Arquitetura de segurança para aplicações clínicas inteligentes de tratamento no domicílio

Felipe Gimenez da Silva - 42469

Dissertação apresentada à Escola Superior de Tecnologia e Gestão de Bragança para obtenção do Grau de Mestre em Informática no âmbito do programa de duplo diploma com a Universidade Tecnológica Federal do Paraná.

Trabalho orientado por:

Prof. Tiago Miguel Ferreira Guimarães Pedrosa

Prof. Paulo Alexandre Vara Alves

Prof. Rogério Ranthum

Esta dissertação não inclui as críticas e sugestões feitas pelo Júri.

Bragança

2021

Arquitetura de segurança para aplicações clínicas inteligentes de tratamento no domicílio

Felipe Gimenez da Silva - 42469

Dissertação apresentada à Escola Superior de Tecnologia e Gestão de Bragança para obtenção do Grau de Mestre em Informática no âmbito do programa de duplo diploma com a Universidade Tecnológica Federal do Paraná.

Trabalho orientado por:

Prof. Tiago Miguel Ferreira Guimarães Pedrosa

Prof. Paulo Alexandre Vara Alves

Prof. Rogério Ranthum

Esta dissertação não inclui as críticas e sugestões feitas pelo Júri.

Bragança

2021

Dedicatória

Dedico este trabalho a minha família, que me incentivou e apoiou em todos os momentos e escolhas. Agradeço principalmente aos meus pais e a minha irmã por também sonharem os meus sonhos.

Agradecimentos

Agradeço aos meus orientadores, Tiago Miguel Ferreira Guimarães Pedrosa e Paulo Alexandre Vara Alves, que acompanharam e me auxiliaram durante toda a elaboração do trabalho. Agradeço também aos meus amigos, em especial o João Pedro Lourenço, Rafael Sprea e Thiago Teixeira por todo apoio que me forneceram. Sou grato pelo apoio que obtive da Universidade Tecnológica Federal do Paraná, e do Instituto Politécnico de Bragança.

Resumo

Este documento aborda a criação de uma arquitetura focada em segurança de dados e de serviços orientada à manipulação de dados clínicos. O objetivo é garantir a integridade, privacidade e disponibilidade de dados médicos, além de prover monitoramento, disponibilidade e escalabilidade dos serviços disponíveis na plataforma. É feita uma análise do estado da arte na área, identificados requisitos de segurança, e levantamento da tecnologia disponível que permita uma implementação deste tipo. Durante o projeto, foi obtido a disponibilidade de serviços e dados através de configurações para clusterização e redundância. O documento também aborda as configurações para automatização em casos de falha de serviço, uso de kubernetes para auxiliar na manutenção e escalabilidade da arquitetura e configurações para a segurança dos componentes do sistema. Como resultado é obtido um modelo de ameaça e as estratégias para as mitigar, além da automatização de monitorias dos serviços e processos de recuperação de falhas. A arquitetura é formada por microsserviços para autenticação, APIs, serviços de monitoramento, roteamento de requisições e clusters de dados, onde foi analisado as características pertencentes a cada um frente á segurança. Também é obtido o modelo de uma arquitetura funcional, as configurações utilizadas, testes de validação das estratégias adotadas e melhorias futuras.

Palavras-chave: Segurança, Nuvem, Dados Clínicos.

Abstract

This document addresses the creation of an architecture focused on data security and services oriented to the manipulation of clinical data. The goal is to ensure the integrity, privacy and availability of medical data, as well as to provide monitoring, availability and scalability of the services available on the platform. An analysis of the state of the art in the area is performed, security requirements are identified, and the available technology that allows such an implementation is surveyed. During the project, the availability of services and data was created through configurations for clustering and redundancy. The document also covers settings for automation in case of service failure, use of kubernetes to aid in the maintenance and scalability of the architecture, and settings for the security of system components. The result is a threat model and strategies to mitigate them, as well as the automation of service monitoring and disaster recovery processes. The architecture is formed by microservices for authentication, APIs, monitoring services, request routing and data clusters, where the characteristics of each one were analyzed in relation to security. The model of a functional architecture, validation tests of the adopted strategies and future improvements are also obtained.

Keywords: Security, Cloud , Clinical Data .

Conteúdo

Lista de Tabelas

Lista de Figuras

Siglas

ABAC Attribute-Based Access Control. 20

ABE Attribute-Based Encription. 17

API Application Programming Interface. xii, 26, 34, 37, 38, 48–50, 59–62, 67, 71–75, 80, 81, 101

BLE Bluetooth Low Energy. 30, 32

CeDRI Research Centre in Digitalization and Intelligent Robotics. 1

CPE Common Plataform Enumeration. 12

CVE Common Vulnerability and Exposure. 12

DDoS Distributed Denial of Service. 81–83

DNS Domain Name System. 59

DREAD Damage, Reproducibility, Exploitability, Affected users, Discoverability. 12

ECDSA Elliptic Curve Digital Signature Algorithm. 39

HIPAA Health insurance portability and accountability act. 8

HMAC Hash-Based Message Authentication Code. 39

HTTP Hypertext Transfer Protocol. 7, 38, 44, 46, 66

HTTPS Hypertext Transfer Protocol Secure. 33, 39, 46

IBE Identity-Based Encryption. 16

IoT Internet of Things. 5–7

IPB Instituto Politécnico de Bragança. 29, 49, 80

IVA Inventory Vulnerability Analysis. 12

JSON JavaScript Object Notation. 38, 42, 45, 66

JWT JSON Web Token. 20, 21, 33, 37–40, 66, 84, 89, 96

MAC Media Access Control Address. 86

MFA multi-factor authentication. 43

MITC man-in-the-cloud. 40

MITM man-in-the-middle. 32, 33

noSQL not-only SQL. 42

OCTAVE Operationality Crítial Threat, Asset and Vulnerability Evaluation. 12

PASTA Process for Attack Simulation and Thread Analisys. 12

RBAC Role-Based Access Control. 20, 48

RSA Rivest–Shamir–Adleman. 39, 68, 86, 100–102

SSD solid-state drives. 10

SSH Secure Shell. 44, 50, 55, 57–59

SSL Secure Sockets Layer. 46, 89

SSO single-sign on. 29, 33, 34, 39, 60

STRIDE Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege. 12, 25, 83, 88

TCP Transmission Control Protocol. 44

TLS Transport Layer Security. 89

TPM Trusted Plataform Module. 20

WSGI Web Server Gateway Interface. 37

xinetd extended internet service deamond. 52, 53, 78, 80

XSS Cross-Site Scripting. 3

YAML Yet Another Markup Language. 45, 48

Capítulo 1

Introdução

Este capítulo é dedicado à introdução do trabalho, onde é abordado o tema, as motivações e objetivos que justificam sua execução. Inicialmente, é descrito o enquadramento do trabalho. Em seguida, é relatado todos os objetivos que são pretendidos alcançar e, ao final, é apresentada a estrutura utilizada no documento.

1.1 Enquadramento

O trabalho atual é destinado ao desenvolvimento de uma plataforma orientada a microsserviços, de modo a que seja fornecida segurança aos serviços, dados e utilizadores. O trabalho encontra-se inserido no projeto NanoStim do Research Centre in Digitalization and Intelligent Robotics (CeDRI). O projeto NanoStim tem como objetivo fornecer tratamento para pacientes com problemas motores sem que haja a necessidade de locomoção a um centro hospitalar [1]. Para isso, dentro do projeto NanoStim é estudado o uso de aplicações móveis, dispositivos eletrônicos e aprendizado máquina como meios facilitadores da comunicação entre profissionais de saúde e paciente, coleta de dados, análise e sugestão de terapia. Esse trabalho trata-se de um estudo voltado à segurança de uma infraestrutura necessária para suportar as necessidades do NanoStim.

A cada dia o uso da tecnologia tem demonstrado suas capacidades de resolver problemas de forma eficiente, tornando-se resposta para as necessidades humanas.

Para que o projeto NanoStim torne-se utilizável em um ambiente real, é necessário alcançar um nível de segurança elevado e para fornecer integridade, privacidade e disponibilidade dos dados. Também é necessário garantir a segurança dos serviços fornecidos, bem como escalabilidade, manutenção e monitoria.

O trabalho documenta as etapas e análises realizadas durante o desenvolvimento da arquitetura para o NanoStim, os serviços utilizados e algumas das suas funcionalidades de segurança.

1.2 Objetivos

O trabalho atual tem o objetivo de prover uma estrutura segura para sustentar o projeto NanoStim, porém não se trata de uma versão final de arquitetura uma vez que as atualizações de funcionalidades e segurança deverão acompanhar todos os processos do ciclo de vida do software. Os objetivos do trabalho podem ser resumidos em:

- Detectar as possíveis ameaças dentro do contexto do projeto NanoStim, bem como as estratégias de mitigação de vulnerabilidades;
- Desenvolver a arquitetura do projeto NanoStim de modo a suportar manutenção e escalabilidade;
- Garantir disponibilidade de serviços e dados;
- Analisar formas seguras de armazenamento de dados;
- Definir métodos de autenticação entre múltiplos serviços da arquitetura.

A segurança, funcionalidade e usabilidade dos sistemas clínicos são as principais características desejadas. Como são aspectos que necessitam de entrar em equilíbrio, é necessário ter em mente as consequências geradas por cada escolha de implementação. Durante o desenvolvimento, é importante possuir uma visão holística da solução afim de maximizar o potencial do sistema.

Um dos desafios é garantir o funcionamento dos principais serviços do sistema. Arquiteturas com a abordagem de microsserviços são compostas por diversos serviços específicos e independentes, e que ao se unirem fornecem todas as funcionalidades. O ponto fraco da arquitetura de microsserviços é quando ocorre falha de um serviço que é necessário a outros, causando problemas nos serviços vizinhos e propagando erros pela plataforma.

Garantir que os utilizadores não serão privados de suas funções dentro do sistema também apresenta-se como um desafio. Os utilizadores, profissional de saúde e paciente, possuem acessos a funcionalidades específicas dentro da plataforma. Um ativo do sistema que não funcione corretamente pode, por exemplo, privar o paciente de seu tratamento, ou um profissional de saúde de gerar sessões de tratamento.

A plataforma deve confiar em suas informações, mas é uma tarefa difícil garantir que os dados armazenados estejam realmente corretos. Os meios de transmissão de dados, assim como os equipamentos dos utilizadores e da própria plataforma são susceptíveis a ataques. Dessa forma, há diversos casos a serem analisados durante todo o ciclo de vida da plataforma.

Os dados deverão ser acessados apenas por utilizadores que realmente tem permissão para isso. A definição de permissões de utilizadores é um processo sensível, uma vez que ataques com foco na elevação de privilégios podem ocorrer.

A plataforma deve ser funcional, porém prevenir-se contra o mal uso das funcionalidades fornecidas. Através de usos intencionalmente errôneos, pode ocorrer ataques à empresa, empregados e pacientes que não se restringem apenas ao sistema sendo desenvolvido. Roubo de tokens de acesso de outras aplicações, roubo de dados, Cross-Site Scripting (XSS), e diversos outros objetivos podem motivar o atacante a explorar o mal uso dos recursos da plataforma.

O trabalho também tem o objetivo de documentar o desenvolvimento da arquitetura, com as estratégias utilizadas para facilitar a escalabilidade horizontal do sistema, manutenção, controle de utilizadores, permissões e monitoramento de serviços.

A detecção de possíveis ameaças ao projeto NanoStim está inclusa como objetivo deste trabalho, afim de prover avanços na segurança do projeto através da mitigação de riscos.

1.3 Estrutura do Documento

O documento foi subdividido em 6 capítulos. O capítulo atual apresenta a introdução, o enquadramento e os objetivos que se pretende alcançar. O restante do documento do documento está organizado da seguinte forma:

- no capítulo 2 é apresentado o estado da arte, entrando em assuntos que envolvem computação em nuvem, segurança de sistemas hospitalares, modelagem de ameaças, criptografia e credenciamento digital;
- no capítulo 3 é abordado os requisitos necessários da plataforma, a proposta e os componentes que envolvem o projeto NanoStim;
- o capítulo 4 é focado no desenvolvimento do projeto, trazendo informações sobre as tecnologias utilizadas, detalhes da arquitetura e as etapas para implementar e integrar os serviços da plataforma;
- o capítulo 5 traz os testes de disponibilidade dos principais serviços, um teste de funcionalidade das estratégias de comunicação adotadas e modelagem de ameaças do projeto;
- o capítulo 6 apresenta as conclusões e direções de trabalho futuro.

