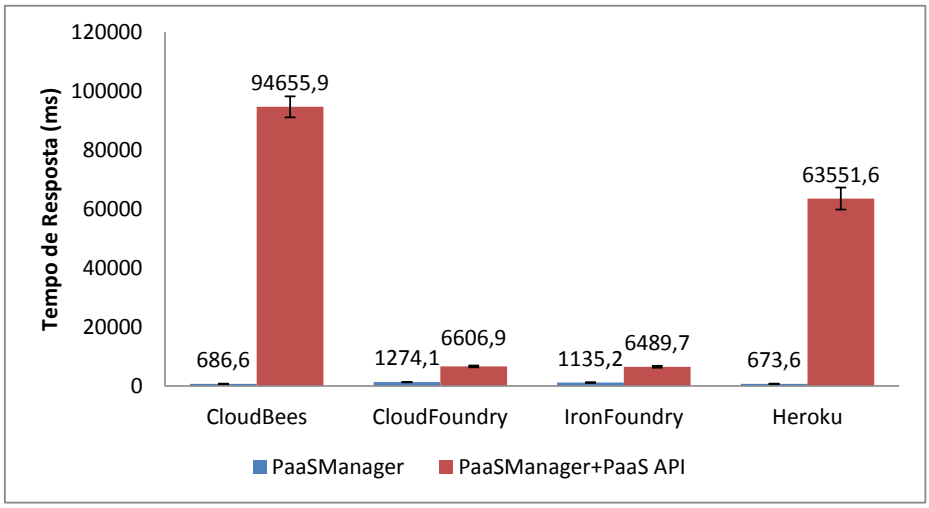


Figura 4.16: Tempo de resposta – *deployApp* 10 utilizadores

Para a mesma operação foi testado o acesso de 30 utilizadores em simultâneo. Os resultados, apresentados na Figura 4.17, não revelaram grande discrepância relativamente aos tempos recolhidos no teste com 10 utilizadores. Mais uma vez é constatada uma boa escalabilidade do sistema sendo que em ambos os cenários estudados apenas uma parte

reduzida do tempo de resposta se deve ao *PaaSManager*.

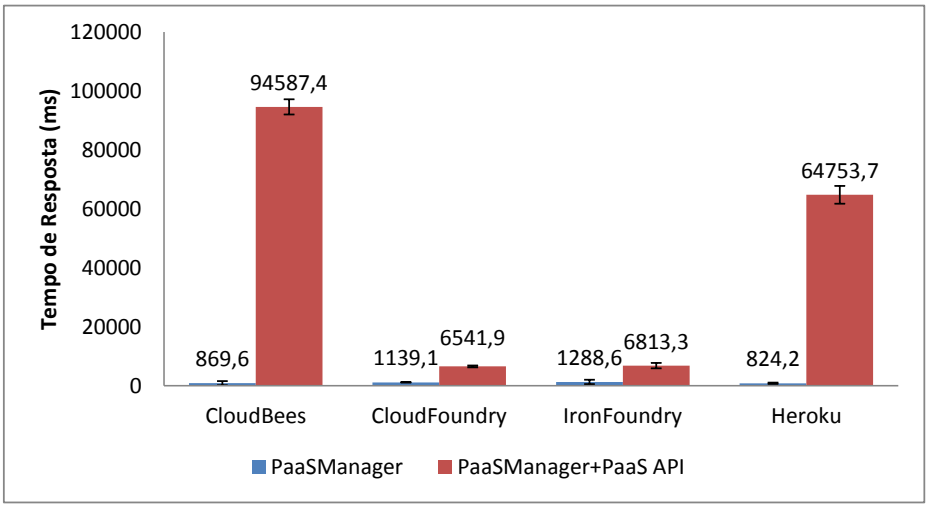


Figura 4.17: Tempo de resposta – *deployApp* 30 utilizadores

migrateApp

A operação *migrateApp*, cujo processo foi detalhado na Figura 3.5, inclui diversos procedimentos que permitem a portabilidade de aplicações entre diferentes fornecedores de PaaS presentes no ecossistema. Inicialmente é confirmado se a plataforma para onde a aplicação será migrada suporta as tecnologias fundamentais para a sua correta execução. Em caso de sucesso, são ativados os procedimentos necessários para a criação e o *deployment* da aplicação na nova plataforma. Por fim a aplicação existente no PaaS anterior é eliminada. Para esta avaliação foram testados todos os casos possíveis de migração entre os fornecedores de plataformas. O *web service Context Enabler*, por ser suportado em todos os PaaS, foi a aplicação selecionada.

Alternativamente aos restantes testes e por limitações inerentes às APIs dos fornecedores, não foram realizados testes de acesso simultâneo mas sim efetuados 30 testes isolados que posteriormente serviram para calcular a média do tempo de resposta. Para a série *PaaSManager*, que apenas inclui os processos de 1 a 5 apresentados na Figura 3.5, observam-se tempos bastantes aproximados entre os 1200 e os 1300 ms em todas as transições efetuadas. A Tabela 4.1 resume os resultados obtidos.

Tabela 4.1: Tempo de resposta - *migrateApp* - série *PaaSManager*(ms)

PaaSManager	**CloudBees**	**CloudFoundry**	**IronFoundry**	**Heroku**
CloudBees	-	1271,5	1191,4	1221,6
CloudFoundry	1260,7	-	1195,5	1262,2
IronFoundry	1287,7	1195,3	-	1319,0
Heroku	1213,2	1203,7	1230,4	-

Por sua vez, a série *PaaSManager+PaaS API* envolve todo o processo de migração incluindo a invocação dos métodos *createApp*, *deployApp*, *getAppStatus* e *deleteApp* através das APIs dos fornecedores. É evidente que para cada plataforma obteve-se em média o agregado do tempo de resposta das várias operações. Os valores obtidos são apresentados na Tabela 4.2.

