MONOGRAFIA DE PROJETO FINAL DE GRADUAÇÃO

MIGRAÇÃO DO CATÁLOGO DE SERVIÇOS DE UMA PLATAFORMA ELETRÔNICA DE SAÚDE

Cássio Resende Pereira

Curso Superior de Engenharia de Redes de Comunicação

DEPARTAMENTO DE ENGENHARIA ELÉTRICA
FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

MONOGRAFIA DE PROJETO FINAL DE GRADUAÇÃO

MIGRAÇÃO DO CATÁLOGO DE SERVIÇOS DE UMA PLATAFORMA ELETRÔNICA DE SAÚDE

Cássio Resende Pereira

Monografia de Projeto Final de Graduação submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Bacharel em Engenharia de Redes de Comunicação

Banca Examinadora

Dr. Ricardo Staciarini Puttini, EnE/UnB _____
Orientador

Dr. Fábio Lúcio Lopes de Mendonça, EnE/UnB _____
Examinador Interno

Dr. Daniel Alves da Silva, EnE/UnB _____
Examinador Interno

FICHA CATALOGRÁFICA

PEREIRA, CÁSSIO RESENDE
MIGRAÇÃO DO CATÁLOGO DE SERVIÇOS DE UMA PLATAFORMA ELETRÔNICA DE SAÚDE [Distrito Federal] 2023.
xvi, 42 p., 210 x 297 mm (ENE/FT/UnB, Bacharel, Engenharia de Redes de Comunicação, 2023).
Monografia de Projeto Final de Graduação - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. SOA	2. Catálogo de serviços
3. WSO2	4. API REST
5. Node.js	6. Puppeteer

REFERÊNCIA BIBLIOGRÁFICA

PEREIRA, C.R. (2023). *MIGRAÇÃO DO CATÁLOGO DE SERVIÇOS DE UMA PLATAFORMA ELETRÔNICA DE SAÚDE* . Monografia de Projeto Final de Graduação, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 42 p.

CESSÃO DE DIREITOS

AUTOR: Cássio Resende Pereira
TÍTULO: MIGRAÇÃO DO CATÁLOGO DE SERVIÇOS DE UMA PLATAFORMA ELETRÔNICA DE SAÚDE .
GRAU: Bacharel em Engenharia de Redes de Comunicação ANO: 2023

Cássio Resende Pereira
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP: 70919-970 - Brasília-DF - Brasil

Dedico este trabalho a família e amigos, que sempre me apoiaram e continuarão apoiando em toda minha jornada.

AGRADECIMENTOS

Desde criança, pelos dez anos de idade, o objetivo já era claro, me tornar um Engenheiro. Esse trabalho marca o início dessa fase na minha jornada da vida, e seria impossível alcançá-la sem todas as pessoas que me ajudaram e estiveram lá por mim.

Gustavo, Henrique, Rafael e Leonardo, mais que amigos ou colegas, irmãos de guerra. Juntos vencemos todas as batalhas e desafios impostos nessa jornada, desde o 1° semestre. A nossa irmandade, momentos, dores e risadas tornou esse curso inesquecível e com certeza dezenas de vezes mais divertido. Obrigado.

Agradeço a toda minha família, mãe, pai, irmãos, avós, tios, primos e a minha futura esposa. O amor, complacência e paciência de todos foram e são essenciais a minha vida. Amo vocês.

A todos os meus professores, não só na universidade mas também no período escolar, obrigado. Prof. Dr. Ricardo Puttini, obrigado por acreditar em mim e me orientado nessa etapa final.

RESUMO

Este trabalho é uma extensão do projeto de [1], aonde é criado uma arquitetura de catálogo de serviços em nuvem para que clientes consumam APIs de forma segura. Ele detalha a criação da infraestrutura, a configuração dos softwares e o consumo dos serviços. O presente projeto aproveita o trabalho de [1], porém adiciona o processo de migração de uma versão mais antiga para a mais atual. O trabalho detalha as ferramentas utilizadas, a arquitetura apresentada, as etapas da proposta, a migração além de testes de funcionamento, averiguando a efetividade da mesma. A abordagem de migração empregada no projeto se provou uma alternativa de menor complexidade em relação ao outro método.

Palavras-chave: SOA, Catálogo de serviços, WSO2, API REST, Node.js, Puppeteer.

ABSTRACT

This work is an extension of [1], where an architecture for a cloud service catalog is created so that clients can safely consume APIs. It details the creation of the infrastructure, the configuration of the software and the consumption of services. The present project is inspired by the work of [1], recreating a similar architecture and configuration, but adds the process of migrating the software used from an older version to the most current one. The work details the tools used, the architecture presented, the stages of the proposal, the migration in addition to functioning tests, verifying its effectiveness. The migration approach employed in the project proved to be a less complex alternative compared to the other method.

Keywords: SOA, Service Catalog, WSO2, API REST, Node.js, Puppeteer.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SÍMBOLOS

Siglas

API	*Application Programming Interface*
ARN	*Amazon Resource Name*
AWS	*Amazon Web Services*
CIAM	*Customer Identity and Access Management*
CPU	*Central Processing Unit*
CRUD	*Create, Read, Update, Delete*
CSV	*Comma Separated Values*
DOM	*Document Object Model*
EC2	*Elastic Compute Cloud*
ESB	*Enterprise Service Bus*
HTML	*Hyper Text Markup Language*
HTTP	*Hyper Text Transfer Protocol*
HTTPS	*Hyper Text Transfer Protocol Secure*
ID	*Identity*
IS	*Identity Server*
ISKM	*Identity Server Key Manager*
JDBC	*Java Database Connectivity*
JDK	*Java Development Kit*
JPEG	*Joint Photographic Experts Group*
JSON	*JavaScript Object Notation*
JWT	*JSON Web Token*
LTS	*Long Term Support*
NPM	*Node Package Manager*
RAM	*Random Access Memory*
RDS	*Relational Database Service*
REST	*Representational State Transfer*
SFTP	*SSH File Transfer Protocol*
SGBD	Sistema Gerenciador de Banco de Dados
SOA	*Service Oriented Architecture*
SSH	*Secure Shell*
SSO	*Single Sign-On*
TI	Tecnologia da Informação
URL	*Uniform Resource Locator*
URN	*Uniform Resource Name*
VM	*Virtual Machine*
XML	*eXtensible Markup Language*

YAML *YAML Ain't Markup Language*

1 INTRODUÇÃO

Uma plataforma eletrônica de saúde é um software com vários componentes complexos que integra informações de prontuários eletrônicos e sistemas de informações terceiros relacionados a saúde. Eu (autor), faço parte de uma empresa que comercializa esse produto, aonde foi necessária uma migração da infraestrutura utilizada. Com isso, também foi aproveitado para realizar a migração da versão do software utilizado para catalogação dos serviços, foco deste projeto. Os serviços seguem a Arquitetura Orientada a Serviços (SOA).

SOA, conforme [2] é, "uma forma de tornar os componentes de software reutilizáveis e interoperáveis por meio de interfaces de serviço". Em uma era digital que está sempre em evolução, e com a crescente dependência de serviços web, a importância de padronizar interfaces e reutilizar componentes é incontestável, produzindo uma série de vantagens em termos de eficácia e economia para as empresas.

Assim como uma interface bem definida, manter um software atualizado é essencial para evitar problemas de segurança em protocolos defasados e maximizar a eficiência operacional. Normalmente, as atualizações introduzem aprimoramentos de velocidade, soluções para falhas e recursos adicionais. Assim, este trabalho concentra-se em possibilitar uma transição para a nova versão do software que seja o menos custoso possível para os usuários finais, ao mesmo tempo em que garante os benefícios da nova versão.

1.1 OBJETIVOS

1.1.1 OBJETIVOS GERAIS

O objetivo geral deste trabalho é criar um ambiente controlado para estudo, implementação e análise de resultados da migração da versão de software de um catálogo de serviços, acompanhando os processos que envolvem desde a migração dos usuários e suas chaves de consumo, até o acesso desses usuários às respectivas APIs.

1.1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos têm como propósito dar suporte ao objetivo geral deste trabalho. Os requisitos, técnicas e descrições a seguir visam implementar em um ambiente de testes e analisar a migração da versão de um catálogo de serviços em uma plataforma de registro eletrônico de saúde. Desta forma, objetiva-se:

- Entender e estudar a importância do catálogo e da catalogação de serviços dentro de uma Arquitetura Orientada a Serviços (SOA);
- Implementar, em um ambiente de testes na nuvem, a nova versão do software de catálogo de serviços para a plataforma de registro de atendimento eletrônico;

- Realizar a migração dos dados e informações da versão antiga para a nova, preservando as funcionalidades e chaves de consumo dos clientes;
- Avaliar a eficácia da migração e realizar testes que garantem a sua operacionalidade.

1.2 ESTRUTURA DOCUMENTAL

Este documento visa o estudo, implementação e análise de resultados de ferramenta de detecção de intrusão para proteção de *endpoints* em ambiente controlado e se encontra segmentado em sete capítulos sendo que:

- O primeiro capítulo se refere a introdução, que tem cunho principal de realizar uma abordagem descritiva das motivações, primeiros estudos e conceitos e objetivos do trabalho;
- O segundo capítulo se refere a fundamentação teórica e tem o objetivo de realizar um estudo mais a fundo dos conceitos utilizados na criação deste trabalho, além de descrever as ferramentas utilizadas;
- O terceiro capítulo retrata a arquitetura do projeto, trazendo o desenho topológico e realizando as configurações necessárias para a correta instalação dos *softwares*;
- O quarto capítulo apresenta a realização da migração e os testes de consumo das APIs, garantindo que a operacionalidade foi preservada;
- O quinto capítulo apresenta uma análise conclusiva sobre os resultados obtidos, sintetizando 1.1.2, e apresentando sugestões de trabalhos futuros relacionados a este estudo;
- Este projeto conta, além dos cinco capítulos apresentados, dois anexos que detalham os códigos utilizados.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo objetiva apresentar uma fundamentação teórica das tecnologias empregadas no projeto, sendo abordados conceitos, termos e teorias adjacentes essenciais para a sua sequência, assim como as ferramentas utilizadas para o seu desenvolvimento.

2.1 ARQUITETURA ORIENTADA A SERVIÇOS (SOA)

De acordo com [3], "Arquitetura orientada a serviços (SOA) é um método de desenvolvimento de software que usa componentes de software chamados de serviços para criar aplicações de negócios."Essa abordagem componentizada promove a reutilização de serviços, facilita a escalabilidade e a manutenção do software. Elaborando os principais conceitos de SOA pertinentes a este trabalho, temos:

- **Interoperabilidade:** É a capacidade de diferentes sistemas ou componentes trabalharem em conjunto, permitindo a troca de informações de forma simples. Utiliza-se formatos e protocolos padronizados para manter a comunicação independente da linguagem ou plataforma, conhecidos como **contratos de serviço**;
- **Contratos de Serviço:** Contratos de serviço especificam a interface do próprio serviço, descrevendo o formato de entrada e saída das suas mensagens e os tipos de operações que podem ser realizadas, sendo fundamental para controlá-lo;
- **Reusabilidade:** É a capacidade de diferentes sistemas ou componentes serem reutilizados em diferentes contextos e aplicações, permitindo uma alta modularidade de aplicação;
- **Escalabilidade:** Os serviços são pensados para funcionarem independentemente do tamanho da arquitetura ou complexidade, prevendo esse comportamento em suas interfaces;
- **Abstração dos serviços:** A abstração da lógica traz agnosticidade aos serviços, tornando-os multipropósito, analogamente a *chips* e processadores. Serviços com agnósticos bem desenhados são reutilizáveis e sujeitos ao uso paralelo, atendendo mais o seu uso;
- **Acoplamento Fraco:** Os serviços em SOA precisam ter um acoplamento fraco, com cada um sendo bem definido em seu papel. Necessitam ser *stateless*, isto é, que não dependem de transações ou informações anteriores, assim ao ser modificado não deve afetar os outros serviços que o utilizam.