Ao final do documento, na seção de anexos, são disponibilizadas as configurações utilizadas para realizar a implementação dos serviços. Também encontram-se alguns dos *scripts* utilizados para facilitar a configuração de integração dos serviços.

Capítulo 2

Estado da arte

Neste capítulo é apresentado os principais conceitos que foram levados em consideração durante as etapas de planejamento, análise e desenvolvimento deste trabalho. Inicialmente, é feito uma breve introdução aos conceitos que envolvem computação em nuvem e suas conexões com bigdata, Internet of Things (IoT) e microsserviços. Em seguida, é abordado os aspectos de segurança da informação ao desenvolver softwares com foco na área da saúde, sendo apresentado algumas características desejáveis e exigências da lei Health Insurance Portability and Accountability Act, formulada para sistemas clínicos. É apresentado as principais características e abordagens presentes em modelagens de ameaça. Para assegurar a proteção de dados, é brevemente apresentado conceitos de criptografia, características das mesmas e uso potencial.

2.1 Computação em nuvem

A computação em nuvem é um modelo de computação que compartilha recursos de modo configurável, permitindo acesso onipresente, conveniente e sob demanda [2]. Com isso é possível realizar tarefas amenizando desperdício de recursos e maximizando o desempenho. O poder computacional produzido por sistemas computação em nuvem é amplamente utilizado em cenários empresariais, e em sistemas com grandes demandas, como ocorre em *big data*.

Por conta de suas características, a computação em nuvem tem-se tornado popular para os problemas da atualidade. Há diversas plataformas que oferecem serviços com alto grau de escalabilidade, custos de acordo com a necessidade de uso e com menores dificuldades aos programadores, como a AWS, Azure, Google Cloud, IBM Cloud e Alibaba Cloud. Dessa forma, a computação em nuvem tem se mostrado uma solução viável para as empresas.

A segurança em ambientes na nuvem ainda encontra-se em seu estágio inicial [3]. É desafiador explorar segurança, balanceamento e eficiência energética entre dispositivos médicos e servidores em nuvem [4]. Ao aumentar a quantidade de e variabilidade de dispositivos, torna-se mais difícil evitar vazamento de dados, além de maiores gastos devido a operações criptográficas e comunicação.

A implementação de meios de segurança em sistemas em nuvem podem influenciar em seu tempo de resposta e custo. É necessário elaborar estratégias para garantir a segurança dos usuários, e ao mesmo tempo equilibrar o preço para manter o sistema ativo, levando em conta que os serviços ainda devem possuir um tempo de resposta aceitável para responder às requisições.

A computação em nuvem corresponde ao desafio de suprir grandes demandas de armazenamento de dados encontrados em '*bigdata*' e as evoluções obtidas contribuem para o avanço da medicina e de prestação de cuidados à saúde com maior apoio tecnológico [5].

As principais características de '*bigdata*' são alta velocidade de geração e analise de informação, alto volume e grande variedade de dados. As características de *bigdata* abrem grandes possibilidades, promovendo novas estratégias para sistemas complexos e inteligentes, como ocorre no caso de sistemas de *Internef Of Things* [6].

IoT tem sido muito usado em sistemas de saúde para melhorar a eficiência no acompanhamento dos pacientes e podem ser aplicados em conjunto com a computação em nuvem para lidar com dados médicos. Porém, ainda é essencial a segurança, baixa latência e baixo consumo de energia. Poucas pesquisas foram envolvidas na melhoria da segurança da transmissão de dados no sistema IoT [7].

A utilização de computação em nuvem centralizada pode se tornar um problema no cenário de IoT quando os dispositivos conectados estão longe geograficamente dos servidores físicos [8]. O problema se agrava com o crescimento da demanda de requisições e armazenamentos, que é uma das caraterísticas de *bigdata*. Como extensão à computação em nuvem, pode ser utilizado *Fog Computing* (computação em névoa) ou *Edge computing* (computação em borda) com o objetivo de atingir alto desempenho em casos de demandas em longas distâncias.

Para a criação de sistemas na nuvem, é utilizado abordagens orientadas em microsserviços.

2.1.1 Arquitetura de microsserviços

Uma arquitetura orientada a micro-serviços é caracterizada pela criação de uma aplicação constituída por um conjunto de serviços pequenos e independentes. Nesse tipo de arquitetura, todos os serviços possuem o seu próprio processo, desempenhando tarefas específicas e sendo capaz de se comunicar com outros serviços através de protocolos leves, como Hypertext Transfer Protocol (HTTP). A abordagem é contrária ao desenvolvimento de sistemas monolíticos [9]. A comunicação de diversos serviços na nuvem são capazes de atingir um objetivo específico conjunto, promovendo melhores práticas de manutenção, atualização e monitoramento de partes isoladas da solução.

A arquitetura traz diversos benefícios relacionados com requisitos funcionais como disponibilidade, confiabilidade, facilidade de manutenção, desempenho e testabilidade de cada serviço. Porém, devido a complexidade gerada pela comunicação entre os múltiplos serviços, torna-se um desafio adicional garantir que os dados enviados sejam armazenados com a devida segurança e fornecer mecanismos de autenticação com serviços de terceiros [10].

Os serviços podem ser caracterizados como stateless, ou como stateful. Serviços stateless não necessitam de armazenar estados para uso futuro. Ao contrário dos serviços stateless, as implementações caracterizadas como stateful necessitam de armazenamento

de estados. Serviços que necessitem recuperar o contexto para realizar operações são exemplos de aplicações stateful, como bancos de dados e caches.

O uso da abordagem de micro serviços também incrementa os esforços para orquestrar todos os componentes contidos na solução desenvolvida [10]. Os serviços em nuvem tem seu foco principal em implementações de serviços stateless, que podem ser destruídos e criados a qualquer momento (serviços efêmeros), fazendo economias de processamento e energia durante períodos de baixa quantidade de requisições e criando múltiplas instâncias em picos de requisições para garantir um bom tempo de resposta. A elasticidade da nuvem é a característica que permite a alteração do número de serviços dinamicamente. Desta forma a conectividade entre os serviços também se torna alvo de alterações constantes, e devem suportar o dinamismo da nuvem.

O uso de serviços stateful possuem grande limitação ao ser implementado manualmente em plataformas elásticas da nuvem. Serviços com caches e estados, como banco de dados, não podem ser simplesmente destruídos e criados dinamicamente, pois poderia acarretar em falhas na consistência dos dados, perdas de informação.

2.2 Segurança da informação na área da saúde

Para que o armazenamento e manipulação dos dados estejam de acordo com a lei Health insurance portability and accountability act (HIPAA), foi definido pontos de segurança para sistemas médicos [11]. A lei tem o objetivo de promover práticas seguras para qualquer entidade médica que cria, mantém, transmite ou recebe dados, além de promover a proteção antecipada a possíveis ameaças e acessos indevidos à informação.

Como obrigação, a entidade deve prevenir fraudes durante o armazenamento de dados, prezando pela integridade dos registros, a fim de evitar alterações indevidas ou perda das informações. Desse modo, a manipulação dos dados deve ser feito apenas por usuários confiáveis e com as devidas permissões [11]. A alteração indevida dos dados contidos em sistemas médicos pode acarretar no tratamento incorreto de pacientes, históricos não confiáveis, documentos falsos, além de possibilitar diversos tipos de falhas.

A entidade deve garantir confidencialidade dos dados, não permitindo o acesso ou divulgação das informações contidas no sistema sem as devidas autorizações [11].

Outra característica importante visada pela lei é a disponibilidade das informações. A entidade deve garantir a possibilidade de acesso ás informações a qualquer momento. Com isso, é necessário que haja um planejamento de recuperação de dados em casos de falhas, ataques ou desastres naturais, além de estratégias para a minimização de crises [11].

Os requisitos de arquiteturas com foco na segurança e privacidade também devem possuir as características [5]:

- Autenticação de usuários e serviços;
- Autorização refinada e controle de acesso;
- Anonimidade dos usuários;
- Confidencialidade dos privilégios de acessos de cada usuário;
- Direito de revogar acesso aos dados médicos quando necessário;
- Exceções emergenciais ao acesso de dados médicos;
- Escalabilidade das medidas de segurança;
- Grandes períodos de arquivamento e disponibilidade de registros médicos;
- Eficiência e usabilidade.

Ao coletar as informações em tempo real de cada interação com uma aplicação, é gerado um grande fluxo de dados que devem ser processados e armazenados de modo eficiente. Sistemas hospitalares geram grandes fluxos de informação durante o monitoramento de seus pacientes.

O *streaming* de dados permite a realização de análises em tempo real, identificação de padrões, predição e uso de sistemas complexos de comunicação. Porém, a grande quantidade de dados se torna um desafio para ser armazenado localmente. Já que as soluções

implementadas em ambientes na nuvem têm como principal característica a escalabilidade, os ambientes devem ser capazes de suportar o aumento da vazão e de armazenamento de informações. O aumento de operações não pode se tornar uma barreira em sistemas que permitem escalabilidade horizontal.

O armazenamento de dados contidos em *streams* pode servir como um histórico de ocorrências, sendo importante em casos hospitalares para rever comportamentos de pacientes mediante um tipo de tratamento, monitorar equipamentos, detectar falhas técnicas e analisar comportamentos suspeitos no uso do sistema. Além disso, é desejável que uma plataforma qualquer possibilite o processamento da stream e o armazenamento dos dados originais paralelamente, promovendo análises em tempo real sem criar atrasos na escrita dos dados.

Para amenizar as dificuldades geradas pela quantidade de informação em termos de agilidade no armazenamento, é possível encontrar serviços de armazenamento distribuído e não relacional, além de optar pelo uso de hardwares específicos para alto desempenho, como solid-state drives (SSD). A utilização de armazenamento distribuídos e não relacionais promovem maior independência entre operações de manutenção de dados quando comparado aos bancos relacionais, enquanto que a alta escalabilidade de bancos distribuídos, em conjunto com o desempenho de leitura e escrita em SSD's, se tornam uma escolha efetiva para problemas com grandes quantidades de operações no banco de dados. Essas características são encontrados nem soluções como AWS DynamoDB, que é focado em alta performance e escalabilidade [12].

É importante criar backups dos dados com frequência, para que seja possível recuperar todos, ou quase todos. Mesmo assim, é importante garantir que os backups estejam salvos de modo seguro para dificultar o acesso indevido às informações.

Durante o funcionamento da plataforma, também pode ocorrer erros que tornam o banco de dados inacessível por tempo indeterminado. Para que a disponibilidade do sistema não seja afetado de forma grave, pode-se configurar réplicas do banco principal para atuar de forma a efetuar substituição rápida do banco principal, além de dividir a carga de operações em múltiplos hosts.

O uso de réplicas pode causar um impacto negativo em termos de tempo de resposta de uma escrita de dados, pois os hosts devem entrar em consenso durante a gravação de um dado, como ocorre em casos de dados síncronos. A distância entre os servidores também influenciam no tempo gasto, devido a trocas de mensagens, porém idealmente os hosts devem estar em localidades geográficas diferentes para que não sejam afetados pelos mesmos problemas como desastres naturais, quedas de energia, roubos e ataques locais.

Em caso de roubo dos arquivos do sistema de banco de dados, ou ainda do hardware utilizado para realizar o armazenamento de dados, deve-se tomar precauções para dificultar a leitura das informações. Os sistemas de banco de dados podem oferecer opções para salvar os dados já criptografados, conhecido como criptografia em repouso. Assim, o atacante que obteve os arquivos necessitará de conhecer ou descobrir o segredo capaz de descriptografar os dados.

2.3 Modelos de identificação de ameaça

Ataques ocorridos na internet estão se tornando sofisticados e simultaneamente requerem menos conhecimentos por parte do atacante [13]. Atualmente é possível obter *scripts* de outros desenvolvedores para performar ataques sem que haja necessariamente o conhecimento dos processos que envolvem o ataque, tornando acessível até mesmo aos utilizadores inexperientes. Assim, torna-se necessário utilizar de técnicas para a detecção de ameaças contidas dentro de sistemas e softwares [13].