Tabela 4.2: Tempo de resposta - *migrateApp* - série *PaaSManager+PaaS API*(ms)

PaaSManager+ PaaS API	**CloudBees**	**CloudFoundry**	**IronFoundry**	**Heroku**
CloudBees	-	14454,2	13628,4	75451,7
CloudFoundry	103536,0	-	11090,5	75130,8
IronFoundry	102339,1	10257,6	-	74708,6
Heroku	101418,4	12241,7	11863,8	-

getAppStatus

O método *getAppStatus*, detalhado anteriormente na Figura 3.6, adquire informação de estado sobre uma aplicação verificando-se fundamental para obter conhecimento sobre o comportamento da mesma. Este processo inclui o pedido a base de dados central de forma a obter a identificação do PaaS onde a aplicação se encontra hospedada para assim ser invocado a operação do respetivo *adapter*.

Com um total de 10 utilizadores em simultâneo, foram obtidos os resultados apresentados na Figura 4.18. A série *PaaSManager* mostra valores de tempo de resposta entre os 60 e os 600 ms. Esta série apenas inclui o processo de aquisição da identificação da plataforma até ser invocado o respetivo *adapter* de PaaS. Na série *PaaSManager+PaaS API*, o CloudFoundry, IronFoundry e Heroku registaram os valores mais elevados entre os

1200 e 1300 ms nos pedidos às APIs nativas. No caso dos *adapters* do CloudFoundry e IronFoundry existe uma verificação nos *logs* de cada instância onde a aplicação executa com o intuito de detetar erros de funcionamento, explicando assim a maior contribuição do *PaaSManager* nos tempos de resposta obtidos.

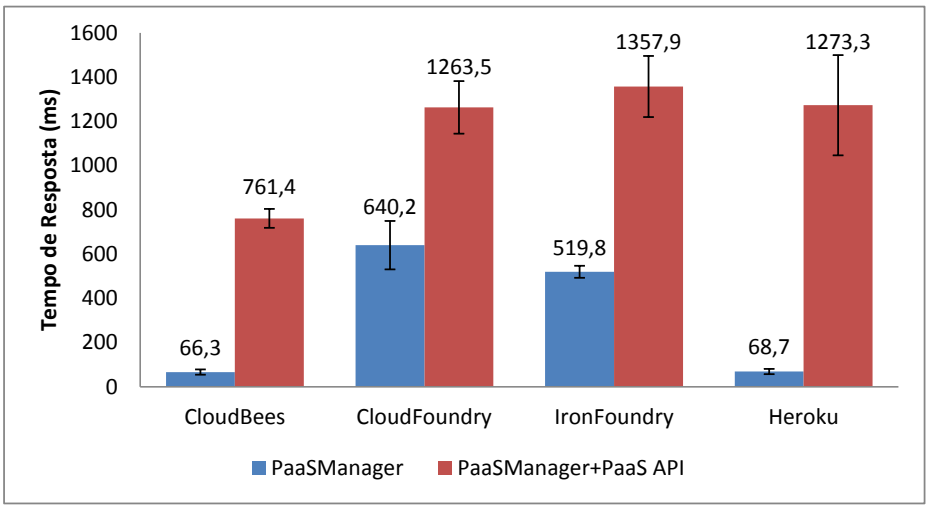


Figura 4.18: Tempo de resposta – *getAppStatus* 10 utilizadores

Igualmente foi testado o acesso de 30 utilizadores em simultâneo. Os resultados apresentados na Figura 4.19, revelam unicamente um ligeiro aumento da média do tempo de resposta obtido. Em termos gerais, e tendo em conta a magnitude dos valores obtidos em ambos os cenários, o *PaaSManager* parece responder mais uma vez bem a cenários de aumento de carga.

4.3.3 Análise de Resultados

Os resultados obtidos revelaram que, em geral, o *PaaSManager* não introduz um *overhead* significativo em algumas nas operações fundamentais, caso do *createApp* e *deployApp*. No caso da operação *getAppStatus* para o CloudFoundry e IronFoundry, uma razoável percentagem do tempo obtido é utilizada no processamento do pedido por parte do *PaaSManager*. Porém verificou-se que interagir com o *PaaSManager* acaba por não colocar mais complexidade nem originar um muito maior tempo de resposta em comparação a interagir diretamente com cada plataforma. Para combater algumas das situações verificadas, será possível proceder a algumas otimizações que poderão ser conduzidas, principalmente nos *adapters* do CloudFoundry e IronFoundry onde foram registados os maiores tempos de

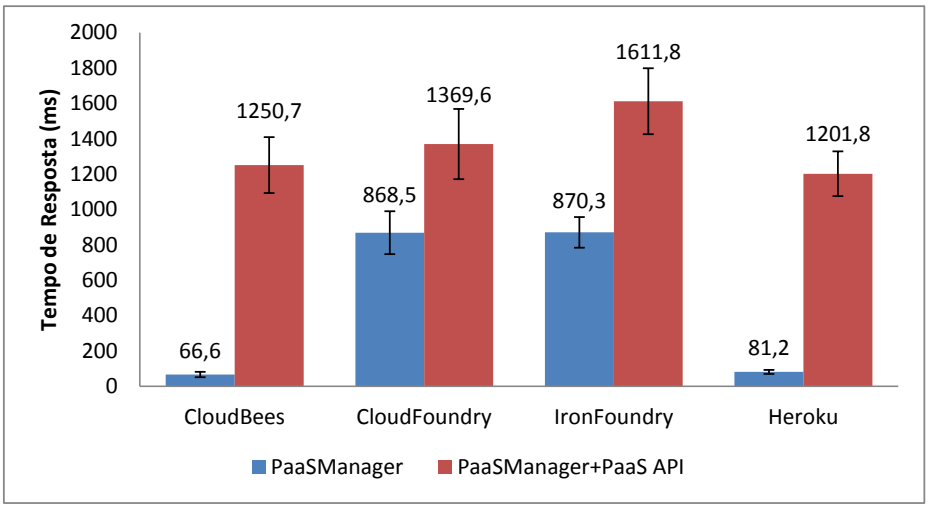


Figura 4.19: Tempo de resposta – *getAppStatus* 30 utilizadores

resposta no processamento interno dos pedidos pelo *PaaSManager*.

