

Universidade do Minho

Mestrado Integrado em Engenharia Informática



Arquitetura e desenvolvimento

Grupo 5

Nuno Leite (A70132)

Raphael Oliveira (A78848)

Bruno Carvalho (A76987)

Joel Rodrigues (A79068)

João Alves (A77070)

Elisa Valente (A79093)

Hugo Oliveira (A78565)

Francisco Araújo (A79821)

24 de Janeiro de 2020

Conteúdo

1	Introdução	1
2	Arquitetura aplicacional	1
2.1	Tecnologias e <i>frameworks</i> utilizadas	3
2.1.1	Camada de dados	4
2.1.2	Camada de negócio	4
2.1.3	Camada de interface	5
3	Cloud providers	5
3.1	Interfaces de acesso	6
3.1.1	Sistema de tradução	6
3.1.2	Diferenças no processo de criação de <i>firewalls</i>	7
3.1.3	<i>Deployment</i> e atualização de uma arquitetura	9
3.2	Adição de novos <i>cloud providers</i>	10
4	Principais funcionalidades	11
4.1	<i>Deployment</i> de instâncias através de uma topologia	11
4.2	Comparação de preços entre <i>cloud providers</i>	12
4.3	Migração de arquiteturas entre <i>cloud providers</i>	13
4.4	Instalação de serviços	14
5	Desenvolvimento do produto	15
5.1	Ferramentas utilizadas	15
5.2	Documentação gerada	16
5.3	Metodologia de trabalho	16
6	Conclusão e trabalho futuro	17

1 Introdução

A *Stratus* consiste numa aplicação dedicada à gestão de máquinas e instalação de serviços na *cloud*.

A ideia surgiu da vontade de otimizar o tempo e custos associados à execução de tarefas de provisionamento e *deployment* na *cloud*. Apesar de já existirem ferramentas, estas requerem que o utilizador possua conhecimento profundo sobre o seu funcionamento. Além disso, exigem a codificação de ficheiros de configuração de grandes dimensões que, dependendo da dimensão do projeto, torna-se um processo tedioso e demoroso. Posto isto, o nosso produto pretende simplificar e abstrair todo este processo através de uma interface *web* que seja intuitiva e *easy-to-use*.



O nome *Stratus* provém do latim "**Stratu**", que significa camada. Além disso, **stratus** é um tipo de nuvem, caracterizado por se apresentar em camadas homogêneas e suaves. Combinando estes dois fatores e, sendo a nossa aplicação uma camada de abstração entre o utilizador e os *cloud providers*, este pareceu-nos o nome mais adequado. O logótipo da *Stratus* é constituído pela sobreposição de três hexágonos. Isto porque ao hexágono está associado o número 6, um número perfeito que simboliza confiança e segurança. As três camadas presentes no logótipo, representam o utilizador, a *Stratus* e os *cloud providers* respetivamente. A camada intermédia representa a nossa aplicação, que pretende facilitar e agilizar o trabalho do utilizador. O último hexágono tem presente um S em representação do nome *Stratus*. Assim, a mensagem que o logótipo pretende passar aos clientes, é que a *Stratus* é uma aplicação confiável e segura, pronta a agilizar os trabalhos na *cloud* através de uma interface *user-friendly*.

2 Arquitetura aplicacional

A *Stratus* está, essencialmente, dividida em duas aplicações principais: o *frontend* e o *backend*. O *frontend* surge como uma necessidade de fornecermos uma interface *user-friendly* aos utilizadores da *Stratus* que, apesar de não ser obrigatória, garante uma melhor interação entre o utilizador e os serviços disponibilizados. Esta interação

só é possível com a existência do *backend*, responsável por tratar de todo o modelo de negócio que a *Stratus* dispõe.

Na figura 1, é apresentada uma arquitetura simplificada da *Stratus*. Através da figura é possível identificarmos as duas aplicações principais, os serviços externos utilizados e, por fim, como é realizada a comunicação entre o *frontend* e *backend*.