A modelagem de ameaças é sistemática e caracteriza os potenciais riscos ao sistema [14]. Após detectar os pontos fracos, é importante descobrir a possibilidade de ocorrência de cada falha, priorizando as que possuem maiores probabilidades de serem exploradas.

As abordagens principais para que seja realizada a modelação de ameaças são geralmente classificadas em abordagens centradas em ataques, centrada em sistema, centradas em ativos [14], [15] e centradas em dados [15]. A Figura 2.1 demonstra os principais atributos de cada abordagem.

Os ativos são os artefatos relevantes para a resolução dos problemas relacionados ao

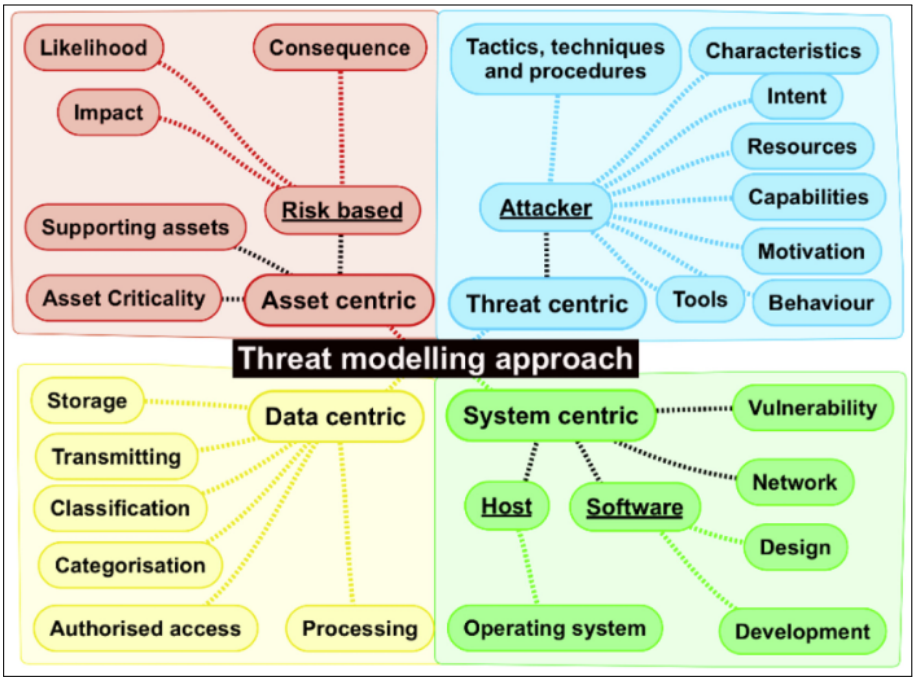

Figura 2.1: Abordagens de modelação de ameaças [15]

funcionamento do sistemas, podendo ser definidos como os objetos físicos, informações, códigos que compõe a solução [14] e as capacidades funcionais. A identificação de todos os ativos é uma tarefa difícil, porém constitui o passo inicial para a elaboração da modelagem de segurança [14]. Ao identificar as relações e interações entre os múltiplos ativos, é possível realizar as análises referentes às medidas de segurança que podem ser adotadas de modo mais detalhado. O aumento de ativos dentro dos sistemas pode provocar o crescimento no número de possíveis vulnerabilidades [14], além de dificultar a análise.

Para auxiliar na identificação de vulnerabilidades, há diversos repositórios que divulgam informações relacionadas a erros de configuração ou vulnerabilidades, como o Common Plataform Enumeration (CPE), Common Vulnerability and Exposure (CVE) e Inventory Vulnerability Analysis (IVA) [15].

Alguns dos modelos de identificação de ameaça mais comuns são Damage, Reproducibility, Exploitability, Affected users, Discoverability (DREAD), TRIKE, Operationality Crítial Threat, Asset and Vulnerability Evaluation (OCTAVE), Process for Attack Simulation and Thread Analisys (PASTA) [14], Spoofing, Tampering, Repudiation, Information

Disclosure, Denial of Service, Elevation of Privilege (STRIDE) e Tree Threat [13].

As abordagens centradas em ativos envolvem a análise para detectar falhas de informação e também realiza análises sobre os impactos sofridos na empresa [14]. Cada ameaça produz diferentes consequências para a empresa, podendo apresentar maior ou menor grau de severidade. Durante a modelagem, pode-se incluir informações como detectar motivos que levariam o atacante a realizar uma determinada ação [14].

As abordagens centradas no sistema geralmente envolvem o design da arquitetura, a rede utilizada para a comunicação, o sistema que é utilizado. Diagramas de fluxo de dados e de componentes auxiliam durante o desenvolvimento.

Quando a análise parte da abordagem centrada a ameaça, os principais objetivos é compreender as decisões tomadas pelo atacante como forma de simular um ataque. Essa análise busca definir quais as motivações que levariam um atacante a optar por atacar um determinado ativo, quais ataques possívelmente seriam aplicados, as características e conhecimentos necessários para fazê-lo e detectar ferramentas que poderiam ser utilizadas para isso.

Quando o foco principal está na análise dos problemas que prejudicam diretamente os dados, a abordagem que deve ser utilizada é a centrada em dados. Ela detecta problemas durante a transmissão, processamento, armazenamento, e autorização de acesso.

Para caracterizar os métodos disponíveis de modelagem de ameaça disponíveis, Matt Tatam *et al.* [15] utilizam as seguintes métricas: representação formal ou gráfica para a análise, e processo manual ou automático, como é mostrado na Figura 2.2.

Matt Tatam *et al.* também fazem a distinção de cada um dos modelos de identificação de ameaças pertencentes às quatro abordagens. As modelagens que compõe as abordagens centradas em dados e sistema são mostradas na Figura 2.3, e as abordagens com foco em ativos e em ameaças são representadas na 2.4.

Ao definir formas de mitigação das vulnerabilidades descobertas, é importante detectar os impactos causados no sistema pelas escolhas elaboradas. A inclusão de novos serviços e hardwares voltados à segurança devem contribuir para sanar ou reduzir as probabilidades de ocorrência das ameaças. Porém é comum que a adição dos novos recursos também

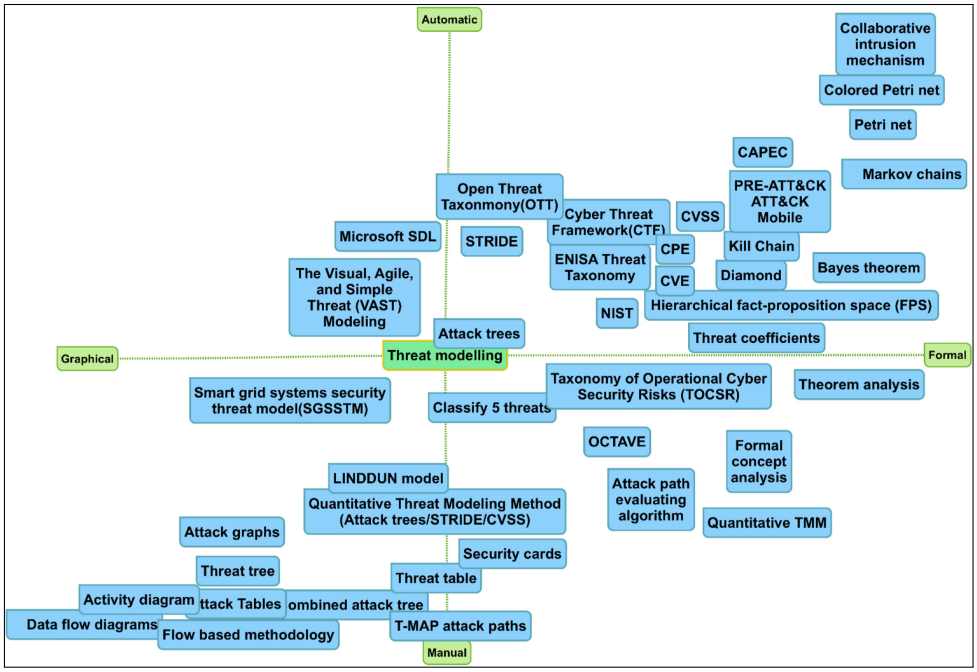

Figura 2.2: Caracterização de métodos para modelação de ameaças [15]

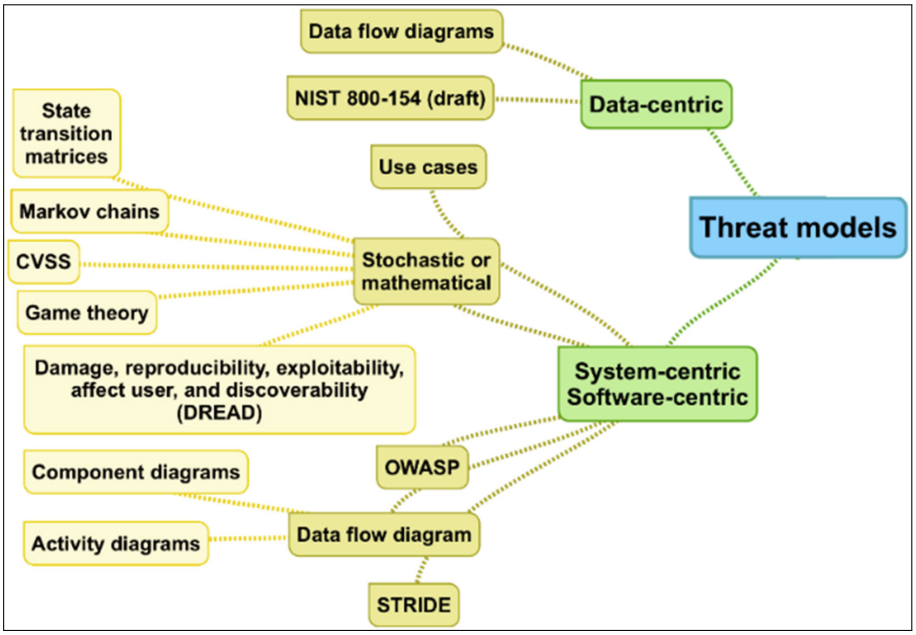

Figura 2.3: Modelagens centradas em dados e sistema [15]

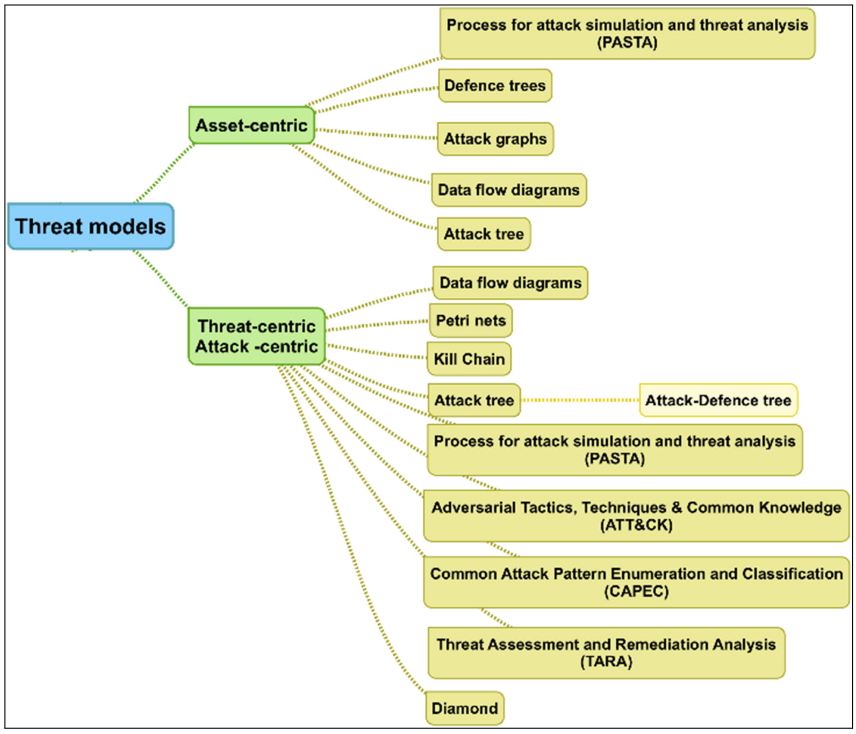

Figura 2.4: Modelagens centradas em ameaça e ativos [15]

necessitem de certos níveis de proteção de modo a aumentar a segurança, e para esse caso é necessário analisar as prioridades e a viabilidade de implementação.

