INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia Eletrónica e de Telecomunicações e Computadores

Desenvolvimento de Vertente SaaS para Produto de Submissão de Dados em Plataformas de Business Intelligence

Diogo Gomes Pinto Guerreiro

Licenciado

Relatório de Desenvolvimento do Trabalho para obtenção do Grau de Mestre em Engenharia Informática e de Computadores

Orientador : Prof. Doutor Ricardo Filipe Silva

Júri:

Presidente: Prof. Doutor José Manuel de Campos Lages Garcia Simão

Vogais: Prof. Doutor Nuno Miguel Soares Datia
Prof. Doutor Ricardo Filipe Silva

Maio, 2022

Aos meus queridos pais

Agradecimentos

Agradeço ao meu orientador Ricardo Silva e a todos os meus professores tanto da licenciatura como do mestrado, por todos os ensinamentos que me deram nos últimos cinco anos.

Ao Ricardo Pires da *Xpand IT*, por todo o apoio e grande disponibilidade, desde o início da dissertação, tendo sido crucial para a sua realização.

À minha colega e melhor amiga Adara Nogueira, que sempre acreditou em mim e me apoiou, sendo uma inspiração para mim.

À minha mãe Maria Jorge Montez, por ter acreditado no meu potencial, pelo apoio incondicional, pelos seus cuidados constantes e pelo orgulho que tem em mim.

Ao meu pai Luis Guerreiro, por me ter proporcionado estudar em Lisboa e também pelo orgulho que tem em mim. Também gostaria de agradecer à Telma Guerreiro e à minha querida irmã Clara Guerreiro, por fazerem parte da minha vida.

Ao Cristóvão Fernandes pelo seu apoio durante a realização desta dissertação.

Às minhas queridas avós, Isabel Montez e Amerilde Guerreiro, por serem sempre tão carinhosas comigo.

Aos meus amigos Gonçalo Almeida e Luis Loureiro, por termos vivido esta jornada juntos.

Resumo

O *Tableau* é uma ferramenta de *Business Intelligence* que contém uma extensão denominada *Write-Back*, criada pela empresa *Xpand IT*, que permite a escrita dinâmica de conjuntos de dados no *Tableau*. Com esta dissertação, pretende-se efetuar a migração da aplicação *Write-Back*, que é uma aplicação *on-premise*, para um *Software as a Service*.

Esta migração foi feita na plataforma da *Microsoft Azure* com um *Azure App Service*, um *Azure SQL Database*, um *Azure Front Door* e um *Azure Application Insights*.

Efetuou-se um estudo da aplicação *Write-Back on-premise* atual e sobre *Software as a Service* (*SaaS*), verificando-se que são precisos assegurar 3 requisitos principais para construir uma aplicação *SaaS*, nomeadamente a segurança e *multi-tenant*, a escalabilidade e a robustez.

Seguidamente, foram efetuadas algumas mudanças para assegurar uma arquitetura *multi-tenant*, como foi o caso da gestão dos *tenants* de forma dinâmica, a otimização das *queries* da base de dados do *Write-Back* e a garantia da segregação total entre os utilizadores.

Por último, foram efetuados testes de desempenho na aplicação *SaaS*, tendo-se chegado à conclusão de que com a capacidade e da versão do servidor adquirido, a aplicação conseguirá suportar entre aproximadamente 400 e 600 interações, com um período de crescimento de 0.1 segundos.

Palavras-chave: Migração; *SaaS*; Computação na Nuvem; *Write-Back*; *Tableau*.

Abstract

Tableau is a Business Intelligence tool that contains an extension called Write-Back, created by the company Xpand IT, which allows dynamic writing of datasets in Tableau. With this dissertation, we intend to migrate the Write-Back application, which is an on-premise application, to a Software as a Service.

This migration was performed on the Microsoft Azure platform with an Azure App Service, an Azure SQL Database, an Azure Front Door and an Azure Application Insights.

A study was carried out on the current on-premise Write-Back application and on Software as a Service (SaaS), verifying that it is necessary to ensure 3 main requirements to build a SaaS application, namely security and multi-tenant, scalability and robustness.

Then, some changes were made to ensure a multi-tenant architecture, such as the dynamic management of tenants, the optimization of the Write-Back database queries and the guarantee of total segregation between users.

Finally, performance tests were carried out on the SaaS application, reaching the conclusion that with the capacity and version of the server purchased, the application will be able to support between approximately 400 and 600 interactions, with a ramp-up period of 0.1 seconds.

Keywords: Migration; SaaS; Cloud Computing; Write-Back; Tableau.

Índice

Lista de Figuras **xv**

Lista de Listagens **xvii**

Lista de Abreviaturas e Siglas **xix**

1 Introdução **1**

1.1 Contexto e Motivação 1

1.2 Problema 2

1.3 Objetivos e Contribuições 3

1.4 Estrutura do Documento 4

2 Trabalho Relacionado **5**

2.1 *Write-Back* 5

2.1.1 Autenticação 10

2.1.2 Front-end 11

2.1.3 Back-end 11

2.1.4 Base de Dados 12

2.1.5 Tecnologias 14

2.2 *Software as a Service* 14

2.2.1 Modelos de Serviço na Nuvem 15

2.2.2 Arquitetura e Implementação 15
2.2.3 Processos de Migração de *On-premise* para *Software as a Service* . . 17

3 *Write-Back SaaS* 21

3.1 Segurança 22
3.1.1 Autenticação 22
3.1.2 Aplicação 23
3.1.2.1 Organização *Multi-tenancy* 23
3.1.2.2 Bases de Dados do *Write-Back* 24
3.1.3 Transmissão 25
3.2 Escalabilidade 25
3.2.1 *Load Balancer* 26
3.2.2 *Front-end* 27
3.2.3 *Back-end* 29
3.2.4 Base de Dados 30
3.3 Robustez 31

4 Implementação 33

4.1 Configuração na Versão *On-premise* 33
4.2 Componentes *Microsoft Azure* 33
4.2.1 *Azure App Service* 34
4.2.2 *Azure SQL Server* 35
4.2.3 *Azure Front Door* 36
4.3 Preocupações com o Multi-tenancy 36
4.3.1 Gestão dos *tenants* de forma dinâmica 37
4.3.1.1 Cenário exemplo da gestão de *tenants* 37
4.3.2 Otimização dos *drivers* e das *queries* das bases de dados 38
4.3.3 Segregação total 39

5 Resultados **41**

5.1 Testes de Desempenho 41

5.2 *TabJolt* 41

5.3 *JMeter* 42

5.3.1 Testes de desempenho com o *JMeter* 43

5.3.2 Resultados dos testes 45

6 Conclusões **47**

6.1 Trabalho Futuro 48

Referências **49**

Lista de Figuras

2.1 *Write-Back Demo* 6

2.2 Exemplo do *Write-Back* para inserção de dados [22] 6

2.3 Arquitetura *Tableau* e *Write-Back* [22] 8

2.4 Arquitetura do servidor *Write-Back* 9

2.5 Arquitetura *Back-end* [22] 11

3.1 Arquitetura Write-Back SaaS 22

4.1 Mapa de servidores da *azure* 34

5.1 Diagrama temporal de um teste de desempenho com o *JMeter* com 200 *Threads* 43

Lista de Listagens

4.1 Código para fazer *deploy* da aplicação 34
4.2 Código para o alocamento do repositório *git* na *Azure App Service* 34
4.3 Código *SQL* para os *logins* 35
4.4 Código *SQL* para o *user* e *schema* para a base de dados interna 35
4.5 Código *SQL* para o *user* e *schema* da base de dados do cliente 36

Lista de Abreviaturas e Siglas

AGU	Address Generation Unit.
AKS	Azure Kubernets Services.
API	Application Programming Interface.
AWS	Amazon Web Services.
BI	Business Intelligence.
BLOB	Binary Large Object.
CDN	Content Delivery Netwrok.
CSS	Cascading Style Sheets.
eDTU	Elastic Database Transaction Unit.
ETL	Extract Transform Load.
FTP	File Transfer Protocol.
GB	Gigabyte.
GUID	Globally Unique Identifier.
HTML	HyperText Markup Language.
HTTP	Hypertext Transfer Protocol.
HTTPS	Hyper Text Transfer Protocol Secure.
IaaS	Infrastructure as a Service.
IdP	Identity Provider.
IP	Internet Protocol.
JAR	Java Archive.
JDBC	Java Database Connectivity.
JMX	Java Management Extensions.
JS	JavaScript.
Json	JavaScript Object Notation.
LDAP	Lightweight Directory Access Protocol.

NPM Node Package Manager.
PaaS Platform as a Service.
RAM Random-access Memory.
REST Representational State Transfer.
SaaS Software as a Service.
SAML Security Assertion Markup Language.
SGBD Sistema de Gestão de Base de Dados.
SPOF Single Point of Failure.
SQL Structured Query Language.
SSH Secure Shell.
SSL Secure Sockets Layer.
SSO Single Sign On.
SSPI Security Support Provider Interface.
stdout standard output.
TCP Transmission Control Protocol.
TLS Transport Layer Security.
URL Uniform Resource Locator.
VPN Virtual Private Network.
WAF Web Application Firewal.
WAR Web Application Resource.
XML Extensible Markup Language.
YAML Yet Another Markup Language.

1

Introdução

Este capítulo visa descrever a motivação, a formulação do problema, os objetivos da dissertação e a estrutura do documento.

1.1 Contexto e Motivação

Business Intelligence (*BI*) refere-se ao processo de recolher, armazenar, e analisar os dados produzidos pelas actividades de uma organização [52].

Este processo tem como objectivo auxiliar as organizações no processo de tomada de decisão, quer estas sejam decisões estratégicas ou tácticas. Por exemplo, as organizações utilizam *BI* para identificar novas oportunidades de negócio, identificar processos de negócio ineficientes, entre outros [52].

As ferramentas de *BI* acedem e analisam conjuntos de dados e apresentam-nos sob a forma de relatórios, sumários, fazendo uso de diferentes visualizações como gráficos, tabelas ou mapas de forma a fornecer aos utilizadores a inteligência detalhada sobre o estado das suas organizações.

Uma das ferramentas de *BI* disponíveis no mercado é o *Tableau* [83].

Diversas organizações optam por utilizar o *Tableau* uma vez que é uma ferramenta que fornece um vasto leque de métodos de visualização de dados, possibilita conectar com variadas fontes de dados e está desenhada para ser utilizada por utilizadores leigos, isto é, não requer formação em ciência da computação [1].

O *Tableau*, por si só, é apenas como uma ferramenta de visualização. No entanto, existem processos de *BI* que quando adicionados ao *Tableau*, garantem outras potencialidades deixando de ser apenas uma ferramenta de visualização (como por exemplo extensões que estendem o *Tableau* tornando-o numa ferramenta mais próxima do *BI*). Estes processos requerem a inserção e configuração dos mesmos pelos utilizadores.

1.2 Problema

A *Xpand It* identificou uma limitação no *Tableau*. Esta limitação deve-se ao facto de o *Tableau* não permitir escrever dados diretamente através do *dashboard*. Um *dashboard* são representações de dados, de forma visual, através de um painel. Com os dados apresentados desta forma, é possível controlar e analisar os dados de forma mais eficiente para alcançar um objetivo com estas análises. Portanto, significa que com apenas o *Tableau* não é possível editar estes mesmos dados diretamente no painel visual do *dashboard*. Isto fez com que a *Xpand It* desenvolvesse o *Write-Back* como resposta ao problema.

O *Write-Back* [91], uma extensão do *Tableau*, é uma solução *on-premise* que consiste na escrita (inserção ou edição) dinâmica de conjuntos de dados diretamente numa plataforma de *Business Intelligence* (*Tableau*). Com a utilização do *Write-Back*, os utilizadores passam a poder inserir os dados diretamente no *dashboard*. O *Write-Back* está desenhado para diversos cenários de utilização, isto é, para vários propósitos que as organizações possam ter.

Um exemplo de utilização do *Write-Back*, encontra-se abaixo apresentado:

Problema: Uma organização "A", com uma base de dados complexa e de grande dimensão, utiliza o *Tableau* de modo a monitorizar e analisar os seus respetivos dados. Entretanto, esta organização pretende fazer um modelo predictivo (*forecasting*), introduzindo dados manualmente. No entanto, apenas com o *Tableau* isso não é possível.

Solução: Utilizar o *Write-Back*. A organização "A" deve instalar a aplicação numa das suas máquinas e configurar o *Write-Back* de modo a que o mesmo se possa ligar à base de dados dessa organização. Com o *Write-Back* funcional, a organização "A" poderá fazer *forecasting*, no qual cada elemento que constitui a equipa do projeto em questão, fica responsável por prever dados até uma certa data, como por exemplo, uma estimativa de valores relativamente aos gastos ou ganhos da organização. No final, todas estas estimativas poderão ser agregadas com o *Write-Back*. Para além dos recursos para fazer *forecasting*, é possível haver um registo de todas as ações realizadas na base de dados da organização, tendo como intermediário o *Write-Back*. Deste modo,

caso a organização queira, a mesma poderá reverter operações e também visualizar as que já foram realizadas, promovendo assim melhor segurança e persistência dos dados.

Esta solução adequa-se em geral a organizações de maiores dimensões e/ou com dados mais sensíveis. Por este motivo, as organizações sentem-se assim mais seguras pois a extensão *Write-Back* e os repetivos dados manipulados pelo utilizador, encontram-se disponibilizados em máquinas/servidores da própria organização, isto é, uma solução *on-premise*.

No entanto, existem organizações mais ágeis e com dados menos sensíveis, que preferem outro tipo de solução. Isto é, uma solução que proporcione uma simplificação no processo de gestão, em que não seja necessário instalar a aplicação nas suas próprias máquinas. Uma das soluções possíveis que proporcione essa simplificação é a utilização do *Software as a Service* (*SaaS*) [68].

Deste modo, pretende-se migrar o produto *Write-Back* com uma arquitetura *on-premise* para uma solução *SaaS*. Para tal, é necessário ter em conta alguns requisitos essenciais numa solução *SaaS*, nomeadamente, a segurança, a escalabilidade, a robustez e a arquitetura.

1.3 Objetivos e Contribuições

O objectivo desta dissertação consiste em desenvolver uma versão SaaS do *Write-Back*.

Deste modo, em primeiro lugar, é necessário desenhar um arquitectura que satisfaça os seguintes requisitos:

- ***Multi-tenant* e Segurança** - será necessário que a aplicação comprove o isolamento entre os clientes [37];
- **Escalabilidade** - capacidade de escalabilidade consoante o número de utilizadores concorrentes [47];
- **Robustez** - um dos requisitos da Xpand IT é assegurar a disponibilidade de forma a minimizar o impacto nos seus clientes. Devemos portanto ter mecanismos adicionais de alarmística, de controlo, de *fail-over*, alta disponibilidade (ativo-passivo, ativo-ativo) [54].

Com este objectivo em mente, foi desenhada uma arquitectura com base em tecnologias da ***Azure*** [56] para migrar a extensão *Write-back* de uma solução *on-premise* para uma solução *SaaS*.

Como contributo final e para ilustrar a viabilidade da arquitectura desenhada, foi desenvolvido um protótipo assente na arquitectura proposta, que transpôs as funcionalidades existentes da solução *on-premise*.

1.4 Estrutura do Documento

Este dissertação está dividida em seis capítulos.

No Capítulo 1, **Introdução**, iniciamos com o contexto e motivação para a realização desta investigação e identificamos o problema a estudar. Seguidamente são apresentados os objetivos e contribuições da investigação e é apresentada a estrutura do documento.

O Capítulo 2, **Trabalho Relacionado**, foca-se em explicar o que é o *Tableau* e o *Write-Back*, bem como tudo o que já foi implementado no mesmo. Também é explicado, tanto para o *Tableau* como para o *Write-Back on-premise*, como é feita a autenticação, a base de dados, as tecnologias utilizadas e como é realizada a instalação nas máquinas locais dos clientes.

No Capítulo 3, **Write-Back SaaS**, é contextualizado o tema *SaaS* e os requisitos do projeto são detalhadamente explicados. Também é apresentada uma possível solução para cada requisito de modo a completar os objetivos.

O Capítulo 4, **Implementação**, foca-se em descrever a implementação proposta para o problema descrito neste projeto. Os problemas e as suas respetivas soluções são aqui também explicadas.

O Capítulo 5, **Resultados**, foca-se em mostrar e explicar os resultados obtidos com a implementação realizada.

No Capítulo 6, **Conclusões**, é apresentado um resumo de tudo o que foi realizado até ao momento, bem como as conclusões e o trabalho futuro.