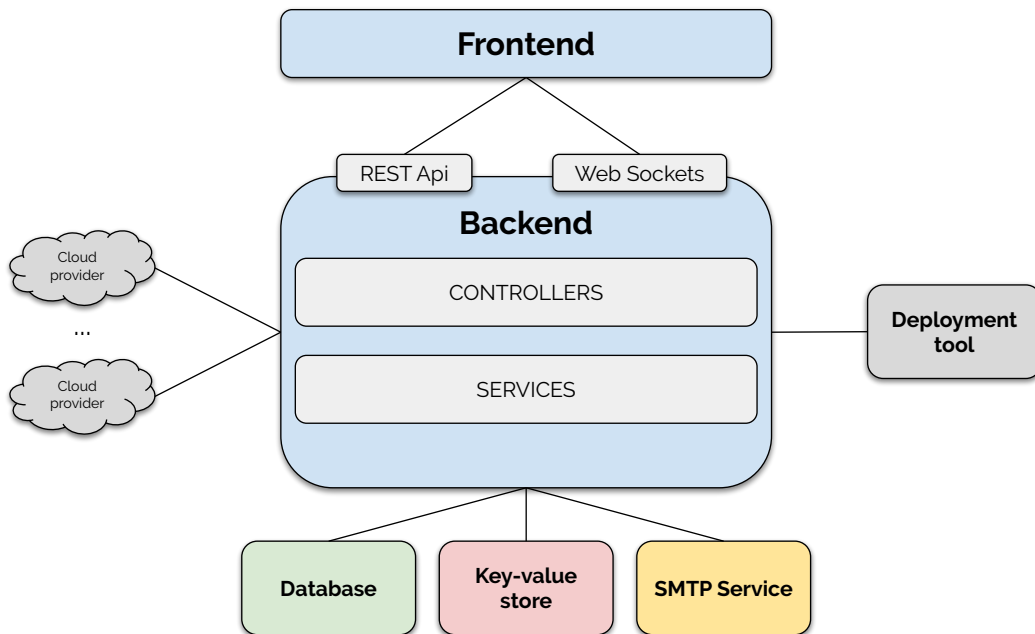


Figura 1: Arquitetura aplicacional adotada.

A *Stratus* foca-se na gestão de infraestruturas na *cloud*. Deste modo, depende, particularmente, dos *cloud providers* existentes.

A instalação automática de serviços é, também, outra funcionalidade muito importante e disponibilizada pela *Stratus*. Apesar desta funcionalidade não necessitar de ferramentas externas de *deployment* automático (por exemplo com a criação de *scripts*), reconhece-se que a sua utilização é uma boa alternativa pois permite agilizar e automatizar o processo de configuração e instalação de serviços em uma ou mais máquinas remotas.

Relativamente à comunicação entre o *frontend* e o *backend*, a *Stratus* recorre à utilização de dois mecanismos de comunicação muito frequentes no desenvolvimento de aplicações atuais: *API REST* e *Web Sockets*.

A utilização de uma *API REST* foi essencial para mantermos a integridade da comunicação entre o *frontend* e o *backend*. Uma interface deste tipo permite que

ambas as equipas do *frontend* e *backend* consigam estabelecer um protocolo bem estruturado, de fácil compreensão e que permita o desenvolvimento independente das duas aplicações. Esta interface foi utilizada para toda a criação, atualização e remoção de informação.

Os *Web sockets* surgem com a necessidade de ser apresentada informação em tempo real: resultado de tarefas, *logs*, etc. O grupo de desenvolvimento percebeu que a requisição contínua de informação ao *backend* é muito custoso. A solução passa por utilizar um paradigma *publish & subscribe* que, ao invés de ser o *frontend* a interrogar continuamente o *backend* sobre alterações, é o *backend* a enviar as atualizações em tempo real com a utilização de *sockets*. Esta integração com *Web sockets* permitiu que a *Stratus* fornecesse notificações idênticas a *push notifications* e ainda a visualização de *logs* em tempo real.

Na próxima secção são apresentadas as tecnologias e *frameworks* que permitiram o desenvolvimento da *Stratus*.

2.1 Tecnologias e *frameworks* utilizadas

Durante a fase inicial e estruturação da *Stratus*, o grupo estabeleceu as tecnologias que seriam utilizadas durante a fase de desenvolvimento e, consequentemente, a fase de produção da aplicação. Na figura 2, apresentamos a *stack* tecnológica da aplicação desenvolvida.

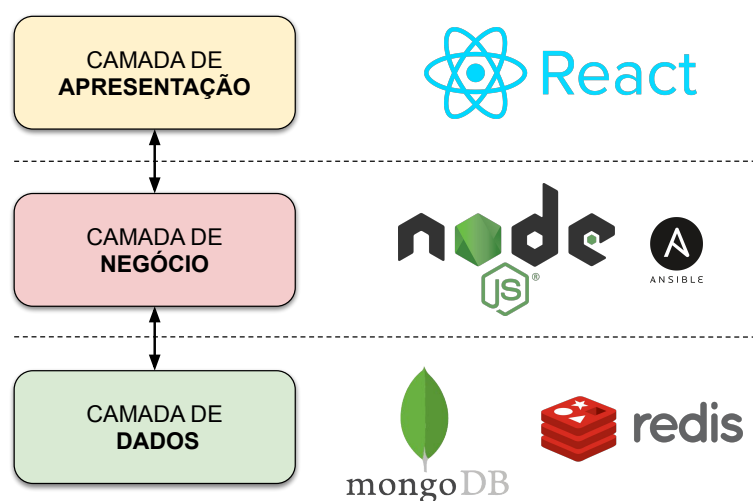


Figura 2: *Stack* tecnológica da aplicação desenvolvida.

2.1.1 Camada de dados

Relativamente à camada de dados, optamos pela utilização do **MongoDB** para a persistência de dados. Este motor de base de dados está bem integrado com o *NodeJS* o que permite a sua fácil utilização e conexão. No entanto reconhecemos que existem outros motores de gestão de base de dados que poderão fornecer outras funcionalidades ou melhorias de desempenho, como é o caso do *MySQL* ou *PostgreSQL*.

Foi ainda utilizado o **Redis**. O *Redis* é um serviço *key-value* que permite o armazenamento e procura de informação de uma forma muito rápida. Apesar de ser bastante utilizado para *caching*, neste projeto o *Redis* tem o papel de armazenar os *refresh tokens* das sessões dos utilizadores, permitindo diminuir a carga na base de dados e uma procura rápida por *token* de utilizador.