2.4 Criptografia

Ao deixar o ambiente interno da nuvem, os dados se tornam vulneráveis, pois podem estar sendo capturados durante a comunicação entre dispositivos. Para que a segurança seja incrementada, uma das técnicas a ser aplicada é a encriptação de dados [16]. A essência da criptografia está em converter um dado simples a fim de obter uma informação ilegível, cifrado [16] [17]. Dessa forma, caso um indivíduo obtenha o texto cifrado sem as devidas permissões, ainda existirá uma dificuldade para que seja realizada a recuperação dos dados através do processos de decrepitação.

A codificação dos dados pode ser feita através da criptografia simétrica ou assimétrica. A criptografia simétrica tem seu foco em garantir a proteção de uma informação sensível

utilizando apenas uma chave privada (*private-key*) em comum para o emissor e receptor dos dados, representado na Figura 2.5. Na criptografia simétrica, qualquer indivíduo que participe da comunicação necessita conhecer previamente a chave privada. e dessa forma, todos com o conhecimento da chave serão capazes de criptografar e descriptografar informações. A partilha de chaves privadas se torna um ponto susceptível a falhas. É inconveniente o uso de chaves simétricas quando o segredo deve ser compartilhado entre muitos utilizadores [16].

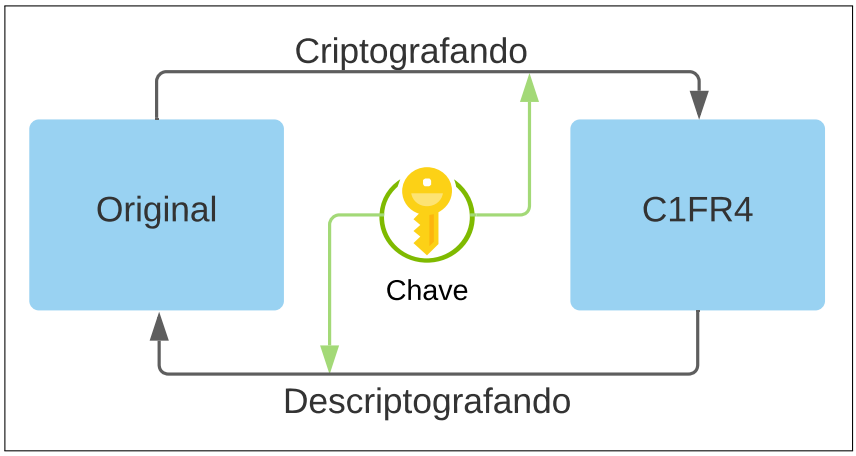

Figura 2.5: Criptografia simétrica

A diferença para a criptografia assimétrica está no uso de chaves públicas (*public-keys*) e chaves privadas para realizar a proteção [17], representado na figura 2.6. A quantidade de bits de uma chave é o parâmetro mais importante para a segurança durante a comunicação [17].

A criptografia assimétrica estabelece que todos os utilizadores devem conhecer a chave pública do receptor, além de possuir suas próprias chaves publicas e privadas. Desse modo, todos os utilizadores conseguem criptografar as mensagem com o uso da chave pública do receptor, mas não são capazes de realizar a descriptografia. Assim, apenas o receptor que têm acesso a chave privada conseguirá descriptografar o conteúdo cifrado e ler as informações.

As tecnologias de encriptação amplamente utilizadas em sistemas nuvem podem consolidar-se em estratégias focadas em identidade,Identity-Based Encryption (IBE), encriptação

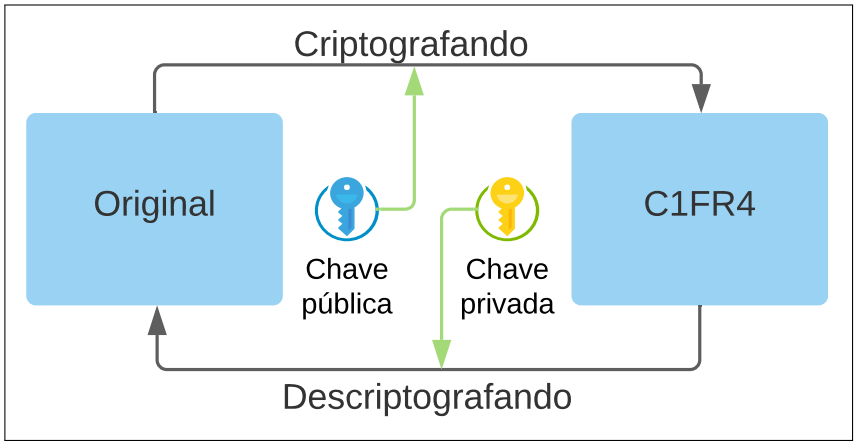

Figura 2.6: Criptografia assimétrica

baseada em atributo, Attribute-Based Encription (ABE) , encriptação homomórfica (*Homomorphic Encription*) e encriptação buscável (*Searchable Encryption*) [16].

2.4.1 Hash

Hash é um algoritmo de criptografia que possui a capacidade de gerar uma sequência de caracteres de tamanho fixo. Para que isso ocorra, os algoritmos de hash recebem uma sequência de caracteres para gerar a saída correspondente, representada pela Figura 2.7. Após utilizar o algoritmo, não há processo de descriptografia, por isso é conhecido como *one-way function* [18].

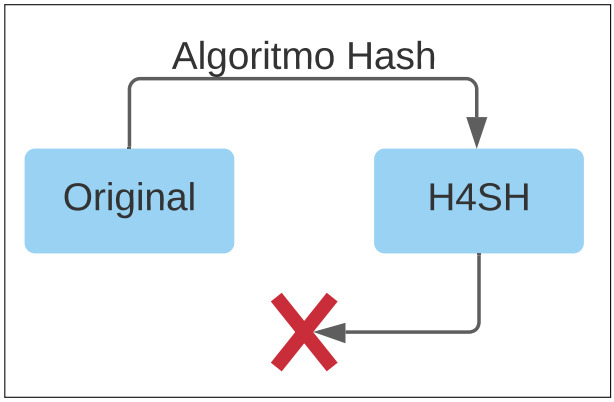

Figura 2.7: Representação simples do funcionamento de algoritmos de hash.

Atualmente, o processo para recuperar a informação é conhecido como força bruta.

Em outras palavras, é necessário aplicar o algoritmo hash em diversas strings até obter o mesmo código da informação a ser recuperada. Quando os códigos hash obtidos são os mesmos, há chance de ter obtido a informação que gerou o hash originalmente, ou que tenha encontrado uma colisão. O processo de força bruta é representado pela Figura 2.8.

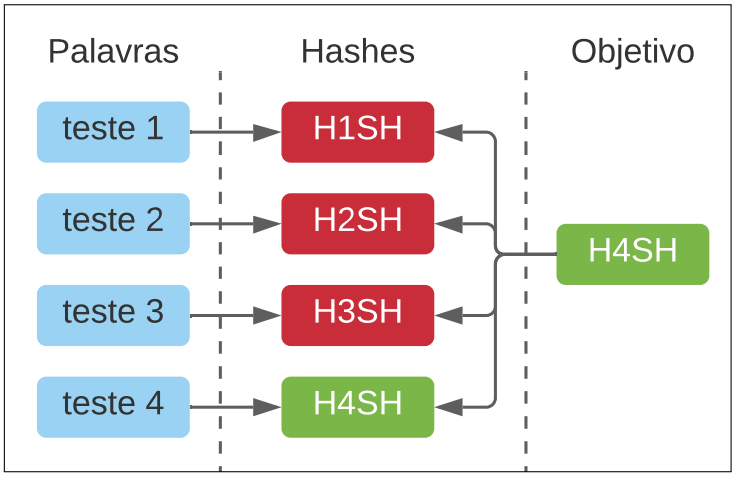

Figura 2.8: Reversão de códigos hash por tentativa e erro.

A colisão de Hash ocorre quando sequências de caracteres diferentes possuem o mesmo código hash ao serem processados por um algoritmo. Quanto menor a colisão, melhor a qualidade do algoritmo hash. Caso um algoritmo possua grandes quantidades de colisões, a chance de acessar um recurso utilizando um segredo inválido é maior.

Novos algoritmos de Hash tem surgido com o tempo para substituir suas versões anteriores, apresentando melhorias em termos de segurança. Entre eles, há os algoritmos SHA2, SHA3, BCrypt e outros [18]

Os algoritmos de Hash são utilizados dentro de sistemas para possibilitar o processo de autenticação por senha, sem armazenar a senha real dos utilizadores, mas representando-as com uma nova sequência de caracteres [18].

Apenas o uso de Hash não garante prevenção contra ataques, como força bruta, dicionários e e tabelas arco-íris. Ataques de força bruta testam todas as possibilidades de senhas até obter a resposta desejada de um sistema. O ataque pode ser optimizado quando o atacante tem o conhecimento de informações sobre a senha, como a quantidade

de caracteres, algumas letras contidas ou símbolos.

Em ataques que utilizam dicionários, o atacante possui uma coleção de senhas padrão, ou ainda a combinação de diversas palavras que poderiam gerar a senha do alvo. É feito o teste com cada uma das possíveis senhas.

Um exemplo de ataque em grande escala são as tabelas arco-íris (rainbow tables). Para performar o ataque, o hacker necessita de um dicionário com strings pré processadas, já tendo o conhecimento de cada uma das strings originais utilizada para gerar os hashs. Assim, o atacante faz a busca no banco de dados por uma sequência hash que já é conhecida, recolhendo todos os utilizadores que contém a senha buscada.

A adição de um texto aleatório , chamado *salt*, reforça as medidas de segurança [18]. O *salt* é uma sequência de letras preferencialmente aleatória, tornando as senhas maiores, e mais imprevisíveis. Ao fazer uso do *salt*, hashes pré computados e as tabelas arco-íris são evitadas, mas mesmo assim, seu uso não garante segurança contra ataques de força bruta e dicionários [18]. O ataque de força bruta é mais lento que a tabela arco-íris, tornando mais custoso ao atacante a obtenção das senhas de múltiplos utilizadores.

2.5 Credenciais

As credenciais são utilizadas para identificar um componente do sistema e pessoas, ou ainda liberar funcionalidades que necessitem de privilégios. Com isso, é possível diferenciar as tarefas que cada componente ou utilizador tem permissão de realizar durante a interação com o sistema.

A autenticação é a profunda convicção na validade da transmissão, mensagem ou validade do emissor, de modo genuíno, verificável e confiável [18].

A identificação de um utilizador autêntico ocorre quando:

- o utilizador prova que é autêntico por meio de características pessoais, como reconhecimento de impressões digitais e reconhecimento de íris;
- o utilizador prova que tem algo autêntico, como o dispositivo para liberar acesso,

cartões, documentos ;

- o utilizador prova que sabe um segredo que qualquer imitador não teria conhecimento, como ocorre ao informar senhas;

Para armazenar segredos no sistema, há cuidados que deverão ser tomados. É importante armazenar as senhas de modo não legível e que não permita recuperação. Essas características podem ser obtidas através do uso de algoritmos de Hash e uso de *salt*.

Para incrementar a segurança, além do uso de senhas, é possível configurar estratégias como two factor authentication (2FA). Assim, após informar o identificador e senha, será solicitado ao utilizador um código temporário que é fornecido por outro recurso, como o telemóvel. A autenticação só é realizada após a inserção do código.

Em arquiteturas com a abordagem de micro serviços, as credenciais e autorizações de cada um dos utilizadores devem ser reconhecidas por múltiplos serviços de modo seguro e prático. Desta forma, atualmente mostra-se vantajoso utilizar estratégias stateless, como JSON Web Token (JWT) com algoritmos de criptografia assimétricos, para a validação de credencias de forma a não impactar a independência de cada micro-serviço.

2.5.1 Autenticação de Serviços e Dispositivos

Devido a terceirização de computação ocorrida em *Edge Computing* e dos múltiplos serviços trabalhando em conjunto na nuvem, é importante garantir modos eficientes para controle de acessos e permissões de cada funcionalidade. A distribuição da computação no cenário de *edge computing* provê um desempenho superior, mas é necessário assegurar a autenticidade do dispositivo. A adoção de uma estratégia de autenticação eficiente desfavorece o uso mal intencionado dos recursos disponíveis da nuvem [19].