2

Trabalho Relacionado

Este capítulo visa descrever a solução atual do *Write-Back* na Secção 2.1. Além disso, também visa explicar o que é e como funciona o *Software as a Service* na Secção 2.2.

2.1 *Write-Back*

O *Write-Back*, como extensão do *Tableau*, permite inserir conteúdo diretamente através do *dashboard*. Existem vários casos de uso que podem ser usados no *Write-Back*, como podemos visualizar na Figura 2.1, a qual representa a *demo* do *Write-Back*. Com o *Write-Back*, não é necessário ter conhecimento em código *SQL*, a interface é intuitiva para fazer todo o tipo de configurações e operações.

Um exemplo de um caso de uso do *Write-Back*, é a atualização dos dados existentes num *dashboard*. Como podemos ver, a Figura 2.2 representa esse caso de uso onde é possível alterar os dados existentes no *dashboard*, inserindo-os no canto superior direito do painel. Depois da inserção, podemos visualizar a modificação no *dashboard* mais abaixo.

A extensão *Write-Back* apenas tem a versão *on-premise*, na qual cada instância poderá ser utilizada apenas por uma organização (os vários elementos de uma organização têm acesso à mesma instância). O *Tableau* apresenta duas formas de utilização, através da versão *on-premise* com o *Tableau Server* [87] ou através da versão *web* (SaaS) diretamente pelo *Tableau Online* [86]. O *Tableau Desktop* [85] é o *software* onde os utilizadores

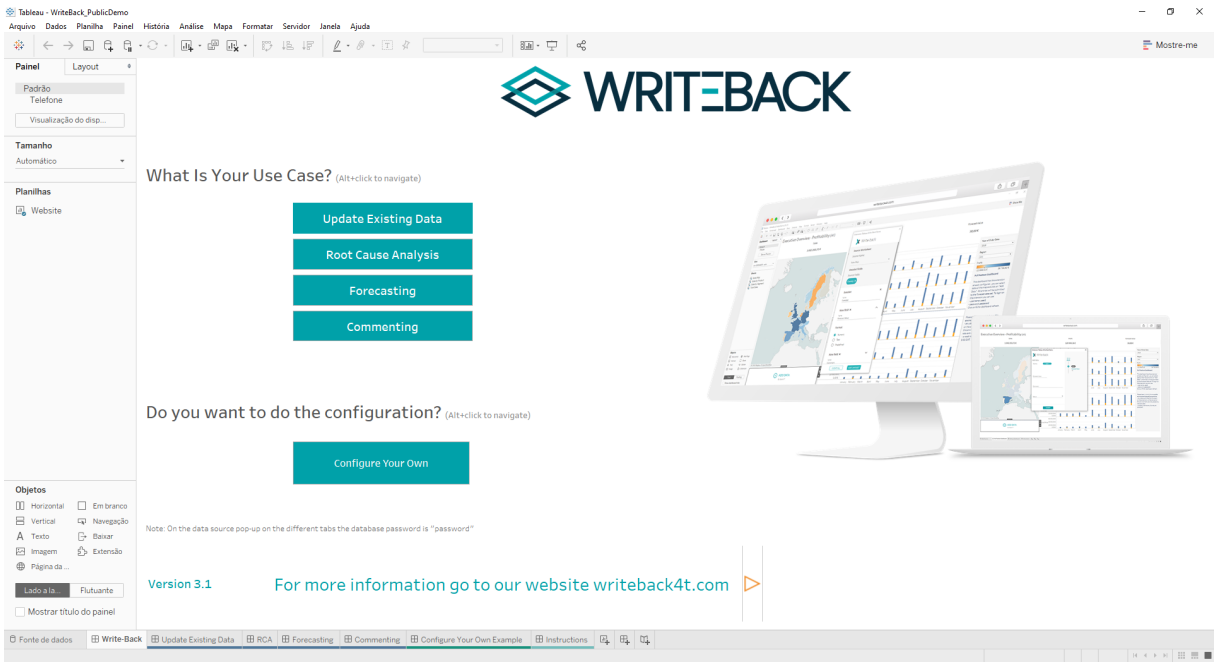

Figura 2.1: *Write-Back Demo*

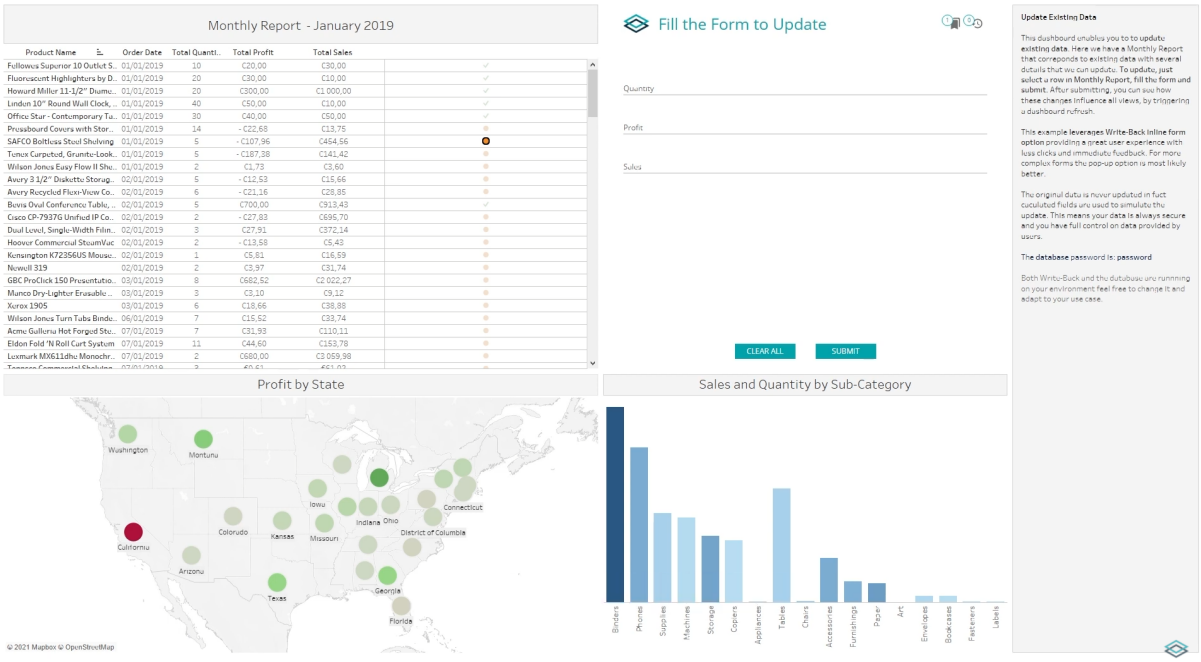

Figura 2.2: Exemplo do *Write-Back* para inserção de dados [22]

publicam os seus dados no *Tableau*, seja para o *Tableau Server*, seja para o *Tableau Online*. Com a versão *desktop*, os utilizadores poderão criar fontes de dados e também criar *dashboards* opcionalmente.

Como já foi referido na Secção 1.2, o *Write-Back* é uma extensão do *Tableau*. Portanto, de modo a utilizar o servidor *Write-Back*, uma organização terá de associar o mesmo

ao seu *Tableau*. Para tal, deve-se ter em consideração os seguintes aspetos:

- O *Write-Back* e o *Tableau*, geralmente são utilizados em máquinas separadas, cada um com o seu próprio servidor. No entanto, ambos podem ser utilizados na mesma máquina;
- O *Write-Back* informa o *Tableau Server* do caminho da localização do servidor *Write-Back*, através de um ficheiro do tipo *trex* [1]. Todos os utilizadores que queiram aceder ao servidor, devem ter esse ficheiro.

O *Write-Back* liga-se ao *Tableau* para obter os dados das visualizações no *dashboard*. Para que o *Tableau* possa receber os dados vindos do servidor *Write-Back*, o cliente tem de os conectar, para que o *Tableau* possa ter acesso aos seus dados (inicialmente, apenas é possível com a aplicação *desktop*). O *Tableau* liga-se aos dados do *Write-Back* através da criação de um *data source* [57] [70] sobre as tabelas de dados geradas pelo *Write-Back*. Isto é, um cliente cria um *data source* no *Tableau* que contém informações da localização dos dados do *Write-Back*. Estas *data sources*, criadas com a aplicação *desktop* do *Tableau*, correspondem a uma camada lógica e incluem o seguinte conteúdo:

- **Conectividade aos dados** - Estes são os dados que o cliente quer proporcionar ao *Tableau* para a sua análise e são conectados através de operações *joins*;
- **Tipo de *data source*** - Um *data source* pode ser do tipo ***extract*** ou ***live***. Nos *data sources* do tipo *extract* os dados são copiados para um ficheiro inteiro no *Tableau*, para serem rapidamente visualizados quando necessário (estes têm de ser atualizados através de um agendamento regular). Nos *data sources* do tipo *live*, os dados são obtidos consultando a fonte. Isto significa que irá sempre ter em conta a base de dados para todas as consultas. Estes terão uma menor velocidade de desempenho, mas irão ter atualizações em tempo real comparando com as do tipo *extract* [70];
- **Manipulações ou customizações de dados** - por exemplo, renomear campos ou criar fórmulas;
- **Informação de como aceder ou atualizar os dados** - por exemplo, qual o caminho para um ficheiro *Excel* onde se encontram os dados.

Com a ligação entre o *Tableau* e o *Write-Back* estabelecida, o *Tableau* vai guardar num ficheiro *XML* (ficheiro com o formato *.twb* ou *Tableau Workbook*) todas as informações

[1] Ficheiro usado pelo *Tableau* para as suas extensões

relacionadas com o que foi produzido no *Tableau*. Nestas informações estão incluídas todas as extensões usadas, tal como a extensão *Write-Back*. Esta extensão, no ficheiro *XML*, tem incluído por exemplo, o *URL* [90] da localização do servidor *Write-Back*, o *site* e as configurações do *dataset* utilizadas na sessão atual. Com esta informação guardada, é possível publicar o *XML* resultante no *Tableau Online* e, por conseguinte, com a finalização desta publicação, pode-se começar a trabalhar sobre o mesmo.

Na Figura 2.3 é possível observar a arquitetura do conjunto do *Tableau* e do *Write-Back*.

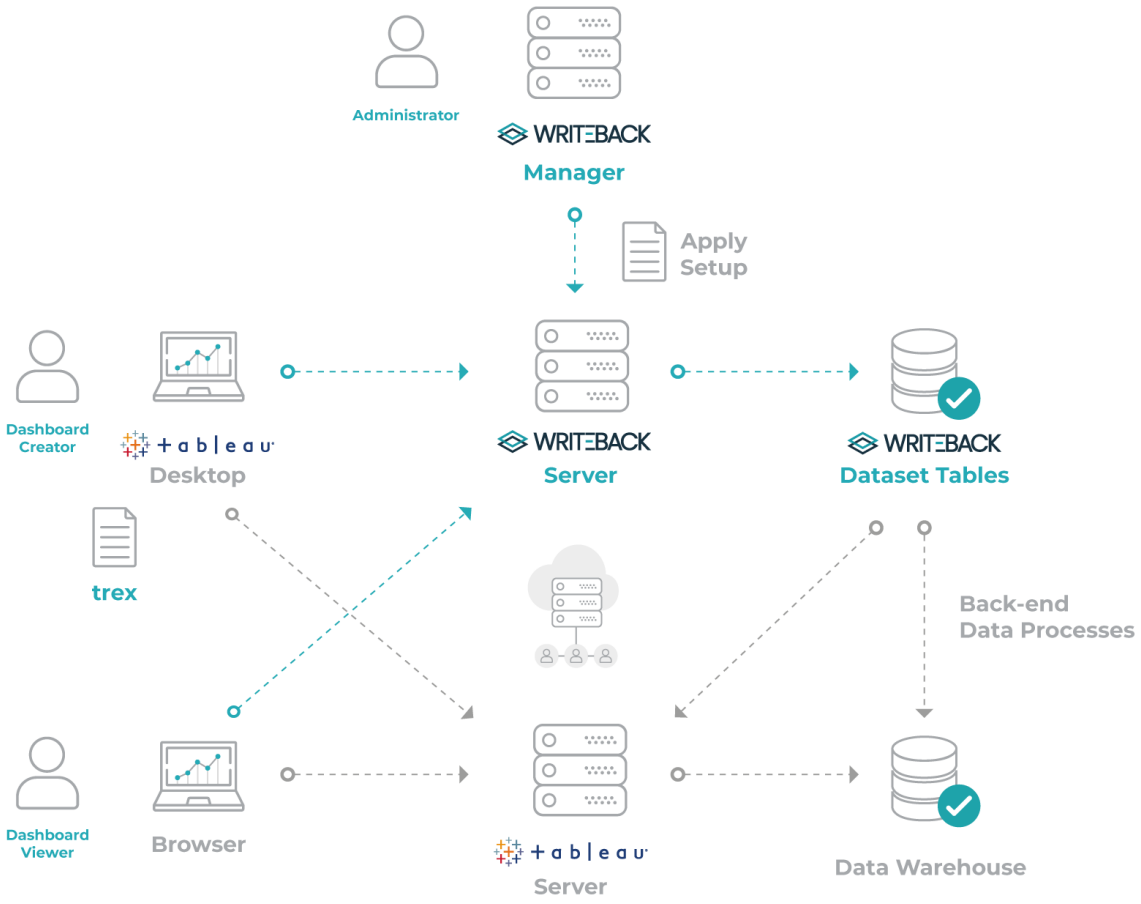

Figura 2.3: Arquitetura *Tableau* e *Write-Back* [22]

Como podemos ver na Figura 2.3, a arquitetura do *Tableau* e *Write-Back*, pode ser dividida em três componentes principais:

- **Apresentação:** na qual se pode visualizar que o *Tableau* tem duas formas de acesso, *browser* e *desktop*. Neste último, é por meio do ficheiro do tipo *trex* que se poderá aceder ao *Write-Back*;

- **Lógica de Negócio:** na qual se podem visualizar os servidores principais para cada aplicação, do *Tableau* e *Write-Back*. O servidor designado por *Write-Back Manager*, é uma interface que ajuda os administradores no processo de configuração do *Write-Back*. Aqui, pode-se gerir as configurações dos métodos de autenticação, as configurações de *sites* e as configurações da base de dados (por exemplo a sua localização). O símbolo central representa um sistema externo de autenticação, que irá ser utilizada consoante as configurações feitas no *Write-Back Manager*;

- **Acesso a Dados:** onde se podem visualizar duas bases de dados. O "*Data Warehouse*" [34] [51] representa a base de dados do cliente, onde o cliente coloca os dados que o *Tableau* utiliza como fonte para as análises. O *Write-Back Dataset Tables* representa a base de dados com o conteúdo concreto do *Write-Back*.

  Também existe o *Write-Back Metadata Tables*, que está incluído dentro do *Write-Back Server* e representa a base de dados interna do *Write-Back*, a qual guarda os metadados do mesmo. O *Tableau* tem uma base de dados interna, que não é representada nesta arquitetura.

Na figura 2.4, é possível visualizar a arquitetura do servidor *Write-Back*.