Além destes conceitos, uma das características de SOA é o elemento centralizador, conhecido como *Enterprise Service Bus* (ESB). Ele atua como um intermediário entre os serviços a ele conectados em uma infraestrutura, roteando mensagens, transformando formatos e lidando com a mediação entre esses serviços, solucionando problemas de compatibilidade e padronização de mensagens.

2.1.1 Design Pattern

De acordo com [4], "Um design pattern é simplesmente uma solução de projeto comprovada para um problema de projeto comum que é formalmente documentado de maneira consistente". O objetivo desta seção é o entendimento do inventário de serviços e sua relação com o catálogo de serviços.

Dentro do conceito de inventário de serviços em SOA, existem diversos design patterns. No WSO2 API Manager são utilizados especialmente os design patterns presentes nos capítulos 7 e 8 de [4].

No capítulo 7, são apresentados os padrões de camadas lógicas do inventário. Esses padrões visam organizar e agrupar os serviços com base em sua funcionalidade e finalidade, facilitando a gestão e o design dos serviços dentro do inventário.

No capítulo 8, são apresentados os padrões de centralização do inventário, que visam centralizar partes-chave da arquitetura do inventário de serviços. Isso inclui camadas lógicas, esquemas, políticas e regras de negócio. Essa centralização contribui para uma gestão mais eficiente e um maior controle sobre o inventário de serviços.

Há outros design patterns adotados, mas esses são os principais para este trabalho.

2.2 CATALOGAÇÃO DE SERVIÇOS

De acordo com [5], "Da mesma forma que um inventário de produtos é documentado com um catálogo de produtos, um inventário de serviços é documentado com um catálogo de serviços". Então a catalogação dos serviços seria a documentação do inventário de serviços criado.

Ainda, a sua função pode ser definida em "... fornece aos usuários e clientes os meios de entender quais serviços eles podem realmente usar. Diferentes visualizações do catálogo de serviços podem fornecer detalhes e informações de serviços em um formato que seja compreendido pelo público relevante", como diz [6].

2.2.1 Importância da catalogação

Um software de catalogação de serviços traz muitos pontos positivos tanto para a empresa quanto para o cliente, pode ser citado, conforme [6]:

- Promove a TI para o papel de um provedor de serviços;
- Facilita que a TI seja administrada como um negócio e aloque custos para departamentos dentro da organização;
- Reduz os custos operacionais de TI, não apenas fornecendo serviços necessários, mas também para o níveis acordados de capacidade e disponibilidade;
- Reduz os custos operacionais identificando e eliminando o desperdício de serviços de TI;

- Reduz as ineficiências de serviços e processos de TI;
- Fornece uma plataforma para desenvolver uma compreensão mais clara dos requisitos e desafios de negócios que devem ser enfrentados;
- Fornece uma plataforma para melhorar a compreensão dos requisitos de negócios e outras questões, como a experiência;
- Permite que usuários e clientes escolham o serviço correto para suas necessidades.

Um catálogo de serviços bem elaborado é uma ferramenta valiosa para a equipe técnica de um negócio. Ele fornece informações cruciais sobre a manutenção e os pontos críticos dos serviços. Entender como eles funcionam e são utilizados é essencial para uma integração eficiente durante o desenvolvimento de aplicações. O uso do catálogo de serviços beneficia tanto a parte técnica, trazendo conhecimento dos serviços oferecidos, tanto a parte de negócios, podendo apresentar e disponibilizar esse serviço aos potenciais clientes.

Além disso, a sua utilização oferece uma visão abrangente e organizada do conjunto de serviços disponíveis, facilitando o entendimento das necessidades e demandas de cada um. Isso possibilita que a empresa apresente e disponibilize esses serviços aos potenciais clientes, tanto de forma técnica, melhorando o conhecimento interno sobre as ofertas, quanto de maneira comercial, promovendo sua divulgação e acesso.

2.3 API REST

O conceito de API REST está entrelaçado com as partes o compõem.

2.3.1 API e Web Service

Uma API (Application Programming Interface), de acordo com [7], é:

> "um conjunto de regras definidas que permitem que diferentes aplicativos se comuniquem entre si. Ele atua como uma camada intermediária que processa transferências de dados entre sistemas, permitindo que as empresas abram seus dados e funcionalidades de aplicativos para desenvolvedores terceirizados externos, parceiros de negócios e departamentos internos de suas empresas".

As APIs podem utilizar a Web como o meio de comunicação e, quando assim fazem, são chamadas de Web Service. Um Web Service é uma forma especializada de API que opera sobre a Internet. Através da utilização de protocolos de comunicação padronizados, como o HTTP, permite a interoperabilidade entre aplicações de diferentes plataformas e linguagens de programação. Desta forma, diferentes aplicações podem comunicar entre si desde que ambas utilizem o mesmo Web Service.

Por conta do uso da Web, esses serviços se tornam acessíveis de praticamente qualquer lugar do mundo, trazendo versatilidade e uma comunicação de fácil padronização. Tanto APIs quanto Web Services podem

ser combinados com regras de segurança, permitindo controle sobre o acesso dos recursos.

Como dito previamente, os Web Services podem utilizar o protocolo HTTP (ou HTTPS), e quando combinados, aproveitam-se os métodos desse protocolo para fazer operações no recurso em questão. As principais operações HTTP são:

Tabela 2.1: Principais operações do protocolo HTTP

Operação HTTP	**Descrição**
GET	Lê as representações de recursos
PUT	Substitui um recurso existente ou cria um novo se não existir
DELETE	Deleta um recurso
POST	Modifica um recurso
PATCH	Atualiza parcialmente um recurso
HEAD	Obtém os cabeçalhos de um recurso
OPTIONS	Descreve as opções de comunicação para um recurso

Essas operações refletem tanto operações de CRUD como as de bancos de dados, facilitando as operações na hora de construir um software. Em suma, as APIs atuam como uma ponte entre aplicações, permitindo a troca de informações de maneira eficiente e padronizada.

2.3.2 REST

Roy Fielding, em sua tese de doutorado [8], trouxe o conceito de *Representational State Transfer*. Diferentemente de um protocolo ou padrão, a REST é definida como um conjunto de princípios e restrições de projeto. A partir do momento em que se aderem a esses conjuntos, a REST torna possível a criação de sistemas distribuídos de alta escalabilidade e com níveis de acoplamento minimizados.

Os principais conceitos são:

- **Cliente-Servidor:** Esse princípio se trata da separação de responsabilidades. Devem ser separadas as preocupações da interface do usuário das preocupações do armazenamento de dados.
- ***Stateless:*** Cada solicitação do cliente ao servidor deve ser independente e conter todas as informações para que o dado seja manipulado.
- **Cacheável:** Os dados devem ser marcados como cacheáveis ou não, assim os clientes podem ver se os dados foram modificados, economizando processamento na requisição.
- **Interface uniforme:** Tem como objetivos a escalabilidade, a identificação e manipulação de recursos, as mensagens auto-descritivas e o uso de metadados.

Ainda, sobre os elementos arquiteturais de REST, no trabalho original de Roy Fielding, é apresentada a seguinte tabela:

Tabela 2.2: Elementos de Dados REST

Elemento de Dado	**Exemplos Modernos na Web**
Recurso	o alvo de uma referência de hipertexto
Identificador de recurso	URL, URN
Representação	documento HTML, imagem JPEG
Metadados de representação	tipo de conteúdo, data e hora da última modificação
Metadados de recurso	link de origem, alternativas, informação de variação
Dados de controle	se modificado desde, controle de cache

Os elementos de dados REST são componentes fundamentais da mesma arquitetura. A ideia principal é que cada recurso tenha um identificador único, fazendo com que se tenha certeza do recurso acessado, uma representação (nas APIs modernas, o REST é amplamente utilizado junto de dados no formato JSON e XML) além de metadados que facilitam o *cache* e se é necessário o cliente atualizar aquele recurso.

Uma arquitetura que respeita todos os conceitos da tese é chamada de RESTful, mas um dos elementos essenciais para a arquitetura ser considerada REST é respeitar os elementos de dados.

Portanto, uma API REST é uma API, geralmente um Web Service, construída respeitando os elementos de dados REST. Em uma API REST, os dados são identificados por URLs, acessados e manipulados usando métodos HTTP. A interação com os recursos é realizada por meio de representações, normalmente no formato JSON ou XML. Caso respeite todos os conceitos da tese de Roy Fielding, é caracterizado como uma API RESTful.

2.4 FERRAMENTAS UTILIZADAS

Esta seção tem como foco apresentar as ferramentas utilizadas, detalhando as partes pertinentes ao projeto.

2.4.1 WSO2

As ferramentas do WSO2, aqui apresentadas, são *open source* e podem ser obtidas em sua página do Github, em [9]. Versões para uso não comercial podem ser encontradas na página oficial do produto, em [10].

2.4.2 WSO2 API Manager

O WSO2 API Manager é uma plataforma de gerenciamento de APIs. De acordo com o provedor [11], as suas funcionalidades englobam "desde o desenho e publicação de APIs, até a gestão do ciclo de vida, desenvolvimento de aplicativos, segurança de API, limitação de taxa, visualização de estatísticas de APIs, além da conexão de APIs, Produtos de API e endpoints".

Entre as principais capacidades do produto, de acordo com o provedor [11], são:

- Desenvolver, implantar e gerir APIs: O API Publisher do WSO2 API Manager orienta desde a criação da API até a sua publicação, mantendo a conformidade com as especificações de cada API.
- Tornar as APIs descobertas: Com a utilização do API Publisher do WSO2 API Manager, é possível criar categorias ou usar tags para classificar as APIs, facilitando a sua descoberta.
- Segurança das APIs: O WSO2 API Manager permite a garantia da segurança das APIs por meio de controle de visibilidade, proteção contra ameaças, validação de carga útil de API, aplicação de políticas de limitação de taxa, além de autenticação e autorização de API.

Explicando mais sobre o funcionamento do produto, é possível criar além de APIs, *applications*, usuários e *roles*. As *applications* são uma coleção de APIs a qual um usuário pode ter acesso, com ela provendo o seu token de acesso às APIs. O token pode ser uma chave fixa ou JWT. Já os usuários podem ser criados por um *Sign Up* (processo em que um usuário cria a conta) ou pelo painel de administrador, aonde também se escolhe as *roles* que os usuários podem possuir. As *roles* são responsáveis pelo gerenciamento e acesso de cada usuário a sua respectiva aplicação.

2.4.3 WSO2 Identity Server

De acordo com [12], O WSO2 Identity Server é um produto que simplifica as necessidades de Gerenciamento de Identidade e Acesso do Cliente (CIAM). O produto garante fácil integração, com uma variedade de aplicativos para facilitar o login único (SSO), login social, federação de identidade, segurança de API e autenticação robusta.

No projeto, foi utilizado para controlar o acesso e autenticação de usuários que consomem as APIs expostas.

2.4.4 Amazon Web Services

A AWS é uma plataforma de computação em nuvem que oferece centenas de serviços sob demanda. Os serviços que fazem parte do escopo desse são: EC2 e RDS.

- **EC2:** É o serviço de máquinas virtuais (VMs) da AWS. Nele é possível escolher uma ampla gama de máquinas que atendem a diversas especificação de uso, tanto em hardware quanto em sistema operacional. Também é possível definir regras de acesso e *firewall*.
- **RDS:** É o serviço gerenciado de bancos de dados relacionais da Amazon, atendendo os principais SGBDs do mercado. Neste trabalho foi utilizado um banco PostgreSQL.

2.4.5 Node.js

Para realizar a migração da versão 2.6 para a 4.2 do WSO2 API Manager, foi necessária uma abordagem de automação de *browser*. O Node.js foi escolhido porque, além da ampla gama de pacotes do NPM, também é um ambiente de execução para JavaScript, trazendo proximidade ao *browser*. Os pacotes utilizados foram o Puppeteer (para automação do browser) e o Papaparse (para leitura e escrita em arquivos CSV). O Papaparse é uma biblioteca focada na análise de arquivos JSON e CSV, possuindo funções como conversão entre formatos, leitura e escrita de arquivos.