A migração, sendo definida por uma agregação de várias operações, não permite ser comparada diretamente com algum dos métodos expostos pelas APIs nativas dos fornecedores suportados. Assim sendo, os resultados obtidos apenas ilustram como a solução se comporta no processo de portabilidade entre diferentes PaaS.

4.4 Sumário

Neste capítulo foram demonstrados alguns dos casos de estudos propostos para a validação da solução. Através de um recomendador de *Platform-as-a-Service* foram enaltecidas as potencialidades da arquitetura desenvolvida. Um interface *web* permitiu fornecer aos utilizadores uma gestão unificada das aplicações e bases de dados criadas em diversos ambientes de *PaaS*. Por fim, foram realizados testes de carga a certas operações suportadas pelo *PaaSManager* com o intuito de se observar o comportamento dos módulos especificados e desenvolvidos. Os resultados obtidos revelaram que a arquitetura não coloca um *overhead* significativo nem uma maior complexidade na interação com cada PaaS final na maioria das operações suportadas.

Capítulo 5

Conclusões

Neste capítulo será feita a síntese de todo o trabalho desenvolvido durante este projeto, como também dos principais resultados obtidos ao longo das várias etapas conduzidas. Por fim, serão apresentados alguns dos tópicos de trabalho futuro.

5.1 Resumo do Trabalho Desenvolvido

De forma a adquirir os conhecimentos fundamentais para a elaboração e desenvolvimento do *PaaSManager*, foram estudados os diversos conceitos associados ao *cloud computing*, alguns dos fornecedores de PaaS com maior quota no mercado, como igualmente as iniciativas de interoperabilidade existentes nesta área de investigação e o desenvolvimento de aplicações segundo conceitos SOA. O *cloud computing* caracteriza-se fundamentalmente por ser *on-demand*, com modelos de negócio *pay-per-use*, promovendo uma rápida elasticidade de recursos computacionais que acompanha a procura em tempo-real. Os serviços de *cloud* são geralmente organizados em 3 tipos: *Infrastructure-as-a-Service*, *Platform-as-a-Service* e *Software-as-a-Service*, que se complementam ao longo de um modelo por camadas. Apesar de terem sido estudados os 3 serviços, o serviço de PaaS foi o mais discutido ao longo deste trabalho sendo enaltecidas as suas vantagens e analisados os vários fornecedores existentes no atual mercado. Nos últimos anos surgiram algumas iniciativas de interoperabilidade entre fornecedores de soluções *cloud* incidindo nomeadamente no serviço de IaaS. Porém, sendo objetivo deste trabalho definir uma solução para a camada de PaaS, foram investigadas algumas iniciativas relevantes seja o exemplo do CAMP. Por

fim, os conceitos SOA aliados ao *cloud computing* permitiram retirar conhecimento sobre como estas duas áreas se interligam trazendo benefícios para o desenvolvimento de serviços fiáveis e interoperáveis.

O desenho da solução envolveu inicialmente uma análise das APIs das diversas plataformas selecionadas bem como a definição de funcionalidades essenciais que a camada agregadora de fornecedores de PaaS deveria suportar. Esta atividade permitiu especificar vários módulos que sustentassem operações relacionadas com a gestão e a aquisição de informação sobre aplicações e bases de dados e processos de migração de aplicações. Um dos módulos, designado por *Management Resources*, é responsável por suportar apenas as funcionalidades de gestão tendo sido ilustrados alguns diagramas de sequência de operações fundamentais, nomeadamente, criar uma aplicação, realizar o *deployment* do respetivo código fonte e migrar aplicações entre diferentes fornecedores. O módulo *Information Resources*, que implementa métodos de aquisição de informação sobre aplicações e bases de dados, foi detalhado tendo sido igualmente apresentado um diagrama de sequência de uma operação de aquisição de estado de uma aplicação. Para a monitorização surge o *Monitoring Engine* que é um elemento indispensável para a recolha de estatísticas em tempo-real, porém, orientada às métricas que cada plataforma fornece. Todas estas funcionalidades são depois expostas através de uma API RESTFul que abstrai as diferenças entre os vários fornecedores suportados. Para finalizar, foi apresentada a *framework* de serviços baseados em informação de contexto da PTIN que é constituída por algumas aplicações que serão portadas para um ambiente PaaS através da arquitetura desenvolvida.

Para avaliar o *PaaSManager* em cenários reais foram definidos casos de estudo. Um dos cenários propostos tinha por base um módulo que recomenda ao utilizador qual a plataforma mais adequada para uma aplicação, por exemplo, perante o perfil de dependências técnicas da mesma. Através de um interface *web* integrado com esse recomendador e por sua vez com o *PaaSManager*, foi possível criar, gerir e monitorizar algumas das aplicações de contexto de forma unificada qualquer que fosse o fornecedor de plataforma. Finalmente foram realizados testes de carga para avaliar o desempenho da solução *PaaSManager*. Os resultados obtidos revelaram que a arquitetura não introduzia um *overhead* significativo na maior parte das operações suportadas. Sendo assim, interagir com o *PaaSManager* não coloca mais complexidade nem origina um muito maior tempo de resposta em comparação a interagir diretamente com cada fornecedor.

Em suma, esta solução é neste momento uma das únicas nesta área de investigação de

interoperabilidade em ambientes PaaS que possui uma implementação estável e com resultados operacionais. Recentemente a temática discutida ao longo deste trabalho tem vindo a receber uma grande atenção por parte da comunidade, esperando-se novas iniciativas e projetos que tencionem dar aos utilizadores a oportunidade de controlar várias plataformas de forma unificada.