2.1.2 Camada de negócio

No que concerne à camada de negócio, utilizamos o **NodeJS**, uma *framework* do *javascript*. A escolha reflete-se dada a grande familiaridade da equipa de desenvolvimento com a mesma, por esta apresentar uma fácil integração com pedidos assíncronos e, ainda, por permitir a fácil criação de uma *API REST*.

O conceito de sessão entre *backend* e *frontend* foi estabelecido através de *JSON Web Tokens*, geralmente conhecidos por *tokens JWT*. A informação/identificação do utilizador é armazenada num formato *json* e posteriormente assinada com uma assinatura presente apenas numa das aplicações - neste caso no *backend*. Após uma autenticação válida, o *token* é partilhado com o *frontend*. Todos os pedidos posteriores efetuados pelo *frontend*, deverão conter o *token* gerado. Deste modo, o *backend* será capaz de validar a origem do *token*, isto é, verificar se foi assinado com a mesma assinatura durante a autenticação.

Como ferramenta de *deployment* automático optamos pela utilização do **Ansible**. Esta é uma ferramenta já utilizada por alguns dos elementos do grupo e apresenta uma documentação e uma comunidade extensa que permite dar suporte ao desenvolvimento de configurações e *scripts* de instalação e instanciação.

2.1.3 Camada de interface

Por fim, na camada de interface optamos pela utilização de **ReactJS**. Esta opção deve-se à sua grande utilização nos dias de hoje no que diz respeito ao desenvolvimento de aplicações *web*. Além disso, o *ReactJS* apresenta uma documentação extensa, uma grande comunidade *online* e um vasto leque de *frameworks* e bibliotecas que fornecem componentes estilizados que respeitam os padrões atuais de desenvolvimento de interfaces *web*.



Nesse sentido, utilizamos o **Material UI** como fonte principal de componentes e estilização, permitindo um desenvolvimento rápido e coerente da interface da *Stratus*. O *Material UI* é uma *framework open source* para desenvolvimento *web* que é mantida por uma comunidade extensa. Assim, a escolha do *Material UI* apresenta ser uma boa aposta a longo prazo.

3 Cloud providers

Sendo a aplicação da *Stratus* considerada uma camada intermediária entre o utilizador e os *cloud providers*, estes são parte integrante do bom funcionamento da aplicação. Por isso, as funcionalidades da *Stratus* dependem, em grande parte, da eficiência das ações nestes *cloud providers*, já que existe uma constante interação entre ambos. Por estes motivos, numa primeira instância foi necessário um estudo dos *cloud providers* mais utilizados e as suas *APIs* disponibilizadas.

Inicialmente e, como prova de conceito, a *Stratus* focou-se em duas grandes empresas de provisionamento na *cloud*: **Amazon Web Services** (AWS) e **Google Cloud Platform** (GCP).

Através do estudo, concluiu-se ainda que o desenvolvimento da *Stratus* era viável consoante as funcionalidades disponibilizadas. No entanto, foram encontrados alguns problemas, tais como restrições no acesso a algumas funcionalidades, bem como uma heterogeneidade de *APIs* entre os *cloud providers*.

3.1 Interfaces de acesso

Relativamente à interface de acesso, a *GCP* fornece uma *API* bastante completa, organizada e documentada que facilitou a sua utilização. Esta plataforma disponibiliza ainda uma biblioteca para *NodeJS*, para as diversas categorias (**Computação** e **Gestão de Projetos**), permitindo uma abstração da *API REST*, o que auxiliou bastante o desenvolvimento. Quanto à segurança, usou-se o protocolo *OAuth 2.0* que permite associar *tokens* do utilizador à *Stratus* permitindo o acesso, por parte destes, à *GCP*.

Em termos de interface de acesso da *AWS*, esta fornece também *APIs* bastante completas, mas muito extensas e esparsas, o que aumenta imenso a complexidade de tentar orquestrar todos os recursos que a mesma oferece quando necessário. Apesar deste problema, o facto de fornecerem um *SDK* passível de ser facilmente instalado pelo *Node Package Manager*, que abstrai as *APIs* web para um conjunto de objetos e métodos facilita, de certa forma, a utilização e compreensão da mesma. Para conseguir executar os pedidos que são despoletados quando chamámos um método de um dos objetos fornecidos pelo *SDK*, a *AWS* utiliza o conceito de credenciais de *IAM* (*Identity Access Management*) que permitem o acesso programático (via *web requests*). Neste sentido, é conveniente que um utilizador da *Stratus* insira as suas credenciais *AWS*, sendo essa a única vez que o terá que fazer, visto que quando é convidado para outro projeto, a *Stratus* trata de criar umas credenciais para o utilizador.

3.1.1 Sistema de tradução

Tendo em conta que ao fazer a pesquisa em ambos os *cloud providers*, se tornou óbvio que a nomenclatura utilizada por exemplo, para as regiões, para o tipo de máquinas, para o tipo de discos, entre outras, é bastante diferente, tornou-se necessário criar uma estrutura única que tornasse a *Stratus* flexível e adaptável em relação ao *cloud provider* utilizado. Dessa forma, através da *Stratus*, a nomenclatura utilizada seria sempre a nomenclatura *Stratus* que, posteriormente, seria traduzida para o *cloud provider* respetivo.

