UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

PORTABILIDADE DE SISTEMAS DE INFORMAÇÃO ENTRE NUVENS

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Engenharia de Software

Sasha André Fonseca Mojoodi

Dissertação orientada por:
Prof. Doutor Mário Luís de Jesus Rodrigues Guimarães
e por Carla Raquel de Faria-Lopes Zibreira

2016

Agradecimentos

Gostaria de agradecer a todos aqueles que me apoiaram durante o meu percurso académico. Em primeiro lugar aos meus pais, Luísa e Farshid, por me terem criado e possibilitado chegar até aqui. Ao meu gato Sonic, por ser o melhor gato do mundo. Aos meus avós, Odete e Basílio, jamais esquecido. Aos meus tios, Zé e Fátima. Aos meus primos Pedro, Filipa e João.

Ao ”Bezanas”, pelos bons momentos ao longo destes anos. Nomeadamente ao Pedro ”Choco” Verdelho, Flávio ”Gancho” Moreira, Tiago ”Buschámi” Chá, André ”Andy” Gonçalves, Gonçalo ”Caco” Homem, Ricardo ”Batman” Neto, Pedro ”Cavalo” Almeida, Sandro ”Pequenino” Nunes, Daniela Freire, Ricardo Deus e Inês Mendes.

”Queria agradecer” à Famiglia, pois sem ela este percurso não teria tido metade do significado. Em especial gostaria de destacar o meu Padrinho Miguel ”Faneca” Maneca e os meus irmãos João ”Lolada” Fernandes e Pedro ”Catarina” Ramos. Um bem haja a todos os que fazem parte.

Quero relembrar quem fez parte dos 910, em especial à Marta Carvalho, Mafalda Gomes e Rita Morais, pelo tempo que já lá vai, assim como o Ricardo ”Fadista” Silva, o Sérgio ”Sargeta” Andrade e o Tiago Duarte.

Aos 1112, pelos dois anos, impossíveis de descrever, que me proporcionaram.

Um obrigado à S21sec e a toda a equipa, por me terem acolhido para a execução deste trabalho.

Por fim, gostaria de agradecer ao meu orientador, o Professor Mário Guimarães, por me ter iniciado no caminho ”funcional” e das ”nuvens”, ensinar-me a ser mais pragmático e sucinto, a sempre boa disposição e conversas interessantes que tem para oferecer, e toda a ajuda e conselhos que disponibilizou ao longo destes meses, e que me acompanharão para sempre. Um muitíssimo obrigado a ele.

Para a FCUL não vai nada, nada, nada... TUDO!

Resumo

A ausência de *Application Programming Interfaces* (*APIs*) comuns aos mesmos serviços de diferentes fornecedores na Nuvem (*Cloud*) [78], limita a portabilidade das aplicações nesta plataforma, impedindo o aproveitamento de condições de exploração mais favoráveis que outro fornecedor venha a oferecer. Este problema é conhecido pelo nome de *vendor lock-in* [81, 62, 94]. Propostas como o JClouds [7] ou o mOSAIC [81] ajudam a criar aplicações agnósticas ao fornecedor, contudo não oferecem uma API única que abstraia as especificidades deste, pois requerem a seleção e configuração de componentes de código consoante o fornecedor escolhido, e introduzem código-morto (i.e., que nunca é executado) na aplicação dado o suporte a vários fornecedores por um mesmo componente.

Neste trabalho é proposto um *SDK* (*Software Development Kit*) que permite abstrair três serviços comuns em duas nuvens populares: a Amazon Web Services [6] e a Google Cloud Platform [47]. Este SDK, designado por Nomad[1], apresenta duas vantagens. Em primeiro lugar, o programador usa uma única API independentemente da nuvem escolhida, não sendo portanto preciso alterar a aplicação para suportar um novo fornecedor (entre os suportados), evitando-se igualmente o esforço e os erros inerentes a tal processo. Em segundo lugar, apenas o código-fonte estritamente necessário é incluído na aplicação, reduzindo o tamanho e aumentando a clareza desta, facilitando a sua testabilidade e facilidade de manutenção.

O Nomad SDK foi criado no contexto do desenvolvimento duma aplicação empresarial para a S21sec, tendo sido determinante no estabelecimento da total portabilidade desta aplicação entre as nuvens indicadas. Este SDK abstrai os serviços de armazenamento de ficheiros, bases de dados SQL e máquinas virtuais, oferecendo meios, para o desenvolvimento e instalação de sistemas de informação na Nuvem.

Palavras-chave: nuvem, vendor lock-in, portabilidade, abstração, software agnóstico à nuvem

[1] Disponível em `https://www.github.com/sashaafm/nomad`

Abstract

The absence of common Application Programming Interfaces (APIs), between the same services from different cloud providers, limits application portability in this platform. This hinders the exploitation of more favorable conditions which another provider may bring. This problem is known as *vendor lock-in*. Efforts like JClouds or mOSAIC help build vendor agnostic software. However, these solutions do not offer a unique API that abstracts the specificities from vendors, requiring the selection of vendor specific code components, and leave dead code (i.e., code that is never executed) in the application, given that several vendors are supported by the same component.

In our work, we propose an SDK (Software Development Kit) that abstracts three common services in two popular clouds: Amazon Web Services and Google Cloud Platform. This SDK, named Nomad SDK[2], presents two advantages. First, the programmer uses a unique API, independently of the chosen cloud provider. This way, the application does not require code changes, in order to support a new cloud vendor (between those which are supported), reducing the effort imposed by this process. Second, only the source code that is strictly necessary is included in the application, inherently reducing its size and increasing clarity, which facilitates testability and maintainability.

The Nomad SDK was created in the context of an enterprise applicatoin built for the S21sec company, and it proved useful to stablish total portability for this application between the mentioned vendors. This SDK abstracts the storage, SQL database and virtual machines services. This was achieved while offering tools to develop and install information systems in the Cloud.

Keywords: cloud, vendor lock-in, portability, abstraction, cloud agnostic software

[2]Available at `https://www.github.com/sashaafm/nomad`

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O trabalho aqui presente foi concretizado no âmbito da disciplina de Projeto de Engenharia Informática na área de especialização de Engenharia de Software do Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa. Este trabalho foi realizado em duas partes, desenvolvidas em paralelo: a primeira, que tomou lugar na organização S21sec [89], sob a forma de estágio profissional, onde foi desenvolvida uma aplicação denominada S21sec Portal — uma *intranet* empresarial para centralização da comunicação interna da empresa e automatização de alguns dos seus serviços; a segunda, correspondeu ao desenvolvimento do Nomad SDK — um *kit* de desenvolvimento de sistemas de informação portáteis na Nuvem — criado fora do âmbito do estágio profissional, mas utilizado no suporte ao desenvolvimento da aplicação para a S21sec. Este capítulo apresenta a motivação, os objetivos, a metodologia e as contribuições deste trabalho, e termina apresentando a estrutura e organização do resto do documento.

1.1 Motivação

São cada vez mais os programadores e organizações que constroem *software* utilizando o paradigma da computação na Nuvem (*Cloud*). Estes produtos de *software* recorrem aos recursos da Nuvem (infraestrutura e serviços) e consomem-nos através de *Application Programming Interfaces* (*APIs*) durante as fases de desenvolvimento, teste e produção. Todavia, devido a existirem distintos fornecedores de nuvens, a dificuldade da escolha do mais indicado pode ser um obstáculo na adoção da mesma [48]. Qualquer que seja a decisão, ela resulta no problema conhecido como *vendor lock-in* [81, 62, 94]. Este problema traduz-se no comprometimento com um determinado fornecedor, não sendo possível utilizar outro fornecedor sem incorrer em alterações manuais ao código-fonte, revisões à arquitetura do *software* ou às funcionalidades do mesmo. O problema descrito tem origem nas diferenças entre APIs, serviços e infraestrutura entre diferentes fornecedores [58, 110, 15].

Alcançar uma solução para o problema do *vendor lock-in* possibilitará a construção de

software agnóstico ao fornecedor de nuvem, removendo o compromisso com um único fornecedor e aproveitando os benefícios da exploração de qualquer serviço na nuvem. Contudo, as soluções atuais para este problema possuem limitações, nomeadamente: não oferecem APIs únicas que sejam aplicáveis da igual forma a todos os fornecedores, e incluem código-fonte não relevante para integrar a aplicação com o fornecedor de nuvem escolhido. Uma API única, que não necessitasse de parametrizações específicas para cada fornecedor, permitiria aliviar os programadores de decisões e configurações nas aplicações — que ditam a arquitetura, detalhes da implementação, ou funcionalidades possíveis — e aumentar o grau de confiança na portabilidade das aplicações. A remoção do código-fonte, não específico à nuvem escolhida, reduziria a quantidade de código-morto (i.e., código nunca executado), aumentando assim a simplicidade, testabilidade e facilidade de manutenção das aplicações. Em suma, a resolução destes dois problemas traria benefícios ao processo de desenvolvimento de *software* para a Nuvem.

O Elixir [108] — uma linguagem de programação moderna para sistemas distribuídos e concorrentes assente na plataforma Erlang [37] — foi a tecnologia eleita para o desenvolvimento de ambas as partes. Os motivos para a escolha do Elixir são o facto de ser uma linguagem bastante expressiva assente sobre uma plataforma cujas características de escalabilidade e tolerância a faltas fazem dela um excelente veículo para o desenvolvimento e exploração de aplicações na Nuvem. Em particular, o mecanismo de meta-programação do Elixir é utilizado em conjunto com uma arquitetura por camadas e adaptadores, por modo a alcançar a API única e eliminar todo o código-morto das aplicações. Adicionalmente, ao escolher o Elixir, e consequentemente a plataforma Erlang, este trabalho contribui com a única solução para o problema do *vendor lock-in* existente nesta plataforma.

1.2 Objetivos

Este trabalho teve por objetivo o desenvolvimento de uma solução de portabilidade, para sistemas de informação na nuvem, e a sua validação no desenvolvimento de um sistema de informação para a empresa S21sec. Desta forma, foram definidos os seguintes objetivos para ambas as partes deste trabalho:

Nomad SDK:

- **N1**: API única, que garanta a portabilidade das aplicações nas nuvens Amazon Web Services e Google Cloud Platform.

    1. Especificação e implementação de API para consumo de serviço de armazenamento de ficheiros (Amazon S3[1] e Google Cloud Storage[2]). Este serviço

[1] https://aws.amazon.com/s3/
[2] https://cloud.google.com/storage/

foi escolhido, pois o S21sec Portal possui funcionalidades de carregamento e descarregamento de ficheiros.

2. Especificação e implementação de API para consumo de serviço de bases de dados SQL (Amazon RDS[3] e Google Cloud SQL[4]). Este serviço foi escolhido, pois o S21sec Portal utiliza intensivamente uma base de dados SQL, para as suas operações.
3. Especificação e implementação de API para consumo de serviço de máquinas virtuais (Amazon EC2[5] e Google Compute Engine[6]). Este serviço foi escolhido, por forma a facilitar a instalação de aplicações Elixir na Nuvem, para além que a utilização de serviços de máquinas virtuais é bastante popular entre os utilizadores das nuvens.

- **N2**: Ferramentas para gestão de recursos e serviços em ambas as nuvens.

  1. Listagem, criação e eliminação de armazenamento de ficheiros. Também deverá ser possível migrar os ficheiros de um fornecedor para o outro.
  2. Listagem, criação, eliminação e reinício de instâncias de bases de dados SQL.
  3. Listagem, criação, eliminação e reinício de instâncias de máquinas virtuais.

- **N3**: Alteração de fornecedor na Nuvem não envolve alterações ao código-fonte.

- **N4**: Exclusão do código-morto (código pertencente ao fornecedor não escolhido).

S21sec Portal:

- O S21sec Portal deve ser uma *intranet*, para comunicação e automatização de processos internos:

  1. Na faceta da comunicação, a aplicação deve oferecer um sistema de *tickets*, com notificações por *email*. A integração com o Nomad SDK tem por alvo a deslocação dos componentes de base de dados SQL e de armazenamento de ficheiros do S21sec Portal, para serviços correspondentes nas nuvens suportadas.
  2. Na faceta de automatização de processos internos, a aplicação deve integrar-se com o sistema de auditorias de vulnerabilidades Nessus[7] [100].

[3] https://aws.amazon.com/rds/
[4] https://cloud.google.com/sql/
[5] https://aws.amazon.com/ec2/
[6] https://cloud.google.com/compute/
[7] Esta faceta da aplicação foi maioritariamente concebida por um aluno do Mestrado em Segurança Informática, que realizou o estágio profissional em simultâneo.

1.3 Metodologia

Uma metodologia tem o propósito de detalhar os passos a seguir, para atingir um determinado objetivo. A primeira etapa foi o levantamento de requisitos funcionais, para o S21sec Portal, através de entrevistas com os *stakeholders* (gestores dos vários departamentos a integrar na aplicação e utilizadores finais). Seguiu-se uma análise dos requisitos não-funcionais, a nível de atributos de qualidade e de tecnologias a utilizar. Feito isto, seguiu-se a exploração de possibilidades arquiteturais, para de seguida realizar um desenho detalhado do sistema.

Inicialmente, o trabalho visava unicamente o desenvolvimento da *intranet* S21sec Portal. Contudo, ao longo da fase inicial de levantamento de requisitos e estudo do estado-de-arte do desenvolvimento de aplicações Web, surgiu o desafio de construir uma solução arquitetural, para a aplicação da S21sec, que também aproveitasse a Nuvem. Foi com este desafio que se desenvolveu a ideia de tornar a aplicação portável entre nuvens. Ao explorar esta possibilidade encontrou-se a problemática do *vendor lock-in* e decidiu-se construir o Nomad SDK.

As nuvens escolhidas para suportar foram a Amazon Web Services[8] e Google Cloud Platform[9], por serem duas das nuvens mais populares na atualidade [44]. Como o S21sec Portal faz uso de uma base de dados SQL e de armazenamento de ficheiros, que devem ser manipulados em tempo de execução, foram escolhidos os serviços de bases de dados SQL (Amazon RDS e Google Cloud SQL) e de armazenamento de ficheiros (Amazon S3 e Google Cloud Storage) destas nuvens. O serviço de máquinas virtuais (Amazon EC2 e Google Compute Engine) foi escolhido, para a eventual possibilidade de instalar a aplicação automaticamente em ambas as nuvens.

A partir deste ponto, o S21sec Portal e o Nomad SDK foram desenvolvidos em paralelo. Ambos usufruíram de processos de desenvolvimento iterativo (três iterações cada um), podendo assim ter os seus requisitos reavaliados ao fim de cada iteração, uma vez que estes não eram bem definidos, e adaptar o processo de desenvolvimento consoante as necessidades [26]. No S21sec Portal, as funcionalidades foram sendo implementadas seguindo uma ordem de prioridade, sendo realizados testes no final de cada etapa. Cada uma destas iterações foi fechada com uma demonstração a *stakeholders*, por forma a validar as funcionalidades e o comportamento do sistema, sendo que as alterações a fazer foram feitas no início da etapa seguinte.

No Nomad SDK, as etapas foram separadas pelos três serviços a implementar. Ao fim da implementação de cada serviço, estes foram testados para ambas as nuvens, por meio de aplicações protótipo desenvolvidas para este fim (um protótipo por etapa). As especificações e o código-fonte do Nomad SDK foram testados por meio de *mock tests*,

[8] https://aws.amazon.com
[9] https://cloud.google.com

uma vez que uma bateria de testes que utilizasse as nuvens não seria viável, pois tal iria requerer longos períodos de espera devido à latência da conexão, distâncias físicas entre os pontos de comunicação e o tempo de processamento dos pedidos (que podem ser superiores a 10 minutos).

Finalmente, aquando da aproximação da fase final de ambos os projetos, foi concretizada a integração do Nomad SDK no S21sec Portal. Para isto, foi feita uma versão do portal utilizando este *sdk*, para as suas funcionalidades de bases de dados SQL e armazenamento de ficheiros. Os resultados obtidos desta integração foram analisados, em comparação com os objetivos delineados inicialmente, tendo-se obtido resultados positivos.

1.4 Contribuições

Com o trabalho aqui apresentado, as contribuições realizadas podem ser sumarizadas da seguinte forma:

1. **Nomad SDK**: um *kit* de desenvolvimento de aplicações Elixir portáveis entre nuvens, que até ao momento conta com suporte para os serviços de armazenamento de ficheiros, bases de dados SQL e de infraestrutura (máquinas e discos virtuais) nas nuvens Amazon Web Services e Google Cloud Platform. As aplicações construídas com este *sdk* utilizam uma API única, que as livra de alterações ao código-fonte para suportar um outro fornecedor de nuvem, e apenas inclui na aplicação o código-fonte estritamente necessário durante a sua execução.

2. **S21sec Portal**: uma *intranet* empresarial para a S21sec, que oferece um sistema de *tickets* e automatização de serviços internos de auditoria de vulnerabilidades, através da integração com a ferramenta Nessus [100] .

3. **GCloudex**: uma biblioteca de clientes Elixir para as APIs da Google Cloud Platform [41]. Esta coleção é equivalente ao projeto ErlCloud [39], mas aplicável à GCP ao invés da AWS.

4. **ExAWS**: duas contribuições para o ExAWS [24], um projeto utilizado no Nomad SDK, relativas à implementação de um cliente para o serviço Amazon RDS e de um cliente para o serviço Amazon EC2.

1.5 Estrutura do Documento

Os capítulos seguintes estão organizados da seguinte forma:

- **Capítulo 2 – O Paradigma de Computação na Nuvem:** são introduzidos e revistos os conceitos relacionados com a computação na Nuvem, e é feita uma comparação

do desenvolvimento de *software* para a Nuvem com o desenvolvimento ”tradicional”.

- **Capítulo 3 - O Problema do *Vendor Lock-in*:** a problemática do *vendor lock-in* é explorada mais aprofundadamente.
- **Capítulo 4 – A Solução Nomad SDK:** a solução desenvolvida é apresentada, sendo o seu contexto e objetivos definidos. A arquitetura da solução é descrita, tal como a sua implementação e bateria de testes. É feita uma introdução à linguagem Elixir e é feita uma análise da sua complementaridade com a Nuvem.
- **Capítulo 5 - O S21sec Portal:** a aplicação desenvolvida para a S21sec é apresentada. São expostas as fases de análise de requisitos e de desenho da aplicação.
- **Capítulo 6 - O Nomad SDK No Mundo Real:** os resultados da integração do Nomad SDK no S21sec Portal são demonstrados.
- **Capítulo 7 - Trabalho Relacionado:** comparação do Nomad SDK com o estado-da-arte das soluções para o problema do *vendor lock-in*.
- **Capítulo 8 - Conclusão:** são apresentadas as conclusões, contribuições e ideias para trabalho futuro.

Capítulo 2

O Paradigma de Computação na Nuvem

Neste capítulo são introduzidos os conceitos ligados à Computação na Nuvem (*Cloud Computing*), assim como as características, vantagens, desvantagens e desafios deste paradigma de computação. É feita uma análise das mudanças que a Nuvem traz à disciplina da Engenharia de Software, quando comparada com a versão "tradicional" da mesma. Os termos e as definições presentes neste documento seguem as mesmas do documento *The NIST Definition of Cloud Computing* [78].

2.1 O Que é a Nuvem?

A Nuvem é um modelo de computação assente na partilha de recursos computacionais (por exemplo, máquinas, unidades de armazenamento, e serviços aplicacionais) distribuídos de forma ubíqua pela Internet. Estes recursos podem ser alocados e removidos com mínimo esforço pelos utilizadores da Nuvem, sendo a interação feita via interfaces HTTP (HyperText Transfer Protocol), gráficas ou de linha de comandos.

Mais especificamente, a Nuvem remove o processamento e armazenamento de dados das nossas máquinas, e localiza-os em servidores distribuídos por centros de dados (*data centers*) distribuídos pelo globo. Estes dados e serviços são acedidos através da Internet, havendo por norma, uma gestão automática de toda a infraestrutura [78]. Uma máquina local pode ser transformada num simples terminal, que apenas apresenta e transmite os dados entre o utilizador e a Nuvem[1].

Uma analogia, feita por Sandholm *et al.* [90], consiste na comparação com a energia elétrica pública. Tal como podemos ligar um eletrodoméstico a uma tomada, e receber energia automaticamente, com a Nuvem podemos realizar um pedido e receber uma resposta calculada algures na Internet.

[1] Esta ideia é aplicada em projetos como o sistema operativo ChromeOS [46]

2.2 Características

O modelo de computação na Nuvem possui cinco características essenciais [78]:

1. **Solicitabilidade:** possibilidade de alocação de novos recursos computacionais, tais como novas máquinas virtuais ou sistemas de armazenamento, sempre que desejado e sem intervenção humana do lado do fornecedor de nuvem. Isto é feito a pedido do utilizador, por meio de uma interface programática ou gráfica. Em geral, os custos dos recursos são aplicados consoante o tempo de utilização dos mesmos.

2. **Acessibilidade:** os recursos da Nuvem podem ser acedidos por mecanismos *standard*, tais como *browsers* Web ou APIs HTTP.

3. **Disponibilidade:** os recursos estão em espera, para servirem qualquer consumidor. A localização exata dos mesmos, e como estes estão a ser oferecidos não é clara. O utilizador apenas recebe o que pede, sem ter noção do que acontece realmente do lado da Nuvem. Os recursos, por norma, são partilhados num modelo de *multi-tenancy*, ou seja, os mesmos recursos físicos são partilhados por mais que um utilizador. É preciso salientar a diferença entre serviços multi-utilizadores e *multi-tenancy*. O primeiro considera um serviço, do qual vários utilizadores usufruem, tendo conhecimento da existência de outros utilizadores nesse mesmo serviço. Em comparação, no segundo conceito é considerada a partilha de recursos na infraestrutura, mas é dada a impressão desta utilização estar a ser exclusiva para cada utilizador. Um exemplo de serviço multi-utilizador será a rede social A a ser acedida por vários utilizadores, enquanto as redes sociais A e B, alojadas na mesma nuvem, são um exemplo de *multi-tenancy* [90]. Contudo, existem nuvens que oferecem a possibilidade de utilização de recursos não-partilhados, embora com custos mais elevados.

4. **Elasticidade:** o utilizador tem a perceção dos recursos serem ilimitados, sendo que estes escalam a pedido do utilizador ou automaticamente consoante as necessidades da aplicação. A elasticidade pode ser concretizada de duas formas:

   (a) Horizontalmente: são alocados novos recursos individuais para uso da aplicação. Por exemplo, adicionar mais máquinas virtuais. Isto pode ser útil para casos em que as máquinas atuais já estão com a sua carga de utilização no máximo e a única forma de aumentar o desempenho total da aplicação é através da adição de máquinas.

   (b) Verticalmente: os recursos já em utilização recebem melhores componentes (p. ex. melhor processador ou mais memória). Esta forma de escalabilidade pode ser utilizada na situação onde o desempenho melhora conforme o *hardware* da máquina (por exemplo, computações paralelas *multi-core*).

Na **Figura 2.1**, é exibida a forma como a Elasticidade da Nuvem permite que as aplicações se adaptem em tempo real, à carga de trabalho. Ao ser possível escalar a aplicação, é possível reduzir os custos na infraestrutura utilizada, pois esta é alocada e removida dinamicamente, conforme as necessidades do sistema.

5. **Métricas do serviço:** as necessidades das aplicações e os recursos por estas utilizados são monitorizados automaticamente. Obtêm-se assim indicadores, para a tomada de decisão de quando e como os recursos utilizados devem escalar, e sobre os níveis de serviço que estão a ser obtidos.

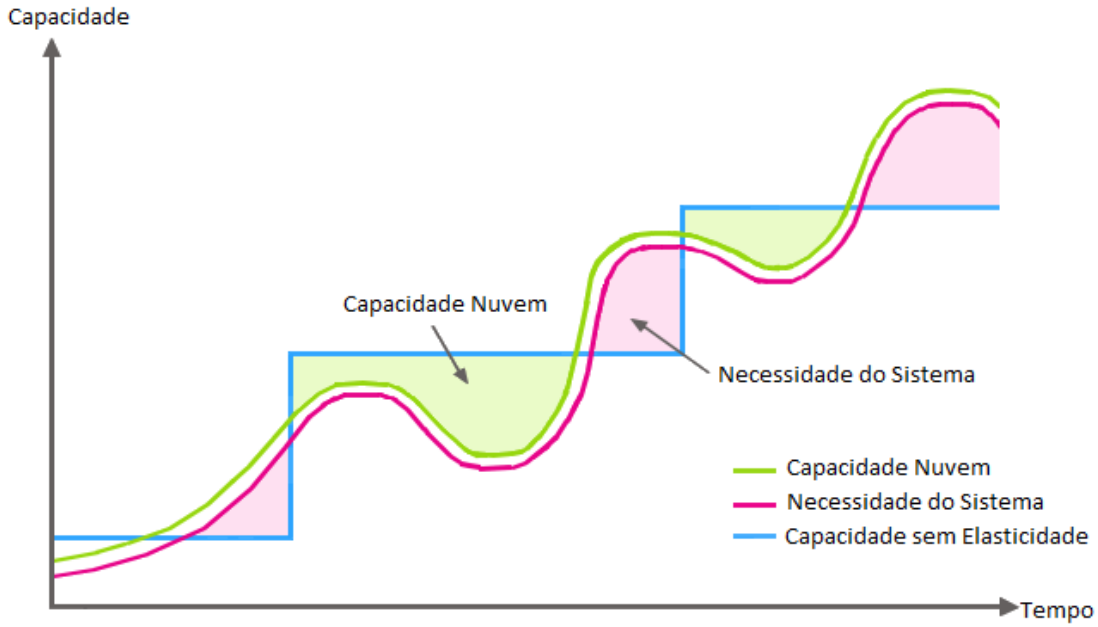


Figura 2.1: Na Nuvem, a aplicação pode escalar consoante a carga de trabalho num dado momento. Em contraste, numa aplicação fora da Nuvem, a infraestrutura não pode ser alterada de forma tão dinâmica, fazendo com o que a carga de trabalho seja superior à capacidade da aplicação em determinados períodos de tempo. Adaptado de [91].