5.2 Principais Contribuições

A principal contribuição desta dissertação foi a definição e desenvolvimento de uma camada de abstração que visasse unificar os processos de gestão e aquisição de informação sobre aplicações criadas através de diversos PaaS, de modo a combater o *lock-in* existente no mercado. Esta temática tem vindo a ser muito discutida na área de investigação da interoperabilidade e portabilidade em ambientes *cloud*. Apesar da existência de algumas iniciativas de projetos europeus ou de cooperação entre grandes empresas, este trabalho verifica-se como sendo das primeiras implementações com resultados práticos. Além disso, o sistema desenvolvido foi integrado com um recomendador de PaaS e respetivo interface *web* de gestão que poderá tornar-se atrativo para os utilizadores deste tipo de serviços de *cloud*. Como fruto de uma parte significativa deste trabalho, foi publicado o artigo [5], apresentado na conferência internacional CLOSER 2012 [6]. Adicionalmente, foi publicado um outro artigo [7], que abrange um dos casos de estudo propostos, na conferência CRC 2012 [8]. E por fim, foi submetido um artigo, que incide exclusivamente na solução *PaaS-Manager*, na conferência SAC 2013 [9]. De momento o artigo encontra-se em processo de revisão.

5.3 Trabalho Futuro

Após o desenvolvimento do *PaaSManager* e da sua aplicação em cenários reais através de um recomendador como o Cloud Service Broker, surgem novos desenvolvimentos que poderão ser efetuados orientados aos utilizadores de PaaS.

A importação de informação para as bases de dados criadas, como a sua posterior exportação, permitirá uma rápida e atrativa migração de aplicações já existentes para ambientes *cloud*. Para isso terão de ser averiguadas as diferentes formas de acesso que cada PaaS fornece de forma a suportar estas novas funcionalidades através do *PaaSManager*. A

lógica destas operações poderá ser reutilizada para desenvolver um método de migração de base de dados entre diferentes fornecedores.

Com o grande número de PaaS existentes no mercado, novos *adapters* poderão ser implementados estendendo o ecossistema de fornecedores suportados. Alguns exemplos de fornecedores são por exemplo o Red Hat OpenShit, o AppFog entre outros. Este processo envolverá estudar às suas APIs, encontrar as semelhanças nos processos e funcionalidades suportadas, de forma a desenvolver os módulos necessários integrando-os com a lógica já existente no *Management Resources* e *Information Resources*.

Dentro do grupo PT, existe a iniciativa SmartCloudPT que se foca maioritariamente no fornecimento de serviços de IaaS e SaaS para as empresas. No entanto, com o elevado crescimento do interesse em soluções orientadas ao desenvolvimento e *deployment* de aplicações, os resultados obtidos neste projeto poderão ser integrados com essa iniciativa e hospedados num ambiente *cloud*. Além disso, novos cenários que envolvam ambientes de PaaS híbridos e privados sobre a infraestrutura SmartCloudPT, serão analisados com o intuito de adquirir o conhecimento que permita a Portugal Telecom fornecer soluções *Platform-as-a-Service* mais orientadas ao requisitos de cada utilizador.

Por fim com a importância que a iniciativa CAMP está a obter nesta área de investigação, faz todo o sentido que a API que será disponibilizada seja implementada pelo *PaaSManager*. Por um lado, devido a possibilitar a extensão do ecossistema atual com as plataformas suportadas pelo CAMP, e por outro lado, devido ao *PaaSManager* tornar-se provavelmente das primeiras soluções que suportam esta recente e interessante iniciativa.

Bibliografia

[1] Shuai Zhang, Shufen Zhang, Xuebin Chen, and Xiuzhen Huo. Cloud computing research and development trend. In *Proceedings of the 2010 2nd International Conference on Future Networks*, ICFN'10, pages 93–97, Washington, DC, USA, 2010. IEEE Computer Society.

[2] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[3] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616, June 2009.

[4] Nikolaos Loutas, Eleni Kamateri, Filippo Bosi, and Konstantinos Tarabanis. Cloud computing interoperability: The state of play. In *Proceedings of the 2011 IEEE 3rd International Conference on Cloud Computing Technology and Science*, CLOUDCOM '11, pages 752–757, Washington, DC, USA, 2011. IEEE Computer Society.

[5] David Cunha, Pedro Neves, and Pedro Sousa. Interoperability and portability of cloud service enablers in a PaaS environment. In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science*, CLOSER'12, pages 432–437, Oporto, Portugal, 2012. SciTePress.

[6] CLOSER'12. 2nd international conference on cloud computing and services science. `http://closer.scitevents.org/?y=2012`, Consultado em Outubro 2012.

[7] Carlos Gonçalves, David Cunha, Pedro Neves, Pedro Sousa, João Paulo Barraca, and Diogo Gomes. Towards a cloud service broker for the meta-cloud. In *Proceedings of the 12ª Conferência sobre Redes de Computadores*, CRC'12, Aveiro, Portugal, 2012.

[8] CRC'12. 12ªconferência sobre redes de computadores. `http://crc2012.av.it.pt/`, Consultado em Outubro 2012.

[9] SAC'13. 28th symposium on applied computing. `http://www.acm.org/conferences/sac/sac2013/`, Consultado em Outubro 2012.

[10] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, May 2010.

[11] Robert L. Grossman. The case for cloud computing. *IT Professional*, 11(2):23–27, March 2009.

[12] National Institute of Standards and Technology. The NIST definition of cloud computing. Technical report, July 2009.

[13] Rajkumar Buyya, James Broberg, and Andrzej M. Goscinski. *Cloud Computing Principles and Paradigms*. Wiley Publishing, 1st edition, 2011.

[14] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: Issues and challenges. In *Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications*, AINA'10, pages 27–33, Washington, DC, USA, 2010. IEEE Computer Society.

[15] Marios D. Dikaiakos, Dimitrios Katsaros, Pankaj Mehra, George Pallis, and Athena Vakali. Cloud computing: Distributed internet computing for it and scientific research. *IEEE Internet Computing*, 13(5):10–13, September 2009.

[16] George Pallis. Cloud computing: The new frontier of internet computing. *IEEE Internet Computing*, 14(5):70–73, September 2010.

[17] Neal Leavitt. Is cloud computing really ready for prime time? *Computer*, 42(1):15–20, January 2009.

