Universidade do Minho
Escola de Engenharia

Diogo Alexandre Gonçalves Machado

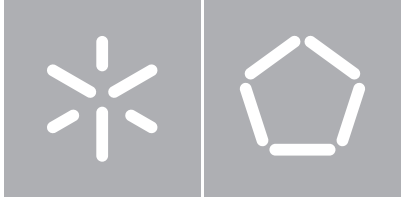
Otimização dos custos de operação de aplicações Web em Cloud

novembro de 2019

Otimização dos custos de operação de aplicações Web em Cloud

Diogo Alexandre Gonçalves Machado

UMinho | 2019



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Diogo Alexandre Gonçalves Machado

Otimização dos custos de operação de aplicações Web em Cloud

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de

Professor António Luís Pinto Ferreira Sousa
Daniela Duarte da Costa

novembro de 2019

Agradecimentos

Em primeiro lugar, gostaria de agradecer ao professor António Luís Sousa pela oportunidade de realizar esta dissertação de mestrado e por todo o acompanhamento e dedicação ao longo do desenvolvimento da mesma.

A toda a equipa de engenharia da Eurotux Informática, S.A., o meu agradecimento por todo o apoio e conhecimento transmitido. Aos meus diretores de equipa Nuno Fernandes e Daniela Costa, um obrigado pela ajuda prestada na concretização deste projeto. Ao Bruno Costa, por todo o tempo e empenho dedicado ao longo destes meses na transmissão e discussão de ideias, a minha especial gratidão.

À minha família e amigos, o meu muito obrigado por toda a compreensão e apoio dado ao longo deste percurso. A vossa persistência permitiu que eu nunca desistisse dos meus sonhos e alcançasse os meus objetivos.

Por último, o meu enorme agradecimento ao Rui Leite por todas as ideias, ajuda e disponibilidade prestada. Ter caminhado ao meu lado durante esta jornada foi incrível.

DECLARAÇÃO DE INTEGRIDADE

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

Resumo

Otimização dos custos de operação de aplicações Web em Cloud

O *cloud computing* tem sido amplamente adotado na área das tecnologias de informação na última década, devido às diversas vantagens que providencia, entre elas a possibilidade de redução de custos com infraestruturas.

Embora a utilização da *cloud* possa minorar os custos de operação de aplicações Web, verifica-se que a definição dos preços praticados pelos fornecedores de serviços tem-se tornado cada vez mais complexa, ameaçando uma das principais razões que leva os utilizadores a migrar para a *cloud*: a redução de custos.

Derivado deste aumento de complexidade, o surgimento de soluções de monitorização e otimização de custos de *cloud* tem vindo a aumentar por forma a combater este problema. Apesar de existirem algumas soluções capazes de auxiliar na otimização de custos, verifica-se que a visibilidade sobre os custos e dados de utilização é limitada, não sendo possível consultar a informação com a granularidade que os utilizadores pretendem.

Por todos estes motivos, a equipa de Investigação e Desenvolvimento da Eurotux Informática, S.A. decidiu investir no desenvolvimento de uma solução que auxiliasse os seus colaboradores e clientes num problema que enfrentam no dia a dia. Após estudar as soluções existentes, identificou-se, junto dos principais intervenientes, os requisitos que a solução deveria cumprir. A criação de uma aplicação em *Flask* em conjunto com uma *Elastic Stack* constitui a base tecnológica da solução. A modularidade, escalabilidade e robustez da solução foi tida em conta em todo o processo de elaboração da solução.

O resultado final é uma ferramenta totalmente funcional que permite satisfazer as necessidades impostas. A integração com os principais fornecedores de *cloud* estudados foi amplamente conseguida. A avaliação da mesma foi realizada tendo por base diversos casos de estudo de clientes reais da empresa.

Palavras-chave: *cloud-computing*, fornecedores de serviços de *cloud*, otimização de custos

Abstract

Cloud web application operation cost optimization

Cloud computing has been widely adopted in the area of Information Technology (IT) over the last decade due to the many advantages it provides, including the possibility of reducing infrastructure costs.

Although the adoption of the cloud can mitigate the costs of operating web applications, it appears that the definition of prices practiced by service providers has become increasingly complex, threatening one of the main reasons that leads users to migrate to the cloud: cost savings.

Due to this increased complexity, the number of cloud cost monitoring and optimization solutions has been growing in order to face this problem. While there are some solutions that can help with cost optimization, it is verified that the visibility into costs and usage data is limited, and it is not possible to query the information with the granularity that users want.

For all these reasons, the Research and Development team (R&D) of Eurotux Informática, S.A. decided to invest in the development of a solution that would help its employees and customers in a problem they face every day. After studying existing solutions, the key actors identified the requirements that the solution should sastified. Creating a Flask application together with an Elastic Stack is the technology foundation of the solution. The modularity, scalability and robustness of the solution has been taken in consideration throughout the solution design process.

The final result is a fully functional tool that allows you to meet the requirements imposed. The integration with the major cloud providers studied has been largely achieved. The evaluation was made based on several study cases of real company customers.

Keywords: cloud-computing, cloud service providers, cost optimization

Conteúdo

Acrónimos

ABC Abstract Base Classe.
ALB Application Load Balancer.
API Application Programming Interface.
ASG Auto-Scaling Group.
AWS Amazon Web Services.

CMDB Configuration Management Database.
CMP Cloud Management Platform.
CPU Central Processing Unit.
CSP Cloud Service Provider.
CSV Comma-Separated Values.
CUR Cost and Usage Report.

DBMS Database Management System.
DBR Detailed Billing Report.
DNS Domain Name System.
DRA Durable Reduced Availability.
DSR Design Science Research.

EBS Elastic Block Store.
EC2 Elastic Compute Cloud.
ECS Elastic Container Service.
ELB Elastic Load Balancing.
ERP Enterprise Resource Planning.
ETL Extract, Transform, Load.

GB Gigabyte.
GCP Google Cloud Platform.
GCS Google Cloud Storage.

HTTP Hypertext Transfer Protocol.

I&O Infraestrutura e Operações.
IaaS Infrastructure-as-a-Service.
IAM Identity and Access Management.
IDG International Data Group.

JSON JavaScript Object Notation.
JVM Java Virtual Machine.

KQL Kibana Query Language.

NIST The National Institute of Standards and Technology.

PaaS Platform-as-a-Service.

QoS Quality of Service.

RDS Relational Database Service.
REST Representational State Transfer.
ROI Return on Investment.

S3 Simple Storage Service.
SaaS Sotware-as-a-Service.
SDK Software Development Kit.

TCP Transmission Control Protocol.
TI Tecnologias de Informação.

UI User Interface.
UML Unified Modeling Language.

VPC Virtual Private Cloud.

Lista de Figuras

Anexos de código

1

Introdução

1.1 Contexto e Motivação

As inovações tecnológicas têm moldado a indústria das Tecnologias de Informação (TI), desde o surgimento da era tecnológica. Na última década, uma das inovações que tem revolucionado o setor das TI é o conceito de *cloud computing*, que tendo vindo a ser amplamente adotado tanto a nível académico como industrial.

O mercado do *cloud computing* possuiu um elevado crescimento na última década, esperando-se que atinja cerca de 240 mil milhões de dólares em 2020 [1]. A utilização do *cloud computing* [2] por parte de várias empresas deve-se essencialmente a fatores computacionais, como a elevada escalabilidade, a eficiência e a qualidade de serviço, e a fatores económicos, nomeadamente a redução de custos e o baixo gasto inicial de capital. Para além disso, a ideia de existência de uma infinidade de recursos ao dispor do utilizador e a possibilidade de aquisição de recursos e serviços a curto prazo – por exemplo, por hora – torna apelativa a adoção deste modelo.

De acordo com o estudo apresentado pelo International Data Group (IDG) em 2018 [3], a adoção da *cloud* continua a manter-se em expansão no mundo das TI. Esta pesquisa demonstrou o crescente investimento e envolvimento nesta área por parte das empresas por forma a progredirem tecnologicamente e impulsionarem os seus negócios. Dos diversos dados estatísticos recolhidos pela IDG, decidiu-se salientar aqueles que são relevantes e suportam a motivação da dissertação, nomeadamente o facto de:

- 73% das organizações possuírem, pelo menos, uma aplicação ou parte da sua infraestrutura computacional na *cloud*;

- 9 em cada 10 empresas migrarem as suas aplicações ou infraestruturas para *cloud* até 2019;
- 38% dos inquiridos manifestarem desejo de efetuar uma migração a 100% para a *cloud*.

Embora várias organizações tenham aderido à *cloud* para reduzir gastos com infraestruturas ou evitar custos iniciais para novos investimentos, ainda existe uma grande parte que não atingiu esse propósito.

O insucesso na concretização destes objetivos depende fortemente da maturidade com que a *cloud* é gerida e as práticas de gestão escolhidas. Por forma a evitar gastos é necessário haver um planeamento cuidadoso por parte dos utilizadores de *cloud* e é crucial analisar os recursos em termos monetários.

Segundo um relatório estatístico de 2018 apresentado pela `RightScale` [4], os utilizadores de *cloud* desperdiçam cerca de 35% do valor investido. Quando questionados acerca dos desafios que possuem para 2018, 77% dos inquiridos identificou a segurança como principal desafio, logo seguido da gestão/otimização de custos, representando uma preocupação para 76% dos inquiridos.

Com a gestão de custos no topo dos desafios, as organizações estão focadas em ganhar o controlo do investimento. Quando interrogados acerca das iniciativas para 2018, destacaram no topo das prioridades a otimização dos custos de utilização da *cloud* com 58% e com 44% a melhoria dos relatórios financeiros.

A complexidade dos custos associados à *cloud* tem vindo a aumentar devido a fatores como o surgimento de novos tipos de instâncias, a diversidade de métodos de aquisição de recursos e a multiplicidade de descontos oferecidos aos utilizadores [5].

Este aumento, em conjunto com a intensificação da utilização e o surgimento de novos recursos, leva a que os profissionais de Infraestrutura e Operações (I&O) dependam cada vez mais de ferramentas que permitam gerir custos operacionais [5, 6].

1.2 Objetivos

Tendo em mente o contexto e a motivação descritos, foram estabelecidos alguns objetivos por forma a orientar o trabalho a realizar. A concretização dos objetivos implicam a compilação e assimilação de uma grande quantidade de informação proveniente de fontes diversas e heterogéneas.

Para este projeto foi estabelecido um conjunto de objetivos a seguir ao longo do seu desenvolvimento, dos quais se destacam:

- Avaliar a adoção de soluções de análise de custos operacionais na *cloud* através da:
  - Recolha e análise de documentos bibliográficos acerca de *cloud computing*, fornecedores de *cloud* e otimização de custos em *cloud*;
  - Investigação das ferramentas de gestão de *cloud* existentes no mercado;

  - Análise das ferramentas avaliadas no estudo de mercado quanto à forma de obtenção de dados e funcionalidades fornecidas;
  - Determinação da viabilidade do sistema a desenvolver.

- Monitorizar custos operacionais de aplicações *web* em *cloud* a partir da:
  - Elaboração do *design* da solução para otimização de custos operacionais a partir das conclusões retiradas anteriormente e com vista na:
    * Obtenção de uma visão geral dos dados de utilização e custos derivados;
    * Descrição da informação com elevado grau de detalhe;
    * Associação dos dados a parâmetros como operações, serviços, contas, etc;
    * Previsão da evolução dos custos operacionais;
    * Identificação de recursos com gastos excessivos;
    * Deteção de recursos não utilizados ou com baixa utilização.
  - Implementação do sistema projetado.

- Garantir a qualidade do sistema desenvolvido recorrendo à:
  - Aplicação da solução em casos de estudo;
  - Avaliação dos resultados obtidos;
  - Comparação do sistema com as ferramentas analisadas.

1.3 Abordagem Metodológica

Uma metodologia de pesquisa pode ser definida como um meio de encontrar uma solução para um problema ou uma investigação que leve à compreensão de um fenómeno [7]. Assim sendo, podem ser tomadas várias direções para alcançar uma possível solução do problema, sendo que algumas metodologias podem ser mais eficazes do que outras e, portanto, é importante adotar aquela que permita obter os melhores resultados possíveis.

No caso das Tecnologias de Informação (TI), ramo onde está inserida a ciência da computação, existe uma metodologia de pesquisa amplamente usada denominada de Design Science Research (DSR) [8] que oferece grandes benefícios no que se refere à resolução de problemas da vida real das organizações. Dado o seu pragmatismo, esta metodologia é adequada à investigação de problemas de natureza prática, em detrimento da verificação de leis ou teorias [9].

Fundamentalmente, o método DSR reitera que o conhecimento, compreensão e solução de um problema passa fundamentalmente pela construção e aplicação de um artefacto, resultante da execução de um conjunto de etapas.

De forma resumida, esta metodologia é composta por duas atividades [7]: uma denominada de *building* – onde ocorre o processo criativo que leva ao surgimento de novos produtos – e outra denominada de *evaluating* – a avaliação da utilidade desses produtos.

As diferentes etapas que esta metodologia pressupõe, bem como os resultados que são expectáveis obter em cada fase podem ser consultados na Figura 1.

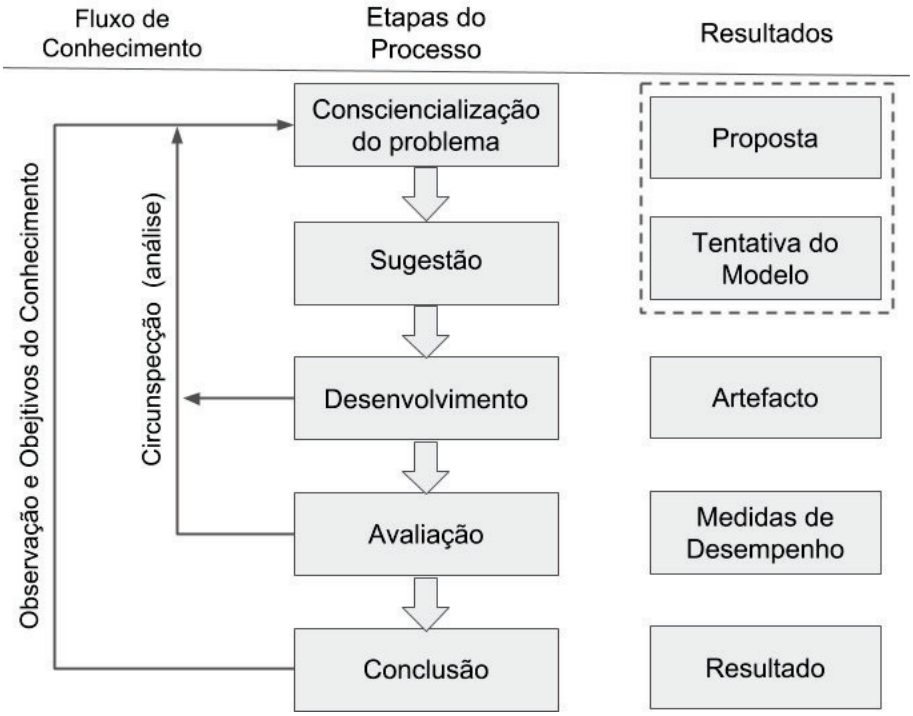


Figura 1: Metodologia geral do *Design Science Research (DSR)* (**Adaptado de:** *Vaishnavi et al. [10]*)

Em termos de processo, esta metodologia é constituída por 5 fases distintas descritas de seguida [10]:

1. Consciencialização do problema : o foco nesta etapa para quem está a efetuar o trabalho de pesquisa passa pela identificação e compreensão do problema em estudo e para o qual se pretende apresentar uma solução.

2. Sugestão : apresentação de possíveis soluções exequíveis para o problema em estudo – artefactos. Esta etapa é marcada pela necessidade de utilização de criatividade e conhecimentos/experiências de modo a propor soluções viáveis.

3. Desenvolvimento : consiste na utilização de um dos artefactos propostos na etapa anterior. Se os desenvolvimentos se mostrarem adequados e capazes, serão futuramente avaliados (quarta fase). Caso tal não se verifique, é necessário utilizar outro artefacto proposto ou, caso não exista, retornar à consciencialização do problema.

4. Avaliação : o artefacto construído deve ser avaliado tendo em conta um conjunto de critérios que, idealmente, se encontram descritos na sua proposta. Os resultados dos testes devem estar de acordo com as expetativas sobre o comportamento idealizado do produto. Caso não se verifique, deve-se tentar compreender o que não funcionou.

5. Conclusão : são apresentados os resultados obtidos e avaliado o sucesso no desenvolvimento do artefacto. Se o objetivo não tiver sido cumprido, pode-se retornar ao início do ciclo na tentativa de obter sucesso.

De modo a auxiliar o desenvolvimento da DSR, Hevner and Chatterjee estabeleceram os seguintes critérios a serem considerados por quem efetua a pesquisa [9]:

Diretrizes do DSR	Descrição
Pragmatismo	Este método procura aprimorar tanto a teoria como a prática. A teoria é avaliada pelo grau em que os seus princípios melhoram a prática.
Relevância do problema	O objetivo do método é desenvolver soluções baseadas em tecnologia para problemas importantes e relevantes.
Avaliação do *Design*	A utilidade, qualidade e eficácia do artefacto devem ser rigorosamente demonstradas por meio de métodos de avaliação bem executados.
Contribuições do *Design*	Este método deve promover contribuições claras e verificáveis nas áreas dos artefactos desenvolvidos e nas metodologias de design aplicadas.
Rigor da pesquisa	A pesquisa baseia-se na aplicação de métodos rigorosos na construção e na avaliação do *design* do artefacto.
Design como um Processo de Pesquisa	A procura por um artefacto deve ser conduzida tendo conhecimento de abordagens concorrentes e sendo aplicada como um processo cíclico até se verificar a eficácia da solução na resolução do problema.
Comunicação da Pesquisa	Os resultados da pesquisa deverão ser apresentados para o público orientado à tecnologia bem como para os orientados à gestão.

Tabela 1: Diretrizes do *Design Science Research*

1.4 Estrutura do documento

A tese de mestrado encontra-se organizada em 6 capítulos. Além deste primeiro capítulo introdutório que contextualiza o leitor do assunto em estudo, as suas motivações e os objetivos deste trabalho, são ainda apresentados os restantes capítulos:

- Capítulo 2 – Estado da arte: revisão da literatura e investigação dos principais conceitos teóricos e científicos associados à computação em nuvem, aos fornecedores de serviços de *cloud* e aos modelos e políticas de preços aplicados na *cloud*;
- Capítulo 3 – O problema e desafios da análise de custos de *Cloud*: caraterização do problema em estudo e identificação dos principais desafios na elaboração da solução;
- Capítulo 4: Sistema de otimização de custos de *Cloud* – exposição das decisões tomadas ao longo da implementação da proposta de solução;
- Capítulo 5: Casos de estudo – avaliação do sistema desenvolvido através da aplicação da solução em diversos casos de estudo reais;
- Capítulo 6: Conclusões e Trabalho Futuro – considerações finais do desenvolvimento da solução e sugestões para trabalho futuro.

2

Estado da arte

Este capítulo é relativo ao estudo do Estado da Arte e da Revisão da Literatura, sendo descritos conceitos teóricos e científicos fundamentais relacionados com o tema desta dissertação como *cloud computing*, fornecedores de *cloud* e processos de definição de preços na *cloud*.

2.1 Computação em Nuvem

Com o rápido desenvolvimento das tecnologias de processamento e armazenamento e o sucesso da Internet, os recursos de computação tornaram-se mais baratos, mais poderosos e mais ubíquos. Desta tendência tecnológica emergiu um novo modelo de computação denominado *cloud computing* [2], no qual recursos (como armazenamento ou processamento) são fornecidos aos utilizadores como serviços de acordo com a política de aquisição pretendida.

A ideia principal por trás do *cloud computing* não é recente. Nos anos 60, John McCarthy previu que as facilidades de computação seriam fornecidas ao público em geral como um serviço. O termo "*cloud*" começou a ganhar popularidade quando, em 2006, Eric Schmidt, na altura CEO da Google, usou esta palavra para descrever o modelo de negócio de fornecimento de serviços através da Internet. Desde então, este termo tem vindo a ser amplamente usado em diversos contextos para representar várias ideias [11].

Devido à falta de uma definição padrão para "cloud computing" e à utilização excessiva deste termo, surgiu a necessidade de o definir de forma a reduzir ceticismos e confusões. A principal razão para existirem diferentes perceções deste conceito deve-se ao facto de, ao contrário de outros termos, este não se tratar de uma nova tecnologia mas sim de um novo modelo operacional que utiliza a tecnologia existente para operar de forma diferente.

Embora não exista uma definição consensualmente aceite, considerou-se que a apresentada pelo The National Institute of Standards and Technology (NIST) é aquela que melhor descreve este conceito [2]:

Definição 1. *Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

2.1.1 Principais características

O elemento chave da computação em nuvem é a "ilusão da infinidade de recursos". De acordo com o modelo proposto pelo NIST, esta ilusão só é garantida graças as seguintes características essenciais:

- **Disponibilidade de recursos a pedido:** os utilizadores podem adquirir recursos computacionais, como tempo de servidor e armazenamento, conforme necessário e de forma automática.
- **Ubiquidade no acesso à rede:** a rede de recursos encontra-se disponível através da utilização de mecanismos padrão de acesso, aptos a serem consumidos pelos clientes.
- **Recursos partilhados:** os recursos são partilhados pelos diversos clientes de forma dinâmica, através da utilização de uma *pool* de recursos, ajustando-se dinamicamente à capacidade de serviço requerida pelos clientes.
- **Rápida elasticidade:** os utilizadores tem a capacidade de escalar recursos de forma a satisfazer as variações na procura de serviços.
- **Serviços mensuráveis:** os recursos são passíveis de serem monitorizados por forma a permitir a otimização de custos e a análise de desempenho.

2.1.2 Modelos de serviço de cloud

O modelo apresentado pelo NIST identifica 3 modelos de serviço a disponibilizar de acordo com o nível de abstração das capacidades fornecidas e com o modelo de serviço dos fornecedores: *Infrastructure-as-a-Service (IaaS)*, *Platform-as-a-Service (PaaS)* e *Sotware-as-a-Service (SaaS)*. O modelo de negócio subjacente ao *cloud computing* é orientado a serviços, isto é, os recursos computacionais são fornecidos como serviços conforme solicitados.

Estes modelos podem ser vistos como uma arquitetura em camadas, onde os serviços de uma camada mais elevada incluem os serviços das camadas inferiores. Cada modelo de serviço disponibiliza uma funcionalidade única de acordo com o utilizador e com o controlo sobre o ambiente de trabalho. Este

controlo diminui à medida que descemos na infraestrutura, sendo menor no modelo SaaS e maior no modelo IaaS [12], de acordo com o representado na Figura 2 .

De seguida, apresenta-se uma breve descrição para cada modelo de serviço:

- **Infrastructure-as-a-Service (IaaS):** Disponibiliza ao utilizador recursos de infraestrutura e serviços como capacidade de processamento, armazenamento, redes e outros recursos computacionais, de modo a possibilitar a execução de aplicações e sistemas operativos. Os utilizadores não gerem a infraestrutura subjacente mas podem controlar o sistema operativo, áreas de armazenamento, aplicações e configurar componentes de rede como *routers*, *firewalls*, entre outros.

- **Platform-as-a-Service (PaaS):** Este modelo pode ser descrito como um ambiente de trabalho para programadores de aplicações em *cloud*, instalado na infraestrutura do fornecedor, tornando desnecessária qualquer instalação de desenvolvimento local. Os serviços disponibilizados permitem executar numa infraestrutura em *cloud* aplicações adquiridas ou desenvolvidas com recurso a linguagens de programação, bibliotecas e serviços suportados pelo fornecedor de *cloud*. O utilizador não gere a infraestrutura mas controla as aplicações e configurações do sistema.

- **Sotware-as-a-Service (SaaS):**

  Trata-se de um modelo que disponibiliza serviços de alto nível aos clientes, fornecendo *software* pronto a usar como *Enterprise Resource Planning (ERP)*. Os serviços facultados utilizam aplicações do fornecedor, são executados numa infraestrutura em *cloud* e acedidas via *browser* (*thin client*) ou *interface* de um programa (*thick client*) [13]. O utilizador não gere a infraestrutura nem pode controlar as aplicações que lhe são facultadas pelo fornecedor de serviços.

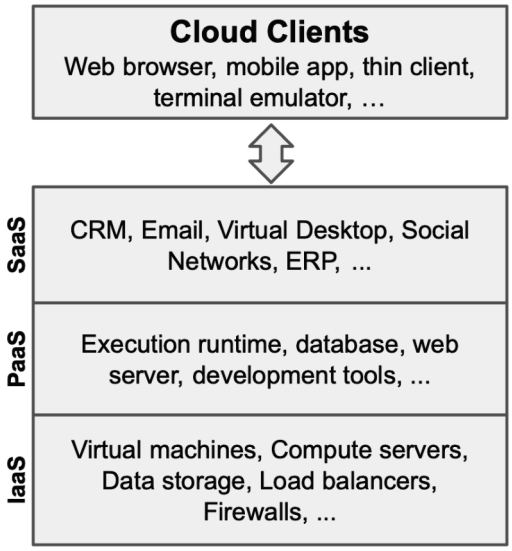


Figura 2: Modelo de negócio do *cloud computing* (**Adaptado de:** *Rodmunkong et al. [14]*)

2.1.3 Modelos de implementação de cloud

Quando se pretende migrar uma aplicação para o ambiente de *cloud* existe um conjunto de fatores que é necessário ter em conta. A título de exemplo, alguns fornecedores estão mais interessados na redução dos custos operacionais dos seus clientes, enquanto outros dão maior destaque à fiabilidade. Deste modo, é importante considerar os diferentes modelos de implementação propostos pelo NIST e quais os seus benefícios e desvantagens:

- **Cloud pública:** modelo em que os recursos são fornecidos como serviços para o público em geral. A *cloud* pública oferece um conjunto de benefícios importantes para os fornecedores de serviço dos quais se destaca a inexistência de investimento de capital inicial em infraestruturas e a transferência de riscos para os fornecedores da infraestrutura. Contudo, neste modelo não existe o controlo sobre configurações de dados, rede e segurança, revelando-se ineficiente para alguns modelos de negócio.

- **Cloud privada:** modelo em que a infraestrutura de *cloud* é disponibilizada para uma única organização, podendo ser criada e gerida pela organização ou por fornecedores externos. A *cloud* privada oferece um elevado grau de controlo sobre o desempenho e segurança, contudo possui um investimento de capital inicial associado.

- **Cloud comunitária:** este modelo disponibiliza a infraestrutura de *cloud* a um conjunto de organizações que partilham os mesmos interesses, como requisitos de segurança. Esta pode ser detida, gerida e operada por várias organizações da comunidade ou por uma entidade externa.

- **Cloud híbrida:** modelo que usa uma infraestrutura de *cloud* proveniente da junção de duas ou mais infraestruturas de *cloud* diferentes (públicas, privadas e/ou comunitárias), por forma a colmatar as limitações de cada abordagem. As *clouds* híbridas oferecem uma maior flexibilidade uma vez que fornecem mecanismos de controlo e segurança mais rigorosos e ainda a facilidade na gestão de serviços *on-demand*. Como desvantagem salienta-se a dificuldade na projeção deste modelo, nomeadamente a divisão dos componentes pelos diferentes modelos de *cloud* que a constituem.

2.2 Fornecedores de Serviços de Cloud

O *cloud computing* é um estilo de computação no qual os recursos são fornecidos como um serviço através da Internet. Atualmente a necessidade por parte das empresas de uma maior variedade e quantidade de serviços provenientes dos diferentes modelos (IaaS, PaaS, SaaS) tem vindo a aumentar, tornando o mercado da *cloud* um dos mais lucrativos do mundo [15].

Tendo em conta o tema desta dissertação considerou-se essencial identificar e estudar os principais Cloud Service Providers (CSPs), bem como os serviços e recursos que oferecem. Um CSP é uma empresa que oferece serviços de rede, infraestrutura ou aplicações na *cloud*. Os serviços na *cloud* encontram-se hospedados num *datacenter*, podendo ser acedido através de conectividade à rede. A utilização de um CSP traz grandes mais valias a nível de eficiência e em termos económicos, uma vez que quem o utiliza evita a construção de uma infraestrutura própria para suportar serviços e aplicações, tirando benefício dos serviços e da infraestrutura fornecida pelo CSP.

Por forma a identificar os CSPs com maior influência no mercado ao longo dos anos foram analisados os principais competidores dos últimos quatro anos (2016, 2017, 2018 e 2019). Para obter esta informação recorreu-se aos estudos realizados pela *Gartner* [16, 17, 18, 19] nos anos referidos onde é avaliado o modelo de IaaS de um conjunto de CSPs.

Por forma a facilitar e agilizar a compreensão dos resultados, apresenta-se a informação sob a forma de um quadrante mágico. Um quadrante mágico da *Gartner* é o ponto culminante da pesquisa num mercado específico, onde é oferecida uma visão ampla das posições dos concorrentes do mercado após ter sido aplicado um tratamento gráfico e diversos critérios de avaliação, dividindo-os em 4 categorias [20]:

- *Leaders* : Com elevada capacidade de visão e execução, os líderes tendem a ser empresas de grande dimensão no mercado com bastantes clientes. Os líderes têm grande influência sobre mercados específicos ditando inclusive a evolução dos mesmos.
- *Challengers* : Embora possuam capacidade de execução apresentam limitações relativamente à visão de mercado. Normalmente são empresas com posição no mercado que não pretendem sair da sua zona de conforto e desenvolver novos serviços.
- *Niche Players* : Trata-se de empresas com menos capacidade de visão ou recentes na área de negócio, focadas num número restrito de funcionalidades ou numa área específica.
- *Visionaries* : Empresas de menor dimensão com consciência da evolução do mercado e com potencial para a inovação, cujo principal objetivo é alcançar os líderes.

Conforme é possível observar nas Figuras 3, 4 e 5 e 6, muito pouco se alterou ao longo dos anos relativamente à liderança do mercado, tendo a Amazon Web Services (AWS) assumido o pódio, seguida pelo *Microsoft Azure*. Da análise destes resultados verifica-se ainda a entrada no ano de 2018 do Google Cloud Platform (GCP) para o segmento de líderes.

No quadrante mágico de 2018 constata-se a retirada de oito CSPs derivado da inclusão de critérios mais rigorosos, dos quais se destaca a necessidade dos fornecedores possuírem ofertas de IaaS e PaaS para serem incluídos no estudo. Em 2019 observa-se que as alterações foram insignificantes, mantendo-se os CSPs nas categorias que estavam no ano anterior.

Por forma a compreender que serviços e recursos disponibilizados pelos principais líderes de mercado e quais as principais diferenças entre eles, procedeu-se a uma análise mais criteriosa dos mesmos [21]. Deste modo as Secções 2.2.1, 2.2.2 e 2.2.3 descrevem os principais resultados obtidos da análise efetuada à AWS, Microsoft Azure e GCP, respetivamente.

Figura 3: *Magic Quadrant for Cloud Infrastructure as a Service, Worldwide, 2016* (**Adotado de:** *Leong et al. [16]*)

Figura 4: *Magic Quadrant for Cloud Infrastructure as a Service, Worldwide, 2017* (**Adotado de:** *Leong et al. [17]*)

Figura 5: *Magic Quadrant for Cloud Infrastructure as a Service, Worldwide, 2018* (**Adotado de:** *Smith et al. [18]*)

Figura 6: *Magic Quadrant for Cloud Infrastructure as a Service, Worldwide, 2019* (**Adotado de:** *Wright et al. [19]*)

2.2.1 Amazon Web Services (AWS)

A plataforma de *cloud* da *Amazon* oferece quase todos os recursos ligados à industria do *cloud computing*, permitindo criar soluções de negócio através de serviços *web* integrados. A AWS disponibliza uma ampla gama de serviços como poder computacional, armazenamento de dados, gestão de bases de dados, *networking*, *analytics*, balanceamento de carga e escalonamento automático. Para além destes, são ainda fornecidos serviços que permitem aumentar a produtividade e eficiência nomeadamente ferramentas de desenvolvimento, ferramentas de gestão, serviços de proteção de identidade e segurança.

Até ao momento, este CSP fornece mais de 100 serviços divididos por cerca de 20 categorias. Dos inúmeros serviços fornecidos decidiu-se salientar alguns dos que constituem o *core* da AWS [22]:

Figura 7: Serviços da *Amazon Web Services (AWS)*

- *Amazon Elastic Compute Cloud (EC2)* : serviço que permite adquirir recursos computacionais de *cloud* como processamento e serviços *web*. Este serviço possui um ambiente de computação virtual que permite aos clientes iniciar instâncias com vários sistemas operativos, gerir as permissões de acesso à rede, entre outras funcionalidades, através das Application Programming Interfaces (APIs) disponibilizadas.

- *Amazon Relational Database Service (RDS)*: serviço que permite a criação, execução e dimensionamento de bases de dados relacionais na *cloud*. Além disso, oferece capacidade de redimensionamento para vários Database Management Systems (DBMSs) padrão da indústria.

- *AWS Direct Connect* : fornece uma rede dedicada para o estabelecimento de conexões entre os serviços da AWS, permitindo a redução de custos e o aumento do *throughput*. A gestão das conexões e interfaces de rede é feita de forma simples e eficaz, possibilitando escalar conexões de acordo com as necessidades do cliente.

- *Amazon Elastic Block Store (EBS)* : sistema de armazenamento em bloco na *cloud*. Trata-se de um serviço que garante elevada disponiblidade no fornecimento de volumes de armazenamento, ao nível dos blocos, para utilização com instâncias EC2 e RDS.

- *Amazon Simple Storage Service (S3)* : trata-se de um exemplo de IaaS para armazenamento de dados altamente escalável, seguro e de baixa latência na *cloud*. Os dados são armazenados sob a forma de objetos em recursos denominados de "*buckets*".

- *Amazon Elastic Load Balancing (ELB)* : distribui automaticamente o tráfego das aplicações por diversos destinos, como por exemplo *containers*, endereços IP e instâncias EC2. O serviço pode lidar com carga variável, oferecendo três tipos de *load balancers* todos eles com alta disponibilidade, segurança e escalabilidade automática.

- *Amazon Route 53*: consiste num serviço *web* de Domain Name System (DNS) na *cloud* com elevada disponibilidade e escalabilidade. Este serviço conecta as solicitações dos utilizadores à infraestrutura em execução na AWS, podendo ainda ser usado em reencaminhamentos para o exterior da infraestrutura.

- *Amazon Virtual Private Cloud (VPC)* : permite o desenvolvimento de uma *cloud* privada com políticas próprias, permissões, endereços IP, sub-redes, configuração de rotas e gestão de recursos. Trata-se de uma *cloud* privada dentro da infraestrutura da AWS, isolada dos serviços da *cloud* pública hospedados na mesma infraestrutura.

- *Elastic IP* : consiste num endereço de IPv4 estático concebido para ser utilizado em ambientes de *cloud* dinâmicos e encontra-se associado a uma conta da AWS. Com a utilização deste IP é possível mascarar falhas de instâncias através do mapeamento para outra instância que o cliente possua.

Em termos de maturidade, a AWS trata-se do líder há mais tempo no mercado. Este CSP destaca-se dos restantes por razões como segurança, relação custo-eficácia, flexibilidade, elasticidade e escalabilidade.

Como limitações à sua utilização destaca-se a acentuada curva de aprendizagem, a dificuldade de utilização (devido ao elevado número de serviços fornecidos) e a complexa gestão de custos [23].

2.2.2 Microsoft Azure

À semelhança da AWS, o *Microsoft Azure* também fornece uma grande variedade de serviços em diversas áreas por forma a satisfazer as necessidades dos seus clientes como [24]:

- Infraestrutura : fornece serviços computacionais como *Azure Virtual Machines*, serviços aplicacionais, nomeadamente *Web* e *Mobile*, mecanismos de computação paralela em larga escala, entre outros;

- *Storage* : possui soluções de armazenamento na *cloud* para gerir e processar dados, sejam estes regulares ou em grande escala. Destes serviços destaca-se ainda o fornecimento de bases de dados SQL, bases de dados NoSQL (nomeadamente *DocumentDB*) e mecanismos de *caching* através da utilização de *Redis*;

- Aplicação: inclui serviços que auxiliam na construção e operação de aplicações designadamente o *Azure Active Directory (Azure AD)*, o *Service Bus* (conexão de sistemas distribuídos), o *HDInsight* (processamento de *big data* com recurso ao *Apache Hadoop*), o *Azure Scheduler* e o *Azure Media Services*;

- *Network* : possui um conjunto de serviços de rede como o *Virtual Networks*, o *ExpressRoute*, o *Azure DNS*, o *Azure Traffic Manager* e o *Azure Content Delivery Network*.

Uma característica a destacar é a capacidade de execução paralela de software em larga escala. Trata-se de um recurso exclusivo deste CSP e da AWS e o que os distingue do GCP.

A escolha do *Azure* em detrimento dos restantes deve-se à rapidez no desenvolvimento, escalabilidade, integração com ferramentas *Microsoft* e existência de suporte *open source*. Como desvantagens destacam-se problemas com o suporte técnico, a documentação disponibilizada e o facto de ser pouco preparado para o contexto empresarial [23].

2.2.3 Google Cloud Platform (GCP)

O GCP fornece um conjunto de serviços de *cloud* análogos aos dois anteriores, dos quais se destaca [25]:

- Mecanismos de computação : fornece IaaS aos seus clientes permitindo a orquestração em grande escala nos servidores virtuais hospedados na infraestrutura da *Google*;

- Desenvolvimento de aplicações : disponibiliza uma PaaS na qual os clientes podem desenvolver aplicações usando uma plataforma integrada de alto desempenho;

- Armazenamento na *cloud*: serviço onde os clientes podem armazenar ficheiros de qualquer tipo e dimensão estando garantida a segurança dos mesmos;

- *Cloud SQL* : fornece bases de dados relacionais com suporte a diversos DBMSs, nomeadamente `PostgreSQL`, `MySQL` e `SQL Server`;

- *Big Query* : trata-se de um *data warehouse* na *cloud* altamente escalável, rápido e económico, concebido para análise de dados com *machine learning*.

Este CSP destaca-se ainda pelo suporte *open source*, elevada portabilidade de sistemas e por ter sido desenhado para negócios baseados em *cloud*. Como desvantagem salienta-se o facto de fornecer um conjunto de serviços e funcionalidades inferior aos restantes, possuir poucos *data centers* espalhados pelo mundo e por se tratar do fornecedor há menos tempo no mercado [23].

2.3 Faturação em ambiente de cloud

Os CSPs fornecem soluções em *cloud* aos seus clientes e utilizadores finais baseados num conjunto de políticas económicas. O processo de determinar o que um fornecedor de serviços receberá de um utilizador final em troca dos seus serviços é denominado de *pricing*. Weinhardt et al. [26] afirmam que o sucesso do *cloud computing* no mercado das TI só pode ser obtido pelo desenvolvimento de técnicas adequadas de *pricing*.

2.3.1 Modelos de definição de preços

Os modelos de definição de preços variam entre os fornecedores de *cloud*. Para os CSPs, os principais objetivos são a rentabilidade e a maximização de lucros, enquanto os utilizadores dão maior relevância à Quality of Service (QoS), elevada disponibilidade dos recursos e relação custo-eficácia.

Uma das condições chave para o sucesso no fornecimento de serviços de *cloud* é a clareza e transparência na definição de preços, tanto para clientes como para CSPs [26]. Uma estratégia de preços bem definida, quando aplicada corretamente, pode alterar o comportamento dos clientes e determinar a posição da oferta no mercado. Graças ao rápido desenvolvimento tecnológico e a crescente competição entre CSPs, a modelação dos preços tem vindo a tornar-se cada vez mais complexa.

Em 2012, Iveroth et al. [27] descreveram uma taxonomia abrangente de modelos de *pricing* assente em cinco dimensões. Segundo os autores, estes modelos podem ser descritos através da caracterização das suas posições em cada dimensão. A taxonomia é chamada de modelo SBIFT, que significa acrónimos das dimensões, e pode ser consultada na Figura 8. As dimensões do modelo apresentado são as seguintes:

- *Scope* : dimensão que refere a granularidade da oferta. Num extremo encontra-se a definição de preços para produtos/serviços de um *Package* e no outro extremo a definição individual de preços – *Attribute*.
- *Base* : consiste na informação base determinante para as tomadas de decisão de políticas de preço. De entre vários modelos, salienta-se nesta dimensão o *Cost-based pricing* e o *Value-based pricing*.