2.3 Modelos de Serviço

A forma como se interage com a Nuvem é considerado um *modelo de serviço*, que pode ser diferenciado consoante o nível de controlo oferecido ao utilizador. Estas interações variam consoante a tarefa que se deve desempenhar. Por exemplo, consideremos uma aplicação na Nuvem, para consulta de números telefónicos. A tarefa de instalação desta aplicação requer uma interação distinta de um pedido de consulta de um número telefónico, quando a aplicação já estiver a executar na Nuvem. Estas duas tarefas são concretizadas por dois modelos de serviços distintos, pois uma delas requer a manipulação

de recursos na Nuvem, enquanto a outra requer o processamento de um pedido a uma aplicação instalada nesses recursos.

Estes modelos de serviços podem ser vistos como um conjunto de camadas, onde cada camada abaixo oferece maior liberdade e controlo sobre os recursos (o diagrama do modelo é apresentado na **Figura 2.2**).

1. **Camada superior - *Software-as-a-Service (SaaS)*:** o utilizador apenas consegue interagir com a aplicação alojada na infraestrutura e, geralmente, através de uma interface *thin client* (p. ex. um *browser* Web). A gestão de recursos, tais como o processador ou sistema operativo da máquinas onde corre a aplicação, por parte do utilizador, não é possível. Alguns exemplos de organizações que oferecem produtos no modelo SaaS são a Salesforce[2] e a Dropbox[3].

2. **Camada intermédia - *Platform-as-a-Service (PaaS)*:** podem ser alojadas aplicações e serviços na infraestrutura (desde que as tecnologias necessárias sejam suportadas). Não existe controlo pelo utilizador sobre os recursos (p. ex., sistema operativo), mas sim sobre a aplicação e o ambiente da mesma. Alguns exemplos de fornecedores deste modelo são a Engine Yard[4] e a Google App Engine[5].

3. **Camada inferior - *Infrastructure-as-a-Service (IaaS)*:** os recursos podem ser totalmente geridos (p. ex., as especificações da máquina virtual ou o sistema operativo instalado na mesma). As aplicações podem ser alojadas sem restrições, evitando-se o condicionamento a um conjunto de tecnologias suportadas. Contudo, não é oferecido controlo sobre a infraestrutura ao nível físico. Alguns exemplos de fornecedores deste modelo são a Amazon Web Services [6] e a Google Cloud Platform [47].

É de realçar que apesar do Nomad SDK evitar as alterações de código-fonte nas aplicações, isto só é admissível aquando da portação destas entre fornecedores de IaaS. Isto acontece, pois as aplicações desenvolvidas sobre PaaS possuem fortes dependências com o ambiente de execução, para o qual foram desenvolvidas. Tal não acontece nas aplicações em IaaS, uma vez que neste caso o próprio ambiente de execução também será portado para um novo IaaS.

2.4 Tipos de Nuvem

As nuvens podem ser classificadas num de quatro tipos consoante a acessibilidade permitida, quem é responsável pela sua gestão, e onde estão alojadas, tal como sumarizado na

[2]https://www.salesforce.com/eu/?ir=1
[3]http://www.dropbox.com/
[4]https://www.engineyard.com/
[5]https://cloud.google.com/appengine/

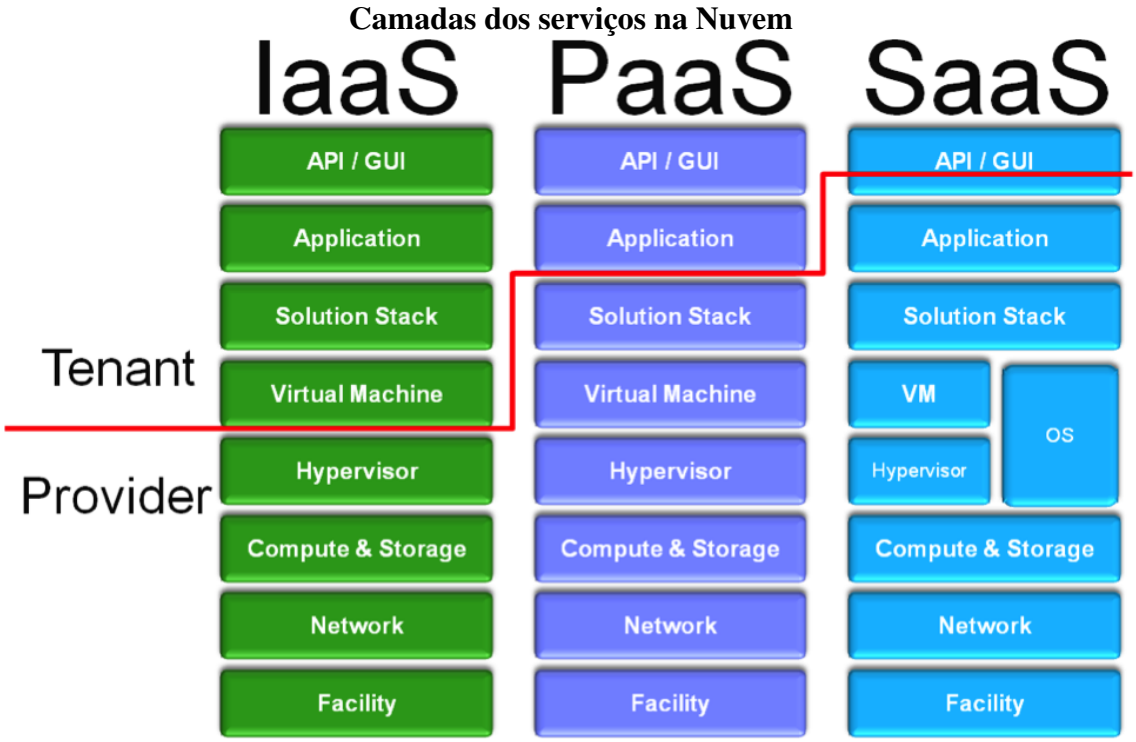


Figura 2.2: Cada camada superior oferece mais funcionalidades ao consumidor (*tenant*) e menos responsabilidade ao fornecedor (*provider*). Retirado de [19].

Tabela 2.1.

1. **Nuvens Públicas:** organizações como a Google[6], Amazon[7] e Microsoft[8] oferecem as soluções mais populares e com o maior número de serviços [44]. Estas nuvens são consideradas *públicas*, pois qualquer utilizador pode servir-se das funcionalidades disponibilizadas, por um determinado montante.

2. **Nuvens Privadas:** soluções como o Eucalyptus[9] da HP, permitem aos utilizadores construirem a sua própria nuvem, ficando esta alojada nas suas instalações. Neste caso quem aloja a nuvem, também será o fornecedor da mesma, pois também será responsável pela sua gestão. Contudo, é de notar a diferença na acessibilidade, pois nas nuvens públicas é permitido o acesso a qualquer utilizador, em troca de um determinado valor, enquanto nas nuvens privadas o acesso é restrito à organização que a instalou.

3. **Nuvens Comunitárias:** são consideradas *nuvens comunitárias*, aquelas que servem utilizadores de uma comunidade (p. ex., universidades, centros de investigação

[6]https://cloud.google.com/
[7]https://aws.amazon.com/
[8]https://azure.microsoft.com/en-gb/
[9]http://www8.hp.com/us/en/cloud/helion-eucalyptus-overview.html

ou empresas do mesmo sector). Novamente, a acessibilidade deste tipo de nuvem é um dos fatores distintivos das nuvens comunitárias, pois o acesso é restrito a utilizadores da comunidade que instalou a nuvem.

4. **Nuvens Híbridas:** as nuvens compostas por nuvens de vários tipos são consideradas híbridas. Por exemplo, uma organização pode utilizar a infraestrutura duma nuvem privada para colocar uma aplicação em produção, e esta fazer uso de bases de dados alojadas numa nuvem pública.

Tabela 2.1: Comparação dos vários tipos de nuvens.

Tipo	Acessibilidade	Proprietário e Gestor	Localização
Privada	Exclusivo à organização	Organização	Dentro da organização
Pública	Público em geral	Fornecedor	Na infraestrutura do fornecedor
Comunitária	Exclusivo à comunidade	Comunidade ou terceiros	Dentro ou fora da comunidade
Híbrida	Pública ou privada	A organização é o proprietário; A organização ou o fornecedor podem ser os gestores	Dentro da organização e nos fornecedores

2.5 Desenvolvimento para a Nuvem vs. desenvolvimento tradicional

O desenvolvimento de *software* para a Nuvem requer atenção a determinados aspetos, para o correto aproveitamento da mesma [97, 81]. A atenção que estes aspetos requerem deriva, maioritariamente, da natureza distribuída da Nuvem, da sua variedade de recursos e serviços, limitações e imposições, que originam diferenças, quando comparado com o desenvolvimento tradicional (i.e., sem ser para a Nuvem):

1. **Escolha de serviços e fornecedores**: as diferenças entre os fornecedores e seus serviços, podem originar desafios na sua escolha. Os programadores devem estudar as várias hipóteses, por modo a entender quais são viáveis de utilizar num dado contexto. Por exemplo, a AWS oferece vários serviços de armazenamento, tais como o Amazon S3[10], Amazon Glacier[11], ou Amazon CloudFront[12]. Estes serviços de armazenamento possuem diferenças entre si, estando cada um direcionado para um certo contexto. Estes devem ser avaliados, por modo a ser escolhido o serviço mais adequado às necessidades de uma aplicação.

2. **Inspeção do estado do sistema:** o estado do sistema permanece distribuído por vários sub-sistemas e serviços, alguns dos quais os programadores têm pouco controlo ou possibilidades de inspeção. Embora este problema seja partilhado com o desenvolvimento tradicional, é de notar que no tradicional o problema é de menor gravidade, pois existe um controlo total do ambiente de instalação.

[10]https://aws.amazon.com/s3/
[11]https://aws.amazon.com/pt/glacier/
[12]https://aws.amazon.com/cloudfront/

3. **Adaptação das aplicações:** os recursos e serviços oferecidos pelas nuvens evoluem. Apesar de isto também acontecer no desenvolvimento tradicional, a diferença reside na instalação. As APIs do desenvolvimento tradicional estão instaladas no ambiente controlado pelo programador, sendo que este apenas utiliza novas versões das APIs caso assim o deseje. Contudo, na Nuvem as APIs são alteradas conforme a vontade do fornecedor. O facto da Nuvem ser um mercado competitivo também leva os fornecedores a tentarem introduzir novas funcionalidades o mais rapidamente possível, por forma a ganharem vantagem aos seus concorrentes. Por consequência, estas novas funcionalidades, introduzem alterações às APIs. Desta forma, existe um maior ênfase na manutenção das aplicações para a Nuvem.

4. **Gestão de recursos:** as nuvens, geridas pelos fornecedores, reduzem o peso das tarefas de gestão e administração dos recursos. Os programadores, ao ficarem aliviados desta tarefa, conseguem maior foco nos objetivos funcionais de um *software* [113].

5. **Aplicações escaláveis:** as aplicações devem ser construídas tendo em atenção as restrições impostas pelos fornecedores. Por exemplo, alguns fornecedores impõe limites à quantidade de memória que um dado utilizador pode requisitar em simultâneo, sendo necessário entrar em contacto com o departamento de apoio aos clientes, por modo a retirar a limitação. Por outro lado, a escalabilidade das aplicações também fica limitada pelos recursos que estão disponíveis (p. ex., as especificações das máquinas virtuais possuem limites, que o programador não pode controlar).

6. **Diferentes ambientes de desenvolvimento e produção:** a transição de um ambiente local, para um ambiente na Nuvem, traz diferenças à tarefa de desenvolvimento. O ambiente da nuvem alvo pode não ser trivial de replicar durante o desenvolvimento, ao invés que no desenvolvimento tradicional, este ambiente pode ser acedido ou reproduzido mais facilmente. Uma aplicação pode ser desenvolvida localmente, contudo os seus ambientes de teste e produção terão de ser na Nuvem. Em contraste, uma aplicação tradicional terá todos os seus ambientes locais.

   Por exemplo, uma aplicação que utilize um sistema de ficheiros na Nuvem, dificilmente poderá ter o seu ambiente replicado localmente. Em tempo de desenvolvimento a API para uso do sistema de ficheiros pode ser implementada, contudo, a sua utilização e testes só poderão ocorrer somente na Nuvem. Apesar de ser possível realizar testes *mock*, aspetos como os atributos de qualidade, ao nível do desempenho ou eficiência, só poderão ser avaliados aquando da utilização do ambiente na Nuvem.

2.5.1 Vantagens

Seguidamente são enumeradas as vantagens do desenvolvimento para a Nuvem, e discute-se como estas afetam o desenvolvimento das aplicações.

1. **Recursos a pedido:** uma organização pode alocar recursos computacionais consoante as necessidades do seu processo de desenvolvimento de *software*. Por exemplo, a organização poderá alocar um conjunto de máquinas virtuais na Nuvem para testar ocasionalmente um novo algoritmo paralelo para análise de grandes volumes de dados, evitando os custos que teria de ter para possuir localmente uma infraestrutura adequada para esse efeito.

2. **Infraestrutura:** a Nuvem pode alojar qualquer aplicação. Isto pode ser útil para montar um negócio via SaaS, ou demonstrações remotas a clientes, por exemplo.

3. **Recursos partilhados:** a partilha de recursos torna os custos para cada utilizador mais baixos, em comparação com os custos de adquirirem os mesmos recursos para uso individual. Isto origina do mesmo recurso alojar vários utilizador, tornando-o mais viável economicamente.

4. **Depreciação dos recursos:** o facto de ser possível utilizar os recursos sem os possuir, faz com que sejam evitadas depreciações de valor dos mesmos [86].

5. **Elasticidade:** a elasticidade proporcionada ao nível dos recursos, tornam o escalamento de aplicações automático e eficaz. Um sistema pode receber recursos consoante a necessidade, tal como a hora do dia em que se encontra (por exemplo, durante o período noturno é expectável que estejam menos utilizadores a interagir com o sistema), região ou eventos pontuais de grandes picos de trabalho.

2.5.2 Desvantagens

Existem também desvantagens e obstáculos no desenvolvimento e na exploração de aplicações na Nuvem [81, 34].

1. **Processo de desenvolvimento:** a falta de código-fonte dos serviços disponibilizados pela Nuvem, pode dificultar a compreensão sobre o funcionamento da aplicação. Isto pode, por exemplo, dificultar o raciocínio sobre o desempenho final da aplicação como um todo. Certos detalhes sobre os serviços oferecidos, também permanecem opacos ao programador.

2. **Imposições do fornecedor:** o programador é obrigado a respeitar as imposições do fornecedor ao nível do desenho e implementação dos seus serviços, como por exemplo, capacidade máxima de memória disponível a uma máquina virtual, ou os motores de bases de dados SQL suportados.

3. **Correspondências entre recursos:** os serviços de nuvens distintas nem sempre possuem correspondências entre os recursos que disponibilizam. No **Apêndice A** é apresentada uma tabela exemplificativa desta situação ao nível de máquinas virtuais disponíveis na AWS e GCP.

4. **Conectividade obrigatória à Internet:** é obrigatório uma conexão à Internet, para consumo da Nuvem. Quebras ou velocidades reduzidas na conexão, dificultam a utilização dos recursos da Nuvem, e podem trazer prejuízo aos programadores e organizações [4, 56].

5. **Gestão pelo fornecedor:** as nuvens são geridas por terceiros, o que poderá ser uma desvantagem, pois, caso surjam problemas com a infraestrutura ou plataforma, os consumidores da Nuvem ficam dependentes do fornecedor e da sua eficácia, para a resolução desses mesmos problemas (por exemplo, correção de erros nos serviços oferecidos).

2.5.3 Desafios

A Nuvem, ao ser um novo paradigma para o desenvolvimento de *software*, traz consigo novos desafios para a Engenharia de Software. Sommerville [96] destaca três desses desafios:

1. Necessidade de desenvolvimento de modelos de programação com foco nas características de paralelismo e elasticidade. Por exemplo, um dos modelos mais populares é o MapReduce [30, 36], que permite processar grandes volumes de dados paralelizando as computações. Este modelo utiliza arquiteturas *multi-core* e de *clusters*, e abstrai os detalhes de paralelização, tolerância a faltas e distribuição dos dados [36].

2. Há que garantir a correção do sistema, independentemente dos recursos que lhe são alocados em determinado período da sua execução. Os sistemas devem utilizar algoritmos e código-fonte flexível aos recursos que possui num dado momento, e que garantam que o programa executa de igual forma com qualquer conjunto de recursos.

3. Aplicação do Budget-Driven Programming, onde os sistemas, ao estarem conscientes dos preços praticados pelos vários fornecedores, seriam capazes de escalar e requisitar recursos em qualquer nuvem, de forma a otimizar os custos e desempenho do sistema.

Zhang *et al.* [114] referem também o desafio de construir *frameworks*, que auxiliem o desenvolvimento de aplicações de larga escala de computação intensiva de dados. Estas *frameworks* teriam o objetivo de colmatar as diferenças das características recursos

disponíveis nas nuvens (p. ex. *hardware*, onde os preços e componentes disponíveis variam consoante o fornecedor). Estas características variantes dificultam o aproveitamento equivalente das várias nuvens, ou seja, uma aplicação pode obter diferentes níveis de desempenho, consoante o fornecedor. Como exemplo, os autores referem que uma aplicação que utilize o MapReduce, pode ter os seus custos e desempenho otimizados, caso sejam aplicados escalonadores adequados a cada operação oferecida por este modelo.

Estes autores também referem a dificuldade da migração das máquinas virtuais inter-nuvens e intra-nuvens. Aquando do surgimento de picos de trabalho, a possibilidade de migrar a máquina virtual para uma nuvem ou região mais propícia seria uma mais valia. Trabalhos como o de Kratzke [64] já começam a tentar solucionar estas questões, por meio de virtualização (neste caso através da ferramenta Docker[13]). O autor explorou o agnosticismo, ao nível do *hardware* e ambiente, por modo a portar aplicações entre IaaS de vários fornecedores. Este problema, contudo, é aliviado por serviços de captura do estado de uma máquina virtual, que permitem posteriormente lançar outras máquinas virtuais, com esse mesmo estado. De notar que esta questão de portabilidade, estudada por Kratzke, é referente unicamente à portabilidade do ambiente de uma dada máquina virtual para outra máquina virtual (i.e., possuir um dado ambiente na máquina virtual A e lançar uma máquina virtual B com o mesmo ambiente), não sendo referente à portabilidade de aplicações.

No trabalho de Dillon *et al.* [31], é referido o desafio de determinar quais as partes de uma aplicação, que devem ser migradas para a Nuvem, pois nem sempre faz sentido migrar toda a aplicação. Contudo, as organizações não confiam o suficiente na Nuvem, para migrar as suas aplicações para a mesma. Esta desconfiança é, maioritariamente, derivada de críticas a cerca da segurança deste modelo de computação.

Desta forma, a segurança, é também um desafio que a Nuvem proporciona. O modelo de *multi-tenancy* assusta as organizações, pois os seus dados sensíveis e confidenciais permanecem alojados em infraestruturas partilhadas, o que origina dúvidas em relação à proteção e privacidade dos dados [80]. A localização física dos *data centers* da Nuvem é, igualmente, um obstáculo para a sua adoção, uma vez que a infraestrutura poderá estar localizada noutro país. Isto faz com que os dados armazenados estejam sujeitos às leis locais, referentes à informação digital. As nuvens também realizam duplicação dos dados, o que reforça as críticas à privacidade da informação dos consumidores.

2.6 Sumário

Neste capítulo estabeleceu-se o significado de computação na Nuvem e a definição da mesma. Foram referidas as vantagens e desvantagens que a utilização da Nuvem proporciona e quais os desafios que surgem para a Engenharia de Software, aquando da

[13] https://www.docker.com

utilização deste paradigma.

Capítulo 3

O Problema do *Vendor Lock-in*

3.1 O que é o *vendor lock-in*

Um dos grandes problemas da Nuvem é o *vendor lock-in*. Ou seja, quando uma aplicação consome os serviços de um determinado fornecedor de nuvem, já não será possível portá-la, de modo a tirar partido dos serviços de outro fornecedor [81, 62, 94]. Esta problemática advém das diferenças nos serviços oferecidos por cada fornecedor, maioritariamente nas suas APIs [50]. As primeiras referências a este problema surgem em 2009 por Chow *et al.* [25] e por Klems *et al.* [61], contudo o problema do *vendor lock-in* ganhou maior exposição em 2010 com a publicação de Armbrust *et al.* [8] (mais de 11.000 citações segundo o Google Scholar[1]). Este último trabalho considera o *vendor lock-in* como sendo o segundo maior obstáculo para a adoção da Nuvem, ficando entre as problemáticas da disponibilidade do serviço (em primeiro lugar) e da confidencialidade de dados (em terceiro). A adoção da Nuvem tem aumentado anualmente e prevê-se que assim continue [27], ou seja, cada vez mais programadores e organizações irão sofrer com este problema.

3.2 O que causa o problema

O consumo de serviços na Nuvem é concretizado, maioritariamente, via APIs REST (Representational State Transfer) [40] e SOAP (Object Access Protocol) [52], ambas assentes sobre HTTP (Hypertext Transfer Protocol). As APIs de serviços equivalentes em duas nuvens distintas, possuem diferenças nas suas implementações e nas suas decisões de desenho. Estas diferenças podem ser o formato de dados em que os pedidos devem ser codificados ou as funcionalidades que são oferecidas. São estas diferenças que causam o problema do *vendor lock-in*. Por modo a aprofundar a origem desta problemática, é de seguida feito um exemplo comparativo dos mecanismos de criação de máquinas virtuais nas nuvens Google Cloud Platform (GCP) e Amazon Web Services (AWS).

[1] https://scholar.google.pt/

Nuvem	Tipo de API	Formato de Dados	Parâmetros Query URL	Instâncias Criadas	Parâmetros no Corpo	São utilizados valores específicos ao fornecedor em algum parâmetro?
GCP	REST	JSON	Sim	1	Sim	Sim
AWS	SOAP	XML	Sim	0-100	Não	Sim

Tabela 3.1: Tabela comparativa das diferenças entre a função de criação de máquinas virtuais nas nuvens GCP e AWS.

3.2.1 Criação de máquinas virtuais na GCP e AWS

Para criação de uma máquina virtual, a GCP fornece uma função via API REST [45], a qual utiliza o formato de dados JSON (Javascript Object Notation) [33], e recebe dois parâmetros pelo URL [14] e um número arbitrário de parâmetros pelo corpo do pacote HTTP. Alguns destes parâmetros têm de obedecer a determinadas regras (p. ex. endereços de recursos específicos da GCP). Esta função só permite criar uma máquina virtual por chamada.

Em comparação, na AWS, é utilizada a função `RunInstances` [5]. Esta está definida numa API SOAP, sendo que os parâmetros são unicamente passados pelo URL (em número arbritrário), e o formato consumido é o XML (Extended Markup Language) [112]. Esta função permite lançar até 100 máquinas virtuais numa uma única chamada.

Tal como apresentado na **Tabela 3.1**, ambas as funções apresentam diferenças suficientes, que impossibilitam a portabilidade de uma aplicação que utilize a função de uma nuvem, para a outra, sem incorrer em esforços extraordinários.

O *vendor lock-in* afeta principalmente pequenas e médias organizações, que não dispõem dos meios e recursos necessários, para portar aplicações entre nuvens [64]. Uma solução para este problema seria uma mais valia, uma vez que abriria portas para situações de maior aproveitamento da Nuvem, tais como portar aplicações para regiões onde apenas um determinado fornecedor possui centros de dados, ou aproveitar os serviços do fornecedor com os custos mais baixos num dado instante, sem incorrer em esforços ou custos extraordinários de reescrita das aplicações.

3.3 Sumário

Neste capítulo foi explicado o problema do *vendor lock-in* e quais as suas causas. Este problema impossibilita a portabilidade de aplicações entre nuvens, impedindo programadores e organizações de beneficiarem da nuvem que melhor condições apresentar, em cada momento, para a exploração de aplicações. Também vimos que as diferenças entre nuvens ocorrem sobretudo ao nível das decisões de desenho, APIs e funcionalidades oferecidas. Uma solução para este problema tornaria mais fácil portar aplicações entre nuvens, sem ser preciso reescrever o código-fonte destas, e a decisão de qual a nuvem a utilizar seria movida do início do processo de desenvolvimento para a fase de produção.

Foi também apresentado um exemplo do problema, no que toca às APIs de criação de máquinas virtuais da Amazon Web Services e Google Cloud Platform.

Capítulo 4

A Solução Nomad SDK

Este capítulo apresenta a solução proposta neste trabalho para o problema do *vendor lock-in*, o Nomad SDK[1]. A **Secção 4.1** apresenta o contexto onde a solução se insere e os objetivos que nos propusemos a atingir. A linguagem Elixir, utilizada para o desenvolvimento do *sdk*, é apresentada na **Secção 4.2**, onde é feita também uma análise à sua complementaridade com a Nuvem. A arquitetura da solução é exposta na **Secção 4.3**, seguida da sua implementação e testes, nas **Secções 4.4** e **4.5**, respetivamente.