[18] John F. Gantz, Anna Toncheva, and Stephen Minton. IDC cloud computing's role in job creation. Technical report, March 2012.

[19] Ben Kepes. Rackspace CLOUDU say goodbye to diy data centers an infrastructure-as-a-service intensive. Technical report, 2012.

[20] George Reese. *Cloud Application Architectures: Building Applications and Infrastructure in the Cloud*. O'Reilly Media, Inc., 1st edition, 2009.

[21] David S. Linthicum. *Cloud Computing and SOA Convergence in Your Enterprise: A Step-by-Step Guide*. Addison-Wesley Professional, 1st edition, 2009.

[22] Nick Antonopoulos and Lee Gillam. *Cloud Computing: Principles, Systems and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[23] Cloud Computing Use Case Discussion Group. Cloud computing use cases whitepaper 4.0. Technical report, 2010.

[24] Ben Kepes. Rackspace CLOUDU understanding the cloud computing stack SaaS, PaaS, IaaS. Technical report, 2012.

[25] Toby Velte, Anthony Velte, and Robert Elsenpeter. *Cloud Computing, A Practical Approach*. McGraw-Hill, Inc., 1st edition, 2010.

[26] Amazon Web Services SLA. `http://aws.amazon.com/ec2-sla/`, Consultado em Julho 2012.

[27] M. Zhou, R. Zhang, D. Zeng, and W. Qian. Services in the cloud computing era: A survey. In *Proceedings of the 2010 4th International Universal Communication Symposium, IUCS 2010*, pages 40–46, Beijing, China, 2010.

[28] Amazon Web Services. `http://aws.amazon.com/`, Consultado em Julho 2012.

[29] Daniel Beimborn, Thomas Miletzki, and Stefan Wenzel. Platform as a service. *Business & Information Systems Engineering*, 3(6):381–384, October 2011.

[30] Stefan Ried. Platform-as-a-service market sizing. Technical report, July 2009.

[31] Michael P. McGrath. *Understanding PaaS*. O'Reilly Media, Inc., 1st edition, 2012.

[32] CloudBees. `http://www.cloudbees.com/`, Consultado em Agosto 2012.

[33] CloudFoundry. `http://www.cloudfoundry.com/`, Consultado em Agosto 2012.

[34] Heroku. `http://www.heroku.com/`, Consultado em Agosto 2012.

[35] OpenShift. `https://openshift.redhat.com/`, Consultado em Agosto 2012.

[36] Amazon Web Services Beanstalk. `http://aws.amazon.com/elasticbeanstalk/`, Consultado em Agosto 2012.

[37] Google App Engine. `https://developers.google.com/appengine/`, Consultado em Agosto 2012.

[38] Michael Cusumano. Cloud computing and SaaS as new computing platforms. *Commun. ACM*, 53(4):27–29, April 2010.

[39] Google. `http://www.google.com/`, Consultado em Julho 2012.

[40] Salesforce.com. `http://www.salesforce.com/eu/`, Consultado em Julho 2012.

[41] Dana Petcu. Portability and interoperability between clouds: challenges and case study. In *Proceedings of the 4th European conference on Towards a service-based internet*, ServiceWave'11, pages 62–74, Berlin, Heidelberg, 2011. Springer-Verlag.

[42] OGF Open Cloud Computing Interface Working Group. `http://occi-wg.org/`, Consultado em Agosto 2012.

[43] Apache DeltaCloud. `http://deltacloud.apache.org/`, Consultado em Agosto 2012.

[44] Cloud4SOA. `http://www.cloud4soa.eu/`, Consultado em Agosto 2012.

[45] Cloud Application Management for Platforms (CAMP). `http://www.cloudspecs.org/paas/`, Consultado em Setembro 2012.

[46] Mark Carlson et al. Cloud application management for platforms (camp). Technical report, August 2012.

[47] Nicolai Josuttis. *Soa in Practice: The Art of Distributed System Design*. O'Reilly Media, Inc., 1st edition, 2007.

[48] JBoss Switchyard. `http://www.jboss.org/switchyard/`, Consultado em Julho 2012.

[49] K. Ramana, T.Hari Krishna, C.V.Lakshmi Narayana, and M.Sankara Prasanna Kumar. Comparative analysis on cloud computing and service oriented architecture. *International Journal of Advanced Research In Technology*, 1(1):22–28, September 2011.

[50] Leonard Richardson and Sam Ruby. *RESTFul Web Services*. O'Reilly Media, Inc., 1st edition, 2007.

[51] Roy Thomas Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[52] Jin Shao and Qianxiang Wang. A performance guarantee approach for cloud applications based on monitoring. In *Proceedings of the 2011 IEEE 35th Annual Computer Software and Applications Conference Workshops*, COMPSACW'11, pages 25–30, Washington, DC, USA, 2011. IEEE Computer Society.

[53] Toni Mastelic, Vincent C. Emeakaroha, Michael Maurer, and Ivona Brandic. M4cloud - generic application level monitoring for resource-shared cloud environments. In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science*, CLOSER'12, pages 522–532, Oporto, Portugal, 2012. SciTePress.

[54] Xu Cheng, Shi Yuliang, and Li Qingzhong. A multi-tenant oriented performance monitoring, detecting and scheduling architecture based on SLA. In *Proceedings of Joint Conferences on Pervasive Computing*, JCPC 2009, pages 599–604, Tamsui, Taipei, 2009. IEEE Internet Computing.

[55] Diogo Gomes, João M. Gonçalves, Ricardo Otero Santos, and Rui Aguiar. XMPP based Context Management Architecture. In *Proceedings of the 2010 IEEE Globecom Workshops*, pages 1372–1377, Miami, Florida, USA, December 2010. IEEE.

[56] C-CAST. `http://www.ict-ccast.eu/`, Consultado em Agosto 2012.

[57] Ingnite Realtime Openfire. `http://www.igniterealtime.org/projects/openfire/`, Consultado em Agosto 2012.