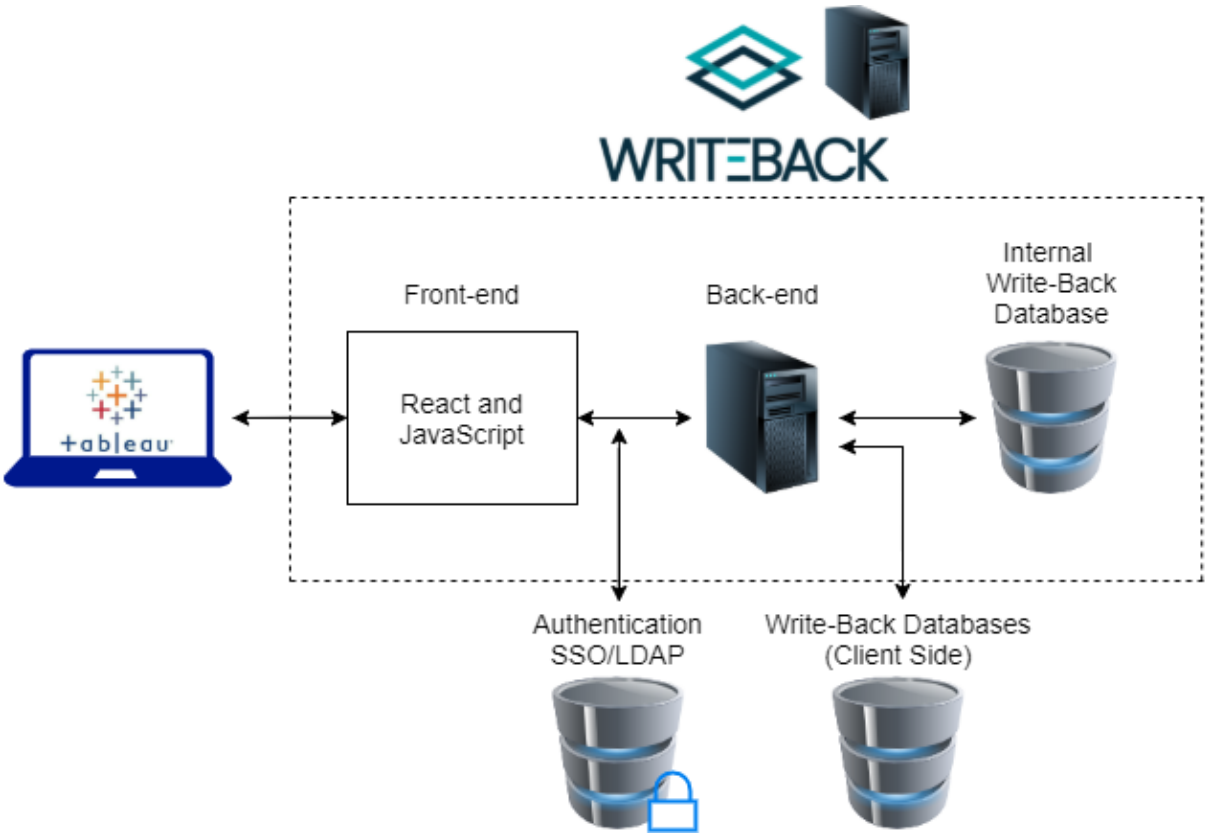

Figura 2.4: Arquitetura do servidor *Write-Back*

A forma de disponibilização do *Write-Back* consiste em instalar o *Tableau* e de seguida instalar o executável da aplicação. De seguida, com o *Write-Back Manager*, é configurado o *Write-Back*.

2.1.1 Autenticação

A autenticação [67] no *Tableau* [84] é realizada através de um conceito denominado *site*, por meio dos mecanismos *Basic*, *Kerberos* [46], *Security Support Provider Interface* (*SSPI*) [77], *Trusted Auth* [89], *Mutual SSL*, *Security Assertion Markup Language* (SAML) [76] ou *OpenID Connect* [63], sendo estes dois últimos do tipo *Single Sign On* (*SSO*). Apesar de já existir a versão *OpenID 2.0*, o *Tableau* suporta o *OpenID Connect* [64].

As identidades autenticadas pelos mecanismos descritos anteriormente, são armazenadas no repositório do *Tableau Server*. Mas para além deste, podem ser usadas outras tecnologias para armazenar a autenticação dos utilizadores, sendo estas *Lightweight Directory Access Protocol* (*LDAP*) e o *Active Directory*.

O *Tableau* pode agregar vários *sites*, isto é, vários clientes (utilizadores), sendo que cada cliente tem acesso ao seu respetivo *site*. Estes *sites* adotam a abordagem ***Mapped Domain Name***, a qual permite que os clientes escolham um nome único para o sub-domínio que representará o seu *site*, por exemplo *www.tableau.com/site1*.

Apesar de estarem todos numa mesma infraestrutura, há uma garantia de que a segregação é completa e que não há o risco de um cliente visualizar os dados de outro cliente. Essa garantia é feita com a ajuda de *cookies* para permitir uma verificação de que o cliente está corretamente autenticado no seu *site*.

O *Write-Back* disponibiliza vários métodos de autenticação [92], sendo estes os seguintes: a forma básica do *Tableau Server*, com um *username* e *password*; *Active Directory* ou *LDAP* (ambos a partir de um servidor existente do cliente); *Kerberos*; *SAML*; *OpenID Connect*. Estes dois últimos, são métodos de autenticação do tipo *SSO*, que por sua vez, caso o cliente já use este método no *Tableau Server*, este poderá configurar o *Write-Back* a utilizar estes mesmos métodos (estes métodos do *Write-Back* são separados dos do *Tableau*).

A segregação do *Write-Back* é garantida através de *sites*, tal como acontece com o *Tableau*.

2.1.2 Front-end

O *front-end* representa a parte visual da aplicação *Write-Back* (visualizar na arquitetura da Figura 2.4), isto é, toda a interação com o utilizador é realizada aqui, como as introduções de dados feitas por esses utilizadores. Este módulo é o que comunica e faz pedidos ao *back-end*, e este último escreve ou lê as bases de dados.

O *Write-Back* utiliza *Json Web Tokens* [44] de modo a guardar uma sessão. O *back-end* gera o *token* e envia o mesmo para o *front-end*, o qual, por sua vez, guarda o *token* nos cookies[2]. À medida que são feitos pedidos (*requests*), o *front-end* envia o *token* juntamente com os pedidos ao *back-end*. O *back-end* por sua vez valida o mesmo para confirmar se este *token* não se encontra expirado ou inválido.

2.1.3 Back-end

O *back-end* (visualizar na arquitetura da Figura 2.4) é a parte da extensão *Write-Back* que controla a parte da lógica, como a criação, leitura, edição e exclusão de dados nas tabelas da base de dados. É onde se estabelece a conexão com a base de dados e gere o processo de autenticação dos utilizadores.

A Figura 2.5, apresenta a arquitetura utilizada no *back-end* do *Write-Back*.

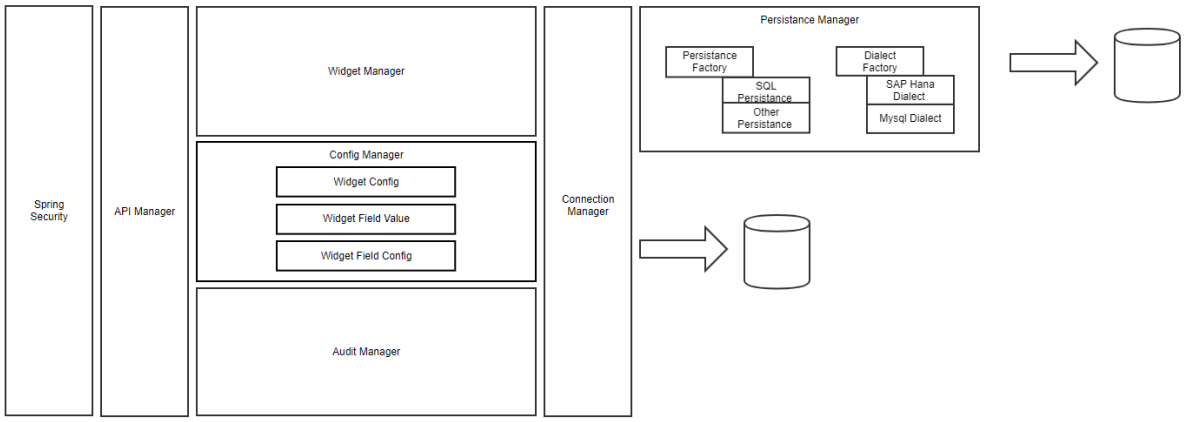

Figura 2.5: Arquitetura *Back-end* [22]

Como é possível observar na Figura 2.5, a arquitetura do *back-end* apresenta várias componentes:

[2]O *Tableau* na versão mais recente já suporta *cookies*.

- ***Spring Security*** [80] - *framework* ***Java*** [38] utilizada para a segurança da aplicação e para providenciar uma autenticação segura, de modo a assegurar a identificação do utilizador;
- ***API Manager*** - utilizado pelo ***Spring framork*** [79], no *back-end*, que estabelece a comunicação com o *front-end*. Este recebe os pedidos enviados pelo *front-end*;
- ***Widget Manager*** - utilizado para guardar e administrar a extensão das configurações, gerir os dados criados e as suas respetivas configurações;
- ***Persistence Manager*** - utilizado para administrar os dados, como por exemplo, a criação de tabelas e as inserções nas mesmas;
- ***Audit Manager***, é utilizado para gravar todas as ações realizadas pelos utilizadores;
- ***Connection Manager*** - utilizado pelo *Audit Manager*, *Persistence Manager* e *Manager* de modo a conectarem-se à base de dados.

2.1.4 Base de Dados

Existem três bases de dados a serem tratadas, sendo estas as seguintes:

- A base de dados interna do *Write-Back*, que se encontra no servidor do *Write-Back*, sendo a base de dados onde irão ser armazenados, em metadados, as ações dos clientes em relação à criação de *datasets* e à gestão dos utilizadores com esses *datasets*.
- A base de dados do *Write-Back*, que é uma base de dados que se encontra no lado do cliente e contem o próprio conteúdo que o *Write-Back* produz, como por exemplo as ações de adicionar, apagar e editar dados. Esta contém os próprios *datasets* com esse conteúdo de cada utilizador, devendo ser acessível via *Internet* para que o servidor *Write-Back* possa aceder à mesma.
- A base de dados do cliente, a qual é lida pelo *Tableau*, contém dados do cliente.

As duas últimas bases de dados, referidas nos pontos anteriores, podem estar no mesmo Sistema de Gestão de Base de Dados (SGBD) [27] (com uma divisão entre dois *schemas* diferentes) ou em duas bases de dados diferentes (por exemplo em máquinas diferentes).

O *Write-Back* regista os dados introduzidos pelo utilizador de forma manual, numa base de dados separada (base de dados do *Write-Back* do lado do cliente), à parte dos dados originais (os dados que são lidos pelo *Tableau* que se encontram na base de dados do cliente). Uma vez realizados os registos na base de dados do *Write-Back* do lado do cliente, é criado um *dataset* (representado por uma tabela) com as informações dessas alterações. Quando um *dataset* é criado, é acrescentada essa informação numa tabela designada *historical audit* (uma das tabelas de metadados para configuração e armazenamento) na base de dados interna do *Write-Back*, para controlar todas estas ações realizadas pelos utilizadores. Também existe a tabela *historical audit* na base de dados do *Write-Back*, que está no lado do cliente, para quando é criado, modificado ou apagado dados. É importante denotar que, caso o cliente queira utilizar o *Extract Transform Load* (*ETL*) [65] ou outro mecanismo, a estrutura dessas tabelas poderão ser alteradas.

Com os dados registados na base de dados do *Write-Back*, estes permitem assim verificar e reverter as modificações incorretas que poderão ser executadas na base de dados do cliente. Para além disso, também é possível monitorizar, de modo a ficar claro quem fez as ações efetuadas. Com esta abordagem é possível garantir uma maior segurança da aplicação.

Essas tabelas onde se encontram os registos do *Write-Back*, garantem que os dados originais nunca sejam alterados ou sobrepostos. No *dashboard* do *Tableau*, os dados do *Write-Back* são exibidos sobre os dados originais (em alguns casos de utilização). Isto permite simular uma substituição entre os dados do *Write-Back* e os dados originais (dados do cliente). Esta situação faz com que o cliente tenha a perceção de que há uma correção/substituição dos dados, mas na verdade é uma simulação.

Nas bases de dados do *Write-Back*, é assegurado que não haja conflito entre escrita de informação quando dois utilizadores, do mesmo *site*, estiverem a gerir a base de dados ao mesmo tempo. Isto é assegurado através do mecanismo de transações do Sistema de Gestão de Base de Dados. Estas transações são aplicadas para que as duas escritas (a escrita na base de dados de metadados e a escrita na base de dados que contém o *dataset* propriamente) sejam atómicas, portanto, é garantida a consistência nos dados.

No contexto desta dissertação, a base de dados do cliente é uma representação lógica, porque o *Tableau* aceita vários tipos de fontes de dados. Para além de bases de dados relacionais (utilizando a linguagem *SQL* [81]), este também permite utilizar dados provenientes do *Excel*, *Json* [43], *Salesforce* [75], um conjunto de *Data Marts* (os quais são um sub-conjunto de dados de um *Data Warehouse*), ou até mesmo dados gerados de forma automática a partir dos sistemas dos clientes.

2.1.5 Tecnologias

Durante a fase de desenvolvimento do *Write-Back*, o *front-end* é instanciado num servidor *web* num porto especifíco, com auxilio da tecnologia ***NPM*** [62]. Relativamente ao *back-end*, durante esta fase de desenvolvimento, é utilizado o *Spring Boot* para instanciar o mesmo.

O *front-end* no *Write-Back* é desenvolvido com as tecnologias denominadas ***React JS*** [71] e *NPM* (apenas para, no desenvolvimento, instanciar a aplicação). Também é utilizado a ***API* do *Tableau*** de modo a que o *Write-Back* possa interagir com o mesmo.

O *back-end* no *Write-Back* é desenvolvido com a linguagem de programação *Java*, com o auxílio das *frameworks* ***JDBC*** [40] (para estabelecer conexões à base de dados), ***Log4j*** [49] (para gerir os *logs* de erros e alertas, podendo estes serem dos seguintes tipos: *DEBUG*, *INFO*, *WARN*, *ERROR*, *FATAL*) e *Spring Security*.

O *Write-Back* utiliza a linguagem *SQL* para armazenar os dados enviados em vários sistemas de base de dados. As seguintes bases de dados são suportadas: *SQL Server*, *PostgreSQL* [69], *MySQL* [58], *Oracle*, *Vertica* ou ainda *Amazon Redshift*. Neste momento o sistema mais utilizado é o *PostgreSQL*.

Uma vez finalizado o *Write-Back*, é utilizado o servidor ***Tomcat*** [88] de modo a poder realizar o ***deploy*** e instanciar a mesma. Para o ambiente de desenvolvimento desta aplicação é utilizado o ***IntelliJ IDEA*** [35].

Relativamente o *Tableau Desktop*, é utilizado um *embed* de um *chromium* [18] que permite fazer a renderização de dados dos ficheiros *HTML* e *JavaScript* na aplicação. Essa informação renderizada é constituída pelo *Tableau* e *Write-Back*, o qual é inserido na página do *Tableau* por meio de uma *tag* <*iframe*>.

2.2 *Software as a Service*

Software as a Service é um modelo de distribuição de *software*, como um serviço, fundamentado na nuvem [3]. O fornecedor do serviço é o responsável por assegurar todos os servidores, configurações, conectividade e segurança necessárias para o mesmo. O cliente, necessita de pagar para poder usufruir desse serviço, o qual pode ser acedido por meio de uma aplicação *web*, não sendo necessário instalar uma aplicação na máquina do cliente [67].

A versão *SaaS* permite que várias organizações tenham acesso à mesma instância do servidor onde se encontra a aplicação *SaaS*.

2.2.1 Modelos de Serviço na Nuvem

Existem três tipos de modelos de distribuições de *software* na nuvem: *Infrastructure as a Service* (*IaaS*), *Platform as a Service* (*PaaS*) e *SaaS* [21] [78].

O modelo *IaaS* é um tipo de serviço de computação na nuvem, na qual são disponibilizados recursos de computação, armazenamento e rede. Neste modelo é possível alugar uma determinada máquina e instalar o serviço desejado. É da responsabilidade do cliente lidar com os problemas que poderão vir a surgir com o serviço, como por exemplo, falhas de conexão.

O modelo *PaaS* é um ambiente de desenvolvimento na nuvem, que ao contrário do *IaaS*, contém um sistema operativo, recursos simples para programação e gestão de base de dados (que permitem o desenvolvimento de aplicações simples).

Relativamente ao modelo *Software as a Service*, o *software* é instalado em várias máquinas, de modo a garantir a escalabilidade.

2.2.2 Arquitetura e Implementação

A arquitetura de uma aplicação *Software as a Service* é constituída por três camadas: apresentação, lógica de negócio e acesso a dados.

De modo a construir uma aplicação *Software as a Service* com sucesso [74], é necessário ter em conta seis fatores: customização, escalabilidade, arquitetura *multi-tenancy*, segurança, integração e tolerância a falhas [73].

Uma aplicação poder ser **customizada**, não estando a mesma instantaneamente pronta para uso. O cliente terá de construir a sua parte do *software* (configuração da base de dados e da sua autenticação) para que o mesmo esteja pronto. Uma customização pode ser caracterizada como manual, automática ou guiada.

A **escalabilidade** é a capacidade de um *software* poder expandir os seus recursos, lidando com um aumento da quantidade crescente de trabalho e de clientes, mantendo o seu desempenho estável.

A escalabilidade pode ser dividida em duas categorias: vertical e horizontal. A primeira, vertical, define-se por correr a aplicação numa máquina com mais memória interna, ou um disco com mais espaço ou um melhor processador, existindo portanto, um custo maior e um limite de desempenho. Enquanto que a segunda, horizontal, define-se em distribuir a aplicação, não por uma só máquina, mas por várias com a mesma configuração, sendo esta mais apropriada para uma aplicação na nuvem. Na

escalabilidade horizontal, para um balanceamento equilibrado e uma boa gestão da distribuição é utilizado um *Load Balancer* no *cluster* das réplicas da aplicação.