4.1 Contexto e Objetivos

O Nomad[2] SDK foi desenvolvido no contexto de resolver o problema do *vendor lock-in* [81, 62, 94]. Como indicado inicialmente, este trabalho evoluiu do desafio de desenvolver uma arquitetura compatível com a Nuvem para o S21sec Portal. Esta ideia foi evoluindo, passando pela hipótese de estudar como se poderia portar a aplicação entre nuvens, chegando ao trabalho presente, de desenvolver uma solução que permita a qualquer aplicação Elixir [108] usufruir de portabilidade entre nuvens.

O Nomad SDK visa solucionar o *vendor lock-in* para aplicações Elixir, oferecendo portabilidade entre nuvens através duma API única. Este *sdk* oferece ainda o benefício de remoção do código-morto, referente às nuvens não desejadas pelo programador. Estes dois fatores, a API única e a remoção do código-morto, fornecem várias vantagens:

1. **API única**:
   (a) Programação para as várias nuvens suportadas através da mesma biblioteca de funções.
   (b) Evita alterações ao código-fonte de uma aplicação, quando esta tem de ser portada para outra das nuvens suportadas. Uma chamada à API única reproduz um comportamento equivalente em qualquer uma destas.

[1] https://www.github.com/sashaafm/nomad

[2] O nome Nomad foi inspirado nas tribos nómadas do Irão, que exercem a tecelagem da tapeçaria persa. Ao serem nómadas, estas tribos estão constantemente a portar os seus trabalhos ao longo de várias regiões.

Isto também é benéfico para a tarefa de manutenção de aplicações, pois código-fonte que não necessite de alterações reduz a hipótese de serem introduzidos erros na aplicação.

(c) Reduz a necessidade do programador ter de possuir um tão elevado grau de conhecimento de cada nuvem. A API única abstrai detalhes da utilização das nuvens, tais como a construção e *parsing* de pedidos às APIs dos fornecedores de nuvem. Isto permite ao programador focar-se mais no desenvolvimento da aplicação.

(d) Possibilidade de suporte de outras nuvens. Para adicionar suporte a um novo fornecedor de nuvem, os programadores devem construir um novo adaptador que siga as especificações da API única. Tipicamente, isto será feito por um programador e utilizado por vários.

(e) A nuvem não tem de ser escolhida durante a fase de desenho da aplicação. Ao possuir uma API única, que garante a portabilidade da aplicação entre as nuvens suportadas, a necessidade de escolher qual o fornecedor de nuvem que se quer consumir é movida da fase de desenho para a fase de produção.

No caso do Nomad SDK, a aplicação pode ser desenhada e implementada utilizando a API única, e quando chegar à fase de produção, o programador apenas tem de alterar um ficheiro de configuração para indicar o fornecedor que deseja.

2. **Remoção de código-morto**:

(a) Auxilia a manutenção da aplicação, reduzindo o código-fonte da mesma. A administração e manutenção de qualquer tipo de código é esforço para os programadores, que o poderiam estar a direcionar para outros projetos ou implementações de futuras funcionalidades.

(b) Simplificação do código-fonte da aplicação. Um programador que tenha de manter uma aplicação não deve ter que raciocinar sobre código-fonte que nunca é utilizado.

(c) No caso da portabilidade entre nuvens, a remoção do código-morto justifica-se pelo facto deste não ser utilizado futuramente, visto que é referente ao consumo de um fornecedor de nuvem que não foi escolhido.

(d) Reduz a dimensão das aplicações.

As aplicações com serviços na Nuvem podem ser integradas de várias formas. Na **Figura 4.1**, são apresentadas três hipóteses distintas de integração, que são comparadas com os objetivos delineados no **Capítulo 1**.

Estas vantagens, oferecidas pelo Nomad SDK, proporcionam um melhor processo de desenvolvimento, quando comparado com outras hipóteses de processos de desenvolvimento de aplicações, que requeiram portabilidade entre nuvens. Estas hipóteses estão esquematizadas na **Figura 4.1**:

1. **Hipótese 1:** uma aplicação desenvolvida para consumir a API da AWS tem de utilizar um cliente para esta API. Isto faz com que determinados pontos do código-fonte desta aplicação façam chamadas através das funções da biblioteca deste cliente (representadas na **Figura 4.1** pela camada a vermelho).

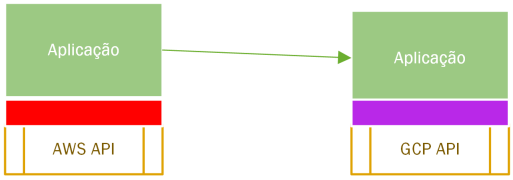
   Para portar esta aplicação para a GCP, é necessário alterar a aplicação para agora utilizar um cliente para a API desta nova nuvem. Os pontos que realizavam chamadas pelo cliente da AWS, têm agora de ser refeitos para utilizaram a biblioteca do cliente para GCP (camada a roxo na **Figura 4.1**).

   Este hipótese tem o benefício de não incluir código-morto, pois a camada para a AWS pode ser removida. Contudo, o programador teve de decidir sobre qual a nuvem a utilizar inicialmente durante a fase de desenho da aplicação. Houve, também, a necessidade de alterar o código-fonte da aplicação, para utilizar a nova camada de nuvem, pois não existia API única.

2. **Hipótese 2:** a aplicação é desenvolvida utilizando uma API única, que suporta ambas as nuvens. Para utilizar a nova nuvem, a aplicação teve de ser recompilada. O programador não teve de escolher, durante a fase de desenho, qual a nuvem que queria utilizar e pôde portar a aplicação sem a alterar. Contudo, a aplicação inclui código-morto. Alguns dos trabalhos relacionados apresentados no **Capítulo 7**, oferecem este processo de desenvolvimento.

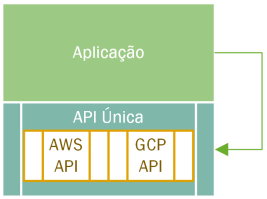
3. **Hipótese Nomad SDK:** a aplicação é construída com o Nomad SDK. A API única do *sdk* permite a implementação da aplicação, sem ser necessário raciocinar sobre a nuvem a utilizar, e permite que esta seja portada sem requerer alterações ao código-fonte. Ao ser portada para o novo fornecedor de nuvem, é alterado um ficheiro de configuração para indicar qual a nuvem escolhida, a aplicação é recompilada, e o código-morto removido automaticamente.

Hipótese 1:
Alteração completa de API p/GCP



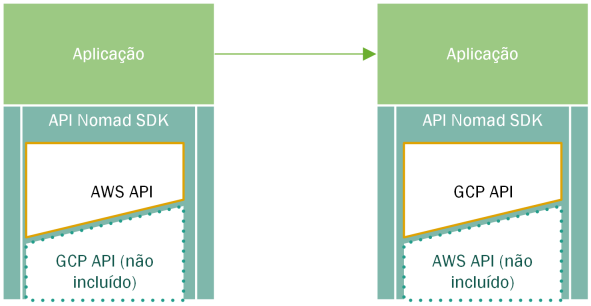
- necessário escolha de fornecedor em fase de desenho
- necessidade de alteração do código-fonte p/ chamar nova API da nuvem
- sem API única
+ não envolve código morto

Hipótese 2:
Utilização de nova nuvem via API única, sendo esta escolhida por configuração



+ sem escolha de fornecedor em fase de desenho
+ sem necessidade de implementação do código-fonte p/ chamar nova API da nuvem
+ com API única
- envolve código morto
- aplicação de maior dimensão

Hipótese Nomad SDK:
Utilização de nova nuvem via API única sendo esta escolhida por configuração.
O código-fonte é injetado na interface via meta-programação



+ sem escolha de fornecedor em fase de desenho
+ sem necessidade de implementação do código-fonte p/ chamar nova API da nuvem
+ com API única
+ não envolve código morto
+ aplicação de menor dimensão

Figura 4.1: Hipóteses para o problema da portabilidade entre as nuvens Amazon Web Services e Google Cloud Platform.

No **Capítulo 1**, foram definidos os objetivos (N1 a N4) específicos ao Nomad SDK. Como apresentado na **Tabela 4.1**, a *Hipótese Nomad SDK* é a única que atinge todos os objetivos traçados:

- **Hipótese 1:**
  - N1: não é atingido, pois não existe API única, mas sim uma camada referente à nuvem escolhida.
  - N2: não é atingido, pois as ferramentas devem ser alteradas para a nova nuvem.
  - N3: não é atingido, pois a camada referente à interação com a nuvem deve ser alterada, para suportar a nova nuvem.
  - N4: é atingido, visto que a camada referente à nuvem da qual se quer portar a aplicação é removida, sendo substituída pela camada da nova nuvem.
- **Hipótese 2:**
  - N1: é atingido, pois existe uma API única.
  - N2: é atingido, pois a API única pode ser utilizada para que as ferramentas suportem ambas as nuvens.
  - N3: é atingido, pois ao utilizar a API única evita-se alterar o código-fonte da aplicação.
  - N4: não é atingido, pois o código-fonte referente à nuvem de onde se portou a aplicação continua presente.
- **Hipótese Nomad SDK:**
  - N1: é atingido, pois existe uma API única,
  - N2: é atingido, pois a API única pode ser utilizada para que as ferramentas suportem ambas as nuvens.
  - N3: é atingido, pois ao utilizar a API única evita-se alterar o código-fonte da aplicação.
  - N4: é atingido, pois o código-fonte referente à nuvem escolhida é injetada na interface, sendo possível remover o código-morto da nuvem não desejada.

Hipótese	N1	N2	N3	N4
1	Não	Não	Não	Sim
2	Sim	Sim	Sim	Não
Nomad SDK	Sim	Sim	Sim	Sim

Tabela 4.1: Comparação dos objetivos atingidos por cada hipótese de integração de uma aplicação com os serviços na Nuvem.

4.2 Elixir

A linguagem de programação Elixir [108] é uma tecnologia criada em 2011 por *José Valim* antigo membro da equipa do Ruby On Rails [83, 84]. As suas aplicações são compiladas para o mesmo *bytecode* da máquina virtual do Erlang (ficheiros `.beam`) [104]. Tal como o Erlang, esta tecnologia é adequada ao desenvolvimento de aplicações concorrentes e distribuídas e que necessitem de elevadas taxas de fiabilidade. Este conjunto de tecnologias são complementares à natureza distribuída e concorrente da Nuvem (mais aprofundado em **Subsecção 4.2.5 Complementaridade com a Nuvem**). Esta linguagem de programação foi escolhida para o desenvolvimento do Nomad SDK por várias razões:

1. O Elixir, otimizado para a computação distribuída e concorrente, é indicado para o desenvolvimento para a Nuvem, tal como será discutido na **Secção 4.2.5**.

2. O S21sec Portal estava a ser desenvolvido nesta linguagem, pelo que uma das razões foi facilitar o desenvolvimento da solução, utilizando a mesma tecnologia.

3. Várias organizações da indústria têm cada vez mais utilizado a plataforma Erlang [37], para o desenvolvimento dos seus produtos, principalmente produtos onde existam sistemas distribuídos e concorrentes. Alguns destes produtos são o WhatsApp [76], os servidores multi-jogador do video-jogo Call of Duty [32], o *chat* em tempo real do Facebook [66], e algumas ferramentas utilizadas no Pinterest [111]. A WhatsApp, Facebook e Pinterest utilizam a computação na Nuvem, para os seus produtos, sendo que as duas primeiras organizações utilizam nuvens proprietárias e privadas e a última utiliza a AWS.

O Elixir pode ser visto como uma versão moderna do Erlang, tendo a mesma base, mas com uma nova sintaxe, funcionalidades e conceitos acrescentados, provenientes de outras linguagens. A funcionalidade mais distintiva é o paradigma de *meta-programação* [93] através de *macros*. Este conceito permite:

1. Adicionar mecanismos e *keywords* à linguagem, estendo-a [68].

2. Construção de código (por exemplo, funções) dinamicamente, através de dados externos.

Utilizando estes conceitos, é possível obter a injeção de código num módulo, sendo esse código definido fora deste. As funções são construídas dinamicamente em tempo de compilação, sendo inseridas no contexto desejado, para posterior utilização. Esta técnica foi utilizada extensivamente no Nomad SDK, a fim de possibilitar a remoção do código-morto.

4.2.1 Erlang e a BEAM VM

O Erlang [103], criado em 1986, é uma linguagem de programação dinâmica assente no paradigma funcional. As suas principais qualidades são o **suporte para concorrência** baseada no Actor Model [3], **comunicação**, **distribuição**, **tolerância a faltas**, **gestão automática de memória** e ***hot-code reloading*** [82, 2, 11, 9]. Estas características têm origem no domínio para o qual a linguagem foi feita para resolver problemas: o domínio das telecomunicações.

No Erlang os dados são **imutáveis** [2]. A imutabilidade facilita bastante a recuperação de erros, pois ao contrário do que acontece na programação imperativa, não é necessário reverter modificações feitas ao estado das variáveis do sistema, bastando recriar o processo com o estado original, que se mantém acessível nos argumentos que lhe foram passados.

A máquina virtual do Erlang, a BEAM, executa como um único processo do Sistema Operativo (SO) garantindo máxima utilização da máquina. Adicionalmente, no Erlang, existem processos implementados ao nível da máquina virtual, os chamados atores (*actors*), que não têm qualquer relação com os processos do SO [9, 82]. A BEAM pode então ser vista como uma máquina virtual de processos.

Os processos do Erlang utilizam um Process Control Block (PCB)[3] [17] com cerca de $\frac{1}{3}$ do tamanho dos PCB Unix (300 *bytes* vs. 1024 *bytes* [2]), o que permite que sejam mantidos mais processos na mesma quantidade de memória. Cada processo tem memória independente, um *heap* de memória para uso próprio e nunca partilhado com outros processos. A única forma de comunicação que os processos possuem são mensagens entre si. Ao não existir estado global estado partilhado entre processos, os erros que levem à falha de um processo ficam isolados no contexto deste, não impedindo geralmente a continuidade da operação dos restantes processos. É assim promovida a isolação de faltas [9, 2].

A plataforma Erlang conta com Garbage Collection (GC) *non-stop the world*[4], sendo esta concretizada pelo próprio processo. A vantagem de uma GC *non-stop the world* reside no melhor desempenho da aplicação, por não ter de ser pausada aquando da limpeza realizada por este mecanismo.

As características descritas relativamente aos processos e Garbage Collection, permitem a criação de um elevado número de processos com um mínimo de recursos e alcançando um comportamento *soft-real time* [18], uma vez que enquanto a GC está em execução sobre um dado processo, os restantes podem continuar a executar a lógica aplicacional paralelamente [2]. Uma vez que um processo pode ter um tempo de vida de tal

[3]Um PCB é uma estrutura de dados que guarda informação pertinente a um dado processo, tal como identificador do processo pai, valores de registos, ou localização dos recursos que lhe pertencem.

[4]*Non-stop the world* significa que a *garbage collection* não pausa a aplicação, para realizar a limpeza da memória da mesma.

forma curto (na programação em Erlang é habitual a criação de processos para executar a maior parte das tarefas), faz com que raramente requisitem memória por extensos períodos de tempo. Por isto, é vulgar que existam processos que nunca cheguem sequer a passar por um único ciclo de GC [82], dado que são terminados antes deste procedimento ocorrer.

4.2.2 *Let It Crash*

O Erlang Run-Time System (ERTS) [38] implementa mecanismos de monitorização e reinício dos processos. Estes mecanismos promovem a filosofia *let it crash* do Erlang e a *Programação Orientada à Concorrência* [9]. Uma aplicação Erlang assente neste paradigma de programação resulta em árvores de processos, onde cada processo é supervisor dos seus filhos e onde cada sub-árvore tem uma estratégia de recuperação, sendo que os processos supervisionados são recuperados pelos seus supervisores. Desta forma a lógica de recuperação de erros fica separada da restante lógica da aplicação. Esta filosofia incita a não programar defensivamente, mas sim a tornar o sistema resiliente, autónomo e fiável. Na **Figura 4.2**, está esquematizado o fluxo de recuperação de um sistema Erlang. Caso o supervisor que está a tentar recuperar o processo também falhe, a falha vai sendo propagada pela árvore de supervisão (tentando agora o supervisor do supervisor reiniciar o mesmo, para que este depois reinicie o processo que falhou originalmente). Caso a falha se propague até à raiz da árvore de supervisão, então o que acontece irá depender do modo como a aplicação foi iniciada [106]. Estes mecanismos de recuperação e supervisão são aplicáveis a faltas que gerem falhas nos processos do sistema.

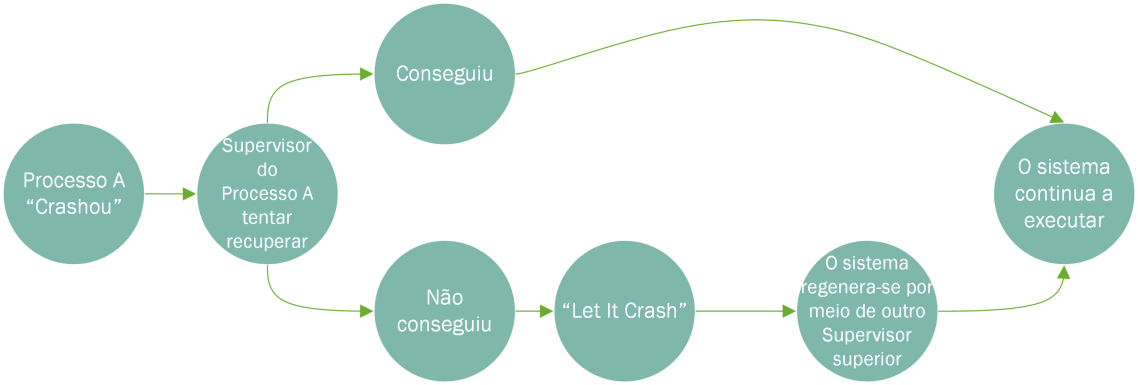


Figura 4.2: Fluxo da recuperação de um sistema Erlang por meio dos mecanismos de supervisão e monitorização

Em contraste às linguagens de programação imperativas, cada processo tem atribuído um objetivo específico, pois os processos trabalhadores (*workers*) realizam as tarefas e os processos supervisores (*supervisors*) observam e recuperam os anteriores em caso de erro, tornando a separação de preocupações clara e explícita. Numa linguagem imperativas as tarefas e mecanismos de recuperação (por exemplo, *constructs try* e *catch*) coexistem no

mesmo processo ou *thread* de execução. Estes mecanismos tentam recuperar a *thread* para um estado consistente, ao invés que no Erlang com o *let it crash*, é logo imposto um reinício limpo. Apesar dos mecanismos *try* e *catch* também existirem no Erlang, isto só é recomendado para quando o programador souber como recuperar do erro. As exceções, tal como o nome indica, representam a ocorrência de uma situação não esperada e continuar a executar o sistema, num estado erróneo ou não previsto aquando do desenho do mesmo, pode ser considerado uma má prática [53].

4.2.3 Meta-programação

A meta-programação é a construção de programas que geram código ou outros programas. Os programas são consumidos como dados, sendo a meta-linguagem capaz de os modificar [29]. No Elixir, a meta-programação é alcançada por meio de uma *abstract syntax tree* (AST) [59], sendo esta a representação interna das expressões da linguagem. A AST é constituída por um tuplo de três valores, onde o primeiro é o nome da função, o segundo uma lista com meta-dados relevantes e o terceiro uma lista de argumentos [68]. Como pode ser visto na **Listagem 4.1**, aplicando o mecanismo `quote` à expressão `1 + 2`, o resultado da representação interna é constituído por um tuplo de três valores:

1. **Nome da função:** Neste caso a função com o nome `+`.
2. **Meta-dados:** É recebida uma lista com o contexto da expressão e a importação do módulo `Kernel`, ao qual a função + pertence.
3. **Lista de argumentos:** Os argumentos que são utilizados na expressão, neste caso `1` e `2`.

Estes tuplos podem ser compostos e manipulados, para construir novos comportamentos, expressões ou código.

```
iex> quote do: 1 + 2
{:+, [context: Elixir, import: Kernel], [1, 2]}
```

Listagem 4.1: A expressão `1 + 2` e a AST que a representa.

As *macros* têm vantagens em relação às funções, nos seguintes casos:

1. As *macros* executam em tempo de compilação, enquanto as funções executam em tempo de execução. Por exemplo, as *macros* no Elixir podem ser aproveitadas para gerar funções com base em informação externa.
2. Criação de Domain Specific Languages (DSL). As *macros* permitem estender o Elixir, criando novas *keywords* e mecanismos. Por exemplo, a DSL de *queries* e manipulação de bases de dados SQL do projeto Ecto [35] foi criada através de *macros*.

3. Permite esconder, criar atalhos e evitar repetição de código (*boilerplate*). Por exemplo, a Phoenix Framework[5], para a criação de aplicações Web em Elixir, com o padrão arquitetural Model-View-Controller[21], permite invocar *macros*, para mais facilmente alcançar o código que é necessário em cada tipo de módulo (controladores, modelos e vistas), por modo a aliviar o programador de ter de estar constantemente a incluir esse código em cada módulo que escreve.

A meta-programação foi utilizada no Nomad SDK, para alcançar a injeção do código do adaptador da nuvem escolhida, e assim obter uma API única sem código-morto (detalhado na **Secção 4.3 Arquitetura**).

4.2.4 Complementaridade com a Nuvem

As telecomunicações e a Nuvem partilham um problema de domínio semelhante. Em ambas as áreas estão presentes ***sistemas de software complexos***, em ***operação contínua com constantes atualizações***, sujeitos a ***requisitos de qualidade exigentes***, tais como a capacidade de ***tolerar faltas*** [9]. O Erlang é uma linguagem apropriada à resolução destes problemas [9, 10]. Na **Tabela 4.2** são apresentados os problemas do domínio da Nuvem e as características da plataforma Erlang que ajudam a resolvê-los.

[5]https://www.phoenixframework.org

Tabela 4.2: Problemas do domínio e as características do Erlang que os resolvem (adaptado de [9]).

Requisito	**Solução**
Sistemas Complexos	Encapsulamento de código e dados em módulos separados
Sistemas Distribuídos	Troca de mensagens entre processos transparente à localização destes (local ou multi-máquina)
Atualizações em operação contínua	Atualizações de código durante a execução (*hot-code reloading*)
Tolerância a Faltas	Supervisão de processos; isolamento de erros; atualizações de código durante a execução
Tempos de latência baixos	Cada processo tem a sua zona de memória (*heap*); recolha de lixo por processo (*non-stop the world garbage collection*)

As aplicações Erlang e Elixir podem ser desenvolvidas numa única máquina e serem migradas para um sistema distribuído com apenas algumas alterações às configurações [102]. A arquitetura por processos, garante que o comportamento e tratamento de erros de um processo será sempre o mesmo em qualquer configuração. Desta forma, é também possível tornar um sistema resiliente a nível de falhas do *hardware*, através da execução dos processos trabalhadores e supervisores em máquinas distintas [9].

No trabalho de Nagappan *et al.* [74], foi identificada uma taxa de falhas de *hardware* anual, da infraestrutura da Nuvem, na ordem dos 8%. O Erlang tem a capacidade de aliviar este problema, pois através dos seus mecanismos de supervisão em sistemas distribuídos entre várias máquinas, permite que o sistema continue a operar, mesmo aquando da falha de um dos nós do mesmo.

Outro ponto de complementaridade encontra-se na escalabilidade da Nuvem, uma característica cada vez mais relevante para os utilizadores das nuvens [110]. Consideremos a escalabilidade, como sendo o grau com que um sistema consegue lidar com subidas na carga de trabalho:

1. Cada processo concorrente requisita recursos de um dado sistema, logo, quanto mais recursos cada processo requisitar, menos processos se poderão alojar num dado sistema. A baixa dimensão dos processos em Erlang permite a esta plataforma criar e alojar mais processos num dado sistema, em comparação com o que seria possível com processos do SO. Isto beneficia a escalabilidade do sistema, pois é possível criar mais processos, para lidar com os aumentos na carga trabalho, no mesmo conjunto de recursos.

2. As aplicações na plataforma Erlang são criadas com base em processos concorren-

tes, cuja distribuição é realizada automaticamente pelos vários núcleos dos processadores da máquina, pelo que o seu grau de escalabilidade aumenta transparentemente com o número de núcleos. Adicionalmente, a falta de estado partilhado e de secções críticas, em conjunto com a técnica de envio de mensagens assíncronas, permite que o sistema escale ao serem alocados mais núcleos. Também, ao existirem padrões comportamentais para concorrência pré-programados, a programação de sistemas paralelos é facilitada.

Ao aumentar o grau de escalabilidade de uma aplicação, é possível tentar atacar um problema (relacionado com a Nuvem) identificado na literatura: a criação de aplicações cuja necessidade de escalar, através do consumo de mais recursos, seja minimizada [58, 87]. O facto dos processos em Erlang serem leves faz com que as aplicações exijam menos recursos da Nuvem.

4.3 Arquitetura

O Nomad SDK está construído e organizado pelos serviços e fornecedores que suporta, tendo para isto, uma arquitetura multi-camada, que separa na camada superior a interface única para os três serviços suportados, na camada intermédia a abstração dos serviços de cada fornecedor e na camada inferior os clientes para o consumo direto dos mesmos, como apresentado na **Figura 4.3**.