[58] Telma Mota, Massimo Valla, and Diogo Gomes. Context-Aware Content Casting. In *Proceedings of the 2010 Networked & Electronic Media*, Barcelona, Spain, 2010.

[59] Apache JMeter. `http://jmeter.apache.org/`, Consultado em Setembro 2012.

Apêndice A

API RESTful do PaaSManager

Neste apêndice encontra-se documentada a API do *PaaSManager* detalhando os vários métodos suportados, bem como os parâmetros de entrada necessários para cada operação e alguns exemplos de pedidos e respostas JSON.

A.1 Serviços de Gestão

A.1.1 createApp : ApplicationCreateResponse

Descrição

Este método cria uma aplicação em um determinado PaaS

***Resource* URL**

POST /mgmt/apps/create/:paasProvider/:appID?appFramework=:framework

Parâmetros

Paramêtro	***Tipo***	***Descrição***
paasProvider	String	Identificação do PaaS
appID	String	Identificação da aplicação
appFramework	String	Identificação da framework suportada pela aplicação

Request

POST "api-key:key"http://"server"/paasmanager/v1/mgmt/apps/create/ cloudbees/aplicacaoteste? appFramework=JAVA_EE6_WEB_PROFILE

***Response* JSON**

```
200 OK
{
"paasProvider": "CLOUDBEES",
"appID": "aplicacaoteste",
"appStatus": "created",
"appUrl": "aplicacaoteste.paasmanager.cloudbees.net"
}
```

A.1.2 deployApp : ApplicationCreateResponse

Descrição

Este método realiza o respetivo *deployment* do código fonte de uma aplicação em um determinado PaaS

***Resource* URL**

POST /mgmt/apps/deploy

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação
appData	byte[]	.zip com o código fonte da aplicação

Request

POST "api-key:key"http://"server"/paasmanager/v1/mgmt/apps/deploy

***Response* JSON**

```
200 OK
{
  "paasProvider": "CLOUDBEES",
  "appID": "aplicacaoteste",
  "appStatus": "deployed",
  "appUrl": "aplicacaoteste.paasmanager.cloudbees.net"
 }
```

A.1.3 migrateApp : ApplicationCreateResponse

Descrição

Este método efetua a migração de uma aplicação para um determinado PaaS

***Resource* URL**

POST /mgmt/apps/migrate/:newpaasprovider/:appID

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação
newpaasprovider	String	Identificação do PaaS para onde se deseja migrar a aplicação

Request

POST "api-key:key"http://"server"/paasmanager/v1/mgmt/apps/migrate/ cloudfoundry/aplicacaoteste

***Response* JSON**

```
200 OK
{
    "paasProvider": "CLOUDFOUNDRY",
    "appID": "aplicacaoteste",
    "appStatus": "deployed",
    "appUrl": "aplicacaoteste.cloudfoundry.com"
 }
```

A.1.4 startApp : ApplicationStartResponse

Descrição

Este método efectua o *start* de uma aplicação

***Resource* URL**

POST /mgmt/apps/start/:appID

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação

Request

POST "api-key:key"http://"server"/paasmanager/v1/mgmt/apps/start/aplicacaoteste

***Response* JSON**

```
200 OK
{
    "paasProvider": "CLOUDBEES",
    "appID": "aplicacaoteste",
    "appStatus": "started",
    "appUrl": "aplicacaoteste.paasmanager.cloudbees.net"
 }
```

A.1.5 stopApp : ApplicationStopResponse

Descrição

Este método efectua o *stop* de uma aplicação

***Resource* URL**

POST /mgmt/apps/stop/:appID

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação

Request

POST "api-key:key"http://"server"/paasmanager/v1/mgmt/apps/stop/aplicacaoteste

***Response* JSON**

```
200 OK
{
    "paasProvider": "CLOUDBEES",
    "appID": "aplicacaoteste",
    "appStatus": "stopped",
```

```
    "appUrl": "aplicacaoteste.paasmanager.cloudbees.net"
}
```


A.1.6 restartApp : ApplicationRestartResponse

Descrição

Este método efectua o *restart* de uma aplicação

***Resource* URL**

POST /mgmt/apps/restart/:appID

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação

Request

POST "api-key:key"http://"server"/paasmanager/v1/mgmt/apps/restart/aplicacaoteste

***Response* JSON**


```
200 OK
{
    "paasProvider": "CLOUDBEES",
    "appID": "aplicacaoteste",
    "appStatus": "restarted",
    "appUrl": "aplicacaoteste.paasmanager.cloudbees.net"
}
```


A.1.7 deleteApp : ApplicationDeleteResponse

Descrição

Este método elimina uma aplicação

***Resource* URL**

DELTE /mgmt/apps/delete/:appID

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação

Request

DELETE "api-key:key" http://"server"/paasmanager/v1/mgmt/apps/delete/ aplicacaoteste

***Response* JSON**

```
200 OK
{
    "paasProvider": "CLOUDBEES",
    "appID": "aplicacaoteste",
    "appStatus": "deleted"
}
```

A.1.8 scaleApp : ApplicationScaleResponse

Descrição

Este método efetua o *scaling* horizontal de uma aplicação até ao número de instâncias pretendidas

***Resource* URL**

POST /mgmt/apps/scale/:appID?appInstances=:num

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação
appInstances	int	Número de instâncias pretendidas

Request

POST "api-key:key" http://"server"/paasmanager/v1/mgmt/apps/scale/

aplicacaoteste?appInstances=4

***Response* JSON**

```
200 OK
{
```

```
  "paasProvider": "CLOUDBEES",
  "appID": "aplicacaoteste",
  "appStatus": "scaled to 4 instances",
  "appUrl": "aplicacaoteste.paasmanager.cloudbees.net"
}
```


A.1.9 updateApp : ApplicationCreateResponse

Descrição

Este método realiza a respetiva atualização do código fonte de uma aplicação

***Resource* URL**

PUT /mgmt/apps/update

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação
appData	byte[]	.zip com o source code da aplicação