Como a *Edge computing* possibilitar um ecossistema de diferentes infraestruturas coexistindo, torna-se essencial o uso de controles de acesso refinados de modo adequado e flexível para múltiplos domínios confiáveis. Podem ser utilizadas diversas estratégias centradas no uso da criptografia, como Role-Based Access Control (RBAC), Attribute-Based Access Control (ABAC) e Trusted Plataform Module (TPM) [19].

Para a autenticação de equipamentos, dispositivos e componentes do sistema, pode ser utilizado estratégias de criptografia assimétrica, como a geração de chaves publicas e privadas, além do uso de JWT. Assim, é possível reconhecer as informações emitidas por uma entidade através do uso de sua chave publica.

Capítulo 3

Arquitetura da Solução

O capítulo atual tem como objetivo introduzir as necessidades do projeto. Esse trabalho trata-se de um fragmento do projeto NanoStim [1], e por isso é abordado aspectos relacionados a segurança de dados, segurança da arquitetura e a proposta elaborada para satisfazer os critérios exigidos pelo projeto.

3.1 Segurança da arquitetura

Esta seção traz as informações referentes aos requisitos de segurança utilizados para elaborar a arquitetura. Inicialmente, é tratado sobre as características e funcionalidades que se pretendia atingir com a arquitetura proposta. Em seguida, é comentado mais detalhadamente sobre a segurança dos dados e sobre o processo de pseudonimização.

3.1.1 Análise de requisitos de segurança

Como objetivo principal de segurança, destaca-se a garantia de segurança de dados. Uma vez que os dados hospitalares podem provocar tratamentos e decisões irreversíveis, além da exposição de informações sensíveis, é necessário contemplar primariamente a integridade, disponibilidade e privacidade das informações. A pseudonimização permite que os dados sejam utilizados por terceiros sem referenciar diretamente os dados pessoais de doutores

e pacientes.

A segurança da arquitetura não deve inutilizar o funcionamento do sistema. Há uma relação estabelecida entre a segurança, a facilidade de uso e as funcionalidades oferecidas por um sistema qualquer, representado pela Figura 3.1. Mesmo sendo fundamental, a segurança do sistema não pode impossibilitar as funcionalidades que motivam o projeto, e não deve dificultar a interação com utilizadores de diferentes níveis de aptidão tecnológica.

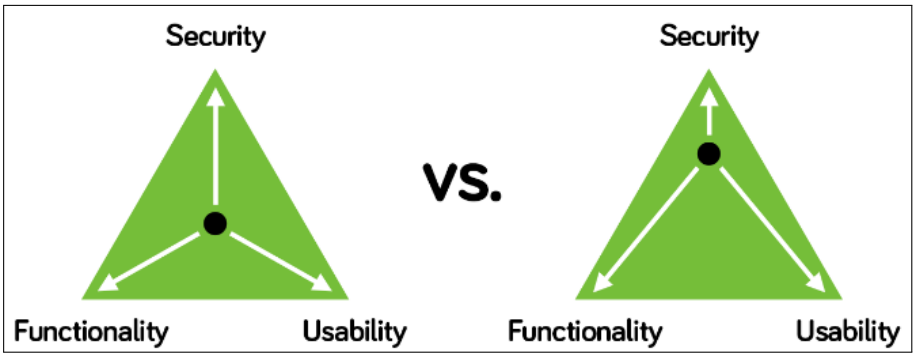

Figura 3.1: Relação entre segurança, funcionalidade e usabilidade [20]

Os requisitos funcionais da plataforma são:

- Fornecer meios para realizar a autenticação de pessoal médico e pacientes em múltiplos serviços;
- Permitir configurações para habilitar o uso de dois fatores durante a fase de autenticação;
- Permitir aos utilizadores editarem as permissões de acesso a aplicativos externos;
- Gerenciamento de sessões ativas, sendo possível revogar as ações que possuam comportamento suspeito;
- Gerenciamento de utilizadores e permissões, para realizar a manutenção dos tipos de utilizadores que interagem com o sistema, e seus respectivos papéis;
- Clusterização de serviços principais, garantindo o funcionamento do conjunto inteiro quando ocorre uma falha isolada;

- Replicação das bases de dados, garantindo a entrega dos dados mesmo que uma instância dos serviços de armazenamento apresentem mal funcionamento;
- Monitoramento de clusters de dados e serviços, para manter os administradores atualizados das situações ocorridas nos clusters;
- Pseudonimização de tratamentos dos pacientes, para integração com serviços que podem ter acesso a dados sem que detectem o utilizador a qual os dados pertencem.

Os requisitos não funcionais da plataforma são:

- Segurança ao acesso indevido de informações armazenadas nas bases de dados;
- Disponibilidade de dados e serviços fornecidos pela arquitetura;
- Garantir integridade das informações armazenadas;
- Acesso facilitado aos serviços internos através de autenticação centralizada;
- Simplicidade de escalabilidade horizontal;
- Facilidade para a manutenção de serviços.

3.1.2 Segurança de dados

A segurança de dados tem sua base formada em três pilares principais, a integridade das informações, a disponibilidade, e a privacidade. A integridade das informações é a propriedade capaz de fornecer confiança nos dados contidos no sistema, tornando assim a base de dados como fonte de verdade. A disponibilidade está relacionada à possibilidade de obter a informação quando necessário. Por último, a privacidade dos dados possui o objetivo de garantir acesso apenas aos utilizadores autorizados.

A segurança de dados foi elaborada tendo em mente os componentes da modelagem de ameaça STRIDE [13]:

- Spoofing, onde o atacante possui personifica outras identidades;

- Tampering, que ocorre ao realizar a modificação indevida dos dados do sistema;
- Repudiation, quando um utilizador declara não ser responsável por um determinado comportamento dentro do sistema;
- Information Disclosure, quando ocorre a divulgação não autorizada de informações;
- Denial of Service, afetando a disponibilidade do sistema, tornando temporariamente ou definitivamente indisponível para um ou mais utilizadores;
- Elevation of privilege, obtendo autorização indevida dentro do sistema.

Para amenizar os problemas causados por mau uso de identidades, foram utilizados protocolos e serviços já consolidados para a realização da autenticação em microsserviços, além da inclusão de estratégias de pseudonimização. As ameaças envolvendo *tampering* foram reduzidas através da adição de autenticação nos serviços de armazenamento de dados e API, bem como o bloqueio de acesso direto aos bancos de dados através da internet. Cada um dos serviços armazenam logs, diminuindo a possibilidade de utilizadores repudiarem ações dentro do sistema. Através das estratégias de políticas de acesso foi amenizado o risco de divulgação de informações. Quando é necessário disponibilizar os dados para indivíduos não envolvidos no tratamento, como ocorre durante o uso do serviço de aprendizado de máquina, é feito a pseudonimização, para proteger a identidade e garantir a privacidade do pessoal médico e pacientes envolvidos nas sessões de tratamentos. A negação de serviço foi amenizada através da replicação de dados e serviços, para que haja serviços substitutos em caso de falha. A elevação de privilégios foi dificultada através do uso de conteinerização dos serviços utilizando a configuração rootless.

3.1.3 Pseudonimização

A pseudonimização é uma técnica que permite esconder a identidade de sujeitos a quaisquer serviços terceirizados através da atribuição de pseudo-identificadores, representado

pela Figura 3.2. Assim, os serviços terceirizados não são capazes de relacionar os pseudônimos com as identificações reais, e por sua vez não reconhecem o autor dos dados fornecidos [21].

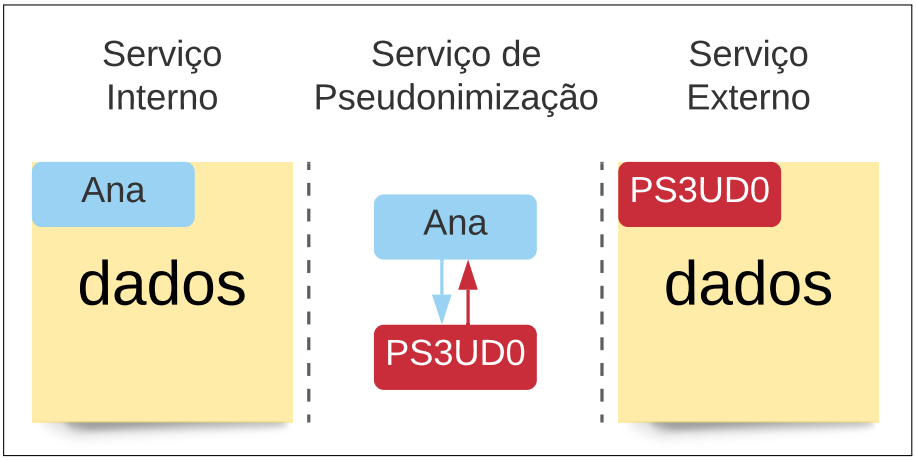

Figura 3.2: Documento pseudonimizado

O uso da pseudonimização é vantajoso em cenários onde um serviço externo ou empresa necessita de informações de pessoas reais dentro do sistema, mas não devem ter o conhecimento das pessoas ao qual os dados pertencem.

Um exemplo de uso da pseudonimização em sistema hospitalar é quando os dados de avaliações médicas feitas em pacientes necessitam ser enviadas a empresas terceirizadas, a fim de fornecer informações para análise. A empresa terceirizada poderá utilizar os dados para detectar novos padrões de doenças e anomalias, e para isso não é necessário ter o conhecimento do indivíduo real que foi avaliado. Então, antes de realizar o envio das avaliações, o hospital remove todas as informações que fazem a identificação do utilizador e substitui por um novo identificador (pseudônimo), que é apenas reconhecido pelo hospital.

A técnica ainda permite que um pseudônimo seja re-identificado pela empresa que possua acesso à tabela de pseudônimos quando necessário. Caso a empresa terceirizada tenha realizado uma descobertas que deve ser informada diretamente ao utilizador real, como a identificação de um problema ou doença, o utilizador ainda poderá ser identificado. A funcionalidade de re-identificação normalmente limita-se à empresa que fornece o pseudônimo, uma vez que os serviços externos não devem ser capazes de identificar os

utilizadores reais sem as devidas permissões.

Se um serviço externo necessita de reconhecer o utilizador real, inicialmente os serviços internos deverão ser acionados para solicitar a permissão. Ao receber o pedido, os serviços internos deverão entrar em um consenso junto ao utilizador real para definir se a verdadeira identidade poderá ser revelada ao serviço externo. A solicitação da identificação por parte de um serviço externo é representado na Figura 3.3

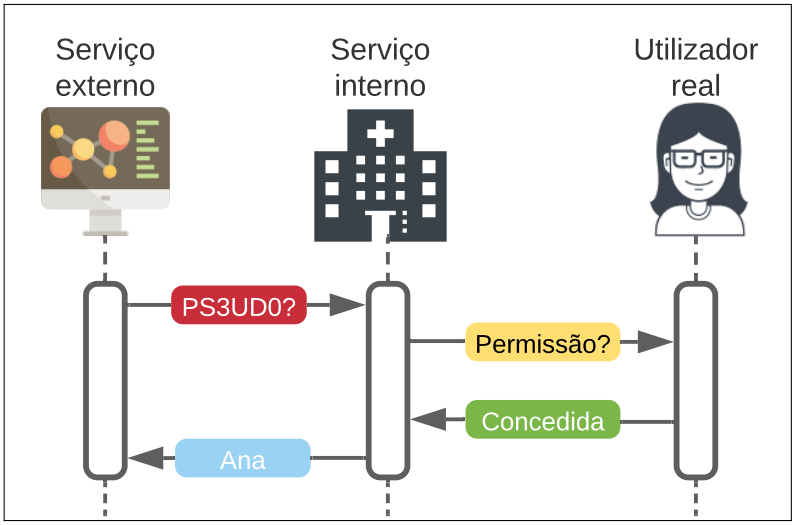

Figura 3.3: Solicitando identificação de um pseudônimo.

Para a implementação das estratégias de pseudonimização, pode se usar abordagens [21]:

- contador, onde cada novo registro é o valor do registro anterior, acrescido por 1. Simples de implementar, mas causa problemas em bancos complexos e escalabilidade;
- numero aleatório, onde cada pseudônimo é gerado randomicamente. Simples de implementar, mas causa problemas ao escalar, já que pode ocorrer colisões (gerar duas ou mais vezes o mesmo pseudônimo para utilizadores diferentes);
- Hash, onde um atributo é utilizado para gerar o pseudônimo através de algoritmos de hash. Fácil de implementar e livre de colisões, porém sofre ataques de força bruta e dicionários;

- Código de mensagem de autenticação, que funciona igual ao hash, porém faz uso de um segredo. Sem o conhecimento do segredo, não é possível mapear para o identificador original. É considerado uma técnica forte de pseudonimização;
- Criptografia assimétrica, que utiliza cifra de bloco e segredos para gerar e mapear os pseudônimos. Também é considerado uma técnica forte para pseudonimização.