- *Influence* : reflete a capacidade dos compradores e vendedores influenciarem o preço. Associados a esta dimensão encontram-se os modelos *Pricelist* e *Pay-what-you-want*.
- *Formula* : determina a conexão entre preço e volume. Nesta dimensão é frequente serem utilizados modelos como o *Fixed fee + per unit price* e o *Per unit price*.
- *Temporal rights*: estabelecimento do período de tempo durante o qual o utilizador pode usufruir da oferta. Os modelos *Subscription* e *Pay per use* tratam-se dos mais utilizados desta dimensão.

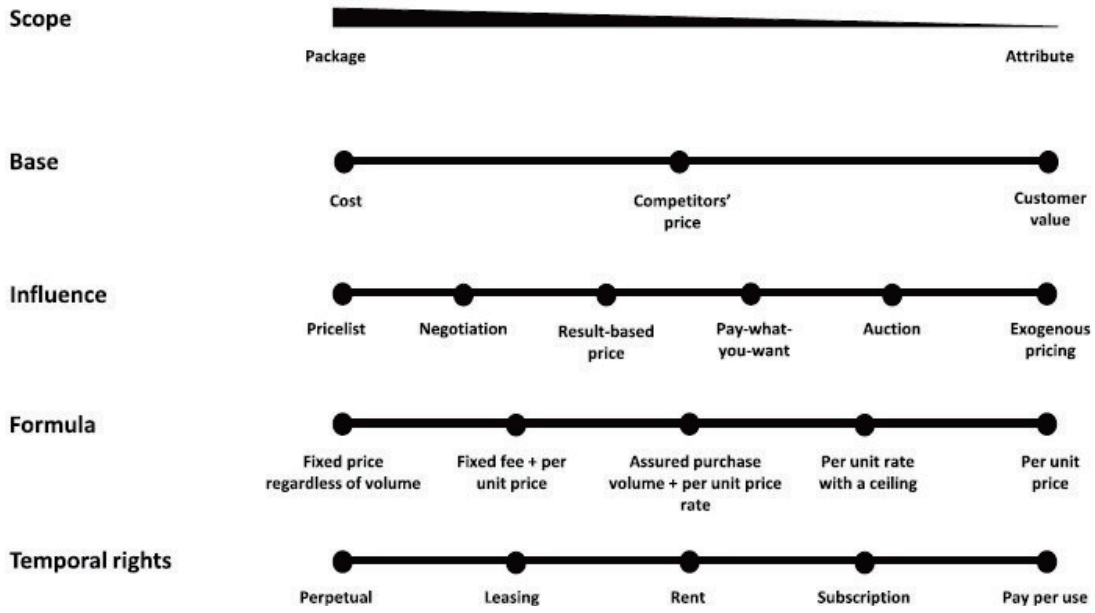


Figura 8: Modelo *SBIFT* (**Adotado de:** *Iveroth et al. [27]*)

Os modelos de *pricing* utilizados pelos CSPs assentam fortemente no modelo SBIFT e nas políticas de preço que este estabelece. Em termos gerais, o processo de definição dos preços de serviços de *cloud* pode-se enquadrar num dos seguintes: fixo, dinâmico ou dependente das condições do mercado [28].

Mecanismos de fixação de preços fixos incluem os modelos *pay-per-use* e *pay-as-you-go*, em que os clientes pagam um valor pelo que consomem de um produto ou pelo tempo que utilizam um determinado serviço. A subscrição é outro tipo de preço fixo, em que o cliente paga uma quantia fixa para usar o serviço por um determinado período de tempo, tipicamente entre 1 a 3 anos, ou até um dado valor que considere conveniente. Outra forma de fixação de preços é baseada em catálogo ou lista. Por outro lado, preços diferenciados ou dinâmicos implicam que o preço mude dinamicamente de acordo com os recursos do serviço, quantidade de volumes adquiridos ou características e preferências do cliente. Por último, os preços dependentes do mercado dependem das condições do mesmo em tempo real, nomeadamente de processos de negociação, leilões e da procura por serviços [29].

A definição de preços é influenciada por um conjunto de fatores, dos quais se salientam os custos iniciais que os CSPs possuem anualmente na aquisição de recursos, o período durante o qual é requerido serviços, a QoS prestada através do fornecimento de tecnologias que visam melhorar a experiência de utilização, a modernidade dos recursos fornecidos e, por fim, o valor gasto anualmente pelos fornecedores em manutenção e segurança [30].

A competição de mercado entre os fornecedores de serviços permite a regulação de preços, na medida em que os clientes procurarão aquele que oferece a melhor QoS ao menor preço. Um dos grandes focos dos CSP é adotar novas tecnologias e técnicas que lhes permita reduzir custos e, consequentemente, os preços praticados [31]. A Figura 9 reflete os diferentes aspetos do *cloud computing* percecionados de acordo com as políticas de *pricing* usualmente adotadas, QoS prestada e período de utilização.

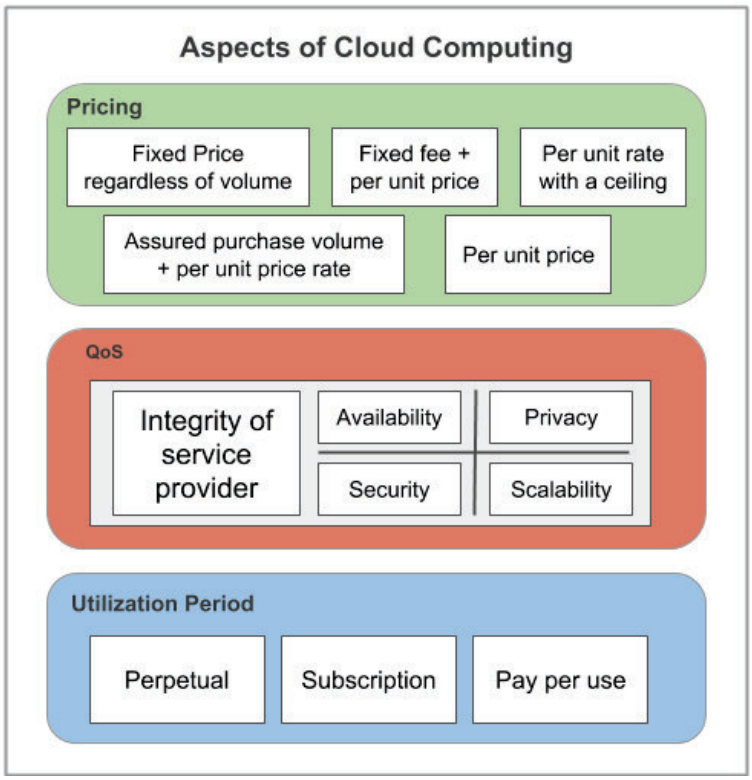


Figura 9: Aspetos da computação em nuvem (**Adaptado de:** *Al-Roomi et al. [29]*)

Embora cada fornecedor de *cloud* possua a sua estratégia de preços para cada serviço, em termos de recursos computacionais a oferta assenta essencialmente em três categorias: *on-demand*, *reserved* e *spot* [32].

No caso de instâncias *on-demand*, o pagamento é feito quando as instâncias são utilizadas, não existindo um compromisso a longo prazo ou pagamentos adiantados. As instâncias reservadas tratam-se de recursos computacionais adquiridos antecipadamente e por um período de tempo, tipicamente de 1 a 3 anos. Esta última categoria exige um maior compromisso e revela-se mais vantajoso quando utilizado a longo prazo.

As instâncias *spot* permitem adquirir capacidade computacional a um custo bastante mais reduzido quando comparado com as instâncias *on-demand*. De realçar que neste tipo de instâncias não existe compromisso por parte do fornecedor de *cloud*, na medida em que a instância pode não ser iniciada imediatamente e terminada quando o preço da instância atinge um valor superior aquele a que foi adquirido.

2.3.2 Políticas de preços dos fornecedores de serviços de cloud

Apesar dos diversos CSPs possuírem políticas de preços semelhantes, existem variantes nos serviços prestados aos utilizadores. De seguida, serão apresentados aspetos relevantes dos esquemas de preços dos fornecedores de serviços de *cloud* apresentados na Secção 2.2 [23]:

- Amazon Web Services (AWS)

  Os modelos de *pricing* da Amazon incluem o *Pay-As-You-Go* onde o utilizador paga pelo que utiliza, o *Pay less on Reserve* que permite poupanças na ordem dos 60% através do uso de instâncias reservadas e ainda o *Pay even less per unit for 100% usage* em que os valores de *storage* e transferência de dados são tributados em patamares.

  Este fornecedor de serviços possui três características fundamentais pelas quais os seus utilizadores pagam e que representam grande impacto no custo: capacidade computacional, *storage* e transferência de dados.

  No custo da capacidade computacional são tidos em conta fatores como o número de instâncias existentes, as características das respetivas máquinas, o modo de aquisição das instâncias e o período de tempo em funcionamento.

  Em termos de *storage* é considerado o tipo de armazenamento, a capacidade de armazenamento e ainda a quantidade da dados transferidos. Tipicamente, estes valores são tributados mensalmente e têm como unidade de medida o Gigabyte (GB).

- Microsoft Azure

  O fornecedor de serviços da Microsoft é o principal concorrente da AWS e, como tal, tenta oferecer serviços semelhantes a um preço mais competitivo.

  Quanto à capacidade computacional, este CSP opera segundo a política *Pay as you go*, possuindo ainda instâncias reservadas. Quando comparado com instâncias EC2 da Amazon, as máquinas virtuais fornecidas podem chegar a ser 72% mais económicas, de acordo com determinados fatores como a região onde se encontra, tipo de instância e política de preço com que foi adquirida [33].

Em termos de capacidade, as instâncias de bases de dados SQL, nomeadamente as SQL Server, podem gerar poupanças de cerca de 85% em comparação com instâncias Amazon RDS, quer seja requerido como um PaaS ou um IaaS [33].

- Google Cloud Platform (GCP)

  Sendo o mais recente dos CSP no mercado, este fornecedor tenta afirmar-se oferecendo preços significativamente mais reduzidos que os restantes. Algumas políticas que permitem alcançar esta vantagem são a cobrança por segundo e a aplicação automática de descontos, sendo que quanto maior a utilização, maior será o desconto.

  À semelhança dos outros fornecedores de serviço, os custos operacionais da *cloud* são maioritariamente provenientes de recursos computacionais e *storage*.

  Em termos de capacidade computacional, a Google fornece dois tipos de máquinas: predefinidas ou customizáveis. Instâncias predefinidas possuem características e preços tabelados enquanto as personalizáveis dependem das configurações que o utilizador desejar, afetando o preço a pagar pela instância. Parâmetros como a região onde a instância se encontra, números de *cores*, sistema operativo, horas de utilização, entre outros, contribuem para o valor a pagar.

  Relativamente aos custos de *storage*, estes dependem do tipo de armazenamento escolhido, podendo ser um dos seguintes: *Standard*, Durable Reduced Availability (DRA) e *Nearline*. Estes diferem na capacidade de armazenamento fornecido bem como operações de I/O, sendo taxados tendo por base o GB como unidade.

2.4 Soluções de monitorização e otimização de custos

Com a complexidade da definição de preços continuamente a aumentar, a oferta de soluções de monitorização e otimização de custos de *cloud* tem vindo a evoluir. Entre outros motivos, o baixo custo inicial, a rapidez no Return on Investment (ROI) e a capacidade de monitorização e otimização dos custos, contribuem para o surgimento de novas ferramentas e o melhoramento das existentes [6].

Para grande parte das organizações, a gestão dos custos de *cloud* ainda é vista como uma atividade nova e desconhecida. Tratando-se de uma nova responsabilidade, é frequente adicionar esta dimensão às funções existentes nos departamentos financeiros bem como nos departamentos de planeamento e gestão de infraestruturas. Regra geral estas ferramentas tendem a ser desenvolvidas em torno de dois *stakeholders* centrais: gestores financeiros – focados exclusivamente na atribuição de custos, faturação, alocação e gestão de verbas – e gestores de *cloud* – responsáveis por gerir recursos e garantir a melhor *performance* sem comprometer o orçamento associado.

O objetivo central no desenvolvimento destas ferramentas consiste no aumento da visibilidade dos custos operacionais de *cloud*, fornecendo informações que permitem organizar, contextualizar e integrar vários sistemas. Entre outras funcionalidades estas soluções permitem visualizar e rastrear custos, gerar e exportar relatórios, analisar opções de poupança, definir recursos e grande parte delas ainda contém mecanismos de análise e automação.

Tendo em conta esta problemática, os CSPs tem vindo a investir em soluções que permitam aos seus utilizadores obter visibilidade sobre os custos de operação e utilização de recursos. De seguida, apresentam-se as ferramentas fornecidas para cada fornecedor de *cloud* referido na Secção 2.2.

2.4.1 AWS Cost Explorer

A AWS disponibiliza um *Cost Explorer* com uma interface fácil de usar que permite ver, compreender e gerir custos de utilização ao longo do tempo. De entre as funcionalidades que fornece salienta-se a possibilidade de criar relatórios personalizados (como tabelas e gráficos) com diferentes níveis de detalhe, a análise pormenorizada dos dados de custos e utilização para identificar tendências de gastos, a deteção de anomalias, entre outras [34].

Esta solução fornece um conjunto de relatórios padrão que podem ser usados como ponto de partida para a análise de custos, sendo possível filtrar e agregar os dados. Na Figura 10 é possível consultar um relatório que se encontra disponível na secção de "Saved Reports" do *Cost Explorer* e que visa analisar custos mensais por serviço, nos últimos 6 meses. Neste relatório, os dados encontram-se filtrados apenas para as instâncias EC2 e agrupados por tipo de instância, sendo apresentado o *top* 5 dos resultados.

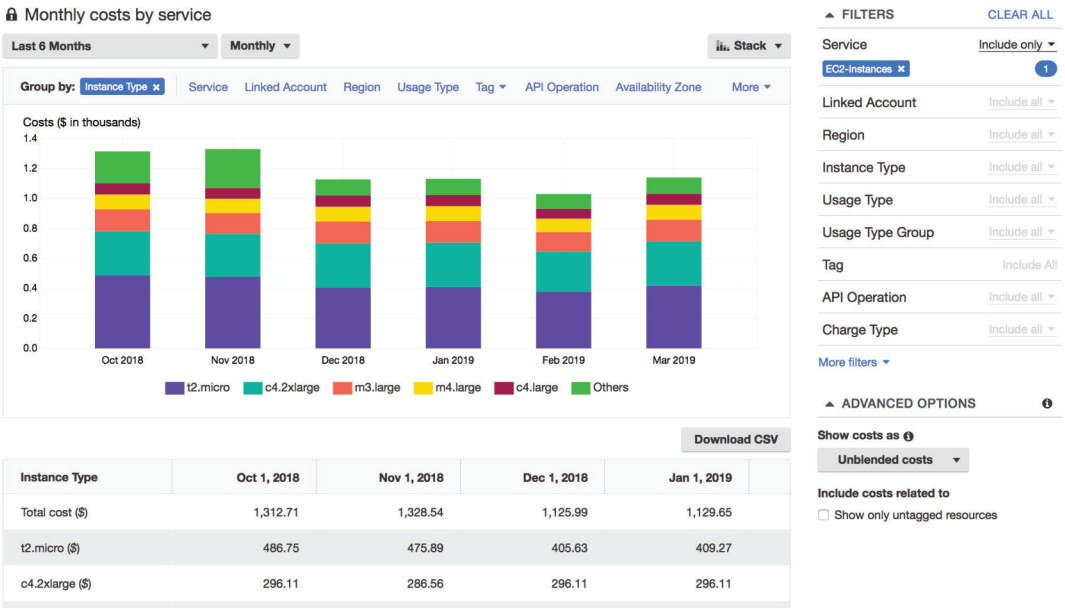


Figura 10: Custos mensais por serviço

Para além deste relatório, são disponibilizados outros que permitem consultar custos e utilização em número de horas das instâncias EC2 (Figura 11), analisar a evolução de custos por conta de utilização (Figura 12), prever custos baseados nos consumos dos últimos meses e ainda relatórios de cobertura e uso de instâncias reservadas.

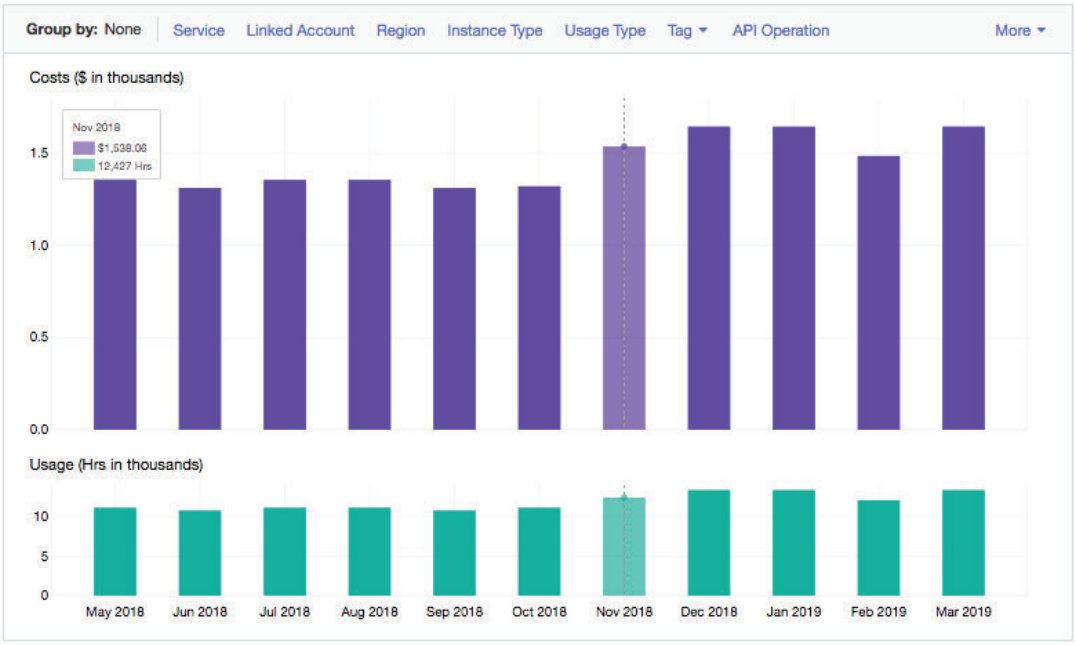


Figura 11: Custo e horas de utilização mensal de instâncias EC2

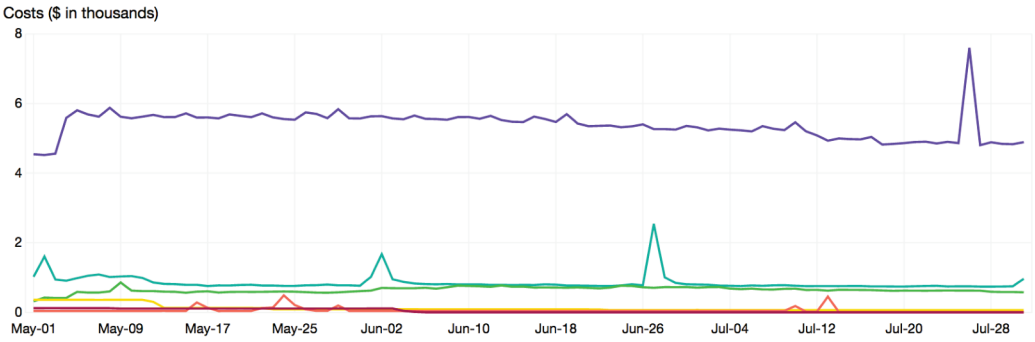


Figura 12: Custo mensal por conta de utilização

Embora esta solução auxilie na análise de custos verifica-se que a exploração dos custos de operação e dados de utilização dos recursos é bastante restritiva e não satisfaz as necessidades dos utilizadores de *cloud*. A incapacidade de consultar a informação para uma dada instância ou conjunto de instâncias bem como para um determinado componente aplicacional constituí algumas das lacunas desta ferramenta.

2.4.2 Azure Cost Management

O *Azure* disponibiliza uma ferramenta para monitorizar a utilização de recursos e gerir custos através de uma vista integrada no portal do CSP. De entre as funcionalidades fornecidas destaca-se a análise da informação em formato gráfico com a possibilidade de filtrar e agrupar dados, a implementação de políticas de gestão de custos com definição de orçamentos e alocação de custos, a definição de alertas, entre outras [35].

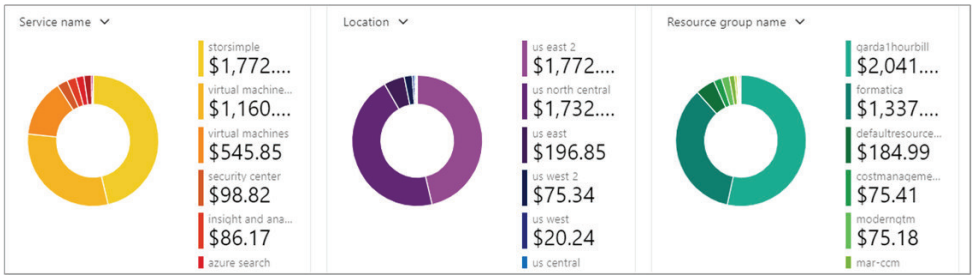


Figura 13: Agrupamento de custos por serviço, localização e recursos

À semelhança da solução fornecida pela AWS, nesta também é possível analisar custos por serviço, localização ou grupo de recursos (Figura 13). A informação pode ser consultada no formato gráfico mas também tabular, sendo exportável no formato Comma-Separated Values (CSV) (Figura 14). A previsão de custos e controlo de orçamentos através de alertas é outra mais valia desta ferramenta (Figura 15).

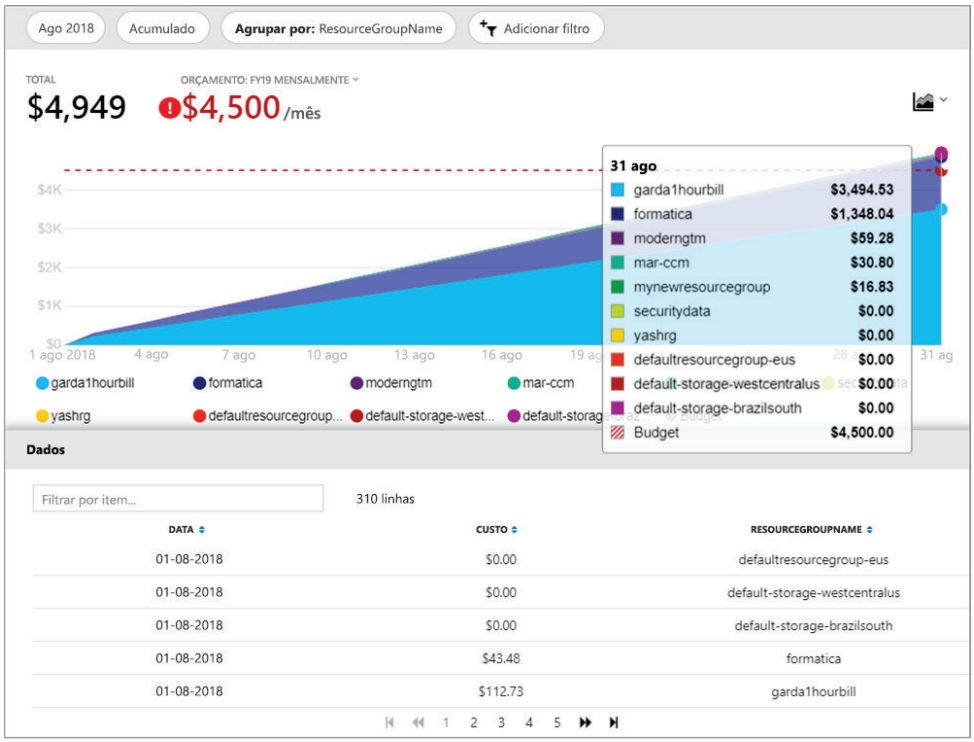


Figura 14: Custos por grupo de recursos

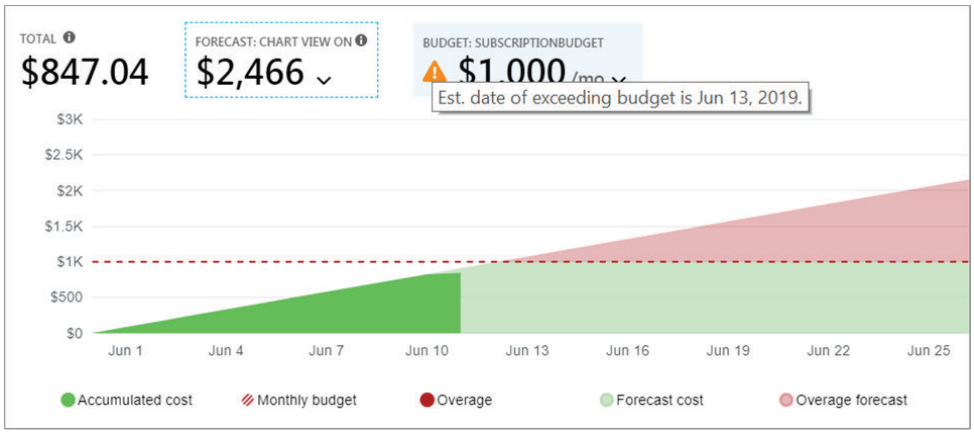


Figura 15: Previsão de custos e controlo de orçamento

Embora esta ferramenta permita a pesquisa por grupo de recursos, verificou-se bastante restritiva na medida em que não permite a criação de novos gráficos ou pesquisar informação para além da que está definida por omissão. Constatou-se ainda que a monitorização dos recursos do ponto de vista da sua utilização não constitui uma funcionalidade desta ferramenta.

2.4.3 Google Cloud Cost Management

O Google Cloud Platform (GCP) disponibiliza um conjunto de ferramentas para auxiliar na gestão dos custos de *cloud*, nomeadamente a existência de *Built-in reports*, *dashboards* customizáveis e *breakdowns* de custos (Figura 16), a criação de *budgets* e alertas, o controlo de permissões para gestão dos recursos, a definição de mecanismos de automatização, a otimização de custos através de sistemas de recomendação, entre outras [36].

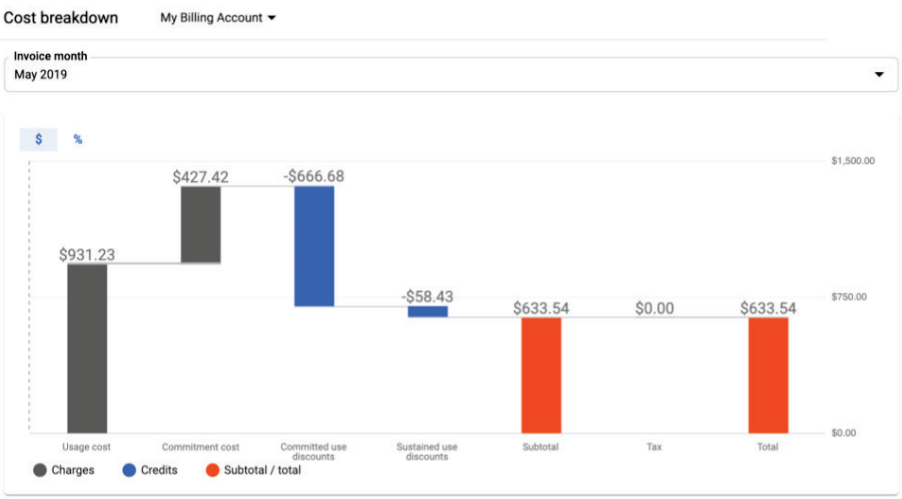


Figura 16: Discriminação de custos

Um exemplo de relatório fornecido pelo CSP pode ser consultado na Figura 17. Estes relatórios podem ser analisados no formato gráfico (linhas/barras) ou tabular. Esta informação pode ser visualizada para um determinado período temporal e filtrada/agregada por um conjunto de parâmetros. Para além disso, a vista sobre os dados pode ser realizada com diferentes granularidades, nomeadamente por dia, semana ou mês. No formato gráfico é ainda possível compreender como deverão evoluir os gastos num futuro próximo, sendo também a previsão total dos custos apresentada ao utilizador.

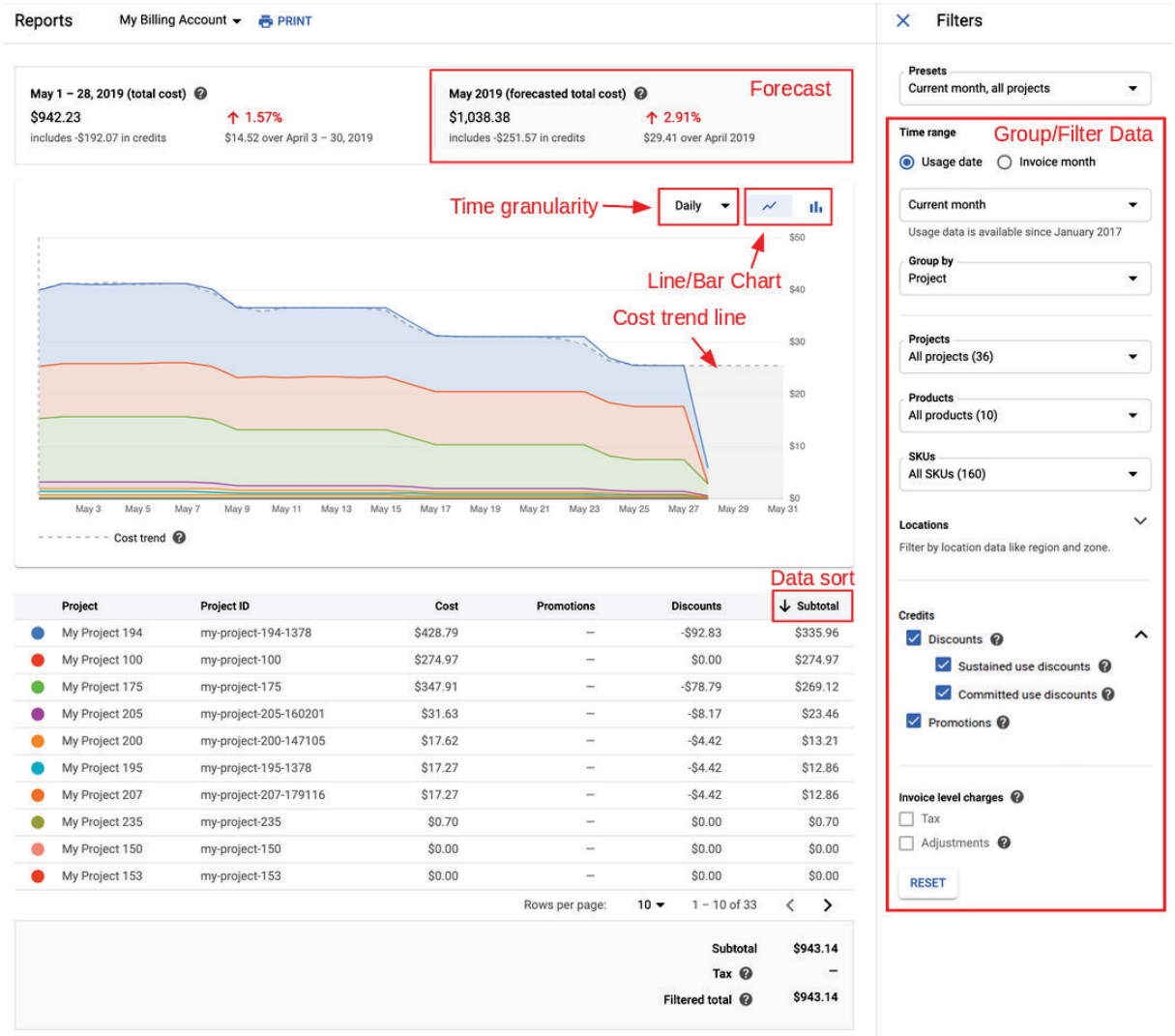


Figura 17: Relatório de faturação

À semelhança das soluções disponibilizadas pelos CSPs concorrentes, a análise permitida é limitada, impossibilitando a pesquisa por instâncias específicas ou filtragens por métricas para além das definidas. Verificou-se ainda que é bastante direcionada aos custos de operação dando menos ênfase ao dados de utilização dos recursos.

Analisadas as soluções fornecidas de forma gratuita por cada CSP e avaliados os prós e contras de cada uma, conclui-se que, embora auxiliem na resolução do problema, estas não o solucionam. Nesse sentido, decidiu-se continuar a investigar plataformas que suportem os CSP estudados e permitam a análise de custos de operação e utilização de recursos.

Por forma a identificar as soluções de monitorização e otimização de custos de *cloud* existentes no mercado foram analisados estudos de mercado relativos ao ano de 2018. Dos vários estudos averiguados, considerou-se que o apresentado por Lauren E. Nelson [6] constitui aquele que melhor avalia cada uma das soluções propostas. Neste estudo, cada uma das soluções foi avaliada de acordo com 25 fatores, que podem ser agrupados em três grandes níveis: a variedade de funcionalidades oferecidas, a estratégia e visão de negócio e ainda a presença que possui no mercado. Para além destes fatores, foi ainda considerado que as soluções alvo de análise teriam de permitir a monitorização e otimização de custos de *cloud*, suportar pelo menos 2 CSP e poder ser comercializada de forma independente.

O estudo realizado pelo Forrester Wave avaliou 9 soluções: Apptio [37], Cloudability [38], CloudCheckr [39], CloudHealth Technologies [40], Densify [41], Microsoft (Cloudyn) [42], RightScale (Optima) [43], Teevity [44] e Turbonomic [45]. Para cada solução foram aplicados os fatores referidos e realizada a classificação numa das seguintes categorias: *leaders*, *strong performers*, *contenders* ou *challengers*, conforme é possível observar na Figura 18.

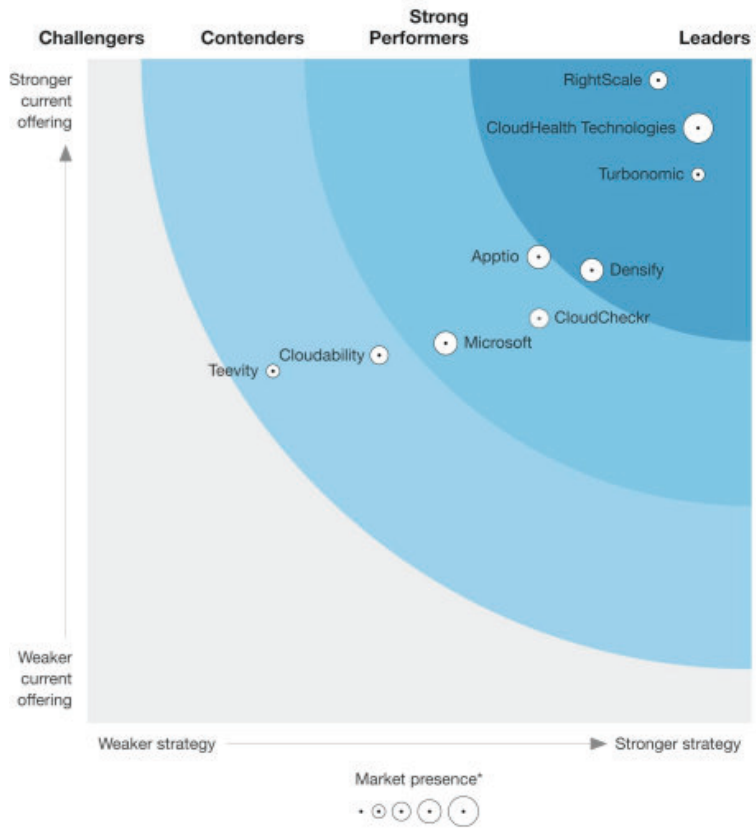


Figura 18: *Cloud Cost Monitoring And Optimization Providers*, Q2 2018 (**Adotado de:** *Lauren E. Nelson [6]*)

Na Figura 19 encontra-se representada a classificação de cada solução nos diversos fatores, bem como o peso atribuído a cada um desses parâmetros. Para poder identificar que funcionalidades estas ferramentas fornecem e compreender os mecanismos de obtenção de dados que possuem, decidiu-se explorar com maior profundidade uma destas soluções. As primeiras que se pretendia analisar eram aquelas que lideravam o mercado. Embora a *RightScale*, a *CloudHealth* e o *Densify* forneçam versões experimentais, estas só são atribuídas para utilizadores que possuam elevados custos de *cloud*, tornando impossível a exploração das mesmas. Em relação à solução oferecida pela *Turbonomic*, embora suporte várias fornecedores de *cloud*, disponibilizava uma solução para cada CSP ao invés de uma única ferramenta e como tal considerou-se que não era uma hipótese viável de analisar.

Observando a Figura 18 verifica-se que a *Apptio* seria a possível próxima solução a explorar, contudo, o facto de fornecer várias soluções de acordo com o objetivo pretendido e não uma única plataforma onde fosse possível realizar todas as operações, considerou-se que não seria a melhor ferramenta a analisar. No seguimento da análise das soluções, verificou-se que o *CloudCheckr* se tratava de um ótimo candidato a examinar pois trata-se de um *strong performer* bastante orientado à análise e otimização de custos de *cloud*. Nesse sentido, a Subsecção 2.4.4 reflete a análise realizada a esta solução, apresentando a ferramenta, as suas principais funcionalidades bem como algumas limitações.

	Forrester's weighting	Apptio	Cloudability	CloudCheckr	CloudHealth	Densify	Microsoft	RightScale	Teevity	Turbonomic
Current offering	50%	3.51	2.77	3.05	4.48	3.41	2.86	4.84	2.65	4.13
Cloud platform resources supported	15%	5.00	3.00	3.00	4.34	3.02	3.00	5.00	4.34	4.34
Cloud service administration and governance	15%	3.20	3.50	3.70	5.00	2.20	3.40	5.00	1.70	3.60
Cloud monitoring	20%	4.00	3.60	2.20	5.00	4.50	3.00	4.20	3.70	4.20
Cost optimization	25%	2.50	2.50	3.00	4.50	3.10	3.00	5.00	2.60	5.00
Support services	10%	3.00	3.00	3.00	3.00	5.00	3.00	5.00	1.00	3.00
Platform experience and SLAs	5%	5.00	1.00	1.00	3.00	3.00	3.00	5.00	3.00	1.00
Integrations and APIs	10%	3.00	1.00	5.00	5.00	3.00	1.00	5.00	1.00	5.00
Strategy	50%	3.40	2.20	3.40	4.60	3.80	2.70	4.30	1.40	4.60
Product vision	35%	3.00	3.00	3.00	5.00	5.00	1.00	3.00	1.00	5.00
Market approach	25%	3.00	3.00	3.00	5.00	5.00	3.00	5.00	1.00	5.00
Partner ecosystem	20%	3.00	1.00	5.00	5.00	3.00	3.00	5.00	1.00	5.00
Commercial model	20%	5.00	1.00	3.00	3.00	1.00	5.00	5.00	3.00	3.00
Market presence	0%	3.64	3.00	3.00	5.00	3.66	3.68	2.36	1.68	1.66
Number of customers	34%	1.00	3.00	3.00	5.00	3.00	5.00	5.00	3.00	1.00
Product revenue	33%	5.00	1.00	3.00	5.00	3.00	1.00	1.00	1.00	3.00
Market share	33%	5.00	5.00	3.00	5.00	5.00	5.00	1.00	1.00	1.00

All scores are based on a scale of 0 (weak) to 5 (strong).

Figura 19: *Cloud Cost Monitoring And Optimization Scorecard*, Q2 2018 (**Adotado de:** *Lauren E. Nelson [6]*)

2.4.4 CloudCheckr

O CloudCheckr foi criado em 2011 com o objetivo de ajudar as organizações a otimizar a mudança para o ambiente *cloud*. Originalmente foi desenvolvido no ecossistema da Amazon, mas desde então tem-se vindo a expandir fornecendo uma Cloud Management Platform (CMP) que serve múltiplos CSP, nomeadamente a Amazon Web Services (AWS), o Microsoft Azure e o Google Cloud Platform (GCP).

Esta CMP oferece aos seus clientes maior visibilidade e controlo sobre os custos, o aumento de desempenho e segurança, bem como a automatização de tarefas que permitem tornar o ecossistema de *cloud* mais eficiente. Responsável por gerir mais de mil milhões de dólares em gastos anualmente na AWS, o CloudCheckr transforma a complexidade e caos em clareza por forma a ajudar a manter os ambientes de *cloud* sob controlo [46].