Request

PUT "api-key:key"http://"server"/paasmanager/v1/mgmt/apps/update

***Response* JSON**


```
200 OK
{
  "paasProvider": "CLOUDBEES",
  "appID": "aplicacaoteste",
  "appStatus": "deployed",
  "appUrl": "aplicacaoteste.paasmanager.cloudbees.net"
}
```


A.1.10 createService : ServiceCreateResponse

Descrição

Este método associa uma base de dados a uma determinada aplicação

***Resource* URL**

POST /mgmt/services/create/:appID/:serviceID?serviceVendor=:vendor

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação associada a base de dados
serviceID	String	Identificação da base de dados
serviceVendor	String	Identificação do *vendor* da base de dados

Request

POST "api-key:key"http://"server"/paasmanager/v1/mgmt/services/create/webapp/cloud_db_app?serviceVendor=POSTGRESQL_9_0

***Response* JSON**

```
200 OK
{
    "appID" : "webapp",
    "serviceID": "cloud_db_app",
    "serviceVendor": "POSTGRESQL_9_0"
 }
```

A.1.11 deleteService : ServiceDeleteResponse

Descrição

Este método elimina uma base de dados

***Resource* URL**

DELETE /mgmt/services/delete/:appID/:serviceID

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação associada a base de dados
serviceID	String	Identificação da base de dados

Request

DELETE "api-key:key"http://"server"/paasmanager/v1/mgmt/services/ delete/webapp/-

cloud_db_app

***Response* JSON**

```
200 OK
{
    "paasProvider": "CLOUDFOUNDRY",
    "appID" : "webapp",
    "serviceID": "cloud_db_app",
    "serviceStatus": "deleted"
 }
```

A.2 Serviços de Informação

A.2.1 getAppStatus : ApplicationStatusResponse

Descrição

Este método devolve o estado de uma aplicação (*running, stopped, crashed, unknown*)

***Resource* URL**

GET /info/apps/:appID/status

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação

Request

GET "api-key:key"http://"server"/paasmanager/v1/info/apps/webapp/status

***Response* JSON**

```
200 OK
{
    "paasProvider": "CLOUDFOUNDRY",
    "appID": "webapp",
    "appStatus": "running",
    "appUrl": "aplicacaoweb.cloudfoundry.com"
```

}

A.2.2 getAppStatistics : ApplicationStatisticsResponse

Descrição

Este método devolve estatísticas em tempo-real de uma aplicação

***Resource* URL**

GET /info/apps/:appID/statistics

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação

Request

GET "api-key:key"http://"server"/paasmanager/v1/info/apps/webapp/statistics

***Response* JSON**

```
200 OK
{
"application": {
"instance": [
{
"id": "1",
"date": "2012-08-05 16:16:04.0",
"metric": [
{
"name": "diskquota",
"metricValue": "2048",
"unit": "MB"
},
{
"name": "usage_cpu",
"metricValue": "0.0",
"unit": "%"
},
```

```
{
"name": "disk",
"metricValue": "11",
"unit": "MB"
},
{
"name": "uptime",
"metricValue": "58h:33m:49s",
"unit": "h:m:s"
},
{
"name": "memquota",
"metricValue": "512",
"unit": "MB"
},
{
"name": "memory",
"metricValue": "135.01",
"unit": "MB"
},
{
"name": "fdsquota",
"metricValue": "256",
"unit": ""
}
]
}
],
"appID": "webapp"
}
}
```

A.2.3 getAppInfo : ApplicationInfoResponse

Descrição

Este método devolve informação útil sobre uma aplicação criada

***Resource* URL**

GET /info/apps/:appID

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação

Request

GET "api-key:key"http://"server"/paasmanager/v1/info/apps/webapp

***Response* JSON**

```
200 OK
{
    "paasProvider": "CLOUDFOUNDRY",
    "appID": "webapp",
    "appStatus": "running",
    "appUrl": "aplicacaoweb.cloudfoundry.com",
    "appMemory": {
          "value" : "512",
          "unit": "MB"},
    "appInstances": "2",
    "appFramework": "JAVA_EE6_WEB_PROFILE",
    "appServices":
      "serviceID": [
           "mysql-teste",
           "mongodb"]
 }
```

A.2.4 getAppListInfo : ApplicationListInfoResponse

Descrição

Este método devolve lista e respetiva informação sobre aplicações criadas em um determinado PaaS

***Resource* URL**

GET /info/apps/:paasProvider/list

Parâmetros

Paramêtro	***Tipo***	***Descrição***
paasProvider	String	Identificação do PaaS

Request

GET "api-key:key"http://"server"/paasmanager/ v1/info/apps/cloudfoundry/list

***Response* JSON**

```
200 OK
{
"applicationInfoResponse": [
    {
     "appFramework": "JAVA_EE6_WEB_PROFILE",
     "appStatus": "running",
     "appID": "CtxE",
     "appUrl": "CtxE.cloudfoundry.com",
     "appMemory": {
          "value": "512",
          "unit": "MB"},
     "appServices": {
     "serviceID": [ ]
      }
    "paasProvider": "CLOUDFOUNDRY",
    "appInstances": "1"
    },
    {
    "appFramework": "JAVA_WEB_APP",
    "appStatus": "running",
    "appID": "xpto",
    "appUrl": "xpto.cloudfoundry.com",
```

```
    "appMemory": {
        "value": "512",
        "unit": "MB"},
    "appServices": {
    "serviceID": [
        "mysql-teste",
        "mongodb"]
      },
    "paasProvider": "CLOUDFOUNDRY",
    "appInstances": "1"
    }
  ]
}
```


A.2.5 getServiceInfo : ServiceInfoResponse

Descrição

Este método devolve informação útil sobre uma base de dados associada a uma aplicação

***Resource* URL**

GET /info/services/:appID/:serviceID

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação associada a base de dados
serviceID	String	Identificação da base de dados

Request

GET "api-key:key"http://"server"/paasmanager/v1/info/services/webapp/cloud_db_app