Um sistema pode também fazer uso de mais de um pseudônimo por utilizador. As politicas que podem ser consideradas para a implementação são [21]:

- pseudonimização determinística, onde um utilizador sempre é referenciado através de um único pseudônimo;
- pseudonimização de documento randomizado, onde um utilizador possui uma quantidade fixa de pseudônimos, e faz o uso de cada de modo arbitrário;
- pseudonimização completamente randomizada, onde cada referência de um utilizador é feita com um pseudônimo diferente.

3.2 Proposta

A solução total é composta por quatro elementos principais. Os elementos referidos são: um dispositivo vestível, uma aplicação para dispositivos móveis, uma interface web e um ambiente implementado em máquinas virtuais do Instituto Politécnico de Bragança (IPB). Para simplificar a compreensão, o ambiente sugerido é fragmentado em serviços para autenticação single-sign on (SSO), manutenção dos recursos da unidade de saúde, e um serviço voltado à aprendizagem de máquina, além dos serviços de armazenamento. A comunicação entre os componentes é representado pela Figura 3.4. Cada um dos componentes são explicados posteriormente.

Para contemplar as necessidades de segurança, manutenção, monitoria e escalabilidade dos serviços contidos no ambiente do IPB, foi elaborado o diagrama da arquitetura, mostrado na Figura 3.5. A complexidade total da solução e quantidade de serviços é

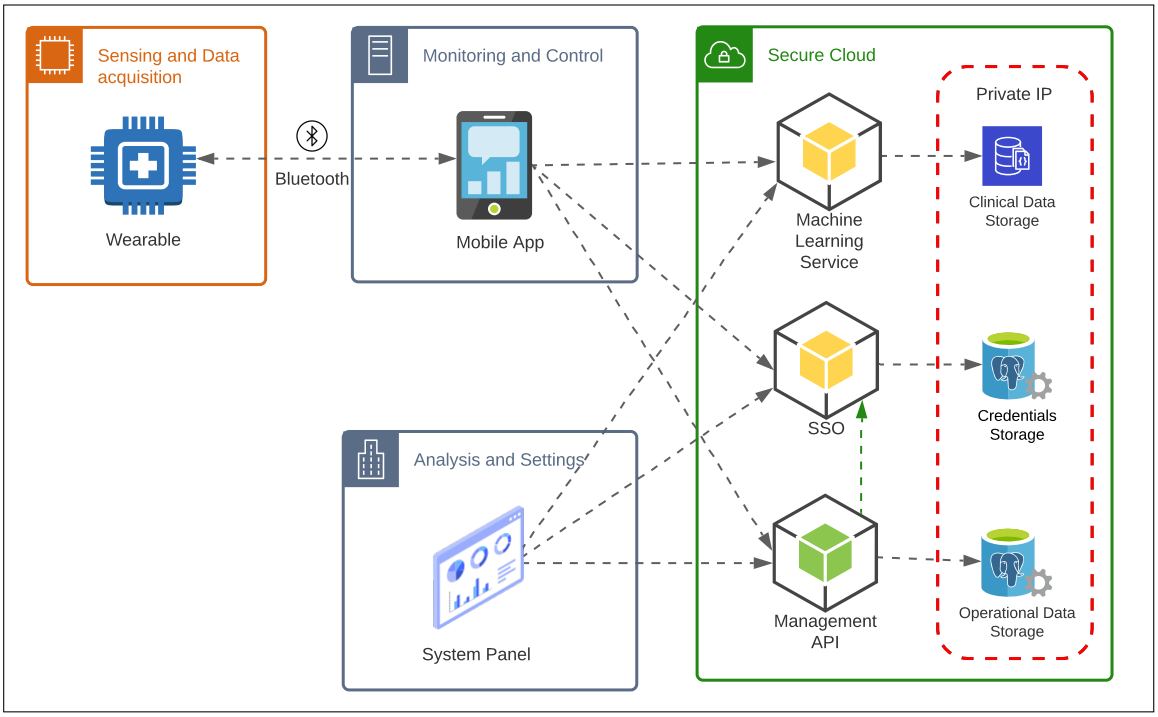

Figura 3.4: Comunicação entre os componentes.

uma consequência das arquiteturas com abordagem de microsserviços, porém provê a atomicidade de funcionalidades e visão pontual de cada componente do sistema.

3.2.1 Componentes

A solução é composta por componentes que funcionam cooperativamente. A seguir é apresentado os principais ativos do sistema, bem como suas características e objetivos.

Dispositivo Vestível

O dispositivo vestível é o componente utilizado dentro do sistema para a coleta de dados referentes à saúde muscular dos pacientes, além de ser responsável por aplicar o tratamento de estimulação dos músculos do utilizador.

Para efetuar a sua interação dentro do sistema, o dispositivo é capaz de se comunicar com aparelhos mobile através da tecnologia Bluetooth Low Energy (BLE). Todos os comandos são recebidos pela aplicação conectada, enquanto que algumas das requisições

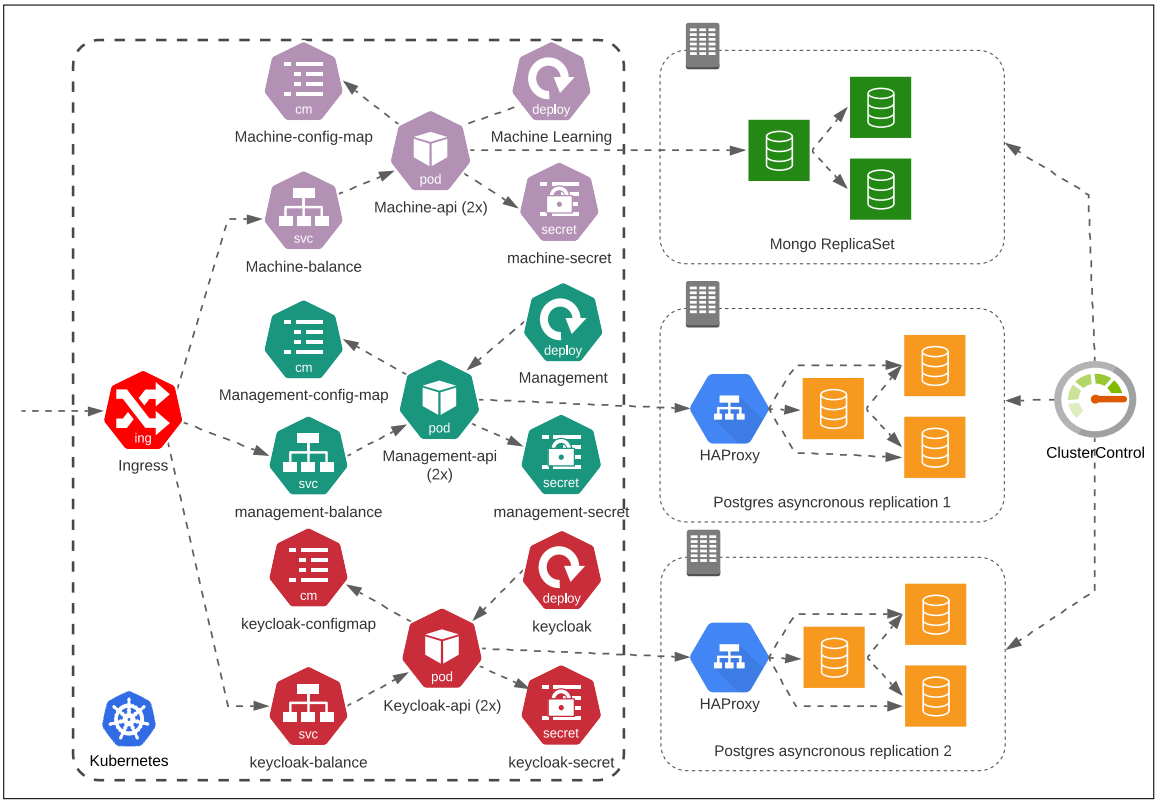

Figura 3.5: Arquitetura Proposta.

emitidas pelo dispositivo podem ser redirecionadas à nuvem, como mostrado na Figura 3.6.

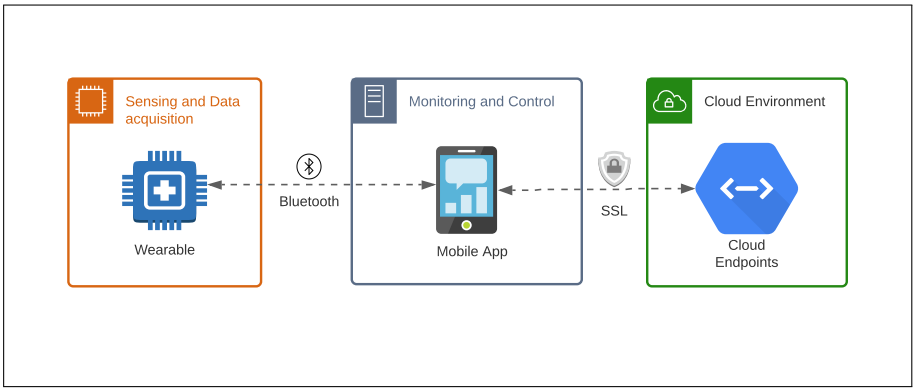

Figura 3.6: Ilustração simplificada do fluxo de informação.

O protocolo de comunicação BLE atualmente se mostra ineficiente contra espionagem e ataques como man-in-the-middle (MITM) [22]. Por utilizar o BLE para realizar interações, ocorre alguns desafios para garantir a segurança. É importante definir métodos para validar as informações recebidas pelo aplicativo, a fim de verificar se a origem das informações é confiável e detectar possíveis alterações durante a transmissão dos dados.

Mesmo com todas as precauções para garantir que os dados vieram de uma fonte confiável, não podemos detectar se a comunicação está sendo observada por um atacante. Há riscos de exposição ao enviar segredos entre o dispositivo e o aplicativo.

Em consequência da adoção das estratégias mencionadas acima, o consumo de energia do dispositivo aumenta, podendo ser um problema para tratamentos de duração mais longa em vestíveis alimentados por baterias, além do possível aumento no custo do desenvolvimento e manutenção.

O dispositivo vestível é o principal provedor de informações para o serviço de aprendizado de máquina, então seu funcionamento afeta os modelos e estudos de históricos do pacientes. É importante realizar vistorias periódicas dos dispositivos, analisando se os sensores estão a funcionar corretamente.

Aplicação Mobile

O aplicativo mobile é utilizado para realizar o monitoramento e controle do dispositivo vestível, além de permitir a comunicação com o sistema hospedado na nuvem, como representado anteriormente pela Figura 3.6.

A aplicação deve ser capaz de ler os dados enviados pelo dispositivo vestível, mas não pode ser capaz de alterá-los. Dessa forma, é possível dar um feedback rápido ao paciente através de dashboards de monitoramento, realizando os processamentos em tempo real para tomada de decisões rápidas e automáticas.

O mobile também precisa ser capaz de enviar comandos ao vestível de modo que a autoria do emissor seja reconhecida através de uma assinatura. Assim, o vestível pode identificar se a origem é confiável para performar ações ou configurações.

Em casos onde o aplicativo é utilizado apenas como um meio para conectar o sistema hospedado em nuvem com dispositivo vestível, a aplicação não pode ser capaz de alterar as informações. Isso pode ocorrer, por exemplo, quando o modelo de um tratamento é emitido por um serviço na nuvem, e para chegar ao vestível, será necessário passar pelo aplicativo. Por acidente ou propositalmente, o utilizador do mobile poderia fornecer um tratamento indevido a um paciente caso a mudanças das informações seja permitidas.

Interface Web

A interface de usuário desenvolvida na web é o componente que contém as funcionalidades que permitem os à equipa médica a interagir com o sistema. O componente deve comunicar-se com o backend hospedado na nuvem através do uso do protocolo Hypertext Transfer Protocol Secure (HTTPS), dificultando a espionagem e MITM.