2.4.6 Puppeteer

O Puppeteer é uma ferramenta que possiblita a simulação de ações de usuário, como cliques e digitação de texto no navegador. Ela facilita a interação, por meio de uma API de alto nível com funções simples o controle do navegador Chrome sobre o protocolo *DevTools*. A documentação pode ser conferida em [13].

A interação com a página Web neste trabalho foi feita de duas formas: pelo XPath, uma linguagem de navegação em documentos que utilizam *Markup*, e por execução de funções JavaScript diretamente na página.

O uso do XPath é relativamente direto, bastando selecionar o elemento da página e inspecioná-lo, com os principais navegadores do mercado suportando essa função (Chrome, Edge, Firefox e Opera). Assim, copia-se o XPath e tem-se o caminho absoluto do elemento na página o qual se deseja manipular programaticamente.

A execução de JavaScript direto no navegador permite o acesso e manipulação do HTML DOM, aumentando a possibilidade de funções a serem executadas.

O Puppeteer também possui funções com *timeouts*, que o usuário pode definir o tempo em que um elemento ou página precisa para aparecer, flexibilizando a automação para tratamento de casos inesperados.

3 ARQUITETURA PROPOSTA

Este capítulo visa descrever as etapas da proposta abordadas para o desenvolvimento do projeto, seguido de seções abordando seu detalhamento.

3.1 ETAPAS DA PROPOSTA

O Projeto visa implementar o software de catálogo de serviços do WSO2 API Manager em sua versão 4.2, junto ao WSO2 Identity Server na versão 6.1.0, como Key Manager. Em seguida foi realizada a migração dos dados, do API Manager 2.6 já existente, em outra nuvem para o 4.2 da nuvem do projeto.

O WSO2 API Manager e o WSO2 Identity Server compartilham o banco de dados 'WSO2_SHARED_DB' e 'WSO2AM_DB'. Na figura 3.1, é possível observar esse esquema:

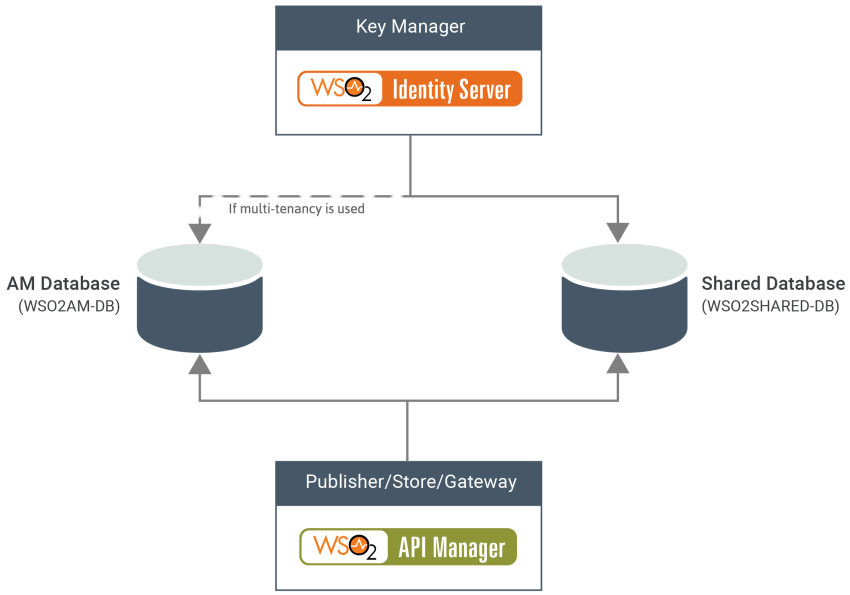

Figura 3.1: Cenário de utilização do Identity Server como *Key Manager*. Fonte: WSO2 APIM Documentation

Para alcançar os objetivos propostos, o trabalho foi divido nas seguintes etapas:

- **Etapa 1 → Inicialização e configuração das máquinas em nuvem;**
- **Etapa 2 → *Build* dos produtos WSO2 API Manager e WSO2 Identity Server;**
- **Etapa 3 → Configurações dos produtos WSO2 nas máquinas em nuvem;**
- **Etapa 4 → Migração das APIs e usuários do ambiente antigo para o novo;**

- **Etapa 5 → Migração automática com o Puppeteer das *applications* e APIs subscritas de cada usuário;**
- **Etapa 6 → Migração automática, manipulando o banco de dados das *Consumer Key* e *Secret Key* das *applications* de cada usuário;**
- **Etapa 7 → Teste de consumo de serviço com as chaves migradas;**
- **Etapa 8 → Análise dos resultados.**

3.2 INICIALIZAÇÃO E CONFIGURAÇÃO DAS MÁQUINAS EM NUVEM

Esta migração se deu em um ambiente de desenvolvimento, no qual não foi necessária uma elevada capacidade computacional. Então, para reduzir custos, os produtos do WSO2 foram instalados em uma única instância EC2, assim como seus bancos de dados em uma única instância RDS.

A instância EC2 escolhida foi a *t3a.medium*, possuindo recursos computacionais de 2 vCPU, 4 GB de RAM e com o sistema operacional Ubuntu 22.04 Server LTS. O seu custo operacional mensal é de US$ 27,07, se ligada ininterruptamente.

Já para os banco de dados, foi selecionada a *db.t3.small*, possuindo recursos computacionais de 2 vCPU, 2 GB de RAM e com mecanismo PostgreSQL na versão 15.2. A atualização de instância no RDS pode ser feita de maneira simples, quando for necessária uma instância com maior performance. O seu custo operacional mensal é de US$ 62,05, se ligada ininterruptamente.

Desconsiderando impostos e conversões monetárias, o custo mensal apenas dos serviços acima é de aproximadamente US$ 89,12.

Com isso, a infraestrutura do projeto pode ser conferido na seguinte imagem:

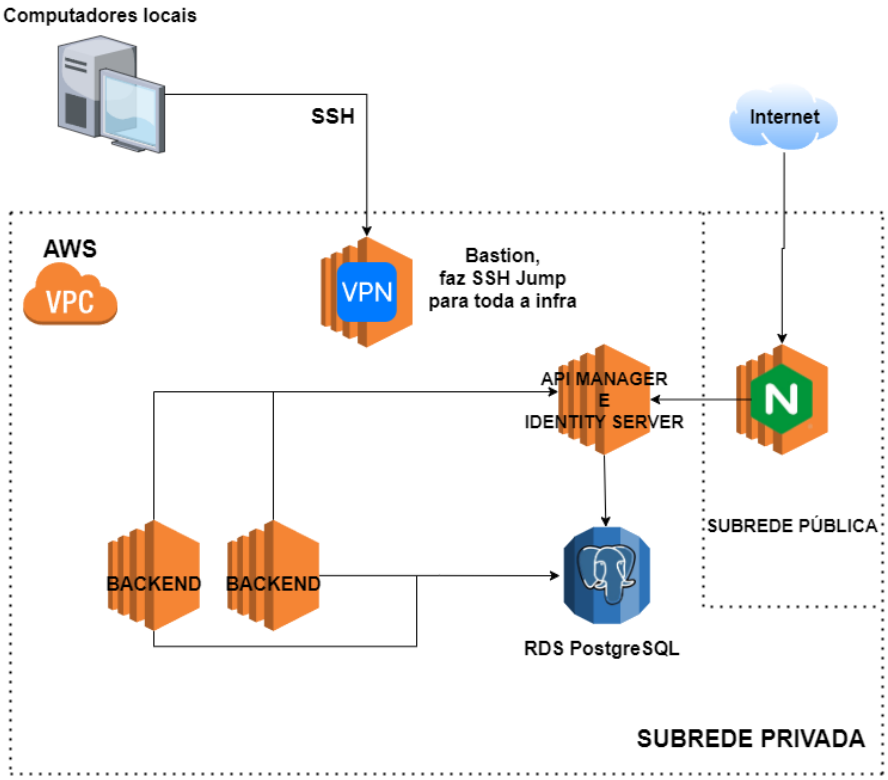

Figura 3.2: Infraestrutura na AWS. Fonte: autor

O *Backend* será implementado futuramente, e as outras máquinas estão presentes na arquitetura, porém não são adjacentes ao projeto.

Com a infraestrutura criada, é iniciada a instância EC2, aonde foi acessada via SSH para a instalação dos produtos e suas dependências.

Os produtos do WSO2 são aplicações Java, e para executá-los é necessário o *Java Development Kit* 11. Foi utilizada a versão *open source* conhecida como OpenJDK.

As etapas para a instalação do OpenJDK 11 foram:

1. Atualização do repositório de pacotes do Ubuntu:

```
sudo apt update
```

2. Instalação do OpenJDK com o seguinte comando:

```
sudo apt install openjdk-11-jdk
```

Com a instalação concluída, foi necessário adicionar a variável de ambiente JAVA_HOME a todos os usuários:

1. Abriu-se o arquivo *environment*:

```
sudo nano /etc/environment
```

2. Modificou-se o arquivo, para que ficasse da seguinte forma:

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin
:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
:/snap/bin:/usr/lib/jvm/java-11-openjdk-amd64/bin"
JAVA_HOME="/usr/lib/jvm/java-11-openjdk-amd64"
```

3. Para que as modificações fossem aplicadas, utilizou-se o seguinte comando como usuário *root*:

```
source /etc/environment
```

4. A conclusão da instalação foi averiguada com os comandos:

```
java -version
echo $JAVA_HOME
```

Foi mostrado na tela, que a versão 11 do Java e a variável 'JAVA_HOME' foram instaladas.

3.3 *BUILD* DOS PRODUTOS WSO2 API MANAGER E WSO2 IDENTITY SERVER

Como citado na seção 2.4, pelo caráter comercial, os produtos foram utilizados em sua versão *open source*, ao invés da distribuição oficial. São necessárias as ferramentas JDK-11, Maven, e Git para o *build*.

Para o *build* dos produtos foi utilizada uma máquina virtual Ubuntu 22.04 no Oracle VM Virtual Box.

- **JDK-11:** Foi utilizado o OpenJDK-11, com seu processo de instalação descrito na seção 3.2.
- **Maven:** O Maven foi instalado no Ubuntu com o comando:

```
sudo apt install maven
```

- **Git:** O Git foi instalado no Ubuntu com o comando:

```
sudo apt install git
```

3.3.1 *Build*

Para o *build* dos produtos do WSO2, foi criada uma pasta na qual foram realizados comandos, utilizando o terminal Ubuntu.

Primeiramente o *build* do WSO2 API Manager:

1. O produto foi baixado com o seguinte comando:

```
git clone https://github.com/wso2/product-apim.git
```

2. Entrou-se na pasta baixada:

```
cd product-apim
```

3. Trocou-se da *branch* Master para a v4.2.0

```
git checkout v4.2.0
```

4. Aplicou-se o *build* do WSO2 API Manager:

```
mvn clean install -Dmaven.test.skip=true
```

O arquivo ZIP do WSO2 API Manager se encontra dentro da pasta:

```
./product-apim/modules/distribution/product/target/wso2am-4.2.0.zip
```

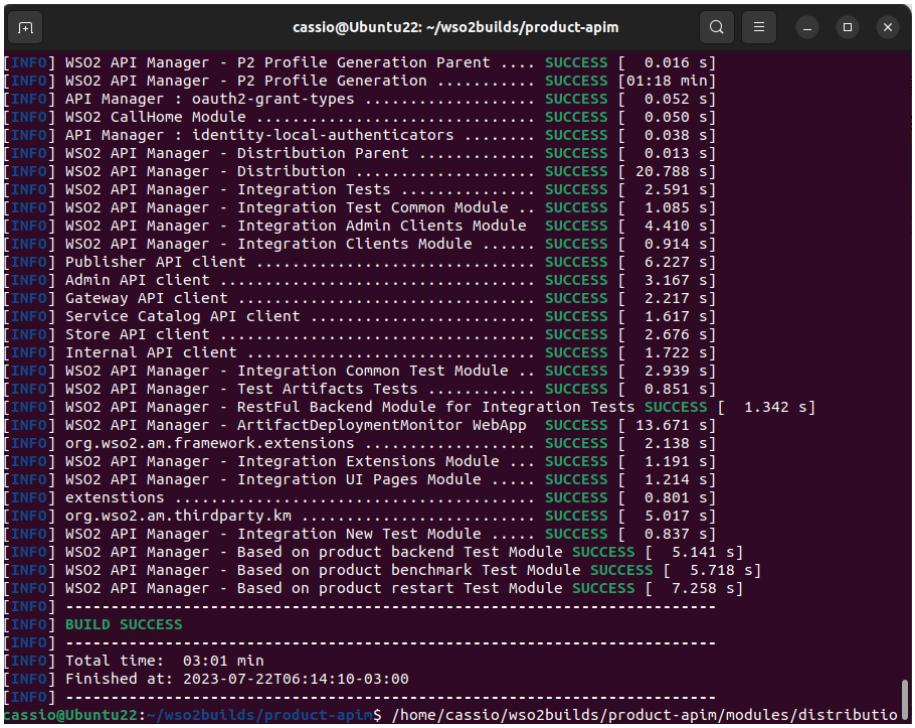

Figura 3.3: Processo de *build* do API Manager realizado com sucesso. Fonte: autor

Em seguida, o WSO2 Identity Server:

1. O produto foi baixado com o seguinte comando:

```
git clone https://github.com/wso2/product-is.git
```

2. Entrou-se na pasta baixada:

```
cd product-is
```

3. Trocou-se da *branch* Master para a v6.1.0

```
git checkout v6.1.0
```

4. Aplicou-se o *build* do WSO2 Identity Server:

```
mvn clean install -Dmaven.test.skip=true
```

O arquivo ZIP do WSO2 Identity Server se encontra dentro da pasta:

```
./product-is/modules/distribution/target/
```

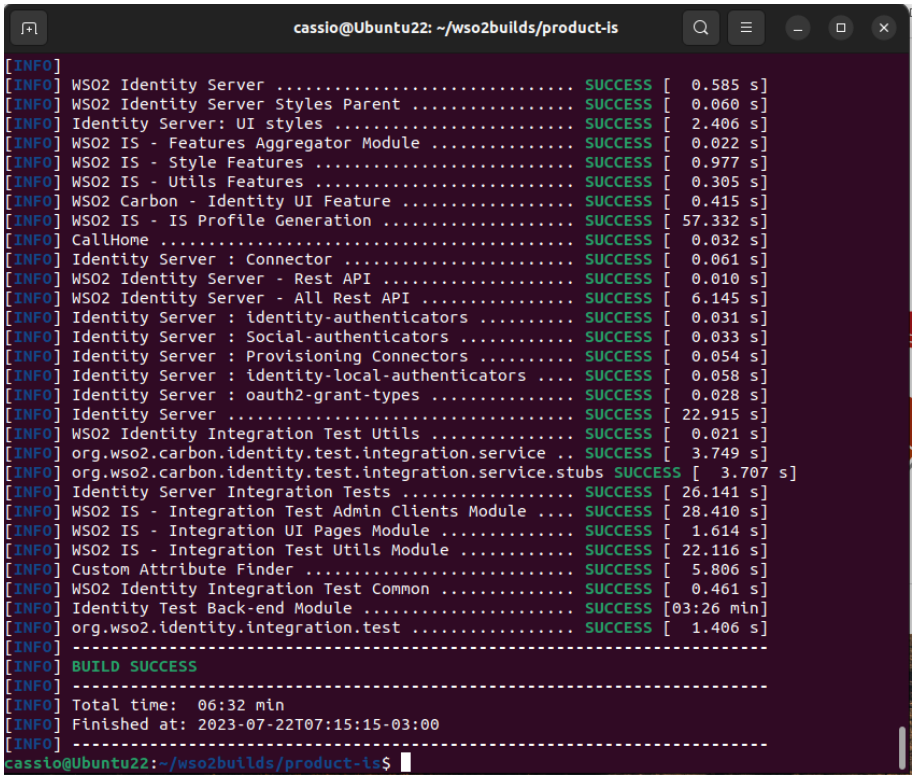

Figura 3.4: Processo de *build* do Identity Server realizado com sucesso. Fonte: autor

3.4 CONFIGURAÇÕES DOS PRODUTOS WSO2 NAS MÁQUINAS EM NUVEM

Os produtos após o processo de *build* são enviados à instância na AWS por SFTP à pasta /home/ubuntu, sendo descompactados:

```
sudo apt install unzip
sudo unzip wso2am-4.2.0.zip
sudo unzip wso2is-6.1.0.zip
```

E colocados na pasta /opt:

```
sudo mv wso2am-4.2.0 /opt/
sudo mv wso2is-6.1.0 /opt/
```

Também foi baixado o *driver* do postgres JDBC (Java 8), disponível em [14], e colocado nas pastas de ambos os produtos.

```
<PRODUCT-HOME>/repository/components/lib
```

3.4.1 Criação dos bancos de dados

O API Manager utiliza o banco de dados 'WSO2AM_DB' e 'WSO2_SHARED_DB', como mostrado na figura 3.1. Então, a partir da máquina virtual, foi acessada a instância RDS com o comando:

```
psql -h {ARN} -U postgres -d postgres -W
```

O ARN foi omitido por razões de segurança, assim como a senha. Acessada a instância do banco de dados, foram inseridos os seguintes comandos para criação dos bancos:

```
create database wso2_shared_db;
create database wso2am_db;
grant all privileges on database wso2_shared_db to postgres;
grant all privileges on database wso2am_db to postgres;
\q
```

Com os bancos criados, foi necessário utilizar os *scripts* do WSO2 API Manager para criar as tabelas e relações:

```
psql -h {ARN} -U postgres -d wso2_shared_db -f
/opt/wso2am-4.2.0/dbscripts/postgresql.sql -W

psql -h {ARN} -U postgres -d wso2am_db -f
/opt/wso2am-4.2.0/dbscripts/apimgt/postgresql.sql -W
```

3.4.2 Instalação dos bancos de dados nos produtos

Por *default*, os produtos do WSO2 utilizam um banco de dados Java H2. Dessa forma, fez-se necessária a subsituição para PostgreSQL. Para isso, são necessárias mudanças no arquivo 'deployment.toml', arquivo que contém as configurações dos produtos WSO2.

Começando pelo WSO2 Identity Server, o arquivo foi acessado com:

```
sudo nano /opt/wso2is-6.1.0/repository/conf/deployment.toml
```

Além das seguintes modificações feitas:

```
[database.identity_db]
type = "postgre"
url = "jdbc:postgresql://{ARN}:5432/wso2am_db"
username = "postgres"
password = "{senha}"
driver = "org.postgresql.Driver"
validationQuery = "SELECT 1"

[database.shared_db]
type = "postgre"
url = "jdbc:postgresql://{ARN}:5432/wso2_shared_db"
username = "postgres"
password = "{senha}"
driver = "org.postgresql.Driver"
validationQuery = "SELECT 1"
```

Em relação ao WSO2 API Manager, o arquivo foi acessado com:

```
sudo nano /opt/wso2am-4.2.0/repository/conf/deployment.toml
```

Também havendo as seguintes modificações:

```
[database.apim_db]
type = "postgre"
url = "jdbc:postgresql://{ARN}:5432/wso2am_db"
username = "postgres"
password = "{senha}"
driver = "org.postgresql.Driver"
validationQuery = "SELECT 1"

[database.shared_db]
```

```
type = "postgre"
url = "jdbc:postgresql://{ARN}:5432/wso2_shared_db"
username = "postgres"
password = "{senha}"
driver = "org.postgresql.Driver"
validationQuery = "SELECT 1"
```

3.4.3 Configuração do WSO2 Identity Server como Key Manager

O WSO2 Identity Server é executado na porta 9443, assim como o WSO2 API Manager. Por isso, foi necessário fazer um *offset* da porta para 9444.

Então novamente o arquivo *deployment.toml* foi acessado:

```
sudo nano /opt/wso2is-6.1.0/repository/conf/deployment.toml
```

E as seguintes modificações foram realizadas:

```
[server]
offset = 1
```

Para a configuração do WSO2 Identity Server como um *Key Manager*, têm-se as seguintes etapas:

1. Realização do download do WSO2 IS Connector 1.6.8, disponível em [15];
2. Extração do arquivo ZIP e cópia dos arquivos do diretório <wso2is-extensions-1.6.8>/dropins, para a instância no diretório:
   - wso2is.key.manager.core-1.6.8;
   - wso2is.notification.event.handlers-1.6.8;

   ```
   /opt/wso2is-6.1.0/repository/components/dropins/
   ```

3. Cópia do arquivo *keymanager-operations.war*, do diretório <wso2is-extensions-1.6.8>/webapps, ao diretório

   ```
   /opt/wso2is-6.1.0/repository/deployment/server/webapps/
   ```

4. Configuração dos *endpoints* do *traffic manager*, com adição das seguintes configurações ao final do arquivo ‘deployment.toml‘ do WSO2 Identity Server:

```
[[event_listener]]
id = "token_revocation"
type = "org.wso2.carbon.identity.core.handler.AbstractIdentityHandler"
name = "org.wso2.is.notification.ApimOauthEventInterceptor"
order = 1

[[resource.access_control]]
context = "(.)/keymanager-operations/user-info/claims(.)"
secure = true
http_method = "GET"
permissions = "/permission/admin/manage/identity/usermgt/list"
scopes = "internal_user_mgt_list"

[[resource.access_control]]
context = "(.*)/keymanager-operations/user-info/claims/generate"
secure = true
http_method = "POST"
permissions = "/permission/admin/manage/identity/usermgt/list"
scopes = "internal_user_mgt_list"

[[resource.access_control]]
context = "(.*)/keymanager-operations/dcr/register"
secure = true
http_method = "POST"
permissions = "/permission/admin/manage/identity/applicationmgt/create"
scopes = "internal_application_mgt_create"

[[resource.access_control]]
context = "(.*)/keymanager-operations/dcr/register(.*)"
secure = true
http_method = "GET"
permissions = "/permission/admin/manage/identity/applicationmgt/view"
scopes = "internal_application_mgt_view"

[[resource.access_control]]
context = "(.*)/keymanager-operations/dcr/register(.*)"
secure = true
http_method = "DELETE"
permissions = "/permission/admin/manage/identity/applicationmgt/delete"
scopes = "internal_application_mgt_delete"

[[resource.access_control]]
```

```
context = "(.*)/keymanager-operations/dcr/register(.*)"
secure = true
http_method = "PUT"
permissions = "/permission/admin/manage/identity/applicationmgt/update"
scopes = "internal_application_mgt_update"

[[resource.access_control]]
context = "(.*)/keymanager-operations/dcr/register(.*)"
secure = true
http_method = "POST"
permissions = "/permission/admin/manage/identity/applicationmgt/update"
scopes = "internal_application_mgt_update"

[tenant_context.rewrite]
custom_webapps = ["/keymanager-operations/"]
```

5. Configuração do *endpoint* do *event listener*, para publicação dos eventos de controle no *traffic manager*, com adição das seguintes configurações ao final do arquivo deployment.toml do WSO2 Identity Server:

```
[event_listener.properties]
notification_endpoint = "https://{apim}:9443/
internal/data/v1/notify"
username = "${admin.username}"
password = "${admin.password}"
'header.X-WSO2-KEY-MANAGER' = "WSO2-IS"
```

No arquivo deployment.toml do WSO2 API Manager, foram feitas as seguintes modificações:

```
[apim.key_manager]
service_url = "https://{iskm}:9444/services/"
type = "WSO2-IS"

[oauth.grant_type.token_exchange]
enable = false
```

Dessa forma, foi concluída a configuração de ambos os produtos.

3.5 MIGRAÇÃO DAS APIS E USUÁRIOS DO AMBIENTE ANTIGO PARA O NOVO

Após a configuração dos produtos, foram inicializados os produtos como *root*. Primeiro o Identity Server com:

```
cd /opt/wso2is-6.1.0/bin/
bash wso2server.sh
```

E após a sua inicialização, o API Manager com:

```
cd /opt/wso2am-4.2.0/bin/
bash api-manager.sh
```

Com a tela inicial sendo:

```
https://{host}:9443/devportal
```

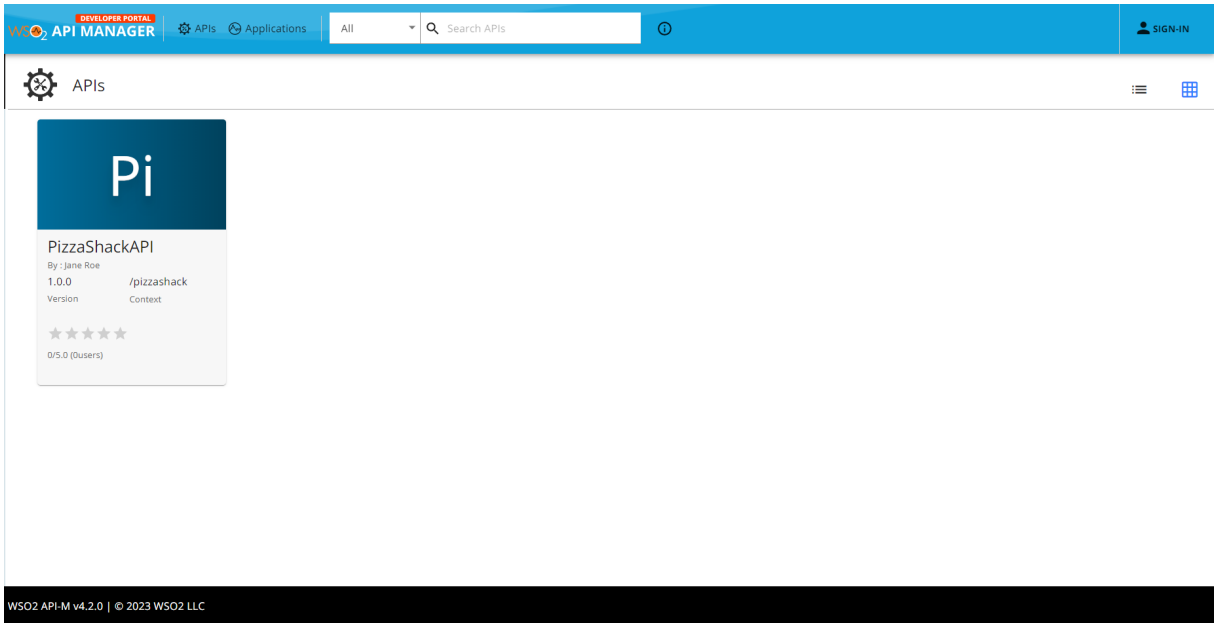

Figura 3.5: Portal de APIs antes da migração. Fonte: autor

PizzaShackAPI é uma API de testes que foi gerada automaticamente no WSO2 API Manager para testar se o *deploy* foi correto. Então foram geradas chaves para o consumo:

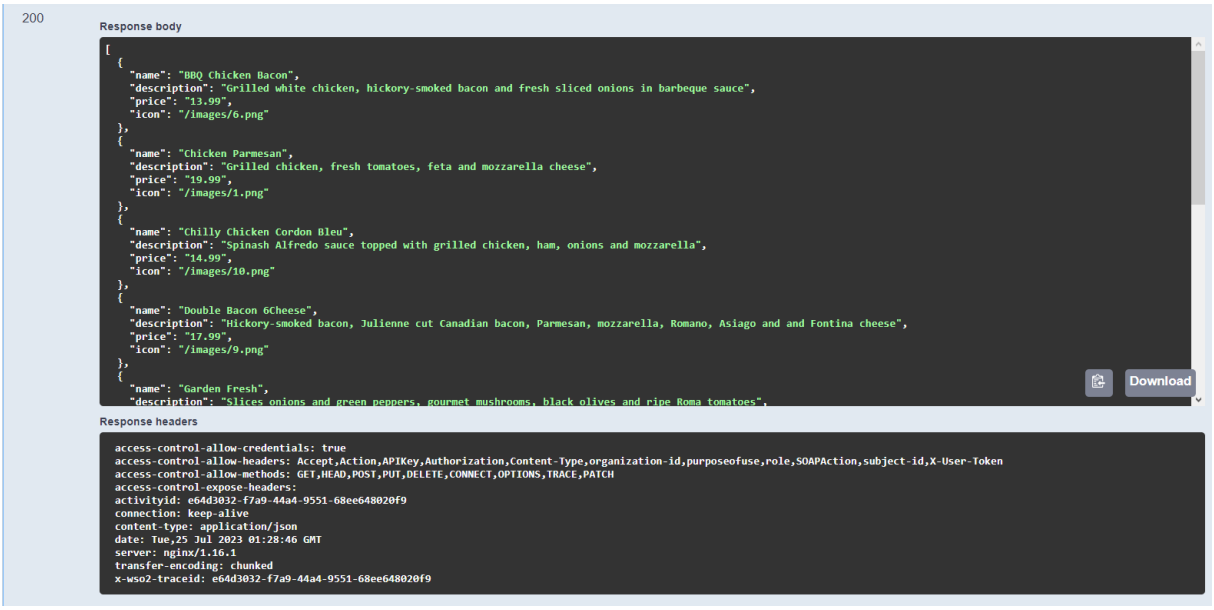

Figura 3.6: Consumo da API de testes com sucesso. Fonte: autor

Verificando que o *deploy* foi um sucesso, cada API teve o seu *Swagger Definition* baixado na página *publisher* do WSO2 API Manager 2.6, do ambiente antigo. Suas particularidades foram configuradas no WSO2 API Manager 4.2 do ambiente novo. Devido ao número diminuto de APIs, a sua automação é um processo imprático.

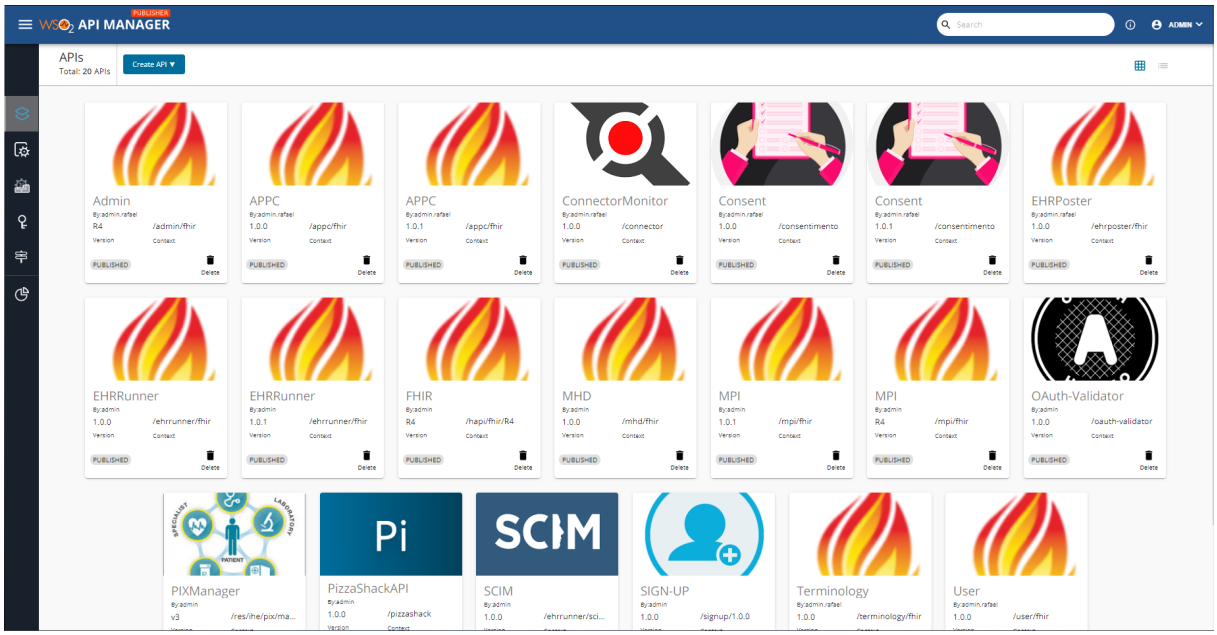

Figura 3.7: Portal de APIs após a migração. Fonte: autor

Após a migração das APIs, o próximo passo foi a migração dos usuários. Para obter os usuários foi realizada a seguinte *query* no banco de dados:

```
select distinct created_by as usuarios from am_subscription;
```

```
usuarios
--------------------
admin
i...
f...
v...
j...
r...
i...
```

Foram migrados os usuários (com exceção do admin, por ser um usuário padrão) com uma funcionalidade nativa do WSO2 API Manager 4.2, o *Bulk Import Users*, que pode ser acessado no seguinte portal, figura 3.8:

```
https://{host}:9443/carbon
```

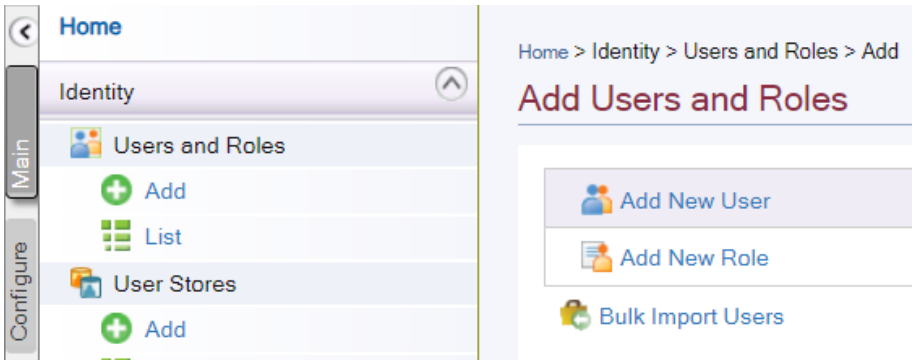

Figura 3.8: Importação automática de usuários. Fonte: autor

Preservando a segurança dos usuários, segue um exemplo fictício do arquivo CSV a ser importado:

```
UserName,Password,Claims
usuario1,senha1,http://wso2.org/claims/roles=Internal/subscriber
usuario2,senha2,http://wso2.org/claims/roles=Internal/subscriber
```

A *role* descrita por "claims/roles=Internal/subscriber"garante que esse usuário tenha o poder de subscrição em APIs. As senhas de cada usuário foram geradas de forma aleatória, podendo serem trocadas após a migração do ambiente.

3.6 MIGRAÇÃO AUTOMÁTICA COM O PUPPETEER DAS *APPLICATIONS* E APIS SUBSCRITAS DE CADA USUÁRIO

Com os usuários e APIs criadas, foi possível automatizar o processo que relaciona os usuários, *applications* e APIs.

3.6.1 Obtenção e relacionamento dos dados

Para obter as informações da versão antiga, foi feita a seguinte *query* em seu banco de dados:

```
SELECT
    app.created_by AS Usuario,
    app.name AS NomeDaAplicacao,
    api.api_name AS NomeDaAPI
FROM
    am_subscription sub
JOIN
    am_application app ON sub.application_id = app.application_id
JOIN
    am_api api ON sub.api_id = api.api_id
    order by
    Usuario, NomeDaAplicacao, NomeDaAPI;
```

	usuario	nomedaaplicacao	nomedaapi
5	admin	oauth-validator	OAuth-Validator
6	fluxme	portalPaciente	Admin
7	fluxme	portalPaciente	APPC
8	fluxme	portalPaciente	Consent
9	fluxme	portalPaciente	EHRRunner
10	fluxme	portalPaciente	MHD
11	fluxme	portalPaciente	MPI
12	fluxme	portalPaciente	OAuth-Validator
13	fluxme	portalPaciente	SCIM
14	fluxme	portalPaciente	User
15	fluxme	portalProfissional	Admin
16	fluxme	portalProfissional	APPC
17	fluxme	portalProfissional	Consent
18	fluxme	portalProfissional	EHRRunner
19	fluxme	portalProfissional	MHD
20	fluxme	portalProfissional	MPI
21	fluxme	portalProfissional	OAuth-Validator
22	fluxme	portalProfissional	SCIM
23	fluxme	portalProfissional	User
24	fluxme	signUp	SIGN-UP
25	ipes-fe	Federation	EHRRunner
26	ipes-pe	app	Admin
27	ipes-pe	app	APPC
28	ipes-pe	app	Consent
29	ipes-pe	app	EHRRunner

Figura 3.9: Informações dos usuários, *applications* e APIs subscritas. Fonte: autor

As informações necessárias para a automatização foram quatro: usuário, senha, nome da aplicação e ID da API. Usuário e senha foram definidos na seção 3.5, enquanto o nome da aplicação foi adquirido conforme mostrado na figura 3.9, mas os IDs da API só existem na versão 4.2. Esses IDs foram gerados na seção 3.5. Com a seguinte *query* foi possível obter o ID das APIs no 4.2:

```
SELECT
    api_name AS NomeDaAPI,
    api_uuid as IDdaAPI
FROM
    am_api
ORDER BY
    api_name;
```

Utilizando a ferramenta DBeaver, foram feitas as *queries* nos respectivos bancos de dados, e então criadas duas tabelas no Microsoft Office Excel: a primeira com o resultado da versão 2.6, e a outra da

versão 4.2. Então ambas as tabelas foram relacionadas entre sí, para a obtenção dos IDs das APIs. Com os dados prontos, foi possível criar um CSV contendo as informações necessárias, apenas adicionando as senhas. Então, o código Puppeteer foi executado para a criação das *applications* de cada usuário e subscrição automática nas APIs.

usuario	nomedaaplicacao	nomedaapi	iddaapi	nomedaapi	iddaapi
admin	ipes-connector	ConnectorMonitor	68e2332b-6c91-4faf-8760-305c8c0725de	Admin	bae06519-dc5f-46a1-a0f0-ef9b04b0017f
admin	ipes-connector	EHRPoster	622bf6bc-fe26-4a4d-8556-0d92574e2d6c	APPC	70d3c880-7eed-4ad4-ada7-96e02a26710e
admin	ipes-connector	MHD	176437ba-5872-4b0a-bf13-4d897acc1b83	APPC	414bdff2-6bf5-4140-9fd7-47d07d72e995
admin	ipes-connector	PIXManager	e76a0ae4-37d6-4b9f-b387-a18baf518bd9	ConnectorMonitor	68e2332b-6c91-4faf-8760-305c8c0725de
admin	oauth-validator	OAuth-Validator	cf1e7d0b-3d9b-4064-9695-dfba552fa9f9	Consent	cedb5f25-b779-4cd3-b6b8-2a0987bb64e3
flux	portalPaciente	Admin	bae06519-dc5f-46a1-a0f0-ef9b04b0017f	Consent	5784187b-b8a4-4b57-a66b-b030fec6277d
flux	portalPaciente	APPC	70d3c880-7eed-4ad4-ada7-96e02a26710e	EHRPoster	622bf6bc-fe26-4a4d-8556-0d92574e2d6c
flux	portalPaciente	Consent	cedb5f25-b779-4cd3-b6b8-2a0987bb64e3	EHRRunner	cd018c32-9c4a-4a48-8178-9528a7cc2f6e
flux	portalPaciente	EHRRunner	cd018c32-9c4a-4a48-8178-9528a7cc2f6e	EHRRunner	40498017-18eb-4b56-b1f0-96ccbd1afdbe
flux	portalPaciente	MHD	176437ba-5872-4b0a-bf13-4d897acc1b83	FHIR	2588c4fe-d52c-4bd7-b4c0-9f7b49a3a8a7
flux	portalPaciente	MPI	8977c0a2-7076-4d96-8ea7-aef2c5efd2bf	MHD	176437ba-5872-4b0a-bf13-4d897acc1b83
flux	portalPaciente	OAuth-Validator	cf1e7d0b-3d9b-4064-9695-dfba552fa9f9	MPI	8977c0a2-7076-4d96-8ea7-aef2c5efd2bf
flux	portalPaciente	SCIM	cc979cf6-8530-42ae-b536-1cb8d2f31a76	MPI	d95312c9-7c0c-419a-823c-f166d14a0ca2
flux	portalPaciente	User	4b5d1f34-666a-4e42-8d9b-afb95ea8c3f7	OAuth-Validator	cf1e7d0b-3d9b-4064-9695-dfba552fa9f9
flux	portalProfissional	Admin	bae06519-dc5f-46a1-a0f0-ef9b04b0017f	PIXManager	e76a0ae4-37d6-4b9f-b387-a18baf518bd9
flux	portalProfissional	APPC	70d3c880-7eed-4ad4-ada7-96e02a26710e	PizzaShackAPI	5c0d00c8-f7e3-43cc-a41c-d51cd972f7af
flux	portalProfissional	Consent	cedb5f25-b779-4cd3-b6b8-2a0987bb64e3	SCIM	cc979cf6-8530-42ae-b536-1cb8d2f31a76
flux	portalProfissional	EHRRunner	cd018c32-9c4a-4a48-8178-9528a7cc2f6e	SIGN-UP	dc3c5d2c-4fb9-4a3b-aab5-f9922d3dc402
flux	portalProfissional	MHD	176437ba-5872-4b0a-bf13-4d897acc1b83	Terminology	2709aa25-afc1-4bd0-bd63-60ef9aca2c6f
flux	portalProfissional	MPI	8977c0a2-7076-4d96-8ea7-aef2c5efd2bf	User	4b5d1f34-666a-4e42-8d9b-afb95ea8c3f7
flux	portalProfissional	OAuth-Validator	cf1e7d0b-3d9b-4064-9695-dfba552fa9f9		

Figura 3.10: Dados preparados para a automação. Fonte: autor

3.6.2 Migração automática com o Puppeteer

O algoritmo funciona da seguinte forma:

1. Entrada no usuário;
2. Exclusão da *DefaultApplication*;
3. Criação da *application* de acordo com o CSV, guardando a chave fixa em um outro arquivo CSV;
4. Subscrição em todas as APIs daquela *application* e, a cada subscrição, recarrega as variáveis a partir do CSV;
5. Verificação da presença de outras *applications* no usuário, caso negativo, *logout* do usuário e retorno ao item 1, com um novo usuário;
6. No caso de existência de *applications* no usuário, é realizado o retorno ao item 3.

A condição de parada se dá quando as últimas variáveis foram carregadas e o arquivo CSV chega ao fim. O código completo se encontra nos anexos. A chave fixa, citada no item 3 do algoritmo, será usada futuramente para mapeamento no Nginx. O mapeamento no Nginx está fora do escopo deste trabalho.

3.7 MIGRAÇÃO AUTOMÁTICA MANIPULANDO O BANCO DE DADOS DAS *CONSUMER KEY* E *SECRET KEY* DAS *APPLICATIONS* DE CADA USUÁRIO

A seguinte tabela e colunas da *query* estão presentes em ambas as versões do WSO2 API Manager:

```
SELECT
    username,
    app_name,
    consumer_secret,
    consumer_key
FROM
    public.idn_oauth_consumer_apps
ORDER
    BY username, app_name;
```

A *query* realizada na versão 2.6 será chamada de Tabela 1, e a da versão 4.2 de tabela 2.

Após os usuários, *applications* e APIs estarem todos criados e subscritos, foi necessário migrar as chaves da versão antiga para a versão nova. A *query* acima traz as informações das chaves das *applications* em ambos os bancos. Na versão 4.2, a coluna app_name se diferencia da versão 2.6, sendo formada por uma concatenação de *username* + _ +*application_uuid* + _ + *PRODUCTION*. O *application_uuid* é o *unique* ID da aplicação, que só existe na versão 4.2, assim como o ID da API citado anteriormente. Então foi realizada mais uma *query* no banco de dados da versão 4.2 para coletar os dados que relacionam o usuário, nome da aplicação e seu *unique* ID.

```
SELECT
    created_by as Usuario,
    name as NomeDaAplicacao,
    uuid
FROM
    public.am_application
ORDER BY
    Usuario, NomeDaAplicacao;
```

A *query* acima será chamada de Tabela 3.

Com esses dados, novas manipulações foram realizadas no Excel. Da coluna app_name da tabela 2, foi extraído o *application_uuid* em uma nova coluna na mesma tabela e, assim, foi possível fazer um PROCV na tabela 3, preenchendo em uma nova coluna o nome real da aplicação. Então, a tabela foi reordenada e ambas ficam com os dados ordenados em paralelo, possibilitando prosseguir para a substituição das chaves.

Usu	IdDaAplicacaoExtraido	NomeDaAplicacaoExtraido	app_name	Co	Co	Usu	NomeDaAplicacao	Cons	Cons
Coluna2		Coluna3	Coluna4			Coluna2			
adn	7a8578a7-5c15-4d9e-b68b	ipes-connector	admin_7a8578a7-5c15-4d9e-b68b-2dad90a9c75f_PRODUCTION	3X	uL	adm	admin_ipes-connector_PRODUCTION	Hm4	aIfT
adn	a2d099e6-eaac-49ea-960	oauth-validator	admin_a2d099e6-eaac-49ea-9609-361c9e27c06e_PRODUCTION	Mi	iA	adm	admin_oauth-validator_PRODUCTION	fuBt	U2O
flux	fesf_0f1a2077-d2b4-4ca1-	portalPaciente	fluxmed_fesf_0f1a2077-d2b4-4ca1-b4fd-b125c2b9fead_PRODUCTION	iq	tU	flux	fluxmed_fesf_portalPaciente_PRODUCTION	ng2f	Ij1rU
flux	fesf_3274525a-2cc8-4046-	portalProfissional	fluxmed_fesf_3274525a-2cc8-4046-93ef-3a5c1e9ba4d9_PRODUCTION	jli	4l	flux	fluxmed_fesf_portalProfissional_PRODUCTION	uNh	gub
flux	fesf_a29cd561-36be-4ff4-	signUp	fluxmed_fesf_a29cd561-36be-4ff4-814c-0c0d97dc687b_PRODUCTION	rm	g8	flux	fluxmed_fesf_signUp_PRODUCTION	ew4	b7gy
ipe	bd3db659-a9c8-49d9-a13	Federation	ipes-federation_bd3db659-a9c8-49d9-a131-f0a20e1e0634_PRODUCTION	07	XIz	ipes	ipes-federation_Federation_PRODUCTION	Atfq	pmI
ipe	46ab2444-a1d2-4484-b26	app	ipes-pe-recife_46ab2444-a1d2-4484-b265-704ede855634_PRODUCTION	OF	3x	ipes	ipes-pe-recife_app_PRODUCTION	166L	c6xy
ipe	8b985af1-471a-4ae9-956a	appSignUp	ipes-pe-recife_8b985af1-471a-4ae9-956a-e03da2ee7d36_PRODUCTION	6D	Gy	ipes	ipes-pe-recife_appSignUp_PRODUCTION	Io4h	Wpu
joa	49c10304-5c11-4695-a756-	v1	joao.ferreira_49c10304-5c11-4695-a756-d52ae190d45a_PRODUCTION	1V	hf	joac	joao.ferreira_v1_PRODUCTION	Dmh	1pn2
rica	6c592c30-72eb-4abc-9ecb	token-test	ricardo.goncalves_6c592c30-72eb-4abc-9ecb-393e1ee6c378_PRODUCTION	a_	qH	rica	ricardo.goncalves_token-test_PRODUCTION	DA8'	K1y_
vita	15e7d8e0-4b1e-4a38-932	vitai-conector-poc-rj	vitai_15e7d8e0-4b1e-4a38-9329-8efde7019353_PRODUCTION	0C	4i	vita	vitai_vitai-conector-poc-rj_PRODUCTION	DRv	gDO
vita	e667d180-5152-4b9f-8106	vitai-conector-teste	vitai_e667d180-5152-4b9f-8106-a669340c9c2b_PRODUCTION	t3	K4	vita	vitai_vitai-conector-teste_PRODUCTION	O2Ic	MLJ6
vita	f7f64e94-c1b1-44d2-838c	vitai-portal-embedded-teste	vitai_f7f64e94-c1b1-44d2-838c-05058ed89289_PRODUCTION	hr	4o	vita	vitai_vitai-portal-embedded-teste_PRODUCTION	SeR	ooZ

Figura 3.11: Dados para a migração das chaves. Fonte: autor

As informações de usuário e as chaves foram omitidas por segurança. Na imagem acima podem ser vistas na ordem, ambas tabela 2 e tabela 1, com as *applications* ordenadas para a correta substituição das chaves.

Com essas informações, foram gerados dois CSVs, ambos com duas colunas. Um arquivo com os *Consumer Keys* antigos (4.2) e novos (2.6), e outro com os *Secret Keys* antigos (4.2) e novos (2.6), para que fossem alteradas as chaves no banco de dados. As seguintes tabelas foram identificadas possuindo *Consumer Keys*:

```
public.am_application_key_mapping
public.idn_oauth_consumer_apps
public.idn_oidc_property
public.sp_inbound_auth
```

Com a tabela ‘public.idn_oauth_consumer_apps‘ sendo a única que também possui *Secret Keys*. Então foi desenvolvido um algoritmo em Python que se conecta ao banco de dados e faz a troca das chaves, utilizando-se dos CSVs. Esse algoritmo está nos anexos.

4 TESTES E RESULTADOS

4.1 TESTES DE CONSUMO DE SERVIÇOS COM AS CHAVES MIGRADAS

Para realizar o teste, entrou-se em um usuário e foi escolhida uma *application* que teve a sua chave migrada. A seguinte *application* de um usuário foi escolhida:

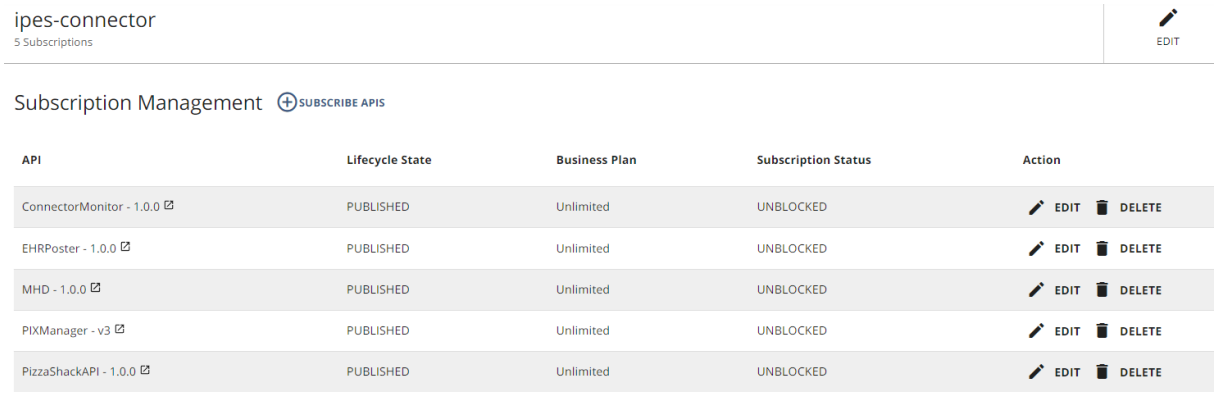

Figura 4.1: *Application* escolhida. Fonte: autor

Por conta do *backend* ainda não ter sido migrado, foi necessário realizar o teste na API de testes Pizzashack. O teste pode ser visto na figura 4.2.

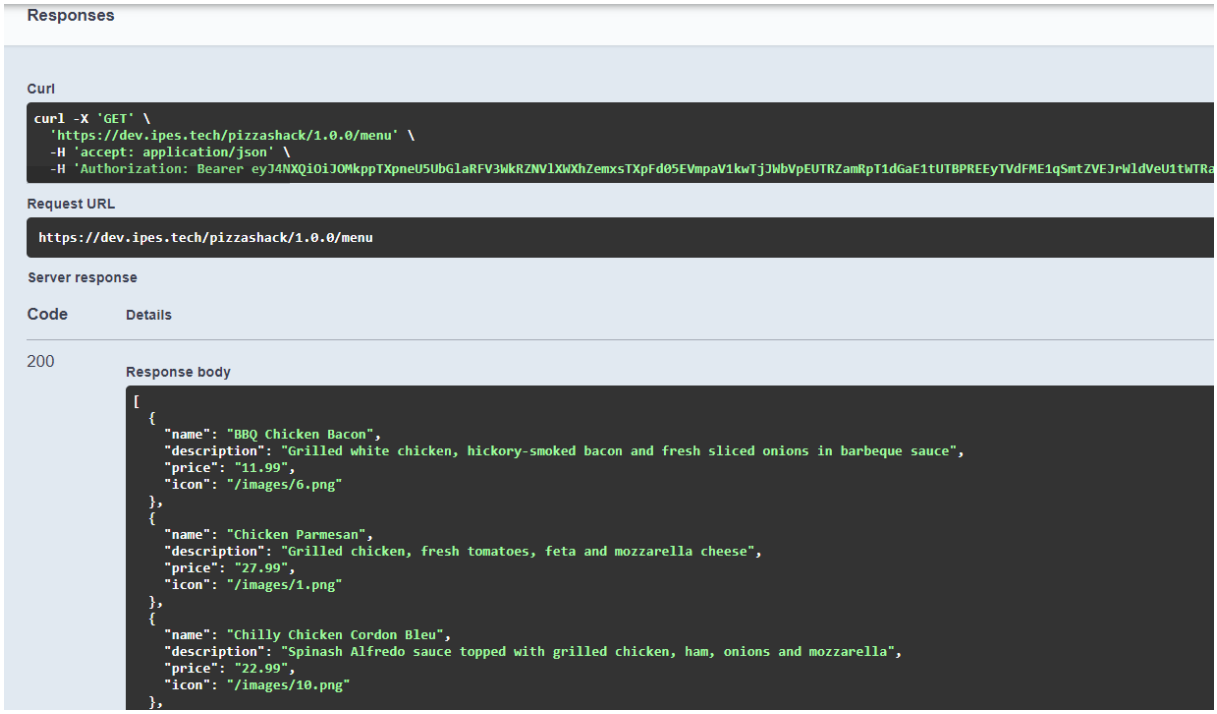

Figura 4.2: Teste da chave migrada bem sucedido. Fonte: autor

O *Access Token* gerado foi conferido no site jwt.io, no qual pode ser encontrada a *Consumer Secret* migrada. Na figura abaixo, é mostrado o local aonde esse *token* pode ser encontrado.

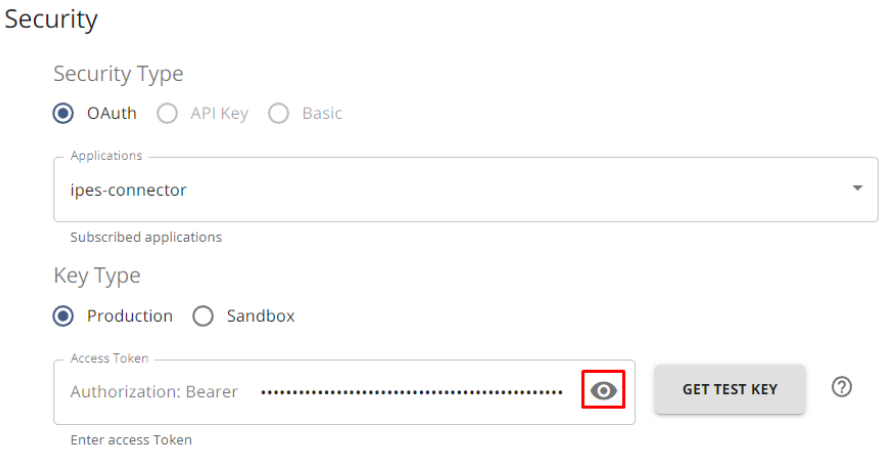

Figura 4.3: Local do *Access Token* no API Manager. Fonte: autor

4.2 ANÁLISE DOS RESULTADOS

Analisando os resultados obtidos pelos testes da seção anterior, foi possível averiguar que, na nova versão do WSO2 API Manager, a recriação dos usuários, *roles*, *applications* e APIs subscritas foi bem sucedida, preservando as funcionalidades do *software* sem prejudicar sua operação.

Com a identificação de todas as tabelas que possuem *Consumer Key* e *Secret Key*, foi possível migrar as chaves das *applications* antigas com êxito. Além disso, as APIs foram testadas com o *token* JWT, que foi gerado corretamente e verificado na página jwt.io. A geração correta do *token* pelo API Manager, seguido da utilização eficaz da API, comprovam que nenhum elemento foi prejudicado na migração das chaves.

Os resultados obtidos atestam o sucesso da migração em todos os aspectos críticos testados, endossando a efetividade do método de migração adotado.

5 CONCLUSÃO

O projeto proposto detalha o processo de migração das versões 2.6 para 4.2 do software WSO2 API Manager em um ambiente de desenvolvimento. A migração via automação pelo navegador possibilitou um atrito mínimo com o banco de dados, elemento extremamente sensível na arquitetura de software.

Foi possível inferir que, para softwares complexos, a abordagem de migração recriando os elementos diretamente na aplicação é mais simples e segura, em relação a outra abordagem na qual fossem feitas manipulações diretamente no banco de dados. Apenas para a substituição de duas chaves pertencentes (*Consumer Key* e *Secret Key*) às *applications*, foram necessárias manipulações em quatro tabelas do banco de dados 'WSO2AM_DB'. Além disso, também seria necessário manipular os outros bancos de dados, o que implicaria uma complexidade considerável.

Foram auferidos todos os objetivos específicos citados na seção 1.1.2. Foi estudada a importância de um catálogo de serviços para os clientes, implementada a nova versão do software API Manager em um ambiente na nuvem, e realizou-se a migração das informações da versão antiga para a nova, com êxito em todos os testes.

O projeto trouxe a documentação do método de migração, sua teoria embasadora, além de credibilidade para aplicação em um contexto real. Um exemplo seria a migração do catálogo de serviços do ambiente de produção, o qual possui um volume de dados consideravelmente maior.

Seguem sugestões de trabalhos futuros que complementem ou sejam inspirados por este, mas não se limitando somente a eles:

- Expansão e otimização do código utilizado para automação, objetivando o seu funcionamento maior flexibilidade;
- Implementação do WSO2 API Manager, com o *software* operando em modo *cluster* ao invés de *single node*, conforme documentação oficial: apim.docs.wso2.com/en/latest/distributed-deployment
- Implementação da monitoração do WSO2 API Manager, em *single node* ou *cluster*, conforme documentação oficial: /on-prem/elk-installation-guide/
- Implementação da arquitetura via contêineres, orquestrados por Kubernetes.

REFERÊNCIAS BIBLIOGRÁFICAS

1 SANTOS, R. A. R. dos. Estudo de catálogo de serviços de uma plataforma de registro eletrônico de saúde. In: DEPARTAMENTO DE ENGENHARIA ELETRICA, UNIVERSIDADE DE BRASILIA. *Monografia de Projeto Final de Graduação*. [S.l.], 2023.

2 IBM. *What is service-oriented architecture (SOA)?* <https://www.ibm.com/topics/soa>. (Accessed on 07/16/2023).

3 AWS. *Arquitetura Orientada a Serviços*. <https://aws.amazon.com/pt/what-is/service-oriented-architecture/>. (Accessed on 07/16/2023).

4 ERL, T. *SOA Design Patterns*. [S.l.]: Prentice Hall, 2009.

5 MAGAZINE, S. W. *Arquitetura Orientada a Serviços*. June 2008. <https://aws.amazon.com/pt/what-is/service-oriented-architecture/>. (Accessed on 07/17/2023).

6 O'LOUGHLIN, M. *The Service Catalog - A Practitioner Guide*. [S.l.]: Van Haren Publishing, 2010.

7 IBM. *What is an API?* <https://www.ibm.com/topics/api>. (Accessed on 07/16/2023).

8 FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. [S.l.]: University of California, Irvine, 2000.

9 WSO2. *GitHub WSO2 Documentation*. <https://github.com/wso2>. (Accessed on 07/16/2023).

10 WSO2. *WSO2*. <https://wso2.com>. (Accessed on 07/16/2023).

11 WSO2. *API Manager Documentation 4.2.0*. 2023. <https://apim.docs.wso2.com/en/latest/get-started/overview/>. (Accessed on 07/16/2023).

12 WSO2. *Identity Server Documentation*. 2023. <https://is.docs.wso2.com/en/latest/get-started/quick-start-guide/>. (Accessed on 16/07/2023).

13 CHROME. *Chrome Dev Tools*. <https://developer.chrome.com/docs/devtools/>. (Accessed on 07/16/2023).

14 POSTGRES. *Postgres JDBC Driver*. <https://jdbc.postgresql.org/download/>. (Accessed on 07/16/2023).

15 WSO2. *WSO2 IS Connector*. <https://apim.docs.wso2.com/en/4.2.0/assets/attachments/administer/wso2is-extensions-1.6.8.zip>. (Accessed on 07/16/2023).

ANEXOS

I. CÓDIGOS UTILIZADOS COMPLETOS

Seguem os códigos-fonte JavaScript e Python utilizados para automação da migração (quebrados para caber no documento):

I.1 AUTOMAÇÃO DO BROWSER COM JAVASCRIPT