***Response* JSON**

200 OK


```
{
    "paasProvider": "CLOUDBEES",
    "appID": "webapp",
```

```
    "serviceID": "cloud_db_app",
    "serviceVendor": "MYSQL_5_0_51",
    "serviceUsername": "foo-barU",
    "servicePassword": "foo-barP",
    "serviceUrl": "ec2-50-19-213-178.compute-1.amazonaws.com"
 }
```

A.2.6 getServiceAppListInfo : ServiceInfoListResponse

Descrição

Este método devolve lista e respetiva informação sobre bases de dados associadas a uma aplicação e criadas em um determinado PaaS

***Resource* URL**

GET /info/services/apps/:appID/list

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação associada às bases de dados

Request

GET "api-key:key"http://"server"/paasmanager/v1/info/services/apps/webapp/list

***Response* JSON**

```
200 OK
{
 "ServiceInfoResponse": [
    {
      "paasProvider": "CLOUDBEES",
      "appID": "webapp",
      "serviceID": "cloud_db_app",
      "serviceVendor": "MYSQL_5_0_51",
      "serviceUsername": "foo-barU",
      "servicePassword": "foo-barP",
      "serviceUrl": "ec2-50-19-213-178.compute-1.amazonaws.com"
    },
```

```
    {
      "paasProvider": "CLOUDBEES",
      "appID": "webapp",
      "serviceID": "database",
      "serviceVendor": "MYSQL_5_0_51",
      "serviceUsername": "123",
      "servicePassword": "abc!",
      "serviceUrl": "ec2-50-19-213-178.compute-1.amazonaws.com"
    },
  ]
}
```


A.2.7 getServiceListInfo : ServiceInfoListResponse

Descrição

Este método devolve lista e respetiva informação sobre bases de dados criadas em um determinado PaaS

***Resource* URL**

GET /info/services/:paasProvider/ list

Parâmetros

Paramêtro	***Tipo***	***Descrição***
paasProvider	String	Identificação do PaaS

Request

GET "api-key:key"http://"server"/paasmanager/v1/info/services/cloudbees/list

***Response* JSON**


```
200 OK
{
 "ServiceInfoResponse": [
    {
      "paasProvider": "CLOUDBEES",
      "appID": "webapp",
      "serviceID": "cloud_db_app",
```

```
    "serviceVendor": "MYSQL_5_0_51",
    "serviceUsername": "foo-barU",
    "servicePassword": "foo-barP",
    "serviceUrl": "ec2-50-19-213-178.compute-1.amazonaws.com"
  },
  {
    "paasProvider": "CLOUDBEES",
    "appID": "webapp",
    "serviceID": "database",
    "serviceVendor": "MYSQL_5_0_51",
    "serviceUsername": "123",
    "servicePassword": "abc!",
    "serviceUrl": "ec2-50-19-213-178.compute-1.amazonaws.com"
  },
 {
    "paasProvider": "CLOUDBEES",
    "appID": "aplicacao",
    "serviceID": "mysql-db-app",
    "serviceVendor": "MYSQL_5_0_51",
    "serviceUsername": "user",
    "servicePassword": "passwd",
    "serviceUrl": "ec2-50-19-213-178.compute-1.amazonaws.com"
  }
 ]
}
```


A.2.8 getAppLogs : ApplicationLogsResponse

Descrição

Este método devolve informação de *logs* de uma aplicação

***Resource* URL**

GET /info/apps/:appID/logs

Parâmetros

Paramêtro	***Tipo***	***Descrição***
appID	String	Identificação da aplicação

Request

GET "api-key:key"http://"server"/paasmanager/v1/info/apps/webapp/logs

***Response* JSON**

```
200 OK
{
    "paasProvider": "CLOUDBEES",
    "appID": "webapp",
    "appLog": "conteudo do log",
 }
```

A.2.9 getPaaSOffering : PaasProviders

Descrição

Este método devolve lista de tecnologias e métricas de monitorização suportadas pelos fornecedores de PaaS implementados pelo *PaaSManager*

***Resource* URL**

GET /info/paas/offering

Parâmetros -

Request

GET http://"server"/paasmanager/v1/info/paas/offering

***Response* JSON**

```
200 OK
{
"paasProvider": [
{
"name": "CLOUDBEES",
"services": {
"service": [
```

```
{
"name": "MySQL",
"version": "5.0.51",
"info": "info",
"id": "MYSQL_5_0_51"
}
]
},
"runtimes": {
"runtime": [
{
"name": "Java",
"version": "1.6",
"info": "info",
"id": "JAVA_1_6"
},
{
"name": "Scala",
"version": "",
"info": "info",
"id": "SCALA"
}
]
},
"frameworks": {
"framework": [
{
"name": "Grails",
"version": "",
"info": "info",
"id": "GRAILS"
},
{
"name": "Java EE6 Web Profile",
```

```
"version": "",
"info": "info",
"id": "JAVA_EE6_WEB_PROFILE"
},
{
"name": "Play!",
"version": "",
"info": "info",
"id": "PLAY"
},
{
"name": "Spring",
"version": "",
"info": "info",
"id": "SPRING"
}
]
}
},
"monitoringMetrics": {
"metric": [
{
"name": "apdex",
"info": "info",
},
{
"name": " usagecpu ",
"info": "info",
},
{
"name": "memory",
"info": "info",
},
{
```

```
"name": "responsetime",
"info": "info",
},
{
"name": "throughput",
"info": "info",
},
{
"name": "usagedatabase",
"info": "info",
},
]
},
...
```

A.3 Códigos de Estado

Código de Estado	***Descrição***
200 OK	*The operation was successful*
400 Bad Request	*The request cannot be fulfilled due to bad syntax*
401 Unauthorized	*You are attempting to access the API with invalid credentials*
404 Not Found	*The API endpoint or resource you are attempting to fetch does not exist*
500 Internal Server Error	*The server encountered an unexpected condition which prevented it from fulfilling the request*

***Response* JSON (Erro)**

```
{
  "errorCode": "codigo de estado",
  "errorMessage": "mensagem de erro"
 }
```