Uma característica muito importante de um *Software as a Service* é que, com a possibilidade de uma grande quantidade de utilizadores, o sistema deverá ter a capacidade de atender os pedidos de forma eficiente, sendo por isso necessário garantir que o mesmo seja escalável. Algumas soluções para uma arquitetura de *Software as a Service* escalável consistem em: *multi-level scalability architecture*, *tenant-awareness*, *automated data migration*, *workload support*, entre outros.

A **Arquitetura *Multi-tenancy***, possibilita numa aplicação *Software as a Service*, que cada cliente (*tenant*) inscrito no serviço seja atendido de tal forma que o mesmo se sinta como se estivesse a conectar-se a um servidor próprio. Enquanto na realidade, para apenas um servidor, estão a ser alocados vários *tenant*. Cada *tenant* pode partilhar componentes (como por exemplo, bases de dados) numa mesma máquina e, por conseguinte, tem de ser assegurado que exista privacidade entre as componentes de cada *tenant*. É necessário reforçar a segurança e escalabilidade de modo a haver *multi-tenancy*.

Existem três formas de aplicar a arquitetura *multi-tenancy*: *data model multi-tenancy* (os *tenant* partilham a mesma base de dados e os dados podem ser acedidos por um *GUID* específico), *application multi-tenancy* (todos os *tenants* partilham a mesma instância da aplicação e a base de dados) e *full multi-tenancy* (os *tenants*, para além de partilharem a instância da aplicação e a base de dados, também lhe são permitidos, se necessário, terem a sua própria variante do produto).

Existem algumas vantagens e desvantagens em aplicar *multi-tenancy*. As vantagens consistem no baixo custo de manutenção, na otimização da quantidade do uso do *hardware* e na facilidade de manutenção do *software*. Relativamente às desvantagens, estas consistem nos problemas que podem vir a surgir para um *tenant*, o que poderá afetar os outros *tenants*, uma vez que eles partilham os mesmos recursos.

A **Segurança** é um dos fatores mais desafiantes do desenvolvimento do *Software as a Service*. Enquanto que numa aplicação *on-premise*, cada cliente pode guardar os seus dados nas suas máquinas locais (estes estarão seguros, devido ao seu próprio controlo de acesso), nas aplicações *SaaS*, os clientes são dependentes da segurança fornecida pelo serviço (os clientes não têm controlo de como asseguram os seus dados). É mais difícil garantir a segurança onde há partilha de recursos, como por exemplo, em *multi-tenancy*, onde as bases de dados são partilhadas. Nestas situações, é necessário reforçar a segurança e a privacidade perante terceiros que se queiram infiltrar (utilizadores maliciosos).

A **Integração** é a capacidade de uma aplicação integrar tanto *on-premise* como em *Software as a Service*. Para tal, a aplicação necessita de integrar outros serviços de aplicações, como por exemplo, dois *SaaS* ou um *SaaS* juntamente com uma aplicação *on-premise*. As aplicações *Software as a Service* poderão providenciar *Application Programming Interface* (*API*) que suportam *batch synchronization* dos dados, para poder integrar a aplicação na nuvem com a aplicação *on-premise*.

A integração é aplicada nas três camadas de um *SaaS*. Os utilizadores vão ter diferentes interfaces, uma para a aplicação *on-premise* e outra para a aplicação *SaaS*. É importante também destacar que, o *SaaS* deverá utilizar o método de autenticação *Single Sign On*, de modo a que o utilizador aceda uma vez e possa ter acesso ao serviço sem ter de autenticar-se novamente. Pode também haver a necessidade de sincronizar ou migrar a informação de uma aplicação *SaaS* para outra ou mesmo para uma aplicação *on-premise*. Com a utilização de uma aplicação *Software as a Service* poderá facilitar a importação e exportação de dados. No entanto, o maior desafio será em estabelecer uma comunicação entre duas aplicações *SaaS*.

A **Tolerância a falhas**, permite que o sistema continue a trabalhar mesmo na ocorrência de uma falha. Uma das formas de resolver esse problema é através da utilização de réplicas, de modo a que quando haja uma falha numa das réplicas poder-se usar outra que contém um *backup* da que entrou em colapso. Assim, é possível diminuir os riscos e o impacto ao acontecer um problema.

Existem duas técnicas mais usadas para tolerância a falhas: *preemptive migration* e *software rejuvenation*.

Plataformas de serviços como a da *Google* providenciam métodos de tolerância a falhas e escalabilidade.

2.2.3 Processos de Migração de *On-premise* para *Software as a Service*

Na literatura foram encontrados vários processos semelhantes ao que está a ser feito nesta dissertação. Portanto, nesta secção irão ser ilustrados alguns exemplos de transições de *on-premise* para *SaaS*.

O primeiro exemplo é o artigo *Business application acquisition: On-premise or saas-based solutions?* [16]. Este artigo diz-nos as vantagens e desvantagens na consideração de migrar um programa de *on-premise* para *SaaS*. Em resumo, mostra que a mudança é atrativa mas que se devem ter algumas considerações. Por exemplo, uma das vantagens são os custos, dado que não haverão custos na manutenção das máquinas, nem

no custo da própria infraestrutura (isso irá ser responsabilidade do fornecedor do serviço); outra vantagem de se migrar para um modelo *SaaS* será, por exemplo, o aumento da produtividade por o mesmo permitir trabalhar-se remotamente; irão reduzir-se trabalhadores (de manutenção, por exemplo) e recursos (as máquinas que iriam hospedadar os serviços *on-premise*), isto porque os mesmos seriam do fornecedor do serviço. Também dever-se-ão ter em atenção desvantagens, como por exemplo: a customização e as configurações são mais difíceis de serem efetuadas num modelo *SaaS*, dado que num modelo *SaaS* o produto terá sempre customizações parecidas para todos os utilizadores; o serviço estará dependente de terceiros; a segurança será menor do que um serviço privado *on-premise*; entre outras.

O segundo artigo a analisar é o artigo *Design of military service framework for enabling migration to military saas cloud environment* [82]. Neste caso, o exército da República da Coreia do Sul queria melhorar as suas aplicações distribuídas e as tecnologias em *cloud*, que neste momento estavam numa metodologia de *Infrastructure as a Service*. Portanto, foi desenvolvido uma migração para *Software as a Service*. Estas aplicações militares necessitam de muitos funcionários para a manutenção dos serviços, enquanto que com uma solução *SaaS* esses custos irão diminuir e haverá uma menor redundância nas várias aplicações do exército. Para a migração num *Software as a Service*, o exército criou uma *framework* própria, a *Military Service Framework*, que irá ajudar a gerir as várias aplicações e reduzir as que são redundantes, permitindo reduzir 36% das aplicações antes existentes.

O terceiro artigo a analisar é o artigo *Migration of an on-premise application to the cloud: Experience report* [66]. Neste artigo, foram analisadas as vantagens e desvantagens para uma melhor decisão na migração de *on-premise* para *SaaS*. A análise teve por base a avaliação dos modelos de pagamento conforme o uso ou mensalmente, a escalabilidade e a capacidade de aceder aos serviços de forma virtual e remota, que são algumas das características principais de um *Software as a Service*. Uma das vantagens referida foi, por exemplo, a capacidade de alta disponibilidade em pontos geograficamente distribuídos e dinamicamente escaláveis. Mas, por outro lado, traz alguns desafios como na segurança, privacidade e performance. Também foram estudados os vários tipos de provedores de programas na nuvem (*cloud computing*) como a *Amazon Web Services* (*AWS*), a *Microsoft Azure* (aplicada nesta dissertação) e a *Google App Engine*, que têm implementações distintas. Acabou por ser decido usar a *Microsoft Azure* para um teste de migração, pelo motivo de que a mesma tem alta compatibilidade com o programa *on-premise* dos escritores do artigo. Foram feitos vários testes apesar de se chegar à conclusão de que não há uma medida única que define o quão bem uma aplicação está a desempenhar. Algumas conclusões sobre esses testes é que a aplicação na nuvem

tem mais latência que a *on-premise* (o que é normal por a mesma não estar na mesma rede). No artigo é aconselhado a efetuar o mínimo possível de transferências entre sistemas e a guardar o mínimo de informação nas sessões (considerar usar um *cache* local), por conta do aumento da latência. Também é aconselhado usar uma escalabilidade dinâmica, que muda ao longo do tempo, consoante a necessidade (à medida que os utilizadores e consequentemente os pedidos aumentam, ou a determinadas horas do dia quando há mais trabalho), o que irá diminuir os custos (que irá ser aplicado também nesta dissertação nos próximos Capítulos).

Por último, o artigo *Migration of an on-premise single-tenant enterprise application to the azure cloud: The multi-tenancy case study* [31] feito por um aluno de mestrado da universidade de Tartu na Estónia, que fez uma migração *single-tenant* de uma aplicação *on-premise* para *Software as a Service*. Ao usar a *Microsoft Azure*, foi facilitada a implementação com, por exemplo: a componente *SQL Federations*, o isolamento dos dados foi aplicado facilmente; a componente *Azure Access Control Service* para uma autenticação e autorização segura das aplicações *web*; entre outras. Seguidamente foram analisadas as arquiteturas *single-tenant* e *multi-tenant*. Diz-se que tem de ser analisado com cuidado, isto porque em *multi-tenant*, dados de vários clientes estão alocados em apenas um servidor, e isso faz com que, caso uma máquina tenha uma avaria, todos os clientes são afetados, enquanto que em *single-tenant* isso já não acontece, dado que há uma máquina para cada cliente.

3

Write-Back SaaS

Tendo em conta os objetivos inicialmente apresentados na Secção 1.3, este capítulo visa discutir aspetos importantes para migrar o *Write-Back* para uma solução *SaaS*. Nesta solução irá ser usado o *Cloud Provider* ***Microsoft Azure***. Para tal, é necessário desenhar uma arquitetura que satisfaça alguns requisitos, nomeadamente segurança, escalabilidade e robustez. Uma visão geral da arquitetura que se propõe nesta dissertação pode ser encontrada na Figura 3.1.

Esta nova versão *SaaS* pressupõe que não serão usadas as duas versões (*on-premise* e *SaaS*) ao mesmo tempo pelo mesmo utilizador. Este utilizador terá sempre de optar por usar uma ou outra versão. Isto não poderá ser feito porque os dados, caso estivessem a ser usados nas duas versões, iriam entrar em conflito entre as versões. Assim, são evitadas réplicas diferentes dos dados. Não poderão ser usadas as duas versões em simultâneo no mesmo caso de utilização (*use case*), no entanto, para casos de utilização diferentes, o mesmo já não se aplica.

Um dos requisitos para esta migração é que se quer manter uma base de código comum, para que a versão *on-premise* e a versão *SaaS* possam continuar a evoluir em conjunto, sem que seja necessário implementar duas vezes a mesma funcionalidade, uma para cada versão.

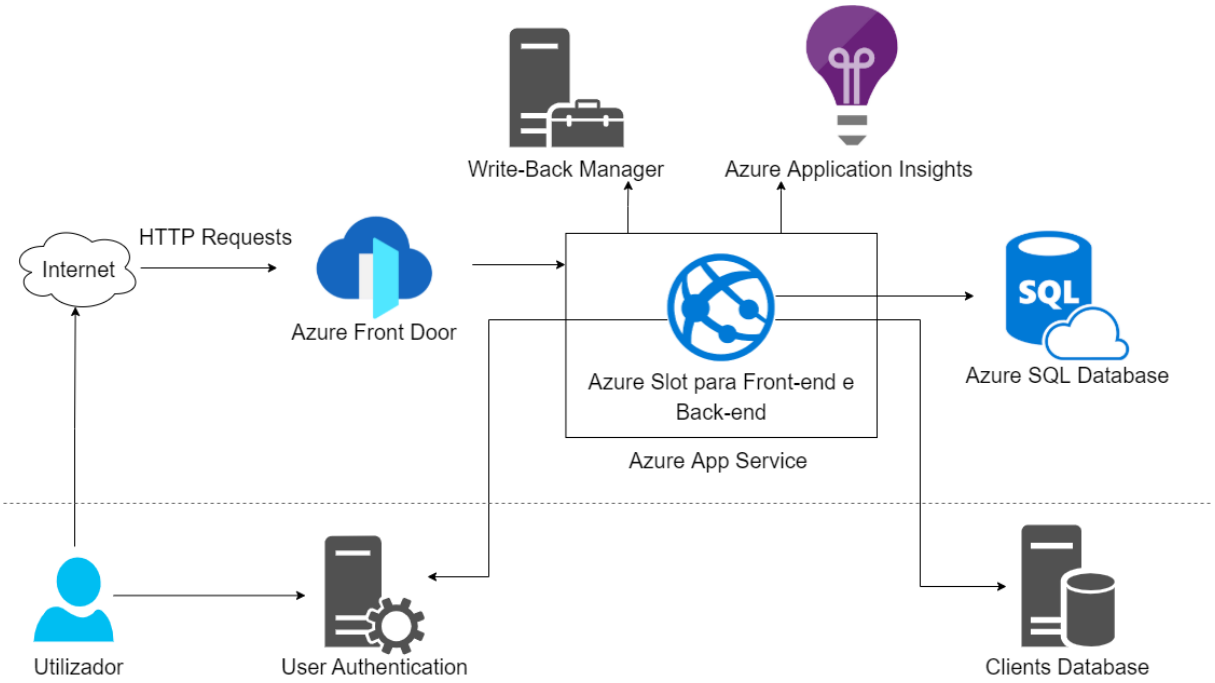

Figura 3.1: Arquitetura Write-Back SaaS

3.1 Segurança

Para realizarmos um serviço que seja seguro a eventuais ações maliciosas ou mesmo eventuais acessos a conteúdos privados por parte de outros clientes, devemos ter atenção a vários aspetos, tais como a **Autenticação**, **Aplicação** e **Transmissão** [50].

Há que salientar também que existem vários tipos de certificações para garantir a segurança de um *Software as a Service*. Estes tipos de certificações não serão aprofundados nesta dissertação pois está fora do âmbito da mesma.

3.1.1 Autenticação

Para uma melhor segurança na autenticação, não se devem ter expostas as informações de autenticação na *internet*. Para se conseguir que as informações estejam mais seguras, é crucial usar sistemas externos, isto é, fora da infraestrutura (por exemplo, de outras organizações que providenciem esse serviço).

O *Tableau* e o *Write-Back* já têm os seus próprios tipos de autenticação, como foi anteriormente referido na Secção 2.1.1, mas não será possível usar todos na nova versão *SaaS*.

Interessa-nos manter alguns dos mecanismos usados no *Write-Back on-premise*, que são os seguintes:

- **Autenticação básica do *Tableau*** - autenticação por *default* através de um *username* e uma *password*;
- ***OpenID Connect*** [59] - um método de autenticação, que usa o protocolo *OAuth 2.0*, do tipo *Single Sign On* [30]. Pode-se utilizar a conta *Google* dos clientes para a autenticação, direccionando para um *Identity Provider* (*IdP*) da própria *Google* para inserir as credenciais;
- ***SAML*** - um método de autenticação, também do tipo *Single Sign On*, que também utiliza um *Identity Provider* para gerir o estabelecimento das conexões, redirecionando também para um sistema de autenticação externo.

Estes dois últimos são sistemas externos que poderão assegurar uma melhor segurança na autenticação [60] [61].

O *Single Sign On*, é um esquema de autenticação que permite ao utilizador autenticar-se com um *ID* e *password*. A um *ID* de autenticação, poderá ser associado um ou mais *sites* do *Write-Back*. Com este esquema, uma vez autenticado, o utilizador poderá automaticamente autenticar-se sem necessidade de introduzir a informação de autenticação novamente. Este métodos de autenticação como o *OpenID Connect* e o *SAML*, usam um gestor das informações de autenticação denominado por *Identity Provider* (*IdP*). Este gestor irá receber pedidos com *tokens* de autenticação, e com esse *token*, irá dar permissão para a lista de *sites* associados à conta desse *token*.

Não será usado o método *LDAP* já usado no *Write-Back on-premise*, pelo facto de ser um método *on-premise*, facilitando ao cliente a configuração do mesmo. Caso algum cliente já tenha uma *active directory* que use o protocolo *LDAP*, será possível continuar a usar o mesmo método. Para isso, o cliente poderia, por exemplo, usar *Virtual Private Network* (*VPN*) entre a máquina do cliente e o servidor *Write-Back* ou então, o cliente poderia disponibilizar na *internet* publicamente o seu *active directory*. Um *active directory* que poderia ser utilizado seria o da *Microsoft Azure*, isto é o *Azure Active Directory* [6] que irá integrar as identidades de autenticação em diversas aplicações, incluindo as da *Microsft Azure*.