```
const puppeteer = require('puppeteer');

let FixAPIKey = '', element = '', Usuario = '', Senha = '',

NomeDaAplicacao = '', IDdaAPI = '', csvData = '', currentIndex = 0;
let UsuarioOld = '', NomeDaAplicacaoOld = '';

const fs = require('fs');
const Papa = require('papaparse');

fs.writeFileSync('chavefixa.csv', 'Usuario,NomeDaAplicacao,FixAPIKey\n');

function insereDadosCSV() {
    let row = `${Usuario},${NomeDaAplicacao},${FixAPIKey}\n`;
    fs.appendFileSync('chavefixa.csv', row);
}

function CarregaVariaveis() {
    // Verifica se atingiu o fim do CSV
    if (currentIndex >= csvData.length) {
        console.log('CSV processing complete.');
        process.exit(0);
    }

    UsuarioOld = Usuario;
    NomeDaAplicacaoOld = NomeDaAplicacao;
    IDdaAPIOld = IDdaAPI;
    let item = csvData[currentIndex];
    Usuario = item.Usuario;
    Senha = item.Senha;
    NomeDaAplicacao = item.NomeDaAplicacao;
```

```
    IDdaAPI = item.IDdaAPI;

    console.log(Usuario, Senha, NomeDaAplicacao, IDdaAPI);

    // Move para o próximo item
    currentIndex++;
}

Papa.parse(fs.createReadStream('./importacao.csv'), {
    header: true,
    complete: function(results) {
        csvData = results.data;
    }
});

async function waitForXAndGetElement(page, xpath) {
    await page.waitForXPath(xpath, { timeout: 10000 });
    const tempElement = await page.$x(xpath);
    return tempElement;
}

const userLoop = async (page) => {
    let formElement;
    await page.goto(`https://{host}:{port}/devportal/apis`,

    { waitUntil: 'networkidle0' });

    await page.evaluate(() => {
        document.getElementById('itest-devportal-sign-in').click();
    });
    await page.waitForNavigation({ timeout: 10000 });
    element = await waitForXAndGetElement(page, `//*[@id="username"]`);
    await element[0].type(Usuario);
    element = await waitForXAndGetElement(page, `//*[@id="password"]`);
    await element[0].type(Senha);
    formElement = await page.$x(`//*[@id="loginForm"]`);
    await page.evaluate(form => form.submit(), formElement[0]);
    await page.waitForNavigation({ timeout: 10000 });
    await page.goto(`https://{host}:

    {port}/devportal/applications`,
```

```
    { waitUntil: 'networkidle0' });
    try {
        await page.waitForXPath(`//*[@id="delete-DefaultApplication-btn"]

        /span[1]/span`, { timeout: 5000 });
        await deleteDefaultApp(page);
    } catch (error) {
        if (error instanceof puppeteer.errors.TimeoutError) {
            console.log("Nao existe DefaultApp nesse usuario");
        } else {
            // Se foi outro erro, relance-o porque algo inesperado aconteceu.
            throw error;
        }
    }
};

const deleteDefaultApp = async (page) => {
    element = await waitForXAndGetElement(page,

    `//*[@id="delete-DefaultApplication-btn"]/span[1]/span`);
    await element[0].click();
    element = await waitForXAndGetElement(page,

    `//*[@id="itest-confirm-application-delete"]/span[1]`);
    await element[0].click();
    await page.waitForTimeout(1000);
};

const AppLoop = async (page) => {
    await page.goto(`https://{host}:{port}/devportal/applications`,

    { waitUntil: 'networkidle0' });

    element = await waitForXAndGetElement(page,

    `//*[@id="itest-application-create-link"]/button/span[1]`);
    await element[0].click();

    element = await waitForXAndGetElement(page,

    `//*[@id="application-name"]`);
    await element[0].type(NomeDaAplicacao);
```

```
element = await waitForXAndGetElement(page,

`//div[@id='per-token-quota']`);
await element[0].click();

element = await waitForXAndGetElement(page,

`//div[@id='menu-throttlingPolicy']/div[3]/ul/li[4]`);
await element[0].click();

await page.evaluate(() => {
    document.getElementById('itest-application-create-save').click();
});
await page.waitForNavigation({ timeout: 10000 });
element = await waitForXAndGetElement(page,

`//*[@id="production-keys"]/p`);
await element[0].click();

element = await waitForXAndGetElement(page,

`//button[@id='generate-keys']/span`);
await element[0].click();

element = await waitForXAndGetElement(page,

`//*[@id="production-keys-apikey"]/p`);
await element[0].click();

element = await waitForXAndGetElement(page,

`//button[@id='generate-key-btn']/span`);
await element[0].click();

element = await waitForXAndGetElement(page,

`//button[@id='generate-api-keys-btn']/span`);
await element[0].click();

element = await waitForXAndGetElement(page, `//*[@id="access-token"]`);
FixAPIKey = await page.evaluate(el => el.textContent, element[0]);
```

```
    insereDadosCSV();
    element = await waitForXAndGetElement(page,

    `//button[@id='generate-api-keys-close-btn']/span`);
    await element[0].click();
};

const APISubscriber = async (page) => {
    await page.evaluate(() => {
        document.getElementById('left-menu-subscriptions').click();
    });
    await page.waitForTimeout(1000);
    element = await waitForXAndGetElement(page,
    `//div[@id='pageRoot']/div/main/div/div/div[2]/div/div/button/span`);
    await element[0].click();
    element = await waitForXAndGetElement(page,

    `//*[@id="pagination-rows"]`);
    await element[0].click();
    element = await waitForXAndGetElement(page,

    `//*[@id="pagination-menu-list"]/li[3]`);
    await element[0].click();
    await page.waitForTimeout(1000);
};

const APILoop = async (page) => {
    await page.evaluate((IDdaAPI) => {
        document.getElementById('policy-subscribe-btn-' + IDdaAPI).click();
    }, IDdaAPI);
    CarregaVariaveis();
    if ((UsuarioOld == Usuario) && (NomeDaAplicacaoOld == NomeDaAplicacao)){
        await APILoop(page);
    }
};

const EndLogout = async (page) => {
    await page.goto(`https://{host}:{port}/devportal/apis`,

    { waitUntil: 'networkidle0' });
    await page.evaluate(() => {
        document.getElementById('userToggleButton').click();
```

```
    });
    await page.waitForTimeout(1000);
    await page.evaluate(() => {
        document.getElementById('logout-link').click();
    });
    await page.waitForNavigation({ timeout: 10000 });
};

(async () => {
    const browser = await puppeteer.launch({ ignoreHTTPSErrors: true,

    headless: false, defaultViewport: { width: 1920, height: 1080 },

    args: ['--start-maximized'] });
    const page = await browser.newPage();

    CarregaVariaveis();

    while(currentIndex <= csvData.length){
        if(UsuarioOld == Usuario){
            await AppLoop(page);
            await APISubscriber(page);
            await APILoop(page);
        }
        else{
            if (UsuarioOld != ''){
                await EndLogout(page);
            }
            await userLoop(page);
            await AppLoop(page);
            await APISubscriber(page);
            await APILoop(page);
        }
    }

    await browser.close();
})();
```

I.2 MIGRAÇÃO DAS CHAVES *CONSUMER KEY* E *SECRET KEY* COM PYTHON

```
import psycopg2
```

```
import csv

# Configurações de conexão ao banco de dados
host = {ARN}
database = "wso2am_db"
user = "postgres"
password = {senha}
port = {port}

# Lista de tabelas e colunas para atualização da consumer_key
tabelas_colunas_consumer = [
    ("public.am_application_key_mapping", "consumer_key"),
    ("public.idn_oauth_consumer_apps", "consumer_key"),
    ("public.idn_oidc_property", "consumer_key"),
    ("public.sp_inbound_auth", "inbound_auth_key")
]

# Tabela e coluna para atualização da consumer_secret
tabela_coluna_secret = ("public.idn_oauth_consumer_apps", "consumer_secret")

# Função para atualizar os valores nas tabelas
def atualizar_valores():
    try:
        # Conecta ao banco de dados
        connection = psycopg2.connect(host=host, database=database,

        user=user, password=password, port=port)

        # Cria um cursor para executar as consultas
        cursor = connection.cursor()

        # Remove a constraint que impede a alteração das keys
        cursor.execute("""
        ALTER TABLE public.idn_oidc_property
        DROP CONSTRAINT idn_oidc_property_consumer_key_fkey;
        """)
        connection.commit()

        # Itera sobre as tabelas e colunas e executa a atualização

        (consumer_key production)
        with open('arquivo_key.csv', 'r') as arquivo:
```

```
    leitor = csv.reader(arquivo)
    next(leitor)  # Pula o cabeçalho
    for linha in leitor:
        valor_antigo_key, valor_novo_key = linha
        for tabela, coluna in tabelas_colunas_consumer:
            query = f"UPDATE {tabela} SET {coluna} =

            REPLACE({coluna}, '{valor_antigo_key}',

            '{valor_novo_key}')"

            cursor.execute(query)
            connection.commit()

# Atualiza a tabela consumer_secret
with open('arquivo_secret.csv', 'r') as arquivo:
    leitor = csv.reader(arquivo)
    next(leitor)  # Pula o cabeçalho
    for linha in leitor:
        valor_antigo_secret, valor_novo_secret = linha
        tabela, coluna = tabela_coluna_secret
        query = f"UPDATE {tabela} SET {coluna} =

        REPLACE({coluna}, '{valor_antigo_secret}',

        '{valor_novo_secret}')"

        cursor.execute(query)
        connection.commit()

# Restabelece a constraint que impede a alteração das keys
cursor.execute("""
ALTER TABLE public.idn_oidc_property
ADD CONSTRAINT idn_oidc_property_consumer_key_fkey
FOREIGN KEY (consumer_key) REFERENCES public.idn_oauth_consumer_apps

(consumer_key) ON DELETE CASCADE;
""")
connection.commit()

# Fecha a conexão ao banco de dados
cursor.close()
```

```
        connection.close()

        print("Atualização concluída com sucesso!")

    except (Exception, psycopg2.Error) as error:
        print("Ocorreu um erro ao atualizar os valores:", error)

# Chamada da função para atualizar os valores
atualizar_valores()
```