Relativamente aos serviços fornecidos, conforme é possível observar na Figura 20, esta solução de gestão de *cloud* disponibiliza as seguintes funcionalidades [39]:

- **Análise e gestão de custos**

  Na área de gestão de custos são fornecidos mecanismos de alocação de custos, otimização de gastos e gestão de despesas. A análise de custos pode ser efetuada por *dashboards* e relatórios, que podem ser customizados através da definição de filtros e agrupamento de informação. Sobre determinados custos é possível estabelecer vistas de visualização de dados para explorar mudanças significativas na faturação. Ainda neste módulo é disponibilizada a análise detalhada de gastos por tipo de instância, sendo fornecidas várias estatísticas gerais, como custos estimados por região, plataforma ou tamanho da instância, bem como estatísticas de utilização e de avisos despoletados. A consulta de informação relativa à utilização de instâncias reservadas bem como instâncias *spot* constitui outra funcionalidade relevante. Por forma a auxiliar na gestão de recursos podem ser definidos alertas por período de tempo, com estabelecimento de diferentes patamares e filtros, notificando o utilizador por email.

- **Segurança e conformidade**

  A atividade dos utilizadores em plataformas públicas de *cloud* é monitorizada pro-ativamente, permitindo a quem administra implementar mecanismos para controlar acessos – Identity and Access Management (IAM). Nas configurações de segurança é possível definir permissões de acesso a dados dentro da CMP e a cifragem dos dados. Este módulo fornece ainda procedimentos de monitorização de recursos, através da documentação de *log* de forma automática, e mecanismos pro-ativos de identificação de riscos. Por forma a reduzir potenciais riscos, podem ser criados alertas relativos a modificações de configurações dos recursos de *cloud*.

- **Inventário de recursos**

  Por forma a compreender os serviços que estão a ser utilizados é fornecido um inventário que o utilizador possui. Para cada serviço é possível consultar um sumário com informações relevantes bem como analisar com detalhe as instâncias existentes. O fornecimento de uma visão das posições geográficas dos serviços permite ao utilizador compreender a distribuição dos mesmos pelo mundo. A gestão dos serviços e recursos é ainda simplificada pela possibilidade de procura por recursos que possuam ou não uma *tag* e pela criação de relatórios à medida. Este módulo ainda fornece funcionalidades de análise do histórico de alguns serviços.

- **Utilização dos recursos**

  Fornece estatísticas resumidas e detalhadas sobre a utilização de recursos na *cloud*, nomeadamente em termos de processador, tráfego de rede, entre outros. As métricas podem ser geridas e customizadas para facilitar a análise da utilização de recursos. Além disso, são oferecidos mecanismos inteligentes para dimensionar e escalar serviços eficientemente. Para auxiliar a atividade de gestão, é ainda possível definir um conjunto de alertas relativos à utilização de recursos, nomeadamente para o número de instâncias, percentagem de utilização, espaço de armazenamento utilizado e número total de objetos armazenados.

- **Recuperação automática**

  A automatização de ações na *cloud*, bem como a monitorização contínua e a execução de *snapshots* automáticos permite aos utilizadores poupar dinheiro, tempo e esforço e ainda criar mecanismos mais eficientes. De entre um conjunto alargado de automatismos destaca-se a gestão de instâncias, a eliminação de volumes e de *snapshots* desnecessários, bem como a deteção de grupos que não possuem recursos associados.

Para além destas funcionalidades, o CloudCheckr também fornece ferramentas específicas para a AWS das quais se destaca o CloudCheckr Migrate, o Reserved Instance Rebalancer e o CloudCheckr Spot Management. Por forma a poupar o mais possível é fornecido um conjunto de boas práticas para as diferentes características e ainda sugestões de poupança através da identificação de recursos com reduzida utilização e recomendações de aquisição de instâncias reservadas.

Em termos dos preços praticados por esta CMP, estes dependem do plano escolhido, sendo que em ambiente empresarial este valor é cerca de 2.5% da fatura de *cloud*, possuindo um valor mínimo de $500.

Embora se trate de uma ferramenta que fornece um conjunto extenso de funcionalidades e que vai de encontro às necessidades da maioria dos utilizadores, o CloudCheckr apresenta algumas limitações nomeadamente o defeituoso funcionamento de alguns serviços, a ineficiência no suporte ao utilizador e o fornecimento de uma interface gráfica demasiado complexa.

Para além disso, o nível de detalhe fornecido na análise de dados bem como nos relatórios gerados poderia ser mais profundo, a procura por um componente específico não se mostrou possível e a criação de novos elementos de análise era bastante limitada.

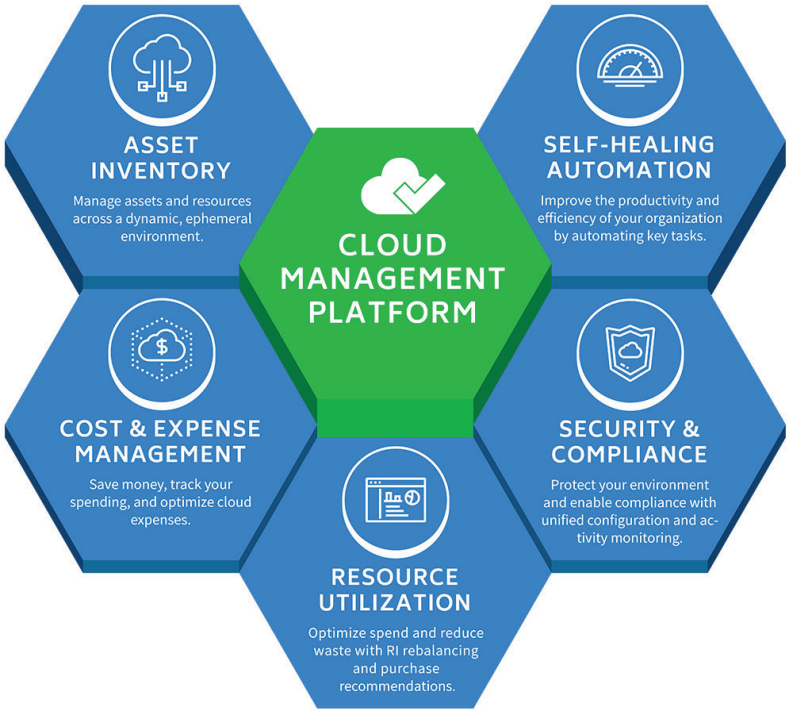


Figura 20: *CloudCheckr - Cloud Management Platform* (**Adotado de:** *Cloudcheckr [39]*)

2.4.5 CloudAware

À semelhança da solução anterior, o CloudAware é uma CMP segura, customizável e altamente escalável [47]. Trata-se de um SaaS desenvolvido tendo como base a plataforma *Force.com*, o que permitiu a criação de uma solução que agrega múltiplas funcionalidades numa única aplicação e que suporta múltiplos CSP: AWS, Microsoft Azure e GCP.

Relativamente aos serviços fornecidos, esta ferramenta de gestão de *cloud* fornece os seguintes módulos de funcionalidades [47]:

- Base de dados de gestão de configurações

  A Configuration Management Database (CMDB) fornecida permite realizar a monitorização em diversas áreas da infraestrutura de *cloud* em tempo real, facilitando a deteção de falhas de monitorização e segurança. A gestão de diversas contas dos vários CSPs é realizada numa única plataforma garantindo maior visibilidade dos recursos, alertando para falhas de segurança e zonas não monitorizadas.

- Cópias de Segurança e Replicação

  Facilita a realização de cópias de segurança e replicação de dados das instâncias utilizando as ferramentas nativas fornecidas pelos CSPs. A utilização deste serviço permite realizar este procedimento de forma manual ou automática, auxiliar na identificação de instâncias que não possuem *backups* e garantir a recuperação dos dados em caso de falha.

- *Cloud Trail +*

  Faculta mecanismos avançados de análise de dados da infraestrutura de *cloud* dos últimos sete anos, evidenciando informação relevante para o utilizador. É ainda possível identificar utilizadores inativos em diversas contas bem como alterações de um objeto específico.

- Gestão de alterações

  Este serviço permite identificar alterações não detetadas ou não revistas nas múltiplas contas dos fornecedores de *cloud*, nomeadamente no sistema operativo e em caso de violações ou intrusões. Além disso, fornece mecanismos que podem ser acionados sob determinadas circunstâncias.

- Gestão de custos

  Possibilita o controlo de custos e a identificação de possíveis poupanças. Para auxiliar este processo são fornecidos relatórios de análises de custos e detetados gastos excessivos ou desperdícios devido a instâncias com reduzida utilização. Os relatórios gerados podem ser customizados, exportados e enviados periodicamente [48].

- Monitorização

  Através do cruzamento de dados provenientes dos CSPs com dados dos agentes de monitorização, este módulo identifica serviços que não estão a ser monitorizados, enviando essa informação diretamente para o CMDB. Deste modo, é possível numa única plataforma identificar falhas na monitorização e mapear dados de utilização e desempenho às respetivas aplicações.

- Centro de gestão de ameaças

  Este módulo utiliza dados do CMDB e do servidor de deteção de intrusões para identificar recursos ou aplicações não protegidas, permitindo que o utilizador atue no sentido de garantir a segurança dos dados. Esta deteção pode ser automatizada através da definição de um conjunto de condições a satisfazer para executar a análise.

- Análise de utilização

  Facilita a análise de dados de utilização em tempo real para as diversas contas de *cloud*, auxiliando as tomadas de decisão da arquitetura da infraestrutura. Esta análise é realizada através da criação de *dashboards* e relatórios detalhados. Este módulo fornece ainda um histórico de utilização passível de ser analisado por forma a compreender não só a utilização nesse instante mas também a evolução ao longo do tempo.

Figura 21: Módulos do *CloudAware*

Para além destas funcionalidades, o CloudAware fornece um conjunto de funcionalidades relevantes como a existência de um analisador de *tags*, que permite identificar recursos etiquetados e auxiliar na pesquisa e filtragem de informação, a possibilidade de definir políticas passíveis de serem aplicadas periodicamente e ainda o agrupamento de recursos em "vistas" permitindo, por exemplo, analisar os recursos de uma dada aplicação. Relativamente aos preços praticados, esta CMP fornece diversos planos tendo por base o número mensal de instâncias. A título de exemplo, a um cliente com 285 instâncias e uma faturação mensal de $70 000, esta solução custaria aproximadamente $2 500/mês.

Apesar de se tratar de uma plataforma de gestão com um elevado conjunto de módulos, apresenta algumas limitações, nomeadamente a dificuldade de utilização, a inexistência de documentação e suporte e ainda o facto de ser bastante vocacionada para a segurança, dando menor importância à gestão de custos. Apesar dos relatórios fornecidos serem customizáveis, verificou-se que não é possível criar relatórios com o detalhe pretendido.

Deste modo, constatou-se que a ferramenta analisada, embora possua módulos que auxiliam na resolução do problema em estudo, estes não solucionam alguns aspetos essenciais que são relevantes para o tema desta dissertação.

Análise comparativa

Após terem sido exploradas duas soluções de otimização e monitorização de custos de *cloud*, considerou-se importante realizar uma análise comparativa entre as plataformas anteriormente apresentadas. Assim sendo, partindo do estudo do problema em causa e da investigação de possíveis soluções para a sua resolução, foi possível enumerar um conjunto de características que uma solução destas deve possuir, utilizando como base as funcionalidades que as soluções analisadas oferecem. Este conjunto divide-se essencialmente em dois grandes módulos: utilização de recursos e custos derivados dessa utilização. De seguida, apresenta-se de forma resumida as funcionalidades que cada um desses módulos deve possuir:

1. Utilização
    1.1. Visão geral sobre dados de utilização, idealmente sob o formato de *Dashboards*, passíveis de serem filtrados e agrupados por diversos parâmetros;
    1.2. Análise de instâncias de recursos com elevado nível de detalhe, explorando características como evolução/histórico da instância;
    1.3. Exploração de instâncias e serviços através da identificação e agrupamento de etiquetas;
    1.4. Geração, exportação e envio periódico de relatórios de utilização categorizados por períodos temporais, serviços, etiquetas, entre outra meta-informação;
    1.5. Criação de alertas para garantir o controlo sobre os dados de utilização;
    1.6. Identificação de recursos de acordo com o nível de utilização, permitindo assinalar instâncias com elevada/reduzida utilização;
2. Custos
    2.1. Visão geral sobre a alocação de custos por forma a ganhar visibilidade sobre o valor gasto num determinado fornecedor de serviços de *cloud*;
    2.2. Exploração de custos através da agregação de dados por diversos parâmetros nomeadamente por serviço, conta, aplicação, entre outros;
    2.3. Geração, exportação e envio periódico de relatórios detalhados de faturação, permitindo a personalização dos mesmos;
    2.4. Análise da alocação de custos através da comparação do valor gasto em relação ao orçamento estipulado, permitindo a identificação de gastos acima do definido;
    2.5. Criação de alertas personalizados para garantir o controlo sobre os custos;
    2.6. Previsão de custos tendo por base o histórico de custos de operação;
    2.7. Sugestão de medidas para redução e otimização de custos de operação;

Tabela 2: Comparação de funcionalidades entre CloudCheckr e CloudAware

	CloudCheckr	CloudAware
1.1	Apresentação da informação sobre o formato de resumo contudo não passível de ser filtrada ou agrupada. Os dados apresentados são suficientes para ter uma visão geral do estado da *cloud*.	Informação fornecida a alto nível, identificando apenas o nº de instâncias ativas de cada serviço. Nenhum dado relativo a recursos é apresentado, não sendo possível filtrar ou agrupar a informação.
1.2	Os recursos podem ser explorados com elevada profundidade, permitindo compreender quantidades e valores gastos. É possível verificar à quanto tempo existem os recursos, contudo a evolução dos mesmos não é fornecida.	As instâncias são altamente detalhadas, sendo possível observar o montante gasto nesse recurso, estado da instância nos últimos 90 dias em termos de performance, bem como histórico de alterações.
1.3	As instâncias possuem informação relativamente à existência ou não de *tags*. Existe a separação clara entre recursos etiquetados e não etiquetados, porém a informação fornecida só permite contabilizar o número de instâncias nestas condições. O agrupamento dos dados não é realizável nesta ferramenta.	Esta solução possui um analisador de *tags* que permite identificar recursos etiquetados e agrupá-los em vistas distintas, possibilitando a análise dos recursos noutra perspetiva. Para cada etiqueta é ainda possível verificar o grau de cobertura dos diversos atributos.
1.4	A geração e exportação de relatórios nos formatos CSV e PDF distinguem esta ferramenta das restantes. Os relatórios são personalizáveis contudo limitam-se aos principais serviços. O envio ocorre diariamente ou semanalmente por email.	Os relatórios de utilização são customizáveis contudo o nível de detalhe é bastante reduzido. A exportação é possível porém só no formato CSV. O envio dos mesmos por email não se verificou possível nesta ferramenta.
1.5	A definição de alertas para o controlo de utilização das instâncias é possível nesta solução através do estabelecimento de valores ou patamares de utilização.	Esta ferramenta permite a criação de um conjunto de *policies* que permitem definir alertas relativamente à utilização excessiva ou reduzida de recursos/serviços.
1.6	Os recursos podem ser explorados por parâmetros como CPU ou *storage*, permitindo identificar instâncias com reduzida ou nenhuma utilização, bem como precaver a sobrecarga das mesmas.	A identificação de recursos por parâmetros não é possível, contudo o detalhe individual de cada um dos recursos permite tirar conclusões acerca do nível de utilização da instância.

2.1	A informação é apresentada sob a forma de um *dashboard* fornecendo informação sumariada de custos por mês bem como pelos diversos serviços do fornecedor de *cloud*. Esta informação pode ainda ser filtrada por períodos temporais.	A informação relativamente aos custos de *cloud* é apresentada maioritariamente sob o formato gráfico mas também tabular. São fornecidos *dashboards* com a informação de faturação permitindo a filtragem por intervalos de tempo.
2.2	Os dados de billing podem ser filtrados por serviço, tipo de custo, mês de faturação e ainda por conta. Esta informação pode ser agrupada por diversos fatores simples ou compostos.	Os dados disponibilizados podem ser filtrados por conta, serviço, tipo de utilização, entre outros. No entanto, esta ferramenta não permite o agrupamento da informação de faturação.
2.3	Esta solução fornece relatórios de custos de utilização bastante detalhados. Estes relatórios podem ser exportados para o formato CSV e PDF.	Os custos de utilização podem ser sumariados em relatórios contudo com reduzido detalhe. A exportação dos mesmos apenas é possível no formato CSV.
2.4	A análise de custos em comparação ao orçamento estipulado pode ser feita por serviço, contudo a alto nível e apenas para os principais serviços.	A gestão de orçamento e valores estipulados não é possível de ser realizada nesta solução de análise.
2.5	O controlo de custos é assegurado pela criação de alertas. Estes alertas podem ser configuráveis tendo em conta o orçamento estabelecido e patamares limites para o alcançar, podendo ainda serem aplicados sob determinadas condições.	À semelhança dos alertas de utilização, é possível criar políticas relativamente aos custos de utilização. As políticas podem ser executadas periodicamente, contudo a criação das mesmas revela-se difícil devido à necessidade de escrita de código.
2.6	É fornecida uma previsão de custos até ao máximo de 30 dias tendo por base os valores gastos no mês anterior. A estimativa é realizada através de uma regressão linear dos valores recolhidos.	Esta ferramenta não fornece qualquer mecanismo de previsão de custos, impossibilitando a estimativa de custos de utilização.
2.7	Este solução fornece um conjunto de boas práticas bem como medidas para reduzir os custos. A aquisição de instâncias mais rentáveis monetariamente e a migração de serviços são exemplos das medidas propostas.	A sugestão de possíveis medidas para a redução e otimização de custos não constitui uma característica desta solução.

3

O problema e desafios da análise de custos de Cloud

Neste capítulo será detalhado o problema em estudo e quais os seus principais desafios tendo como base a revisão da literatura apresentada no Capítulo 2. Partindo da informação recolhida e analisada, será apresentada uma possível proposta de solução.

3.1 Problema

Apresentado o contexto e motivação que impulsionou o desenvolvimento deste projeto – Secção 1.1 – e avaliada a viabilidade de adotar soluções de análise de custos operacionais na *cloud* – Secção 2.4 – identifica-se, de seguida, o problema alvo de estudo deste trabalho.

Conforme introduzido na Secção 2.3, as políticas de definição de preços exercidas pelos CSP têm vindo a tornar-se cada vez mais complexas e difíceis de utilizar. O aumento desta complexidade e o fornecimento crescente de um maior número de serviços por parte dos fornecedores de *cloud* dificulta a tarefa de gerir os custos provenientes da utilização destes serviços. A ineficiência na gestão de custos põe em causa um dos principais motivos dos utilizadores na escolha da *cloud*: a redução de custos.

Por forma a combater este problema, foram exploradas as soluções de análise de custos disponibilizadas gratuitamente por cada CSP. Embora permitissem a consulta de custos, eram bastante limitativas e não iam de encontro aos objetivos pretendidos. De seguida, estudaram-se duas soluções de gestão de *cloud*: CloudCheckr (Secção 2.4.4) e CloudAware (Secção 2.4.5). Apesar destas ferramentas ajudarem a combater o problema, a visibilidade sobre os custos e dados de utilização é limitada, não sendo possível consultar a informação com a granularidade pretendida. Além disso, nenhuma destas ferramentas permite gerir os custos derivados dos diversos componentes pela qual uma aplicação *web* é constituída.

Nesse sentido, com este projeto pretende-se desenvolver uma solução que auxilie os utilizadores de *cloud* a gerir os custos de operação de aplicações *web* em *cloud*. A ferramenta de análise tem como principal objetivo fornecer uma visão sobre os dados de utilização e custos de operação dos diversos componentes das aplicações, permitindo examinar detalhadamente estes dados sobre diversos prismas.

3.2 Desafios

Ao longo do desenvolvimento da dissertação foram enfrentados diversos desafios na tentativa de solucionar o problema identificado na Secção 3.1. Alguns destes desafios levaram a tomadas de decisão que, por sua vez, condicionaram a abordagem ao problema e a proposta de solução do mesmo.

3.2.1 Obtenção de informação

A obtenção de informação para análise de custos e dados de utilização constituiu o primeiro desafio encarado nesta dissertação. Uma vez que se pretende analisar dados de diversos CSPs é importante apresentar uma solução genérica, isto é, independente de qual é o CSP escolhido. Deste modo, considerou-se que os mecanismos de angariação de dados constituem um ponto fulcral na elaboração da solução.

Primeiramente foi investigado quais as possíveis fontes de informação para obter dados de faturação bem como de utilização. Nesta análise foi tido em conta, não só o formato dos dados mas também a forma de obtenção dos mesmos e custos derivados dessa aquisição.

Analisados os CSPs apresentados na Secção 2.2, foi possível verificar a existência de fontes de informação semelhantes entre eles. Após estudar os serviços e informação fornecida por cada um, foram identificadas três possíveis fontes de dados:

- Relatórios - Os fornecedores de *cloud* investigados fornecem dados relativos à utilização e custos de operação na *cloud* sob a forma de relatórios, alguns deles altamente detalhados. Embora a informação que contêm difira entre eles, verificou-se que possuem a informação essencial para a solução a desenvolver.

- APIs - A existência de APIs disponibilizadas pelos CSPs permite obter informação de faturação bem como dados de utilização programaticamente. Embora algumas destas APIs sejam taxadas, constatou-se que o número de invocações disponibilizado pela camada gratuita de cada CSP – um milhão de pedidos mensais – é suficiente para adquirir esta informação sem custos adicionais.

- Agentes - O desenvolvimento de um agente capaz de ser instalado nas máquinas sobre as quais se pretende adquirir informação permite obter dados de utilização dos recursos e em conjunto com a informação dos preços praticados pelo CSP, possibilitar a estimativa dos custos dessa utilização.

Embora estas três alternativas sirvam de fonte de dados à solução a desenvolver, cada uma possui as suas vantagens e desvantagens que serão analisadas de seguida.

Os relatórios providenciados pelos fornecedores de *cloud* podem ser exportados no formato CSV, o que permite que sejam facilmente processados. Como desvantagem salienta-se o período de atualização destes relatórios, podendo variar desde uma atualização diária até intervalos de 8 horas, dependendo da utilização, dos custos de operação e do CSP.

A utilização de uma Representational State Transfer (REST) API [49] permite que a informação desejada seja obtida através de pedidos Hypertext Transfer Protocol (HTTP) [50] e com recurso aos métodos disponibilizados – GET, PUT, POST, DELETE e PATCH. Das APIs analisadas verificou-se que todas fornecem os dados sob o formato JavaScript Object Notation (JSON) [51], o que facilita o tratamento e processamento dos mesmos. Negativamente destaca-se o facto de em alguns CSPs ser necessário efetuar vários pedidos a diversos *endpoints* das APIs para conseguir obter a informação pretendida. Além disso, algumas destas APIs revelaram-se difíceis de usar devido às políticas de segurança e autenticação que as mesmas implementam. Para facilitar a sua utilização, os CSPs disponibilizam Software Development Kits (SDKs) implementados em diversas linguagens de programação.

A utilização de agentes tem como vantagem a monitorização e obtenção de informação em tempo real, contudo a diversidade de sistemas operativos e serviços a monitorizar dos vários CSP implica o desenvolvimento de diversos agentes. Para além disso, a consulta constante dos preços que o fornecedor de *cloud* pratica representa um desafio em termos de implementação e de custos associados.

Investigadas as diversas fontes de informação disponibilizadas e explorados os diversos prós e contras de cada uma, concluiu-se que a solução a desenvolver deveria combinar a informação fornecida pelos relatórios e pelas APIs. De seguida, apresenta-se os relatórios e APIs fornecidas por cada fornecedor de CSPs referido na Secção 2.2.

Relatórios

A AWS disponibiliza um elevado número de relatórios com diversos conteúdos e níveis de detalhe, fornecendo informações dos recursos utilizados e custos associados a essa utilização. Dos diversos relatórios salientam-se de seguida aqueles que são relevantes para o objeto em estudo [52]:

- *Cost and Usage Report (CUR)*: trata-se do relatório mais recente da Amazon e que permite controlar a utilização dos recursos e os custos provenientes dessa utilização. Este relatório contém a informação organizada por itens para cada combinação única do produto, tipo de utilização e operação utilizadas. A agregação da informação deste relatório pode ser realizada por hora ou dia. Este relatório encontra-se armazenado num *bucket* S3 e é atualizado até 3 vezes por dia.

- *Detailed Billing Report (DBR)*: relatório similar ao *AWS CUR*. A informação que contém relativamente aos custos é semelhante mas a forma como os itens são calculados difere. Relativamente à frequência de atualização, este relatório é recriado múltiplas vezes por dia sendo reescrito em cada momento. No final do mês é criado um relatório final e armazenado num *bucket* S3.

- *Detailed Billing Report with Resources and Tags*: consiste num relatório semelhante ao DBR contudo mais completo devido a presença da informação dos recursos e das *tags* existentes. Graças a estes atributos é possível com este relatório identificar recursos dos diversos serviços da AWS através de filtragens e agregações de dados.

- *Monthly Report*: relatório gerado com características semelhantes ao DBR porém a agregação dos dados é realizada ao mês. Os dados são atualizados várias vezes por dia. É ainda possível consultar relatórios de meses anteriores.

- *AWS Usage Report*: permite obter relatórios relativos a um único serviço, sendo possível definir o tipo de utilização, operação e período temporal a incluir. Além disso, o utilizador pode escolher agregar os dados por hora ou dia.

Dos relatórios apresentados anteriormente verificou-se que o CUR é aquele que fornece informação mais completa, apesar de implicar uma maior complexidade de análise uma vez que se trata de um ficheiro CSV com informação e número de colunas variável. Este tipo de relatório não se encontra ativo por omissão na AWS, contudo a sua criação é bastante simples. Para isso basta criar um *bucket* S3 ou utilizar um já existente e, na categoria de "Billing", criar o relatório pretendido com o nome, prefixo e granularidade temporal desejada [53]. Dado que os restantes relatórios são automaticamente disponibilizados ao utilizador, considerou-se que a solução a desenvolver deveria não só suportar os CURs mas também outros que fossem relevantes, nomeadamente os DBRs.

À semelhança da AWS, o Microsoft Azure permite a exportação dos dados de custos para ficheiros CSV [54]. Contrariamente ao CSP analisado anteriormente, o Azure apenas disponibiliza um único tipo de relatório. Para poder aceder a este relatório é necessário criar um "*Schedule export*" que é executado na granularidade pretendida: diariamente, semanalmente ou mensalmente. Este componente é responsável por criar o relatório e armazená-lo no *container* desejado. Este *container* é semelhante ao *bucket* da AWS e tem a si associado uma *Storage Account*, onde este se encontra e através da qual é possível aceder ao mesmo. Na criação do *exporter* é ainda solicitado o nome do relatório bem como a diretoria onde se pretende que seja guardado. A solução a desenvolver deve utilizar este relatório para obter a informação relativa a custos de utilização de uma conta do Microsof Azure.

Relativamente ao Google Cloud Platform (GCP), constatou-se que este CSP também permitia a exportação de dados de faturação sob o formato de ficheiros CSV ou JSON . Estes ficheiros são armazenados num *bucket* do Google Cloud Storage (GCS) escolhido pelo utilizador e acessíveis via API [55]. Para a

criação deste relatório é necessário criar um *bucket* de armazenamento e ativar a exportação dos dados para ficheiro na secção de "*Billing*", escolhendo um prefixo para o ficheiro e o formato pretendido [56]. Posto isto, assume-se que a solução a projetar deve suportar o relatório fornecido pelo GCP.

Application Programming Interface (API)

A AWS disponibiliza um conjunto de APIs que permitem a obtenção de informação relativa aos custos de utilização dos serviços de *cloud*. Das APIs estudadas verificou-se que a *AWS Billing and Cost Management API* e a *AWS Cost Explorer API* são aquelas que fornecem os dados pretendidos para a análise por parte da solução a desenvolver [57]. Os *endpoints* facultados por estas APIs possibilitam a consulta de diversa informação nomeadamente métricas de consumo relativas a custos e utilização de recursos, previsão da evolução destas métricas, aquisição e cobertura de instâncias reservadas, entre outras [58].

Por forma a enriquecer a informação relativa a dados de utilização de recursos foi realizada uma pesquisa aprofundada de outros serviços que disponibilizam esta informação através de uma API. O *CloudWatch* é um serviço de monitorização fornecido pela AWS responsável por recolher dados de operação sob a forma de *logs*, métricas e eventos [59]. Este serviço fornece uma API que permite publicar, monitorizar e gerir métricas de análise dos recursos. O acesso a esta API é facilitado através da utilização de um dos SDK fornecido pelo CSP, implementados em diversas linguagens de programação [60].

O Microsoft Azure faculta APIs para adquirir dados referentes à faturação e utilização de recursos, concretamente a *Azure Billing API* e a *Azure Resource Usage API* [61]. Embora estas APIs forneçam a informação desejada, apenas se encontram disponíveis para contas do tipo *enterprise*. Dado que esta dissertação de mestrado se encontra inserido num âmbito empresarial, considerou-se que a solução deveria suportar a integração com estas APIs.

À semelhança do *CloudWatch* da AWS, o Azure também oferece um serviço de recolha e análise de dados telemétricos da infraestrutura criada na *cloud*, designado de *Azure Monitor* [62]. O *Azure Monitor* fornece uma API que permite um conjunto alargado de operações das quais se destacam a listagem de definições das métricas recolhidas e a obtenção de valores para uma métrica num dado intervalo temporal com uma determinada frequência [63].

Contrariamente aos dois CSP, o GCP não fornece qualquer API que permita obter informação relativamente a custos de utilização. Até ao momento desta dissertação, apenas disponibiliza uma API para gerir configurações de faturação (*Cloud Billing API*) [64].

Relativamente ao dados de utilização de recursos, o GCP fornece, a semelhança dos restantes CSP, um serviço de monitorização e gestão de serviços, aplicações e infrastruturas denominado de *Stackdriver* [65]. O *Stackdriver* agrega métricas, *logs* e eventos ocorridos na infraestrutura e faculta essa informação através de um conjunto de APIs. Das APIs analisadas, verificou-se que a *Stackdriver Monitoring API* é aquela que possibilita a obtenção de métricas de utilização de recursos [66].

3.2.2 Processamento de informação

Outro desafio que surgiu durante o desenvolvimento desta dissertação relaciona-se com o processamento da informação a recolher. Independentemente da abordagem escolhida, qualquer uma das duas alternativas apresentadas na Secção 3.2.1 representam um desafio em termos de processamento.

Embora os relatórios e APIs fornecidas pelos CSPs estejam em formatos fáceis de trabalhar, a informação que contém pode relevar-se algo complexa de processar. A diversidade de relatórios disponibilizados pelos vários fornecedores de serviços de *cloud* implica diferentes métodos de processamento. Para além disso, a própria heterogeneidade de alguns campos exige que estes métodos sejam robustos.

Outro fator a destacar nesta abordagem é o tamanho dos ficheiros CSV. Dado o alto nível de detalhe destes relatórios, estes facilmente possuem um elevado número de entradas – na ordem das centenas de milhares a milhões de registos – gerando ficheiros de grandes dimensões, na ordem do GB. Caso semelhante acontece com as APIs, uma vez que se estivermos perante uma infraestrutura de dimensões consideráveis, a quantidade de informação disponibilizada será na grandeza dos milhares de registos diários.

Deste modo, a solução a propor deverá ser robusta, de modo a suportar a quantidade de informação a tratar, mas também ser capaz de processar informação heterogénea proveniente de diferentes fontes de informação.

3.2.3 Identificação de recursos

A identificação de recursos constituiu outro desafio na elaboração da solução para o problema em estudo. Conforme referido na Secção 3.1, deverá ser possível analisar custos de operação dos diversos componentes que constituem uma aplicação. Para tal, foi necessário investigar possíveis alternativas para solucionar este problema e concluiu-se que a utilização de *tags* seria a melhor solução, uma vez que é possível de aplicar em qualquer um dos CSP em estudo [67, 68, 69]. Para além de permitir a alocação de custos é ainda útil para controlar acessos ou implementar mecanismos de automação.

O mecanismo de *tagging* é um dos mais importantes para obter visibilidade de custos e de utilização de recursos na *cloud*. *Tags* são *labels* atribuídas aos recursos, constituídas por metadados e formadas por uma combinação chave-valor [70].

A definição do conjunto de *tags* a aplicar aos recursos de uma infraestrutura deverá seguir uma política bem definida por forma a maximizar a gestão da *cloud* [71, 72]. Existem diversas estratégias e boas práticas no que se refere ao processo de *tagging*, porém estas assentam, na generalidade, nas categorias representadas na Figura 22.

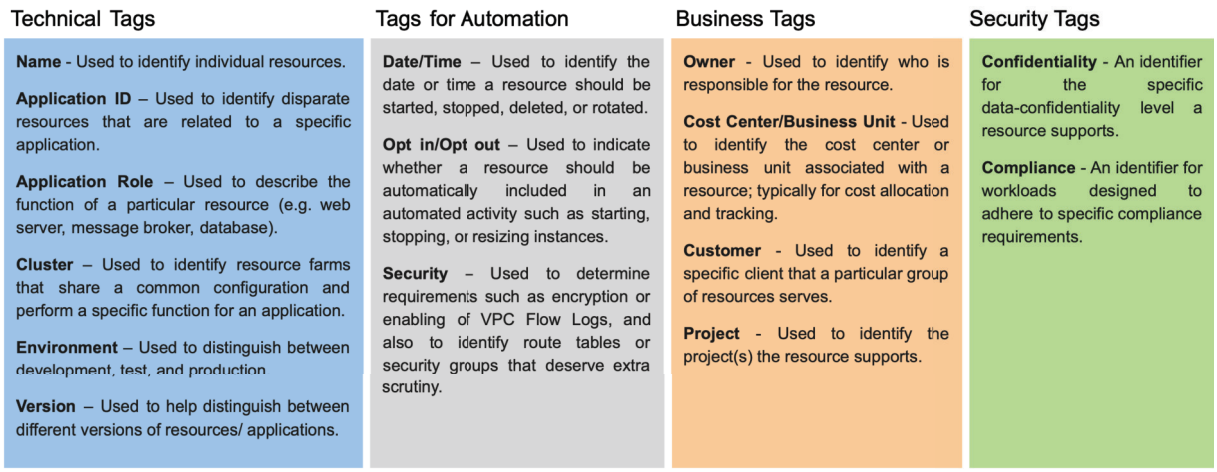


Figura 22: *Tagging Categories* (**Adotado de:** *AWS Tagging Strategies [67]*)

Das categorias de *tags* apresentadas, salientam-se como mais relevantes para a análise pretendida aquelas que permitem identificar um recurso tendo em conta o seu carácter técnico e área de negócio. Embora seja importante criar um conjunto de *tags* que satisfaçam as necessidades dos utilizadores, é necessário clarificar o contexto de cada *tag* por forma a evitar *labels* redundantes *(overtagging)* [73].

Uma estratégia eficaz de *tagging* implica a definição clara das *tags* a utilizar e a implementação de forma consistente das mesmas por todos os recursos da infraestrutura. Por forma a garantir a consistência na atribuição de *tags* é sugerido pelos diversos CSP a utilização de APIs desenvolvidas com esse propósito [67]. Esta coerência pode ainda ser garantida programaticamente caso os recursos sejam criados através de ferramentas de gestão de infraestruturas como, por exemplo, o `Terraform`[74] ou o `Chef`[75].

3.3 Proposta de solução

Descrito o problema a resolver e os desafios enfrentados no decorrer desta dissertação, apresenta-se de seguida uma possível solução para o problema em estudo – o artefacto.

A solução a desenvolver deve suportar diversos fornecedores de *cloud*, nomeadamente os estudados na Secção 2.2. Para além disso, conforme referido na Subsecção 3.2.1, esta deve ter como fonte de informação os relatórios e APIs fornecidos pelos CSPs. De acordo com o exposto na Subsecção 3.2.2, a solução proposta deverá ser robusta o suficiente para processar dados heterogéneos e de grandes dimensões. Tratada a informação, esta deverá ser armazenada por forma a poder ser analisada.

Outro aspeto essencial na elaboração da proposta de solução é a definição dos requisitos e características que a aplicação a desenvolver deverá suportar. Estudando as funcionalidades que as soluções referidas na Secção 2.4 fornecem e investigando os tipos de utilizador da ferramenta determinou-se que os principais módulos devem residir na gestão de custos e na análise da utilização de recursos. Deste

modo, os requisitos que a solução deverá cumprir foram identificados nessa Secção, no momento de comparação das soluções.

Da pesquisa efetuada conclui-se ainda que os utilizadores da ferramenta serão essencialmente profissionais de I&O, pelo que dado o elevado conhecimento do domínio e capacidade de lidar com ferramentas de gestão e análise de dados, a solução deve assentar em *dashboards* e relatórios técnicos. Expostas todas as condicionantes que possam restringir o *design* da solução, descreve-se de seguida a tecnologia usada na proposta de solução.

Em primeiro lugar decidiu-se que deveria ser criada uma aplicação que permitisse a angariação de informação dos diversos CSPs. Esta aplicação deve ser modular e escalável tendo em vista a integração com outros CSPs e a obtenção de informação para além da já mencionada. A linguagem de programação escolhida foi o *Python* por ser uma linguagem de fácil aprendizagem e amplamente usada no mundo da Ciência de Dados e do *Big Data*. Uma vez que seria necessário fornecer uma API para permitir a angariação dos dados, decidiu-se utilizar uma *framework* de *Python* denominada *Flask* [76].

De seguida, foram estudadas as tecnologias que permitissem o processamento, armazenamento e consulta de informação. Atendendo à área em que se enquadra a ferramenta a desenvolver, considerou-se que a solução *open source* fornecida pela *Elastic*, denominada por *Elastic Stack*, satisfazia as necessidades impostas. O facto de ser amplamente usada em I&O, nomeadamente no âmbito de *DevOps*, contribuiu positivamente para esta decisão [77].

A *Elastic Stack* é constituída por um conjunto de produtos *open source* criada com o objetivo de auxiliar os utilizadores a recolher dados de qualquer tipo de fonte e em qualquer formato e facilitar a pesquisa, análise e visualização desses dados em tempo real. Esta solução pode ser disponibilizada como um SaaS na *cloud* pela *Elastic* ou implementada localmente pelos utilizadores [78].

Dos componentes que constituem a *Elastic Stack*, destacam-se como relevantes para a solução o *Logstash*, o *ElasticSearch* e o *Kibana*. Para a implementação do sistema de alarmística, decidiu-se utilizar um componente *open-source* que integra diretamente com o *ElasticSearch* denominado Elastalert [79].

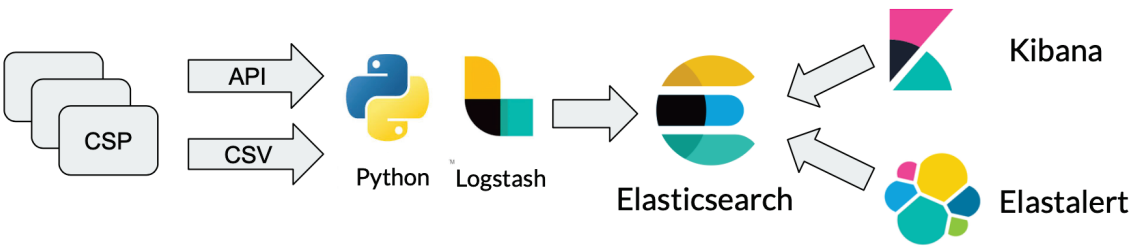


Figura 23: Esquema da proposta de solução

Logstash

O *Logstash* é um motor de recolha de dados que permite a unificação de dados de diferentes fontes, a sua normalização e distribuição dos mesmos pelo "stash" desejado [80]. Este componente da *stack* implementa um mecanismo de Extract, Transform, Load (ETL) essencial ao desenvolvimento da solução.

De forma genérica, o *Logstash* é orquestrado através de ficheiros *YAML* e ficheiros de configuração onde são definidas as *pipelines* para o tratamento de dados. Uma *pipeline* é constituída por: um conjunto de *inputs* de onde são obtidos os dados, *filters* a aplicar aos dados recolhidos e *outputs* para onde enviar a informação tratada. Em cada uma destas secções podem estar definidos um ou mais *plugins*, sendo que no caso dos *filters* estes são aplicados pela ordem que aparecem no ficheiro.

Estes *plugins* são *packages* denominados por `gems` que se encontram hospedados no `RubyGems.org`. O *Logstash* disponibiliza um conjunto alargado de *plugins* de *inputs*, *filters* e *outputs* suportados pela *Elastic* e sua comunidade. De entre os *inputs plugins* destaca-se o `file`, `tcp` e `http`, enquanto nos *output plugins* o `elasticsearch` e o `csv`. Nos *filter plugins* salientam-se os seguintes: `grok`, `json`, `mutate`, `date`, `split`, `translate` e `ruby`. Para além destes é possível ainda criar *plugins* próprios, sendo a linguagem de base o *Ruby*.