3.1.2 Aplicação

3.1.2.1 Organização *Multi-tenancy*

Uma das abordagens existentes para uma melhor segurança nas arquiteturas *SaaS* é o *multi-tenancy*. Esta abordagem permite que os ***Cloud Providers***, tal como, a *Amazon*

AWS ou a *Microsoft Azure*, possam gerir os recursos com mais eficiência. Para tal, uma instância da aplicação com as suas componentes é partilhada por vários clientes.

Como foi referido na Secção 2.1.1, o *Write-Back on-premise* já tem uma organização em *sites*, sendo esta uma organização *multi-tenancy*. Sendo assim, pode-se utilizar essa organização na versão *SaaS* do *Write-Back*. No entanto, na versão *on-premise* que foi implementada, não foi considerada a segurança dos *sites*, mais propriamente com a privacidade entre eles. Isto é, um cliente poderia ter acesso a dados de outro *site* que não o dele, pelo facto de que a aplicação *Write-Back* estaria alocada na infraestrutura do próprio cliente, proprietário do seu respetivo *site*. Os *sites*, ao encontrarem-se em máquinas separadas, já contêm um isolamento seguro na sua própria rede privada. Não houve assim a necessidade de um aprimoramento da segurança a esse nível. Para além disso, todas as alterações na base de dados são registadas, o que torna possível a monitorização e uma melhor segurança.

Na nova versão *SaaS*, vários clientes (*tenants*) irão ter os seus dados (dados esses provenientes de vários *sites*, que se encontram na base de dados interna do *Write-Back*) na mesma máquina (servidor do *Write-Back*). Portanto, agora há uma necessidade de garantir a privacidade dos *sites* de cada cliente. Esta segurança tem de ser aplicada a seguir à fase de autenticação, ou seja, após o cliente se autenticar no seu *site*. Como foi explicado na Secção 2.1.2, quando um cliente se autentica, é gerado um *token* que representa a sessão e que contém o *site* encriptado. Com a posse do *token* no fluxo principal da aplicação, e consequentemente o *site* correspondente, para uma melhor segurança na privacidade dos *sites*, o que poderá ser realizado é: para cada pedido realizado à base de dados, aplicar um filtro na *query* desse pedido através do respetivo *site*, que se encontra no *token* gerado após a autenticação. Ao aplicar o filtro, o cliente só poderá fazer pedidos ao *site* em que se autenticou.

O *back-end* não guarda qualquer informação de estado entre os pedidos, portanto não há qualquer ligação entre os pedidos, sendo necessário aplicar o filtro em todos eles, trazendo isto uma melhor segurança ao isolar os pedidos entre si.

3.1.2.2 Bases de Dados do *Write-Back*

Em relação à base de dados do *Write-Back* que está no lado do cliente, como esta não se encontra no servidor do *Write-Back*, este não tem de se preocupar com a configuração das bases de dados dos clientes. É uma responsabilidade do cliente fazer com que a base de dados esteja funcional e que a mesma esteja acessível via *internet*. Para gerir todas as conexões das bases de dados do *Write-Back* irão ser usados ficheiros no formato *Yet Another Markup Language* (*YAML*) [94], um para cada conexão.

Relativamente à base de dados interna do *Write-Back*, como é a base de dados que irá estar no servidor *Write-Back*, numa cloud da *Azure*, terá de se ter em conta a segurança do mesmo.

Geralmente é tomada uma das três estratégias para a organização das bases de dados:

- Criar uma **base de dados** separada para cada cliente - esta técnica aplica-se quando a prioridade da aplicação é o isolamento entre os utilizadores;
- Criar um ***schema*** dentro de uma base de dados para cada cliente;
- As **tabelas** da base de dados serem partilhadas por todos os clientes - esta técnica é aplicada quando a prioridade é a escalabilidade, ou seja, quando a quantidade de utilizadores da aplicação aumenta progressivamente. Neste caso há uma maior necessidade de facilitar o acesso aos dados, guardando esses mesmos dados, dos clientes, nas mesmas tabelas.

No *Write-Back* versão *SaaS*, irá ser utilizada a terceira estratégia de organização na base de dados interna do mesmo. Cada tabela desta base de dados de metadados irá ter segmentado os *sites* através de uma coluna. Esta coluna representa exatamente o *site* de cada cliente, por cada linha. Irá ser aplicada a organização *multi-tenancy* nesta base de dados como referido na Secção 3.1.2.1.

3.1.3 Transmissão

Também devemos assegurar a segurança na comunicação entre o servidor *Write-Back* e a base de dados dos clientes. Para isso devemos garantir que haja sempre uma comunicação encriptada com certificados, como por exemplo *SSL* ou *TLS* entre os dois.

Também para uma maior simplificação, a base de dados dos clientes deverá estar disponível via *internet*, para não haver a necessidade de se criar uma *VPN* para o acesso e comunicação da mesma.

3.2 Escalabilidade

Para ser assegurada a escalabilidade nos vários módulos da aplicação, irão ser utilizadas componentes da *Azure* para o efeito. Portanto, vão ser discutidas as componentes da *Azure* utilizadas para garantir a escalabilidade. O objetivo para um sistema escalável, é que este seja distribuído, ou seja, escalável horizontalmente e que não tenha

Single Point of Failure (*SPOF*). Por exemplo, uma aplicação que tenha apenas uma base de dados, caso essa mesma base de dados falhe, o sistema falha por completo. O sistema falha nestas condições porque a base de dados é um ponto único de falha (não está distribuída ou não existe um *backup*).

Os principais módulos a analisar da arquitetura para a escalabilidade são:

- ***Load Balancer*** - representa o ponto de entrada onde se iniciam todos os pedidos da aplicação. É quem distribui os pedidos para os servidores correspondentes, que irão por sua vez responder a esses pedidos;
- ***Front-end*** - representa um conjunto de recursos estáticos que, no caso do *Write-Back*, é tudo o que foi desenvolvido em *React*. Estes recursos estáticos representam conjuntos de ficheiros *JavaScript* [39], *HTML* [33], *CSS* [19], imagens, entre outros;
- ***Back-end*** - representa a lógica de negócio ou lógica de processamento da aplicação;
- **Base de dados** - representa a base de dados, neste caso, a base de dados interna de metadados do *Write-Back*.

3.2.1 *Load Balancer*

O *Load Balancer* é o ponto de entrada que irá permitir que não se aceda ao servidor aplicacional de forma direta. Isto permite controlar o encaminhamento de pedidos, *firewall* de portos, *cashing* dos recursos estáticos dos pedidos e uma transmissão segura para o exterior (utilizando os certificados de segurança *SSL* ou *TLS*, e por sua vez o protocolo *HTTPS*). Em relação à transmissão, depois de um pedido passar pelo *Load Balancer*, esse mesmo pedido irá entrar na rede protegida da *Microsoft Azure*. Portanto, as comunicações feitas dentro da aplicação não têm a necessidade de utilizar o protocolo *HTTPS*, podendo ser feitas através do protocolo *HTTP*.

A solução para a escalabilidade no *Load Balancer*, é usar o ***Azure Front Door*** [10]. O *Azure Front Door* é um ponto de entrada global e de alta escalabilidade da *Microsoft Azure*, que usa a sua *global edge network* para criar de forma rápida, segura e robusta aplicações *web* escaláveis. O *Azure Front Door* usa o protocolo *anycast* [32] [53] via *TCP* e a rede global da *Microsoft* para melhorar a conectividade global.

Algumas das funcionalidades do *Azure Front Door* são as seguintes:

- Mecanismo *Web Application Firewal (WAF)* para a segurança do mesmo;
- Métodos de roteamento para tornar mais rápidas as distribuições dos pedidos pelos vários servidores *back-end*;
- O método denominado *session affinity*, baseada em *coockies*, que tem capacidade de monitorizar o direcionamento dos pedidos de um utilizador para o mesmo *back-end*;
- O encaminhamento dos pedidos baseados nos *URL-path*, com regras de roteamento;

Uma das preocupações que se tem perante este tema é garantir que o mesmo não seja um *Single Point of Failure* (um único ponto de falha), para evitar que, caso exista alguma falha, o sistema não falhe por completo. Entretanto, o *Azure Front Door* é tolerante a falhas, isto porque o mesmo irá alocar o serviço em várias regiões, mais especificamente, toda a *Azure Region*. Com a distribuição pelo mundo inteiro, o *Azure Front Door* tem a capacidade de diminuir a latência da procura de servidores *back-end*. O *Azure Front Door* permite que dependendo de gama de *Internet Protocol* (*IP*), por exemplo de uma certa região, o pedido irá ser redirecionado para um servidor específico.

3.2.2 *Front-end*

O *front-end* é um conjunto de recursos estáticos, que representa a camada de apresentação da aplicação. Para o tema da escalabilidade, no *front-end*, o objetivo é responder o mais rápido possível aos pedidos. Devemos ter em conta que quanto mais recursos estiverem disponíveis em *cashing*, as interações entre cliente e servidor serão mais rápidas.

Como os recursos estáticos só são alterados quando houver uma nova versão (*realease*), vai haver poucas atualizações neste módulo.

Uma solução possível que assegura a escalabilidade para o *front-end* numa solução *cloud* é usar as *Azure Content Delivery Netwrok* (*CDN*) [9] [26]. As *CDN* são redes distribuídas de servidores que entregam de forma eficiente conteúdos, geralmente estáticos, para aplicações *web*. Para diminuir a latência, as mesmas armazenam os conteúdos em *cache*, em servidores *CDN* que estão perto dos utilizadores. Caso não exista um servidor perto do cliente, o mesmo poderá sentir uma latência mais alta, mas existem muitos servidores da *Azure* disponíveis para este serviço, sendo assim pouco provável o mesmo acontecer. Para além de aumentar a velocidade das respostas aos pedidos

a um *site* dinâmico, suporta comunicações com o protocolo *HTTPS*, compressão de ficheiros e filtros geográficos para, por exemplo, restringir a países o acesso das *CDN*. As *CDN* também executam a distribuição de pedidos dos utilizadores maximizando assim a disponibilidade e desempenho. Sobre o tópico de tolerância a falhas, a aplicação não é 100% tolerante a falhas, no entanto caso a aplicação falhe, os clientes poderão utilizar cópias dos recursos que foram colocados em *cache* localizadas no lado do cliente. Outra possibilidade é implementar uma funcionalidade que faça com que os clientes obtenham os recursos estáticos a partir do servidor de origem, enquanto as *CDN* ainda não estiverem disponíveis.

Ao contrário de quando a aplicação corre normalmente, ao fazer uma mudança dos recursos estáticos para uma nova *release* (versão) poderão haver problemas ao usar as *CDN*. Isto porque as *CDN* são objetos simples que apenas guardam ficheiros de forma crua (*raw*), em formato binário para leitura e escrita, como um *Binary Large Object* (*BLOB*), e poderão ocorrer inconsistências ao substituir os ficheiros. Estas inconsistências devem-se ao facto de, ao se estar a carregar os ficheiros da nova *release* para a *CDN*, poderá eventualmente um utilizador usar a aplicação enquanto ainda não foram carregados todos os dados. Caso isso aconteça, poderá haver alguns ficheiros estáticos numa versão antiga e outros na versão nova, podendo causar *bugs* para o utilizador. Por exemplo, se houver um ficheiro *JavaScript* que faça referência a outro ficheiro *JavaScript*, ao atualizar os dois ficheiros vai ser atualizado um e a seguir o outro. Se por acaso for feito um pedido entre as atualizações dos dois ficheiros, pode haver uma incoerência.

Uma solução para este problema descrito acima seria colocar todo o conteúdo estático num único ficheiro. Mas uma solução mais plausível será, em vez de usar as *Content Delivery Netwrok*, usar um ***Azure App Service*** [7]. Um *Azure App Service* é um serviço que permite implantar e escalar eficazmente aplicações *Web* e *APIs*. Com o *Azure App Service* é possível utilizar uma grande variedade de linguagens de programação, desde *Java* até *Python*, e em diferentes ambientes de execução como *Windows* ou *Linux*. Este contém uma alta disponibilidade e tal como as *CDN*, contém tolerância a falhas distribuindo o serviço por várias regiões da *Azure region*. Também como as *CDN* é um serviço *Web Application Firewal* [14] que fornece uma proteção centralizada na aplicação contra explorações e vulnerabilidades comuns.

Com uma *Azure App Service* é possível colocar os ficheiros estáticos numa *Azure Slot* e por conseguinte reduzir o tempo de inatividade, minimizando o risco de inconsistências nas actualizações da aplicação. Assim, ao colocar todos os ficheiros estáticos num *slot*, quando houver uma nova *release*, basta criar um novo *slot* diferente, colocar todos os ficheiros atualizados no novo *slot* e mudar o *slot* principal do *front-end* para

esse novo *slot*. Isto, para que todos os pedidos destinados ao *front-end* sejam redirecionados ao novo *slot*. Com este novo mecanismo de troca de *slots*, também é guardada a versão anterior para o caso de ser necessário voltar atrás com a decisão da nova versão, sendo possível voltar rapidamente. A troca de *slots* é nativo e atómico das *Azure App Service*, o que assegura um utilizador a usar uma versão da aplicação ou outra, sem haver inconsistências.

A *Azure App Service* trabalha também em *cache*, assim tornar-se-ão rápidas as respostas que incluem os ficheiros estáticos do *front-end*, tal como no caso das *CDN*.

Apesar da *Azure App Service* e das *CDN* serem igualmente rápidas nas respostas, como a *Azure App Service* tem a vantagem das *Azure Slots* como foi explicado anteriormente, irá ser utilizado uma *Azure App Service* para os conteúdos do *front-end*.

3.2.3 *Back-end*

Como foi referido no Capitulo 2, no *back-end*, na solução atual temos uma *API* em *Java*, onde contém um *token* de autenticação, ou seja, não é necessário existir sessão para diferentes utilizadores, tratam-se todos os pedidos na mesma sessão. Portanto os pedidos são tratados da mesma forma individual e sem relação.

Existem três hipóteses plausíveis para assegurar a escalabilidade num cenário *SaaS* onde todos os clientes acedem ao mesmo serviço [8].

Uma hipótese seria abdicar da arquitetura atual (Figura 2.4), partindo de uma arquitetura monolítica com uma *web aplication* e migrar para uma lógica de **micro-serviços** [48], onde funções se executam umas às outras encadeadas. Uma solução para a *Azure Cloud* seria, por exemplo, utilizar os micro-serviços da *Azure* [55]. Com esta solução, ter-se-ia de ter um micro-serviço responsável por escrever os dados que o cliente enviou num pedido e um segundo micro-serviço, que seria chamado a seguir, para validar o pedido e assim sucessivamente. Segmentar-se-iam as funções da aplicação em componentes independentes, os micro-serviços. Isto requereria modificar bastante a arquitetura da aplicação, o que poderá não ser o ideal visto que não vai manter a lógica da aplicação *on-premise*, conforme o requisito referido no início do Capítulo 3. Portanto, o ideal será que as duas versões funcionem com a mesma lógica aplicacional. A lógica de micro-serviços, é uma abordagem moderna mas não a mais eficaz, em relação ao que temos na solução atual, a *on-premise*.

A segunda hipótese seria manter a arquitetura atual (Figura 2.4), aplicando algumas adaptações. Para isto, ir-se-ia alocar numa *Web Application* a aplicação e alterar algum código caso fosse necessário (o que vai ser analisado no Capitulo 4). Pode-se usar

tal como no *front-end*, uma *Azure App Service*, um serviço dinâmico com *auto-scaling*. Para assegurar a escalabilidade ir-se-ia distribuir a aplicação, ou seja, em vez de se ter um servidor *web* com a aplicação alocada, ir-se-iam ter vários servidores distribuídos. Portanto, caso seja necessário mais processamento (caso o serviço passe de um certo limite), este poderá arrancar automaticamente (com o mecanismo das *Azure App Service*) mais instâncias iguais à inicial e distribui-las automaticamente com balanceamento de carga.

A *Azure App Service* tem a capacidade de instalar um *Java Archive* (*JAR*) na mesma, ou seja, pode-se instalar diretamente na *Azure App Service* o *back-end* através do seu *JAR*. Ao arrancar este serviço, o mesmo dá a escolher qual a tecnologia a usar nesse serviço *web*, por exemplo *Java* com *Tomcat* ou mesmo *GlassFish* [29]. Também é possível alterar as configurações do *Tomcat* neste serviço, idêntico à alteração do ficheiro *web.xml* do *Tomcat*. Com as *Azure App Service*, a arquitetura será simplificada comparado com o uso dos micro-serviços da *Azure*.