A autenticação e autorização de um médico ocorre através do serviço de SSO, que retorna os tokens JWT com prazos de validade.

No caso da interface é importante se proteger dos ataques mais comúns e principalmente de ataques que fazem o uso de *client-side scripting*. Os ataques *client-side scripting* permitem a um atacante executar códigos através do browser do utilizador, sendo efetivo

para forjar requisições e roubos de informações contidas no navegador.

Single-Sign On

O serviço de SSO cria um ponto único de autenticação dentro do sistema. Dessa forma, cada um dos micro-serviços não necessitam de conter as regras de autenticação.

Os problemas relacionados ao serviço está em criar um ponto único de falha no sistema. Caso o serviço de SSO fique indisponível, os utilizadores não serão capazes de realizar autenticação.

Para aumentar a disponibilidade do serviço de SSO, pode-se criar um cluster, assim há mais serviços de SSO exatamente iguais, prontos para dividir a carga das requisições e substituir a instância indisponível.

Dentro da plataforma foi utilizado o *Keycloak* para essa função, por ser uma solução robusta que permite configurações avançadas e tecnologias atuais.

API de Gerenciamento

A API de gerenciamento contém a regra de negócio da unidade de saúde. Dentro do serviço é definido os agendamentos, históricos de consultas, entre outros. Também contém as políticas de acesso a algumas informações, para garantir que apenas utilizadores com características específicas possam obter determinadas informações.

Um exemplo de política definida é que apenas o médico responsável por aplicar um tratamento e o paciente tratado podem ter acesso ao histórico dos tratamentos e feedbacks. Essa política entra em conflito quando incluímos o uso do serviço de aprendizado.

A equipa responsável pelo desenvolvimento, acompanhamento e manutenção do serviço de aprendizagem de máquina necessitam do acesso aos dados para desenvolverem seus estudos. Para satisfazer esse critério, a API de gerenciamento utiliza a técnica de pseudonimização para partilhar as informações.

A pseudonimização permite que uma entidade terceirizada tenha acesso aos dados necessários, sem possuir o conhecimento do dono dos dados. Dessa forma, o serviço de

aprendizado de máquina pode utilizar o histórico de tratamentos de um paciente, mas não é capaz de identificar qual paciente é o dono do histórico.

A abordagem utilizada para realizar a pseudonimização é determinística. Nessa abordagem um utilizador é sempre representado pelo mesmo pseudônimo [21]. Abordagens com múltiplos pseudônimos impossibilitaria o serviço de aprendizado de máquina a estudar todo o histórico de um paciente, uma vez que o serviço não seria capaz de identificar os diferentes pseudônimos que pertencem ao mesmo paciente.

Aprendizado de máquina

O serviço de aprendizado de máquina é o responsável por analisar os registros de cada um dos tratamentos, de modo descobrir padrões e sugerir planos de tratamentos ideais para cada paciente. O aprendizado é influenciado pela qualidade dos dados gerados, então é essencial garantir que os dados são confiáveis, validos e com o mínimo de ruídos possível.

Os ruídos podem ocorrer de modo intencional ou não intencional. O modo não intencional é quando o dispositivo vestível não está bem posicionado, por exemplo. Assim as medições feitas pelos sensores serão imprecisas, consequentemente afetando o serviço de aprendizagem.

Os ruídos também podem ser intencionalmente causados quando os sensores do dispositivo vestível estão danificados, ou ainda quando as informações são modificadas durante a transmissão. Um atacante pode danificar os sensores, ou manipular os meios de transmissão de informações, afim de causar imprecisão das informações.

Após a sugestão de um plano de tratamento, este deve ser revisado por um médico antes de ser aplicado em um paciente. Com isso, o risco de um paciente ser submetido a um tratamento prejudicial é amenizada. Mesmo assim, é importante garantir que não há como atacantes modificarem o plano de tratamento gerado.

Capítulo 4

Implementação

Nesse Capítulo é apresentado as tecnologias utilizadas para o desenvolvimento da arquitetura e de microsserviços, contendo também os detalhes da arquitetura e dos processos relacionados ao desenvolvimento da solução.

4.1 Tecnologias de desenvolvimento

Nessa seção é abordado brevemente as tecnologias utilizadas para realizar a implementação do projeto. É abordado o framework *Flask* utilizado para o desenvolvimento de APIs, o protocolo JWT para transporte de informação em meios não confiáveis, o *Docker* para conteinerização de serviços, os bancos de dados *MongoDB* e *Postgres*, o serviço *Keycloak* como solução para autenticação em microsserviços, o *ClusterControl* para monitoramento de clusters de dados, o *HAProxy* para roteamento de requisições e os recursos do *kubernetes*.

4.1.1 Flask

Flask é um micro framework da linguagem python para auxiliar no desenvolvimento de Web Server Gateway Interface (WSGI). O Flask é intitulado como micro framework por conter um núcleo simples e extensível, fornecendo maior flexibilidade ao desenvolvedor

[23].

O framework não fornece uma camada de abstração dos dados automatizada, deixando-os a critério do próprio desenvolvedor [23]. Além disso, também fica sob a responsabilidade do desenvolvedor a inclusão de validação dos dados recebidos e os devidos processamentos. Dessa forma, utilizar o Flask em sua forma pura pode tornar o desenvolvimento de aplicativos mais complexos.

Atualmente, há diversas extensões que podem ser utilizadas em conjunto com o framework para facilitar a integração de serviços. Assim torna-se melhor a experiência de implementação e manutenção de sistemas desenvolvidos com o Flask.

Como alguns exemplos de extensões disponíveis para o Flask, está disponível o *Flask-SQLAlchemy* para integração de bancos de dados SQL, *flask-pymongo* para integração com banco de dados mongoDB, Flask-Login para gerenciamento de utilizadores logados em arquiteturas statefull, Flask-Mail para envio de emails através do protocolo de transferência de correio simples (Simple Mail Transfer Protocol, SMTP) e Flask-JWT para utilizar JWT.

A praticidade de implementar APIs utilizando Flask, somado com a quantidade de bibliotecas voltadas à aprendizagem de máquina na linguagem python, tornam-se um grande atrativo para o desenvolvimento de aplicações de inteligências artificiais que utilizam streams de informações como base para o estudo, como em implementações de sistemas de análises de tempo real.

4.1.2 Norma JSON Web Token

JSON Web Token JWT é uma norma aberta (RFC 7519) para realizar transmissão de informações de modo seguro, compacto e independente utilizando o formato JavaScript Object Notation (JSON) [24]. Por ser compacto, é possível realizar a transmissão dos tokens JWT através de URL, parâmetros do método POST ou dentro do cabeçalho de HTTP [24].

Os tokens enviados contém as informações necessárias para ser utilizado dentro de um

sistema. Isso os torna independentes, já que não há necessidade de realizar consultas em bancos de dados com grande frequência [24].

O JWT é utilizado em casos onde é necessário autorização ou troca de informação segura [24]. Os tokens são confiáveis apenas por ser assinados digitalmente. As assinaturas permitem detectar caso haja mudança no conteúdo durante sua transmissão [25].

O JWT é composto por três partes, sendo o cabeçalho, os dados e a assinatura. O cabeçalho é responsável por informar qual o algoritmo de criptografia utilizado para realizar a assinatura, e qual o tipo de token. Os dados contém as informações a serem enviadas e informações do próprio JWT, como tempo de validade, emissor e receptor. A assinatura é gerada através da utilização do cabeçalho e dos dados. Assim, caso a assinatura não seja compatível com o processamento das informações, é detectado a falha na integridade dos dados [24].

Os tokens são apenas assinados, e não criptografados, então qualquer sistema que os recebe é capaz de ler as informações contidas. Para garantir maior proteção, é importante tomar medidas como o uso de HTTPS durante a transmissão, para dificultar que as informações sejam vistas por pessoas não autorizadas [25].

Os JWTs podem ser assinados através dos algoritmos Hash-Based Message Authentication Code (HMAC), Rivest–Shamir–Adleman (RSA) e Elliptic Curve Digital Signature Algorithm (ECDSA) [25]. A assinatura só pode ser feita pelo serviço que possui acesso ao segredo. Dessa forma, apenas o serviço é capaz de gerar tokens válidos. Por isso, os tokens podem ser utilizados em autenticações stateless, por SSO, sem a necessidade de armazenar as sessões no backend.

Ao optar pelo uso do algoritmo HMAC em sistemas distribuídos, todos os componentes do sistema que realizam a validação dos tokens também necessitará de conhecer o segredo utilizado. Dessa forma damos ao atacante maiores possibilidades de obter o segredo utilizado para a geração e validação dos tokens de todo o sistema. Em casos de micro serviços, por sua natureza distribuída, optar pelo uso de criptografia assimétrica com chaves diferentes para cada serviço dará menores poderes para o atacante.

As estratégias de autenticação baseadas em tokens JWT utilizam três tipos principais

de token:

- ID token, contendo as informações do utilizador e alguns dados relacionados ao serviço provedor de identidade;
- Access Tokens, que dá acesso por tempo determinado a recursos do sistema;
- Refresh Tokens, que permitem a geração de novos tokens de acesso sem a necessidade de fornecer novamente as credenciais de acesso.

A proteção dos tokens é primordial para a segurança. Caso um token de acesso seja capturado, o atacante poderá realizar interações com os servidores em nome do utilizador durante o tempo que o token permanecer válido. Ainda existem formas de ataques aos sistemas devido ao mau uso de tokens JWT. Técnicas como man-in-the-cloud (MITC) já foram identificadas em softwares de grande porte, que se tornaram possível através do roubo do token de refresh [26]. Outros ataques fazem uso da modificação do algoritmo de criptografia utilizado para validar os tokens.

4.1.3 Docker

Docker é uma plataforma aberta para a execução, envio e desenvolvimento de aplicativos. O uso da plataforma acelera a entrega de software, pois realiza a separação entre os aplicativos e a infraestrutura [27].

Ao integrar o Docker a uma aplicação é criado um ambiente isolado, chamado de contêiner. Dentro de um contêiner deve conter apenas os arquivos necessários para o funcionamento da aplicação. Assim, para realizar o teste, execução ou até mesmo o desenvolvimento, basta apenas executar o contêiner, sem a necessidade de instalar as dependências no host. Além disso, a plataforma também permite a execução de múltiplos contêineres [27].

Os contêineres possuem opções de configurações para maximizar o funcionamento do sistema. Como exemplo, o Docker permite configurar um contêiner para reinicializar todas as vezes que ocorrer a finalização do aplicativo. No caso citado, é afetado principalmente

a disponibilidade do software, já que a instância estaria indisponível por pouco tempo, mas automaticamente seria reinicializada.

O Docker também fornece configurações de redes virtuais, permitindo que os múltiplos contêineres se comuniquem entre si. Ao utilizar características elásticas, onde a quantidade de recursos podem crescer ou diminuir de acordo com a necessidade, é vantajoso utilizar estratégias para orquestrar de contêineres.

Para estratégias onde exige gerenciamento de clusters criados no docker, é possível fazer uso do modo Swarm. Assim, o escalonamento de serviços é realizado pela declaração da quantidade de serviços que se deseja obter[28].

Ao utilizar o modo swarm, um dos nós ficará responsável por monitorar os serviços que foram criados, de modo a garantir que as configurações fornecidas sejam obedecidas. Em outras palavras, caso o conjunto de processo do Docker não obedeça as configurações desejadas, o gerenciador tratará automaticamente a falha [28].

O gerenciador do swarm pode agir, por exemplo, quando um dos processos do Docker finaliza inesperadamente, tornando a quantidade de seviços ativos inferior à quantidade especificada nas configurações. Para reparar, o gerenciador pode criar outro processo para substituir o anterior, garantindo que a quantidade de serviços declarados seja obedecida.

Com o modo swarm ativo, é possível trabalhar com redes para diversos hosts, além de prover balanceamento de carga para serviços replicados. O swarm é seguro por padrão, pois faz uso de autenticação e criptografia mútua TLS [28].

O Docker também possui estratégias rootless. Dessa forma, o contêiner não é executado com privilégios de super usuário, garantindo maior segurança. Porém, o uso do rootless reduz diversas funcionalidades, como a integração com o swarm, uma vez que o tipo de conexão usada pelo swarm deixa de ser suportada, algumas portas não podem ser expostas e configurações de rede podem não ser permitidas.