A instalação e utilização de *plugins* é bastante simples uma vez que apenas é necessário executar o comando fornecido pelo *Logsatsh* para o efeito, por exemplo `bin/logstash-plugin install logstash-input-tcp`, e utilizar o nome atribuído ao *plugin* (`tcp`) no ficheiro de configuração com as opções pretendidas.

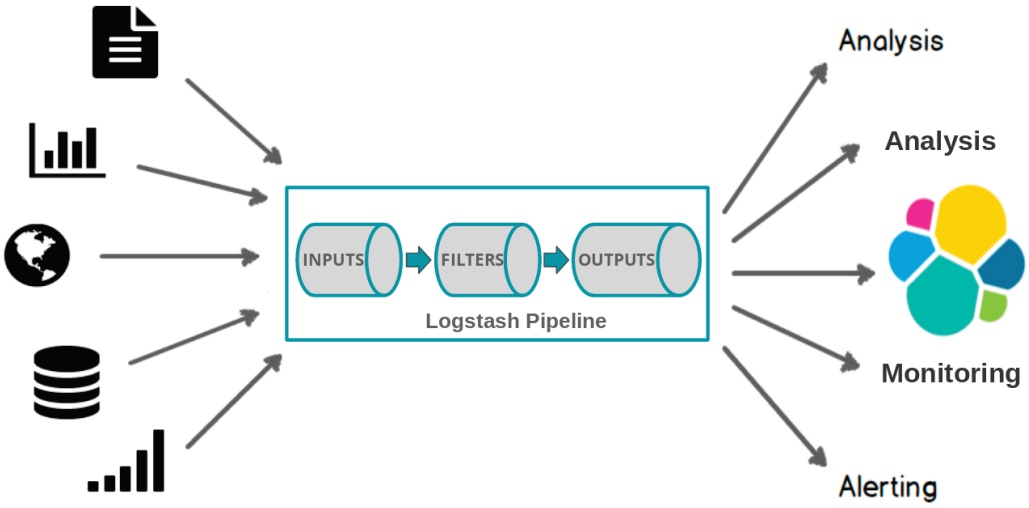


Figura 24: Logstash Pipeline

ElasticSearch

O *ElasticSearch* é um motor de pesquisa e análise distribuído baseado no *Apache Lucene* [81]. O *Apache Lucene* é uma biblioteca escrita em *Java* que fornece mecanismos de pesquisa de texto com elevado desempenho, que permite fornecer dados em praticamente tempo real. Dada a sua arquitectura, o *ElasticSearch* fornece alta disponibilidade através de mecanismos de *clustering*. A inserção e consulta de dados é ainda facilitada graças a uma *RESTful* API.

Existem alguns conceitos teóricos relativos à arquitectura do *ElasticSearch* que é essencial compreender para o desenvolvimento da solução [82]. O *ElasticSearch* é constituído por uma colecção de um ou mais *nodes* onde se encontram armazenados os dados e que, no seu conjunto, formam um *cluster*. Um *node* é um servidor único que faz parte do *cluster* e cujas principais funções passam pelo armazenamento de dados, bem como auxiliar na indexação e execução de pesquisas. Num único *cluster* é possível definir diversos índices [83].

Um índice é uma coleção de documentos com características semelhantes e identificado por um nome único. Esse nome é usado para referenciar o índice durante a execução de operações de indexação, pesquisa, atualização e exclusão em relação aos documentos nele contidos. Um documento é a unidade básica de informação a ser indexada e é representado sob o formato JSON [83].

Quando comparados estes conceitos com os de base de dados relacionais, verifica-se que um índice pode ser visto como uma base de dados. Os diferentes tipos de documentos representam uma analogia para as tabelas que constituem o esquema da base de dados. Por sua vez, um documento pode ser equiparado a uma linha de uma tabela e os campos que o constituem às colunas dessa tabela [82].

Outra característica interessante desta tecnologia é a capacidade de subdividir um índice em vários fragmentos denominados "shards" [83]. Cada *shard* é ele próprio um "índice" funcional e independente que pode ser alocado em qualquer *node* dentro do *cluster*. Este mecanismo de *sharding* é útil em casos em que o espaço disponível num *node* não é suficiente para albergar um índice, devendo ser subdividido entre vários *nodes*. Os *shards* permitem ainda distribuir e paralelizar operações, aumentando o desempenho das mesmas.

Por forma a fornecer alta disponibilidade em caso de falha de um *node*, o *ElasticSearch* fornece ainda um mecanismo de replicação. A criação de uma ou mais cópias de um *shard* permite que as pesquisas sejam executadas em todas as réplicas em paralelo [83].

Relativamente ao armazenamento de dados, este é feito sobre a forma de documentos através de uma técnica denominada *inverted index* [84]. O propósito desta técnica é armazenar texto numa estrutura de dados que permita pesquisas rápidas e eficientes. Deste modo, quando são realizadas *queries*, estas são efetuadas sobre os *indexes* e não diretamente nos documentos armazenados.

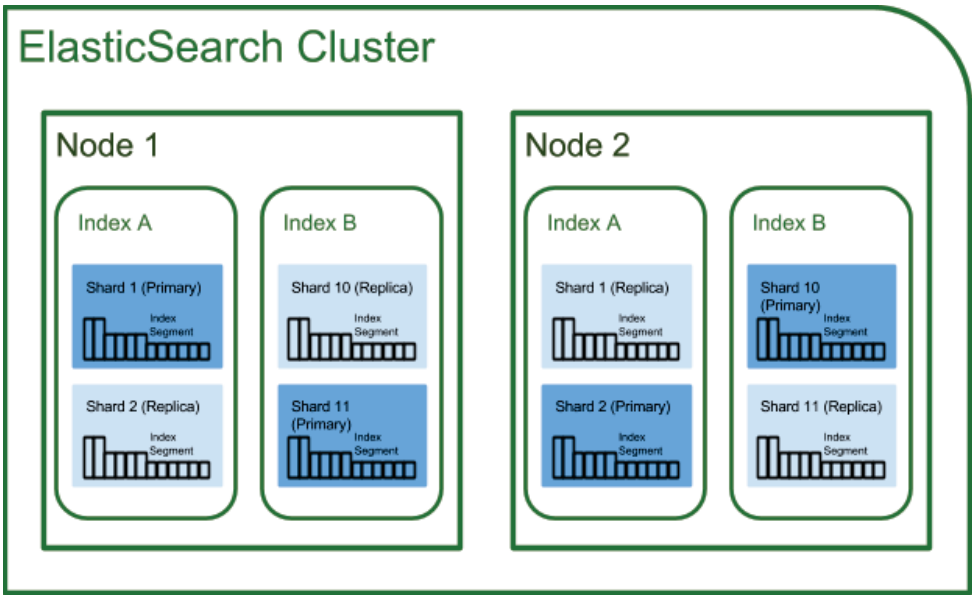


Figura 25: *ElasticSearch Cluster Architecture*

Kibana

O *Kibana* é uma ferramenta de visualização desenvolvida para analisar grandes volumes de dados de forma gráfica. Dos diversos gráficos que disponibiliza destacam-se os gráficos de linhas, de barras, circulares bem como *heat maps*, *region maps*, *gauge*, *goals*, entre outros [85].

Através do *Kibana* é possível executar *queries*, agregações e filtragens sobre os dados armazenados no *ElasticSearch*. Trata-se de uma aplicação constituída por diversos componentes, denominados de *plugins*, dos quais se salientam os seguintes [86]:

- *Discover* - consulta de documentos armazenados num dado índice;
- *Visualize* - elaboração de visualizações sob a forma de gráficos;
- *Dashboard* - construção de *dashboards* através do agrupamento de visualizações;
- *Canvas* - criação de *workspaces* semelhantes ao *Dashboard* com interfaces customizáveis.

Para além destes, o *Kibana* fornece ainda outros *plugins* interessantes, nomeadamente o *Dev Tools*, o *Stack Monitoring*, o *Management* e o *Security*, úteis para auxiliar no desenvolvimento e gestão de toda a *Elastic Stack*, bem como para garantir o controlo de acessos e a encriptação das comunicações.

Dos diversos conceitos que é necessário possuir para compreender esta ferramenta salienta-se o conceito de *space*. Os *spaces* permitem organizar e separar visualizações, *dashboards*, bem como outros objetos de acordo com um dado critério. No ambiente em que está a ser aplicada esta tecnologia, este conceito é importante, por exemplo, para efetuar uma separação entre contas de utilização, entre fornecedores de *cloud*, entre outros.

Outro conceito que também é importante assimilar é o de *index pattern*. Os *index patterns* permitem indicar ao *Kibana* sobre que índices do *ElasticSearch* deve executar as suas pesquisas, bem como criar visualizações. Um *index pattern* pode corresponder a um único índice armazenado no *ElasticSearch* ou a um conjunto de índices. Por exemplo, se tivermos índices dinâmicos como "`logstash-7-2019.08.20`" e "`logstash-7-2019.08.21`", podemos criar um *index pattern* com o padrão "`logstash-*`" que permitirá agregar estes dois índices.

Este componente da *Elastic Stack* disponibiliza também um conjunto de APIs bastante úteis na manipulação de objetos das quais se destacam a `Kibana Spaces API`, a `Kibana Role Management API`, a `Saved Objects API` e a `Dashboard Import API`.

O *Kibana* trata-se de uma aplicação desenvolvida em *JavaScript* com recurso à *framework React*. Para além da ferramenta em si, é disponibilizada uma biblioteca de User Interface (UI) denominada por `Elastic UI Framework`. Esta biblioteca tem como objetivo garantir que as contribuições para o *Kibana* seguem os mesmo padrões de desenho e nela podem-se encontrar vários componentes de UI como botões, modais, tabelas, entre muitos outros.

Para além dos *plugins* oferecidos pela *Elastic* é ainda possível criar *plugins* próprios através da execução de comandos criados para o efeito e adicioná-los ao *Kibana* através do comando de instalação.

Na Figura 26 encontra-se representado um exemplo desta *interface* gráfica constituída por uma *sidebar* de navegação, uma barra de pesquisa e filtragem, uma opção para alternar entre *spaces*, um espaço de utilizador e, neste caso, a página de *Dashboard*.

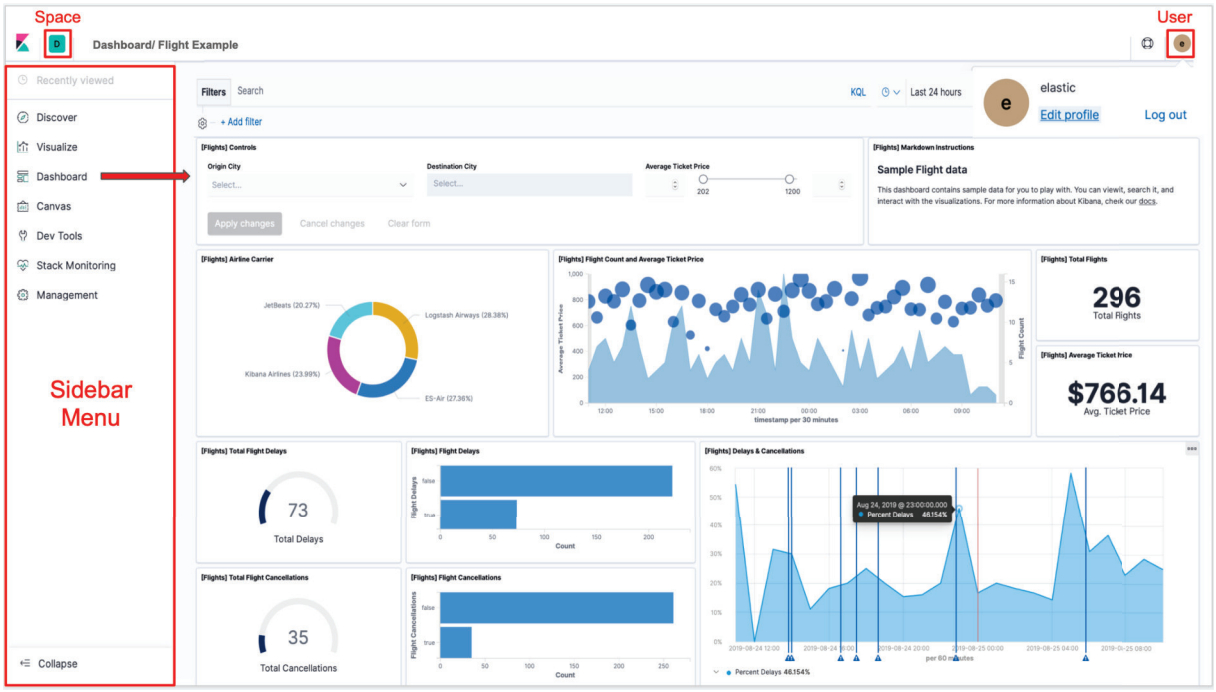


Figura 26: *Kibana Interface*

A página de *Discover* permite explorar os dados armazenados, possibilitando o acesso aos documentos guardados nos índices que correspondem ao *index pattern* escolhido. A informação é apresentada num histograma onde os documentos são distribuídos ao longo do período temporal definido. Além disso, é possível pesquisar e filtrar a informação e concluir acerca do número de documentos que satisfazem as condições impostas. Os documentos podem ser consultados no formato tabular ou JSON. As pesquisas efetuadas podem ser guardadas e utilizadas posteriormente em visualizações.

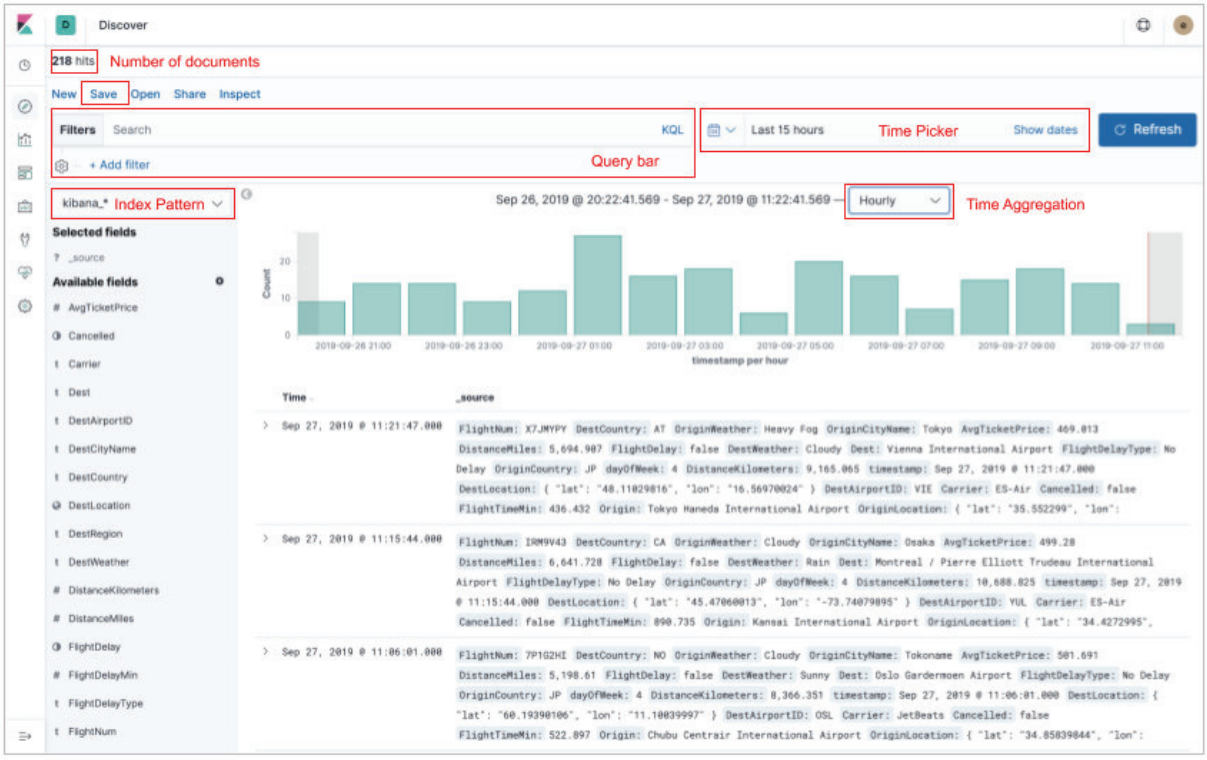
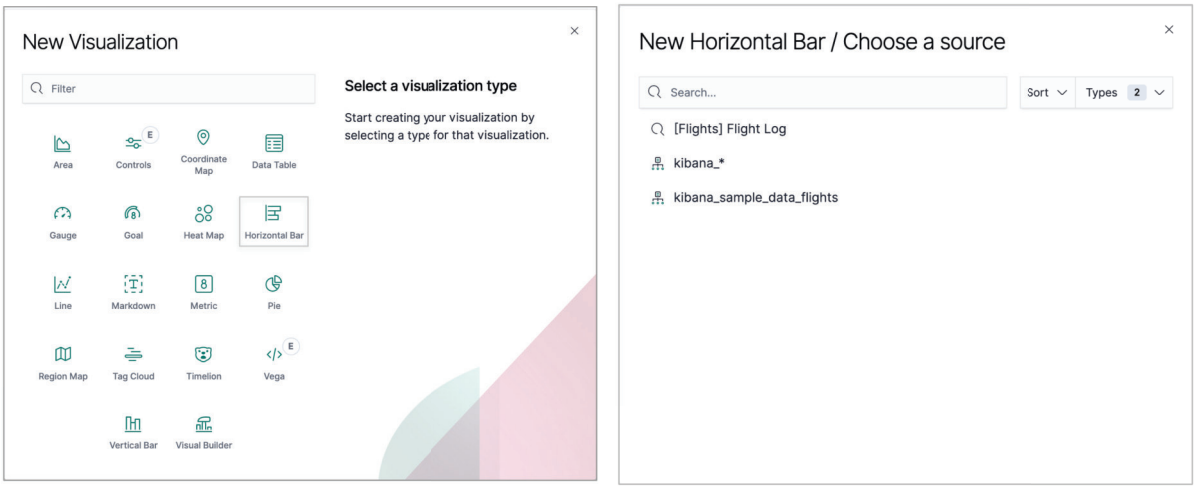


Figura 27: *Kibana Discover*

O *Visualize* permite a criação de visualizações tendo por base os índices armazenados no *ElasticSearch*. As visualizações do *Kibana* são baseadas em consultas ao *ElasticSearch* através de um conjunto de agregações para extrair e processar os dados, possibilitando a elaboração dos diferentes tipos de gráficos. A criação de visualizações deverá ser feita neste menu, começando pela escolha do tipo de visualização pretendida: *Basic charts*, *Data*, *Maps*, *Time Series*, entre outros (Figura 28a). Posto isto, deverá ser selecionada a fonte de dados sobre a qual se pretende elaborar a visualização: um *index pattern* ou uma pesquisa guardada no *Discover* (Figura 28b). No construtor da visualização proceder-se-á à definição das métricas e das agregações. Regra geral, este processo inicializa-se pela determinação da agregação dos dados no eixo das ordenadas (Y), podendo ser uma agregação simples – *count*, *average*, *sum*, *min*, *max*, *median* e *unique count* – ou uma agregação composta – *cumulative sum*, *average bucket*, *sum bucket*, *min bucket* e *max bucket*.

Escolhida a agregação de métricas pretendida, dever-se-á proceder à parametrização do eixo das abcissas (X), escolhendo como deverão ser distribuídos os valores recolhidos (*Buckets*): *date histogram*, *range*, *terms*, *filters*. Cada visualização tem ainda várias possibilidades de customização de *design*, nomeadamente em termos de palete de cores, tamanho de letra, entre outras (Figura 29).



(a) Opções de visualizações

(b) Fonte de dados

Figura 28: Criação de visualizações

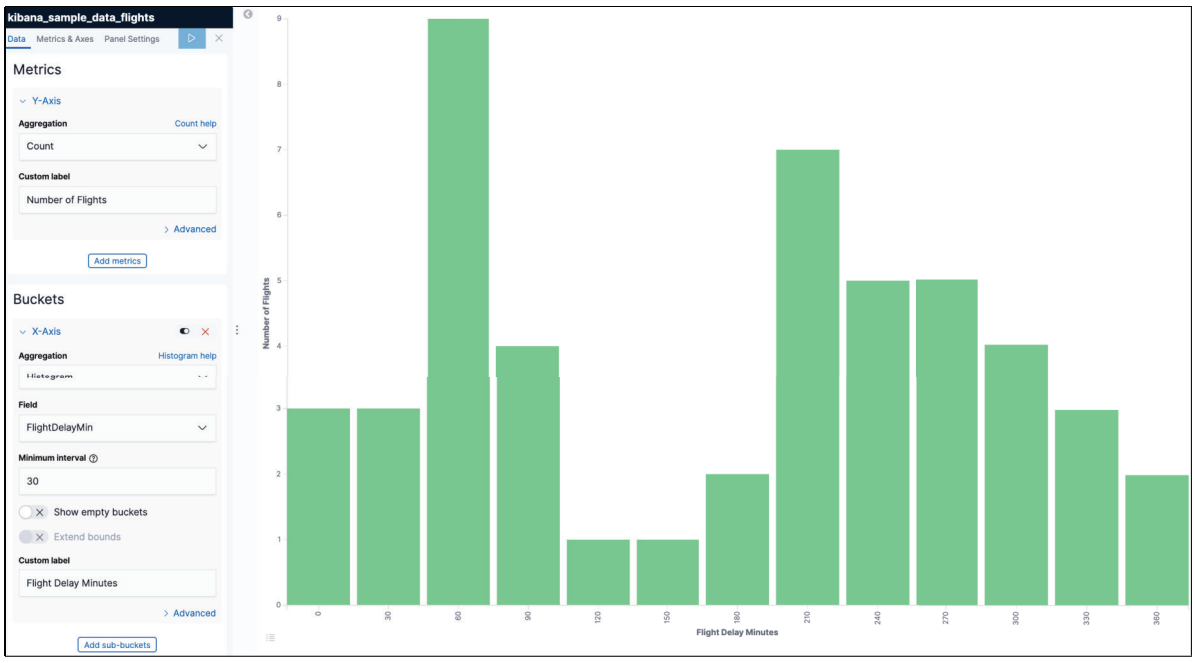


Figura 29: *Kibana Visualizion*

4

Sistema de otimização de custos de Cloud

Apresentada na Secção 3.3 a proposta de solução ao problema, serão expostas no presente capítulo as decisões tomadas ao logo do desenvolvimento do sistema de otimização de custos de *Cloud* bem como obstáculos enfrentados na implementação do mesmo.

4.1 Cloud Cost Analysis

A aplicação desenvolvida em *Flask* foi nomeada de *Cloud Cost Analysis*. Esta aplicação oferece como principais funcionalidades a obtenção de dados dos diferentes CSPs, algum tratamento dos mesmos e o redirecionamento destes para o *Logstash*.

Como principal desafio no *design* da aplicação destaca-se a modularidade do código, a escalabilidade da solução, a integração com diversos fornecedores de *cloud* com suporte a várias contas de utilização e o tratamento da informação de forma genérica.

Primeiramente definiu-se que todas as configurações deveriam estar isoladas da restante lógica aplicacional. Para tal, criou-se uma diretoria denominada `config` onde se encontram configurações aplicacionais bem como dados de acesso aos CSPs. Esta configuração é escrita no formato JSON num ficheiro designado por `config.json` que pode ser consultado no Anexo B.1. Por forma a garantir a correta escrita deste ficheiro foi criado um *schema* onde é possível consultar as configurações disponíveis e respetivos valores aceitáveis para as mesmas. Este *schema* foi intitulado de `config.schema` e pode ser consultado no Anexo B.2. De realçar ainda que toda as credenciais de acesso aos CSPs, sejam elas *Access Tokens*, *API Keys* ou *Secret Access Keys*, encontram-se armazenadas de forma cifrada em ficheiros `YAML` ou JSON para garantir a segurança e privacidade dos utilizadores.

4.1.1 Arquitetura da aplicação

Por forma a compreender melhor a solução desenvolvida, elaborou-se um diagrama da aplicação com recurso a UML, que pode ser consultado na Figura 42 do Anexo A.1.

O módulo `run.py` é responsável por executar a aplicação *Flask* criada no módulo `__init__.py` do *package* `app`, fornecendo um *web server* à escuta na porta 5000. No módulo `__init__.py`, para além da criação da aplicação, são ainda invocados métodos da classe `Main`, responsáveis pela inicialização da aplicação, e exportados os *endpoints* da API desenvolvida.

Conforme referido anteriormente, as configurações encontram-se definidas num ficheiro `config.json`. Estas configurações são lidas e tratadas por um módulo denominado `settings.py`, onde é validada a escrita do ficheiro com recurso ao `config.schema`, configurados os SDKs para acesso aos CSPs e ainda exportadas variáveis que permitem o acesso à *Elastic Stack*.

A classe `Main` é o *core* de toda a aplicação e encontra-se declarada no módulo `main.py`. Esta classe possui duas variáveis de instância denominadas `reporters` e `monitores`, que permitem a angariação de dados para cada conta dos CSPs definida nas configurações. O *bootstrap* da aplicação é executado num método designado por `init`, em que é feita a verificação da conectividade de todos os elementos da *Elastic Stack*, atualizada a informação relativa a custos e dados de utilização com recurso aos `reporters` e aos `monitores` e realizada a configuração inicial de elementos de análise no *Kibana*. A configuração do *Kibana* inclui a criação de um *space* para cada conta dos CSP e a inserção de visualizações e *dashboards* padrão desse fornecedor de *cloud*. Ainda nesta classe é criado um *cron* que permite executar em *background* funções para a atualização dos dados a analisar com um determinado intervalo temporal, tipicamente de 6 ou 12 horas.

Para facilitar o acesso e interação com a *Elastic Stack* foi criada a classe `Tools`. De entre os métodos disponibilizados por esta classe, salienta-se o `check_elk_connection` (que garante que a *stack* está operacional), o `post_data_logstash` (que permite o envio de informação para o *Logstash*), métodos relacionados com a gestão de índices no *ElasticSearch* e ainda funções para a criação de objetos no *Kibana*. Os elementos de análise do *Kibana* como *index patterns*, *visualizations* e *dashboards* são criados a partir da APIs disponibilizadas pelo mesmo e tendo por base os objetos definidos nos ficheiros `index-patterns.json`, `advance-settings.json`, `visualizations.json` e `dashboards.json` que se encontram armazenados na diretoria `resources`. Além das funções relacionadas com a *Elastic Stack* são ainda facultadas outras de carácter mais utilitário como a `decompress_file`, bastante útil para lidar com os diferentes tipos de compressão dos relatórios.

Reporter

A obtenção de informação de custos de operação é da responsabilidade dos `reporters`. Esta variável de instância declarada na classe `Main` trata-se de um dicionário que a cada conta de utilização faz corresponder uma instância da classe `Reporter`. Esta classe encontra-se implementada no módulo `reporter.py` e disponibiliza um conjunto de métodos que permitem garantir a consistência de dados bem como algum tratamento dos mesmos, nomeadamente na transformação de campos e conversão de tipos. Este tratamento de dados é conseguido graças à definição de dois ficheiros *YAML* definidos na directoria `resources` denominados `keyTransformer.yaml` e `mapProperties.yaml`, cujo exemplos podem ser consultados, respectivamente, no Anexo B.3 e B.4. Para cada CSP são criados estes ficheiros, permitindo deste modo, estender a lógica desenvolvida a novos CSPs.

O construtor desta classe recebe as configurações da conta do fornecedor de *cloud* bem como ficheiros referidos anteriormente. Por forma a adquirir a informação do CSP, é criado um conector através do método `getConnector` disponibilizado pelo *middleware*. Este conetor expõe métodos para comunicar com o CSP correspondente através do *middleware* e assim isolar essa lógica do `Reporter`. A sua descrição e caracterização encontra-se mais à frente nesta dissertação. Ainda no construtor é invocado um método denominado por `check_es_template`. Neste método é verificado se existem dados armazenados no *ElasticSearch* para a conta em questão. Caso tal não se verifique, é criado um *index template* no *ElasticSearch* ao qual vão estar associados os dados a armazenar. Um *index template* permite definir um *template* que será automaticamente aplicado quando novos índices forem criados, sendo nele estabelecido *settings* como o número de *shards* ou réplicas desejadas, *mappings* onde são definidas as propriedades e os seus tipos e ainda os padrões que determinam quando este *template* deve ser aplicado. Os *index templates* criados pela aplicação têm por base um conjunto de *settings* e *mappings* que se encontram definidos no ficheiro `es_template.json`, armazenado na diretoria `resources`, sendo depois estendido de acordo com o *index template* que se pretende criar. A nomenclatura estabelecida para o nome do *index template* relativo a custos de operação contêm o prefixo "*billing*" seguido do nome da conta em análise. Um exemplo de *index template* pode ser consultado no Anexo B.5.

O principal método desta classe e que é responsável para obtenção de dados de custos de operação sob o formato de relatórios denomina-se `check_reports`. Este método começa por invocar o método `get_reports_metadata` para obter a informação relativa aos *reports*, utilizando para isso uma função do *middleware* denominada de `get_billing_objects`. Esta função retorna um dicionário em que a chave é o ano e mês dos dados – sobre o formato "`YYYY-MM`" – e o valor é um *array* de objetos a adquirir referentes a esse período temporal. A informação é confrontada com a que já se encontra armazenada no *ElasticSearch* e definida apenas aquela que é necessária obter.

Para adquirir a informação é invocado o método `download_report`. Este método acede novamente ao conector chamando a função `download_billing_object` que recebe como parâmetro o objecto

a transferir. Para os ficheiros CSV descarregados são aplicadas operações de transformação de dados com recurso a uma biblioteca de *Python* denominada `Pandas` [87] e tendo por base os ficheiros *YAML* referidos anteriormente. O objectivo deste tratamento de dados é poder não só normalizar os dados mas também modificar os campos de forma a serem compatíveis com o modelo genérico de dados definido na Figura 43 do Anexo A.1.

Tendo em conta a informação que cada CSP fornece, foi possível criar este modelo de dados genérico em que são definidos os principais elementos que os custos de operação devem possuir. De forma geral, verificou-se que cada conta de utilização (`Account`) possui um identificador único e um nome que a caracteriza e tem a si associadas diversas faturas (`Bill`). Estas faturas tem um *ID* único, o período ao qual se refere (`BillingStartDate` e `BillingEndDate`), qual a moeda utilizada e ainda a conta e entidade responsável pelo pagamento da mesma. Uma fatura é constituída por diversas linhas de fatura (`LineItem`), sendo cada uma destas linhas caracterizada com um *ID* e uma descrição, um período de utilização (`UsageStartDate` e `UsageEndDate`), a quantidade, custos e taxas associadas e o serviço/produto a que se refere (`Product`). Produtos como instâncias computacionais ou serviços como transferência de dados são descritos genericamente como `Product` e são identificáveis por um *ID* único, um nome, o tipo e família a que pertencem e, opcionalmente, uma localização. Estes produtos podem possuir `Tags` para auxiliar na rastreabilidade de custos, sendo estas descritas por um dado nome e tipo a que pertencem. Um produto pode ser adquirido através de uma reserva (*Reservation*), sendo definida por um `SubscriptionID`, um nome, um período de reserva e o valor pago no ato da reserva. Uma linha de fatura tem ainda uma política de preços associada (`Pricing`) no qual são descritos elementos como taxas gerais, termos de aquisição, opções de pagamentos dos produtos, entre outras.

Dada a diversidade de CSPs e formas de armazenamento de dados implementadas pelos mesmos, foi importante estudar os vários cenários possíveis. O primeiro cenário é aquele em que a informação é guardada em ficheiros organizados por dia e em que estes são criados e mantêm-se inalterados. Neste cenário, a informação obtida não necessita de mais nenhum tratamento pela aplicação. Outro cenário que ocorre é aquele em que é criado, pelo CSP, um conjunto de ficheiros referente ao mês em questão e em que esses ficheiros são reescritos a cada atualização, até encerrar a faturação. Este outro cenário envolveu a necessidade de tratar os dados de forma diferente, uma vez que não podiam ser armazenados tal como estão, pois iriam originar registos duplicados. Dado que nessas reescritas ocorre a eliminação de algumas linhas do ficheiro e a criação ou atualização das restantes, foram ponderadas as seguintes hipóteses: a primeira consiste em criar um ficheiro com o prefixo "delete" que contem as linhas a remover enquanto a segunda passa pela eliminação do *index* do *ElasticSearch* que contém a informação relativa ao mês em tratamento e à reindexação da mesma. A primeira hipótese implica que seja possível identificar o registo a apagar de forma inequívoca, contudo evita que o utilizador fique temporariamente sem dados, o que se verifica na segunda opção. Dada a diversidade de CSPs e o objetivo de manter a generalidade

da implementação da solução considerou-se que esta deveria suportar as duas alternativas sendo da responsabilidade do conector identificar qual deverá ser aplicada.

Tratada a informação relativa a custos de operação, esta é enviada para ser processada pelo *Logstash* e posteriormente armazenada no *ElasticSearch* através do método `post_data_logstash` .

Por forma a compreender mellhor a lógica criada para a angariação dos dados e os componentes da arquitetura envolvidos na mesma, decidiu-se criar um diagrama de sequência que pode ser consultado na Figura 44 do Anexo A.1.

Monitor

Os *monitores* são responsáveis por obter informação dos CSPs relativa à utilização de recursos, possuindo uma lógica semelhante à dos *reporters*. A variável de instância declarada na classe `Main` (`monitores`) consiste num dicionário que a cada conta de utilização faz corresponder uma instância da classe `Monitor`, implementada no módulo `monitor.py`. O construtor desta classe recebe as configurações da conta do CSP do qual se pretende obter a informação e instancia um conector através do método `getConnector`.

Dos métodos definidos no `Monitor` salientam-se o `check_es_template` e o `check_usage`. O primeiro método verifica se existe um *index template* no *ElasticSearch* relativo a dados de utilização para a conta fornecida, criando um com o prefixo "usage" seguido do nome da conta, caso tal não se verifique. O segundo método permite angariar dados de utilização dos recursos, começando por verificar se existem dados armazenados no *ElasticSearch*. Se não houver qualquer informação, serão obtidos dados desde o início do dia atual. Caso contrário, serão angariados dados a partir do último instante recolhido.

A obtenção da informação proveniente dos CSPs é feita com recurso ao conector e ao *middleware* implementado através da invocação do método `get_usage`. Este método recebe como parâmetro a data de início, data de fim e o período com que essa informação deve ser angariada, estando estabelecido por omissão a data e hora atual como data de fim e o valor de uma hora para o período de tempo, em milissegundos. Os dados angariados são enviados para o *Logstash* no formato JSON por Transmission Control Protocol (TCP) [88].

Middleware

O *midddleware* responsável por abstrair a lógica de interação com os fornecedores de *cloud* encontra-se implementado no módulo `cloudProviderConnector`. A sua implementação consistiu na definição de uma *interface* com recurso às Abstract Base Classes (ABCs) [89], denominada por `CloudProviderConnector`, e num conjunto de métodos abstratos: `get_billing_objects`, `download_billing` e `get_usage`.

Cada CSP implementa esta interface e os métodos abstratos nela definidos, criando assim um conector ao respetivo fornecedor de *cloud*: `AWSConnector`, `AzureConnector` e `GCPConnector`. Deste modo é possível estender a lógica a outros CSPs que a solução possa vir a suportar no futuro. Por forma a facilitar a instanciação dos conectores, este módulo fornece ainda um método denominado de `getConnector` que recebe como parâmetro o CSP e as configurações da conta a utilizar e retorna o conector pretendido.

O conector à AWS foi programado com o auxílio do SDK para *Python* denominado `Boto3` [90]. O recurso ao SDK facilita a criação, configuração e gestão de serviços da AWS bem como o acesso aos recursos do fornecedor de *cloud*. Para o problema em estudo, o `Boto3` simplificou a obtenção dos *reports* armazenados nos *buckets* S3 e o uso das APIs do *Cost Explorer* e do *CloudWatch*. A sua configuração é bastante simples uma vez que utiliza os dados de acesso definidos nos ficheiros de credenciais referidos anteriormente, evitando a necessidade de efetuar autenticação nos serviços ou criar *tokens* para as APIs.

À semelhança da AWS, o conector para o *Microsoft Azure* também foi desenvolvido com auxílio de bibliotecas de *Python*. Dado que só era necessário aceder a determinados serviços do *Azure*, apenas foram utilizadas as bibliotecas que facilitam o acesso aos mesmos, nomeadamente [91]:

- `azure.common.client_factory`: disponibiliza métodos que permitem a criação de clientes para acesso aos serviços, como o `get_client_from_auth_file`, tendo por base um ficheiro de configuração definido na variável `AZURE_AUTH_LOCATION`;
- `azure.storage.blob`: permite o acesso a elementos de *storage* como os *blobs*, local onde se encontram armazenados os ficheiros CSV de faturação;
- `azure.mgmt.resource`: biblioteca usada para a gestão dos recursos do *Azure* (`ResourceManagementClient`), útil para listar os recursos disponíveis e sobre os quais se pretende obter informações como dados de utilização;
- `azure.mgmt.monitor`: fornece um cliente para aceder ao *Azure Monitor* denominado por `MonitorManagementClient`, permitindo obter informação relativa a dados de utilização de recursos, evitando a complexidade de utilização da API do *Monitor*.

Para aceder ao GCP foi desenvolvido um conector com recurso a uma biblioteca de *Python* denominada por `google.cloud` [92]. Esta biblioteca permite instanciar clientes para diversos serviços, dos quais se destacam o `storage`, o `monitoring_v3` e o `resource_manager`. Através do cliente de *storage* é possível aceder ao *bucket* onde estão armazenados os relatórios de faturação, listar os *blobs* existentes – objetos onde estão esses ficheiros – e transferir os pretendidos. Para obter os dados de utilização é necessário recorrer ao *resource_manager* para listar os projetos existentes na *cloud* e, através do serviço de *monitoring*, listar as métricas existentes para cada projeto. Tendo esta informação é assim possível obter os valores para essas métricas no intervalo temporal pretendido utilizando novamente o cliente do *monitoring* que interage com a API do *Stackdriver*.

API

Para permitir que outros sistemas interajam com a aplicação foi criada uma REST API. Dado que esta aplicação utiliza a *framework* de *Python Flask*, decidiu-se tirar partido de um recurso amplamente utilizado com esta *framework* denominado *Blueprint* [93]. O conceito de *blueprints* permite construir aplicações modulares e é utilizado no desenvolvimento de APIs uma vez que possibilita a sua criação num módulo isolado da restante lógica aplicacional.

Deste modo, foi criado um *package* chamado `api`, definido o módulo e os *endpoints* da API. A criação do *Blueprint* foi realizada no `__init__.py` deste *package* e importados os *endpoints* descritos no módulo `endpoints.py`.

Os *endpoints* criados tratam-se essencialmente de métodos HTTP `GET`, `POST` e `PUT` que permitem manipular a informação da aplicação, nomeadamente obter a informação de faturação que está indexada, atualizar os dados relativos a custos de operação e utilização de recursos, reindexar informação de faturação ou recriar visualizações e *dashboards* padrão do *Kibana*. A título de exemplo, apresenta-se a seguir o *endpoint* criado para angariar novos dados de utilização. Este *endpoint* trata-se de um método `POST` para a rota "updateUsage" que pode receber no corpo do pedido, no formato JSON, as contas de utilização a atualizar.

```
@bp.route('/updateUsage', methods=['POST'])
def update_usageData():
    accounts = []
    if request.is_json:
        body = request.get_json()
        if 'accounts' in body:
            accounts = body["accounts"]

    # Verify if account names provided exists
    try:
        tools.check_accountsNames(accounts)
    except ValueError as error:
        return jsonify(str(error)), status.HTTP_400_BAD_REQUEST

    # Execute function
    t = Thread(target=update_usage, args=(accounts,))
    t.start()

    # Immediately return a 200 response to the caller
    return "Usage Data fetched!", status.HTTP_200_OK
```

4.2 Logstash

Conforme referido na Secção 3.3, o *Logstash* é o componente da *Elastic Stack* responsável por tratar a informação recebida da aplicação e enviá-la para o *Elasticsearch*. De seguida, será descrito como foi configurado e implementado este componente na solução proposta.

4.2.1 Configuração do Logstash

A configuração do *Logstash* baseia-se num conjunto de ficheiros de definições onde são especificadas as opções de controlo de execução. Destes ficheiros destacam-se relevantes para a orquestração do componente o `logstash.yml` e o `pipelines.yml`.