Outra hipótese para alocar a aplicação numa *cloud* seria usar *containers*. Estes *containers* trazem uma agilidade para o *deploy* da aplicação. Na *Microsoft Azure* pode-se instanciar um *container* sobre um *Azure App Service*. Para a organização de várias instâncias de *container*, poderá ser usado o ***Azure Kubernets Services*** (*AKS*) [11] que facilita e aumenta a velocidade do *deploy* das aplicações. Mas não é necessário usar nem *containers* nem *Kubernets* ao início de um projeto, isto porque *Azure Kubernets Services* são usados para projetos de grandes dimensões com milhares de *containers* e milhares ou milhões de utilizadores. Só seria necessário usar *Azure Kubernets Services* quando os custos da *Azure App Service* se tornassem muito elevados. Os custos das *Azure App Service* são pagos por número de instâncias, enquanto que os *Azure Kubernets Services*, podem-se pagar por poder computacional das máquinas. O *Azure App Service* consegue criar novas instâncias, gerir as mesmas e fazer balanceamento, como os *AKS*, o que é mais um motivo para não haver necessidade de usar os *AKS*.

Sendo assim, para a versão *SaaS* do *Write-Back*, haverá uma *Azure App Service*, com um *slot* para o *front-end* e *back-end*.

3.2.4 Base de Dados

As duas bases de dados a serem tratadas é a base de dados interna do *Write-Back*, de metadados, e as bases de dados do *Write-Back* do lado do cliente, uma para cada cliente, com os dados do *Write-Back*.

Na solução *on-premise* é usado *SQL* para gerir a base de dados. Uma hipótese também

plausível é usar *NoSQL*, onde através de uma interação via *API* ter-se-ia menor latência, por exemplo, por conta de se ter menos restrições nas verificações das transações.

A solução adotada para a base de dados interna do *Write-Back SaaS* é a ***Azure SQL Server*** [13]. Este serviço tem a capacidade de gerir as bases de dados com alta disponibilidade [17], escalabilidade e segurança a um preço flexível, sendo mais vantajoso comparando com uma base de dados *on-premise* [72]. A mesma é capaz de partilhar a base de dados com várias instâncias de servidores centralizados, aplicando uma escalabilidade horizontal (*horizontal scaling*) com o mecanismo *Hyperscale* [12]. Também contém um controlo flexível de configuração no seu respetivo portal.

Para além das características do *Azure SQL Server*, que asseguram o controlo do crescimento da quantidade de utilizadores e da escalabilidade, como já foi referido na Secção 2.1.4, é assegurada a persistência dos dados, isto por conta das transações feitas na base de dados.

3.3 Robustez

Os serviços *Azure* que vão ser usados para o *Write-Back SaaS* e que foram mencionadas na Secção 3.2, são serviços altamente disponíveis, assegurando a robustez do *Software as a Service*.

Apesar da *Azure App Service* ser usada como um serviço *multi-tenancy*, a mesma irá ter várias instâncias para resolver a questão da tolerância de falhas. As instâncias, ao estarem geograficamente situadas de forma estrategica, levam a que a robustez da aplicação seja assegurada.

Um dos maiores problemas da *Azure App Service* é a escrita para disco, por exemplo de ficheiros temporários. Esta escrita é lenta, o que poderia levar a dificuldades na robustez da aplicação. Mas no caso deste projeto não haverá problemas com a escrita de dados no sistema de ficheiros do servidor, isto porque só é usada a escrita para *logs* (*logs* relacionados com o processo de registo de eventos relevantes para o sistema) para a base de dados. Não existe qualquer escrita para a sessão porque a mesma é feita através do *token* de autenticação (melhor explicado na Secção 2.1.2 e 3.1.1). Sendo assim, a escrita no sistema de ficheiros não é um problema.

Para os ***logs* dos dados**, está a ser usado na versão *on-premise* o *Log4j* (referido na Secção 2.1.5), que permite que a gestão dos *logs* seja de forma centralizada, sendo que atualmente os *logs* são enviados para o *Tomcat*. Não irá ser necessário alterar o código da aplicação atual neste ponto, bastando apenas configurar o *Log4j* para enviar todos

registos dos *logs* para o *stdout* da *Azure App Service* em vez de ser para um ficheiro do *Tomcat*. O *stdout* é o canal de comunicação de saída a nível do sistema operativo de uma máquina. A *Azure* contém um serviço que permite fazer *stream* direto dos *logs* para o *stdout* da *web aplication*. Este serviço é o ***Application Insights*** [5] que está integrado na *Azure App Service*. Com o *Application Insights* é possível redirecionar os *logs* de todas as instâncias das aplicações *web* centralizadas num só lugar.

Portanto o risco de haver um problema é minimizado por conta de haver uma monitorização feita através do *Application Insights* da *Azure*.

4

Implementação

Este capítulo contém a explicação de como foi implementado o *Write-Back SaaS*. Irá explicar as suas componentes da *azure* assim como o que foi implementado em código para o melhor funcionamento do mesmo.

4.1 Configuração na Versão *On-premise*

O primeiro passo feito foi instalar o *Tableau* e o *Write-Back* na própria máquina local, para se poder ter uma versão em que se possa executar e analisar o funcionamento da aplicação. Com esta versão funcional e sem erros, podemos avançar para a versão *SaaS* do mesmo.

Para isso foi criado um novo perfil *maven* no ficheiro *pom.xml* da raiz do projeto, para que se possam ter as preferências desejadas.

Com este perfil criado, de seguida criou-se um ficheiro de configurações *YAML* para a conexão à base de dados, as credenciais de autenticação e o domínio usado para o servidor.

4.2 Componentes *Microsoft Azure*

Depois das configurações criadas, foi iniciada a criação das componentes da *azure*. Nesta secção vai ser explicado em detalhes a criação das mesmas.

4.2.1 *Azure App Service*

Começamos com a ***Azure App Service***, a *web app* que irá conter a aplicação *Write-Back*. Esta *Azure App Service* está alocada inicialmente em França que é o servidor da *azure* mais próximo que temos no momento, como se pode ver na Figura 4.1. Para fazer *deploy* da aplicação na *Azure App Service*, foi usado um *plugin maven* para a *azure*. Ao autenticar esse plugin ao *Azure App Service*, através de *maven*, foi possível aplicar o *deploy* da aplicação com o ficheiro *WAR*.

Figura 4.1: Mapa de servidores da *azure*

O *deploy* pode ser feito da seguinte maneira:

```
1 az login
2 mvn com.microsoft.azure:azure-webapp-maven-plugin:1.12.0:config
3 mvn package azure-webapp:deploy
```

Listagem 4.1: Código para fazer *deploy* da aplicação

Mais tarde, foi usado o *git* para podermos alocar o repositório do projeto na máquina hospedeira do *Azure App Service*. Assim é possível fazer o *deploy* diretamente a partir da máquina da *azure* através de *SSH*.

Este alocamento pode ser feito da seguinte maneira:

```
1 git remote add azure https://dggo@writebacksaaswebapp.scm.azurewebsites.net
    /writebacksaaswebapp.git
2 git push azure master
```

Listagem 4.2: Código para o alocamento do repositório *git* na *Azure App Service*

4.2.2 *Azure SQL Server*

O passo seguinte, depois de ter sido alocado o *Azure App Service*, foi criar um servidor de base de dados com a componente ***Azure SQL Server***. Este servidor contém as bases de dados necessárias para a aplicação *SaaS*.

Foi necessário criar um novo *SQL Login*, para não ser sempre usado o *login* de administrador. Poderá haver assim uma melhor segurança ao não usar sempre o *login* de administrador, diminuindo as hipóteses de poderem identificar as credenciais do mesmo. Foi criado um *SQL Login* para os *SQL Users*, que acedem às bases de dados dos clientes, ou outro *SQL Login* para os *SQL Users*, que irão aceder à base de dados interna do *Write-Back*. Estes *SQL Logins* foram criados através da base de dados *Master*, acedida pelo *SQL User* de administrador, ambos criados por *default* pelo servidor de base de dados.

Um código de exemplo utilizado para criar os logins é o seguinte.

```
1 CREATE LOGIN writebacksaaswebapp WITH PASSWORD = '################';
2 CREATE LOGIN writeback_clientdb WITH PASSWORD = '################';
```

Listagem 4.3: Código *SQL* para os *logins*

Com os *SQL Logins* criados, foram criadas as bases de dados. Inicialmente foi criada a base de dados interna do *Write-Back* com a componente ***Azure SQL database***. Com o uso das *elastic pools* [23], esta é elástica, isto é, tem a possibilidade de gerir grupos de bases de dados com uma alta escalabilidade. Também possibilita uma replicação geográfica por várias regiões do mundo [2]. Para uma primeira experiência, foi usada a versão básica da base de dados com menos memória e processamento, para ter menos custos. Foi criado o *SQL User* (associado ao *SQL Login* correspondente) para poder aceder a essa mesma base de dados. Foram dadas as permissões de criação de tabelas e escritas nas mesmas a este *user*. Para além do *user*, também foi criado um *schema*, próprio para as tabelas dos *metadados* do *Write-Back*.

A seguir é possível visualizar um exemplo de código para criação do *user*, da permissão de criação de tabelas e do *schema* para a base de dados interna do *Write-Back*.

```
1 CREATE USER writebacksaaswebapp_user FOR LOGIN writebacksaaswebapp;
2 GRANT CREATE TABLE TO writebacksaaswebapp_user;
3 CREATE SCHEMA writeback AUTHORIZATION writebacksaaswebapp_user;
```

Listagem 4.4: Código *SQL* para o *user* e *schema* para a base de dados interna

Seguidamente, criou-se outra base de dados. Esta será uma base de dados de teste para simular uma base de dados de um cliente. Também foram criados dois *SQL Users* para esta base de dados, associados aos dois *SQL Logins* criados. Um dos *users* para simular o acesso do próprio *Write-Back* à base de dados do cliente, e outro *user* para simular o próprio cliente. Também foi criado um *schema* para esta base de dados, como na anterior.

Para esta base de dados foi usado como exemplo o seguinte código.

```
1 CREATE USER writebacksaaswebapp_user FOR LOGIN writebacksaaswebapp;
2 GRANT CREATE TABLE TO writebacksaaswebapp_user;
3 CREATE USER writeback_clientdb_user FOR LOGIN writeback_clientdb;
4 CREATE SCHEMA writeback AUTHORIZATION writebacksaaswebapp_user;
5 ALTER ROLE db_datareader ADD MEMBER writeback_clientdb_user;
```

Listagem 4.5: Código *SQL* para o *user* e *schema* da base de dados do cliente

4.2.3 *Azure Front Door*

Já com as duas componentes principais a funcionar (a aplicação e a base de dados), foi criada a ***Azure Front Door*** que irá proteger as comunicações feitas à *Azure App Service*.

O passo seguinte foi criar o ***Azure Application Insights***, que automáticamente dá para associar à *Azure App Service*. Assim, o *application insights* vai ler automáticamente o *stdout* das *web applications*, isto apenas se verifica, ao configurar as aplicações para que o *log4j* reencaminhe os *logs* para o *stdout*.

Para testar se o *application insights* está a funcionar corretamente, foi removido o acesso do *app service* à base de dados interna através da *firewall*, para criar um erro propositado. Assim, quando o *app service* tentou comunicar com a base de dados, surgiu um erro que o *application insights* detetou.

4.3 Preocupações com o Multi-tenancy

Existem alguns pontos a serem tratados na implementação do *Write-Back SaaS* em relação à organização *multi-tenancy* dos clientes. Alguns desses tópicos são: a possibilidade

de gerir (adicionar e remover) clientes (*tenants*) de forma dinâmica à aplicação; otimização dos *drivers* e das *queries* das bases de dados; a segregação total dos clientes, isto é, haver uma separação segura dos dados dos clientes.

4.3.1 Gestão dos *tenants* de forma dinâmica

No servidor *SaaS* do *Write-Back* existem vários ficheiros do tipo *YAML*, um para cada *site* com informações do mesmo.

Na versão *on-premise* por cada pedido, é também realizada uma leitura ao ficheiro *YAML* correspondente ao *site* desse pedido, para que se possa, por exemplo, obter as configurações para a conexão à base de dados desse cliente. Para uma versão *SaaS* estas operações todas devem ser otimizadas, isto é, sem estar sempre a ler os ficheiros *YAML* por cada pedido.

Para isso, foi implementado um mecanismo onde são lidas as informações dos *sites* em memória e não diretamente dos ficheiros *YAML*. Isto faz com que melhore a *performance*, reduzindo as várias operações de leitura a ficheiros.

Haveria a hipótese de usar uma base de dados para guardar estas informações em vez dos ficheiros *YAML*, mas esta não foi criada porque os ficheiros *YAML* são mais fáceis de ler e dão uma maior flexibilidade para configurações distintas numa aplicação *SaaS* [45].

Para se poder gerir as informações dos clientes (*tenants*), foi aproveitada a otimização dos ficheiros *YAML* explicada no parágrafo anterior. Com esta otimização serão mais rápida e facilmente adicionadas e removidas as configurações dos *sites* de um cliente.

Para a implementação da gestão dos *tenants*, foi usada uma *Thread* para verificar se houve uma adição ou remoção de um ficheiro *YAML* de um *site*.

Também foi usada uma classe estática para que se pudesse guardar as informações do ficheiro *YAML* estaticamente. Assim é possível aceder, através dos atributos da própria classe, a essas informações.

4.3.1.1 Cenário exemplo da gestão de *tenants*

Um cenário possível que um cliente poderá presenciar é, por exemplo, quando é ligado o *Write-Back SaaS*, este irá guardar em memória todos os *sites* já registados no momento, através dos ficheiros *YAML*. Caso o cliente queira adicionar um novo *site* ao *Write-Back*, o administrador do mesmo tem de criar um ficheiro *YAML*, colocando-o na

diretoria correta. Ao adicionar o ficheiro, o *Write-Back* irá adicionar automaticamente as informações do *YAML* à memória que contem as informações do *site*.

Caso o cliente queira modificar ou apagar um *site* (tendo que o administrador do *Write-Back* editar ou apagar o ficheiro *YAML* correspondente), o *Write-Back*, com o auxilio de uma *Thread*, que é instanciada ao instanciar a aplicação do *Write-Back SaaS*, irá automaticamente detetar que houve uma mudança ou o desaparecimento de um ficheiro e, por sua vez, alterar as informações que estão em memória sobre os *sites*.

4.3.2 Otimização dos *drivers* e das *queries* das bases de dados

Na versão *on-premise* do *Write-Back*, o cliente instala o *Write-Back* na sua própria máquina e configura a base de dados do mesmo. Assim sendo, o cliente irá escolher o *driver* para a base de dados que necessita e instalar o mesmo.

Na versão *SaaS* do *Write-Back*, iriam ter de existir vários *drivers* para as várias base de dados dos clientes no mesmo servidor, o que não é a melhor solução. Uma possibilidade para solucionar este problema seria permitir que o cliente enviasse um *driver* correspondente à base de dados utilizada por si. Mas essa não é uma opção viável porque, ao permitir que o cliente enviasse um ficheiro *JAR* (*driver*), faria com que houvesse a possibilidade de um cliente malicioso poder modificar esse ficheiro *JAR* para prejudicar o servidor *Write-Back*. Sendo assim, essa não será uma solução viável.

Portanto, a melhor solução para este problema, será restringir as bases de dados a um conjunto limitado, sendo estas, *MySQL*, *PostgreSQL*, *Snowflake*, *Redshift*, *Oracle*, *SQL Server*, *Redshift*, *Vertica* e *SAP Hana*, nas suas versões mais actualizadas (correspondentes aos drivers previamente escolhidos). Esta otimização dos *drivers* das bases de dados, irá facilitar a configuração destas, mais especificamente na escolha da versão da base de dados e no *driver* utilizado.

No *Write-Back* é guardado, num ficheiro *properties*, todas as *queries*, de todos os tipos de base de dados, que vão ser pedidas. Foi aplicada aqui uma otimização destas *queries* guardadas, sendo que esta otimização é dada por uma matriz onde é introduzido um tipo de base de dados e devolvido outro tipo. Ao receber um tipo de base de dados como *input*, caso o formato textual da *query* dessa base de dados de *input* seja igual a outras bases de dados, apenas vai ser obtida do ficheiro *properties* uma dessas base de dados de formato textual igual. Por exemplo, as *queries* de uma base de dados *MySQL* são textualmente iguais às *queries* de uma base de dados *PostgreSQL*. Portanto, sempre que for pedida uma *querie* de uma dessas duas bases de dados, a matriz irá retornar apenas uma, como por exemplo, apenas *MySQL*.

Com esta optimização das *queries*, é possível, num trabalho futuro, haver uma simplificação destas mesmas *queries*. Para isso, seria necessário retirar as *queries* que correspondem às bases de dados com formato de texto igual a outras bases de dados, explicadas anteriormente.