Para este efeito foi criado um sistema de tradução composto essencialmente por um conjunto de dicionários (chave-valor) e um conjunto de métodos capazes de

consultar o valor respetivo a um determinado *cloud provider*.

O sistema de tradução desenvolvido tem o objetivo de traduzir de um ponto comum para o respetivo cloud pro atributos existentes em qualquer fornecedor de *cloud provider*:

- **regiões** - A região onde um recurso deve ser implantado;
- **tipos de máquina** - O tipo da máquina que vai ser utilizado, isto é, um nome que indica uma máquina com determinadas características. Existiu uma clara tentativa neste ponto que a tradução fosse para máquinas o mais semelhantes possíveis;
- **tipos de disco** - O tipo de disco a utilizar, como por exemplo, *HDD* ou *SSD*, sendo que a nomenclatura dos mesmos varia de fornecedor para fornecedor;
- **Tipos de execução** - Como a máquina será executada no fornecedor. *On-demand* se é para estar sempre disponível para o utilizador ou utilização *Spot* que se refere a uma utilização dos recursos disponíveis do fornecedor;
- **sistemas Operativos** - O sistema operativo que irá executar na máquina escolhida.

3.1.2 Diferenças no processo de criação de *firewalls*

A criação de *firewalls* é um processo que serve para o utilizador ampliar os acessos a cada máquina (por norma, todo o tráfego de entrada está impedido), definindo em cada *firewall* o conjunto de protocolos (*TCP*, *UDP*, *ICMP*, entre outros) e portas associadas a esses protocolos. Desta forma, se o acesso externo às instâncias for permitido pelo utilizador, é possível a instalação de serviços nessas mesmas instâncias.

No lado da *GCP*, as *firewalls* são denominadas de "Regras de Firewall". Cada regra está associada a uma rede, ficando assim disponível para cada sub-rede dessa mesma rede. Assim, cada instância, que está inserida numa certa sub-rede, está sujeita às regras de *firewall* da sua sub-rede.

Nesta criação, é permitido incluir o conjunto de protocolos e o conjunto de portas para cada protocolo, como mencionado anteriormente. Ainda, é possível definir um intervalo de *IPs*, só aplicando essas regras às instâncias que estejam nesse intervalo de *IPs*. Por último, esta plataforma usa também o conceito de *tags*, no processo de criação de instâncias, isto é, os utilizadores podem introduzir umas certas *tags* nas instâncias que pretenderem e, aquando da criação de uma regra de *firewall*, pode associar também essa regra a umas certas *tags*, aplicando-a só às instâncias que possuam essas *tags*.

Na AWS, a criação de *firewalls* é baseada num recurso denominado de Grupo de Segurança cuja funcionalidade é, segundo a AWS, "atuar como uma firewall virtual que controla o tráfego de entrada e saída na instância ao qual está aplicado". Estes grupos atuam ao nível da instância, isto é, cada instância pode ter vários grupos de segurança, sendo que as regras aplicadas à instância são as regras resultado da junção de todos os grupos. Um grupo de segurança é descrito por:

- **nome:** Um nome que permita ter uma noção do que o grupo permite.
- **descrição:** Algo que permita identificar de uma forma mais sucinta, as regras que o grupo aplica.
- **identificador de rede:** A rede VPC da AWS na qual o grupo pode ser aplicado, ou à qual pertence.
- **permissões:** Um conjunto de regras que indicam a porta, protocolo e que origem está autorizada a aceder à porta definida nessa regra com esse protocolo.

É notório que, em termos conceptuais, tanto na GCP como na AWS, a aplicação de *firewalls* é baseada na atribuição de certas regras a uma rede, regras essas que podem, posteriormente, ser utilizadas por uma instância que pertença a uma sub-rede dessa rede, mas existe uma clara diferença na aplicação do conceito visto que, na GCP, cada regra singular é associada à rede e posteriormente aplicada, enquanto que no caso da AWS é criado um recurso chamado Grupo de Segurança, que contém um conjunto de regras, que é associado à rede, grupo esse que é depois totalmente associado a uma instância, sendo aplicadas à mesma todas as regras do grupo de segurança.

3.1.3 *Deployment* e atualização de uma arquitetura

Nesta secção pretendemos abordar um dos aspetos que, de facto, torna a *Stratus* diferenciadora e inovadora tanto no que diz respeito à sua concorrência como no que diz respeito à complexidade do problema que enfrentava, visto que, orquestrar um *deployment* conjunto, composto por várias máquinas e redes e garantir que o mesmo é viável em qualquer um dos fornecedores de cloud é uma tarefa realmente árdua. Basta termos em mente que, de um *cloud provider* para outro, os processos de criação de instâncias e redes ou até de criação de *firewalls*, por exemplo, mudam drasticamente, o que faz com que a *Stratus* tenha que ter a capacidade de se adaptar aos diferentes fornecedores e ser flexível o suficiente para aceitar novas formas de implantação de recursos num dado fornecedor, até porque é necessário este *future-proofing* para que seja possível, no futuro, adicionar mais *cloud providers* na *Stratus*. A premissa principal desta funcionalidade é a seguinte: Para exactamente o mesmo objecto de dados, contendo informação sobre as instâncias, redes e sub-redes a implantar no fornecedor de cloud, criar exactamente a mesma arquitectura em qualquer um dos fornecedores de cloud suportados que seja escolhido. Para conseguir cumprir com essa premissa criámos uma junção de mecanismos que nos permitissem implantar e manter atualizada a arquitetura em qualquer um dos fornecedores de cloud:

- tratamento conjunto (uniforme) no *frontend* como *Stratus*, uniformizando toda a interacção, sendo que este tratamento uniforme se estende à entrada dos dados no *backend*, isto é, o objeto de dados é igual seja dado como entrada num método da API da GCP ou num método da API da AWS.
- tratamento lógico separado no *backend*, facilitando o desenvolvimento e a futura adição de novos fornecedores de cloud.
- finalmente, o sistema de tradução anteriormente abordado que permite, a partir do objeto de dados único, chegar ao fornecedor de cloud escolhido.

Esta uniformização feita na interface e separação lógica no *backend* permite que, facilmente, se migre a arquitectura criada para outro fornecedor de cloud. Denotar que, neste momento, a *Stratus* apenas cria a mesma infraestrutura noutro fornecedor, não

recriando qualquer conteúdo que a infraestrutura anterior tenha. Esta tarefa de migração total seria algo a implementar numa versão futura.

3.2 Adição de novos *cloud providers*

Como já referido, neste momento, a *Stratus* tem dois fornecedores de *cloud*, *GCP* e *AWS*, mostrando assim ao utilizador que permitimos uma heterogeneidade de fornecedores. No entanto, num futuro próximo a ideia é associar mais empresas, por exemplo a **Azure** e **IBM Cloud**, dando assim, ao utilizador uma grande variedade de escolhas no momento do *deploy*. Tendo em conta isto, o *backend* da aplicação já foi desenvolvido de uma forma modular, permitindo que se adicionem mais *providers* de uma forma mais simplista. Isto é, por cada adição, é necessário criar um módulo que contenha a implementação de todas as interações necessárias com o *provider* (*APIs*) e incorporar no sistema de tradução todas as correspondências necessárias. O módulo a criar é, na realidade, um conjunto de módulos que têm que ser criados, de acordo com a arquitetura aplicacional definida na secção 2. Conceptualmente, esse módulo pode ser descrito, em termos das suas componentes, da seguinte forma:

- **routes** - A definição das rotas pertencentes à API REST, que permitem a interação com os recursos desse fornecedor de cloud.
- **controllers** - Os métodos responsáveis por implementar a funcionalidade respeitante a cada uma das rotas.
- **services** - Os métodos responsáveis por interagir com o fornecedor de cloud respetivo e com a base de dados da *Stratus*, mantendo os dados íntegros em ambas as partes.
- **common information** - A informação que deve ser acrescentada aos dicionários de tradução.

Este módulo, composto pelas 4 componentes, pode ser desenvolvido de forma completamente independente, tendo noção apenas dos dados de entrada e do que deve ser os dados de saída (especificação da *API*), o que destaca a modularidade da aplicação *Stratus*.

4 Principais funcionalidades

Esta secção tem como objetivo destacar os fatores que diferenciam a aplicação *Stratus* dos diversos concorrentes, representando as principais funcionalidades desta aplicação.

4.1 *Deployment* de instâncias através de uma topologia

Um dos principais objetivos da *Stratus* é fornecer um serviço intuitivo e simples de utilizar, abstraindo o utilizador de toda a complexidade exigida pelos problemas que pretendemos resolver. Desta forma, será possível não só ganhar a confiança de consumidores que enfrentam estes desafios no dia-a-dia, mas também maximizar o nosso público alvo.

A principal característica que nos distingue do resto da concorrência consiste na projeção de uma topologia que representa a arquitetura que se pretende instalar. A representação da arquitetura é simples, no entanto a principal preocupação foi fornecer uma boa experiência ao utilizador. Desta forma, a sua representação pode otimizar e rentabilizar melhor o seu tempo, que de outra forma seria gasto nesta tarefa de configurar extensivamente cada componente da sua arquitetura. Além disso, permite também, que os utilizadores com menos conhecimento técnico neste tipo de atividades, possam usar a nossa ferramenta.

Na Figura 3, encontra-se uma ilustração, retirada da aplicação *Stratus*, que representa um exemplo de uma arquitetura onde se poderia efetuar o seu *deployment* na *cloud* com apenas um clique. Este nível de flexibilidade traduz-se pela existência de um modelo capaz de representar, fielmente, todos os elementos que constituem uma arquitetura, assim como as suas ligações e características fundamentais, sob a forma de um diagrama.