O ficheiro `logstash.yaml` é útil para indicar um conjunto de propriedades globais ao *Logstash* como dados de acesso ao *ElasticSearch* e definições de *pipelines* a executar. No excerto de código seguinte é possível observar a definição de algumas destas propriedades.

```
# ------------ Pipeline Settings --------------
# The ID of the pipeline.
pipeline.id: main
# Number of workers that will execute the stage of the pipeline.
pipeline.workers: 1
# How many events to retrieve from inputs before sending to filters+workers.
pipeline.batch.size: 125

# ------------ Metrics Settings --------------
# Bind address and port for the metrics REST endpoint
http.host: "0.0.0.0"
http.port: 9600-9700

# ------------Debugging Settings --------------
log.level: info
path.logs: /var/log/logstash

# ------------ X-Pack Settings --------------
xpack.monitoring.enabled: true
xpack.monitoring.elasticsearch.username: ${ES_USERNAME}
xpack.monitoring.elasticsearch.password: ${ES_PASSWORD}
xpack.monitoring.elasticsearch.hosts: [ "http://elasticsearch:9200" ]
```

O ficheiro `pipelines.yml` contêm as instruções necessárias para a execução de múltiplas *pipelines*. Para cada *pipeline* é necessário indicar um identificador único e o caminho para o respetivo ficheiro de configuração da *pipeline*, podendo ainda ser especificadas propriedades a aplicar exclusivamente à *pipeline*, evitando que sejam atribuídas as propriedades descritas no `logstash.yaml`. De seguida, apresenta-se um exemplo de um destes ficheiros de configuração:

```
# List of pipelines to be loaded by Logstash
- pipeline.id: billing_pipeline
  path.config: "/usr/share/logstash/pipeline/billing_pipeline.conf"
  queue.type: persisted

- pipeline.id: usage_pipeline
  path.config: "/usr/share/logstash/pipeline/usage_pipeline.conf"
  pipeline.workers: 3
```

A utilização de várias *pipelines* é útil quando se possuí um fluxo de eventos que não partilham os mesmos *inputs/filters* e *outputs*, tendo de ser separados com recurso a *tags* e estruturas condicionais. Considerando a proposta de solução verifica-se que a informação é enviada para o *Logstash* sobre o formato de ficheiros CSV ou no formato JSON (através de TCP) e as transformações a aplicar aos dados são bastantes distintas. Deste modo, considerou-se a utilização de duas *pipelines* para separar o mecanismo de ETL a aplicar à informação: `billing_pipeline` e `usage_pipeline`.

4.2.2 Pipelines

A *pipeline* responsável pelo processamento dos ficheiros de CSV com os custos de operação é denominada de `billing_pipeline` e pode ser consultada no Anexo B.7.

Para a análise dos ficheiros CSV ponderou-se a utilização de um *input plugin* que permite a leitura de ficheiros e a aplicação de um *filter plugin* para processamento de CSV. Embora esta combinação de *inputs* fosse viável, após a sua implementação verificou-se um conjunto de fatores que impossibilitaram a sua utilização na proposta de solução, nomeadamente o facto de no *filter plugin* ser necessário especificar as colunas do ficheiro CSV. Dada a diversidade de ficheiros, ter-se-ia que recorrer a *tags* e a estruturas condicionais para separar o fluxo de eventos do *input* e definir os nomes das colunas corretos para cada ficheiro de origem. Decidiu-se então desenvolver um *input plugin* que tivesse por base o *file input plugin* e ao qual fosse adicionado a lógica do *csv filter plugin*.

Para o desenvolvimento do `CSV Input Plugin`, o *Logstash* disponibiliza um comando que facilita a criação do mesmo, permitindo criar um *plugin* totalmente funcional com a estrutura apresentada de seguida:

```
bin/logstash-plugin generate --type input --name logstash-input-csv
```

```
logstash-input-csv
├── Gemfile
├── LICENSE
├── lib
│   └── logstash
│       └── inputs
│           └── csv.rb
├── logstash-input-csv.gemspec
└── spec
    └── inputs
        └── csv_spec.rb
```

Conforme é possível observar, a estrutura do *plugin* é constituída por diversos elementos. O ficheiro criado em `lib/logstash/inputs` contém o código do *plugin* a desenvolver e o ficheiro localizado em `spec/inputs` possui testes para esse *plugin*. As dependências e configurações do *plugin* encontram-se nos ficheiros `Gemfile` e `.gemspec`. Para além destes, contém ainda ficheiros relativos a licenças e documentação. O código do *input plugin* gerado possui uma estrutura bem definida, como se constata no exemplo de código a seguir:

```ruby
# encoding: utf-8
require "logstash/inputs/file"
require "logstash/namespace"

class LogStash::Inputs::Example < LogStash::Inputs::Base
    config_name "example"
    config :message, :validate => :string, :default => "Hello World!"

    public
    def register
        @host = Socket.gethostname
    end # def register

    def run(queue)
        Stud.interval(@interval) do
          event = LogStash::Event.new("message" => @message, "host" => @host)
          decorate(event)
        end # loop
    end # def run
end # class LogStash::Inputs::Example
```

Para a escrita do *plugin* é necessário começar por definir o *enconding* bem como as dependências a importar. No corpo do *plugin* é declarada a classe a desenvolver que estende a classe base dos *inputs* ou outra compatível. Dentro desta classe é configurado o nome do *plugin* a utilizar nas *pipelines* bem como os respetivos parâmetros. Os *inputs* de *Logstash* implementam 2 métodos principais denominados de *register* e *run*, sendo o primeiro um método de inicialização e o segundo responsável por transformar os fluxos de dados em eventos a serem processados pelos *filters*.

No Anexo B.6 é possível consultar o código desenvolvido para o *input plugin* proposto, destacando-se os seguintes aspetos:

- A classe de base ao *plugin* é a utilizada para ficheiros – `LogStash::Inputs::File`;
- O nome a utilizar na secção de *inputs* das *pipelines* é "`csv`";
- Nos parâmetros de configuração do *input* podem ser definidas as colunas do CSV (`collumns`) ou se se pretende utilizar a primeira linha para a definição das colunas (`header`), qual o elemento de separação das colunas (`separator`), o caractere usado nos campos do ficheiro (`quote_char`) e uma variável que estabelece a inclusão de valores vazios de uma coluna no resultado final (`skip_empty_columns`);
- A inicialização da variável `fileColumns` no método `register`;
- A utilização da biblioteca de *Ruby* para `csv` que permitiu a leitura do ficheiro e construção dos eventos a serem processados pelos *filters*;

Para utilizar este *input* nas *pipelines* é necessário construir o ficheiro `gem` a instalar. Para isso, deverão ser executados os seguintes comandos:

```
bundle install
gem build logstash-input-csv.gemspec
```

O primeiro comando é responsável pela instalação das dependências referidas no `Gemfile`, originando um ficheiro `Gemfile.lock`. O segundo comando permite construir o ficheiro `logstash-input-csv-0.1.2.gem` em que o nome e a versão se encontram especificadas no `.gemspec`. Este *plugin* pode ser instalado através da execução do comando: `logstash-plugin install logstash-input-csv-0.1.2.gem`

Deste modo, a `billing_pipeline` usa como *input* o *plugin* desenvolvido com opções para identificar automaticamente as colunas do ficheiro, não considerar as colunas que não possuam valores, começar a leitura do ficheiro no início e apagá-lo quando terminar. Dada a necessidade de implementar o mecanismo referido na SubSecção 4.1.1 relativo à atualização da informação, separou-se estes *inputs* por forma a ler ficheiros de determinados *paths* e atribuir as respetivas *tags*: *csv* e *delete*.

Após serem lidos os ficheiros, os eventos são processados no `filter`. Nesta etapa é criado dinamicamente o nome do *index* onde os dados serão armazenados no *Logstash* tendo em conta o formato `billing-account-year-month` por forma a serem mapeados no *index_template* referido anteriormente. Para tal recorreu-se a um *filter plugin* denominado de `grok` que permite fazer o *parse* de informação e realizar a correspondência de campos.

Nesta etapa é criado um *document_id* a partir da concatenação de um conjunto de variáveis que permitem identificar univocamente a linha lida como o *id* da linha e o intervalo de tempo a que corresponde os custos registados nessa linha. Para a sua criação utilizou-se um *plugin* designado por `mutate` que permite a manipulação de campos de um evento como a criação, atualização e remoção dos mesmos.

Aos eventos são ainda aplicadas operações de tratamento de dados, nomeadamente a transformação de campos temporais para formatos *standard* através do *date plugin*, a conversão de localizações em *geopoints* utilizando o *translate plugin* e a separação de campos agregados através do `json` *plugin* e do *ruby plugin*. O *ruby plugin* permite aceder ao evento e aplicar as operações que se desejar através da escrita de código *Ruby*.

Por fim, a informação processada é enviada para o *output*. Nesta secção é verificada a existência do nome do *index*, do *document_id* e qual a ação a aplicar ao registo: *delete* ou *update*. Assim sendo, a informação é enviada para o *ElasticSearch* e aplicada a ação pretendida com as configurações especificadas. De salientar que os dados de acesso ao *ElasticSearch* encontram-se armazenados em variáveis de ambiente: `ES_USERNAME` e `ES_PASSWORD`. Estes dados são definidos num ficheiro `.env`, por motivos de segurança.

A *pipeline* encarregue do processamento dos dados de utilização de recursos é designada de `usage_pipeline` e pode ser consultada no Anexo B.8. Esta pipeline foi mais simples de definir uma vez que os *plugins* disponibilizados pela *Elastic* permitiam a sua implementação. Deste modo, o *input* é constituído por um *plugin* TCP que permite receber dados no formato JSON na porta 5140 (ou na que estiver definida na variável `TCP_PORT` no ficheiro `.env`) e transformar esses dados em eventos a serem processados. Na secção do *filter* são aplicadas operações semelhantes às aplicadas na outra *pipeline*, nomeadamente a criação do nome do *index* no formato `usage-account-year-month` e a transformação e eliminação de campos. Por fim, os eventos são enviados para o *output* que trata de os reencaminhar para o *ElasticSearch* para serem indexados.

4.3 ElasticSearch

O *ElasticSearch* é o componente da *ElasticStack* responsável pelo armazenamento dos dados e pela análise e pesquisa dos mesmos. Os índices onde é armazenada a informação encontram-se na diretoria `$ES_HOME/data`.

A configuração do *ElasticSearch* é bastante simples uma vez que este já possui bastantes definições padrão, sendo apenas necessário definir os seguintes ficheiros de configuração na diretoria `$ES_HOME/config`:

- `elasticsearch.yml`: ficheiro com as configurações gerais do *ElasticSearch*;
- `jvm.options`: ficheiro com as opções da Java Virtual Machine (JVM) a fornecer ao *ElasticSearch*;
- `log4j2.properties`: ficheiro com as propriedades de *logging* a serem mantidas pelo *ElasticSearch*.

No ficheiro `jvm.options` apenas foi definido o valor da *heap size* da JVM: `-Xmx1G -Xms1G`. As configurações de *logging* não foram alteradas, mantendo-se as que estavam por omissão. O ficheiro `elasticsearch.yml` pode ser consultado no Anexo B.9 e nele encontram-se as configurações básicas relativas ao *cluster* e *nodes* do *ElasticSearch*, a diretoria onde armazenar os dados e os *logs* do mesmo e propriedades de memória, rede, entre outras. Ainda neste ficheiro é definido o tipo de licença a utilizar na *Elastic Stack* bem como ativadas as extensões de segurança e monitorização.

À semelhança de outros ficheiros de configuração apresentados na Secção 4.2, nestes ficheiros também é possível utilizar as variáveis de ambiente definidas no ficheiro `.env`, como é o caso do `ES_NETWORK_HOST`.

De acordo com as definições especificadas nos ficheiros de configuração, o *ElasticSearch* cria por omissão um conjunto de índices e *index templates* para suportar as funcionalidades ativadas, nomeadamente: `.monitoring`, `.security` e `.kibana`. O índice `.monitoring` armazena dados dinamicamente por componente da *Elastic Stack* e por dia, por exemplo ".monitoring-kibana-2019.08.26". O `.security` é onde estão armazenadas informações de segurança como credenciais e políticas de acesso. Por fim o índice `.kibana` contêm todas as definições relativas ao *Kibana* como configurações de gestão, *index patterns*, *spaces*, visualizações, *dashboards*, entre outras.

Conforme referido anteriormente, este componente da *Elastic Stack* disponibiliza um conjunto de APIs REST que podem ser utilizadas para aceder ou configurar recursos do *ElasticSearch* das quais se destacam a `cat API`, a `Document API`, a *Index API* e a `Search API`. Estas APIs permitem, entre muitas outras funcionalidades, consultar o estado dos diversos elementos do *ElasticSearch* – como *clusters*, *nodes*, *indexes* e *templates* –, interagir com diversos componentes dos índices – nomeadamente *index settings*, *mappings* e *templates* – e executar *queries* sobre os dados armazenados.

De seguida, apresentam-se alguns exemplos de *endpoints* das APIs referidas. Na execução destes pedidos abstraiu-se os dados de acesso ao *ElasticSearch* através das variáveis `ES_HOST` e `ES_PORT`. Caso o *plugin* de segurança esteja ativo, deverá ainda ser passado nos pedidos as credenciais de acesso ao mesmo.

```
$ curl 'ES_HOST:ES_PORT/_cat/health?v&h=timestamp  cluster,status,node.total,shards'
timestamp   cluster         status  node.total   shards
10:20:07    docker-cluster  green       1          6

$ curl 'ES_HOST:ES_PORT/_cat/indices?v&h=health,i  dex,pri,rep,docs.count,store.size'
health  index                              pri   rep  docs.count  store.size
green   .monitoring-logstash-7-2019.08.26   1    0        135248      11.2mb
green   .monitoring-es-7-2019.08.26         1    0        145332      75.6mb
green   .monitoring-kibana-7-2019.08.26     1    0          4896         1mb
green   .security-7                         1    0            38      63.9kb
green   .kibana                             1    0           229     248.7kb
green   billing-account-2019-08            1    0       1096271     877.9mb

$ curl 'ES_HOST:ES_PORT/_template/.logstash-management?pretty'
{
  ".logstash-management" : {
    "index_patterns" : [".logstash"],
    "settings" : {
      "index" : { "number_of_shards" : "1", "codec" : "best_compression"  }
    },
    "mappings" : {
      "dynamic" : "strict",
      "properties" : {
        "description" : { "type" : "text" },
        "last_modified" : { "type" : "date" },
        "pipeline" : { "type" : "text" },
        "pipeline_settings" : { "dynamic" : false, "type" : "object" }
      }
    }
  }
}
```

4.4 Kibana

O *Kibana* é o último componente da *Elastic Stack* e é responsável pela camada de visualização e análise dos dados armazenados no *ElasticSearch*.

À semelhança dos outros elementos da *stack*, a sua configuração é feita através de um ficheiro *YAML* denominado por `kibana.yml` que pode ser consultado no Anexo B.10. Neste ficheiro são definidas configurações relativas à orquestração do serviço, fornecidas as credenciais de acesso ao *ElasticSearch* e especificado o índice onde armazenar os dados, sendo por omissão atribuído o nome `.kibana`.

Conforme referido na Subsecção 4.1.1, a aplicação criada fornece uma API com métodos para gerir a angariação de dados e elementos do *ElasticSearch* e do *Kibana*. Embora estes métodos possam ser invocados por linha de comando, decidiu-se que tal deveria ser possível de executar através da interface gráfica. Considerando a extensibilidade que o *Kibana* fornece, decidiu-se criar um *plugin* que permitisse alcançar esse objetivo.

4.4.1 CCA Management

O *plugin* criado para interagir com a aplicação *Python* é denominado de `Cloud Cost Analysis (CCA) Management` e possui uma *interface* gráfica bastante simples. Para a sua criação recorreu-se a uma ferramenta disponibilizada pelo *Kibana* designada por `Kibana Plugin Generator` que permite criar um *plugin* totalmente funcional. Para a utilização desta ferramenta é necessário obter o código fonte do *Kibana*, idealmente do repositório público do `GitHub` e com recurso a `Git`, e proceder a execução dos seguintes comandos:

```
yarn kbn bootstrap
node scripts/generate_plugin cca_management
```

O primeiro comando instala as dependências do *Kibana* e de todos os sub-projetos/*plugins* que possui. O segundo comando cria um *plugin* com uma estrutura base pronto a ser trabalhado:

```
kibana-extra
└── cca_management
    ├── public
    ├── server
    ├── node_modules
    ├── index.js
    ├── package.json
    ├── yarn.lock
    └── .eslintrc
```

No `package.json` encontra-se definido um conjunto de variáveis essenciais ao desenvolvimento, nomeadamente o nome, versão e descrição do *plugin*, a versão do *Kibana* que é suportada, os *scripts* fornecidos e as dependências a serem instaladas. No `index.js` é criada uma instância da classe `Plugin` e são definidos os pontos de entrada e configurações do servidor que dá suporte ao *plugin*.

Na diretoria `server` encontra-se um ficheiro denominado por `cca_api.js` onde são estabelecidas as rotas da aplicação e os redirecionamentos para a API da *Cloud Cost Analysis*. O desenvolvimento da UI ocorre na diretoria `public` onde se encontra a `app.js` e os componentes que esta utiliza. Todo o estilo da aplicação é feito através da escrita de ficheiros SCSS e com recurso aos componentes da *Elastic UI Framework*, como `Forms`, `Buttons`, `Modals`, `Spacers`, entre outros.

Para o desenvolvimento do *plugin* é aconselhável a utilização do comando `yarn` em conjunto com os *scripts* definidos no `package.json`, nomeadamente o `yarn start`, `yarn test` e `yarn build`. Este último permite construir um ficheiro denominado por `cca_management-VERSION.zip` que pode ser instalado no *Kibana* através do comando:

```
bin/kibana-plugin install file:///plugins/cca_management-VERSION.zip
```

Após ser instalado, o *plugin* aparece junto dos restantes na barra lateral de navegação e está pronto a ser utilizado. O resultado final deste *plugin* pode ser consultado na Figura 30.

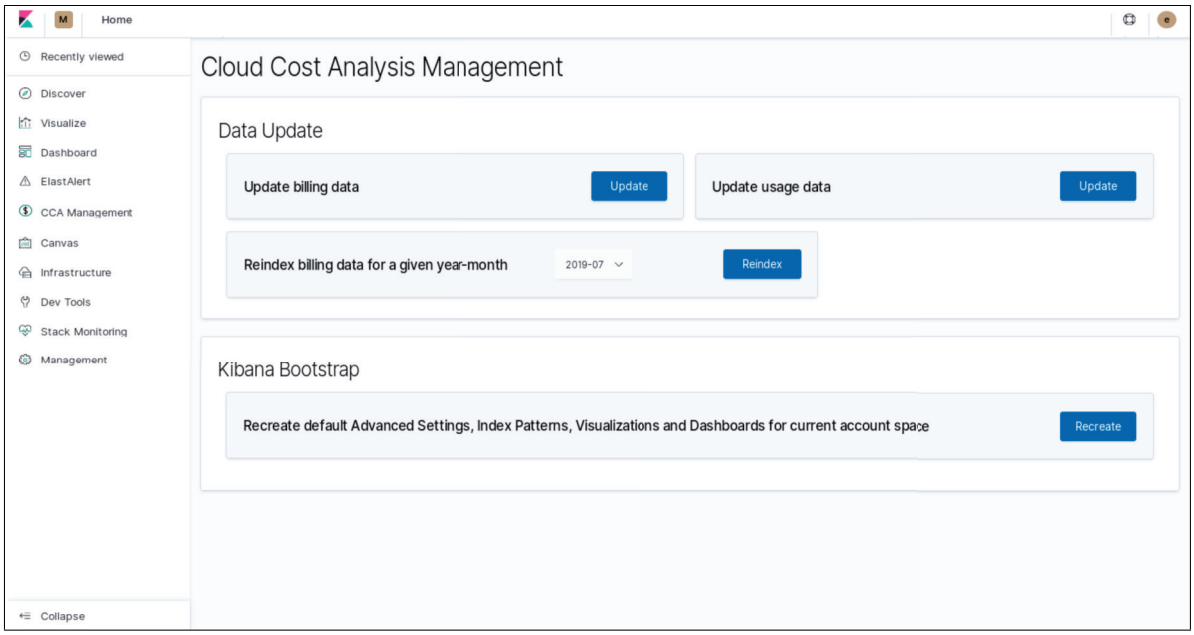


Figura 30: *Cloud Cost Analysis Management Plugin*

4.4.2 Customização da Interface

Para além da criação do *plugin* que permite estender as funcionalidades já fornecidas, foram ainda realizadas algumas modificações no código fonte para customizar a interface do *Kibana*. Estas modificações basearam-se em alterações de UI essencialmente na página de *login* e na de seleção do *space* ao qual se pretende aceder.

Em ambas as páginas *Web* foram feitas modificações em elementos textuais, na palete de cores utilizada e em elementos gráficos como logótipos e *wallpapers*. Estas alterações foram realizadas para que a interface se assemelhasse à imagem da empresa, tendo sido escolhidas as cores e o logótipo da mesma. O resultado final desta customização pode ser consultado na Figura 45 do Anexo A.2.

Para poder aplicar estas customizações ao *Kibana* foi criado, com auxílio do comando `diff`, um ficheiro com as alterações efetuadas e, através do comando `patch`, efetuadas essas modificações. De seguida, apenas foi necessário construir o componente utilizando para isso o comando `yarn build` com as opções pretendidas.

4.5 ElastAlert

O controlo e gestão de custos através de um sistema de alarmística constitui um dos requisitos da ferramenta a desenvolver, conforme referido na Secção 3.3. Embora a *ElasticStack* forneça um módulo de `Alerting` este só se encontra disponível em licenças pagas. Dado que se pretende criar uma ferramenta *open source* sem qualquer custo associado, decidiu-se não incluir este módulo na solução e procurar alternativas para implementar este sistema.

O `ElastAlert` é uma solução *open source* altamente modular e de fácil configuração que permite detetar e notificar os utilizadores acerca de alterações nos dados [79]. A lógica por detrás deste sistema baseia-se na criação de um conjunto de regras a executar periodicamente e na definição de alertas a enviar quando as condições indicadas forem satisfeitas. O `ElastAlert` fornece ainda uma API REST para poder gerir estas regras. Para facilitar esta gestão, o `ElastAlert` possui ainda um *plugin* para o *Kibana* que permite criar, editar, apagar e testar regras, tirando partido da API disponibilizada.

A configuração deste componente é feito através dos ficheiros `config.json` (Anexo B.11) e `elastalert.yaml` (Anexo B.12). Nestes ficheiros é efetuada a configuração do servidor do `ElastAlert` sendo definidas opções como a porta a utilizar, a localização das regras e *rules_templates*, as configurações e credenciais do *ElasticSearch* e ainda opções a aplicar globalmente a todas as regras.

Uma regra é definida num ficheiro *YAML* e é constituída por um conjunto de elementos que a permitem identificar (como nome, descrição e *rule_type*), uma interrogação a executar tendo em conta determinados parâmetros e diversos alertas a emitir caso se satisfaçam as condições pretendidas. Por forma a facilitar a sua escrita é fornecida um conjunto de *rule_templates* que podem ser utilizados como base.

Dos diversos *rule_types* que são disponibilizados salientam-se o `frequency`, o `spike`, o `flatline`, o `any` e o `change`. Além destes tipos básicos de regras existem ainda outros mais complexos que envolvem agregações de dados: `metric_aggregation`, `spike_aggregation` e `percentage_match`. Relativamente às integrações que o `ElastAlert` possui para o envio de alertas destacam-se o *Email*, o *Slack*, o *Telegram* e o *GoogleChat*. Para além destes, é ainda possível desenvolver *rule types* e alertas a utilizar na definição das regras.

O `ElastAlert` utiliza o *Elasticsearch* para armazenar várias informações sobre seu estado, evitando perdas de dados ou duplicação de alertas quando o serviço é reiniciado. Esta informação encontra-se guardada no `writeback_index` definido nas configurações e é criado através do *script* disponibilizado (`elastalert-create-index`). Neste índice é possível encontrar 3 tipos de documentos: `elastalert_status`, `elastalert` e `elastalert_error`. Os documentos do tipo `elastalert_status` permitem manter um *log* das *queries* efetuadas e são utilizados pelo `ElastAlert` para determinar quando executar novamente as regras e, desse modo, evitar a duplicação de *queries*. Todos os alertas emitidos bem como o seu conteúdo são registados em documentos

do tipo `elastalert`. Quando ocorrem erros no `ElastAlert`, estes são enviados para o *stderr* mas também são armazenados em documentos do tipo `elastalert_error`.

Como prova de conceito para a implementação do sistema de alarmística foi criado um conjunto de regras de diversos *rules types* para controlar tanto custos de operação como utilização de recursos e notificar os utilizadores via *Slack*.

A regra apresentada no Anexo B.13 auxilia os utilizadores a garantir que o orçamento diário estipulado não é excedido. Trata-se de uma regra do tipo `metric_aggregation` em que é despoletado um alerta caso o custo diário for superior ao `max_threshold`, neste exemplo de $1500, sendo enviada uma mensagem customizada para um canal do `Slack`.

Outra regra criada consiste na verificação da cobertura de instâncias EC2 do tipo *Spot* e pode ser consultada no Anexo B.14. Dado que este tipo de instâncias é consideravelmente mais barato que as restantes, considerou-se que os utilizadores deveriam ser alertados caso a percentagem de instâncias deste tipo fosse inferior a um dado valor. Deste modo, a regra criada é do tipo `percentage_match` e definiu-se uma `min_percentage` de 5%, sendo enviado um alerta para o `Slack` caso tal se verifique.

Dado que uma das funcionalidades pretendidas com esta solução é identificar instâncias com reduzida utilização, decidiu-se especificar uma regra que, tendo em conta a utilização dos recursos, notificasse os utilizadores acerca de recursos subaproveitados. Das diversas métricas disponíveis, decidiu-se recorrer à utilização de Central Processing Unit (CPU) para concluir acerca da utilidade das instâncias. Assim sendo, a regra apresentada no Anexo B.15, do tipo `frequency`, permite alertar os utilizadores via `Slack`, caso uma instância esteja durante 24 horas com uma carga de CPU inferior a 10%.

Para testar as regras definidas foi criado um canal no `Slack` denominado de "*cost-optimization*". A integração com o `ElastAlert` é realizada graças à criação de um *webhook* a ser utilizado na escrita das regras. Este *webhook* tem a si associado uma aplicação designada por "*ELK Cloud Cost Analysis*" que é responsável por publicar as mensagens no canal. Os alertas despoletados pelas regras apresentadas podem ser consultados na Figura 31.

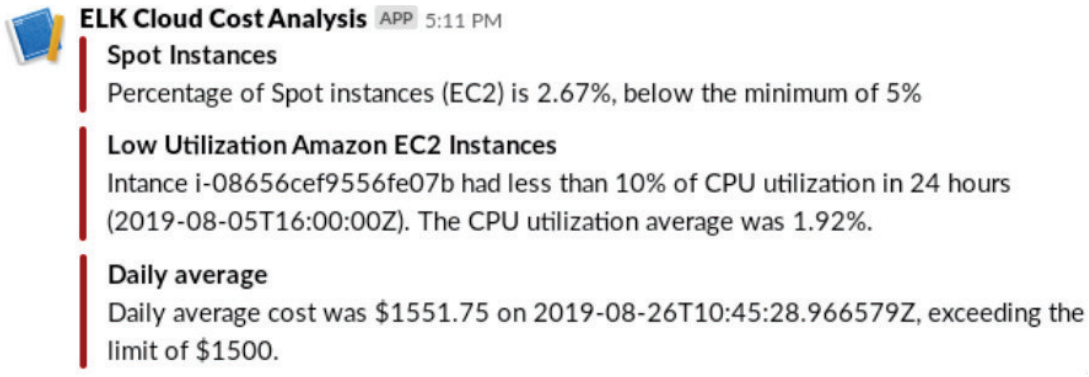


Figura 31: Exemplo de alertas

4.6 Ambiente de Desenvolvimento

O ambiente de desenvolvimento da solução foi construído com recurso a *Docker* [94]. O *Docker* é uma ferramenta desenhada para facilitar a criação, implementação e execução de aplicações utilizando *containers*. Os *containers* permitem criar um ambiente virtual em que todas as bibliotecas, dependências e conteúdos são isolados num único *package* denominado por `image`. Estas imagens podem ser criadas através da definição de um ficheiro designado por `Dockerfile`. A utilização desta ferramenta oferece inúmeras vantagens nomeadamente a interoperabilidade entre sistemas operativos.

Para a aplicação `Flask` criou-se um `Dockerfile` que pode ser consultado no Anexo B.16. Neste ficheiro são utilizadas, como base, as imagens de *Ubuntu* e de *Python*, são instaladas as dependências necessárias e definido o *entrypoint* para a execução da aplicação. Relativamente aos componentes da *Elastic Stack*, utilizou-se para o `ElasticSearch` e para o `Logstash` as imagens fornecidas pela *Elastic*. O *Dockerfile* para criação da imagem do *Logstash* com os *plugins* pretendidos encontra-se no Anexo B.17 e o do *ElasticSearch* no Anexo B.18. Devido às modificações ao código fonte que eram necessárias efetuar no `Kibana`, optou-se por não utilizar a imagem oficial de *Docker* e construir o componente no `Dockerfile` – Anexo B.19. Neste ficheiro utilizou-se um conceito designado por *multi-stage* que permite a optimização dos *Dockerfiles* e facilita a leitura e manutenção dos mesmos. Neste *Dockerfile* foram definidos dois *stages*: no primeiro *stage* é obtido o código fonte do *Kibana*, aplicado o *patch* desenvolvido e criada a *build* de produção a ser utilizada no segundo *stage* onde são instalados os *plugins* referidos e definido o *entrypoint* a executar. A *BitSensor* também disponibiliza uma imagem de *Docker* do `ElastAlert` e, como tal, tirou-se partido da sua existência para construir o ambiente de desenvolvimento.

A orquestração de vários *containers* pode ser feita através de um ficheiro *YAML* intitulado de `docker-compose.yaml`. Neste ficheiro são definidos diversos serviços, sendo possível configurar portas, *networks*, *volumes*, variáveis de ambiente, entre outras propriedades. O ficheiro criado para orquestrar o sistema desenvolvido pode ser consultado no Anexo B.20. A criação dos diversos componentes da solução é realizada através da execução do comando `docker-compose build` e a inicialização dos mesmos com recurso ao comando `docker-compose up`.

5

Casos de Estudo

Neste capítulo serão apresentados os casos de estudo que serviram para avaliar o sistema desenvolvido e tirar conclusões acerca da solução proposta. Dado a diversidade de CSPs que o sistema suporta, decidiu-se analisar clientes aos quais a empresa presta serviço e que utilizam a AWS e o *Microsoft Azure* para orquestrar a sua infraestrutura. Para demonstrar a integração com o GCP, criou-se uma conta neste fornecedor de *cloud* com alguns serviços básicos. Para cada caso de estudo são ainda propostas possíveis medidas de otimização de custos tendo em conta a análise de dados efetuada.

5.1 Caso de Estudo 1

O primeiro caso de estudo que serviu de objecto de análise é relativo a um cliente real que utiliza a AWS como CSP. O cliente em questão possui um site de *eCommerce* onde os utilizadores adquirem produtos e serviços eletronicamente. Esta aplicação constitui a base de trabalho deste caso de estudo.

Na generalidade, estas aplicações são constituídas por componentes arquiteturais comuns como o catálogo de produtos, o carrinho de compras, entre outros. Conforme referido no momento da identificação do problema (Secção 3.1), um dos aspetos que motivou o desenvolvimento desta dissertação foi a impossibilidade de análise de custos, com especial destaque para a análise por componentes.

Para além de possuir uma infraestrutura na *cloud* que conta com centenas de instâncias – o que, por si só, faz com que seja um bom caso de análise –, trata-se ainda de um cliente com diversas contas de serviço da AWS com uma política de "*Consolidated Billing*" [95], tornando este exemplo mais interessante e complexo. O facto de adotarem esta política permite que seja gerada uma única fatura que englobe todas as contas, facilitando a obtenção da informação relativa a custos de operação e dados de utilização.

No caso concreto deste cliente, existe uma conta responsável pela faturação e outras 6 contas de utilização criadas e organizadas de acordo com diversos fatores inerentes à política que o mesmo estabeleceu como o ambiente de desenvolvimento, a região onde se encontram, entre outros.

A infraestrutura na *cloud* deste cliente é gerida com recurso a uma tecnologia designada por `Terraform` [74]. O `Terraform` é uma ferramenta que permite construir, alterar e configurar infraestruturas de forma segura e eficiente, através da descrição dos componentes em ficheiros de configuração.

A aplicação que suporta o caso de estudo utiliza `Docker` no seu ambiente de desenvolvimento e produção. No caso prático desta aplicação, é criada uma imagem para cada um dos componentes arquiteturais (serviços) que constitui a aplicação. Essas imagens podem ser usadas nos ficheiros de configuração de `Terraform` para criar ou alterar *containers* de `Docker` e poder gerir a infraestrutura.

Conforme referido na Subsecção 3.2.3, a identificação dos recursos é realizada através de um mecanismo de *tagging*. A atribuição de *tags* aos recursos pode ser feita via consola *web*, contudo considerou-se que, para este cliente, este processo deveria ser realizado utilizando o `Terraform`. A possibilidade de estabelecer um conjunto de *tags* a atribuir a um recurso bem como efetuar a validação dos valores associados a essas *tags* são alguns dos motivos que levaram à escolha desta solução. Para além de permitir rotular os recursos no momento de criação dos mesmos, é ainda possível garantir que este processo é executado corretamente, por exemplo, com um *script* de verificação. Adotando esta solução é garantida a atribuição de *labels* aos recursos assim como a homogeneidade dos valores.

A titulo de exemplo, apresenta-se de seguida a criação de um *cluster* de Elastic Container Service (ECS) relativo a um sistema de *messaging* constituído por 3 instâncias *spot* com uma dada configuração e conjunto de *tags*:

```
module "ecs" {
  source                = "../terraform-aws-ecs"
  environment           = "${var.environment}"
  name                  = "${var.name}"
  image_id              = "${var.image_id}"
  region                = "eu-west-3"
  instance_type         = "i3.large"§
  spot_instance_price   = "0.18"
  desired_capacity      = "3"
  aws_profile           = "${var.aws_profile}"
  aws_region            = "${var.aws_region}"

  tags = {
    Name           = "messaging-kafka"
    Environment    = "tst"
    CostCenter     = "messaging"
  }
}
```

Ainda na Subsecção 3.2.3 foram nomeadas as principais categorias de *tags* a atribuir aos recursos. Por este ser o caso de estudo mais complexo, decidiu-se realizar as várias fases que envolvem o processo de criação de uma estratégia de *tagging*, sendo que para os restantes casos de estudo seria semelhante.

Primeiramente, foi necessário identificar os principais intervenientes da organização que influenciam o processo de *tagging*. Estes *stakeholders* incluíram essencialmente membros do departamento de consultoria, da equipa de desenvolvimento e da equipa de *DevOps*.

De seguida, foi importante compreender junto de cada elemento as suas necessidades relativas à separação de recursos de acordo com a sua área funcional. Desta análise verificou-se que a maioria das *tags* identificadas recaem nas categorias de *Business* (*Owner*, *CostCenter* e *Project*) e *Technical* (*Name*, *Environment* e *Role*). Além disso, foi ainda possível concluir que os diversos *stakeholders* possuíam necessidades idênticas, contudo utilizavam diferentes *tags* para o mesmo objetivo. Deste modo, a sua uniformização foi essencial para garantir a consistência do processo de *tagging*.

Para além de identificar as *tags* que cada recurso deve possuir, foi ainda importante estabelecer como seria realizada a atribuição de valores em recursos partilhados. Para tal, estabeleceu-se que a nomenclatura a utilizar em *tags* compostas seria no formato "`Tag: Key1=Value1/Key2=Value2`", em que os valores deverão ser números decimais e a sua soma totalizar uma unidade. A título de exemplo, considere-se um dado componente que é partilhado de igual forma, por questões de otimização de custos, pelo projeto "A" e pelo projeto "B". Se pensarmos ainda que estes projetos podem estar em contas separadas mas utilizam o mesmo componente, torna-se muito difícil realizar a rastreabilidade de custos. Utilizando uma *tag* composta "`Project: A=0.5/B=0.5`" é possível proceder à correta atribuição de custos, evitando que seja cobrado a apenas um elemento.

Terminado este processo, foi criado um ficheiro denominado por `resourceTags.json`, onde para cada uma das *tags* foram identificados os possíveis valores a atribuir, no seguinte formato:

```
{
    "Name": ["ResrouceName1", "ResourceName2"],
    "Environment": ["prd", "tst", "dev", "stout", "stg", "qal"],
    "Role": ["Frontend", "Backend","Messaging", "Redis", "Monitoring", "Gateway"],
    "Owner": ["Owner1", "Shared"],
    "CostCenter": ["Centro Custos X", "Online", "Sites"],
    "Project": ["Marketplace", "Digital"]
}
```

No momento da criação dos recursos é invocado um *script* que utiliza este ficheiro para validar a atribuição de *tags*, alertando o utilizador para eventuais irregularidades neste processo e abortando a criação do componente por forma a garantir a consistência das *tags*.

Terminado este processo, deu-se início à análise dos custos de operação e dados de utilização deste cliente. Por forma a compreender melhor a necessidade de este cliente possuir esta solução, apresenta-se na Figura 32, um gráfico que demonstra a evolução do número de registos de custos de operação ao longo do último ano. Conforme é possível constatar, até Outubro de 2018 a média do número de registos rondava os 200 000, contudo a partir dessa data este valor passou a ascender 1 200 000 de registos, em grande parte devido à criação de novas contas. Na altura deste acontecimento verificou-se um ligeiro aumento da faturação mensal, contudo este valor acabou por estabilizar nos $70 000/mês.

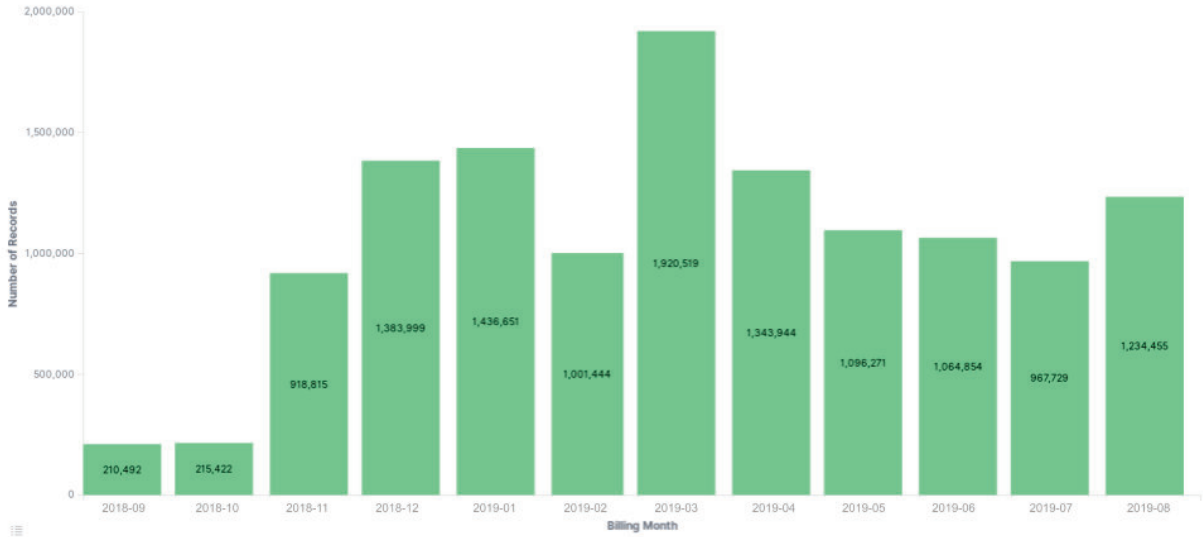


Figura 32: Evolução do nº de registos entre Setembro 2018 e Agosto de 2019

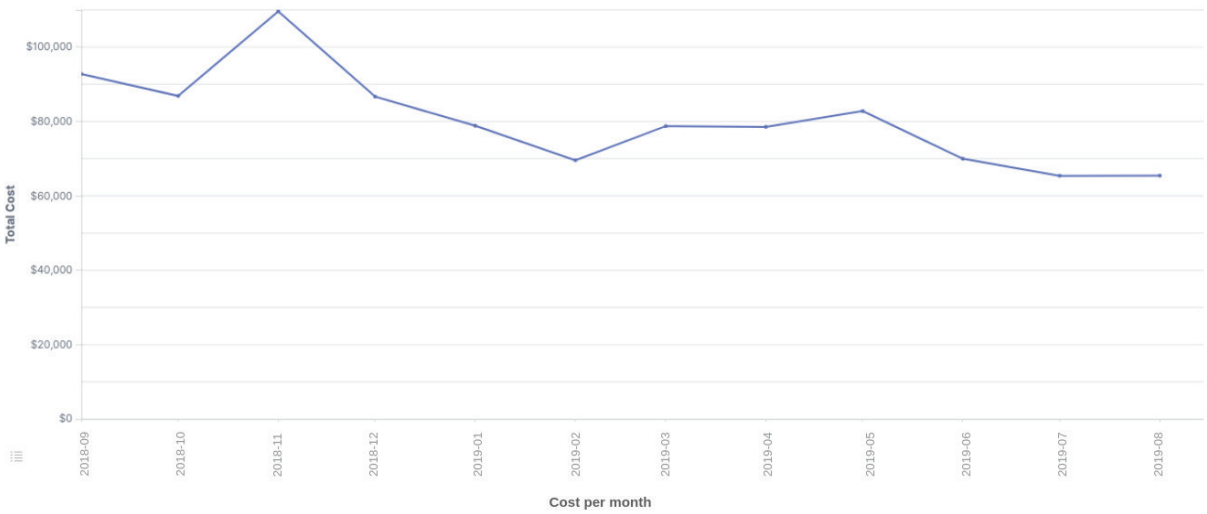


Figura 33: Evolução dos custos entre Setembro 2018 e Agosto de 2019

Para a análise dos custos de operação e dados de utilização criou-se um conjunto de visualizações e *dashboards* padrão, como foi referido na Subsecção 4.1.1. Estas visualizações são essencialmente tabelas de dados, gráficos circulares, de barras e de linhas, desenvolvidas tendo por base as necessidades dos utilizadores inquiridos, podendo ser alteradas e recriadas se este o pretender. As visualizações foram posteriormente agrupadas em *dashboards* de acordo com o contexto de análise de cada uma. No total foram criadas 50 visualizações, distribuídas por 6 *dashboards* distintos que podem ser consultados no Anexo A.3: *Billing-General* (Figura 46), *Billing-Tags* (Figura 47), *Billing-Networking* (Figura 48), *Billing-Reserved* (Figura 49), *Billing-EC2* (Figura 50) e *Usage Dashboard* (Figura 51).