Também foi aplicada uma otimização de leitura deste ficheiro *properties*, colocando num atributo estático todas as *queries* necessárias em memória, tal como foi feito para os ficheiros *YAML*.

Combinando esta simplificação (otimização das *queries*) com a otimização dos *drivers* e o mecanismo de otimização na leitura e gestão em memória dos ficheiros *YAML*, irá haver um aumento na eficiência nas respostas dos pedidos, tanto de *queries* como para as informações dos *sites* que se encontram nos ficheiros *YAML*.

4.3.3 Segregação total

Houve a necessidade de uma maior segurança para a aplicação das *queries* nos *sites* dos clientes (uma segregação total entre os clientes). Isto deve-se ao facto de não haver nenhuma confirmação que um cliente não irá usar as *queries* para alterar alguma base de dados de outro cliente (noutro *site*). Portanto, foi aplicado um filtro através dos *sites* nas *queries* das base de dados, numa ferramenta denominada por *Query Manager*. Assim, quando um utilizador (pertencente a um cliente) realizar uma ação, o *Write-Back* aplicará uma *querie* correspondente e a mesma irá ter um filtro com o comando *SQL WHERE* pelo *site*, este *site* obtido através do *token* que o utilizador autenticou.

Nem todas as *queries* serão filtradas pelo *site*. As que não vão ser filtradas serão: as *queries* que apenas verificam a existência de alguma coluna ou tabela com o comando *SQL SELECT*; as *queries* que apenas adicionam uma coluna a uma tabela com o comando *SQL ALTER TABLE*; as *queries* de criação de tabelas com o comando *CREATE TABLE*.

Na maioria das *queries*, será aplicado um filtro feito em código adicionando automaticamente o comando *SQL WHERE*, com o *site* filtrado, no final da *query*.

Em relação às *queries* onde se adiciona uma linha de dados, com o comando *SQL INSERT*, um dos parâmetros a serem inseridos será o próprio *site*. Neste tipo de *queries*, o filtro também é aplicado automaticamente como no caso anterior. Também nas *queries* do tipo *UPDATE*, onde se faz um *SET* do *site*, aplica-se o mesmo filtro.

Com este mecanismo finalizado, o *Connection Manager* (a classe *java* que controla todas as conexões às bases de dados) garante que sempre que haja um pedido, a classe consulta o *site* (que se encontra no *token* de autenticação) e dará sempre a conexão certa,

devolvendo o filtro aplicado pelo *Query Manager*. Sendo assim, não há ligações diretas às bases de dados através de outros lugares, apenas no *Connection Manager* devolvendo as conexões já com as restrições necessárias. Estas restrições garantem que os clientes só irão conectar-se às bases de dados correspondentes aos seus *sites*. Assim a segurança na arquitetura *multi-tenancy* fica reforçada, tanto por serem feitos estes filtros restringindo o uso dos *sites* dos clientes, como também pelas conexões estarem centralizadas sempre que se faça um pedido. Caso se implemente alguma nova funcionalidade na aplicação, o programador não terá de se preocupar com a segurança e a segregação entre as conexões das base de dados dos clientes, isto porque está tudo centralizado no *Connection Manager* que usa o *Query Manager*.

O padrão de desenho aplicado no código está a forçar o uso desta restrição que garante que as conexões são feitas através do *site* certo. Mesmo que um programador se esqueça de colocar na *query* a restrição do *site*, não será um problema, porque o *Query Manager* é que vai tratar do mesmo.

5

Resultados

Este capítulo contém os resultados dos testes de performance aplicados no *Write-Back SaaS*. Também foram analisadas algumas formas de realizar esses mesmos testes.

5.1 Testes de Desempenho

Os testes de desempenho [41] avaliam um programa e as componentes de *hardware* do servidor que está alocado e se as mesmas estão aptas para suportar a quantidade de utilizadores que têm no momento.

Há 3 requisitos que estão associados aos testes de desempenho numa aplicação *SaaS*, que são os seguintes: escalabilidade, elasticidade e eficiência [4].

5.2 *TabJolt*

O *Tabjolt* [28] [36] é uma ferramenta de testes especificamente desenhada para o *Tableau* que permite automaticamente carregar, executar e interagir com o *Tableau*. O *Tabjolt* também é capaz de calcular a taxa de transferência, o tempo médio de resposta, entre outros cálculos. Tem ainda outras vantagens, como por exemplo, poder calcular quais as configurações para alocar um servidor de *Tableau* dependendo da capacidade do servidor e verficar se o *hardware* é suficiente para alocar um *Tableau Server*.

O *Tabjolt* usa *JMeter* como base e consegue executar grandes cargas de pedidos ao *Tableau* e reunir métricas de sistema e infraestrutura, como as métricas *Java Management Extensions* (*JMX*) [15].

Não é recomendado usar o *Tabjolt* sem um vasto conhecimento da arquitetura do *Tableau* e pode não reproduzir exatamente a interação de um utilizador [24].

5.3 *JMeter*

O *JMeter* [42] é um programa *open-source* em *Java* desenvolvido para executar testes de desempenho, inicialmente usado para testes em aplicações *web*, mas poderá exercer também outras funções [25].

Geralmente são aplicados 2 tipos de testes no *JMeter*, o *Load Testing* e o *Stress Testing*. No primeiro é feito um modelo de teste onde se simulam vários utilizadores em concorrência. O segundo é um teste que leva ao limite da capacidade máxima de carga, onde são simulados vários utilizadores até ao limite do servidor, ao ponto do servidor começar a responder de forma lenta e com certos erros.

Para realizar um módulo de testes, o *JMeter* dividide-se em *Thread Groups*. Estes *Thread Groups* são grupos de *Threads*, em que em cada *Thread* definem-se várias funções que esse grupo de *Threads* vai realizar. Essas funções correspondem a pedidos a um servidor, por exemplo com uma arquitetura do tipo *REST*, sendo que estes pedidos podem ser do tipo *HTTP request*, *File Transfer Protocol* (*FTP*) *request*, *JDBC requests*, entre outros.

O *JMeter* também permite fazer gráficos, como o da Figura 5.1. Essa imagem corresponde a um teste feito com o *JMeter* ao servidor do *Write-Back SaaS* com 200 *Threads*. Nesse mesmo gráfico podemos visualizar:

- Pontos **pretos**: que corresponde ao número total de pedidos enviados;
- Linha **azul**: que corresponde à média atual de todas as amostras enviadas;
- Linha **vermelha**: que corresponde ao desvio padrão atual;
- Linha **verde**: que corresponde à taxa de transferência, isto é, o número de pedidos por minuto que o servidor conseguiu tratar.

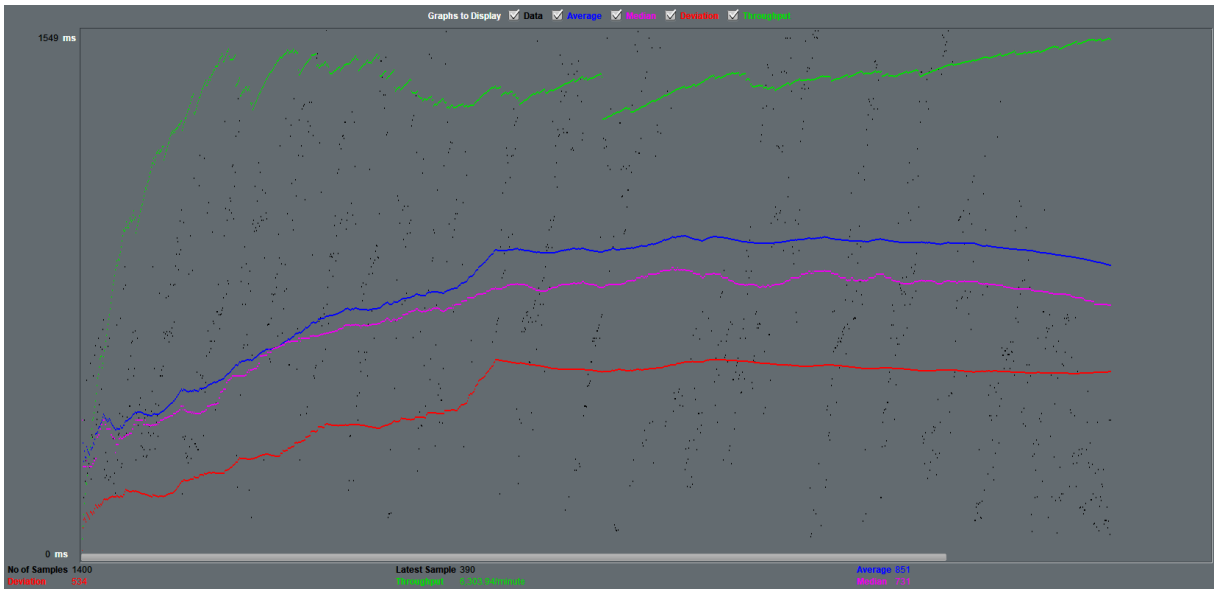

Figura 5.1: Diagrama temporal de um teste de desempenho com o *JMeter* com 200 *Threads*

Podemos dizer que quanto maior for a taxa de transferência, maior a capacidade do servidor em suportar uma grande quantidade de pedidos e quanto menor o desvio padrão, melhor o desempenho do servidor.

5.3.1 Testes de desempenho com o *JMeter*

Foi criado um teste de desempenho em *JMeter* para simular vários utilizadores a executarem certas funções do *Write-Back*, descritas a seguir. Estes testes têm foco no requisito da eficiência da aplicação.

Estes testes também poderão ser utilizados, não só para a versão *SaaS*, como também para fazer testes na versão *on-premise*. Para isso, basta apenas mudar o domínio de destino dos testes, para cada pedido feito, para o da máquina que está instalada a versão *on-premise*.

Para a realização destes testes foram usadas as seguintes funções:

- Autenticar através de ***Login***;
- Obter um ***dataset***;
- **Inserir** os dados;
- Obter os **dados** para colocar na *timeline*;

- **Editar** os dados.

Cada sequência de ações replica uma interação de um utilizador do sistema.

Para executar estas funções, foram usados pedidos *Representational State Transfer* (REST) [93] ao servidor da aplicação *SaaS*.

Para cada função iremos ter um *endpoint link* que irá comunicar com a API, sendo os seguintes:

- ***Login***: `https://writebacksaaswebapp.azurewebsites.net/#/writebacktwbe/api/user/login`
- ***Get Dataset***: `https://writebacksaaswebapp.azurewebsites.net/#/writebacktwbe/api/dataset/getWidgetConfiguration/DATASET_ID`
- ***Get Data***: `https://writebacksaaswebapp.azurewebsites.net/#/writebacktwbe/api/dataset/getData/`
- ***Insert Data***: `https://writebacksaaswebapp.azurewebsites.net/#/writebacktwbe/api/dataset/appendData/`
- ***Edit Data***: `https://writebacksaaswebapp.azurewebsites.net/#/writebacktwbe/api/dataset/editData/`

Deste modo, com os *endpoints* definidos, para poder definir os parâmetros do pedido, foi utilizada a extensão de *debug* para o *Tableau* [20] para obter o mesmo. Esta extensão é uma extensão genérica que dá para fazer *debug* de qualquer site.

Esta extensão permite incorporar uma página *web*, que possa ser acedida por um *link*, e por conseguinte, com esse *link*, através de um navegador *Chromium*, como por exemplo o *Google Chrome*, permite rastrear todos os pedidos e os seus parâmetros correspondentes feitos no *Tableau*. O *Tableau* tem que ser iniciado em modo de *debug*.

Com a extensão foi observado que os parâmetros necessários para executar a função de *login* são, o *username*, a *password* e o parâmetro *Content-Type*, que contém a informação do tipo de dados que irá ser enviado nos parâmetros do pedido, sendo neste caso, *multipart/form-data*. Para além disso, em todas as funções é necessário colocar o parâmetro *extensionsite* que contém o *site* do cliente. Um utilizador, ao fazer *login*, terá de, em cada pedido, colocar o parâmetro *Authorization* (que é obtido através do *login*) que contém o *token* de autenticação.

Foram executados testes, com 2 e 4 *Thread Groups*, onde estes foram divididos em 2 *sites* distintos e alterando a ordem das funções em cada um. Em cada *Thread Group* foram

executados 4 testes diferentes, sendo estes com um número total de Threads de 350, 700, 1000 e 2000.

5.3.2 Resultados dos testes

Na Tabela 5.1 podem verificar-se os resultados dos testes, onde cada linha da tabela representa um teste e estão representados vários parâmetros, nomeadamente:

- O **número total de *threads*** (NTT) que foram executadas no teste;
- O **número de processos** (NP) executados, simbolizando o número *Threads Groups* a serem usados no *JMeter*, que equivale ao número de utilizadores em simultâneo;
- O **número de *threads* por processo** (NTpP);
- O **ramp-up perior** (RuP) que se encontra em segundos e simboliza o tempo de espera antes de executar o próximo utilizador (*thread*);
- O **loop count** (LC) que simboliza o número de vezes que vai ser executado o teste;
- A **taxa total de erro** que simboliza a quantidade de erros resultante desse teste.

NTT	NP	NTpP	RuP (s)	LC	Erro (%)
350	2	175	0.1	1	0
700	2	350	0.1	1	**0.469**
1000	2	500	0.1	1	**2.283**
2000	2	1000	0.1	1	**5.655**
700	4	175	0.1	1	0
1400	4	350	0.1	1	**2.122**
2000	4	500	0.1	1	**4.229**
4000	4	1000	0.1	1	**8.176**

Tabela 5.1: Resultados dos Testes de Desempenho

A capacidade do servidor em uso é a seguinte: para a *Azure App Service*, cujo a versão é a *Standard*, contém um processador de 1 núcleo e 100 *Address Generation Unit* (AGU), 1.75 *Gigabyte* (GB) de *Random-access Memory* (RAM) e 50GB de armazenamento; para o *Azure SQL Server*, cujo a versão é Básica, contém 50 *Elastic Database Transaction Units* (*eDTUs*) e 4.48GB de armazenamento.

Com estes testes pode-se concluir que, com esta capacidade de servidor, a aplicação consegue suportar (numa estimativa) entre aproximadamente 400 e 600 interações com um período de crescimento (*ramp-up*) de 0.1 segundos.

Tendo em conta que as interações dos utilizadores são manuais, e por conseguinte, irá ter sempre um intervalo de tempo entre as interações; pelo facto destes testes dependerem de qual caso de utilização está a ser usado; e também por não haver acesso às informações da versão *on-premise*, não é possível estimar qual a média da frequência de utilização do sistema por parte dos utilizadores.

Para um aumento do desempenho, ter-se-á de melhorar as componentes da aplicação *web Azure App Service* e da base de dados *Azure SQL Database* para um nível superior *Standard* ou mesmo *Premium*.

6

Conclusões

O objetivo desta dissertação é migrar o *software Write-Back* de um *software on-premise* para um *Software as a Service*, tendo em conta todos os fatores importantes como a segurança e a escalabilidade e robustez.

Com o *Software as a Service* é possível distribuir e comercializar um *software* via *internet on cloud*, sendo o mesmo atualizado e gerido apenas pela parte do desenvolvedor e não pelo cliente.

Nesta dissertação foram efetuados alguns passos para uma possível solução do problema.

Inicialmente foi compreendido como está implementada a versão *on-premise* do *Write-Back*, analisando como está dividida a lógica da aplicação *front-end* e *back-end*, as suas formas de autenticação e as suas bases de dados e tecnologias.

Seguidamente, analisaram-se os conceitos e as características de *Software as a Service*, tais como a escalabilidade, a arquitetura *multi-tenancy*, a segurança, a integração e a tolerância a falhas.

No passo seguinte, foram analisadas as diferentes possibilidades para o desenvolvimento, bem como a sua arquitetura. Estas possibilidades são referentes à segurança (para a autênticação, a organização *multi-tenancy* e a transmissão entre servidor e cliente), à escalabilidade para as diferentes componentes (*load balancer*, *front-end*, *back-end* e base de dados) e à robustez. Posteriormente, foi desenvolvida a aplicação na nuvem com os serviços da *Azure*, com a *Azure App Service* para o alocamento da aplicação, a

Azure SQL Database para a base de dados interna do *Write-Back*, a *Azure Application Insights* para a gestão dos *logs* e a *Azure Front Door* para o *load balancer*, tornando-se uma entrada segura da aplicação.

Por último, com a aplicação já alocada na nuvem, foram efetuadas alterações no código base para a melhoria da segurança entre os utilizadores e para otimização da aplicação, tanto para as configurações dos utilizadores, como para as bases de dados. Importa referir que não foi disponibilizado o código por este ser privado da empresa.