Figura 4.3: Diagrama de camadas do Nomad SDK

Esta arquitetura está organizada da seguinte forma:

1. **Camada Superior:** formada pelo módulo base `Nomad`, as interfaces das APIs de cada serviço — `Nomad.Storage`, `Nomad.SQL` e `Nomad.VirtualMachines` — que irão ser injetadas com o código relativo ao fornecedor de nuvem via meta-programação.

2. **Camada Intermédia:** nesta camada encontram-se os *adaptadores* específicos para os serviços de cada fornecedor. São estes módulos que irão injetar o código via meta-programação, correspondente ao fornecedor escolhido. Este código é injetado nas interfaces da camada superior — nos módulos `Nomad.Storage`, `Nomad.SQL` e `Nomad.VirtualMachines` — para que em tempo de execução as funções da API estejam definidas e possam ser utilizadas nestes módulos.

3. **Camada Inferior:** formada pelos clientes específicos a cada fornecedor de nuvem. Estes clientes servem para consumir as APIs dos fornecedores, realizando pedidos HTTP e recebendo as respetivas respostas. Estes clientes são utilizados nos adaptadores, para implementar o código-fonte específico a cada fornecedor, que será injetado nas interfaces, formando assim a API única.

Na **Figura 4.4** podemos ver como estes três módulos interagem e quais as relações que têm entre si. As camadas denotam também o fluxo de execução do Nomad SDK. As *keywords* `use` e `import` na imagem são as relações entre os módulos:

1. `import`: permite invocar as funções do módulo importado, sem ter que escrever o seu caminho. Por exemplo, uma função `abc` de um módulo `Alphabet`, tem de ser invocado da seguinte forma: `Alphabet.abc`. Utilizando `import Alphabet`, passamos a poder invocar a função pelo seu nome apenas, ou seja, invocando `abc`.

2. `use`: este mecanismo faz com que o código definido dentro da *macro* `__using__` de um dado módulo, seja injetado no módulo que o invocou através do `use`. Este mecanismo será melhor detalhado ao longo deste capítulo, por modo a explicar a injeção do código via meta-programação.

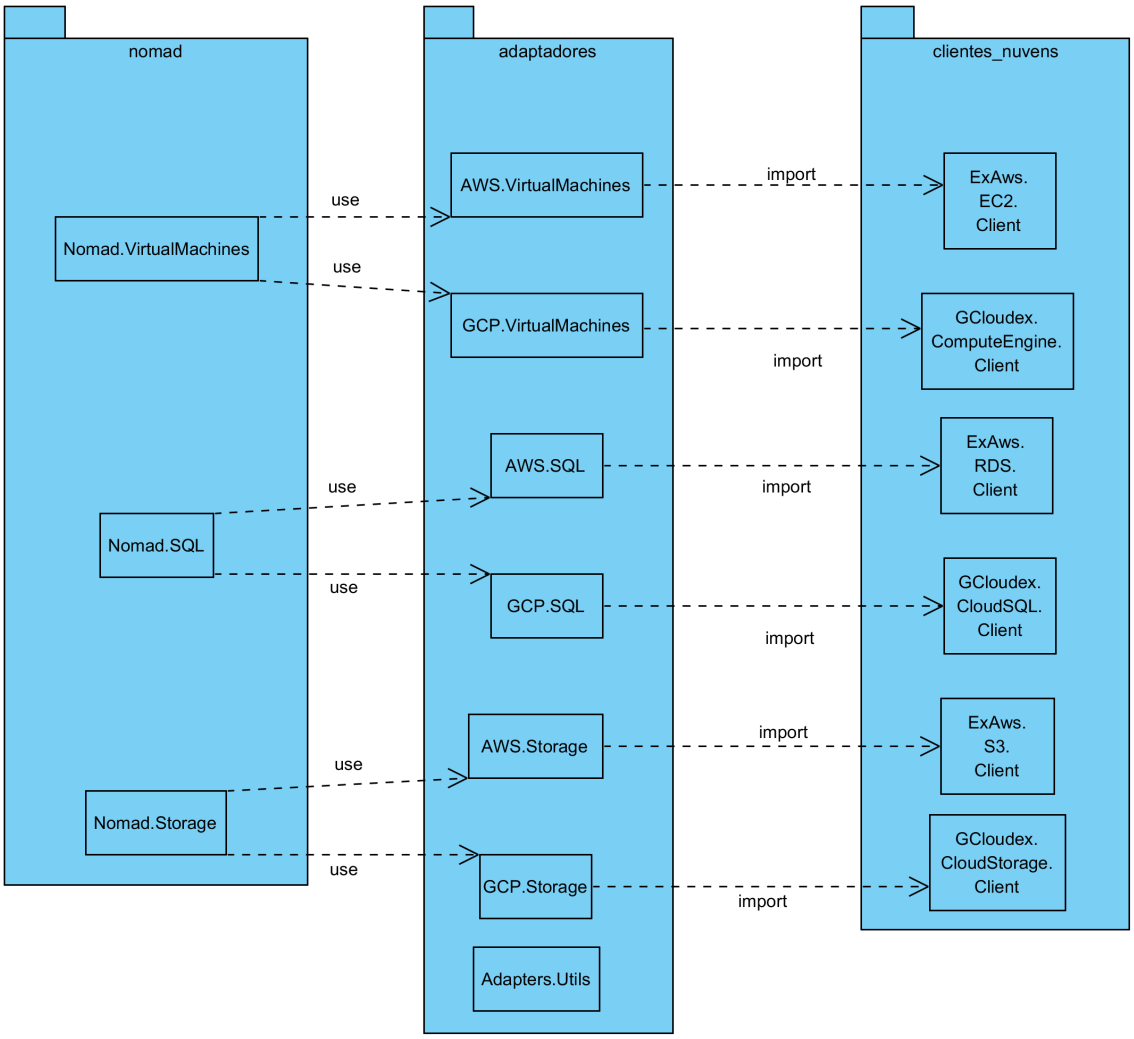


Figura 4.4: Diagrama de relações entre as camadas do Nomad SDK e seus módulos.

4.3.1 Estilos Arquiteturais

Um estilo arquitetural é um conjunto de decisões de desenho, para arquiteturas de sistemas informáticos. Estas decisões, na forma de componentes e conectores, oferecem um conjunto de qualidades ao sistema, quando o desenho é aplicado na sua arquitetura [43, 99, 73]. Foram aplicados vários estilos arquiteturais no Nomad SDK, por modo a atingir os objetivos definidos.

Camadas

A nossa solução conta com três camadas: a interface, adaptadores e clientes específicos de cada nuvem. Cada uma destas camadas tem uma responsabilidade única e separam as preocupações entre si. É alcançado assim um estilo arquitetural *multilayered architecture* [22].

REST, SOAP e SOA

A comunicação e pedidos às nuvens são concretizados via HTTP. Esta utilização de serviços Web é uma forma de implementação de uma Arquitetura Orientada a Serviços (Service-Oriented Architecture ou SOA) [72]. A troca de dados entre os clientes e as nuvens é feita seguindo o estilo arquitetural REST [40][6] e o protocolo SOAP [52].

Plug-in Oriented

O estilo arquitetural Plug-ins (ou Plug-in Oriented) [98] é também aplicado no Nomad SDK. Este padrão dita que um *plug-in* é uma componente que estende ou altera o comportamento e funcionalidades de um dado *software*. Os *plug-ins* podem ser desenvolvidos separadamente e posteriormente acoplados à base. No Nomad SDK os *plug-ins* são os adaptadores para a interação com cada nuvem, podendo estes serem desenvolvidos externamente e de seguida acoplados ao SDK, por modo a oferecer as funcionalidades da API única, para um novo fornecedor de nuvem. Estas alterações são feitas pelos adaptadores dos fornecedores de nuvem, atuando como os *plug-ins* que alteram o comportamento da API, aquando do uso das interfaces.

4.3.2 Camada Superior: Nomad

As especificações que o código implementado pelos adaptadores devem seguir surgem sob a forma de *Behaviours*, um conceito Elixir semelhante à *Interface* em Java [102, 109]. No Nomad SDK, os Behaviours especificam as funções das APIs dos serviços de armazenamento de ficheiros, bases de dados SQL e máquinas virtuais. Estas especificações devem ser implementadas pelos adaptadores, para que os argumentos e retornos das funções sejam do mesmo tipo. Isto tem o propósito de garantir a portabilidade e utilidade, de uma aplicação, entre nuvens, pois os detalhes específicos são assim abstraídos, e as componentes de código passam a seguir estruturas e regras bem definidas e sempre iguais.

Por exemplo, a funcionalidade de listagem dos itens inseridos num determinado armazenamento é comum a ambas as nuvens. No Behaviour apropriado (`Nomad.Storage`) foi especificada uma função `list_items/1`[7], que representa esta funcionalidade. Como se pode ver na **Listagem 4.2**, a especificação desta função indica que deve ser passado um argumento do tipo `binary` (i.e., uma *string*), que será o nome do *storage* cujos itens devem ser listados, e deverá ser devolvido um de três retornos:

1. **`[]:`** uma lista vazia (caso não existam itens a listar).
2. **`[binary]:`** uma lista de `binary` (caso existam itens a listar).
3. **`binary:`** um único `binary` (caso ocorra um erro a nível do serviço, como um `404: Not Found`, ou um problema no pedido ligação HTTP).

[6] Roy Fielding, na tese onde apresentou o REST, definiu este como sendo um estilo arquitetural.
[7] A sintaxe `fun/n` significa que a função `fun` tem aridade `n`, ou seja, recebe `n` argumentos.

```
...

@doc"""
Lists all the available files in the given 'storage'.
"""
@callback list_items(storage :: binary) :: [] | [binary] |
   binary

...
```

Listagem 4.2: Excerto do Behaviour para API de armazenamento de ficheiros. A função `list_items/1` aparece especificada, com documentação textual, especificação da assinatura da funções com os tipos dos argumentos e os tipos de retornos possíveis.

Estas especificações devem ser consideradas em tempo de desenvolvimento, uma vez que garantem que independentemente da nuvem escolhida, os tipos programáticos dos *inputs* e *outputs* serão sempre os mesmos.

Contudo, algumas destas funcionalidades existem numa das nuvens mas não nas outras, pelo que, como será explicado na **Secção 4.4 Implementação**, algumas das funções implementando as especificações tiveram de ser alcançadas por outros meios, para alcançar o mesmo comportamento.

4.3.3 Camada Intermédia: Adaptadores

A ideia de geração de código a partir de adaptadores foi inspirada no projeto *Ecto*[8] [35]. No ficheiro de configuração do Nomad SDK, foi definido um par *chave-valor* `cloud_provider`, o qual indica o fornecedor desejado. Na **Listagem 4.3** é apresentado um excerto do ficheiro de configuração, utilizando a Google Cloud Platform como fornecedor de nuvem.

```
use Mix.Config

config :nomad, cloud_provider: :gcl
```

Listagem 4.3: Excerto de um possível ficheiro de configuração para o Nomad SDK. A chave indicadora do fornecedor escolhido encontra-se na última linha. Foram escolhidos os valores `:aws` para a Amazon Web Services e `:gcl` para a Google Cloud Platform.

Esta chave é utilizada nos módulos da API única que irão receber o código injetado pelo adaptador. Para isto, uma expressão condicional é avaliada, para os valores possíveis

[8] O Ecto é um projeto em Elixir, que utiliza adaptadores e meta-programação, por forma a criar uma camada de abstração para vários motores SQL, fornecendo também uma Domain Specific Language para interrogações e manipulação das bases de dados.

da chave. Caso o valor no ficheiro de configuração esteja indicado na condicional, o módulo irá utilizar a *keyword* `use` com o adaptador necessário, como pode ser visto na **Listagem 4.4**. Esta *keyword* invoca a *macro* `__using__`, uma *macro* especial no Elixir, através da qual é possível incorporar funcionalidades externas no contexto onde o `use` foi invocado [105]. É a partir deste mecanismo que o código de cada adaptador é injetado no módulo destino.

```
defmodule Nomad.Storage do

  ...

  case Application.get_env(:nomad, :cloud_provider) do
    :aws -> use Nomad.AWS.Storage, :aws
    :gcl -> use Nomad.GCL.Storage, :gcl
  end
end
```

Listagem 4.4: Módulo `Nomad.Storage` da camada superior, que irá receber o código gerado por um dos adaptadores da camada intermédia. A expressão `case do` é a expressão condicional que invoca a *macro* `__using__` do adaptador da nuvem escolhida. A função `Application.get_env(:nomad, :cloud_provider)` verifica qual o valor da chave `:cloud_provider` definida no ficheiro de configuração.

Na **Listagem 4.5** é apresentado um excerto do adaptador de armazenamento de ficheiros para a GCP. A *macro* `__using__` definida tem no seu corpo as implementações das funções especificadas no Behaviour adequado (`Nomad.Storage`, apresentado na **Listagem 4.2**). A *keyword* `quote` transforma todo o código no seu interior na representação interna do Elixir, que em tempo de compilação é injetada no módulo que invoca o `use`, e em tempo de execução usufrui das funções do código injetado. A linha `import GCloudex.CloudStorage.Client`, na **Listagem 4.5**, realiza a importação das funções do módulo para consumo do serviço Google Cloud Storage, presente na dependência GCloudex da terceira camada.

```
defmodule Nomad.GCL.Storage do

  @moduledoc"""
  Google Cloud Storage adapter for Nomad. API interaction is
     done through
  GCloudex.
  """

  defmacro __using__(:gcl) do
    quote do
      # API functions will be used from this client
      import GCloudex.CloudStorage.Client
      import Adapters.Utils

      def list_storages(fun \\ &list_buckets/0) do
        case fun.() do
          {:ok, res} ->
            case res.status_code do
              200 ->
                res.body
                |> Friendly.find("name")
                |> Enum.map(fn bucket -> bucket.text end)
              _  ->
                res |> show_error_message_and_code
            end
          {:error, reason} ->
            parse_http_error reason
        end
      end
  ...
  end
end
```

Listagem 4.5: Excerto inicial do adaptador do armazenamento de ficheiros da Google Cloud Platform (serviço Google Cloud Storage). De notar a implementação da *macro* `__using__`. Podemos também ver o `import` do módulo `Adapters.Utils`, que tem funções para o tratamento de respostas com erros, tais como as funções `show_error_message_and_code` e `parse_http_error`.

4.3.4 Camada Inferior: Clientes das Nuvens

O Nomad SDK consome as APIs dos fornecedores de nuvem, por meio de clientes Elixir para o efeito. Na **Listagem 4.5**, da secção anterior, está presente a importação do `GCloudex` que será utilizado pela *macro* *__using__*. Esta aplicação é uma biblioteca de clientes Elixir, para consumo da Google Cloud Platform. Tal como esta biblioteca, o Nomad SDK utiliza uma equivalente para a Amazon Web Services, a ExAWS. Estas duas

aplicações irão realizar a interação com as nuvens, que como indicado anteriormente, é concretizado por intermédio de serviços Web, maioritariamente APIs REST (*Representational State Transfer*) e SOAP (*Simple Object Access Protocol*). O transporte dos pedidos a estas APIs é realizado por *Hypertext Transfer Protocol* (HTTP). Estes clientes realizam os pedidos e recebem as respetivas respostas, que serão transformadas na camada intermédia, os adaptadores.

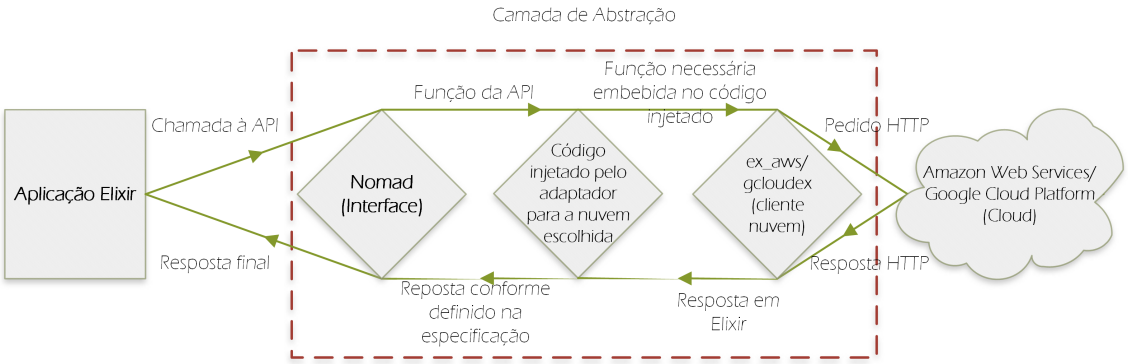


Figura 4.5: Fluxo de dados do Nomad SDK. Um pedido através da API única invoca o código injetado pelo adaptador, que utiliza o cliente Elixir para a nuvem escolhida. A resposta é devolvida ao cliente e transformada no adaptador, para garantir que a especificação da função da interface é obedecida. Por fim os dados são devolvidos à aplicação, que utiliza o Nomad SDK.

Na **Figura 4.5** pode ser visto o fluxo de dados do Nomad SDK, incluindo a utilização das dependências externas (os clientes Elixir para consumo das nuvens). Como referido, foram utilizadas duas bibliotecas Elixir, para consumo das duas nuvens suportadas:

1. **GCloudex [41]:** para consumo da Google Cloud Platform
2. **ExAWS [24]:** para consumo da Amazon Web Services

GCloudex

O GCloudex[9] é uma coleção de componentes Elixir, para o consumo das APIs dos serviços oferecidos pela Google Cloud Platform (GCP). Esta biblioteca foi desenvolvida no âmbito deste trabalho, uma vez que até ao momento do início da implementação do Nomad SDK, não existia nenhuma outra em Elixir para a GCP.

Esta biblioteca de clientes implementa por completo as APIs dos serviços Google Cloud Storage, Google Cloud SQL e Google Compute Engine. Aquando da escrita deste relatório, o GCloudex contava com 95 *commits*[10], 11 ”estrelas”[11], 2 *forks*[12] e 2 contribuidores no seu repositório no GitHub.

[9]https://www.github.com/sashaafm/gcloudex

[10]*Commit* significa uma gravação do estado do código num dado momento.

[11]No GitHub, as estrelas têm o propósito de indicar quais e quantos utilizadores estão a seguir o repositório e têm interesse no mesmo.

[12]Um *fork* é uma cópia integral do repositório, para a conta de um outro utilizador, que o pode modificar

ExAWS

O projeto ExAWS[13] é, semelhantemente ao GCloudex, uma biblioteca para consumo das APIs dos serviços da Amazon Web Services. O ExAWS é mantido pela empresa de SaaS para logística e fornecimento, chamada CargoSense[14]. Inicialmente, este projeto apenas oferecia o consumo do serviço de armazenamento de ficheiros da Amazon. Desta forma, foram feitas duas contribuições no âmbito deste trabalho para o repositório do projeto: a implementação de um cliente para consumo do serviço SQL Amazon RDS e do serviço de infraestrutura Amazon EC2. O ExAWS contava, aquando da escrita deste relatório, com 488 commits, 160 ”estrelas”, 58 forks e 24 contribuidores no seu repositório no GitHub.

4.4 Implementação

Feitas as especificações da API única do Nomad SDK, o esforço de implementação residiu maioritariamente na camada intermédia, ou seja, no desenvolvimento dos adaptadores. Estes foram desenvolvidos com dois objetivos em mente:

1. Garantir o comportamento desejado em cada função, e a conformidade desta com a especificação da API única. Os adaptadores tiveram de ser desenvolvidos fazendo uso das funcionalidades de meta-programação, por modo a terem o seu código injetado nas interfaces desta API.

2. Estar completamente desacoplados entre si e da camada superior, para possibilitar a remoção do código-morto.

4.4.1 API única

Como demonstrado no **Capítulo 3**, as diferenças entre os serviços das nuvens são acentuadas. Por exemplo, na API do serviço de máquinas virtuais do *Google Cloud Platform*, a função de criação de um disco virtual necessita que o nome desejado para o disco seja especificado. Em contraste, nos *Amazon Web Services* a função de criação de um disco virtual impossibilita a escolha de um nome para o mesmo (atribui um identificador automaticamente).

Por forma a obtermos uma API única, simples de usar, e que não requeira alterações às aplicações aquando da troca de fornecedor, é necessário que os argumentos a passar sejam os argumentos mínimos comuns de ambas as Nuvens. No exemplo referido, a limitação no adaptador da *Google Cloud Platform* foi contornada gerando uma *string*, para o nome

à sua vontade, sem interferir no repositório original. Costuma ser feito para isolar correções de problemas ou propor novas funcionalidades, por exemplo.

[13] https://www.github.com/cargosense/ex_aws

[14] http://www.cargosense.com/

do disco virtual, com o formato `disk-<epoch>`, onde `<epoch>` é o valor do *UNIX Epoch* [79].

Outras diferenças tiveram de ser superadas ao nível do comportamento das nuvens. Por exemplo, o serviço de bases de dados SQL da Google Cloud Platform realiza os pedidos independentemente da região em que a base de dados se encontra. No pedido é enviado o nome da base de dados, que será procurada no contexto da conta de utilizador que realizou o pedido, não sendo passados quaisquer parâmetros relacionados com a região. Em contraste, o serviço equivalente da Amazon Web Services realiza os pedidos segundo a conta de utilizador, o nome da base de dados e a região, para onde o pedido deve ser feito. Apesar do real funcionamento de ambos os sistemas não ser completamente claro, os pedidos para a AWS aparentam ter de ser enviados diretamente para o centro de dados onde o recurso se encontra, enquanto que os pedidos para a GCP parecem ser enviados para um sistema mais centralizado, que trata de encaminhar os pedidos para a região apropriada.

Para tornar o comportamento de ambos os adaptadores idênticos, os pedidos realizados à AWS tiveram de ser construídos de modo a percorrer todas as regiões suportadas, dispensando região onde se encontra a base de dados (o que faria com que a função da API única recebesse um argumento inútil, quando utilizada para o serviço de bases de dados SQL da Google Cloud Platform). Apesar da função para a AWS ficar com um baixo grau de eficiência, esta foi a única forma encontrada para garantir um comportamento equivalente em ambas as nuvens.

Para superar estas diferenças entre as APIs das várias nuvens foram seguidas duas estratégias durante o desenho da API única:

1. **Ao encontrar as funcionalidades semelhantes:** funções das APIs de ambas as nuvens que fossem suficientemente semelhantes podiam ser retratadas através da assinatura de uma função na API única. Por exemplo, consideremos uma função, existente em ambas as nuvens, que retorna a descrição de um determinado recurso, cujo único parâmetro é o identificador desse recurso. Esta função é facilmente retratada na API única, pois pode ser construída a assinatura de uma função, que recebe o argumento com o identificador, e cuja especificação indica que o retorno seja uma *string* com a descrição do recurso, ou o valor de `-1`, caso o recurso não exista. Posteriormente, no adaptador de cada fornecedor, a função deve ser implementada, de modo a que garanta a especificação e para que retorne o resultado pretendido.

2. **Ao superar as diferenças entre funcionalidades:** como indicado, nem todas as funções são semelhantes o suficiente para que seja fácil abstrair as diferenças entre estas. Por vezes foi necessário procurar comportamentos que fossem possíveis de reproduzir em ambas as nuvens, e a partir desse comportamento construir a assinatura para uma função da API única, para posteriormente implementar.

A API única apenas procura modelar as funcionalidades que são comuns, ou possíveis de tornar comuns, a ambas as nuvens. O Nomad SDK não oferece funcionalidades específicas a cada nuvem, propositadamente, uma vez que a sua utilização iria quebrar a portabilidade. Isto acontece pois ao construir uma aplicação que utilize uma funcionalidade específica, caso esta seja portada para uma outra nuvem, a funcionalidade específica deixará de operar. A inclusão de funcionalidades específicas obrigaria o programador a alterar o código-fonte de alguma forma. O programador ao utilizar o Nomad SDK deve fazer o compromisso de ganho de portabilidade, pela perda das funcionalidades específicas de cada nuvem.

4.4.2 Remoção do código-morto

Ao desenhar uma arquitetura, que aplica um estilo arquitetural por camadas e por *plug-ins*, e que aproveita a meta-programação, para realizar a injeção do código nas interfaces da camada superior, foi possível excluir o código-morto da API única. Isto deve-se ao desacoplamento dos adaptadores entre si e em relação à camada superior. O código-morto pode ser removido, eliminando o *bytecode* onde reside, pois todo este *bytecode* estará encapsulado em *plug-ins* individuais, com os quais o sistema não terá quaisquer dependências.

Por exemplo, os módulos `Nomad.Storage`, `Nomad.SQL` e `Nomad.VirtualMachines` não possuem funções definidas dentro de si, apenas invocam o `use` com o adaptador da nuvem definida no ficheiro de configuração. Contudo, quando iniciamos o Nomad SDK, ele injeta o código do adaptador do fornecedor escolhido nestes três módulos da camada superior. Isto pode ser verificado na consola interativa do Elixir [107], onde o mecanismo de completar automaticamente (*autocompletion*) a linha de comandos, mostra todas as funções da API única, depois de terem sido injetadas a partir do código do adaptador, tal como apresentado na **Listagem 4.6**.

```
Interactive Elixir (1.2.3) - press Ctrl+C to exit (type h()
   ENTER for help)
iex(1)> Nomad.Storage.
create_storage/3    delete_item/2    delete_storage/1
get_item/2          get_item_acl/2   get_storage_acl/1
get_storage_class/1 get_storage_region/1 list_classes/0
list_items/1        list_storages/0     put_item/3
```

Listagem 4.6: Ao pedir as funções possíveis do módulo `Nomad.Storage` na consola interativa, as funções da API de armazenamento de ficheiros surgem como opção, pois foram injetadas via meta-programação.