A utilização desta ferramenta revelou-se bastante útil na análise geral de custos, nomeadamente por serviços, contas, tipo de instância, entre outro outros fatores. Além disso, foi ainda possível investigar com maior profundidade estes dados e tirar conclusões relativamente a custos provenientes de transferências de dados, instâncias EC2, reserva de instâncias, cobertura de *tags* e utilização de recursos. Este tipo de análise não é possível em grande parte das soluções estudadas e discutidas na Secção 2.4.

Analisando os *dashboards* em anexo, foi possível retirar o seguinte conjunto de conclusões:

- A maioria das instâncias EC2 foram adquiridas com a política *On-Demand*, sendo a percentagem de instâncias *Spot* e *Reserved* bastante reduzida. Dado que instâncias destes tipos podem custar cerca de um terço das *On-Demand*, uma das possíveis medidas de otimização seria optar por este tipo de políticas de *pricing* nas instâncias em que tal fosse viável.

- Relativamente a custos derivados de operações de *networking*, verificou-se que existia um custo significativo associado à transferência de dados. Embora grande parte dos dados tenham sido transferidos dentro da sua região (*IntraRegion*) e portanto a um preço reduzido, constatou-se que aqueles que eram transferidos entre regiões ou como tráfego externo representavam um custo relevante da faturação e, como tal, deveria ser alvo de otimização. Outro custo que se destacou desta análise é relativo aos pedidos feitos a diversas APIs, nomeadamente, as que envolvem operações com *buckets* S3.

- Todas as contas do cliente possuem instâncias reservadas, tipicamente adquiridas por um período de 1 ano, contudo em reduzida quantidade. Contrariamente às ferramentas analisadas, a solução desenvolvida permite reunir informação sobre a reserva de instâncias de diversas contas num único local e obter maior visibilidade sobre a mesma, podendo averiguar dados como: datas de aquisição e termo das reservas, modo de aquisição (1 ou 3 anos), opções de pagamento (*All Upfront*, *Partial Upfront* e *No Upfront*), unidades reservadas, entre outros dados. Esta informação pode ser utilizada pelo sistema de alarmística para notificar o término de reservas.

- A identificação de recursos através de *tags* permitiu filtrar resultados e explorar custos de acordo com os valores atribuídos às mesmas. Por forma a garantir o sucesso deste processo é possível consultar para cada *tag* a taxa de cobertura e os valores atribuídos.

- Os dados de utilização dos recursos permitiram derivar diversas métricas das instâncias – como CPU, *network* e *disk* –, assim como de outros serviços nomeadamente S3, RDS, ECS, entre outros. Estas métricas foram úteis para sugerir diversas medidas de redução de custos como a remoção de instância desligadas ou com baixa utilização, o redimensionamento de instâncias, o deslocamento de serviços para outras regiões e a utilização de opções de *storage* com custo mais reduzido – como o S3 *Glacier*, por exemplo para *backups*.

Um dos principais elementos desta solução, e que a distingue das restantes ferramentas analisadas, é a existência de uma barra de pesquisa partilhada nos ambientes de *Discover*, *Visualize* e *Dashboard* do *Kibana*. Neste componente pode-se efetuar análises sobre os dados armazenados nos índices do *ElasticSearch* através da escrita de *queries* sobre os formato Kibana Query Language (KQL) ou *Lucene*. Além da simplicidade de utilização, disponibiliza poderosas funcionalidades como *auto-complete*, inclusão, exclusão e filtragem de resultados, suporte a expressões regulares, entre outras. No momento em que a *query* é efetuada, todos os elementos, sejam eles gráficos, tabelas, histogramas, listas, ou qualquer outro elemento visual, são afetados por esta pesquisa. Na Figura 34 é apresentado o resultado da aplicação de uma *query* que visa filtrar os documentos apresentados no *Discover* pelo ambiente de "*tst*", cujo componente contêm "Kafka" no seu nome e é uma instância do tipo EC2.

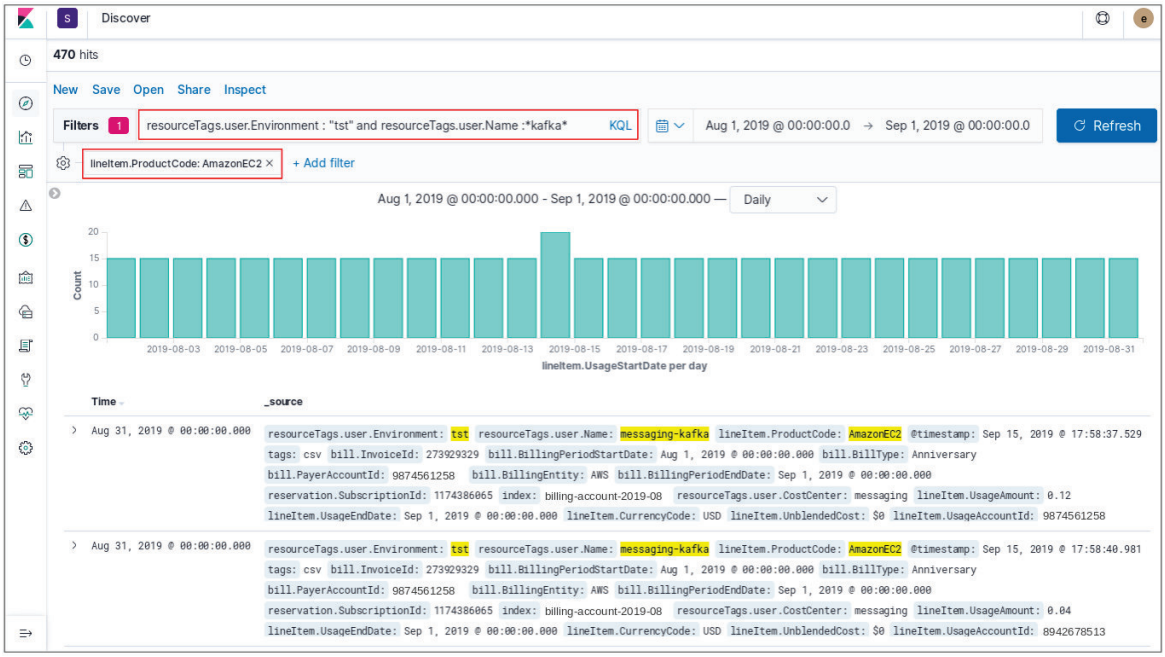


Figura 34: Pesquisa e filtragem de documentos no Discover

A elevada interatividade que oferece constitui outras das vantagens desta solução. A criação de novas visualizações e *dashboards*, a exploração de documentos e a construção de novas métricas fornece a liberdade que o utilizador necessita para tirar conclusões acerca de custos de operação e dados de utilização, contrariamente às soluções estudadas em que estes dados eram estáticos e limitados.

O sistema de alarmística desenvolvido foi outro componente fulcral no processo de otimização de custos. No caso deste cliente foram despoletados avisos relativos a custos acima do orçamento estipulado, à reduzida percentagem de instâncias *spot* e *reserved*, ao elevado número de pedidos a determinadas APIs e ainda a instâncias com reduzida utilização. A título de exemplo, este último alerta permitiu identificar numa conta de ambiente de desenvolvimento cerca de 32 instâncias com reduzida utilização, tendo-se vindo a verificar que a grande maioria não estava a produzir trabalho útil e as restantes possuiam recursos para além dos necessários, tendo sido redimensionadas. A destruição e redimensionamento destas instâncias permitiu gerar uma poupança de aproximadamente $1 000.

5.2 Caso de Estudo 2

O segundo caso de estudo que foi objeto de análise nesta dissertação diz respeito a um cliente que possui uma loja de aplicações móveis para *Android* e cuja infraestrutura se encontra na AWS. Dado que o CSP é o mesmo que o apresentado na Secção 5.1, os *dashboards* padrão expostos foram também alvo de análise.

No caso deste cliente foi realizada uma análise com elevado nível de detalhe por serviço. Uma vez que se trata de um cliente de menor dimensão quando comparado com o apresentado em 5.1, este possui um número inferior de instâncias e serviços aos quais recorre. Para além da exploração de custos e utilização das instâncias EC2, analisou-se os principais serviços que este cliente utiliza: *ElastiCache*, S3, *Load Balancers*, *Route 53* e RDS.

Analisando a faturação deste cliente num período de 4 meses – Maio de 2019 a Agosto de 2019 –, averiguou-se que este gastava, em média, cerca de $32 000/mês (Figura 35).

Amazon Elastic Compute Cloud (Amazon EC2)

O serviço de *compute* da AWS (*Amazon EC2*) é aquele possui o maior peso na faturação do cliente, representando cerca de 60% do custo total mensal, isto é, $\approx$ $19 000.

A nível de servidores EC2, verificou-se a existência de 120 instâncias, estando 70 delas em Auto-Scaling Groups (ASGs). No caso das instâncias presentes em ASGs, constatou-se a presença de uma série de instâncias reservadas do tipo `c5.xlarge`, `m5.large` e `m5.xlarge`, contudo o período de reserva das mesma terminou no inicio de Agosto. Posto isto, estas instâncias passaram para *On-Demand* fazendo com o que o custo com este serviço ascendesse para os $25 000 mensais (Figura 35).

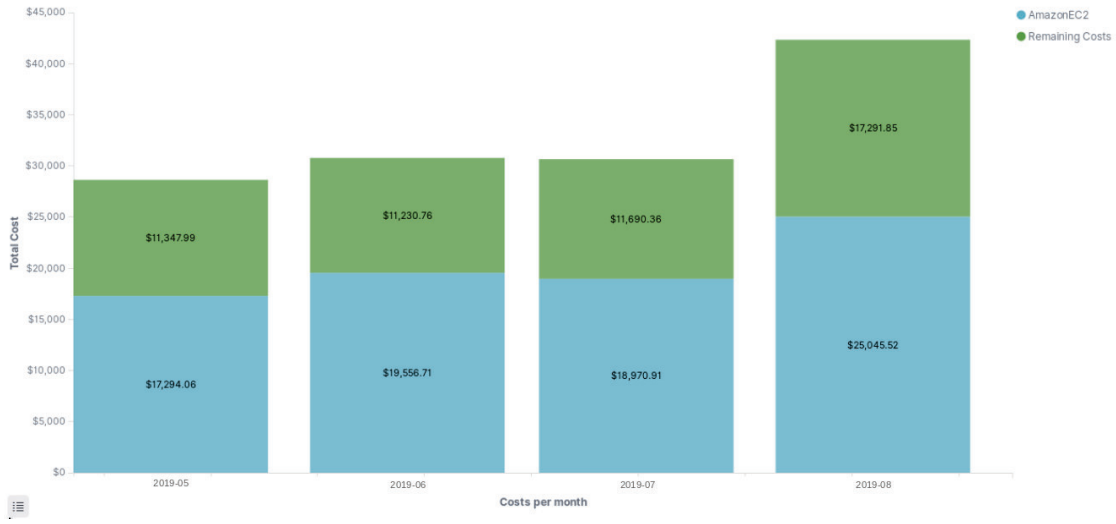


Figura 35: Custos com instâncias EC2

Dado o potencial de usar instâncias *Spot* nos ASGs, sugere-se como medida de otimização a revisão destas instâncias. Para além disso, é possível adicionalmente colocar algumas instâncias reservadas para colmatar a insuficiência de *spots*. A utilização de *spots* permitiria reduzir até $5 463/mês face aos valores das *On-demand*.

ElastiCache

A nível de *ElastiCache* [96], constatou-se a existência de 4 recursos *Memcached* e 30 recursos *Redis*. O custo mensal deste serviço ronda os $5 000, contudo verificou-se um aumento significativo em Agosto de 2019 uma vez que parte destes recursos estavam reservados até essa data (Figura 36).

Figura 36: Custos com ElastiCache

Na componente de *ElastiCache* o foco principal são as instâncias do tipo `cache.r3.large` e `cache.r3.xlarge`, uma vez que representam grande parte do custo associado a este serviço. Como possíveis melhorias destaca-se a possibilidade de realizar o *upgrade* destas instâncias para umas de geração mais atual como as `cache.r5.large` e `cache.r5.xlarge` pois possuem características semelhantes a um menor custo em *On-demand*. Dado que estas instâncias tem sido mantidas ao longo do tempo, destaca-se ainda a possibilidade de adquirir instâncias reservadas deste último tipo pela duração de um ano.

Dos 30 *clusters* de *Redis* existentes, 18 utilizam instâncias `cache.r3.large`. Estes *clusters* em *On-Demand* possuem um custo de $6 583/mês. Se optarmos por mudar para instâncias do tipo `cache.r5.large`, obtêm-se uma redução para os $6 247/mês. Caso se reserve estas mesmas instâncias pelo período de um ano, este valor reduz ainda para os $4 251/mês.

Relativamente aos 4 *clusters* de *Memcached*, verifica-se que 2 utilizam instâncias do tipo `cache.r3.xlarge`. Estes *clusters* numa política *On-demand* representam um custo de $2 195/mês. Elegendo-se instâncias do tipo `cache.r5.xlarge` reduz-se este valor para os $2 082/mês, sendo que se forem reservadas por 1 ano, este valor passaria para $1 416/mês.

Assim sendo, com estas otimizações é expectável uma redução total de custos de cerca de $3 111/mês no componente de *ElastiCache*.

Amazon Simple Storage Service (Amazon S3)

Relativamente ao serviço *Amazon* S3, verificou-se a existência de diversos *buckets* de dimensão razoável com *storage* repartida maioritariamente pelos tipos *Standard* ou *Glacier*. Os custos com este serviço são superiores a $2 000 mensais, sendo os gastos com `EU-TimedStorage-ByteHrs`, `EU-Requests-Tier1` e `EU-TimedStorage-GlacierByteHrs` os mais relevantes (Figura 37).

O `EU-TimedStorage-ByteHrs` trata-se de um componente deste serviço responsável pelos custos com armazenamento de objetos em *buckets* S3 do tipo *Standard*. De entre os diversos *buckets* que este cliente possuí, destaca-se no *top* 3 aqueles que possuem maior espaço de armazenamento ocupado: 21 TB, 19.1 TB e 4.75 TB. O custo para armazenamento dos objetos nestes *buckets* é cerca de $1 031/mês.

Considerando que o *bucket* com 4.75 TB é destinado a *backups*, destaca-se como medida de otimização a possibilidade de migrar, pelo menos de forma parcial, alguns destes objetos para *Glacier*, obtendo uma redução de $109/mês para $19/mês.

Para os outros dois *buckets*, constatou-se que o número de pedidos `GET` e `HEAD` ao `Requests-Tier2` é bastante baixo quando comparado com os pedidos `POST` e `PUT` ao `Requests-Tier1`, fazendo destes *buckets* bons candidatos para usar o *S3 Infrequent Access* [97]. Esta categoria é indicada para *buckets* cujos dados são acedidos com pouca frequência, mas que exigem baixa latência e alto *throughput*. Neste caso em concreto, a utilização desta solução permitia reduzir o custo de $922/mês para $501/mês.

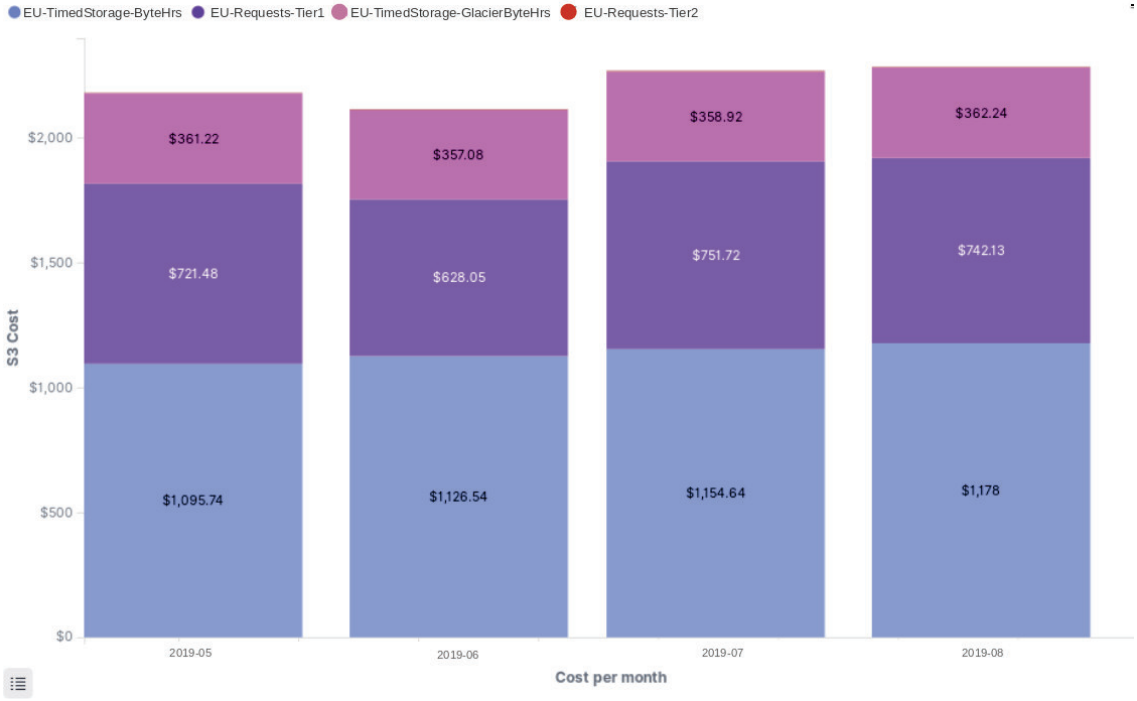


Figura 37: Custos com Amazon S3

Da análise efetuada a este serviço verificou-se ainda a inexistência de uma *S3 VPC Gateway*, fazendo com que todo o tráfego tenha custo de *Data Transfer* para a *Internet*. Este custo é identificado como `InterZone-Out` e atinge cerca de $400 mensais. Como otimização sugere-se então a criação de uma *VPC Gateway* para redução de custos com *Data Transfer*.

Tendo em conta as medidas sugeridas apenas para o *top* 3 dos *buckets* identificados, é expectável uma poupança de pelo menos $511 mensais neste serviço.

Load Balancers

Este cliente possuí 15 Application Load Balancers (ALBs) e um ELB, distribuídos pelos diversos componentes das aplicações. Os custos com este serviço representam uma fatia significativa da faturação total, ascendendo a $6 000 mensais. Deste valor, verificou-se que metade era devido a tráfego de *Downloads* – `EU-DataTransfer-Out-Bytes` (Figura 38).

Analisando os ALBs constatou-que que em 5 deles existiam diversos *Target Groups*, baseados em *Path* e/ou *Host-header*, e devido a carga existente nos mesmos, estes deveriam ser mantidos intactos. Relativamente aos restantes 10, estes apenas possuíam um único *Target Group* e, como tal, deveriam ser otimizados. Uma vez que os ALBs suportam múltiplos certificados, *Host-headers* e *rules*, sugere-se como medida de otimização a concentração destes componentes num único, possibilitando uma redução de $166/mês.

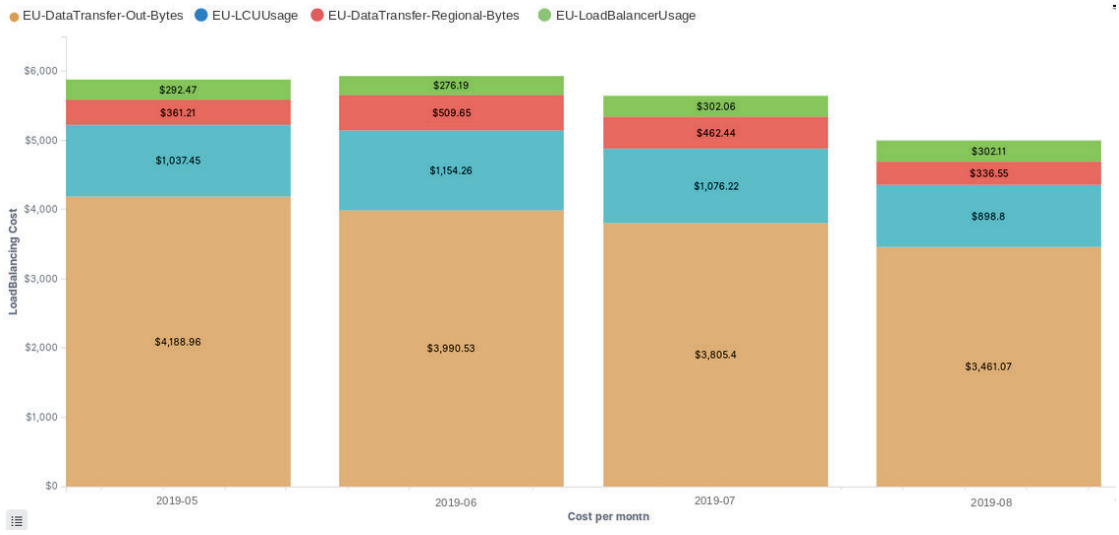


Figura 38: Custos com Load Balancing

Route 53

Relativamente ao serviço de *Route 53* verificou-se a existência de 15 *Hosted Zones* e um total 967 registos de DNS, tendo um custo total de $\approx$ \$1 000 mensais (Figura 39). Embora este serviço não costume ter grandes custos associados, constatou-se que o número de DNS *queries* é elevado, mais concretamente as *Geo-queries* [98]. Este tipo de de *queries* são efetuadas quando se deseja reencaminhar o tráfego com base no local dos recursos ou quando se pretende desviar tráfego de recursos num local para outros noutra localização.

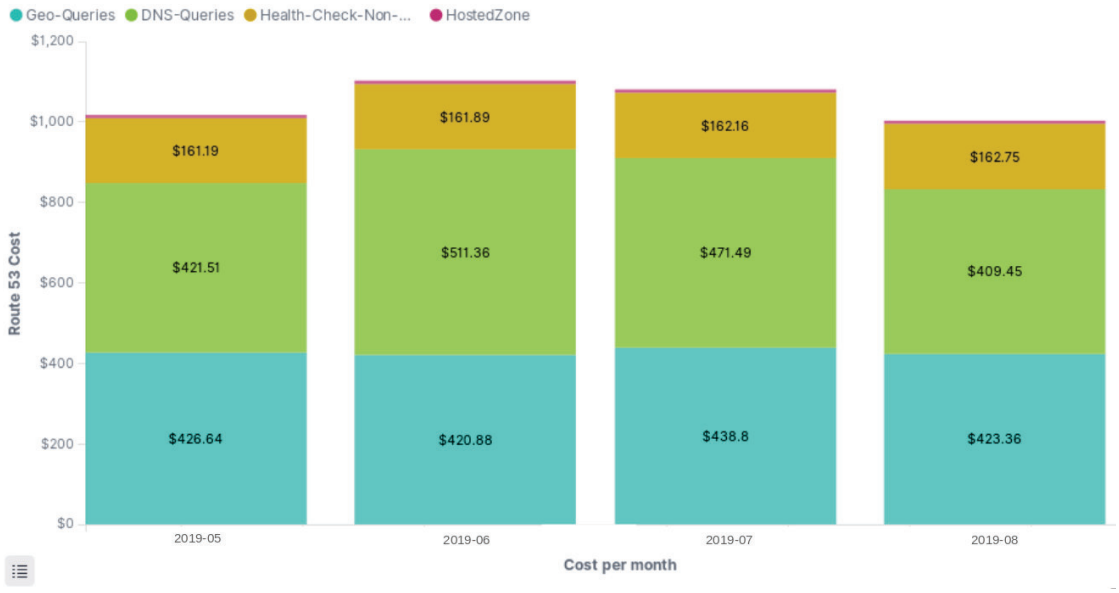


Figura 39: Custos com Route 53

Da análise levada a cabo conclui-se que a grande parte dos registo de DNS com geolocalização ativa apenas possuem um registo simples de DNS. Nesses casos, os pedidos de DNS são mais caros, podendo ser utilizadas *Simple Queries* e, assim, reduzir o custo. Com a adoção desta medida prevê-se uma redução de custo dos $438/mês para os $250/mês, obtendo-se uma poupança de $188/mês com este serviço.

Amazon Relational Database Service (RDS)

Este cliente possui um número considerável de Base de Dados RDS, sendo na sua maioria `PostgreSQL`. Os custos com este serviço rondam os $1 500 mensais até Julho de 2019, tendo-se verificado um aumento significativo no mês de Agosto devido à perda de instâncias reservadas (Figura 40).

Analisando as Bases de Dados deste cliente, destacam-se 4 devido à sua dimensão: a primeira constituída por um *master* `db.m4.large` e 2 réplicas `db.m4.xlarge`, a segunda por um *master* e 1 réplica ambas do tipo `db.m4.2xlarge` e as outras duas do tipo `db.m4.xlarge`.

Como medida de otimização sugere-se a renovação da reserva de instâncias pelo período de um ano. Considerando as 4 bases de dados de maior dimensão, constata-se que a reserva de 3 `db.m4.large`, 2 `db.m4.xlarge` e 2 `db.m4.2xlarge` permitem satisfazer as necessidades do *cliente* e gerar uma poupança de $519/mês.

Adicionalmente, verificou-se um excesso de alocação de disco em RDS nas 3 instâncias de uma destas bases de dados de grande dimensão. Nestas instâncias estão alocados 6 TB representando um custo de aproximadamente $762/mês. A redução para 600 GB em cada uma das instâncias possibilita a poupança de $228/mês sem prejuízo para o cliente.

A aplicação das medidas propostas permite uma poupança total de $747 mensais.

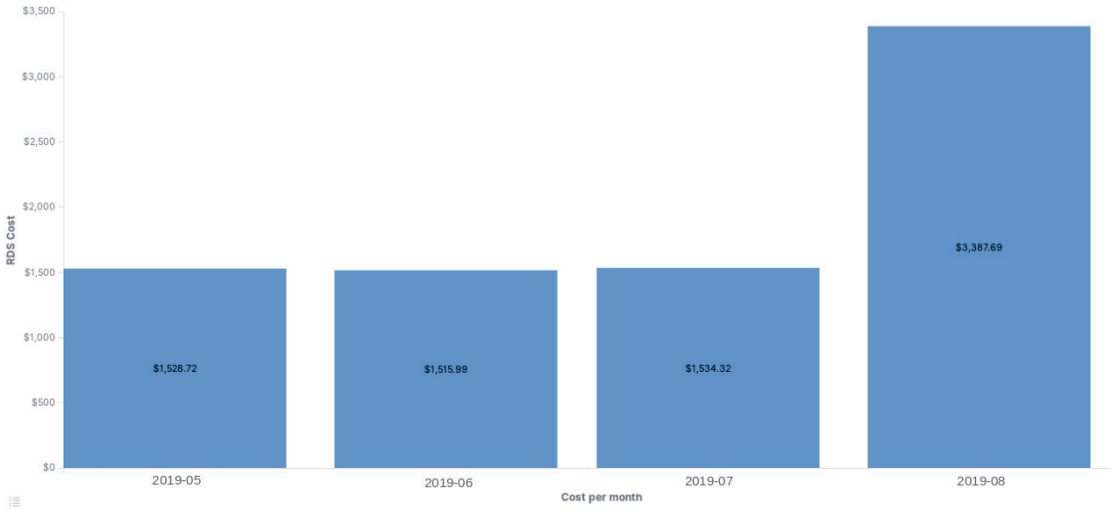


Figura 40: Custos com RDS

Análise de soluções

Ao longo desta secção foi realizada uma análise detalhada pelos principais serviços que o cliente usufrui. Para cada um foram identificados os principais elementos que contribuem para o seu custo e propostas possíveis medidas de otimização de acordo com cada cenário. Considerando estas soluções, a poupança total expectável ascende os $10 000 mensais.

No momento da análise destes custos – Setembro de 2019 – verificou-se que a faturação do cliente se situava nos $42 000 mensais, contrariamente à média mensal dos últimos meses de $30 000. Este aumento de custos deveu-se essencialmente à perda das instâncias reservadas mas também ao crescimento da infraestrutura. Assim sendo, com as medidas propostas é expectável uma redução de custos superior a 20%, possibilitando que o cliente mantenha a sua faturação mensal nos $32 000.

5.3 Caso de Estudo 3

O terceiro caso de estudo diz respeito a um cliente industrial que possui parte da sua infraestrutura na *cloud*, sendo o *Microsoft Azure* o CSP eleito. À semelhança da AWS, também foram criadas visualizações e *dashboards* padrão para este CSP, as quais foram utilizadas para a análise de custos.

Comparativamente aos clientes apresentados nas Secções 5.1 e 5.2, verificou-se que este detém um menor número de instâncias e serviços dos quais usufrui e, por isso, os custos de operação são bastante mais reduzidos. Analisando os 3 últimos meses de faturação constata-se que o custo mensal é aproximadamente 185€ (Figura 41).

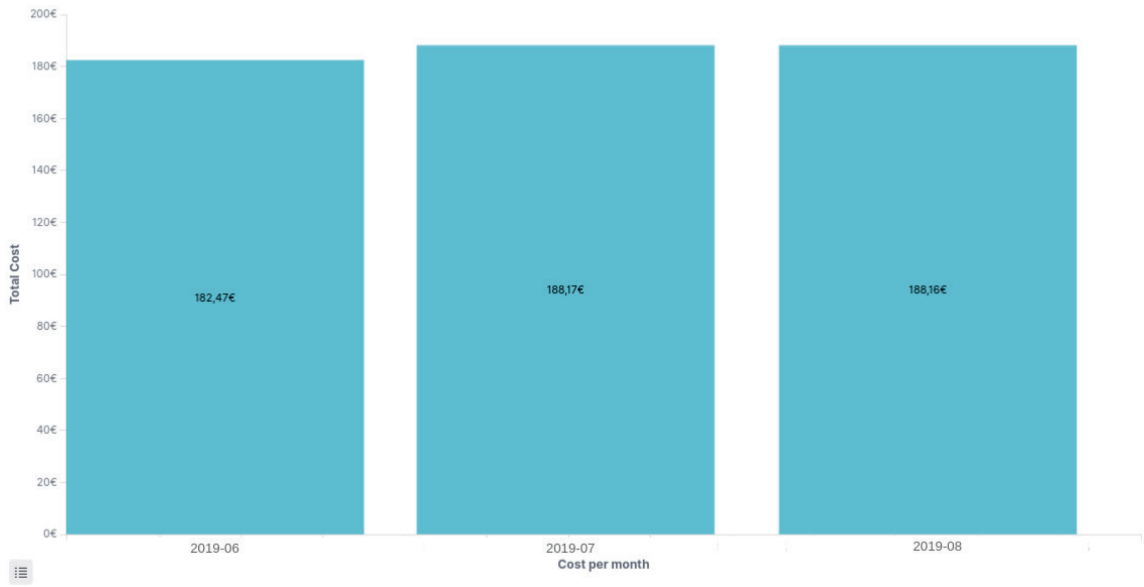


Figura 41: Faturação mensal no período de Junho de 2019 a Agosto de 2019

No total este cliente possui 37 recursos, sendo os mais importantes para a análise de custo os dos seguintes tipos: *Virtual Machines* (9), *Storage Account* (4), *Network Interface* (9) , *Public IP address* (9), *App Service* (1) e *Event Hub* (1).

Analisando o *dashboard* relativo a custos de operação (Figura 52) e o *dashboard* referente aos dados de utilização (Figura 53) presentes no Anexo A.4, é possível tirar o seguinte conjunto de conclusões:

- O custo diário de utilização destes recursos mantêm-se praticamente inalterado ao longo do mês, sendo em média 6,07€.
- No *top* 3 de custos encontram-se os serviços de *Compute* (*Virtual Machines*) a representar mais de 90% dos mesmos, o serviço de *Event Hub* com 5% e custos com *Storage* a pesar 3% da faturação.
- Das 9 *Virtual Machines* identificadas, apurou-se que 7 delas eram do tipo `A1`, 1 do tipo `A0` e 1 do tipo `D1 v2/DS1 v2`. Dos cerca de 170€ gastos em *Virtual Machines*, cerca de 120€ é gasto com as do tipo `A1`.
- A marcação de recursos é bastante reduzida cobrindo apenas 15% dos mesmos, essencialmente o *Event Hub* e algumas das *Virtual Machines*. Assim sendo, deveria ser realizado um processo semelhante ao apresentado na Secção 5.1 com o objetivo de aproximar a cobertura dos 100% e facilitar a rastreabilidade de custos.
- Analisando as políticas de obtenção dos recursos, averiguou-se que todos são adquiridos segundo o mecanismo *pay as you go*. Dado a estabilidade dos recursos que o cliente possui, seria expectável que parte desses recursos fossem reservados.

Para além da elaboração de uma estratégia de *tagging*, sugere-se que este cliente adquira recursos através de políticas de reserva, idealmente pelo período de 1 ano. Dado que a maioria das *Virtual Machines* pertencem a série A e, como tal, não podem ser reservadas, apenas é possível realizar a reserva da *Virtual Machine* do tipo `D1 v2/DS1 v2`. A reserva desta instância pelo período de um ano permite reduzir o custo dos 42,41€ para os 22,94€, obtendo-se uma poupança de 19,42€.

Analisando os dados de utilização das *Virtual Machines*, foi possível concluir que 3 do tipo `A1` possuíam excesso de recursos. Como medida de otimização de custos aconselha-se a redução destas instâncias para o tipo `A0`, possibilitando a poupança de 5,65€ por *Virtual Machine* e, por isso, uma redução total de 16,95€.

A aplicação das medidas sugeridas garante uma otimização dos custos de operação deste cliente em cerca de 20%, obtendo uma poupança final de 36,37€. Deste modo, a faturação mensal do cliente reduz dos 185€ mensais para aproximadamente 150€.

5.4 Caso de estudo 4

O quarto e último caso de estudo visa comprovar a integração da solução criada com o Google Cloud Platform (GCP). Uma vez que no contexto empresarial em que foi desenvolvida esta dissertação não existia nenhum cliente que pudesse servir de caso de estudo, decidiu-se criar uma conta neste CSP e utilizá-la para testar a aplicação.

Para ser possível obter dados de custos de operação e utilização dos recursos, decidiu-se criar uma infraestrutura que desse suporte a um sistema desenvolvido na área da mobilidade urbana e dos transportes públicos. Este sistema é constituído por uma aplicação *web* e respetivo *backend* orientado aos micro-serviços. Para além disso, esta conta ainda oferece diversas APIs à aplicação, nomeadamente as do *Google Maps* como a `Maps JavaScript API`.

A infraestrutura arquiteturada para o projeto é composta por 2 instâncias computacionais de uso geral do tipo `f1.micro`, um *bucket* onde são armazenados dados (GCS) e uma VPC. Para além destes recursos, foram ainda ativadas as APIs necessárias para o funcionamento da solução desenvolvida: `Cloud Resource Manager API`, `Compute Engine API` e `Stackdriver Monitoring API`.

À semelhança dos outros casos de estudo, também neste foram utilizados os *dashboards* padrão criados para o GCP e que podem ser consultados no Anexo A.5: *Billing-General* (Figura 54) e *Usage-General* (Figura 55).

Conforme é possível constatar pelo *dashboard* de *billing*, a faturação do mês de Julho de 2019 ronda os $45 mensais, uma média de $1.46/dia. Analisando este valor, destacam-se como mais relevantes os custos com recursos computacionais que representam cerca de 35% do valor total ($\approx$ $15), as APIs de mapas com 24% desse valor ($\approx$ $11), gastos com *networking* no valor aproximado de $10 (22%) e os serviços de *storage* aos quais é atribuído 15% do total ($\approx$ $7).

Da análise levada a cabo foi ainda possível tirar algumas conclusões relativamente aos pedidos que estavam a ser efetuados aos serviços da infraestrutura. Cerca de 50% dos pedidos eram do tipo `Multi-Regional Storage`, sendo que 8% eram referentes a `Class A Requests` (inserções e listagens de objetos e *buckets*) e 42% a `Class B Requests` (obtenção e acesso aos objetos). As invocações às APIs do *Google Maps* representam cerca de 23% do número total de pedidos e os restantes 27% são pedidos à `Monitoring API`.

O *dashboard* que agrega dados de utilização dos recursos permite tirar conclusões acerca de diversas métricas em termos computacionais como CPU, *network*, *disk* e *firewall*, *storage*, entre outras. Esta análise deve ser feita orientada ao projeto e instância ou grupo de instâncias. Mais uma vez, o uso de *tags* é essencial tanto para a análise de custos como de utilização de recursos.

6

Conclusões e trabalho futuro

O último capítulo desta dissertação expõe as considerações finais obtidas do desenvolvimento de uma solução de otimização de custos de operação no ambiente de *cloud*. As principais conclusões e resultados obtidos deste desenvolvimento encontram-se expostas na Seccção 6.1. Na Secção 6.2 são descritas algumas sugestões para trabalho futuro deste projeto tendo em vista a continuação e melhoramento da solução desenvolvida.

6.1 Considerações Finais

Esta dissertação de mestrado teve como principal objetivo o desenvolvimento de uma ferramenta que permitisse a monitorização e análise de custos operacionais derivados da utilização de recursos na *cloud*, tendo como finalidade a otimização dos mesmos.

A crescente utilização da *cloud* para orquestração de aplicações e infraestruturas computacionais, o aumento da oferta de serviços pelos CSPs e a definição de políticas de *pricing* cada vez mais complexas motivaram o desenvolvimento desta dissertação.

No decorrer desta dissertação de mestrado foram seguidas as diversas etapas que a metodologia do DSR depreende. Para compreender o problema em estudo levou-se a cabo uma investigação sobre os principais conceitos teóricos e científicos que o mesmo envolve. Nesse sentido, exploraram-se os termos fundamentais do *cloud computing* e identificaram-se os fornecedores de serviços de *cloud* com maior influência no mercado da *cloud*: Amazon Web Services (AWS), *Microsoft Azure* e Google Cloud Platform (GCP). Para cada CSP foram caracterizados os principais serviços que cada um oferece bem como as

politicas de *pricing* praticadas. Considerando que se pretende otimizar custos operacionais e, dada a complexidade dos preços praticados pelos CSPs, decidiu-se aprofundar a análise relativamente aos modelos de *pricing* aplicados na *cloud*. Terminada esta investigação procurou-se as soluções existentes no mercado para combater este problema. Embora cada CSP possua a sua solução de análise de custos, estas revelaram-se muito limitadas e incapazes de satisfazer as necessidades básicas deste problema. Seguidamente, efetuou-se um estudo de mercado para identificar plataformas que suportem os CSPs pretendidos e permitam a monitorização de custos de operação e utilização de recursos. Desta pesquisa resultou a análise de 2 soluções: `CloudCheckr` e `CloudAware`. Embora estas soluções fossem bastante melhores em termos de funcionalidades que as fornecidas gratuitamente pelos CSPs, não solucionavam o problema uma vez que a visibilidade sobre custos de operação e dados de utilização de recursos era reduzida e limitada e a análise por componentes arquiteturais ou aplicacionais era praticamente inexistente.