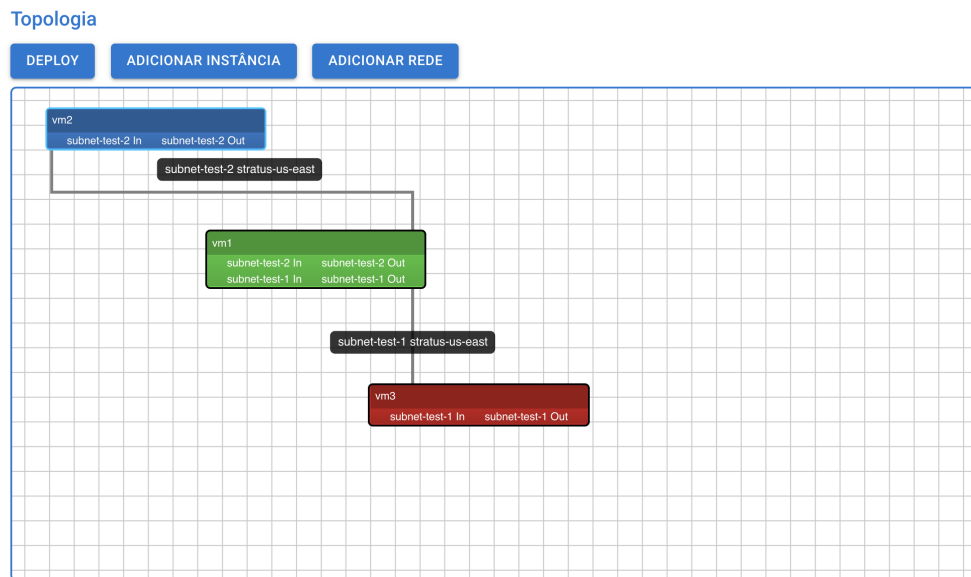


Figura 3: Visão topológica de uma arquitetura na *Stratus*.

Numa primeira instância, o utilizador necessita de associar o seu projeto a uma zona geográfica onde os seus recursos vão ficar alojados aquando do *deployment* num *cloud provider*. Posteriormente, será possível ao utilizador desenhar uma arquitetura.

A primeira etapa lógica no desenho de uma arquitetura é adicionar uma rede ao seu *workspace*, inserindo as características constituintes - por exemplo o nome, intervalo de *IPs* e *firewalls*. Numa fase seguinte, já com redes associadas ao projeto, é possível adicionar as instâncias. O utilizador deverá preencher um formulário com os atributos essenciais de uma instância - por exemplo o seu tipo, os discos e as redes a que pertence. Por fim, é possível o utilizador conectar duas instâncias que estejam na mesma rede.

Finalmente, com o desenho concluído, o utilizador pode fazer o *deploy* da arquitetura num dos *cloud providers* ou guardar a mesma para poder ser usada no futuro.

4.2 Comparação de preços entre *cloud providers*

Como mencionado anteriormente, um dos principais problemas com que muitos profissionais se deparam é encontrar o *cloud provider* que oferece os melhores preços. Neste seguimento, depois de o utilizador implementar a sua arquitetura (exemplo

da figura anterior) e quiser efetuar o *deploy* da mesma, a nossa aplicação apresenta a listagem dos preços, para cada *cloud provider*, de forma a dar a possibilidade ao utilizador de escolher a opção mais rentável.

Escolher provisionador de cloud

Provisionador de cloud	Instâncias	Preço (\$ / Mês)	Total (\$ / Mês)	
GCP	vm3	3.17	9.51	<input checked="" type="radio"/>
	vm1	3.17		
	vm2	3.17		
AWS	vm3	2.81	8.43	<input type="radio"/>
	vm1	2.81		
	vm2	2.81		

VER DETALHES CANCELAR CONFIRMAR

Figura 4: Comparação de preços entre *cloud providers*

Como se pode verificar na figura anterior, antes de associar um projeto a um *cloud provider*, isto é, antes de realizar um primeiro *deploy*, o utilizador tem acesso aos preços de cada máquina, em cada *cloud provider*, bem como os preços totais (por mês). Além disto, o utilizador pode consultar os preços mais detalhadamente (preços por cada componente da máquina), como apresentado na seguinte figura:

Máquina Virtual: vm3 - Preemptível

		Preço AWS	Preço GCP
Tipo de máquina	stratus-cpu1-ram1	2.56 \$	2.73 \$
Tipo de disco	stratus-standard	0.025 \$/mês por Gb	0.044 \$/mês por Gb
Tamanho do disco	10 Gb	0.25	0.44
Discos adicionais			
Preço total		3.17	2.81

Figura 5: Detalhes dos preços.

4.3 Migração de arquiteturas entre *cloud providers*

Uma das características a destacar na *Stratus*, é a possibilidade de um utilizador efetuar migrações de arquiteturas entre *cloud providers*. No entanto, é importante

perceber que esta migração não é da arquitetura já em produção. Isto é, apenas a infraestrutura é migrada - as suas instâncias e rede. Todas as instalações de serviços e adições prévias de membros ao projeto não são, efetivamente, transpostas para o novo projeto criado após a migração.