Estes módulos da camada superior podem agora usufruir do consumo da nuvem escolhida. Os ficheiros `.beam`, específicos aos adaptadores das nuvens não escolhidas, po-

dem então ser removidos do sistema, não afetando de qualquer forma, o funcionamento da aplicação que utiliza o Nomad SDK. A remoção pode ser automatizada através de um *script* para o efeito.

Até ao momento da escrita deste documento, o Nomad SDK apenas suportava as nuvens AWS e GCP. Contudo, caso no futuro seja adicionado suporte para outros fornecedores de nuvem, a remoção de código-morto será cada vez mais vantajosa, visto que com cada fornecedor suportado, mais código-morto estaria presente nas aplicações.

A menor dimensão das aplicações é uma vantagem para as mesmas em geral, e para a construção de aplicações em sistemas embebidos, com Erlang e Elixir (por exemplo, através do projeto Nerves[15]), cuja memória disponível é, por norma, mais reduzida do que nas aplicações Web ou *desktop*. Com o advento da Internet das Coisas [57], este benefício é cada vez mais relevante.

4.5 Testes

As baterias de testes do Nomad SDK validam os possíveis caminhos de execução das funções dos adaptadores (os três caminhos de execução referidos anteriormente: sucesso, erro no pedido à nuvem e falha no pedido HTTP), também como as funções de módulos auxiliares.

Para isto, criou-se para cada adaptador e serviço, um módulo *mock* de teste [70], que serviu para reproduzir o comportamento desejado, sem ter de recorrer a chamadas e utilização reais dos serviços dos fornecedores de nuvem.

A estas funções *mock* é passado um argumento que indica qual dos 3 caminhos de execução deverá ser reproduzido. O *parsing* dos resultados também é testado. Para isto foram colecionados alguns exemplares de respostas dos serviços, que foram utilizadas nos testes, por modo a simular o retorno de respostas reais. Para a construção destas baterias de testes foi utilizado o projeto ExUnit[16], uma *framework* para testes unitários, semelhante ao JUnit[17].

Foi necessário recorrer ao *mock testing*, uma vez que efetuar chamadas reais às nuvens não seria viável, pois o desempenho da bateria de testes iria ser prejudicado. Para além do desempenho, uma bateria de testes que consumisse as nuvens iria originar custos monetários elevados.

Apesar da solução apenas testar o comportamento dos adaptadores, as bibliotecas Elixir para interação com as APIs das nuvens também contam com as suas próprias baterias de testes, por forma a aumentar a confiança em todo o fluxo de execução. Ao confiar nos testes destas bibliotecas, os testes necessários ao Nomad SDK foram simplificados, visto

[15] http://nerves-project.org/
[16] http://elixir-lang.org/docs/stable/ex_unit/ExUnit.html
[17] http://junit.org/

que para este apenas foi necessário testar que as chamadas às funções das bibliotecas estavam a ser realizadas corretamente, e os respetivos retornos estavam a ser devidamente transformados. No total foram construídos 548 casos de teste, tal como descriminado no **Apêndice D**.

4.6 Sumário

Neste capítulo foi apresentada a solução proposta, para o problema do *vendor lock-in* em aplicações Elixir. A arquitetura dinâmica e o código injetado através da meta-programação, foram técnicas valiosas para o Nomad SDK. Todos os objetivos definidos na **Secção 4.1 Contexto e Objetivos** (N1, N2, N3 e N4) foram atingidos:

1. **N1**: API única para serviços comuns (armazenamento de ficheiros, bases de dados SQL e máquinas virtuais) nas duas nuvens: a Amazon Web Services e Google Cloud Platform.

   (a) Armazenamento de ficheiros: foi desenvolvida uma especificação, na forma de um Behaviour, para este serviço. Esta foi implementada nos adaptadores da AWS e GCP, para utilização do Amazon S3 e Google Cloud Storage, respetivamente.

   (b) Bases de Dados SQL: foi desenvolvida uma especificação, na forma de um Behaviour, para este serviço. Esta foi implementada nos adaptadores da AWS e GCP, para utilização do Amazon RDS e Google Cloud SQL, respetivamente.

   (c) Máquinas Virtuais: foi desenvolvida uma especificação, na forma de um Behaviour, para este serviço. Esta foi implementada nos adaptadores da AWS e GCP, para utilização do Amazon EC2 e Google Compute Engine, respetivamente.

2. **N2**: Ferramentas para gestão de recursos e serviços das nuvens.

   (a) Armazenamento de ficheiros: foram criadas ferramentas de linha de comandos para listagem, criação, eliminação e migração de armazenamentos de ficheiros, nas duas nuvens suportadas.

   (b) Bases de dados SQL: foram criadas ferramentas de linha de comandos para listagem, criação, eliminação e reinício de bases de dados SQL, nas duas nuvens suportadas.

   (c) Máquinas virtuais: foram criadas ferramentas de linha de comandos para listagem, criação, eliminação e reinício de máquinas virtuais, nas duas nuvens suportadas.

3. **N3**: Alteração de fornecedor na Nuvem não envolve alterações ao código-fonte.

(a) API única: foi desenvolvida uma API única, cujo código-fonte relevante à nuvem escolhida é injetado, via meta-programação e em tempo de compilação, nas interfaces dos módulos da API única.

Uma aplicação, que utilize o Nomad SDK, pode ser portada para outra nuvem, alterando um ficheiro de configurações. Ao alterar este ficheiro, a aplicação será recompilada, e terá o novo código-fonte do fornecedor escolhido injetado nas interfaces.

4. **N4**: Exclusão do código-morto (código pertencente ao fornecedor não escolhido).

(a) Ao possuir uma arquitetura que utiliza o estilo arquitetural *plug-ins* (os adaptadores), que são injetados nas interfaces da API única, e com um elevado grau de desacoplamento, o Nomad SDK permite que o código-fonte, referente aos fornecedores de nuvem não escolhidos pelo programador, possam ser removidos sem afetar o sistema de qualquer forma.

O Nomad SDK permite a portabilidade de aplicações Elixir e Erlang entre nuvens, através da sua API única, e remove o código-morto via meta-programação.

Capítulo 5

O S21sec Portal

Neste capítulo será apresentada a *intranet* S21sec Portal, uma aplicação Web construída em Elixir com a Phoenix Framework[1], a *framework* Web *standard* para a linguagem. Na **Secção 5.1** serão apresentados os objetivos que se pretendiam atingir com o desenvolvimento desta aplicação. Na **Secção 5.2** são descritos os passos de levantamento e análise dos requisitos do S21sec Portal, e na **Secção 5.3** apresenta-se o desenho desta aplicação.

5.1 Objetivos

O objetivo com o S21sec Portal era o desenvolvimento de uma aplicação Web que centralizasse a comunicação dos colaboradores da organização, e automatizasse alguns dos serviços e procedimentos da organização que ainda necessitavam de esforço manual, particularmente a avaliação de vulnerabilidades em sistemas informáticos através da ferramenta Nessus [100]. Mais especificamente, o S21sec Portal deveria:

1. Permitir a gestão de recursos, tais como utilizadores, clientes e projetos.
2. Oferecer um sistema de *ticketing*, para facilitar a comunicação entre utilizadores.
3. Integrar com o sistema de auditoria de vulnerabilidades, de Miguel Cabral [23].

5.2 Análise

Segundo Sharlin *et al.* [92], uma aplicação necessita de uma análise cuidada de requisitos, funcionais e não-funcionais, de modo a adequar-se ao propósito da sua criação. Por esta razão, o levantamento de requisitos teve início com o agendamento de três reuniões com os *stakeholders* da S21sec (um utilizador final dos serviços automatizados e dois gestores, cujos departamentos iriam utilizar a aplicação), a fim de identificar os requisitos funcionais e não-funcionais a atingir.

[1] https://www.phoenixframework.org

Nestas reuniões, foram realizados questionários, para promover o levantamento de requisitos e riscos, assim como a exploração de várias ideias para a aplicação. O questionário realizado foi baseado no trabalho de Brugger [20], sendo que as questões foram separadas em dois grupos: as questões de resposta fechada — que permitem obter respostas mais objetivas — e de seguida as questões de resposta aberta — que oferecem o potencial de troca de ideias, respostas inesperadas e oferecem maior à vontade ao entrevistado [85]. A partir destas entrevistas, derivaram-se os requisitos funcionais e não-funcionais para o S21sec Portal.

5.2.1 Requisitos Funcionais

São reconhecidos como requisitos funcionais todas as funcionalidades que uma aplicação informática deve oferecer aos seus utilizadores.

Modelo de Casos de Uso

Seguindo a metodologia do Unified Process explicada por Craig Larman [65], os requisitos funcionais foram capturados como um conjunto de casos de uso realizados por diversos atores, tal como mostrado na **Figura 5.1**.

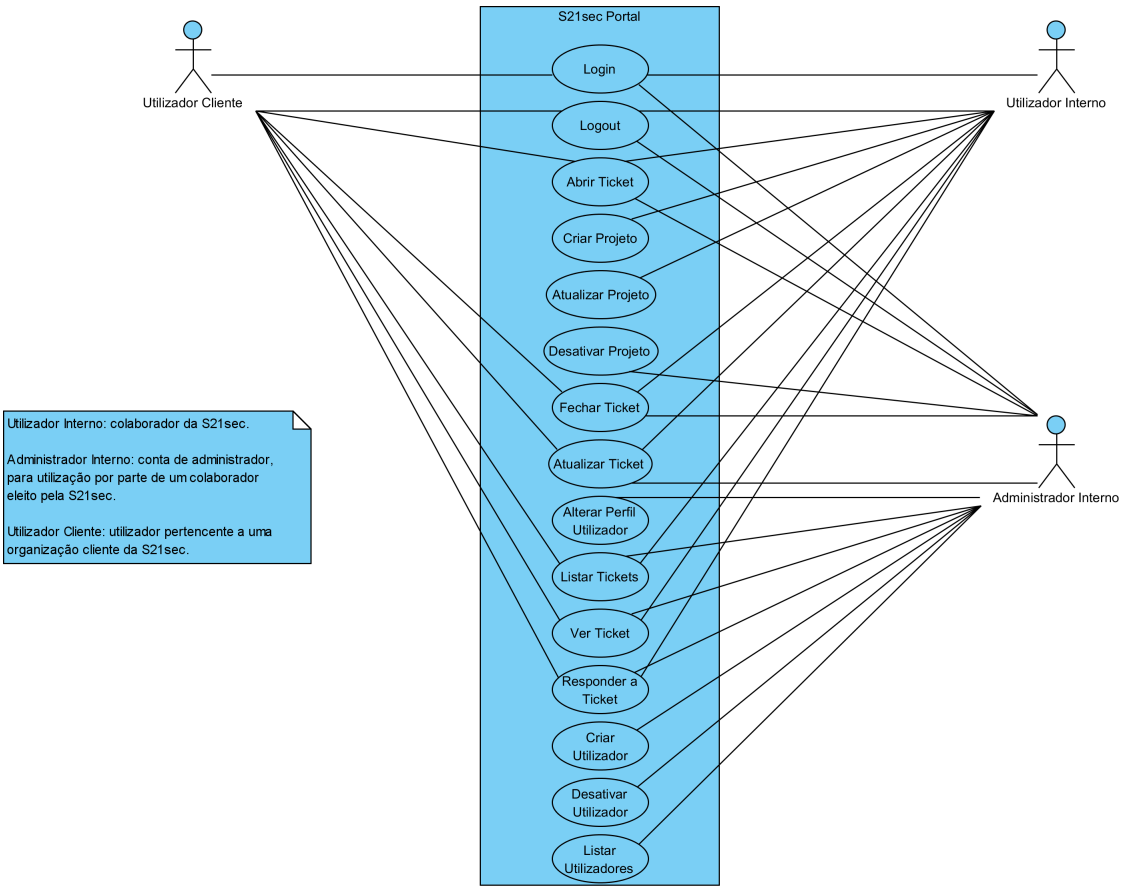
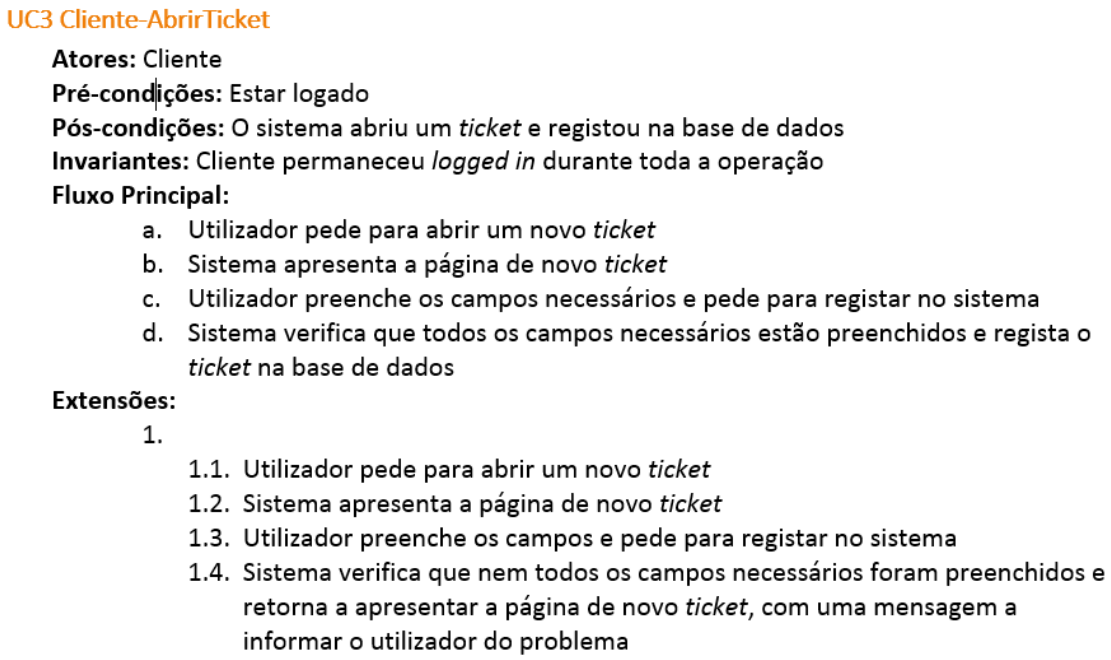


Figura 5.1: Diagrama de casos de uso da aplicação S21sec Portal.

Este diagrama ilustra as funcionalidades, na forma de casos de uso representados por elipses, que a aplicação deve suportar. Cada caso de uso captura um ou mais cenários de interação, i.e., sequências de trocas de informação, entre entidades externas à aplicação — utilizadores humanos ou outros sistemas, designados por atores — e esta. Na **Figura 5.2** é ilustrado o detalhe de um destes casos de uso relativo à função de abertura de um *ticket*.



UC3 Cliente-AbrirTicket

Atores: Cliente
Pré-condições: Estar logado
Pós-condições: O sistema abriu um *ticket* e registou na base de dados
Invariantes: Cliente permaneceu *logged in* durante toda a operação
Fluxo Principal:

a. Utilizador pede para abrir um novo *ticket*
b. Sistema apresenta a página de novo *ticket*
c. Utilizador preenche os campos necessários e pede para registar no sistema
d. Sistema verifica que todos os campos necessários estão preenchidos e regista o *ticket* na base de dados

Extensões:

1.
   1.1. Utilizador pede para abrir um novo *ticket*
   1.2. Sistema apresenta a página de novo *ticket*
   1.3. Utilizador preenche os campos e pede para registar no sistema
   1.4. Sistema verifica que nem todos os campos necessários foram preenchidos e retorna a apresentar a página de novo *ticket*, com uma mensagem a informar o utilizador do problema

Figura 5.2: Caso de uso de Abrir Ticket

A vantagem de capturar os requisitos funcionais como casos de uso reside no facto de se conseguir modelar exatamente a forma como os atores utilizam a aplicação. Este método revela-se muito mais eficaz que o simples levantamento das funcionalidades a suportar pela aplicação — o conhecido *feature model* — o qual não captura como é que a aplicação é usada pelos seus utilizadores, resultando muitas vezes em frustração e atrasos para revisão dos requisitos [65].

5.2.2 Requisitos Não-Funcionais

Os requisitos não-funcionais podem ser interpretados, como as propriedades e qualidades que devem ser mantidas, ou atingidas, no cumprimento dos requisitos funcionais [28, 16]. A qualificação de um requisito não-funcional é a indicação de qual o desempenho da função, ou quão resistente a erros esta deve ser. Estas qualidades são satisfeitas, por meio de estruturas introduzidas na arquitetura do *software*, que possuem comportamentos e interações apropriadas, para atingir uma dada qualidade [12].

Contudo, estas qualidades devem poder ser testadas e caracterizadas de uma forma

objetiva. Isto pode ser solucionado através de *cenários de atributos de qualidade* [12]. Um cenário é geralmente composto da seguinte forma:

- **Estímulo:** um evento que está a chegar ao sistema, como por exemplo um comando do utilizador.
- **Fonte do Estímulo:** a origem do estímulo pode alterar a forma como o sistema deve lidar com o mesmo.
- **Resposta:** constitui a forma como o sistema deve lidar com o estímulo. Isto consiste de responsabilidades em tempo de execução, ou a implementação que os programadores devem desenvolver.
- **Medida de Resposta:** determinação do nível de satisfação da resposta. Por exemplo, uma medida de latência da resposta.
- **Ambiente:** as circunstâncias em que o evento ocorreu.
- **Artefacto:** a porção do sistema a qual o requisito se aplica, por exemplo, a interface do utilizador ou base de dados.

Os atributos de qualidade devem ser atingidos de alguma forma, para tal, o arquiteto de *software* deverá aplicar técnicas que o auxiliem nessa mesma tarefa. Bass *et al.* [12] denominam estas técnicas por *táticas arquiteturais*. Uma tática é uma decisão de desenho, que influencia o cumprimento de uma resposta de um atributo de qualidade, ou seja, são decisões que afetam diretamente a resposta de um sistema a um dado estímulo. A combinação destes atributos, em conjunto com os requisitos funcionais e objetivos de negócio, originam os *architectural drivers* (i.e., são os requisitos que esculpem a arquitetura que um dado *software* deverá possuir).

Na **Tabela 5.1** e na **Tabela 5.2** são apresentados dois cenários, para a obtenção de atributos de qualidade, nomeadamente ao nível da segurança e desempenho. Na primeira tabela, é estabelecida a resposta ao estímulo proveniente de um utilizador, que deseja aceder a um *ticket* relativo a um projeto ao qual não pertence (i.e., não está na equipa do mesmo). Para resposta, foi decidido a aplicação de um sistema de permissões, que impede utilizadores não pertencentes à equipa de um dado projeto, de visualizar os *tickets* deste. Na segunda tabela, é concetualizado o cenário de pedidos de duzentos clientes em simultâneo. Apesar de não se esperar um elevado número de utilizadores para a aplicação, esta deve estar preparada para servir todos os colaboradores da organização em simultâneo, no mínimo. Como resposta, foi estabelecida a utilização de um servidor Web, que possibilite ligações concorrentes a todos os utilizadores, num dado momento. O servidor Web, que acabou por ser escolhido, foi o Cowboy[2], um servidor construído em

[2]http://ninenines.eu/

Erlang, que utiliza as suas capacidades de programação por processos, de modo a atribuir um processo a cada utilizador, de modo a servi-lo. Todos os cenários concebidos podem ser consultados no **Apêndice C**.

Segurança 4	Um membro júnior da equipa tentou ver um ticket de um projeto onde não está inserido
Estímulo	Utilizador a enviar pedido
Fonte do Estímulo	Browser cliente
Ambiente	Aplicação S21sec Portal em execução
Artefacto	S21sec Portal
Resposta	Necessidade de privilégios para concretizar ação
Medida de Resposta	Implementação de um sistema de utilizadores com privilégios distintos, que apenas oferece privilégios para ações críticas aos utilizadores relevantes; Apenas os utilizadores inseridos num dado projeto podem ver tickets relativos ao mesmo

Tabela 5.1: Cenário de Segurança #4

Desempenho 2	200 clientes realizaram *login* e estão a utilizar a aplicação, enviando pedidos
Estímulo	Pedidos de utilizadores
Fonte do Estímulo	Browser cliente
Ambiente	Aplicação S21sec Portal em execução
Artefacto	S21sec Portal
Resposta	Estabelecimento de conexões concorrentes a todos os clientes
Medida de Resposta	Utilização de um servidor Web de alto desempenho e eficiência, capaz de estabelecer ligações concorrentes com pelo menos 500 clientes e de processar um elevado volume de pedidos sem perda dos mesmos

Tabela 5.2: Cenário de Desempenho #2

5.3 Desenho

Nesta secção será apresentada a arquitetura do S21sec Portal. Esta será apresentada, por meio de vistas arquiteturais, estilos e padrões arquiteturais. As vistas arquiteturais foram baseadas em Garlan *et al.* [42]. As três categorias de vistas apresentadas partilham a utilidade comum de educarem sobre o sistema, e de promoverem a comunicação a cerca do mesmo:

1. **Vista de Módulos:** esta vista foi escolhida, pois é uma vista referente à aplicação em tempo de implementação. Esta vista permite observar os vários elementos programáticos do sistema, como estes se relacionam e estão organizados.

2. **Vista de Componentes & Conectores:** aqui são demonstrados elementos presentes em tempo de execução. Esta vista oferece possibilidades de educação a cerca do fluxo de execução do sistema.

3. **Vista de Alocação:** documenta o ambiente do *software* desenvolvido em fase de produção. Esta vista permite visualizar os vários elementos de *software* e os recursos que lhes correspondem. Estes recursos poderão ser elementos de *hardware*, por exemplo.

5.3.1 Vista de Módulos

A Vista de Módulos é uma das três categorias de vistas em [42], as outras sendo a Vista de Componentes & Conetores e a Vista de Alocação. Como referido, a Vista de Módulos documenta as principais unidades de implementação, ou seja, esta deverá no mínimo, apresentar a forma como o código-fonte de um sistema é decomposto em unidades geríveis, quais as suposições que estas unidades deverão ter em relação aos serviços das outras unidades, e como é que estas unidades interagem entre si. Foram criadas três vistas de módulos relativamente a outros tantos aspetos da estrutura da aplicação:

1. **Decomposição:** mostra como a aplicação está estruturada numa hierarquia de módulos compostos por sub-módulos, permitindo identificar rapidamente as responsabilidades de cada parte da aplicação. Esta vista é apresentada na **Figura B.1** do **Apêndice B**.

2. **Utilização:** tem o propósito de documentar dependências funcionais entre módulos. Este estilo foi escolhido, uma vez que complementa o estilo de decomposição, pois permite perceber como é que os módulos apresentados neste se relacionam. Esta vista está apresentada na **Figura B.2** no **Apêndice B**.

3. **Modelo de Dados:** apresenta as relações entre as entidades que formam a camada de dados do sistema, neste caso, uma base de dados SQL. A razão para a escolha deste estilo deveu-se à natureza do S21sec Portal, uma vez que é um sistema de informação que faz um elevado uso de uma base de dados SQL. Esta vista está apresentada na **Figura B.3** no **Apêndice B**.

5.3.2 Vista de Componentes & Conectores

A Vista de Componentes & Conectores mostra os elementos presentes em tempo de execução, tais como processos, objetos, clientes, servidores e armazéns de dados [42]. Estes elementos são os *componentes*, enquanto as ligações de interação entre estes são os *conectores*. Os componentes possuem interfaces de interação entre si, designadas por portos. Os conectores possuem perfis (*roles*), que representam a forma como o conector é utilizado para interligar componentes (p. ex., por HTTP).

Na **Figura 5.3** é apresentada a Vista de Componentes & Conectores do S21sec Portal, que ilustra o estilo *cliente-servidor*. Neste estilo os componentes interagem com outros componentes, ao invocarem serviços (funções ou operações de outros componentes,

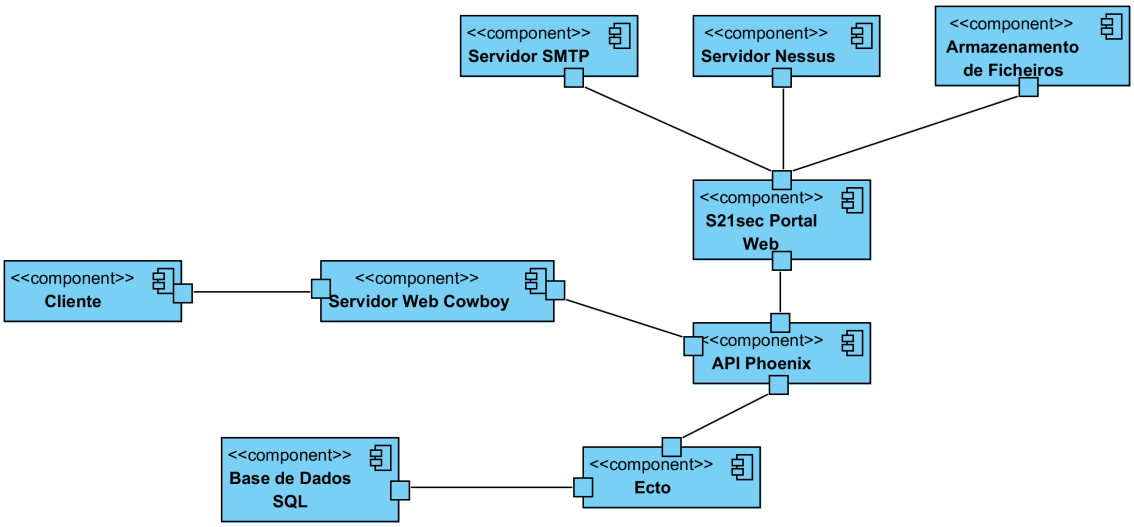


Figura 5.3: Vista de Componentes & Conectores S21sec Portal - Estilo Cliente-Servidor

através de um conector de chamada e retorno) oferecidos por estes. Os componentes que pedem o serviço são os *clientes*, enquanto os componentes que oferecem o serviço são os *servidores*. Um componente pode ser um cliente e servidor em simultâneo.