Terminada a consciencialização do problema, procedeu-se à elaboração de possíveis soluções exequíveis para o problema em estudo. No decorrer deste processo foram expostos os principais desafios enfrentados e as decisões encontradas para os ultrapassar. Considerando os requisitos impostos e os utilizadores da solução, efetou-se uma análise de possíveis tecnologias a usar no produto final. Deste processo resultou uma proposta de solução assente numa aplicação *Flask* para obtenção e tratamento de informação e numa *Elastic Stack* constituída por um *Logstash*, um *ElasticSearch* e um *Kibana*.

De seguida, levou-se a cabo o desenvolvimento desta proposta tendo sido explicado todo o processo e decisões tomadas. De acordo com a metodologia utilizada, a implementação desta solução resultou num artefacto a ser avaliado posteriormente. A ferramenta desenvolvida revelou-se bastante útil na análise dos custos de operação e dados de utilização. A possibilidade de efetuar *queries* e filtragens sobre os dados armazenados bem como a criação de visualizações e *dashboards* customizáveis e interativos são algumas das características que distinguem esta solução das restantes. O sistema de alarmística criado permitiu o controlo de custos e de recursos, assim como auxiliar na identificação de recursos que satisfaçam determinadas condições, por exemplo, instâncias subutilizadas.

Posto isto, avaliou-se a solução criada tendo por base 4 casos de estudo distintos: 2 clientes reais que utilizam como CSP a AWS, 1 cliente real cujo CSP é o *Microsoft Azure* e 1 cliente simulado para testar a integração com o GCP. A utilização do sistema desenvolvido nos casos de estudo referidos possibilitou a identificação dos serviços com custos mais elevados, sendo possível explorar os principais motivos para esse custo. A definição de *tags* permitiu analisar custos operacionais por componente aplicacional ou arquitetural, proporcionando a separação dos mesmos e a correta atribuição de responsabilidades. Dos casos de estudo analisados verificou-se que grande parte dos custos eram derivados de recursos computacionais. De entre várias medidas, a aquisição de instâncias reservadas ou *spots* e o redimensionamento de recursos são aquelas que mais se destacaram na análise. Para além disso, a exploração de custos derivados de *storage* e *networking* foram também alvo de otimização. Considerando as medidas propostas em cada caso de estudo, constata-se uma otimização de custos na ordem dos 20%.

Tendo em conta os objetivos definidos na Secção 1.2, pode-se concluir que estes foram amplamente alcançados. A solução desenvolvida foi de encontro aos requisitos levantados e permitiu solucionar as principais vertentes do problema.

6.2 Trabalho futuro

O protótipo apresentado nesta dissertação de mestrado satisfaz o propósito inicial e serve como prova de conceito ao problema em estudo. Mesmo assim, várias podem ser as melhorias efetuadas à solução desenvolvida.

Para enriquecer a ferramenta elaborada destaca-se a possibilidade de integração com novos CSPs nomeadamente a *Oracle*, *IBM*, *Alibaba Cloud* e *DigitalOcean*. Nenhuma das soluções analisadas na Secção 2.4 possibilitava a monitorização de custos e dados de utilização de qualquer um dos fornecedores de *cloud* referidos.

O aperfeiçoamento do sistema de alarmística constituí outra possível melhoria à ferramenta criada. A definição de novas *rules* para controlo de custos e de recursos e a utilização de novos mecanismos de notificação dos utilizadores como *Email* e o *Telegram* são algumas das funcionalidades que permitiriam a evolução da solução.

Embora o sistema se baseie na análise de dados de forma altamente interativa através de construção de visualizações e *dashboards*, considera-se relevante a criação de um módulo de *reporting*. Neste módulo poderiam ser definidos *templates* de relatórios de diversos tipos – uns de carácter mais geral e outros mais específicos, por exemplo por serviço, conta, região, entre outros – e exportados para diversos formatos, como PDF ou *LaTeX*. Outra funcionalidade interessante seria o envio periódico para o cliente dos principais relatórios, por exemplo, semanalmente ou mensalmente via email. Provavelmente este processo envolverá a melhoria da API desenvolvida na aplicação *Flask* (*Cloud Cost Analysis*).

A elaboração de mecanismos de automatização melhoraria significativamente a solução. O controlo, realocação e redimensionamento de instâncias computacionais, a aquisição automática de instâncias reservadas e *spots* e a gestão de elementos de *storage* são algumas das possíveis otimizações. Para o desenvolvimento deste componente dever-se-á tirar partido das *tags* dedicadas à automação (Figura 22).

Por último, sugere-se a introdução de *machine learning* no protótipo criado. Das diversas melhorias que podem ser realizadas recorrendo à inteligência artificial, salienta-se a criação de um mecanismo para previsão de custos e um sistema de recomendação de medidas de otimização de custos e recursos.

Bibliografia

[1] Forbes. Roundup Of Cloud Computing Forecasts And Market Estimates. https://www.forbes.com/sites/louiscolumbus/2018/09/23/roundup-of-cloud-computing-forecasts-and-market-estimates-2018/#1c9aefc4507b, 2018. Acedido a 18-12-2018.

[2] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. Technical report, National Institute of Standards & Technology, 2011.

[3] IDG Communications Inc. Cloud Computing Survey: 2018. Technical report, 2018.

[4] RightScale. State of the Cloud Report. Technical report, 2018. Acedido a 25-11-2018.

[5] Sophia I. Vargas. Emerging Role Profile: Cloud Cost Manager. Technical report, Forrester, Março 2018. Acedido a 30-11-2018.

[6] Lauren E. Nelson. The Forrester Wave™: Cloud Cost Monitoring And Optimization, Q2 2018 The Nine Providers That Matter Most And How They Stack Up. Technical report, Forrester Research, 2018. Acedido a 02-12-2018.

[7] N J Manson. Is operations research really research? 2006. Volume: 2, Páginas: 155-180.

[8] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A Design Science Research Methodology for Information Systems Research. Technical report, Journal of Management Information Systems, 2007.

[9] Alan Hevner and Samir Chatterjee. *Design Research in Information Systems*. Springer US, 1st edition, 2004.

[10] Vijay Vaishnavi, Bill Kuechler, and Stacie Petter. Design Science Research in Information Systems. Technical report, Association for Information Systems, 2017.

[11] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo. Cloud computing: An overview. 2009.

[12] Anthony T Velte, Toby J Velte, and Robert Elsenpeter. *Cloud Computing: A Practical Approach*. McGraw-Hill Companies, 2010. ISBN 9780071626958.

[13] Mario Höfer, Gernot Howanitz, O Univ, and Wolfgang Pree. The Client Side of Cloud Computing. Technical report, 2009.

[14] T. Rodmunkong, P. Wannapiroon, and P. Nilsook. The Challenges of Cloud Computing Management Information System in Academic Work. *International Journal of Signal Processing Systems*, 2014.

[15] Katie Costello and Sarah Hippold. Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.3 Percent in 2019. https://www.gartner.com/en/newsroom/press-releases/2018-09-12-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2019, 2018. Acedido a 23-11-2018.

[16] Lydia Leong, Gregor Petri, Bob Gill, and Mike Dorosh. Magic Quadrant for Cloud Infrastructure as a Service, Worldwide. https://www.gartner.com/doc/3400818/magic-quadrant-cloud-infrastructure-service, 2016. Acedido a 23-11-2018.

[17] Lydia Leong, Raj Bala, Craig Lowery, and Dennis Smith. Magic Quadrant for Cloud Infrastructure as a Service, Worldwide. https://www.gartner.com/doc/3738058/magic-quadrant-cloud-infrastructure-service, 2017. Acedido a 23-11-2018.

[18] Dennis Smith, Lydia Leong, and Raj Bala. Magic Quadrant for Cloud Infrastructure as a Service, Worldwide. https://www.gartner.com/doc/3875999/magic-quadrant-cloud-infrastructure-service, 2018. Acedido a 23-11-2018.

[19] David Wright, Dennis Smith, Raj Bala, and Bob Gill. Magic Quadrant for Cloud Infrastructure as a Service, Worldwide. https://www.gartner.com/en/documents/3947472/magic-quadrant-for-cloud-infrastructure-as-a-service-wor. Acedido a 25-07-2019.

[20] Gartner. Gartner Magic Quadrant. https://www.gartner.com/en/research/methodologies/magic-quadrants-research, 2018. Acedido a 22-11-2018.

[21] Ahmed Youssef, Abdulelah Almishal, and Ahmed E Youssef. Cloud Service Providers: A Comparative Study. *International Journal of Computer Applications & Information Technology*, 5:2278–7720, 2014. Acedido a 30-11-2018.

[22] Andreas Wittig and Michael Wittig. *Amazon Web Services in Action*. Manning Publications Co., 1st edition, 2015. ISBN 9781617292880.

[23] Cynthia Harvey and Andy Patrizio. AWS vs. Azure vs. Google: Cloud Comparison. https://www.datamation.com/cloud-computing/aws-vs.-azure-vs.-google-cloud-comparison.html, 2017. Acedido a 28-11-2018.

[24] Michael Collier and Robin Shahan. *Fundamentals of Azure*. Microsoft Azure, 2nd edition, 2016.

[25] JJ Geewax. *Google Cloud Platform in Action*. Manning Publications Co., 1st edition, 2018.

[26] Christof Weinhardt, Arun Anandasivam, Benjamin Blau, Nikolay Borissov, Thomas Meinl, Wibke Michalk, and Jochen Stößer. Cloud Computing – A Classification, Business Models, and Research Directions. *Business & Information Systems Engineering*, 2009.

[27] Einar Iveroth, Alf Westelius, Carl Johan Petri, Nils Göran Olve, and Fredrik Nilsson. How to differentiate by price: Proposal for a five-dimensional model. *European Management Journal*, 2013.

[28] Alexander Osterwalder. *The Business Model Ontologya Proposition in a Design Science Approach*. PhD thesis, University of Lausanne, 2004.

[29] May Al-Roomi, Shaikha Al-Ebrahim, Sabika Buqrais, and Imtiaz Ahmad. Cloud Computing Pricing Models: A Survey. *International Journal of Grid and Distributed Computing*, 2013.

[30] Bhanu Sharma, Ruppa K. Thulasiram, Parimala Thulasiraman, Saurabh K. Garg, and Rajkumar Buyya. Pricing cloud compute commodities: A novel financial economic model. In *Proceedings - 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2012.

[31] Artan Mazrekaj, Isak Shabani, and Besmir Sejdiu. Pricing Schemes in Cloud Computing: An Overview. *International Journal of Advanced Computer Science and Applications*, 2016.

[32] An Introduction to Cloud Pricing. https://www.parkmycloud.com/cloud-pricing/, 2018. Acedido a 29-11-2018.

[33] Microsoft. Azure vs. AWS - Comparação de Custos. https://azure.microsoft.com/pt-pt/overview/azure-vs-aws/cost-savings/. Acedido a 06-12-2018.

[34] Amazon Web Services. AWS Cost Explorer. https://aws.amazon.com/aws-cost-management/aws-cost-explorer/?nc1=h_ls, 2019. Acedido a 09-03-2019.

[35] Microsoft Azure. Azure Cost Management. https://azure.microsoft.com/pt-pt/services/cost-management/, 2019. Acedido a 15-03-2019.

[36] Google Cloud. Cost management. https://cloud.google.com/cost-management/, 2019. Acedido a 18-03-2019.

[37] Apptio. IT Financial Transparency, Benchmarking and Metrics. https://www.apptio.com/. Acedido a 05-12-2018.

[38] Cloudability. Cloud Cost Management, Efficiency and Optimization. https://www.cloudability.com/. Acedido a 07-12-2018.

[39] CloudCheckr. Cloud Management Platform - CloudCheckr. https://cloudcheckr.com/platform/. Acedido a 02-12-2018.

[40] CloudHealth Technologies. Cloud Management & Optimization. https://www.cloudhealthtech.com/. Acedido a 12-12-2018.

[41] Densify. Next-Generation Cloud Optimization for CloudOps. https://www.densify.com/. Acedido a 14-12-2018.

[42] Cloudyn. Cloud Management Platform & Cloud Cost Optimization Tool. https://www.cloudyn.com/. Acedido a 13-12-2018.

[43] RightScale Optima. Cloud Cost Management. https://www.rightscale.com/products-and-services/products/rightscale-optima. Acedido a 06-12-2018.

[44] Teevity. Cloud costs analytics. https://www.teevity.com/. Acedido a 11-12-2018.

[45] Turbonomic. Public Cloud Management. https://turbonomic.com/. Acedido a 17-12-2018.

[46] Siliconreview Team. The Ultimate Cloud Management Platform: CloudCheckr. https://thesiliconreview.com/magazines/the-ultimate-cloud-management-platform-cloudcheckr/, 2017. Acedido a 30-11-2018.

[47] CloudAware. The Most Complete Cloud Management Platform. https://www.cloudaware.com/. Acedido a 11-12-2018.

[48] CloudAware. Cost Management. Technical report, 2018. Páginas: 1-5.

[49] Roy Thomas Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. Acedido a 13-07-2019.

[50] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0, 1996.

[51] Douglas Crockford. The application/json Media Type for JavaScript Object Notation (JSON). Technical report, IETF, 2006.

[52] Amazon Web Services. Understanding Your Usage with Billing Reports. https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/billing-reports.html, 2019. Acedido a 05-02-2019.

[53] Amazon Web Services. Creating an AWS Cost and Usage Report. https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/billing-reports-gettingstarted-turnonreports.html, 2019. Acedido a 05-02-2019.

[54] Microsoft Azure. Create and manage exported data. https://docs.microsoft.com/en-us/azure/cost-management/tutorial-export-acm-data, 2019. Acedido a 13-02-2019.

[55] Google Cloud. Creating storage buckets. https://cloud.google.com/storage/docs/creating-buckets, 2019. Acedido a 20-02-2019.

[56] Google Cloud. Export Billing Data to a File. https://cloud.google.com/billing/docs/how-to/export-data-file, 2019. Acedido a 20-02-2019.

[57] Amazon Web Services. Billing and Cost Management API Reference. https://docs.aws.amazon.com/aws-cost-management/latest/APIReference/Welcome.html, 2019. Acedido a 04-03-2019.

[58] Amazon Web Services. AWS Cost Explorer Service. https://docs.aws.amazon.com/aws-cost-management/latest/APIReference, 2019. Acedido a 04-03-2019.

[59] Amazon Web Services. Amazon CloudWatch. https://aws.amazon.com/pt/cloudwatch/, 2019. Acedido a 11-03-2019.

[60] Amazon Web Services. Amazon CloudWatch API. https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference, 2019. Acedido a 11-03-2019.

[61] Microsoft Azure. Reporting APIs for Enterprise customers. https://docs.microsoft.com/en-us/azure/billing/billing-enterprise-api, 2019. Acedido a 19-03-2019.

[62] Microsoft Azure. Azure Monitor. https://azure.microsoft.com/pt-pt/services/monitor/, 2019. Acedido a 19-03-2019.

[63] Microsoft Azure. Azure Monitor REST API. https://docs.microsoft.com/en-us/azure/azure-monitor/platform/rest-api-walkthrough, 2019. Acedido a 20-03-2019.

[64] Google Cloud. Cloud Billing API. https://cloud.google.com/billing/reference/rest/, 2019. Acedido a 25-03-2019.

[65] Google Cloud. Google Stackdriver. https://cloud.google.com/stackdriver/, 2019. Acedido a 26-03-2019.

[66] Google Cloud. Stackdriver Monitoring API. https://cloud.google.com/monitoring/api/v3/, 2019. Acedido a 26-03-2019.

[67] Amazon Web Services. AWS Tagging Strategies. https://aws.amazon.com/pt/answers/account-management/aws-tagging-strategies/, 2019. Acedido a 04-02-2019.

[68] Microsoft Azure. Use tags to organize your Azure resources. https://docs.microsoft.com/pt-pt/azure/azure-resource-manager/resource-group-using-tags, 2019. Acedido a 01-02-2019.

[69] Google Cloud. Creating and Managing Labels. https://cloud.google.com/resource-manager/docs/creating-managing-labels, 2019. Acedido a 09-03-2019.

[70] Kim Weins. Better cloud management through cloud resource tagging. https://www.infoworld.com/article/3246986/better-cloud-management-through-cloud-resource-tagging.html, 2018. Acedido a 04-02-2019.

[71] Gavin Cahill. 5 Best Practices for Building a Cloud Tagging Strategy. https://blog.cloudability.com/cloud-tagging-best-practices/, 2018. Acedido a 02-02-2019.

[72] Emily Wanless. 8 Tag Categories You Must Include In Your Cloud Tagging Strategy. https://www.cloudhealthtech.com/blog/8-tag-categories-you-must-include-your-cloud-tagging-strategy, 2017. Acedido a 02-02-2019.

[73] Mike Mackrory. AWS Tagging Best Practices – The Ultimate Guide. https://www.metricly.com/aws-tagging-best-practices/, 2019. Acedido a 04-02-2019.

[74] HashiCorp. Terraform. https://www.terraform.io/. Acedido a 12-07-2019.

[75] Chef. Chef: Deploy new code faster and more frequently. https://www.chef.io, 2019. Acedido a 18-07-2019.

[76] Pallets. Flask Documentation. https://palletsprojects.com/p/flask/, 2019. Acedido a 12-05-2019.

[77] Elastic. What is the ELK Stack? https://www.elastic.co/pt/what-is/elk-stack, 2019. Acedido a 04-04-2019.

[78] Margaret Rouse. Elastic Stack. https://searchitoperations.techtarget.com/definition/Elastic-Stack. Acedido a 04-04-2019.

[79] Yelp. ElastAlert - Easy & Flexible Alerting With Elasticsearch. https://elastalert.readthedocs.io/en/latest/. Acedido a 20-07-2019.

[80] Elastic. Logstash: Collect, Parse, Transform Logs. https://www.elastic.co/pt/products/logstash, 2019. Acedido a 06-04-2019.

[81] Elastic. Elasticsearch: RESTful, Distributed Search & Analytics. https://www.elastic.co/pt/products/elasticsearch, 2019. Acedido a 07-04-2019.

[82] Andrew Lin. Elasticsearch Architectural Overview. https://buildingvts.com/elasticsearch-architectural-overview-a35d3910e515. Acedido a 12-04-2019.

[83] Victor Melo. A Practical Introduction to Elasticsearch with Kibana. https://medium.com/@victorsmelopoa/an-introduction-to-elasticsearch-with-kibana-78071db3704, 2018. Acedido a 19-04-2019.

[84] Bo Andersen. Understanding the Inverted Index in Elasticsearch. https://codingexplained.com/coding/elasticsearch/understanding-the-inverted-index-in-elasticsearch, 2018. Acedido a 09-04-2019.

[85] Elastic. Kibana: Your window into the Elastic Stack. https://www.elastic.co/products/kibana, 2019. Acedido a 22-04-2019.

[86] Elastic. Kibana features list. https://www.elastic.co/pt/products/kibana/features, 2019. Acedido a 21-04-2019.

[87] McKinney Wes. *Python for Data Analysis*. O'Reilly Media, Inc., 1 edition, 2012.

[88] Vinton G. Cerf and Robert E. Khan. A Protocol for Packet Network Intercommunication. *IEEE TRANSACTIONS ON COMMUNICATIONS*, 22:637–648, 1974.

[89] Python Software Foundation. Abstract Base Classes. https://docs.python.org/3/library/abc.html, 2019. Acedido a 18-07-2019.

[90] Amazon Web Services. Boto 3 Documentation. https://boto3.amazonaws.com/v1/documentation/api/latest/index.html, . Acedido a 24-04-2019.

[91] Microsoft Azure. Azure SDK for Python. https://azure.github.io/azure-sdk-for-python/, 2019. Acedido a 30-04-2019.

[92] Google Cloud. Python on Google Cloud Platform. https://cloud.google.com/python/, 2019. Acedido a 07-05-2019.

[93] Flask. Modular Applications with Blueprints. https://flask.palletsprojects.com/en/1.1.x/blueprints/, 2019. Acedido a 12-05-2019.

[94] Docker Inc. Enterprise Container Platform. https://www.docker.com. Acedido a 20-07-2019.

[95] Amazon Web Services. Consolidated Billing for Organizations. https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/consolidated-billing.html, 2019. Acedido a 21-05-2019.

[96] Amazon Web Services. Amazon ElastiCache. https://aws.amazon.com/elasticache/, 2019. Acedido a 10-06-2019.

[97] Amazon Web Servcies. Amazon S3 Storage Classes. https://aws.amazon.com/s3/storage-classes/, 2019. Acedido a 10-06-2019.

[98] Amazon Web Services. Choosing a Routing Policy. https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routing-policy.html, . Acedido a 12-08-2019.

A

Material de suporte

A.1 Modelação UML

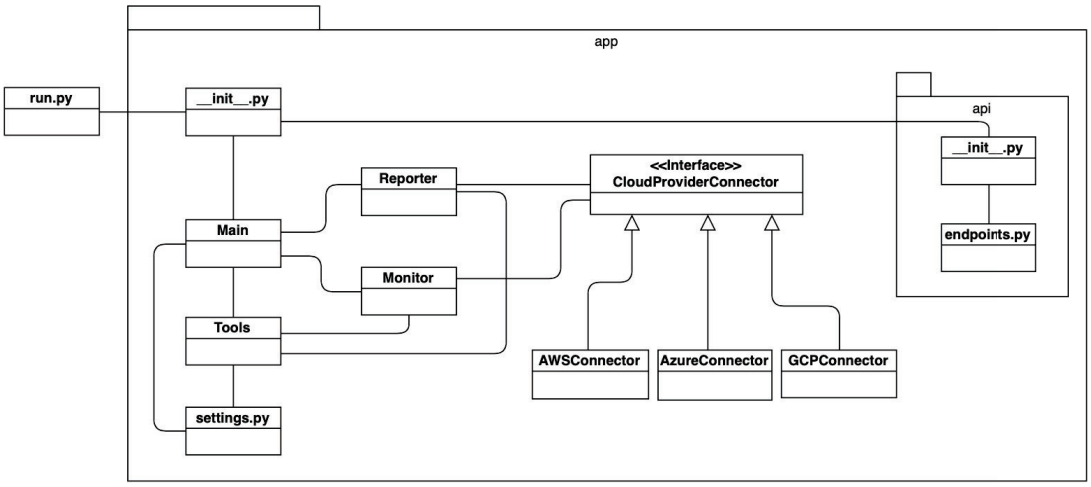


Figura 42: Diagrama de arquitetura da solução

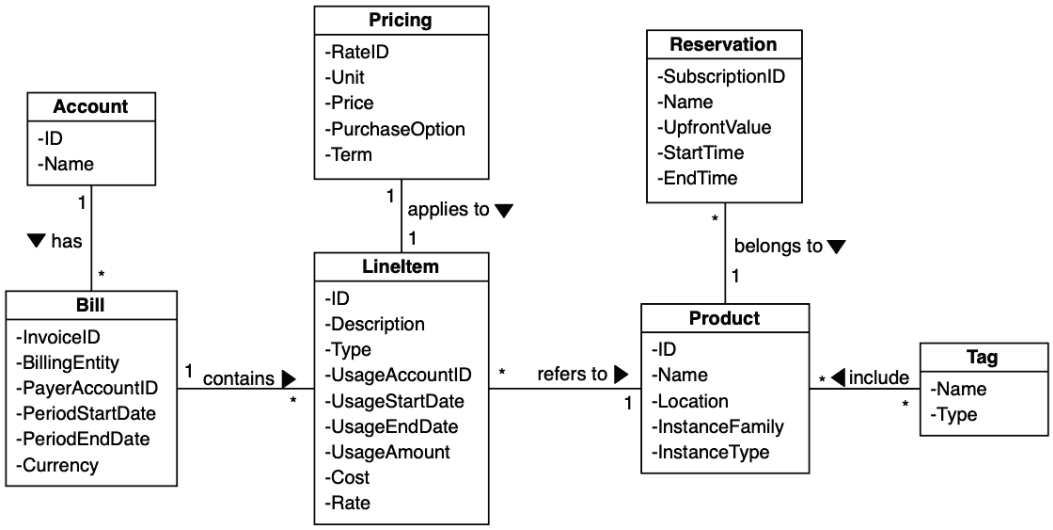


Figura 43: Modelo de dados de custos de operação

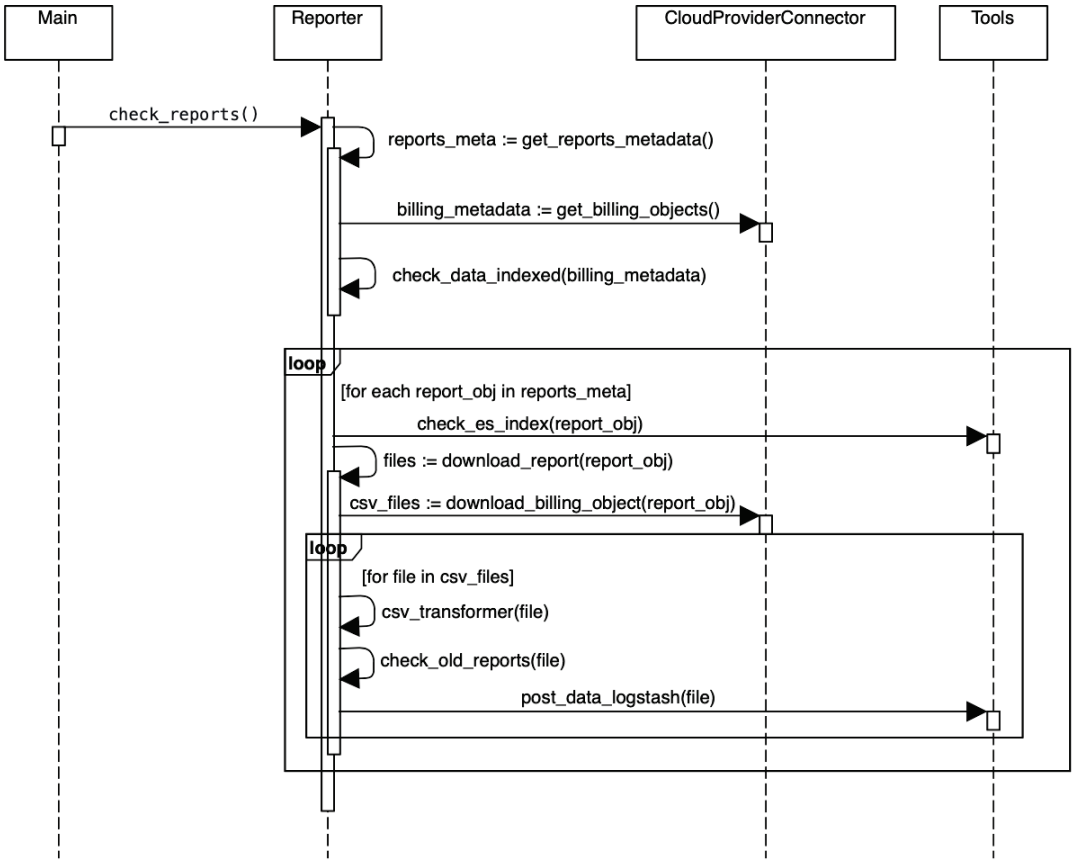
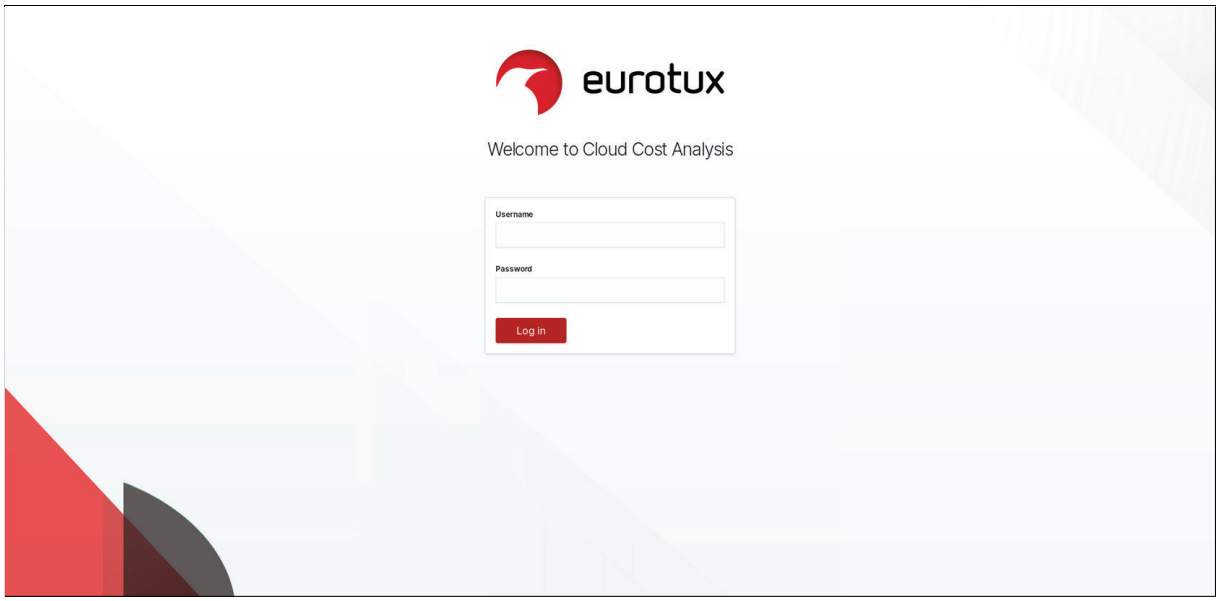
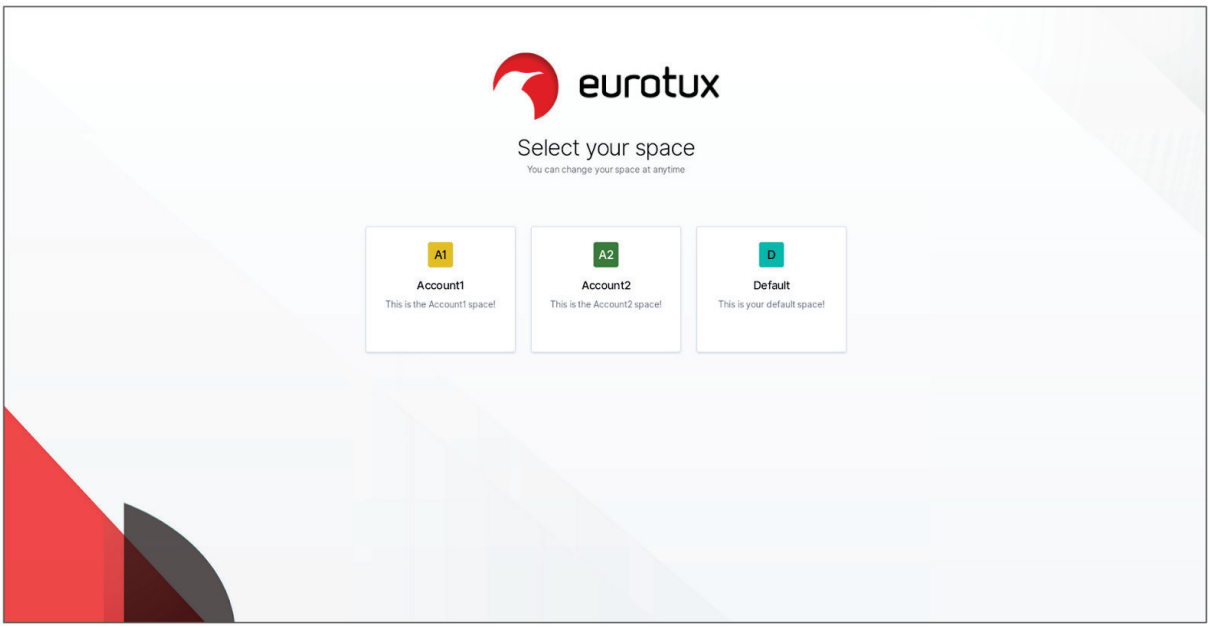