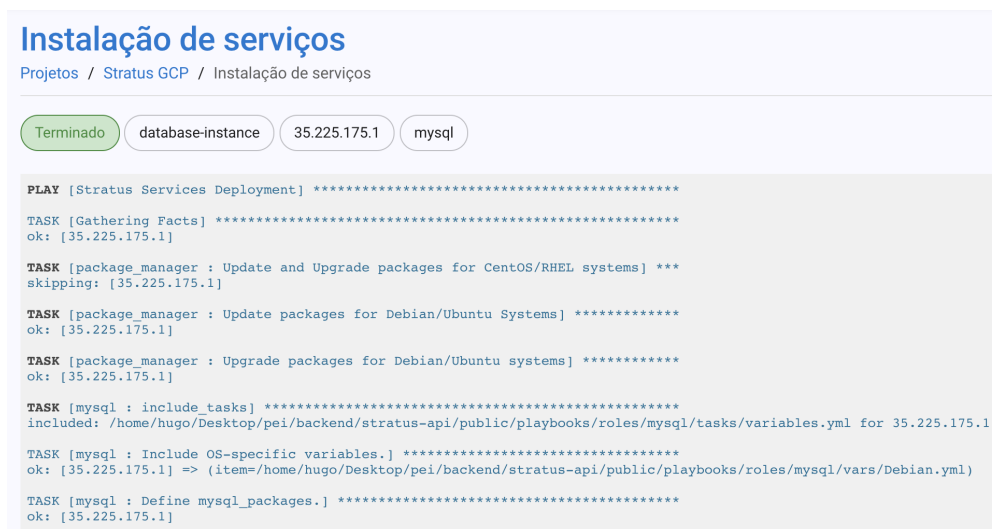
Este conceito de migração foi pensado apenas, para uma fase permatura do projeto do utilizador. Para projetos complexos e com arquiteturas já em produção - por exemplo bases de dados com informação - este conceito deixa de ter relevância, pois nenhuma da informação contida nas instâncias será transferida para as novas instâncias.

4.4 Instalação de serviços

Após o *deployment* de uma arquitetura ter sido efetuado com sucesso, é possível instalar serviços numa dada instância.

Tal como foi mencionado anteriormente, recorre-se ao *Ansible* para efetuar a instalação automática de serviços, no entanto a sua utilização é transparente ao utilizador.

Uma instalação pode conter um ou mais serviços e está associada a uma única instância. Durante o processo de instalação, é possível utilizar *templates* de configuração previamente definidos e indicar os valores necessários para melhor satisfazer as necessidades do utilizador.



The screenshot displays the 'Instalação de serviços' (Service Installation) page. At the top, the breadcrumb navigation shows 'Projetos / Stratus GCP / Instalação de serviços'. Below this, a status bar indicates 'Terminado' (Completed) in a green circle, followed by the instance name 'database-instance', the IP address '35.225.175.1', and the service 'mysql'. The main content area shows the Ansible execution logs for the 'Stratus Services Deployment' play. The logs detail the tasks performed: 'Gathering Facts', 'Update and Upgrade packages for CentOS/RHEL systems' (skipped), 'Update packages for Debian/Ubuntu Systems', 'Upgrade packages for Debian/Ubuntu systems', 'include_tasks' for the MySQL role, 'Include OS-specific variables', and 'Define mysql_packages'. All tasks completed successfully on the host 35.225.175.1.

```
PLAY [Stratus Services Deployment] *****
TASK [Gathering Facts] *****
ok: [35.225.175.1]

TASK [package_manager : Update and Upgrade packages for CentOS/RHEL systems] ***
skipping: [35.225.175.1]

TASK [package_manager : Update packages for Debian/Ubuntu Systems] *****
ok: [35.225.175.1]

TASK [package_manager : Upgrade packages for Debian/Ubuntu systems] *****
ok: [35.225.175.1]

TASK [mysql : include_tasks] *****
included: /home/hugo/Desktop/pei/backend/stratus-api/public/playbooks/roles/mysql/tasks/variables.yml for 35.225.175.1

TASK [mysql : Include OS-specific variables.] *****
ok: [35.225.175.1] => (item=/home/hugo/Desktop/pei/backend/stratus-api/public/playbooks/roles/mysql/vars/Debian.yml)

TASK [mysql : Define mysql_packages.] *****
ok: [35.225.175.1]
```

Figura 6: Logs da instalação automática de um serviço.

Após a submissão do pedido de instalação de um ou mais serviços, o utilizador pode visualizar, em tempo real, os *logs* que o *Ansible* gera durante a instalação, como se pode observar na Figura 6. A funcionalidade de visualização em tempo real foi desenvolvida com a utilização de *Web Sockets*.

5 Desenvolvimento do produto

Ao longo desta secção serão abordadas algumas características relativas ao desenvolvimento do produto, tais como, ferramentas utilizadas, documentação gerada e metodologia de trabalho.

5.1 Ferramentas utilizadas

Uma vez que o *backend* da *Stratus* utiliza vários serviços externos, foi necessário adotar medidas que permitissem que a equipa de desenvolvimento apresentasse um ambiente de desenvolvimento estável e igual entre os vários elementos.

Nesse sentido, durante o desenvolvimento do *backend*, a equipa recorreu à utilização de *containers* através do **Docker**. A equipa utilizou o *Docker* para instalar os serviços *MongoDB* e *Redis*.