No S21sec Portal, os pedidos são enviados por clientes (*browsers* Web), que são recebidos pelo componente Servidor Web Cowboy. Este servidor irá pedir à API da Phoenix Framework, que direcione o pedido para o S21sec Portal Web, onde este será tratado, seguindo a lógica do negócio definida. A componente S21sec Portal Web também age como um cliente, pois envia pedidos às componentes de serviços externos à aplicação, nomeadamente o Servidor SMTP, Servidor Nessus e Armazenamento de Ficheiros. Contudo, o S21sec Portal Web também envia pedidos à API da Phoenix Framework, para escritas e leituras na Base de Dados SQL, que as realiza por meio da componente Ecto.

5.3.3 Vista de Alocação

A Vista de Alocação mapeia os elementos de software da aplicação nos vários recursos disponibilizados para o seu desenvolvimento e exploração. Estes recursos poderão ser, por exemplo, os membros das equipas de programadores responsáveis por criar ou manter a aplicação, e elementos de hardware onde esta será instalada e executada. No caso do S21sec Portal, foi criada a vista de alocação do tipo instalação exibida na **Figura 5.4**.

O S21sec Portal executa num ambiente de uma máquina Linux, que age como servidor aplicacional. Este servidor irá interagir com outros recursos, tais como um servidor SMTP (Simple Mail Transfer Protocol) ou bases de dados MySQL. Os clientes realizarão pedidos via *browser* Web, sendo que estes serão recebidos por um balanceador de carga Nginx[3], que os irá direcionar para o servidor aplicacional. Este balanceador de carga

[3]https://www.nginx.com/

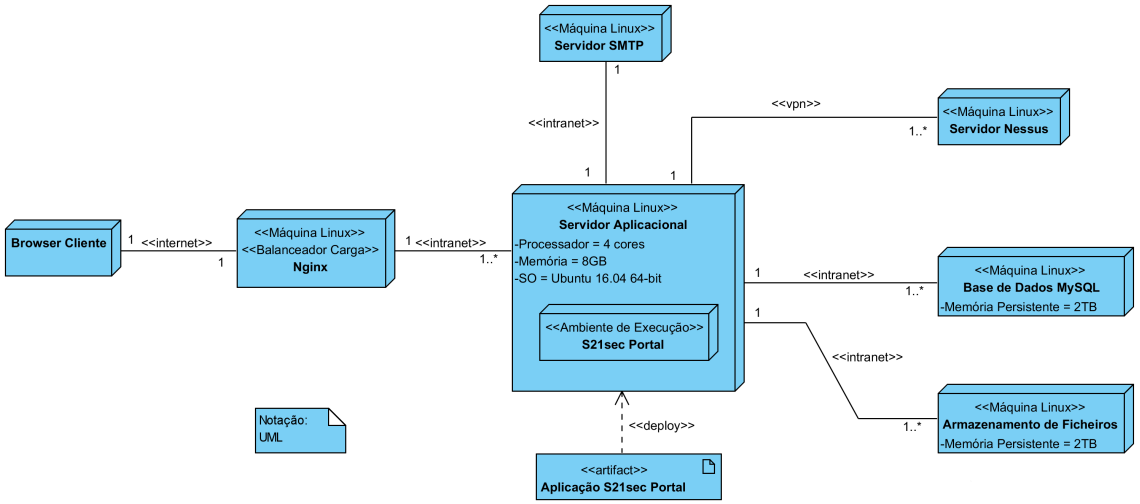


Figura 5.4: Vista de Alocação do S21sec Portal - Instalação

tem o objetivo de possibilitar a utilização de mais que um servidor aplicacional (tal como indicado pela multiplicidade na **Figura 5.4**), distribuindo a carga por estes.

5.3.4 Estilos e Padrões Arquiteturais

Os seguintes estilos e padrões arquiteturais foram aplicados no S21sec Portal.

Estilos Arquiteturais

1. ***Database-centric*:** este estilo arquitetural denota a extensa utilização de um repositório de dados persistente, por várias componentes do sistema [51]. A base de dados SQL do S21sec Portal atua como o repositório de dados persistente, sendo acedida pelos vários controladores da aplicação.

2. ***Tiered*:** a separação da camada de apresentação, lógica do negócio e dados persistentes, representa um estilo arquitetural chamado *tiered*. No caso do S21sec Portal, este estilo é *3-tiered*, sendo realizado da seguinte forma:

   (a) *Browser* cliente, que atua como a camada de apresentação.

   (b) Os controladores que recebem e tratam dos pedidos, atuando como a camada da lógica do negócio

   (c) A base de dados SQL, que atua como a camada de dados persistentes.

3. **REST**: tal como o Nomad SDK, o S21sec Portal realiza chamadas REST. Estas são feitas à API da ferramenta Nessus, e também à API do Twitter[4].

[4]https://twitter.com/

Padrões Arquiteturais

Os estilos arquiteturais representam primitivas a usar no desenho da arquitetura duma aplicação, um padrão arquitetural representa uma solução de desenho arquitetural, frequentemente associada a um domínio aplicacional específico, construída com base na utilização de um ou mais estilos arquiteturais. Sendo assim, um padrão arquitetural confere à aplicação um conjunto de qualidades derivadas da combinação das qualidades associadas aos estilos utilizados. No S21sec Portal, o padrão arquitetural escolhido foi o Model-View-Controller (MVC) [21]:

1. **Model-View-Controller:** este padrão visa solucionar a apresentação de informação ao utilizador, tornando modular as componentes que a constroem, por forma a que seja possível modificá-las individualmente. Para isto são utilizados controladores (*controllers*), modelos (*models*) e vistas (*views*).

   O padrão MVC utilizado no S21sec Portal diverge da definição tradicional do mesmo. No S21sec Portal os pedidos são recebidos pelo controlador, que interage com a base de dados (ao invés desta interação ser feita pelo modelo, tal como é feito no padrão tradicional), sendo que o modelo irá validar os dados (maioritariamente verificando a obediência a regras de restrição de integridade), antes destes serem enviados para a base de dados. Após a validação dos dados, o controlador interage com a base de dados, sendo que ao receber a resposta desta, os dados são passados à vista relevante, por modo a ser construída a página HTML a apresentar.

5.4 Sumário

Neste capítulo foi apresentada a aplicação S21sec Portal e os seus objetivos. O processo de análise foi descrito, reportando a metodologia e tipos de requisitos levantados. Por fim, o desenho arquitetural da aplicação foi apresentado, através das várias vistas arquiteturais desenvolvidas e dos estilos e padrões arquiteturais aplicados.

Capítulo 6

O Nomad SDK No Mundo Real

Um dos objetivos definidos no capítulo inicial foi a validação do Nomad SDK em contexto real. Para isto, a solução foi integrada numa versão independente da aplicação S21sec Portal, passando esta usufruir de portabilidade entre as nuvens Amazon Web Services (AWS) e Google Cloud Platform (GCP). A portabilidade residiu nos seguintes serviços:

1. **Armazenamento de ficheiros:** o S21sec Portal oferece uma funcionalidade de anexar ficheiros a *tickets*. Para isto, o anexo deve ser carregado para um armazenamento de ficheiros, para posteriormente poder ser acedido através da página do *ticket* correspondente.

2. **Base de Dados SQL:** a aplicação, sendo um sistema de informação, utiliza uma base de dados SQL extensivamente.

6.1 Integração do Nomad SDK no S21sec Portal

O primeiro passo para a integração do SDK no S21sec Portal consistiu na criação dos recursos necessários: o armazenamento de ficheiros e a base de dados SQL. Ambos os recursos foram criados em cada uma das nuvens suportadas. Para isto, utilizaram-se as ferramentas de linha de comandos desenvolvidas para o Nomad SDK. As duas ferramentas utilizadas foram:

1. **`Nomad.Storage.Create`:** este comando cria um novo armazenamento de ficheiros na nuvem escolhida no ficheiro de configuração. Para isto o comando recebe como argumentos o nome desejado para o armazenamento de ficheiros e a região onde se quer que este seja alojado.

2. **`Nomad.DatabaseInstance.Create`:** este comando cria uma nova instância de base de dados SQL na nuvem escolhida no ficheiro de configuração. Para isto o comando executa um *script*, pedindo ao utilizador vários parâmetros, tais como o nome da base de dados, o motor SQL, ou a região onde a instância deve ser criada.

Feito isto, o Nomad SDK foi instalado no S21sec Portal, como uma dependência deste, através do gestor de pacotes do Elixir. Após a instalação, as configurações necessárias a cada fornecedor de nuvem, a AWS e a GCP, foram introduzidas, simultaneamente, no ficheiro de configuração do S21sec Portal. Estas configurações foram as seguintes:

1. **Fornecedor de nuvem:** o fornecedor de nuvem a utilizar teve de ser especificado através de um par chave-valor, tal como descrito no **Capítulo 4**.

2. **Nome do armazenamento de ficheiros:** foi utilizado o mesmo nome para cada uma das nuvens: `nomad-portal-demo`.

3. **Informação da base de dados SQL:** foram configurados o endereço IP e as credenciais de acesso à bases de dados SQL criadas em cada nuvem.

4. **Credenciais de acesso:** foram configuradas as credenciais de acesso às APIs de ambos os fornecedores de nuvem. Estas credenciais são necessárias para se poderem realizar as chamadas às APIs das duas nuvens suportadas.

Finalmente, foi alterado o código-fonte do S21sec Portal, para que realizasse as chamadas necessárias à API de armazenamento de ficheiros do Nomad SDK.

6.1.1 Integração do Armazenamento de Ficheiros

O S21sec Portal já contava com a funcionalidade de carregar ficheiros via SSH (Secure Shell) [95]. Ao portar esta funcionalidade para a Nuvem, foi necessário reescrever a aplicação para usar a funcionalidade correspondente na API do Nomad SDK. Na **Listagem 6.1** e **Listagem 6.2**, é apresentado o código aplicacional desta funcionalidade antes e depois da integração com o Nomad SDK.

```
defp upload({attachment, ticket}) do
  filename = Map.get(attachment, :filename)
  |> String.split
  |> strip_whitespace("")

  filepath = Map.get(attachment, :path)
  attach_url =
     "/home/#{@username}/attachments/#{ticket.id}/#{filename}"

  {_, _}  = System.cmd "ssh",
             [@username <> "@" <> @server,
              "mkdir
                 /home/#{@username}/attachments/#{ticket.id}"]

  {_, res} = System.cmd "scp",
             ["-i",
              @ssh_key_path,
              filepath,
              @username <> "@" <> @server <> ":" <> attach_url]

  case res do
    0 -> {:ok, attach_url}
    _ -> :error
  end
end
```

Listagem 6.1: Função de carregamento de ficheiros do S21sec Portal, antes da integração do Nomad SDK.

```
defp upload({attachment, ticket}) do
  filename = Map.get(attachment, :filename)
  filepath = Map.get(attachment, :path)
  bucket = Application.get_env(:nomad, :bucket)
  sto_path = "#{ticket.id}/#{filename}"
  res    = Nomad.Storage.put_item(bucket, filepath, sto_path)

  case res do
    :ok -> {:ok, sto_path}
    _  -> :error
  end
end
```

Listagem 6.2: Função de carregamento de ficheiros do S21sec Portal, após a integração do Nomad SDK. De notar a chamada à API do Nomad SDK `Nomad.Storage.put_item(bucket, filepath, sto_path)`.

6.1.2 Integração da Base de Dados SQL

Passar a utilizar o serviço SQL na Nuvem apenas requereu inserir no ficheiro de configuração o endereço IP da base de dados e as respetivas credenciais de acesso (nome de utilizador e palavra-passe) para cada uma das nuvens.

Feita a configuração, o Ecto desligou o S21sec Portal da base de dados local e ligou-se à base de dados da nuvem escolhida. As abstrações oferecidas pela DSL do Ecto fizeram com que não fosse necessário qualquer tipo de alteração de código, visto que a biblioteca permite realizar uma ligação a qualquer base de dados, desde que esta seja de acesso público, e que as credenciais de acesso sejam passadas ao estabelecer a conexão.

6.1.3 Portação da aplicação

Após a integração do Nomad SDK, a portação do S21sec Portal da Amazon Web Services para a Google Cloud Platform foi trivial:

1. Primeiro, foi validada a correção da aplicação, ao utilizar a nuvem AWS para as suas funcionalidades.

2. De seguida, para portar a aplicação, bastou unicamente alterar o par chave-valor do ficheiro de configuração, para que agora indicasse a GCP, como sendo a nuvem desejada.

3. Feita a alteração à configuração, a aplicação foi recompilada e passou a utilizar o outro fornecedor de nuvem, automaticamente e sem quaisquer alterações de código-fonte.

4. Um sistema de informação, onde os dados persistentes têm um grande relevo na sua utilidade, não usufrui totalmente de portabilidade, se os seus dados não forem também portáveis. A nossa solução oferece uma ferramenta de linha de comandos, para migração de ficheiros entre serviços de armazenamento de ficheiros. Esta ferramenta foi utilizada, após a recompilação para a nuvem GCP, de modo a transferir os ficheiros do serviço Amazon S3, para o serviço Google Cloud Storage. Desta forma, o Nomad SDK permitiu alcançar portabilidade ao nível da lógica do negócio e dos dados do S21sec Portal.

   Da mesma forma, os dados da base de dados SQL também foram portados. Contudo, isto foi feito através da própria linha de comandos das bases de dados, pois estas permitem exportar os dados para um ficheiro adequado, para posteriormente ser importado noutra base de dados.

5. Finalmente, a correção da aplicação foi novamente validada, estando agora esta a consumir a GCP. O código-morto, relativo à AWS, pôde ser removido, sem afetar de qualquer forma o sistema.

6.2 Validação de Resultados

Os resultados obtidos da integração do Nomad SDK no S21sec Portal foram positivos, sendo que os dois objetivos principais do *sdk* foram atingidos:

1. A aplicação foi portada entre as duas nuvens, sem requerer alterações ao código-fonte. A integração da solução no S21sec Portal foi feita através da API única, que oferece as mesmas funcionalidades nas duas nuvens suportadas.

2. A arquitetura ao estilo de camadas e *plug-ins*, aliada à injeção de código-fonte via meta-programação, fez com que fosse possível recompilar a aplicação para tomar partido do novo fornecedor de nuvem, ao mesmo tempo que foi removido o código-morto referente ao fornecedor original.

A aplicação ficou agnóstica ao fornecedor de nuvem, sendo transparente qual a nuvem que está a ser utilizada. Através da análise do código-fonte da aplicação, ou através da utilização da mesma, não é possível depreender qual a nuvem em uso (tal como pode ser visto na chamada à API do Nomad na **Listagem 6.2**).

Os serviços de armazenamento de ficheiros funcionaram tal como esperado, sendo possível carregar (através de formulários HTML) e descarregar (através de HTTP) ficheiros, não havendo qualquer perceção destas transferências estarem a ocorrer a partir de outro país ou região.

Em contraste, nas bases de dados SQL, ocorreu uma perceção de latência. O carregamento de páginas, onde existia um maior volume de dados a serem devolvidos para visualização, foi mais lento, em comparação com a base de dados instalada na máquina local ao servidor aplicacional do portal. Suspeitou-se que esta latência pudesse ter originado de problemas de desempenho, ao invés de ter sido causada pela distância geográfica ou conexão à Internet, uma vez que tinham sido utilizadas bases de dados de mais baixa gama em cada uma das nuvens.

6.2.1 Base de dados local vs. na Nuvem

De modo a aprofundar esta questão, foram realizados testes à latência de uma interrogação utilizada na aplicação, na base de dados local (L), em comparação com bases de dados de baixa (B) e alta (A) gama na GCP. A interrogação utilizada foi a recolha de todos as entradas na tabela de utilizadores (`SELECT * FROM users`), contudo, é de salientar que esta foi concretizada através da DSL da camada de abstração para bases de dados Ecto, o que terá acrescentado computações extraordinárias à interrogação, quando comparado com uma interrogação direta. A tabela contava com doze entradas de utilizadores, e foi utilizada a base de dados MySQL na versão 5.6.

Na **Tabela 6.1** são apresentadas as especificações das máquinas utilizadas. As máquinas da nuvem estavam alojadas em St. Ghislain, na Bélgica. Na **Tabela 6.2**, podemos ver

a comparação da latência (em microsegundos) entre os três casos descritos. A diferença na latência de L com as máquinas A e B é notória. A L obteve uma média de latência de acesso de 1,21E-03 μs, enquanto as bases de dados A e B obtiveram 5,28E-02 μs e 5,34E-02 μs, respetivamente. A diferença das latências entre as bases de dados A e B (6,00E-04 μs) é desprezável, podendo ter sido originada por outros fatores para além da latência ou desempenho das bases de dados (p. ex., desempenho da máquina local ao realizar a chamada à base de dados). Isto é indicativo do desempenho não ser a causa da latência, pois a máquina da base de dados A possui especificações que lhe garantem um desempenho ordens de magnitude superior a B, o que não é refletido na média de latências obtidas.

Máquina	Processador	Memória
Local (L)	2 núcleos	4GB
Baixa-gama na Nuvem (B)	1 núcleo	128MB
Alta-gama na Nuvem (A)	16 núcleos	60GB

Tabela 6.1: Máquinas utilizadas para teste das latências de base de dados locais vs. na Google Cloud Platform

Adicionalmente, para comparar o desempenho de A e B, foram capturados os tempos totais de migração do modelo de dados da aplicação, em cada uma das bases de dados. Estes tempos podem ser vistos na **Tabela 6.3**, sendo que a migração da base de dados B foi concretizada em 1:47 minutos, aproximadamente, enquanto que a migração na A foi concretizada em cerca de 21 segundos. Neste teste, podemos ainda reconfirmar que a suposição original estava errada, ou seja, a latência é originada da distância geográfica ou da conexão à Internet, visto que a migração da base de dados L ocorreu em cerca de $\frac{1}{2}$ do tempo da base de dados A, apesar desta última possuir especificações vastamente superiores à de L.

Concluímos, que a latência foi causada pela conexão à Internet ou distância geográfica, pois a base de dados L, ao estar num patamar intermédio, no que toca às especificações da máquina, obteve menor latência. Caso a latência tivesse sido causada unicamente pelo desempenho das bases de dados, então a base de dados A deveria ser aquela com menor latência. Contudo, o desempenho poderá ser um fator do aumento de latência nos acessos na nuvem, quando estes forem feitos através de interrogações ou funções com maior grau de complexidade. Como pôde ser visto na **Tabela 6.3**, a migração na base de dados A foi concretizado em um $\frac{1}{5}$ do tempo que foi obtido na migração da base de dados B.

Acessos Locais (μs)	Accessos GCP Baixa-Gama (μs)	Accessos GCP Alta-Gama (μs)
2,40E-03	8,91E-02	9,38E-02
8,23E-04	6,07E-02	4,65E-02
1,08E-03	4,92E-02	4,95E-02
7,74E-04	4,80E-02	4,89E-02
1,27E-03	4,61E-02	4,73E-02
7,81E-04	6,03E-02	5,13E-02
1,16E-03	5,05E-02	4,78E-02
1,52E-03	4,57E-02	4,84E-02
1,10E-03	4,20E-02	4,82E-02
1,21E-03	4,25E-02	4,66E-02
Média	Média	Média
1,21E-03	5,34E-02	5,28E-02

Tabela 6.2: Comparação do tempo de execução de uma interrogação a uma base de dados local vs. uma base de dados de baixa gama na GCP vs. uma base de dados de alta gama na GCP.

Máquina	Tempo Migração
Local	11.061 seg.
Baixa gama na Nuvem	1:47.38 min.
Alta gama na Nuvem	21.110 seg.

Tabela 6.3: Comparação dos tempos de migração de base de dados na Nuvem.

6.2.2 Ferramentas de Linha de Comandos

As ferramentas de linha de comandos também se provaram úteis. A criação dos recursos foi facilitada, sendo possível utilizar os comandos sem ser necessário ter conhecimento do funcionamento das nuvens. Estas ferramentas serão úteis aos programadores que possuam pouco conhecimento dos fornecedores de nuvem suportados, e permite-lhes lançar recursos rapidamente sem terem de estudar as APIs ou as ferramentas proprietárias destes fornecedores. Para além disto, as ferramentas de linha de comandos do Nomad SDK também permitem destruir ou reiniciar recursos, através de comandos que ficam disponíveis após a instalação da aplicação onde o *sdk* foi integrado, tornando expedito portar a aplicação entre fornecedores, e prontamente cessar os custos exploração na mesma no fornecedor original.

6.3 Sumário

Neste capítulo foi relatada a integração do Nomad SDK numa aplicação do mundo real, o S21sec Portal. A integração mostrou-se trivial, sendo apenas necessário introduzir as

configurações dos fornecedores de nuvem a utilizar, e respetivos recursos. A portabilidade de aplicações Elixir entre nuvens, sem requerer alterações de código-fonte e sem incluir código-morto, foi alcançada.

Capítulo 7

Trabalho Relacionado

Vários esforços têm sido feitos para resolver o problema do *vendor lock-in*. No trabalho conduzido por Silva *et al.* [94] em 2013, 78 potenciais soluções para o problema foram avaliadas. Neste conjunto, a maioria das soluções podem-se categorizar como **plataformas** ($\approx 13\%$), **APIs** ($\approx 12\%$), **arquiteturas** ($\approx 10\%$) e ***frameworks*** ($\approx 9\%$). O Nomad SDK é considerado como fazendo parte da categoria de APIs. Neste capítulo são apresentadas algumas soluções presentes na literatura, que visam resolver a problemática do *vendor lock-in*. Estas serão comparadas com o Nomad SDK.

7.1 JClouds

O JClouds[1] é uma API de desenvolvimento de aplicações entre nuvens, para a linguagem Java. São oferecidas interfaces para os serviços de máquinas virtuais, armazenamento de ficheiros e balanceamento de carga. Em comparação, o Nomad SDK oferece interfaces para os serviços de máquinas virtuais, armazenamento de ficheiros e bases de dados SQL. No que toca aos fornecedores de nuvem suportados, o JClouds conta com mais de dez fornecedores, enquanto o Nomad SDK apenas suporta dois.

Contudo, o JClouds exibe desvantagens face ao Nomad SDK, no que toca às alterações de código-fonte e na remoção de código-morto:

1. **Alterações de código:** o JClouds compromete a portabilidade das aplicações por requerer parametrizações e funcionalidades específicas ao fornecedor.

   (a) Parametrizações: o JClouds requer que sejam passados parâmetros específicos ao fornecedor. Por exemplo, a construção de um novo objeto para armazenamento de ficheiros, requer que seja passado um parâmetro `aws-s3`, por modo a que seja criado um objeto compatível com o serviço Amazon S3.

   A existência de pontos no código-fonte, que requeiram parametrizações específicas ao fornecedor, fragilizam a portabilidade da aplicação, uma vez que

[1] https://jclouds.apache.org/

estas parametrizações irão requerer alterações, para que estejam adaptadas ao contexto de outros fornecedores.

(b) Funcionalidades: o JClouds permite a utilização de funcionalidades específicas ao fornecedor. Estas funcionalidades quebram a portabilidade, pois uma funcionalidade específica da API de um fornecedor não existirá na API de outro. O Nomad SDK não oferece a possibilidade de utilização de funcionalidades específicas, de modo a maximizar o grau de portabilidade e evitar a alteração de código-fonte.

2. **Remoção de código-morto:** certos casos de uso da API do JClouds requerem que a aplicação verifique o tipo de objeto retornado para fazer uso correto deste. Por exemplo, na **Listagem 7.1**, a aplicação verifica o tipo de meta-dados retornado pela API do JClouds para decidir qual o código a executar subsequentemente. Este último dependerá do fornecedor de nuvem escolhido, pelo que o código correspondente aos outros fornecedores nunca será executado pela aplicação, tornando-se morto. Adicionalmente, este exemplo ilustra uma limitação à portabilidade da aplicação, pois esta terá de ser alterada para suportar novos fornecedores ou alterações à API do JClouds no que diz respeito aos tipos de objetos de suporte a cada fornecedor.

```
// Use Provider API
Object object = null;
Object object = null;
if (apiMetadata instanceof S3ApiMetadata) {
   S3Client api = context.unwrapApi(S3Client.class);
   object = api.headObject(containerName, blobName);
} else if (apiMetadata instanceof SwiftApiMetadata) {
  ...
} else if (apiMetadata instanceof AzureBlobApiMetadata) {
  ...
} else if (apiMetadata instanceof AtmosApiMetadata) {
  ...
} else if (apiMetadata instanceof
   GoogleCloudStorageApiMetadata) {
  GoogleCloudStorageApi api =
     context.unwrapApi(GoogleCloudStorageApi.class);
  object = api.getObjectApi().getObject(containerName,
     blobName);
}
```

Listagem 7.1: Exemplo oficial do JClouds, onde os possíveis retornos desta API têm de ser antecipados pelo programador.