Figura 44: Diagrama de Sequência para obtenção de custos de operação

A.2 Customização do Kibana



(a) *Kibana Login Page*



(b) *Kibana Space Selector Page*

Figura 45: Customização da Interface do Kibana

A.3 Dashboards do Caso de Estudo 1

Figura 46: *AWS Billing-General Dashboard*

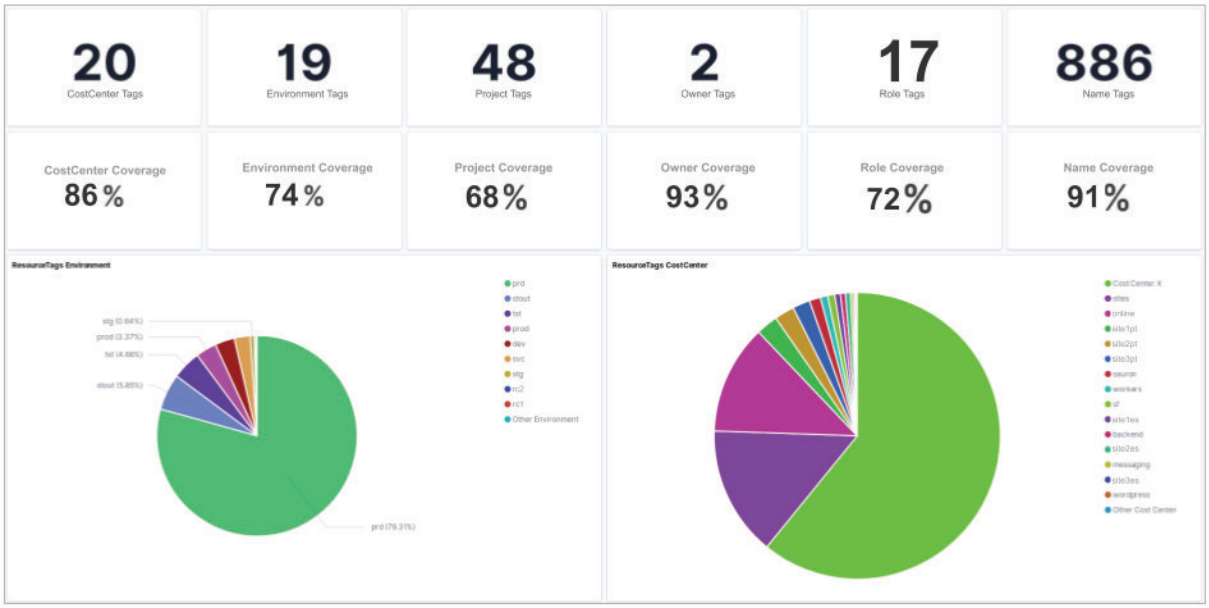


Figura 47: *AWS Billing-Tags Dashboard*

Figura 48: *AWS Billing-Networking Dashboard*

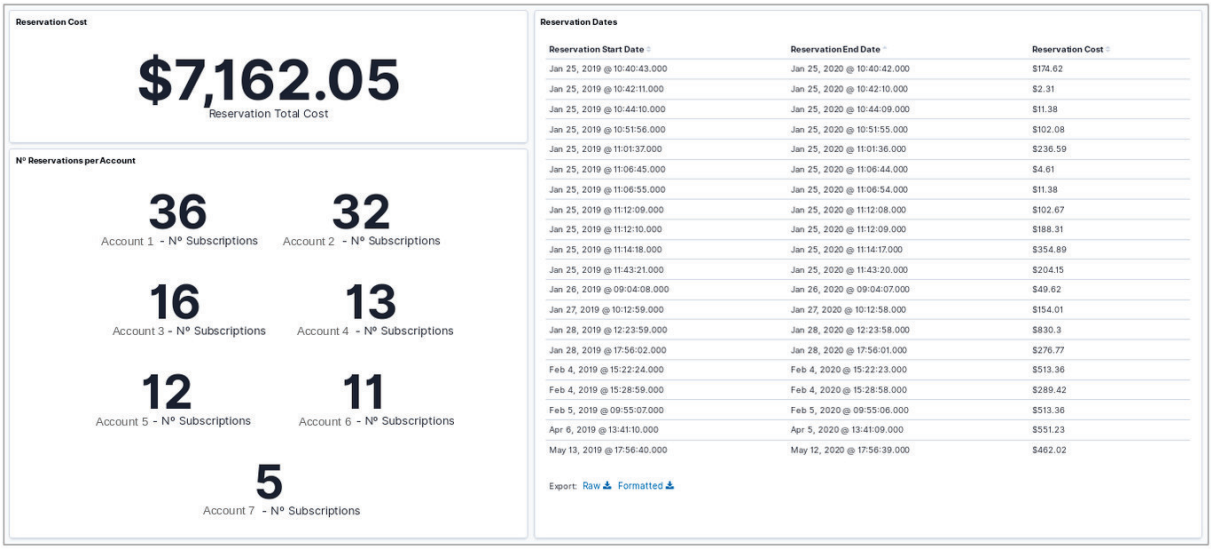


Figura 49: *AWS Billing-Reserved Dashboard*

Figura 50: *AWS Billing-EC2 Dashboard*

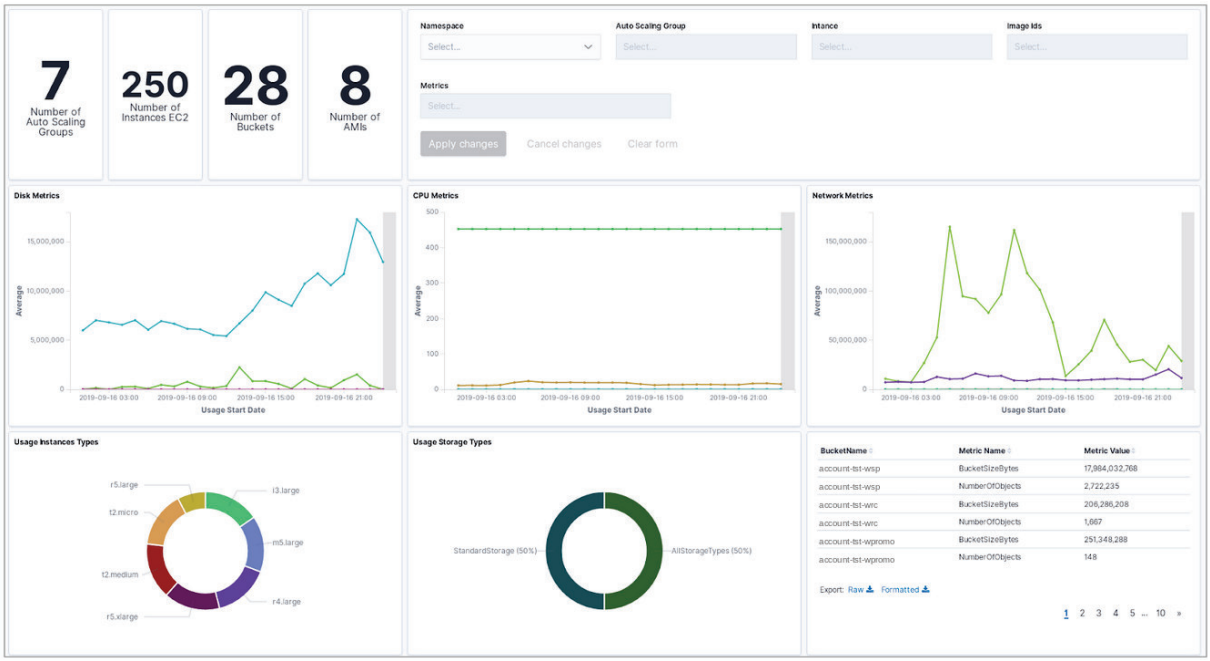


Figura 51: *AWS Usage Dashboard*

A.4 Dashboards do Caso de Estudo 3

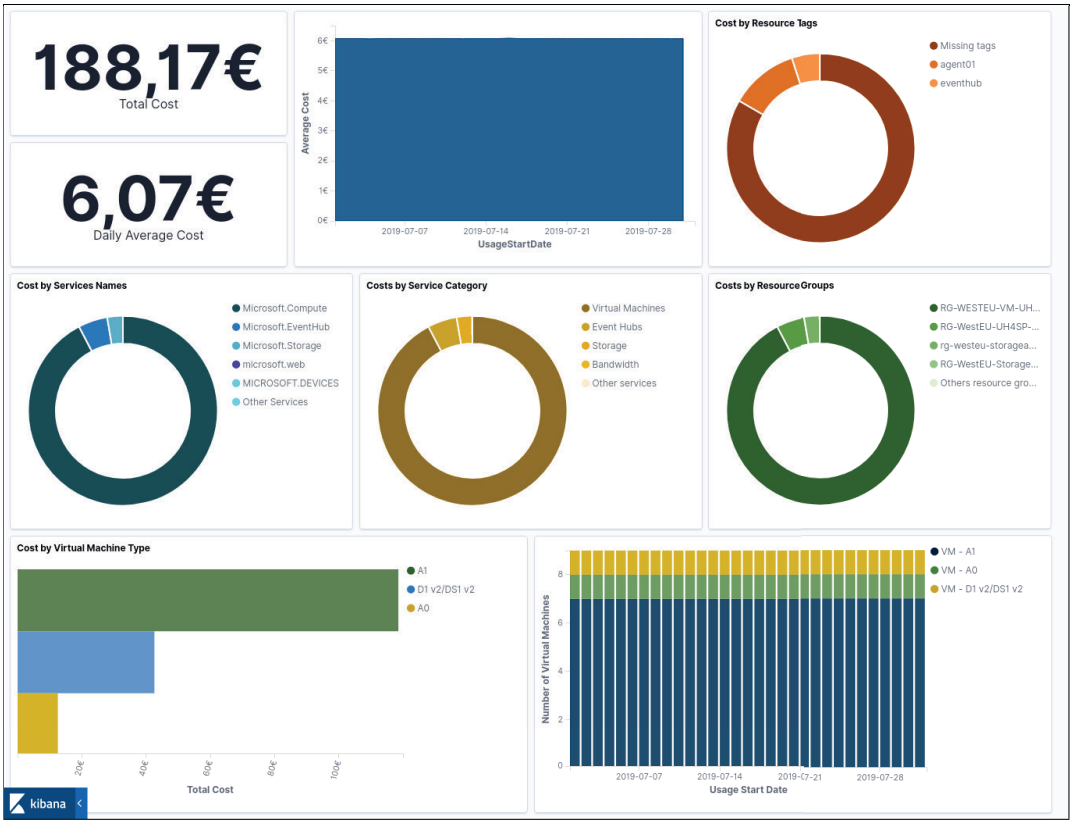


Figura 52: *Azure Billing Dashboard*

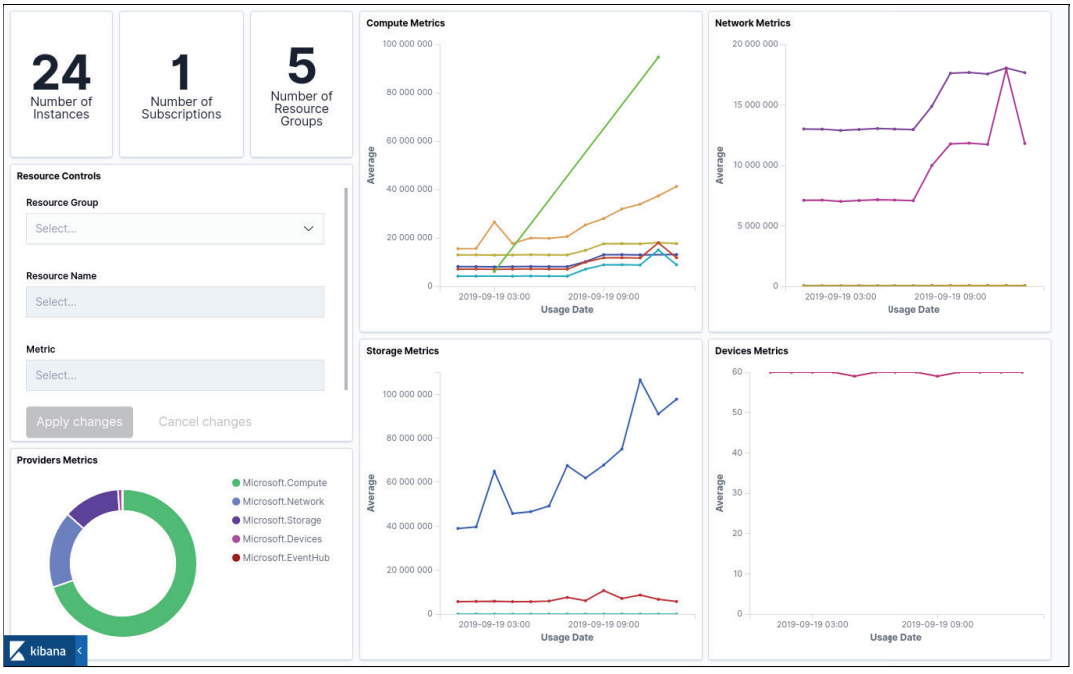


Figura 53: *Azure Usage Dashboard*

A.5 Dashboards do Caso de Estudo 4

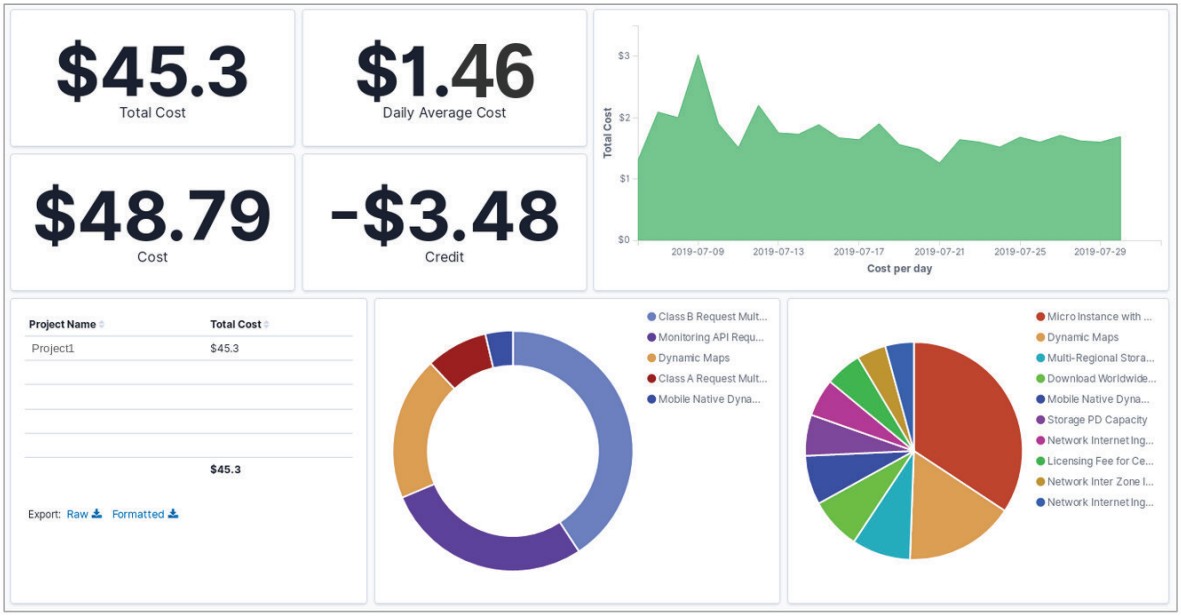


Figura 54: *GCP Billing Dashboard*

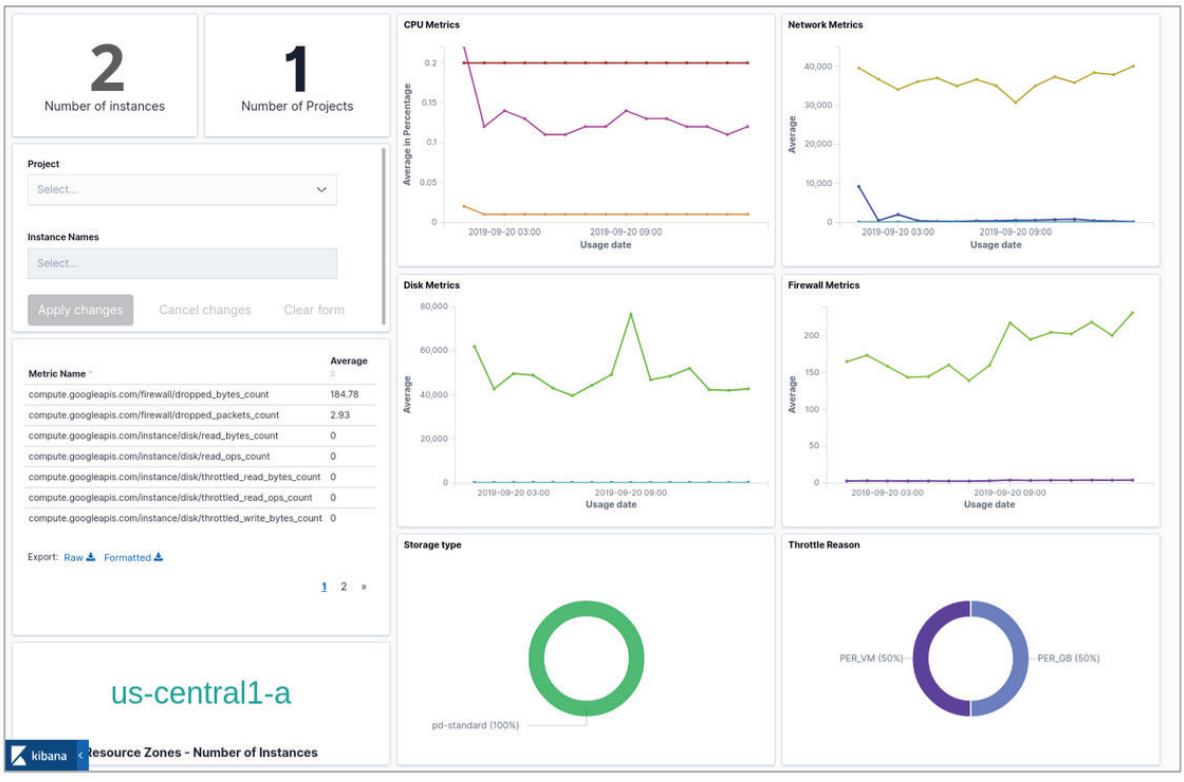


Figura 55: *GCP Usage Dashboard*

B

Anexos de Código

B.1 Cloud Cost Analysis

```
{
  "CloudProviders": {
    "AWS": {
      "Accounts": [
        {
          "Name": "Account1",
          "Profile": "default",
          "Region": "eu-west-2",
          "S3_Bucket_Name": "billingreports",
          "S3_Report_Path": "/quickreports",
          "S3_Report_Name": "quickreports",
          "Report_Type": "cur",
          "Cloudwatches": [
            {
              "Profile": "default",
              "Region": "eu-west-3"
            }
          ]
        }
      ],
      "CredentialsPath": "/config/credentials.yaml",
      "KeyTransformerPath": "src/app/resources/AWSKeyTransformer.yaml",
      "MapPropertiesPath": "src/app/resources/AWSMapProperties.yaml"
    },
    "AZURE": {
      "Accounts": [
```

```
        {
          "Name": "Account2",
          "Storage": {
            "ReportName": "DailyBillingExport",
            "AccountName": "billing",
            "ContainerName": "acm",
            "ContainerDirectory": "reports",
            "Granularity": "Monthly"
          }
        }
      ],
      "CredentialsPath": "/config/azure_credentials.json",
      "KeyTransformerPath": "src/app/resources/AzureKeyTransformer.yaml",
      "MapPropertiesPath": "src/app/resources/AzureMapProperties.yaml"
    },
    "GCP": {
      "Accounts": [
        {
          "Name": "Account3",
          "Bucket_Name": "billing-reports",
          "Report_Prefix": "billing",
          "Active": false
        }
      ],
      "CredentialsPath": "/config/keys.json",
      "KeyTransformerPath": "src/app/resources/GCPKeyTransformer.yaml",
      "MapPropertiesPath": "src/app/resources/GCPMapProperties.yaml"
    }
  },
  "ElasticStack": {
    "Kibana": {
      "URL": "http://kibana:5601",
      "Username": "elastic",
      "Password": "changeme"
    },
    "Logstash": {
      "Host": "logstash",
      "Port": 5140
    },
    "Elasticsearch": {
      "URL": "http://elasticsearch:9200",
      "Username": "elastic",
      "Password": "changeme"
    }
  }
}
```

Exemplo de código B.1: Exemplo de ficheiro de configuração

```
{
    "$id": "configSchema",
    "type": "object",
    "required": ["CloudProviders", "ElasticStack"],
    "properties": {
        "CloudProviders": {
            "$id": "CloudProviders",
            "type": "object",
            "anyOf": [{"required":["AWS"]}, {"required":["AZURE"]},
                      {"required":["GCP"]}],
            "properties": {
                "AWS": {
                    "$id": "AWS",
                    "type": "object",
                    "required": [ "Accounts", "CredentialsPath",
                        "KeyTransformerPath", "MapPropertiesPath"
                    ],
                    "properties": {
                        "Accounts": {
                            "$id": "Accounts",
                            "type": "array",
                            "items": {
                                "$id": "Accounts/items",
                                "type": "object",
                                "title": "The Items Schema",
                                "required": ["Name", "Profile", "Region",
                                    "S3_Bucket_Name", "S3_Report_Path",
                                    "S3_Report_Name", "Report_Type"],
                                "properties": {
                                    "Name": {
                                        "$id": "Accounts/Name",
                                        "type": "string",
                                        "examples": ["AccountName"]
                                    },
                                    ...
                                }
                            }
                        },
                        "CredentialsPath": {
                            "$id": "AWS/properties/CredentialsPath",
                            "type": "string",
                            "examples": ["/credentials"]
                        },
                        "KeyTransformerPath": {
                            "$id": "AWS/properties/KeyTransformerPath",
                            "type": "string",
                            "examples": ["/AWSKeyTransformer.yaml"]
                        },
                        "MapPropertiesPath": {
                            "$id": "AWS/properties/MapPropertiesPath",
```

```
                                "type": "string",
                                "examples": ["/AWSMapProperties.yaml"]
                            }
                        }
                    },
                    "AZURE": ... ,
                    "GCP": ...
                }
            },
            "ElasticStack": {
                "$id": "ElasticStack",
                "type": "object",
                "properties": {
                    "Kibana": {
                        "$id": "ElasticStack/Kibana",
                        "type": "object",
                        "required": ["URL", "Username", "Password"],
                        "properties": ...
                    },
                    "Logstash": {
                        "$id": "ElasticStack/Logstash",
                        "type": "object",
                        "required": ["Host", "Port"],
                        "properties": ...
                    },
                    "Elasticsearch": {
                        "$id": "ElasticStack/Elasticsearch",
                        "type": "object",
                        "required": ["URL", "Username", "Password"],
                        "properties": ...
                    }
                }
            }
        }
}
```

Exemplo de código B.2: Excerto do *schema* de configuração

```
"InvoiceID": "bill/InvoiceId"
"PayerAccountId": "bill/PayerAccountId"
"BillingPeriodStartDate": "bill/PeriodStartDate"
"BillingPeriodEndDate": "bill/PeriodEndDate"
"Currency": "bill/Currency"
"RecordId": "lineItem/Id"
"ItemDescription": "lineItem/Description"
"RecordType": "lineItem/Type"
"LinkedAccountId": "lineItem/UsageAccountId"
"UsageStartDate": "lineItem/UsageStartDate"
"UsageEndDate": "lineItem/UsageEndDate"
"UsageQuantity": "lineItem/UsageAmount"
"UnBlendedCost": "lineItem/Cost"
"UnBlendedRate": "lineItem/Rate"
"RateId": "pricing/RateId"
"ReservedInstance": "pricing/PurchaseOption"
"UnitOfMeasure": "pricing/Unit"
"UnitPrice": "pricing/Price"
"Term": "pricing/term"
"ProductCode": "product/Id"
"ProductName": "product/Name"
"Location": "product/Location"
"instanceFamily": "product/InstanceFamily"
"instanceType": "product/instanceType"
"SubscriptionId": "reservation/SubscriptionId"
"ReservationARN": "reservation/Name"
"ReservationStartTime": "reservation/StartTime"
"ReservationEndTime": "reservation/EndTime"
"user:Environment": "tags/user:Environment"
```

Exemplo de código B.3: Exemplo de um *Key Transformer*

```
"bill/PeriodStartDate": "date"
"bill/PeriodEndDate": "date"
"lineItem/Cost": "float"
"lineItem/Rate": "float"
"lineItem/UsageAmount": "float"
"lineItem/UsageStartDate": "date"
"lineItem/UsageEndDate": "date"
"pricing/Price": "float"
"product/Location": "geo_point"
"reservation/UpfrontValue": "float"
```

Exemplo de código B.4: Exemplo de um *Map Properties*

```
{
    "index_patterns" : ["billing-*"],
    "settings" : {
      "index" : {
        "number_of_replicas" : "0",
        "refresh_interval" : "10s"
      }
    },
    "mappings" : {
      "dynamic_templates" : [{
        "string_fields" : {
            "mapping" : {
              "copy_to" : "line",
              "type" : "keyword",
              "fields" : {
                "raw" : {
                  "ignore_above" : 256,
                  "type" : "keyword"
                }
              }
            },
            "match_mapping_type" : "string",
            "match" : "*"
        }
      }],
      "properties" : {
        "lineItem" : {
          "properties" : {
            "vcpu" : {
              "type" : "integer"
            },
            "geoLocation" : {
              "type" : "geo_point"
            },
            "UnblendedCost" : {
              "type" : "float"
            }
          }
        }
      }
   }
}
```

Exemplo de código B.5: Exemplo de um *Index Template*

B.2 Logstash

```
# encoding: utf-8
require "logstash/inputs/file"
require "logstash/namespace"
require "csv"

class LogStash::Inputs::Csv < LogStash::Inputs::File
 config_name "csv"

 # Define a list of column names (in the order they appear in the CSV).
 config :columns, :validate => :array, :default => []
 # Bool flag enables sourcing of column names from first line of each file.
 config :header, :validate => :boolean, :default => false
 # Define the column separator value. The default separator is a comma ','.
 config :separator, :validate => :string, :default => ","
 # Define the character used to quote CSV fields. Default: double quote.
 config :quote_char, :validate => :string, :default => '"'
 # Define whether empty columns should be skipped.
 config :skip_empty_columns, :validate => :boolean, :default => false

 public
 def register
  @fileColumns = Hash.new
  super()
 end

 def run(queue)
  super(queue)
 end # def run

 def decorate(event)
  super(event)
  message = event.get("message")
  return if !message

  begin
   values = CSV.parse_line(message,:col_sep=>@separator,:quote_char=>
    @quote_char)
   return if values.length == 0

   if @header
    cols = getSchemaForFile(event, values)
   else
    cols = @columns
   end
```

```
44      if !event.get("csvHeader")
45       values.each_index do |i|
46        unless (@skip_empty_columns && (values[i].nil? || values[i].empty?))
47          field_name = cols[i] || "column#{i+1}"
48          event.set(field_name, values[i])
49        end
50      end
51    end
52    rescue => e
53      return event.tag "_csvparsefailure"
54    end # begin
55  end # decorate()
56
57  def getSchemaForFile(event, parsedValues)
58    path = event.get("path")
59    if !path
60      return []
61    end
62
63    csvLine = readSchemaLineFromFile(path)
64    if !csvLine || csvLine.length == 0
65      return []
66    end
67
68    schema = CSV.parse_line(csvLine,:col_sep=>@separator,:quote_char=>
        @quote_char)
69    @fileColumns[path] = schema
70    if @fileColumns[path].join == parsedValues.join
71      event.set("csvHeader", true)
72      return []
73    else
74      return schema
75    end
76  end
77
78  def readSchemaLineFromFile(path)
79    csvLine = ""
80    File.open(path, "r") do |f|
81      while csvLine.length == 0 and csvLine = f.gets
82      end
83    end
84    csvLine
85  end
86 end # class LogStash::Inputs::Csv
```

Exemplo de código B.6: *Logstash CSV Input Plugin*

```
input {
  csv {
    path => "/usr/share/logstash/data/delete_*.csv"
    tags => ["delete"]
    sincedb_path => "/dev/null"
    start_position => "beginning"
    mode => "read"
    header => "true"
    skip_empty_columns => "true"
  }

  csv {
    path => "/usr/share/logstash/data/*.csv"
    sincedb_path => "/dev/null"
    start_position => "beginning"
    mode => "read"
    header => "true"
    skip_empty_columns => "true"
  }
}

filter {
  if ([csvHeader]) {
    # Drop event if it's mark with "csvHeader" flag.
      drop{}
  } else {
    # Create index according to filename in path
    grok {
      match => {"path"=>"-%{INT:year}-%{INT:month}-%{INT:order}-%{WORD:
          account}"}
      add_field => ["index", "billing-%{account}-%{year}-%{month}"]
    }

    if [identity/TimeInterval] {
      mutate {
        add_field => { "[@metadata][doc_id]" => "%{identity/LineItemId}-%{
            lineItem/UsageStartDate}" }
        remove_field => ["identity/LineItemId", "identity/TimeInterval"]
      }
    }

    if [lineItem/LineItemId] {
      if [lineItem/LineItemId] != "0" {
        mutate { add_field =>{ "[@metadata][doc_id]"=>"%{lineItem/
            LineItemId}" }}
      } else {
        mutate { add_field => { "[@metadata][doc_id]" => "%{lineItem/
            UsageAccountId}-%{lineItem/UsageStartDate}" } }
      }
    } else {
```

```
47        if [lineItem/LineItemType] in ["Rounding", "InvoiceTotal","
              AccountTotal"] {
48          drop{}
49        }
50      }
51
52      # Remove default fields created by Logstash and auxiliars fields.
53      mutate {
54        remove_field => [ "message", "path",  "year", "month", "account", "
              order"]
55      }
56
57      if "delete" in [tags] {
58        mutate { add_field => { "[@metadata][action]" => "delete" } }
59      } else {
60        mutate { add_field => { "[@metadata][action]" => "update" } }
61      }
62
63      # Convert data fields to date type
64      date {
65        match => [ "lineItem/UsageStartDate", "yyyy-MM-dd HH:mm:ss", "ISO8601
              " ]
66        timezone => "Etc/UTC"
67        target => "lineItem/UsageStartDate"
68      }
69
70      # Create object with aggregate info about resource tags
71      if [resourceTags] {
72        json {
73          source => "resourceTags"
74          target => "resourceTags"
75        }
76      }
77
78      # Create geopoints to product regions
79      translate {
80        field => "product/location"
81        destination => "product/geoLocation"
82        dictionary_path => "/usr/share/logstash/pipeline/resources/geopoints.
              yaml"
83      }
84
85      # Split fields according to parent category.
86      ruby {
87        code => ' begin
88          keys = event.to_hash.keys
89          keys.each{|key|
90            if ( key =~ /\// )
91              res = key.split("/")
92              field = res[0]
93              subfield = res[1]
```

```
94              if ( field == "resourceTags" )
95                tags = subfield.split(":")
96                event.set("[#{field}][#{tags[0]}][#{tags[1]}]",event.remove(
                      key))
97              else
98                event.set("[#{field}][#{subfield}]", event.remove(key))
99              end
100           end
101         }
102       end'
103     }
104   }
105 }
106
107 output {
108     if [@metadata][action] == "delete" {
109         elasticsearch {
110             hosts => [ "elasticsearch:9200" ]
111             index   => "%{index}"
112             doc_id => "%{[@metadata][doc_id]}"
113             action => "delete"
114             user => "${ES_USER}"
115             password => "${ES_PASS}"
116         }
117     } else {
118         if [@metadata][doc_id] {
119             elasticsearch {
120                 hosts => [ "elasticsearch:9200" ]
121                 index   => "%{index}"
122                 doc_id => "%{[@metadata][doc_id]}"
123                 doc_as_upsert => true
124                 action => "update"
125                 user => "${ES_USER}"
126                 password => "${ES_PASS}"
127             }
128         } else {
129             elasticsearch {
130                 hosts => [ "elasticsearch:9200" ]
131                 index   => "%{index}"
132                 user => "${ES_USER}"
133                 password => "${ES_PASS}"
134             }
135         }
136     }
137 }
```

Exemplo de código B.7: *Logstash Billing Pipeline*

```
input {
    tcp {
        port => "${TCP_PORT:5140}"
        codec => json_lines
        tags => ["tcp"]
    }
}

filter {
    date {
        timezone => "Etc/UTC"
        match => ["Timestamp" , "ISO8601", "yyyy-MM-dd HH:mm:ss.SSS"]
        target => "@timestamp"
    }

    grok {
        match => ["@timestamp", "%{YEAR:year}-%{MONTHNUM2:month}-"]
        add_field => ["[@metadata][index]", "usage-%{account}-%{year}-%{
            month}"]
    }

    mutate {remove_field=>["host","port","account","year","month","
        Timestamp"]}

    ruby {
        code =>
            'begin
                event.to_hash.keys.each{|key|
                    event.set(key.capitalize, event.remove(key))
                }
            end'
    }
}

output {
    if [@metadata][index] {
        elasticsearch {
            hosts => [ "elasticsearch:9200" ]
            index   => "%{[@metadata][index]}"
            user => "${ES_USER}"
            password => "${ES_PASS}"
        }
    }
    else {
        stdout { codec => rubydebug }
    }
}
```

Exemplo de código B.8: *Logstash Usage Pipeline*

B.3 ElasticSearch

```
# ---------------------------  Cluster -----------------------------------
# Descriptive name for your cluster:
  cluster.name: "elk-cluster"

# ----------------------------  Node ------------------------------------
# Use a descriptive name for the node:
  node.name: node-1
# Add custom attributes to the node:
  node.attr.rack: r1

# ---------------------------- Paths ------------------------------------
# Path to directory where to store the data
  path.data: /var/lib/elasticsearch
# Path to log files:
  path.logs: /var/log/elasticsearch

# ---------------------------- Memory -----------------------------------
# Lock the memory on startup:
  bootstrap.memory_lock: true

# ---------------------------- Network ----------------------------------
# Set the bind address to a specific IP (IPv4 or IPv6):
  network.host: ${ES_NETWORK_HOST}
# Set a custom port for HTTP:
  http.port: 9200

# ---------------------------- Discovery --------------------------------
# Node discover type
  discovery.type: single-node
# List of hosts to perform discovery when this node is started:
  discovery.seed_hosts: ["host1"]
# Bootstrap the cluster using an initial set of master-eligible nodes:
  cluster.initial_master_nodes: ["node-1"]

# ---------------------------- Gateway ----------------------------------
# Block initial recovery after cluster restart until N nodes are started:
  gateway.recover_after_nodes: 1

# ---------------------------- Various ----------------------------------
# X-Pack settings
  xpack.license.self_generated.type: basic
  xpack.security.enabled: true
  xpack.monitoring.collection.enabled: true
```

Exemplo de código B.9: Ficheiro de configuração do *ElasticSearch*

B.4 Kibana

```
# This setting specifies the Kibana server port to use.
  server.port: 5601

# Specifies the address where Kibana will bind. The default is localhost.
  server.host: "127.0.0.1"

# Specifies the default route when opening Kibana.
  server.defaultRoute: /app/kibana

# The maximum payload size in bytes for incoming server requests.
  server.maxPayloadBytes: 1048576

# The Kibana server's name.  This is used for display purposes.
  server.name: "kibana"

# The URLs of the Elasticsearch instances to use for all your queries.
  elasticsearch.hosts: [ "http://elasticsearch:9200" ]

# Kibana uses an index in Elasticsearch to store saved searches,
    visualizations and dashboards. Kibana creates a new index if the index
    doesn't already exist.
  kibana.index: ".kibana"

# If your Elasticsearch is protected with basic authentication, these
    settings provide the username and password that the Kibana server uses
    to operate.
  xpack.monitoring.ui.container.elasticsearch.enabled: true
  elasticsearch.username: ${ES_USERNAME}
  elasticsearch.password: ${ES_PASSWORD}

# Time in milliseconds to wait for ES to respond to pings and requests.
  elasticsearch.pingTimeout: 1500
  elasticsearch.requestTimeout: 30000

# List of Kibana client-side headers to send to Elasticsearch.
  elasticsearch.requestHeadersWhitelist: [ authorization ]

# Enables you specify a file where Kibana stores log output.
  logging.dest: stdout

# Plugins Configuration
  elastalert-kibana-plugin.serverHost: elastalert
  elastalert-kibana-plugin.serverPort: 3030

  cca_management.serverHost: cloud-cost-analysis
  cca_management.serverPort: 5000
```

Exemplo de código B.10: Ficheiro de configuração do *Kibana*

B.5 ElastAlert

```
{
  "appName": "elastalert-server",
  "port": 3030,
  "elastalertPath": "/opt/elastalert",
  "verbose": false,
  "debug": false,
  "rulesPath": {
    "relative": true,
    "path": "/rules"
  },
  "templatesPath": {
    "relative": true,
    "path": "/rule_templates"
  },
  "es_host": "elasticsearch",
  "es_port": 9200,
  "writeback_index": "elastalert_status"
}
```

Exemplo de código B.11: Ficheiro de configuração da API REST do *ElastAlert*

```
# The elasticsearch hostname and port for metadata writeback
es_host: elasticsearch
es_port: 9200

# Option basic-auth username and password for elasticsearch
es_username: ${ES_USERNAME}
es_password: ${ES_PASSWORD}

# This is the folder that contains the rule yaml files
rules_folder: rules

# The index on es_host which is used for metadata storage
writeback_index: elastalert_status

# ElastAlert rules configurations
run_every:
  hours: 1
buffer_time:
  minutes: 1
alert_time_limit:
  hours: 1
```

Exemplo de código B.12: Ficheiro de configuração do servidor do *ElastAlert*

```yaml
# Rule name and description
name: Daily average
description: "Daily Budget Control"

# Index to search, wildcard supported
index: billing-account-*

# Time field (default: @timestamp)
timestamp_field: "lineItem.UsageStartDate"

# Rule type
type: metric_aggregation

# Name of the field over which the metric value will be calculated.
metric_agg_key: lineItem.UnblendedCost

# The type of metric aggregation to perform on the metric_agg_key field.
metric_agg_type: sum

# ElastAlert will buffer results from the most recent period of time.
buffer_time:
  hours: 24

# If metric value is greater than this number, an alert will be triggered.
# This threshold is exclusive.
max_threshold: 1500

# The alert is use when a match is found.
alert:
- "slack"

# Slack webhook
slack_webhook_url: "https://hooks.slack.com/services/new/incoming-webhook"

# Slack custom message
alert_text_type: alert_text_only
alert_text: "Daily average cost was ${0:.2f} on {1}, exceeding the limit of
    {2}."
alert_text_args:
  - metric_lineItem.UnblendedCost_sum
  - lineItem.UsageStartDate
  - max_threshold
```

Exemplo de código B.13: *Daily Cost Rule*

```
# Rule name and description
name: Spot Instances
description: "5% of EC2 instances should be Spot"

# Index to search, wildcard supported
index: billing-account-*

# Time field (default: @timestamp)
timestamp_field: "lineItem.UsageStartDate"

# Rule type
type: percentage_match

buffer_time:
  hours: 24

query_key: product.ProductName

filter:
- query:
    query_string:
      query: "product.ProductName : \"Amazon Elastic Compute Cloud\""

match_bucket_filter:
- query_string:
    query: "*Spot*"

min_percentage: 5

# The alert is use when a match is found
alert:
- "slack"

# Slack webhook
slack_webhook_url: "https://hooks.slack.com/services/new/incoming-webhook"

# Slack custom message
alert_text_type: alert_text_only
alert_text: "Percentage of Spot instances (EC2) is {0:.2f}%, below the
   minimum of {1}%"
alert_text_args:
  - percentage
  - min_percentage
```

Exemplo de código B.14: *Spot Coverage Rule*

```yaml
# Rule name and description
name: Underutilized Instances
description: "Low Utilization Amazon EC2 Instances"

# Index to search, wildcard supported
index: usage-account-*

# Rule type
type: frequency

# Alert when this many documents matching the query occur within a timeframe
num_events: 24

# num_events must occur within this amount of time to trigger an alert
timeframe:
  hours: 24

query_key: Instanceid

filter:
- query:
    query_string:
      query: "Instanceid: *"
- term:
    Metricname: "CPUUtilization"
- range:
    Average:
      from: 0
      to: 10

# The alert is use when a match is found
alert:
- "slack"

# Slack webhook
slack_webhook_url: "https://hooks.slack.com/services/new/incoming-webhook"

# Slack custom message
alert_text_type: alert_text_only
alert_text: "Intance {0} had less than 10% of CPU utilization in 24 hours ({1}). The CPU utilization average was {2:.2f}%."
alert_text_args:
  - Instanceid
  - "@timestamp"
  - Average
```

Exemplo de código B.15: *Underutilized Instances Rule*

B.6 Dockerfiles e Docker Compose

```
FROM ubuntu:16.04
FROM python:3.7

RUN apt-get update -y && apt-get install -y python-pip python-dev

COPY ./requirements.txt /cloud-cost-analysis/requirements.txt

WORKDIR /cloud-cost-analysis

RUN pip install --upgrade pip
RUN pip install -r requirements.txt

COPY ./src /cloud-cost-analysis/src

CMD ["python", "-u", "src/run.py" ]
```

Exemplo de código B.16: *Cloud Cost Analysis Dockerfile*

```
ARG ELK_VERSION

FROM docker.elastic.co/logstash/logstash:${ELK_VERSION}

WORKDIR /usr/share/logstash/

COPY logstash-input-csv/logstash-input-csv-0.1.2.gem /tmp/

RUN ./bin/logstash-plugin install /tmp/logstash-input-csv-0.1.2.gem
```

Exemplo de código B.17: *Logstash Dockerfile*

```
ARG ELK_VERSION

FROM docker.elastic.co/elasticsearch/elasticsearch:${ELK_VERSION}
```

Exemplo de código B.18: *ElasticSearch Dockerfile*

```
ARG ELK_VERSION

###############################################################################
# Build stage 0
# Extract Kibana and make various file manipulations.
###############################################################################
FROM centos:7 AS prep_files
ARG ELK_VERSION
ENV VERSION=$ELK_VERSION

RUN yum update -y && yum install -y sudo gcc-c++ make git patch wget

RUN curl --silent --location https://dl.yarnpkg.com/rpm/yarn.repo | sudo
    tee /etc/yum.repos.d/yarn.repo
RUN curl -sL https://rpm.nodesource.com/setup_10.x | sudo -E bash -
RUN yum install -y nodejs yarn

ENV NVM_DIR /usr/local/nvm
ENV NODE_VERSION 10.15.2
ENV PATH $NVM_DIR/versions/node/v$NODE_VERSION/bin:$PATH

RUN curl --silent -o- https://raw.githubusercontent.com/creationix/nvm/v0
    .33.6/install.sh | bash
RUN source $NVM_DIR/nvm.sh && nvm install $NODE_VERSION && nvm alias
    default $NODE_VERSION && nvm use default

## Provide a non-root user to run the process.
RUN adduser -u 1002 kibana && usermod -aG wheel kibana
USER kibana

WORKDIR /home/kibana

# Get Kibana and apply patch to customize Kibana
RUN git clone https://github.com/elastic/kibana.git && cd kibana && git
    checkout v${VERSION}
COPY apply.patch .
RUN patch -p0 < apply.patch

RUN cd kibana && yarn kbn bootstrap && yarn build --no-oss --skip-os-
    packages
RUN tar xzf /home/kibana/kibana/target/kibana-${VERSION}-linux-x86_64.tar.
    gz

###############################################################################
## Build stage 1
## Copy prepared files from the previous stage and complete the image.
###############################################################################
FROM centos:7
EXPOSE 5601
ARG ELK_VERSION
```

```
45 ENV VERSION=$ELK_VERSION
46
47 RUN yum update -y && yum install -y fontconfig freetype && yum clean all
48
49 # Bring in Kibana from the initial stage.
50 COPY --from=prep_files --chown=1000:0 /home/kibana/kibana-${VERSION}-linux-
       x86_64 /usr/share/kibana
51 WORKDIR /usr/share/kibana
52 RUN ln -s /usr/share/kibana /opt/kibana
53
54 ENV ELASTIC_CONTAINER true
55 ENV PATH=/usr/share/kibana/bin:$PATH
56
57 # Set some Kibana configuration defaults.
58 COPY --chown=1000:0 config/kibana.yml /usr/share/kibana/config/kibana.yml
59
60 # Add the launcher/wrapper script. It knows how to interpret environment
61 # variables and translate them to Kibana CLI options.
62 COPY --chown=1000:0 bin/kibana-docker /usr/local/bin/
63
64 # Ensure gid 0 write permissions for OpenShift.
65 RUN chmod g+ws /usr/share/kibana && find /usr/share/kibana -gid 0 -and -not
       -perm /g+w -exec chmod g+w {} \;
66
67 # Provide a non-root user to run the process.
68 RUN groupadd --gid 1000 kibana && useradd --uid 1000 --gid 1000 --home-dir
       /usr/share/kibana --no-create-home kibana
69 USER kibana
70
71 LABEL org.label-schema.schema-version="1.0" org.label-schema.vendor="
       Elastic" org.label-schema.name="kibana" org.label-schema.version=${
       VERSION} org.label-schema.url="https://www.elastic.co/products/kibana"
       org.label-schema.vcs-url="https://github.com/elastic/kibana" license="
       Elastic License"
72
73 # Add your kibana plugins setup here
74 ADD cca_management-1.0.zip /plugins/
75
76 # Install plugins
77 RUN bin/kibana-plugin install file:///plugins/cca_management-1.0.zip
78 RUN bin/kibana-plugin install https://github.com/bitsensor/elastalert-
       kibana-plugin/releases/download/1.1.0/elastalert-kibana-plugin-1.1.0-${
       VERSION}.zip
79
80 CMD ["/usr/local/bin/kibana-docker"]
```

Exemplo de código B.19: *Kibana Dockerfile*

```yaml
version: '2.2'

services:
 elasticsearch:
  build:
   context: elasticsearch/
   args:
    ELK_VERSION: $ELK_VERSION
  container_name: elasticsearch
  volumes:
   - ./config/elasticsearch.yml:/usr/share/elasticsearch/config/
    elasticsearch.yml
   - elasticsearch:/usr/share/elasticsearch/data
  ports:
   - "9200:9200"
   - "9300:9300"
  environment:
   ES_JAVA_OPTS: "-Xmx512m -Xms512m"
  networks:
   - elk

 logstash:
  build:
   context: logstash/
   args:
    ELK_VERSION: $ELK_VERSION
  container_name: logstash
  volumes:
   - ./logstash/config:/usr/share/logstash/config:ro
   - ./logstash/pipeline:/usr/share/logstash/pipeline:ro
   - reports:/usr/share/logstash/data
  ports:
   - "5140:5140"
  environment:
   LS_JAVA_OPTS: "-Xmx512m -Xms512m"
   ES_USER: $ES_USER
   ES_PASS: $ES_PASS
  networks:
   - elk
  depends_on:
   - elasticsearch

 kibana:
  build:
   context: kibana/
   args:
    ELK_VERSION: $ELK_VERSION
  container_name: kibana
  volumes:
   - ./kibana/config/kibana.yml:/usr/share/kibana/config/kibana.yml:ro
```

```
    ports:
      - "5601:5601"
    networks:
      - elk
    depends_on:
      - elasticsearch

  cloud-cost-analysis:
    build: ./cloud-cost-analysis
    container_name: cloud-cost-analysis
    ports:
      - "5000:5000"
    volumes:
      - ./cloud-cost-analysis:/cloud-cost-analysis
      - ./cloud-cost-analysis/data:/data
      - ./cloud-cost-analysis/config:/config
      - reports:/reports
    networks:
      - elk

  elastalert:
    image: bitsensor/elastalert:3.0.0-beta.1
    container_name: elastalert
    ports:
      - "3030:3030"
      - "3333:3333"
    volumes:
      - ./elastalert/config/elastalert.yaml:/opt/elastalert/config.yaml
      - ./elastalert/config/config.json:/opt/elastalert-server/config/config.
        json
      - ./elastalert/rules:/opt/elastalert/rules
      - ./elastalert/rule_templates:/opt/elastalert/rule_templates
    depends_on:
      - elasticsearch
    networks:
      - elk

networks:
  elk:
    driver: bridge

volumes:
  reports:
  elasticsearch:
```

Exemplo de código B.20: *Docker Compose*