A utilização de *containers* permitiu, desde logo, evitar alguns erros associados à instalação e configuração dos serviços:

- existem várias versões de sistemas operativos diferentes na equipa: *MacOS*, *Linux* e *Windows*. A utilização do *Docker* permitiu abstrair a instalação e configuração de serviços em cada um dos sistemas operativos;
- a utilização do *Docker* permitiu que todos os elementos utilizassem a mesma versão para cada um dos serviços. Durante o desenvolvimento é importante que sejam despistados erros associados a versões incompatíveis com o ambiente de produção.

Além da utilização do *Docker*, foi unânime a escolha de *plugins* que permitissem a normalização do código desenvolvido. Isto é, *plugins* que obrigam todos os elementos do grupo a utilizarem a mesma identificação e formatação do código. Isto

permitiu manter a consistência de todo o código desenvolvido. Os *plugins* utilizados foram o **ESLint** e o **Prettier**.

5.2 Documentação gerada

Desde a fase inicial do desenvolvimento que a equipa decidiu efetuar uma documentação detalhada e robusta da *API REST* do *backend*, essencialmente por dois fatores:

- na perspetiva do desenvolvimento, sendo a equipa de vários elementos, é essencial ter uma documentação detalhada para todos os elementos terem conhecimento do trabalho desenvolvido;
- na perspetiva da aplicação, a documentação fornece aos utilizadores, todos os detalhes de forma a interagirem com a *Stratus* por meio de uma *API REST*.

Neste sentido, usou-se a biblioteca **Api Doc** que permite exportar a documentação num formato *HTML* de fácil visualização. Além disso, o *ApiDoc* possui uma fácil integração com o *NodeJS*.

5.3 Metodologia de trabalho

No que toca ao desenvolvimento do produto, tentou-se seguir sempre uma metodologia de **desenvolvimento ágil**. Esta metodologia permitiu que fosse desenvolvida uma aplicação de alta qualidade e que respeitasse os objetivos definidos pela *Stratus*.

Na fase de iniciação, identificaram-se todos os problemas subjacentes à utilização da *cloud*, permitindo identificar os principais requisitos e funcionalidades da aplicação bem como a arquitetura geral da mesma. Durante esta fase foi ainda importante dividir a equipa de desenvolvimento pelas mais variadas funcionalidades.

Nas fases subsequentes do *sprint* de desenvolvimento, a equipa optou por se reunir duas vezes por semana: uma reunião em contexto da unidade curricular, onde eram definidas as novas metas de trabalho; outra reunião, essencial para validar as funcionalidades desenvolvidas, ajudar nos problemas encontrados e para desenvolvimento em grupo.

Relativamente às fases posteriores - *solution release* e *post production* - o grupo foi, durante o desenvolvimento da aplicação, efectuando alguns testes para validar as funcionalidades. No entanto, não foram efetuados testes de produção que permitissem validar todo o desenvolvimento, dada a dificuldade em encontrarmos utilizadores com disponibilidade para o teste da aplicação.

6 Conclusão e trabalho futuro

Tendo concluído o trabalho a que nos propusemos inicialmente, é importante inserir nesta secção uma reflexão sobre o que foi o trabalho desenvolvido, incluindo a qualidade do mesmo, as dificuldades encontradas e as perspectivas para o futuro da *Stratus*.

É consensual entre o grupo que, tendo em conta a complexidade das questões abordadas neste projecto, tais como a uniformização de recursos entre vários *cloud providers* e o deploy de uma arquitectura em qualquer um dos mesmos, o trabalho desenvolvido é bastante satisfatório, principalmente no que diz respeito ao nível técnico.

O facto de não possuímos experiência sobre o negócio subjacente ao nosso produto e o facto do desenvolvimento do produto ser extremamente técnico e alusivo à área da computação na *cloud*, dificultou a explicação do modelo a pessoas não relacionadas com a área.

O grupo tem plena confiança que o produto que desenvolveu facilita um processo que é usualmente moroso, o que significa poupança de tempo e, nos dias que correm, tempo é dinheiro. Esta confiança foi aumentada pelo Eng. Hugo Portela e por outros elementos da *Accenture Technology Center* em Braga, que viram potencial no produto, especialmente por este apresentar uma enorme capacidade de evolução. Isto é, se até ao momento oferecemos uma interface gráfica para que utilizadores menos experientes possam utilizar a *cloud* sem preocupações, o futuro pode reservar, com certeza, uma interface de linha de comandos que, usualmente, é preferível quando a *cloud* é utilizada em processos de integração contínua no desenvolvimento de *software*.

Além disso, é expectável que sejam adicionados ainda mais serviços e *cloud providers*, o que permitirá uma escolha bem mais diversificada e uma comparação de preços com uma base bem mais forte. Outra funcionalidade que se perspetiva adicionar à *Stratus* passa por permitir construir arquiteturas híbridas, isto é, arquiteturas com recursos provenientes de *cloud providers* diferentes. Pretende-se ainda, desenvolver um sistema de monitorização com *dashboards* que permitam a visualização do uso dos recursos de cada projeto.

Finalmente, também se idealiza incorporar mecanismos que permitam ao utilizador gerir a faturação dos respetivos *cloud providers*, diminuindo a necessidade destes terem de aceder às interfaces dos *providers*, tornando a *Stratus* uma aplicação completamente independente.