Por fim, outra diferença entre o JClouds e o Nomad SDK, encontra-se na transversalidade dos serviços suportados. O Nomad SDK suporta fornecedores que oferecem os

três serviços das interfaces do *sdk* (armazenamento de ficheiros, bases de dados SQL e máquinas virtuais). Para garantir a máxima portabilidade, os serviços devem estar disponíveis em todos os fornecedores suportados. Contudo, o JClouds também suporta fornecedores que não oferecem todos os serviços das suas interfaces. Por exemplo, o JClouds suporta o fornecedor de nuvem Digital Ocean[2], cujo único serviço que oferece é o de máquinas virtuais.

7.2 ErlCloud

O ErlCloud [3] é uma coleção de clientes Erlang, para consumo das APIs do fornecedor AWS. É um dos projetos, relacionados com a Nuvem, mais populares deste ecossistema. Contudo, ao apenas oferecer suporte para a AWS, este projeto não pode ser considerado uma solução de portabilidade de aplicações Elixir e Erlang entre nuvens.

7.3 mOSAIC

O mOSAIC [4] foi um esforço, a nível Europeu, iniciado por Dana Petcu *et al.* [81]. Esta solução visa oferecer portabilidade e interoperabilidade a aplicações, criando ambientes multi-nuvem (i.e., ambientes que utilizam várias nuvens simultaneamente), que conseguem interoperar entre si e entre os seus serviços [49], ou seja, são criados ambientes que são capazes de utilizar várias nuvens, de igual forma. Neste projeto foi criada uma camada de abstração, entre a aplicação desenvolvida e a nuvem escolhida. Para isto é utilizado um componente intermediário, que faz a ligação entre os recursos pedidos e os correspondentes nas várias nuvens, abstraindo as diferenças e oferecendo a utilidade do recurso via uma única API.

São exibidas duas desvantagens do mOSAIC:

1. **Complexidade e dificuldade de utilização:** a arquitetura do mOSAIC assenta sobre um modelo de programação por eventos [81]. Os autores referem este modelo como sendo uma desvantagem, pois é considerado um paradigma de programação mais complexo e que dificulta o desenvolvimento. Em contraste, o Nomad SDK visa oferecer simplicidade, por meio da sua API única, através da qual os programadores podem realizar chamadas às APIs dos fornecedores, sem criar dependências com estas.

2. **Execução num sistema operativo próprio:** os autores indicam que as aplicações desenvolvidas com o mOSAIC necessitam de executar numa distribuição Linux de

[2] https://www.digitalocean.com/
[3] https://github.com/erlcloud/erlcloud
[4] http://www.mosaic-cloud.eu/

32-bit própria para o projeto[5], o que limita severamente a escolha do ambiente de desenvolvimento e produção. Em comparação, o Nomad SDK pode ser executado, sem requerer alterações, nos sistemas operativos Linux, Windows e MacOS.

O projeto mOSAIC já se deu por encerrado. Este teve a sua última apresentação numa conferência em 2013, e a sua última atualização, no seu repositório de código público, ocorreu em 2014. A implementação da API do mOSAIC em Java foi a única realizada.

7.4 Fog

O Fog [6] é um SDK para a linguagem Ruby, com o objetivo de solucionar o problema do *vendor lock-in* para esta linguagem, e ao mesmo tempo auxiliar programadores não familiares com a Nuvem no desenvolvimento de aplicações para esta. Semelhantemente à nossa solução, o Fog utiliza um conjunto de adaptadores, por forma a estabelecer a sua camada de abstração.

Tal como o projeto JClouds, o Fog sofre dos mesmos problemas, relacionados as parametrizações específicas e de código-morto:

1. **Parametrizações:** devem ser utilizadas parametrizações específicas, tal como apresentado na **Listagem 7.2**. Estas parametrizações baixam o grau de portabilidade das aplicações, pois é necessário realizar alterações ao código-fonte, quando estas têm de ser portadas para outras nuvens. Para além das parametrizações, o Fog também inclui retornos específicos.

2. **Código-morto:** o Fog suporta mais de dez fornecedores de nuvem. Apesar destes estarem separados, maioritariamente em módulos específicos, ao estudar o repositório de código do Fog, foi possível constatar que existem outras componentes de código específico aos fornecedores espalhados pelos outros módulos. Para além disto, o módulo principal do Fog carrega sempre todos os módulos específicos aos fornecedores de nuvem. Desta forma, o Fog conta com código-morto, relativo às nuvens não utilizadas.

[5] http://developers.mosaic-cloud.eu/confluence/display/MOSAIC/mOS
[6] http://fog.io/

```
# create a compute connection
compute = Fog::Compute.new({
  :provider => 'AWS',
  :aws_access_key_id => ACCESS_KEY_ID,
  :aws_secret_access_key => SECRET_ACCESS_KEY
})
# compute operations go here

server = Compute[:aws].servers.create(:image_id =>
   'ami-5ee70037')
# <Fog::AWS::EC2::Server [...]>

# create a storage connection
storage = Fog::Storage.new({
  :provider => 'AWS',
  :aws_access_key_id => ACCESS_KEY_ID,
  :aws_secret_access_key => SECRET_ACCESS_KEY
})
# storage operations go here
```

Listagem 7.2: Exemplo oficial do Fog, onde é necessário passar parâmetros específicos e onde os resultados retornados são específicos ao fornecedor de nuvem.

7.5 Service Delivery Cloud Platform

O Service Delivery Cloud Platform (SDCP) [13], iniciado em 2011, é uma solução para o problema do *vendor lock-in*, desenvolvida por Bastião *et al.* da Universidade de Aveiro. Esta solução também se baseia na conceção de uma API única disponível através de um SDK, que permite o consumo das funcionalidades transversais às várias nuvens suportadas. Inicialmente, este projeto era apenas utilizável na linguagem Java, tendo sido construído em cima do JClouds. Contudo, foi desenvolvido posteriormente um *middleware*, na forma de um serviço Web, para que a solução pudesse ser utilizada por qualquer linguagem.

O SDCP oferece uma API única que suporta as nuvens Amazon Web Services, Google Cloud Platform e Rackspace[7]. Ao nível dos serviços suportados, o SDCP conta com os serviços de armazenamento de ficheiros, bases de dados NoSQL e de notificações. Estes serviços foram suportados, pois o objetivo do SDCP foi auxiliar a construção de um sistema de armazenamento de imagens clínicas na Nuvem.

Este trabalho, apesar de ter várias parecenças com o Nomad SDK, também exibe algumas desvantagens em comparação com este:

1. **Funcionalidades oferecidas:** o SDCP oferece poucas funcionalidades para cada

[7]https://www.rackspace.com/

serviço, quando comparado com o Nomad SDK. Por exemplo, o serviço de armazenamento de ficheiros apenas oferece funções de leitura, escrita e alteração da lista de controlo de acesso. Em contraste, o Nomad SDK oferece mais nove funções, para além das três implementadas no SDCP, tais como a remoção de um ficheiro ou a criação de um armazém.

2. **Parametrizações específicas:** os objetos do SDCP requerem parametrizações específicas. Por exemplo, o objeto `CloudSocket`, para a envio ou receção das `streams` dos objetos de leitura e escrita do serviço de armazenamento de ficheiros, possui atributos globais que representam o fornecedor de nuvem e as credenciais de acesso ao mesmo. Desta forma, a mudança de fornecedor irá obrigar a alterações no código-fonte.

   O ficheiro de configurações, tal como aplicado no Nomad SDK, oferece as vantagens de retirar estas parametrizações específicas e de possibilitar a utilização do mesmo ficheiro de configurações, em várias aplicações.

3. **Inclusão de código-morto:** o facto do SDCP estar assente no JClouds faz com o que o código-morto deste continue incluído.

Capítulo 8

Conclusão

O problema do *vendor lock-in* impossibilita as aplicações de serem portadas entre os vários fornecedores de nuvem. Esta problemática é reconhecida na literatura e na indústria, havendo esforços de ambas as partes em solucioná-la, pois isso permitiria aos programadores e organizações beneficiar dos vários fornecedores de nuvem, conforme a melhor oferta entre estes no momento.

As soluções conhecidas em desenvolvimento incorrem em alterações de código-fonte nas aplicações, quando estas têm de ser portadas, ou incluem código-morto referente aos fornecedores não desejados.

Neste trabalho foi proposta uma solução para o problema do *vendor lock-in*, denominada Nomad SDK, que ao oferecer uma API única, permite aos programadores desenvolverem aplicações portáveis entre duas nuvens populares (Amazon Web Services e Google Cloud Platform), sem que tenham de realizar alterações de código-fonte no momento da portação. Esta solução foi construída com uma arquitetura por camadas e baseada em adaptadores, que ao fazerem uso da meta-programação para injetar o código relevante ao fornecedor de nuvem escolhido, permitem a remoção do código-morto nos adaptadores não incluídos relativos aos outros fornecedores de nuvem.

8.1 Contribuições Realizadas

A execução deste trabalho resultou nas contribuições apresentadas de seguida.

8.1.1 Nomad SDK

Foi desenvolvido o *kit* Open-Source Nomad SDK[1], para desenvolvimento de aplicações Elixir portáveis entre nuvens. O Nomad SDK, até ao momento da escrita deste relatório, era o único projeto que visava solucionar o problema do *vendor lock-in* para aplicações Elixir e Erlang.

[1] `https://www.github.com/sashaafm/nomad`

Esta solução oferece uma API única, para utilização de três serviços distintos (armazenamento de ficheiros, bases de dados SQL e máquinas virtuais), em duas das nuvens mais populares da atualidade, a Amazon Web Services, e a Google Cloud Platform). Esta API única oferece as seguintes vantagens:

- Programação através duma mesma interface independentemente da nuvem escolhida;
- Evita a necessidade de alterar o código-fonte das aplicações aquando da portação para outra nuvem;
- Não exige que os programadores tenham conhecimento das APIs específicas de cada nuvem;
- Fácil expandir o suporte a novas nuvens através da criação de novos adaptadores implementando esta API (bem documentada);
- A nuvem a utilizar não tem de ser escolhida durante a fase de desenho da aplicação.

Adicionalmente, a remoção de código-morto também introduz um conjunto de vantagens:

- Reduz a quantidade de código considerada pelo programador nas tarefas de manutenção corretiva e evolutiva da aplicação que faz uso da API;
- Evita-se raciocinar sobre código que nunca é utilizado.
- Reduz a dimensão das aplicações que fazem uso da API, facilitando a execução destas em sistemas restritos, particularmente sistemas embebidos, sejam estes clientes da Nuvem ou utilizados por esta para execução das aplicações nela instaladas.

Para além disto, o *kit* inclui um conjunto de ferramentas de linha de comandos, que permitem gerir os recursos das nuvens (armazenamentos de ficheiros, bases de dados SQL e máquinas virtuais). Estas têm o propósito de possibilitar a manipulação destes recursos fora de uma aplicação, ou seja, sem que sejam realizadas chamadas pela API do Nomad SDK no código-fonte de uma aplicação desenvolvida com o mesmo. Desta forma, os programadores podem manipular estes recursos, nas nuvens suportadas, sem ser exigido que estudem as APIs ou ferramentas proprietárias dos fornecedores.

O Nomad SDK tem recebido uma reação positiva nas redes sociais. Alguns programadores Elixir têm mostrado interesse no projeto, sendo que já existiram afirmações de utilização do *kit* nas suas aplicações. De seguida encontram-se duas citações de utilizadores que manifestaram o seu interesse no fórum oficial do Elixir[2] (no **Apêndice E** encontram-se capturas dos comentários retiradas diretamente do fórum).

[2]http://elixirforum.com/

«This looks so amazing. Once my application becomes deploy-ready I'll definitely try it out. What a wonderful idea! — Qqwy»

«Awesome! Great tool you're making @sashaafm! Looking forward to use it on my elixir project. — cmelgarejo»

8.1.2 S21sec Portal

A *intranet* S21sec Portal ofereceu à S21sec um sistema de informação para a centralização dos seus dados, vias de comunicação e automatização das avaliações de vulnerabilidades pela ferramenta Nessus. Com isto, a organização beneficiará de uma aplicação, que lhe trará maior valor. Este sistema de informação foi desenvolvida e os objetivoso principais do projeto atingidos. A aplicação encontra-se neste momento numa fase de fecho de ciclo, onde está a ser concretizada a documentação.

8.1.3 GCloudex

Foi desenvolvido a biblitoeca *open-source* Elixir GCloudex[3] para consumo dos serviços da Google Cloud Platform por aplicações Elixir. É oferecido total suporte à utilização das APIs dos serviços Google Cloud Storage, Google Cloud SQL e Google Compute Engine. Este projeto tem vindo a receber algumas contribuições no seu repositório, por parte de programadores que o têm integrado nas suas aplicações.

8.1.4 ExAWS

Foram feitas duas contribuições para o projeto ExAWS, uma biblioteca Elixir para consumo das APIs da Amazon Web Services. As contribuições foram a implementação das APIs do serviço Amazon RDS e do serviço Amazon EC2.

8.2 Trabalho Futuro

Existem várias direções para o trabalho futuro possíveis:

- Em relação ao Nomad SDK:
  - Continuar o desenvolvimento de uma ferramenta de linha de comandos, que permite o empacotamento, transferência e instalação automática de aplicações Elixir numa máquina virtual remota (p. ex., na nuvem).
  - Aquando da adição de um novo adaptador, para suporte a um novo fornecedor de nuvem, é necessário alterar a expressão condicional de cada módulo, que recebe o código injetado dos adaptadores, por modo a avaliar também o

[3] `https://www.github.com/sashaafm/gcloudex`

novo fornecedor inserido (descrito na **Secção 4.3.3**). Neste momento já se encontra em desenvolvimento (num estado quase final) uma solução para este problema, que fará com que possa ser adicionado suporte a novos fornecedores no Nomad SDK, sem que nenhum outro módulo existente tenha de ser alterado.

  - Suportar a nuvem Microsoft Azure, também uma das mais populares.
  - Suportar outros serviços comuns entre nuvens, tais como bases de dados NoSQL ou Content Delivery Network.
  - Realização de um estudo, por modo a quantificar as vantagens do Nomad SDK em relação a outros trabalhos relacionados:
    1. Quantificar as alterações de código-fonte que são evitadas ao utilizar o nosso *sdk*.
    2. Quantificar quanto código-morto é removido das aplicações que utilizem o Nomad SDK, em comparação com a quantidade de código-morto que permanece nas aplicações que utilizem as outras soluções relacionadas.
  - Desenvolvimento de um serviço Web, consumido via API REST, que possibilite o desenvolvimento para a Nuvem, utilizando qualquer linguagem de programação. Este serviço serviria como um *middleware*, com as quais as aplicações poderiam interagir, através da utilização de um estilo arquitetural universal.

    Contudo, apesar deste serviço Web assemelhar-se ao serviço desenvolvido no Service Delivery Cloud Platform (discutido na **Secção 7.5**), este novo serviço possuiria uma diferença fulcral, pois não transpareceria qual a nuvem que estava a ser utilizada. Ou seja, o programador apenas requisitaria um determinado recurso, e este seria fornecido pelo serviço, que iria requisitar o recurso em qualquer uma das nuvens suportadas, conforme fosse mais conveniente.

- Continuar a desenvolver o S21sec Portal, focando na criação de uma API para integração com outras aplicações internas, e de um módulo de avaliação de métricas dos serviços prestados aos clientes.

- Suportar no GCloudex o serviço de bases de dados NoSQL Cloud Bigtable[4], e o serviço de balanceamento de carga Cloud Load Balancing[5].

- Continuar a contribuir para a manutenção do projeto ExAWS

[4]https://cloud.google.com/bigtable/
[5]https://cloud.google.com/load-balancing/

Apêndice A

	Amazon EC2		Google Compute Engine	
Tipo	Capacidade Computacional Mínima	Capacidade Computacional Máxima	Capacidade Computacional Mínima	Capacidade Computacional Máxima
Propósito Geral	1vCPU/3.75GB	8vCPU/30GB	1vCPU/3.75GB	32vCPU/120GB
Computação Otimizada	2vCPU/3.75GB	36vCPU/60GB	2vCPU/1.80GB	32vCPU/28.8GB
Memória Otimizada	2vCPU/15.25GB	32vCPU/244GB	2vCPU/13GB	32vCPU/208GB
Placa Gráfica Otimizada	8vCPU/15GB	N/A	N/A	N/A

Tabela A.1: Tabela comparativa das várias possibilidades de máquinas virtuais nos serviços Amazon EC2 e Google Compute Engine. Adaptado de [1].

Apêndice B

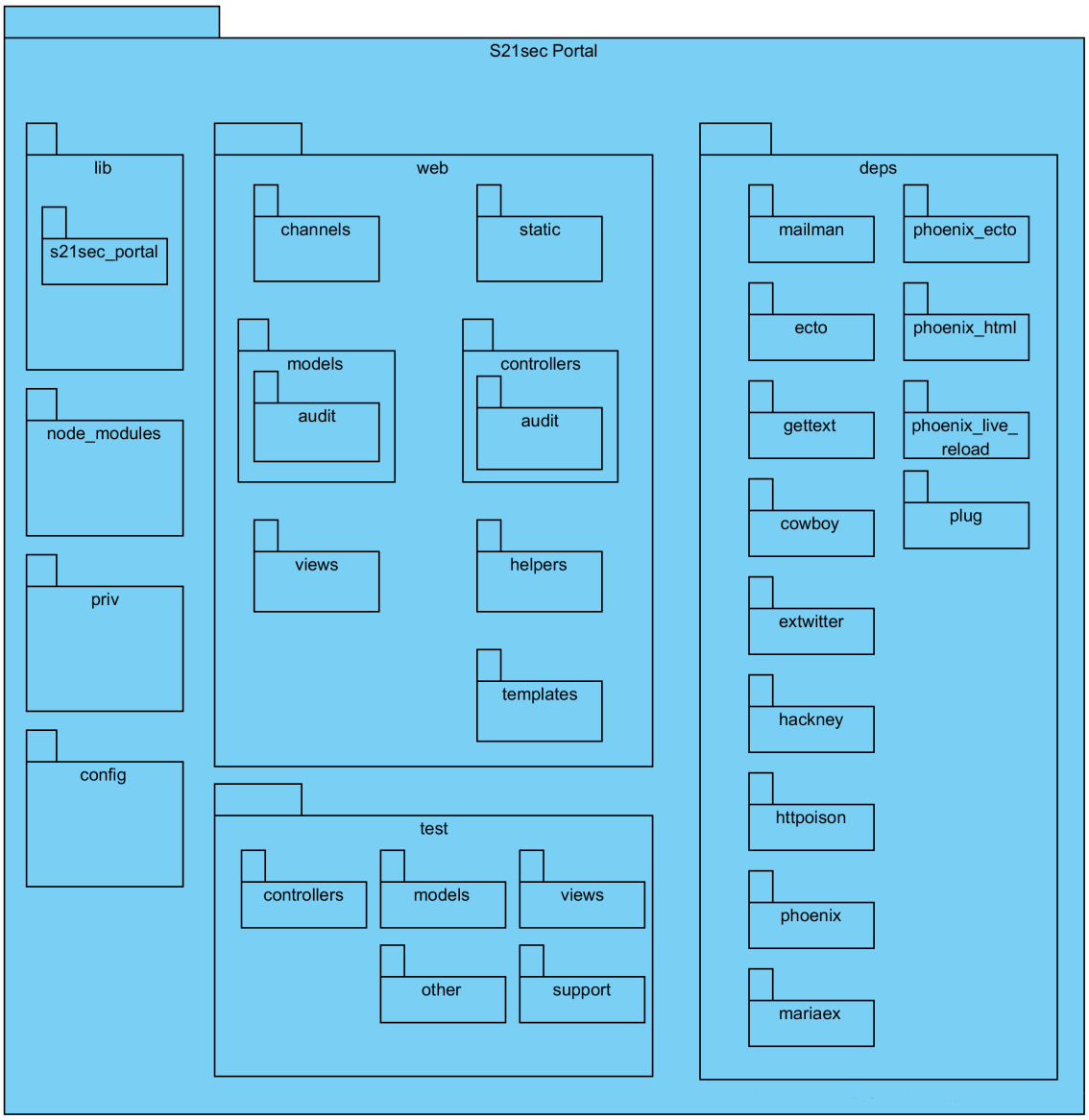


Figura B.1: Vista de Módulos S21sec Portal - Decomposição

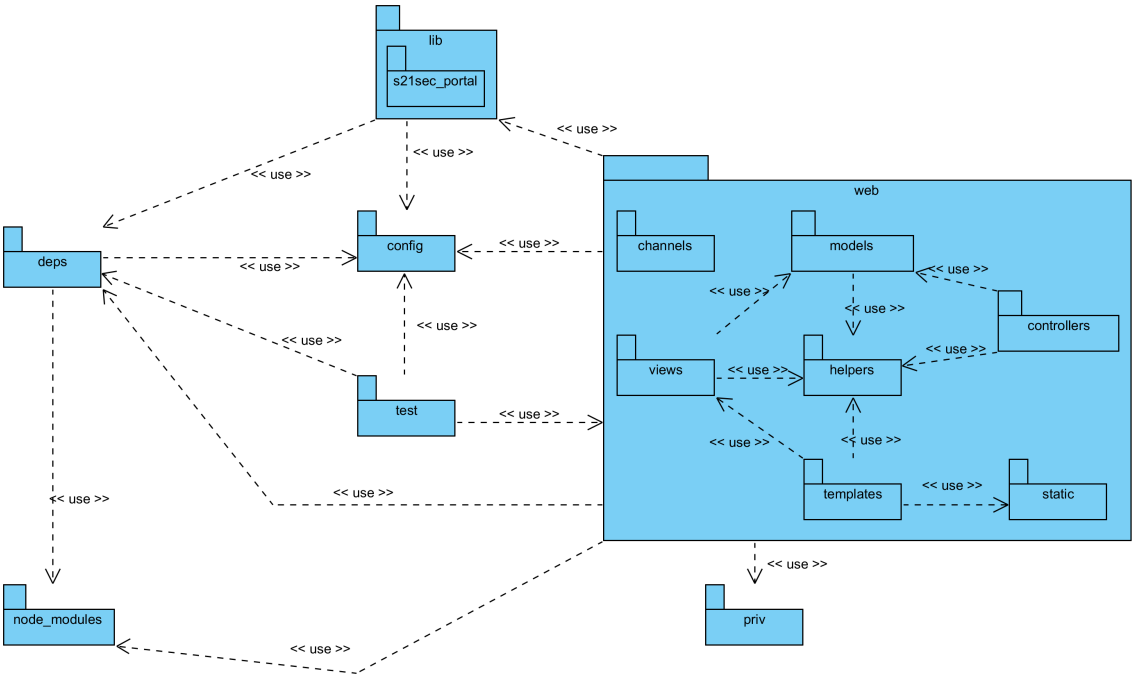


Figura B.2: Vista de Módulos S21sec Portal - Utilização

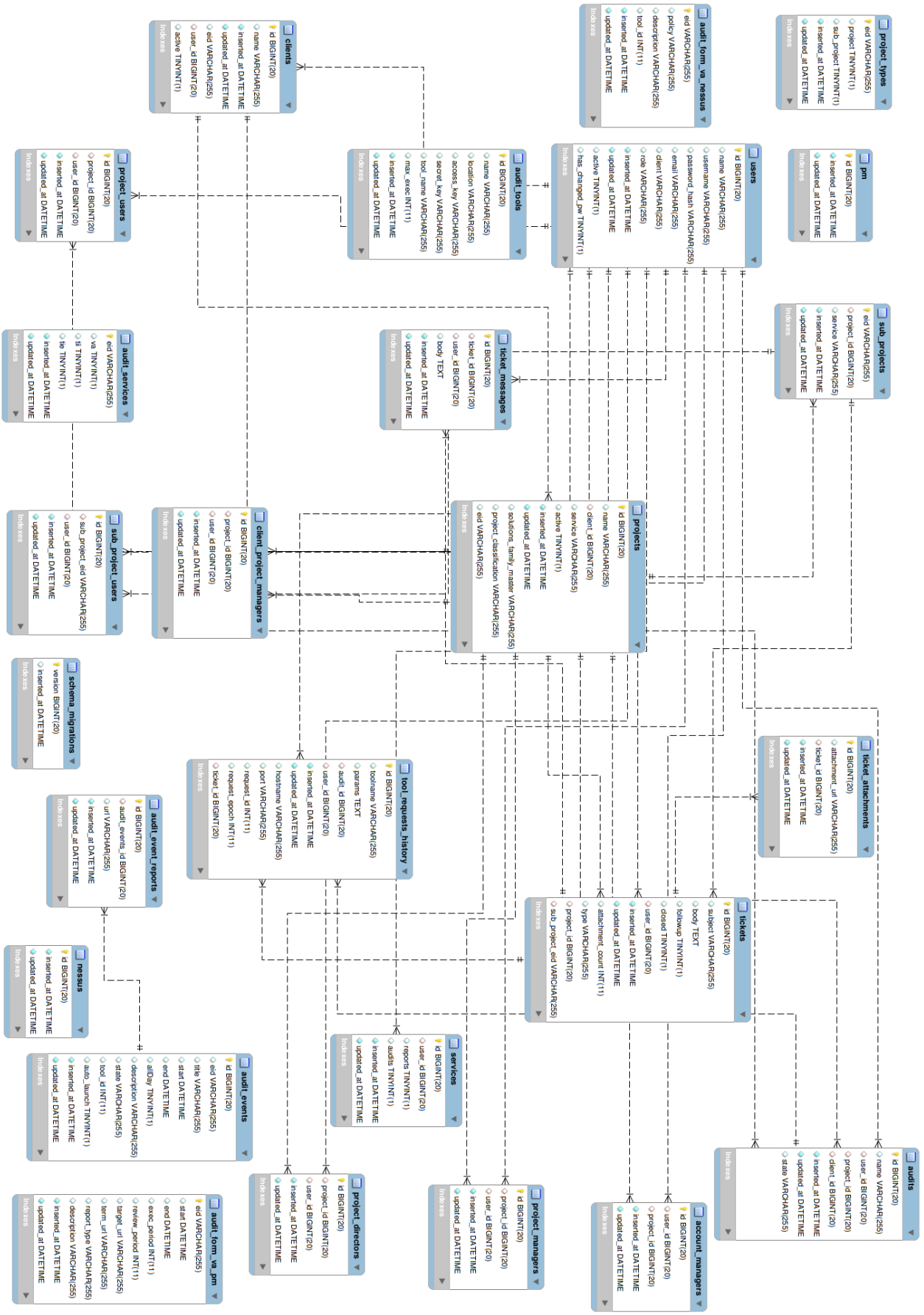


Figura B.3: Vista de Módulos S21sec Portal - Estilo de Modelo de Dados

Apêndice C

Segurança 1	O cliente solicitou o descarregamento de um ficheiro com dados confidenciais
Estímulo	Pedido de descarregamento
Fonte do Estímulo	Browser cliente
Ambiente	Aplicação S21sec Portal em execução
Artefacto	S21sec Portal
Resposta	Envio do ficheiro através de uma ligação segura
Medida de Resposta	Utilização do protocolo HTTPS em toda a aplicação.