Após a conclusão dos passos mencionados, foram efetuados testes de performance para o *Write-Back SaaS* com o programa *JMeter*.

Em síntese, com o estudo desenvolvido nesta dissertação, conclui-se que, com a versão básica das componentes requisitadas da *Azure*, teremos uma estimativa entre 400 e 600 iterações com o serviço, num período de crescimento de 0.1 segundos, atingindo os objetivos inicialmente propostos.

6.1 Trabalho Futuro

Para investigações futuras, poderão ser desenvolvidos trabalhos que permitam melhorar quatro áreas essenciais.

O primeiro desenvolvimento possível, será adaptar o *Write-Back Manager* para funcionar num contexto de *Software as a Service*, sendo que aqui é que os utilizadores irão configurar os seus *sites* e o servidor *Write-Back Manager SaaS* é que irá criar o ficheiro *YAML* correspondente;

Em segundo, será criar um sistema de notificações que irá notificar um utilizador de que está a haver uma utilização de um *dataset* ao mesmo tempo, isto para facilitar a utilização e permitir que os utilizadores tenham noção do que outros utilizadores estão a fazer ou fizeram no mesmo *dataset*. Por exemplo, se alguém está a aceder a dados de um *dataset* e outro utilizador selecionar os mesmos dados que o primeiro (correspondente ao mesmo *dataset*), sendo que o primeiro ainda não os submeteu, o segundo irá ser notificado de que alguém já está a usar esses dados que foram selecionados por ele;

No terceiro ponto, deve-se finalizar a otimização das *queries* do ficheiros *properties*, retirando as *queries* com formato de texto igual;

Em quarto e último ponto, será fazer testes de escalabilidade. Neste caso pode-se por exemplo acrescentar um novo *Azure App Service*, duplicando o atual, e testar outra vez os testes com o *JMeter*, para ter uma avaliação linear em relação ao número de pedidos e ao número de serviços.

Referências

[1] Aarjavi Bhombe, Kshitij Walukar, Yash Thakare & Shailesh Kamble, "Comparative analysis of two bi tools: Micro strategy and tableau", em *Proceedings of International Conference on Advancements in Computing & Management (ICACM)*, 2019.

[2] *Active geo-replication*. URL: https://docs.microsoft.com/en-us/azure/azure-sql/database/active-geo-replication-overview, (accessed: 17.01.2022).

[3] Amit Gyandev Prajapati, Shankarlal Jayantilal Sharma & Vishal Sahebrao Badgujar, "All about cloud: A systematic survey", em *2018 International Conference on Smart City and Emerging Technology (ICSCET)*, IEEE, 2018, páginas 1–6.

[4] Amro Al-Said Ahmad & Peter Andras, "Scalability analysis comparisons of cloud-based software services", *Journal of Cloud Computing*, vol. 8, n.º 1, páginas 1–17, 2019.

[5] *Application Insights*. URL: https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview, (accessed: 20.09.2021).

[6] *Azure Active Directory*. URL: https://azure.microsoft.com/pt-pt/services/active-directory/, (accessed: 13.10.2021).

[7] *Azure App Service*. URL: https://azure.microsoft.com/en-us/services/app-service/, (accessed: 20.09.2021).

[8] *Azure Compute Platforms*. URL: https://docs.microsoft.com/en-us/dotnet/architecture/modernize-with-azure-containers/modernize-existing-apps-to-cloud-optimized/choosing-azure-compute-options-for-container-based-applications, (accessed: 20.09.2021).

[9] *Azure Content Delivery Network*. URL: https://docs.microsoft.com/en-us/azure/cdn/, (accessed: 20.09.2021).

[10] *Azure Front Door*. URL: https://docs.microsoft.com/en-us/azure/frontdoor/front-door-overview, (accessed: 15.03.2020).

[11] *Azure Kubernetes Service*. URL: https://azure.microsoft.com/en-us/services/kubernetes-service/, (accessed: 20.09.2021).

[12] *Azure SQL - Hyperscale service tier*. URL: https://docs.microsoft.com/en-us/azure/azure-sql/database/service-tier-hyperscale, (accessed: 17.01.2022).

[13] *Azure SQL Server*. URL: https://azure.microsoft.com/en-us/services/sql-database/campaign/#overview, (accessed: 17.01.2021).

[14] *Azure Web Application Firewall*. URL: https://docs.microsoft.com/en-us/azure/web-application-firewall/ag/ag-overview, (accessed: 20.09.2021).

[15] Benjamin G. Sullins & Mark Whipple, *JMX in Action*. Manning Publications Co., 2002.

[16] Stamatia Bibi, Dimitrios Katsaros & Panayiotis Bozanis, "Business application acquisition: On-premise or saas-based solutions?", *IEEE software*, vol. 29, n.º 3, páginas 86–93, 2012.

[17] *Business continuity and HADR for SQL Server on Azure Virtual Machines*. URL: https://docs.microsoft.com/en-us/azure/azure-sql/virtual-machines/windows/business-continuity-high-availability-disaster-recovery-hadr-overview, (accessed: 17.01.2022).

[18] *Chromium*. URL: https://www.chromium.org/, (accessed: 15.03.2020).

[19] *CSS*. URL: https://www.w3.org/TR/CSS/#css, (accessed: 15.03.2020).

[20] *Debug Extensions in Tableau Desktop*. URL: https://tableau.github.io/extensions-api/docs/trex_debugging.html, (accessed: 21.01.2022).

[21] Dimpi Rani & Rajiv Kumar Ranjan, "A comparative study of saas, paas and iaas in cloud computing", *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, n.º 6, 2014.

[22] *Documentação Write-Back*. URL: https://writeback4t.atlassian.net/wiki/spaces/WBTABEXT/overview, (accessed: 15.03.2020).

[23] *Elastic pools in Azure SQL Database*. URL: https://docs.microsoft.com/en-us/azure/azure-sql/database/elastic-pool-overview, (accessed: 17.01.2022).

[24] Alan Eldridge, "Best practices for designing efficient tableau workbooks", *Tableau Workbooks, Jan*, vol. 31, pág. 2013, 2013.

[25] Emily H. Halili, *Apache JMeter*. Packt Publishing Birmingham, 2008.

[26] Gang Peng, "Cdn: Content distribution network", *arXiv preprint cs/0411069*, páginas 2–5, 2004.

[27] Gelogo Yvette E & Sunguk Lee, "Database management system as a cloud service", *International Journal of Future Generation Communication and Networking*, vol. 5, n.º 2, páginas 71–76, 2012.

[28] *GitHub Tabjolt*. URL: https://github.com/tableau/tabjolt, (accessed: 22.01.2022).

[29] *GlassFish*. URL: https://javaee.github.io/glassfish/, (accessed: 15.03.2020).

[30] Guilin Wang, Jiangshan Yu & Qi Xie, "Security analysis of a single sign-on mechanism for distributed computer networks", *IEEE Transactions on Industrial Informatics*, vol. 9, n.º 1, páginas 294–302, 2012.

[31] Halil Ibrahim Karaca, "Migration of an on-premise single-tenant enterprise application to the azure cloud: The multi-tenancy case study",

[32] Hitesh Ballani, Paul Francis & Sylvia Ratnasamy, "A measurement-based deployment proposal for ip anycast", em *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 2006, páginas 231–244.

[33] *HTML*. URL: https://html.spec.whatwg.org/, (accessed: 15.03.2020).

[34] William H. Inmon, "What is a data warehouse", *Prism Tech Topic*, vol. 1, n.º 1, páginas 1–5, 1995.

[35] *IntelliJ IDEA*. URL: https://www.jetbrains.com/idea/, (accessed: 15.03.2020).

[36] *Introducing TabJolt: A point-and-run load and performance testing solution for Tableau Server*. URL: https://www.tableau.com/about/blog/2015/4/introducing-tabjolt-point-and-run-load-testing-solution-tableau-server-38604, (accessed: 22.01.2022).

[37] Jasti Amarnath, Payal Shah, Rajeev Nagaraj & Ravi Pendse, "Security in multi-tenancy cloud", em *44th Annual 2010 IEEE International Carnahan Conference on Security Technology*, IEEE, 2010, páginas 35–41.

[38] *Java*. URL: https://www.java.com/, (accessed: 15.03.2020).

[39] *Javascript*. URL: https://www.javascript.com/, (accessed: 15.03.2020).

[40] *JDBC*. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/, (accessed: 15.03.2020).

[41] Jerry Gao, Pushkala Pattabhiraman, Xiaoying Bai & Wei-Tek Tsai, "Saas performance and scalability evaluation in clouds", em *Proceedings of 2011 IEEE 6th international symposium on service oriented system (SOSE)*, IEEE, 2011, páginas 61–71.

[42] *JMeter*. URL: https://jmeter.apache.org/, (accessed: 08.11.2021).

[43] *Json*. URL: https://www.json.org/json-en.html, (accessed: 15.03.2020).

[44] *Json Web Tokens*. URL: https://jwt.io/, (accessed: 15.03.2020).

[45] Juan C. Quiroz, Tim Chard, Zhisheng Sa, Angus G. Ritchie, Louisa Jorm & Blanca Gallego, "Extract, transform, load framework for the conversion of health databases to omop", *medRxiv*, 2021.

[46] *Kerberos*. URL: https://web.mit.edu/kerberos/, (accessed: 26.08.2020).

[47] Lehrig Sebastian, Hendrik Eikerling & Steffen Becker, "Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics", em *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*, 2015, páginas 83–92.

[48] Leonardo P. Tizzei, Marcelo Nery, Vinícius C. V. B. Segura & Renato F. G. Cerqueira, "Using microservices and software product line engineering to support reuse of evolving multi-tenant saas", em *Proceedings of the 21st International Systems and Software Product Line Conference-Volume A*, 2017, páginas 205–214.

[49] *Log4j*. URL: https://logging.apache.org/log4j/2.x/, (accessed: 15.03.2020).

[50] Lubna Alhenaki, Alaa Alwatban, Bashaer Alamri & Noof Alarifi, "A survey on the security of cloud computing", em *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, IEEE, 2019, páginas 1–7.

[51] Felipe Nery Rodrigues Machado, *Tecnologia e projeto de Data Warehouse*. Saraiva Educação SA, 2004.

[52] Maribel Yasmina Santos & Isabel Ramos, *Business Intelligence: tecnologias da informação na gestão de conhecimento*. FCA-Editora de Informática, Lda, 2006.

[53] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan & Jitendra Padhye, "Analyzing the performance of an anycast cdn", em *Proceedings of the 2015 Internet Measurement Conference*, 2015, páginas 531–537.

[54] Mazzucco Michele & Marlon Dumas, "Achieving performance and availability guarantees with spot instances", em *2011 IEEE International Conference on High Performance Computing and Communications*, IEEE, 2011, páginas 296–303.

[55] *Micro-Serviços na Azure*. URL: https://azure.microsoft.com/en-us/solutions/microservice-applications/, (accessed: 20.09.2021).

[56] *Microsoft Azure*. URL: https://azure.microsoft.com/, (accessed: 15.03.2020).

[57] Daniel G Murray, *Tableau your data!: fast and easy visual analysis with tableau software*. John Wiley & Sons, 2013, (accessed: 02.09.2020).

[58] *MySQL*. URL: https://www.mysql.com/, (accessed: 15.03.2020).

[59] Natsuhiko Sakimura, John Bradley, Mike Jones, Breno De Medeiros & Chuck Mortimore, "Openid connect core 1.0", *The OpenID Foundation*, S3, 2014.

[60] Nitin Naik & Paul Jenkins, "An analysis of open standard identity protocols in cloud computing security paradigm", em *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, IEEE, 2016, páginas 428–431.

[61] ——, "Securing digital identities in the cloud by selecting an apposite federated identity management from saml, oauth and openid connect", em *2017 11th International Conference on Research Challenges in Information Science (RCIS)*, IEEE, 2017, páginas 163–174.

[62] *NPM*. URL: https://www.npmjs.com/, (accessed: 15.03.2020).

[63] *OpenID Connect*. URL: https://openid.net/connect/, (accessed: 26.08.2020).

[64] *OpenID Connect Tableau*. URL: https://help.tableau.com/current/server/en-us/openid_auth.htm, (accessed: 15.09.2021).

[65] Panos Vassiliadis, Alkis Simitsis & Spiros Skiadopoulos, "Conceptual modeling for etl processes", em *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP*, 2002, páginas 14–21.

[66] Pavel Rabetski & Gerardo Schneider, "Migration of an on-premise application to the cloud: Experience report", em *European Conference on Service-Oriented and Cloud Computing*, Springer, 2013, páginas 227–241.

[67] Pedro Guedes De Miranda, "Saas (software as a service)-infrastructures and applications in real scenarios", Tese de Doutoramento, 2010.

[68] Peter Buxmann, Thomas Hess & SonjaLehmann, "Software as a service", *Wirtschaftsinformatik*, vol. 50, n.º 6, páginas 500–503, 2008.

[69] *PostgreSQL*. URL: https://www.postgresql.org/, (accessed: 15.03.2020).

[70] *Publish Data Sources Tableau*. URL: https://help.tableau.com/current/pro/desktop/en-us/publish_datasources_about.htm, (accessed: 02.09.2020).

[71] *React JS*. URL: https://reactjs.org/, (accessed: 15.03.2020).

[72] Robert Győrödi, Marius Iulian Pavel, Cornelia Győrödi & Doina Zmaranda, "Performance of onprem versus azure sql server: A case study", *IEEE Access*, vol. 7, páginas 15 894–15 902, 2019.

[73] Saiqa Aleem, Faheem Ahmed, Rabia Batool & Asad Khattak, "Empirical investigation of key factors for saas architecture dimension", *IEEE Transactions on Cloud Computing*, 2019.

[74] Saiqa Aleem, Rabia Batool, Faheem Ahmed, Asad Khatak & Raja Muhammad Ubaid Ullah, "Architecture guidelines for saas development process", em *Proceedings of the 2017 International Conference on Cloud and Big Data Computing*, 2017, páginas 94–99.

[75] *Salesforce*. URL: https://www.salesforce.com/, (accessed: 15.03.2020).

[76] *SAML*. URL: https://wiki.oasis-open.org/security/FrontPage, (accessed: 15.03.2020).

[77] *Security Support Provider Interface*. URL: https://docs.microsoft.com/en-us/windows-server/security/windows-authentication/security-support-provider-interface-architecture, (accessed: 26.08.2020).

[78] SK Sowmya, P. Deepika & J. Naren, "Layers of cloud–iaas, paas and saas: A survey", *International Journal of Computer Science and Information Technologies*, vol. 5, n.º 3, páginas 4477–4480, 2014.

[79] *Spring*. URL: https://spring.io/, (accessed: 15.03.2020).

[80] *Spring Security*. URL: https://spring.io/projects/spring-security, (accessed: 15.03.2020).

[81] *SQL*. URL: https://www.iso.org/standard/63555.html, (accessed: 15.03.2020).

[82] Sungrim Cho, Sunwoong Hwang, Woochang Shin, Neunghoe Kim & Hoh Peter In, "Design of military service framework for enabling migration to military saas cloud environment", *Electronics*, vol. 10, n.º 5, pág. 572, 2021.

[83] *Tableau*. URL: https://www.tableau.com, (accessed: 15.03.2020).

[84] *Tableau Authentication*. URL: https://help.tableau.com/current/server/en-us/security_auth.htm, (accessed: 20.12.2021).

[85] *Tableau Desktop*. URL: https://www.tableau.com/products/desktop, (accessed: 20.08.2020).

[86] *Tableau Online*. URL: https://www.tableau.com/products/cloud-bi, (accessed: 20.08.2020).

[87] *Tableau Server*. URL: https://www.tableau.com/products/server, (accessed: 20.08.2020).

[88] *Tomcat*. URL: http://tomcat.apache.org/, (accessed: 15.03.2020).

[89] *Trusted Authentication*. URL: https://help.tableau.com/current/server/en-us/trusted_auth.htm, (accessed: 26.08.2020).

[90] *URL*. URL: https://url.spec.whatwg.org/, (accessed: 15.03.2020).

[91] *Write-Back*. URL: https://writeback4t.com/, (accessed: 15.03.2020).

[92] *Write-Back Authentication*. URL: https://writeback4t.atlassian.net/wiki/spaces/WBTABEXT/pages/332693963/Setup+Configure+with+Write-Back+Manager, (accessed: 20.12.2021).

[93] *Write-Back Back-end Procedures (REST Requests)*. URL: https://writeback4t.atlassian.net/wiki/spaces/WBTABEXT/pages/332759672/Back-end+Procedures, (accessed: 20.01.2022).

[94] *YAML*. URL: https://yaml.org/, (accessed: 19.09.2020).