Tabela C.1: Cenário de Segurança #1

Segurança 2	A aplicação sofreu um ataque de Cross-site Request Forgery
Estímulo	Ataque malicioso
Fonte do Estímulo	Browser cliente
Ambiente	Aplicação S21sec Portal em execução
Artefacto	S21sec Portal
Resposta	Impossibilidade de realização de Cross-site Request Forgery
Medida de Resposta	Utilização de mecanismos para prevenção deste tipo de ataques

Tabela C.2: Cenário de Segurança #2

Segurança 3	Um desastre natural deitou abaixo a base de dados da aplicação, deixando a aplicação sem a conseguir aceder
Estímulo	Pedido à base de dados
Fonte do Estímulo	Aplicação S21sec Portal
Ambiente	Aplicação S21sec Portal em execução
Artefacto	S21sec Portal
Resposta	Garantia da disponibilidade e salvaguarda dos dados
Medida de Resposta	Estabelecimento de réplicas da base de dados noutro ponto geográfico distinto

Tabela C.3: Cenário de Segurança #3

Segurança 4	Um membro júnior da equipa tentou ver um ticket de um projeto onde não está inserido
Estímulo	Utilizador a enviar pedido
Fonte do Estímulo	Browser cliente
Ambiente	Aplicação S21sec Portal em execução
Artefacto	S21sec Portal
Resposta	Necessidade de privilégios para concretizar ação
Medida de Resposta	Implementação de um sistema de utilizadores com privilégios distintos, que apenas oferece privilégios para ações críticas aos utilizadores relevantes; Apenas os utilizadores inseridos num dado projeto podem ver tickets relativos ao mesmo

Tabela C.4: Cenário de Segurança #4

Segurança 5	Um dos componentes do sistema falhou durante a execução
Estímulo	Falha de um componente
Fonte do Estímulo	Componente do sistema
Ambiente	Aplicação S21sec Portal em execução
Artefacto	S21sec Portal e sistema total
Resposta	O sistema recupera e continua a executar
Medida de Resposta	Utilização de metodologias para sistemas tolerantes a faltas, tanto ao nível do servidor, como da aplicação

Tabela C.5: Cenário de Segurança #5

Desempenho 1	O sistema recebeu 1500 pedidos num período de 30 segundos
Estímulo	Pedidos de clientes
Fonte do Estímulo	Browser cliente
Ambiente	Aplicação S21sec Portal em execução
Artefacto	S21sec Portal
Resposta	Processamento de todos os pedidos e envio das respetivas respostas
Medida de Resposta	Utilização de um servidor de alto desempenho e eficiência, de forma a conseguir obter um {throughput} de 5000 pedidos/segundo

Tabela C.6: Cenário de Desempenho #1

Desempenho 2	200 clientes realizaram *login* e estão a utilizar a aplicação, enviando pedidos
Estímulo	Pedidos de utilizadores
Fonte do Estímulo	Browser cliente
Ambiente	Aplicação S21sec Portal em execução
Artefacto	S21sec Portal
Resposta	Estabelecimento de conexões concorrentes a todos os clientes
Medida de Resposta	Utilização de um servidor Web de alto desempenho e eficiência, capaz de estabelecer ligações concorrentes com pelo menos 500 clientes e de processar um elevado volume de pedidos sem perda dos mesmos

Tabela C.7: Cenário de Desempenho #2

Eficiência 1	A aplicação sofre um elevado pico de trabalho momentâneo
Estímulo	Pedidos de clientes
Fonte do Estímulo	Browser cliente
Ambiente	Aplicação S21sec Portal em execução
Artefacto	S21sec Portal
Resposta	Utilização de poucos recursos
Medida de Resposta	Separação das máquinas do servidor aplicacional e das bases de dados de forma a consumir recursos em máquinas distintas; Utilização de tecnologias que tenham um consumo de recursos relativamente baixos

Tabela C.8: Cenário de Eficiência #1

Apêndice D

Tabela D.1: Quantidade de testes desenvolvidos por aplicação

Aplicação	**Nº de Testes**
Nomad	185
GCloudex	211
ExAws (RDS)	15
ExAws (EC2)	137

Apêndice E

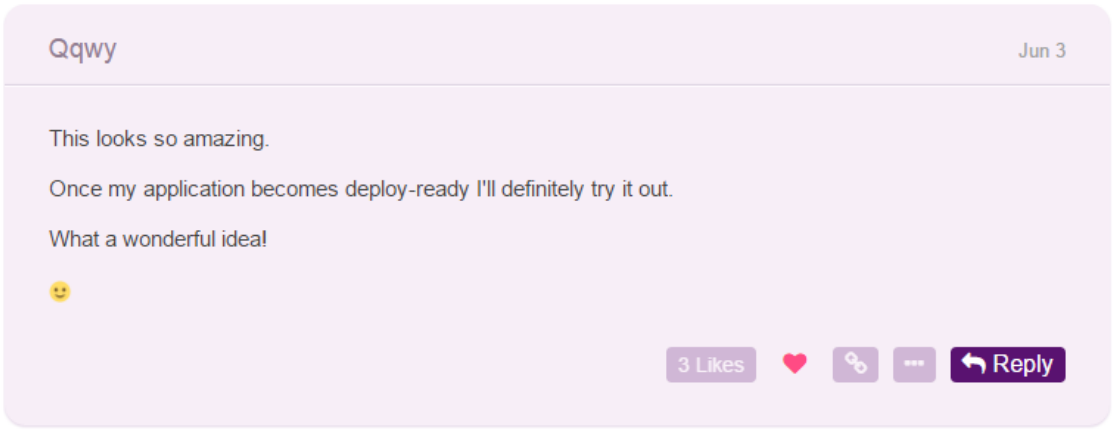


Figura E.1: Comentário a cerca do Nomad SDK, no fórum oficial do Elixir, pelo utilizador Qqwy.

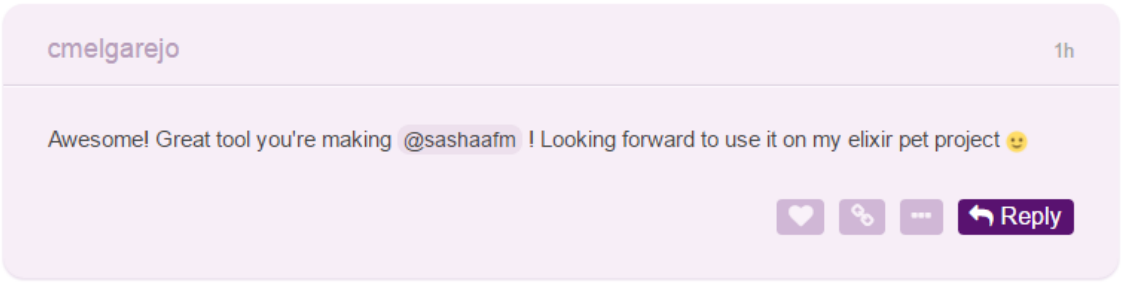


Figura E.2: Comentário a cerca do Nomad SDK, no fórum oficial do Elixir, pelo utilizador cmelgarejo.

Bibliografia

[1] Cloud Academy. Ec2 vs google compute engine: comparing the big players in iaas. http://cloudacademy.com/blog/ec2-vs-google-compute-engine/.

[2] Agarwal and Lakhina. Erlang - Programming the Parallel World. pages 1–8.

[3] Gul Abdulnabi Agha. Actors: A model of concurrent computation in distributed systems. Technical report, Massachussets Institute of Technology, Massachussets, Estados Unidos da América, 1985.

[4] Shakeel Ahmad, Bashir Ahmad, Sheikh Muhammad Saqib, and Rashid Muhammad Khattak. Trust model: Cloud's provider and cloud's user. *International Journal of Advanced Science and Technology*, 44, 2012.

[5] Amazon. Runinstances. https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_RunInstances.html.

[6] Amazon. Amazon web services. https://aws.amazon.com/, 2016. [Online; accessed 05-June-2016].

[7] Apache. Jclouds. https://jclouds.apache.org/.

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

[9] Joe Armstrong. *Making reliable distributed systems in the presence of software errors*. PhD thesis, Royal Institute of Technology of Stockholm, Department of Microelectronics and Information Technology, 12 2003.

[10] Joe Armstrong. Erlang. Technical report, ACM, 2010.

[11] Joe Armstrong. *Programming Erlang: Software for a Concurrent World*. Pragmatic Bookshelf, Dallas, Texas, 2013.

[12] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition, 2012.

[13] Luís A. Bastião Silva, Carlos Costa, and José Luís Oliveira. A common api for delivering services over multi-vendor cloud resources. *The Journal of Systems & Software*, 86(9):2309–2317, 2013.

[14] Tim Berners-Lee. Uniform resource locators (url). `https://www.w3.org/Addressing/URL/url-spec.txt`.

[15] Alexandre Beslic, Rada Bendraou, Julien Sopena, and Jean-Yves Rigolet. Towards a solution avoiding vendor lock-in to enable migration between cloud platforms. *CEUR Workshop Proceedings*, 1118:5–14, 2013.

[16] Gregor Bochmann. Non-functional requirements. `http://www.site.uottawa.ca/~bochmann/SEG3101/Notes/SEG3101-ch3-4%20-%20Non-Functional%20Requirements%20-%20Qualities.pdf`.

[17] Tim Bower. Process contents. `http://faculty.salina.k-state.edu/tim/ossg/Process/p_contents.html`.

[18] Scott Brandt. Soft real-time systems. `https://users.soe.ucsc.edu/~scott/courses/Winter00/290S/SRT.html`.

[19] Chris Brenton. Pen testing in the cloud. `https://pen-testing.sans.org/blog/2012/07/05/pen-testing-in-the-cloud`.

[20] Jeremy Brugger. Importance of interview and survey questions in systems analysis. Technical report, University of Missouri, Missouri, Estados Unidos da América, 2010.

[21] S. Burbeck. *Applications Programming in Smalltalk-80: How to Use Model-View-Controller (MVC).* Softsmarts, Incorporated, 1987.

[22] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley Publishing, 1996.

[23] Miguel Cabral. Web security auditing. Master's thesis, Faculdade de Ciências da Universidade de Lisboa, Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa, 2016.

[24] CargoSense. Exaws. `https://github.com/CargoSense/ex_aws`, 2014. [Online; accessed 04-June-2016].

[25] Richard Chow, Philippe Golle, Markus Jakobsson, Ryusuke Masuoka, and Jesus Molina. Controlling data in the cloud: Outsourcing computation without outsourcing control. *CCSW '09 Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 85–90, 2009.

[26] Alistair Cockburn. Using both incremental and iterative development. *Software Engineering Technology*, 2008.

[27] Louis Columbus. Roundup Of Cloud Computing Forecasts And Market Estimates Q3 Update, 2015. http://www.forbes.com/sites/louiscolumbus/2015/09/27/roundup-of-cloud-computing-forecasts-and-market-\penalty\@M\hskip\z@skip\discretionary{-}{}{}\penalty\@M\hskip\z@skipestimates-q3-update-2015/#163ba85d6c7a, 2015. [Online; accessed 04-June-2016].

[28] Armin Cremers and Sascha Alda. Chapter 1 requirements engineering (introduction). http://www.iai.uni-bonn.de/III/lehre/vorlesungen/SWT/RE05/slides/01_%20Requirements%20Engineering%20Intro.pdf.

[29] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[30] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Operating Systems Design and Implementation*, 2014.

[31] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: Issues and challenges. *IEEE International Conference on Advanced Information Networking and Applications*, 24, 2010.

[32] Malcolm Dowse. Erlang and first-person shooters. http://www.erlang-factory.com/upload/presentations/395/ErlangandFirst-PersonShooters.pdf.

[33] ECMA. Json. http://www.json.org/.

[34] Jorge Ejarque, Andras Micsik, and Rosa Badia. Towards automatic application migration to clouds. In IEEE Computer Society, editor, *IEEE 8th International Conference on Cloud Computing*. IEEE, 2015.

[35] Elixir. Ecto. https://github.com/elixir-lang/ecto, 2013. [Online; accessed 04-June-2016].

[36] Dick Epema and Alexandru Iosup. Cloud programming models. http://www.ds.ewi.tudelft.nl/~iosup/Courses/2012_IN4392_Lecture-5_CloudProgrammingModels.pdf.

[37] Ericsson. Erlang. https://www.erlang.org/.

[38] Ericsson. Erlang run-time system application (erts). http://erlang.org/doc/apps/erts/erts.pdf.

[39] erlcloud. erlcloud: Cloud computing apis for erlang. https://github.com/erlcloud/erlcloud, 2010. [Online; accessed 04-June-2016].

[40] Roy Fielding. Representational state transfer (rest). https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[41] Sasha Fonseca. Gcloudex. https://github.com/sashaafm/gcloudex, 2016. [Online; accessed 05-June-2016].

[42] David Garlan, Felix Bachmann, James Ivers, Judith Stafford, Len Bass, Paul Clements, and Paulo Merson. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, 2nd edition, 2010.

[43] David Garlan and Mary Shaw. An introduction to software architecture. 1994.

[44] GeekWire. Amazon web services dominates cloud survey, but microsoft azure gains traction. http://www.geekwire.com/2015/aws-remains-top-dog-in-cloud-survey-but-microsoft-\penalty\@M\hskip\z@skip\discretionary{-}{}{}\penalty\@M\hskip\z@skipazure-gains-traction/.

[45] Google. Instances: insert. https://cloud.google.com/compute/docs/reference/latest/instances/insert.

[46] Google. Chromebooks. https://www.google.com/chromebook/, 2016. [Online; accessed 25-June-2016].

[47] Google. Google cloud platform. https://cloud.google.com/, 2016. [Online; accessed 05-June-2016].

[48] John Grundy, Gerald Kaefer, Jacky Keong, and Anna Liu. Software engineering for the cloud. *IEEE Software*, 29(2):26–29, 2012.

[49] Joaquin Guillén, Javier Miranda, and Juan Manuel Carlos Canal. A service-oriented framework for developing cross cloud migratable software. *The Journal of Systems & Software*, (9):2294–2308, 2013.

[50] Mohammad Hamdaqa and Ladan Tahvildari. Prison break: A generic schema matching solution to the cloud vendor lock-in problem. *IEEE 8th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems*, pages 37–46, 2014.

[51] Denis Helic. Software architecture: architectural styles. `http://coronet.iicm.tugraz.at/sa/s5/sa_styles.html`.

[52] F. Hirsch, J. Kemp, and J. Ilkka. *Mobile Web Services: Architecture and Implementation*. Wiley, 2007.

[53] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[54] Koray Inki, Ismail Ari, and Hasan Sozer. A survey of software testing in the cloud. *Proc SERE*, 46(6):18–23, 2012.

[55] Software Engineering Institute. Glossary. `https://www.sei.cmu.edu/architecture/start/glossary/`.

[56] Masudul Islam and Mijanur Rahaman. A review on multiple survey report of cloud adoption and its major barriers in the perspective of bangladesh. *I. J. Computer Network and Information Security*, 5, 2016.

[57] ITU. Internet of things global standards initiative. `http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx`.

[58] Keith Jeffery, Geir Horn, and Lutz Schubert. A vision for better cloud applications. In *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds - MultiCloud '13*, 2013.

[59] Joel Jones. Abstract syntax tree implementation idioms. Technical report, Department of Computer Science, University of Alabama, Alabama, Estados Unidos da América, 2003.

[60] Mike Kavis. Google provides a glimpse into the future of cloud computing. `http://www.forbes.com/sites/mikekavis/2016/03/25/google-provides-a-glimpse-into-the-\penalty\@M\hskip\z@skip\discretionary{-}{}{}\penalty\@M\hskip\z@skipfuture-of-cloud-computing/#69c8bd6e529f`.

[61] Markus Klems, Jens Nimis, and Stefan Tai. Do clouds compute? a framework for estimating the value of cloud computing. *Lecture Notes in Business Information Processing*, (22):110–123, 2009.

[62] Stefan Kolb, Jörg Lenhard, and Guido Wirtz. Application migration effort in the cloud - the case of cloud platforms. *2015 IEEE 8th International Conference on Cloud Computing*, pages 41–48, 2015.

[63] Gerald Kotonya and Ian Sommerville. Cs 531 software requirements analysis and specification - nonfunctional requirements. `http://www.csm.ornl.gov/~sheldon/cs531/ch8.pdf`.

[64] Nane Kratzke. Lightweight virtualization cluster how to overcome cloud vendor lock-in. *Journal of Computer and Communications*, (2):1–7, 2014.

[65] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[66] Eugene Letuchy. Erlang at facebook. `https://www.erlang-factory.com/upload/presentations/31/EugeneLetuchy-ErlangatFacebook.pdf`.

[67] Antónia Lopes. Design de software i - o que caracteriza a actividade de design. medidas de qualidade. Faculdade de Ciências da Universidade de Lisboa, 2014.

[68] Chris McCoord. *Metaprogramming Elixir*. Pragmatic Bookshelf, Dallas, Texas, 2015.

[69] Cameron McKenzie. Developing for the cloud: How developing in the cloud is different. `http://www.theserverside.com/feature/Developing-for-the-Cloud-How-Developing-in-the-Cloud-\penalty\@M\hskip\z@skip\discretionary{-}{}{}\penalty\@M\hskip\z@skipis-Different/`.

[70] Gerar Meszaros. Mocks, fakes, stubs and dummies. `http://xunitpatterns.com/Mocks,%20Fakes,%20Stubs%20and%20Dummies.html`.

[71] Gerar Meszaros. Mocks, fakes, stubs and dummies. `http://xunitpatterns.com/Mocks,%20Fakes,%20Stubs%20and%20Dummies.html`.

[72] Microsoft. Chapter 1: Service oriented architecture (soa). `https://msdn.microsoft.com/en-us/library/bb833022.aspx`.

[73] Microsoft. Chapter 3: Architectural patterns and styles. `https://msdn.microsoft.com/en-us/library/ee658117.aspx`.

[74] Kashi Vishwanath Nachi Nagappan. Characterizing cloud computing hardware reliability. IEEE, June 2010.

[75] John Nickolls and William J. Dally. The gpu computing era. *IEEE Micro*, 30(2):56–69, 2010.

[76] Ainsley O'Connell. Inside erlang, the rare programming language behind whatsapp's success. https://www.fastcompany.com/3026758/inside-erlang-the-rare-programming-language-\penalty\@M\hskip\z@skip\discretionary{-}{}{}\penalty\@M\hskip\z@skipbehind-whatsapps-success.

[77] Institute of Electrical and Electronics Engineers. 830-1998 - ieee recommended practice for software requirements specifications. Technical report, Institute of Electrical and Electronics Engineers, 1998.

[78] National Institute of Standards and Technology. The nist definition of cloud computing. Technical report, U.S. Department of Commerce, National Institute of Standards and Technology Gaithersburg, MD, 2011.

[79] OpenGroup. Seconds since the epoch. http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_15, 2016. [Online; accessed 05-June-2016].

[80] Moussa Ouedraogo, Severine Mignon, Herve Cholez, Steven Furnell, and Eric Dubois. Security transparency: the next frontier for security research in the cloud. *Journal of Cloud Computing*, 4(1):4–12, 2015.

[81] Dana Petcu, Georgiana Macariu, Silviu Panica, and Ciprian Cračiun. Portable cloud applications - from theory to practice. *Future Generation Computer Systems*, 29(6):1417–1430, 2013.

[82] Tõnis Pool. Comparison of erlang runtime system and java virtual machine.

[83] Rails. Rails. http://rubyonrails.org/.

[84] Rails. Rails is made of people. http://rubyonrails.org/community/, 2016. [Online; accessed 04-June-2016].

[85] John Richardson. Open versus closed ended questions. Technical report, University of California, Califórnia, Estados Unidos da América, 2002.

[86] Leah Riungu-Kalliosaari, Ossi Taipale, and Kari Smolander. Testing in the cloud: Exploring the practice. *IEEE Software*, 29(2):46–51, 2012.

[87] Luis Rodero-Merino, Luis M. Vaquero, Victor Gil, Fermín Galán, Javier Fontán, Rubén S. Montero, and Ignacio M. Llorente. From infrastructure delivery to service management in clouds. *Future Gener. Comput. Syst.*, 26(8):1226–1240, October 2010.

[88] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley Longmann, Inc., 1999.

[89] S21sec Géstion S.A. S21sec. `http://s21sec.com/`, 2016. [Online; accessed 25-June-2016].

[90] Thomas Sandholm and Dongman Lee. Notes on cloud computing principles. *Journal of Cloud Computing: Advances, Systems and Applications*, 3(1):21–31, 2014.

[91] Elastic Security. How to detect side-channel attacks in cloud infrastructures. `https://elastic-security.com/2013/09/11/how-to-detect-side-channel-at\penalty\@M\hskip\z@skip\discretionary{-}{}{}\penalty\@M\hskip\z@skiptacks-in-cloud-infrastructures/`.

[92] Micole Sharlin, Evelyn Tu, and Thomas Bartus. *Guide to Creating Website Information Architecture and Content Creating*. Princeton University, 2008.

[93] Tim Sheard. *Semantics, Applications, and Implementation of Program Generation: Second International Workshop, SAIG 2001 Florence, Italy, September 6, 2001 Proceedings*, chapter Accomplishments and Research Challenges in Metaprogramming, pages 2–44. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[94] Gabriel Costa Silva, Louis M. Rose, and Radu Calinescu. A systematic review of cloud lock-in solutions. *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, pages 363–368, 2013.

[95] The Internet Society. The secure shell (ssh) protocol architecture. `https://tools.ietf.org/html/rfc4251`.

[96] Ian Sommerville. Challenges for cloud software engineering. Lecture.

[97] Thomas Østerlie. Cloud computing: Impact on software engineering research and practice. Lecture.

[98] Jeffrey Stylos. Plug-in architectures. `https://www.cs.cmu.edu/~bam/uicourse/830fall04/JeffStylosPlug-inArchitectures.ppt`.

[99] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing, 2009.

[100] Tenable. Nessus. https://www.tenable.com/products/nessus-vulnerability-scanner, 2016. [Online; accessed 03-June-2016].

[101] Abha Tewari, Pooja Nagdev, and Aarti Sahitya. Sky computing: The future of cloud computing. *International Journal of Computer Science and Information Technologies*, 6(4):3861–3864, 2015.

[102] Dave Thomas. *Programming Elixir*. Pragmatic Bookshelf, Dallas, Texas, 2014.

[103] Ericsson (Erlang/OTP unit). Erlang. https://www.erlang.org/, 2016. [Online; accessed 05-June-2016].

[104] José Valim. Adding elixir to existing erlang programs. http://elixir-lang.org/crash-course.html#adding-elixir-to-existing-erlang-programs.

[105] José Valim. alias, require and import. http://elixir-lang.org/getting-started/alias-require-and-import.html#use.

[106] José Valim. Application behaviour. http://elixir-lang.org/docs/master/elixir/Application.html#c:start/2.

[107] José Valim. Iex. http://elixir-lang.org/docs/stable/iex/IEx.html.

[108] José Valim and Plataformatec. Elixir. http://elixir-lang.org/.

[109] José Valim and Elixir Core Team. Typespecs and behaviours. http://elixir-lang.org/getting-started/typespecs-and-behaviours.html#behaviours, 2016. [Online; accessed 05-June-2016].

[110] Luis M. Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, 2011.

[111] VentureBeat. Why pinterest just open-sourced new tools for the elixir programming language. http://venturebeat.com/2015/12/18/pinterest-elixir/.

[112] W3. Xml. https://www.w3.org/XML/.

[113] Randy Wheeler. Cloud versus traditional software – from a business software developer's perspective. http://www.rootstock.com/erp-blog/cloud-versus-traditional-software-from-a-business-\penalty\@M\hskip\z@skip\discretionary{-}{}{}\penalty\@M\hskip\z@skipsoftware-developers-perspective/.

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