4.1.4 MongoDB

O MongoDB é um banco de dados para aplicativos modernos, que necessitem das características oferecidas por sistemas not-only SQL (noSQL). É um banco de propósito geral, que pode ser distribuído entre múltiplos hosts [29].

A sua arquitetura provê uma boa adaptação para sistemas na e maior facilidade durante o período de desenvolvimento. Como sintaxe, os dados são organizados em documentos utilizando JSON, o que permite que cada documento possa conter documentos e arrays aninhados [29].

Para utilizar o MongoDB, é possível optar pela versão community, que não contém custos e apresenta algumas imitações, ou optar pela versão enterprise, que fornece recursos adicionais para empresas. Serviços como criptografia em repouso possuem apenas suporte na versão enterprise [30].

O banco de dados, apresenta algumas vantagens por sua arquitetura, como alta performance, alta disponibilidade e escalabilidade horizontal. A alta performance ocorre por dar suporte à modelos de dados aninhados, auxiliando na redução de operações de entradas e saídas. A alta disponibilidade pode ser obtida através da ativação de *replicasets*, que geram redundância dos dados e *faillover* automático. A escalabilidade horizontal pode ser alcançada pela criação de *shards*, que distribuem os dados para diferentes hosts, aumentando a capacidade de armazenamento e processamento [30].

4.1.5 Postgres

O Postegres é um sistema de banco de dados relacional, criado com o objetivo de estender a segurança e a escalabilidade do SQL. Devido ao seu uso, o Postgres já possui uma forte reputação quando se trata de extensibilidade, recursos robustos, confiabilidade e integração de dados [31].

O sistema é gratuito e de código aberto, permitindo que comunidade de desenvolvedores sejam capaz de realizar melhorias e submetê-las ao código original [31] .

Para garantir a segurança, o postgres oferece configurações para recuperação em caso

de desastre, replicação de servidores nos modo assíncrono, síncrono e lógico, algoritmos para autenticação, sistema robusto para controle de acesso, segurança a nível de linha e coluna, e também dá suporte para autenticação multi fator multi-factor authentication (MFA) [31].

4.1.6 Keycloak

O Keycloak é uma solução da Red Hat para o gerenciamento de identidades e acessos de utilizadores e serviços. A solução é focada em sistemas modernos, e possui seu código aberto, permitindo que cada utilizador seja capaz de melhorar o produto. [32].

O uso do Keycloak permite adicionar estratégias seguras de autenticação de modo simples e prático com serviços internos ou serviços de terceiros, como redes sociais [33].

Para realizar a autenticação utilizando serviços externos como provedores de identidade, é possível optar por padrões como openID Connect e SAML 2.0 [32].

O serviço também pode ser utilizado para autenticação centralizada em sistemas complexos e distribuídos [32]. Desse modo o Keycloak realiza o login dos utilizadores, e emite tokens que poderão ser utilizados em diversos serviços.

O Keycloak não necessita de linhas de código para gerenciar recursos ou ser configurado, pois fornece uma interface para os utilizadores. A interface de console admin traz as principais funcionalidades do sistema, como ativar e desativar recursos, definir políticas, gerenciar utilizadores e gerenciar permissões[32].

Utilizar um único serviço para autenticação centralizada apresenta um risco ao sistema. Caso a única instância do serviço pare de funcionar, os utilizadores ficam impossibilitados de realizar cadastro, login e logout, enquanto que os serviços podem não ser capazes de validar os utilizadores já autenticados. Idealmente, é importante configurar o Keycloak em um cluster para reduzir o problema.

4.1.7 ClusterControl

ClusterControl é um serviço criado pela empresa Severalnines, capaz de auxiliar durante o processo de deploy e configuração de clusters de bancos de dados gratuitos, além de prover o monitoramento para cada cluster criado. Atualmente, o serviço é compatível com MySql, MariaDB, Percona, MongoDB, PostgresSQL, Galera Cluster e outros [34].

O serviço possui duas versões, sendo uma empresarial, e outra para a comunidade. É um serviço gratuito para o uso, porém, sua versão empresarial fornece funcionalidades adicionais para gerenciar backups, reparar e realizar a recuperação de clusters de modo automático, gerenciar a performance de cada cluster, além de configurações para aprimorar a segurança dos clusters [34].

O ClusterControl centraliza o gerenciamento de clusters de bancos de dados, e automatiza processos para garantir um bom funcionamento do sistema. Todo o controle pode ser feito através de uma interface gráfica, ou ainda através de comandos [34].

O serviço permite a configuração de avisos e alertas [34]. Dessa forma, é possível notificar os mantenedores caso tenha ocorrido algum problema com um cluster, como alta demanda de requisições, falhas e desligamento do cluster.

Por ser um serviço que centraliza o gerenciamento, é importante garantir que apenas as pessoas autorizadas tenham acesso. Para isso, o serviço também conta com mecanismos de autenticação e autorização.

4.1.8 HAProxy

O HAProxy é uma solução para garantir alta disponibilidade [35], aumentando a performance dos trabalhos através da distribuição de requisições entre múltiplos servidores [36]. O HAProxy permite conexão através de protocolos HTTP e por Transmission Control Protocol (TCP) [35], dando suporte a certificados de Secure Shell (SSH) [36].

A distribuição de requisições para múltiplos servidores é feita através do balanceamento de carga. O serviço realiza o balanceamento de carga [35], permitindo diferentes estratégias de escalonamento de requisições. As estratégias são [36]:

- Round Robin, onde o escalonador utiliza uma sequência específica de servidores para enviar a requisição;
- Última conexão, onde o escalonador envia a primeira requisição de um utilizador para o servidor escolhido, e a seguir todas as novas solicitações são redirecionadas ao mesmo servidor;
- Recurso, onde as requisições são enviadas ao servidor que realizou a última resposta ao cliente.

O HAProxy suporta a configuração de múltiplos backends. O balanceamento de carga é feito com base no endereço de URL/URI [36], e assim é possível definir diferentes endereços para cada backend.

4.1.9 Kubernetes

O kubernetes é uma plataforma capaz de orquestrar serviços conteinerizados [37]. A plataforma recebe configurações declarativas no formato JSON e Yet Another Markup Language (YAML), além de possuir fácil portabilidade e grande capacidade de extensão, com diversas estratégias de automação já implementadas [37].

A plataforma colabora com a redução de tempo de queda do sistema [37], pois permite gerenciamento automático de contêineres em casos de falha. Também providencia padrões para o desenvolvimento de serviços.

Para tornar simples o uso do kubernetes em servidores locais, foi desenvolvido o minikube [38]. O minikube é uma implementação com maior leveza do kubernetes, fazendo uso de uma máquina virtual para criar um cluster simples e com um nó apenas [39]. O minikube também possui uma interface de linha de comando para realizar as operações [39].

Mesmo utilizando um único nó, o kubernetes permite a criação de diversos serviços conteinerizados para auxiliar durante a construção da arquitetura. Por padrão, o kubernetes já fornece alguns serviços para a orquestração e automação de contêineres. A seguir

é brevemente apresentado os principais serviços oferecidos.

Ingress

O serviço Ingress é oferecido pelo kubernetes com o objetivo de gerenciar todos os acessos externos [40]. Através do serviço, é possível redirecionar requisições para as outras aplicações disponíveis dentro do kubernetes.

Para utilizar o Ingress, é necessário optar por um dos controladores do serviço [40]. Há diversas opções que podem ser escolhidas de acordo com as necessidades da solução em desenvolvimento. O Minikube possui integração simplificada com o NGINX Ingress Controller.

Transpassar serviços já existentes para a arquitetura do kubernetes traz desafios e complexidades adicionais de em relação ao tráfego de informação [41]. Os controladores funcionam como abstração para suprir o dinamismo presente no Kubernetes, simplificando as configurações necessárias para realizar o roteamento.

O Ingress realiza o balanceamento de carga, permite conexão através de Secure Sockets Layer (SSL) e permite a configuração de hospedagem virtual [40]. A hospedagem virtual é feita ao definir os URLs validas para cada serviço. Como permite o uso de SSL, o Ingress expõe portas HTTP e HTTPS para habilidar a recepção de requisições.

Pods

Os pods oferecidos pelo kubernetes são as menores unidades de componentes que poderão ser utilizados para compor a solução [42]. São componentes que podem ser destruídos e recriados a qualquer momento, com IPs únicos e dinâmicos [42], [43].

Geralmente o uso de pods não é feito diretamente, mas ocorre através de outros serviços, como o Deployment [42]. Isso ocorre devido à característica dinâmica dos pods, que a qualquer momento podem simplesmente serem recriados com endereços diferentes, por exemplo. O uso de outros serviços permite automatizar o gerenciamento dos pods do sistema [42].

Deployment

O Deployment fornecido pelo kubernetes é a descrição de estados desejáveis de pods ou replicasets [44]. Ao definir um conjunto de características desejáveis, o serviço de Deployment realizará os ajustes para que a configuração seja mantida. Como exemplo, ao definir a quantidade de pods que deverão estar em funcionamento, o serviço de Deployment poderá criar ou remover pods caso não estejam em conformidade com a quantidade declarada.

Quando o servidor permite auto escalabilidade dos serviços, o Deployment poderá ser utilizado para definir o número mínimo e máximo de serviços disponíveis [44]. Também é utilizado para gerenciar as versões de atualização dos pods em escala horizontal [44].

Service

O Service oferecido pelo kubernetes é capaz de expor as aplicações internas como serviços acessíveis pela internet [43]. Os pods do kubernetes são efêmeros e com endereços de IP dinâmicos, tornando a tarefa mais complexa para ser implementada de modo manual.

O Service resolve os problemas relacionados à identificação dos diversos endereços de IP dinâmicos, realizando também a inclusão de balanceamento de carga entre os múltiplos pods mapeados [43].

Ao utilizar Services, cria-se um conjunto lógico de pods que permite a adição de politicas de acesso [43]. Esse componente não costuma ser destruído com tanta frequência que os pods.

ConfigMap

O serviço de ConfigMap fornecido pelo kubernetes tem seu foco principal no armazenamento de dados não confidenciais [45]. Seu uso não foi projetado para o armazenamento de grandes quantidades de dados, já que seu objetivo é prover dados de configuração para outros componentes [45].

Os pods são capazes de consumir os dados informados no ConfigMap como variáveis

ambiente [45]. Também fornece uma API que permite a modificação dos valores contidos [45]. Caso as alterações dos valores ofereça riscos intencionais ou não intencionais, é possível alterar as configurações de imutabilidade das informações [45]. A configuração também aumenta a performance por parar serviços desnecessários [45].

Secret

O Secret é oferecido pelo kubernetes como uma solução para o armazenamento de dados sensíveis [46]. A configuração dos Secrets que são feitos à parte, permitindo que não haja informações sensíveis nas configurações dos outros serviços, como Deployments [45].

O secrets é consumido por pods [46]. O uso dos segredos é feito através de variáveis de ambiente, como ocorre com o ConfigMap. A aplicação tem o dever de não expor os valores, uma vez que o segredo estará em variáveis de ambiente no formato de texto simples dentro da aplicação.

Por padrão, a funcionalidade realiza o armazenamento dos dados sem aplicar criptografia e permite que qualquer serviço com acesso à API faça requisições e alterações dos dados [46]. Cabe ao desenvolvedor realizar a configuração necessária para atingir maiores níveis de segurança.

O uso do serviço promove menor risco de exposição dos dados sensíveis durantes os processos de criação, visualização e edição dos pods [46], atividades naturais de DevOps. Também pode ocorrer exposição quando o utilizador possui acesso para ler qualquer Secret em um determinado namespace do kubernetes [46].

Os dados do Secret são armazenados em memória volátil [46], dificultando sua obtenção por roubo de hardware. Os segredos escritos no arquivo YAML para configur o Secret são codificados em base64 [46]. Assim, a obtenção do arquivo de configuração por um utilizador não autorizado irá comprometer os segredos armazenados.

Para obter melhores proteções, o Secrets oferece configurações para regras RBAC e ativação de criptografia em repouso [46].

4.2 Implementação da arquitetura

Para a criação da arquitetura, foram utilizadas máquinas virtuais disponibilizadas pelo IPB a fim de hospedar os serviços propostos. A solução desenvolvida utilizou dez máquinas virtuais, organizadas em três clusters e uma máquina para recepção, representado na Figura 4.1.

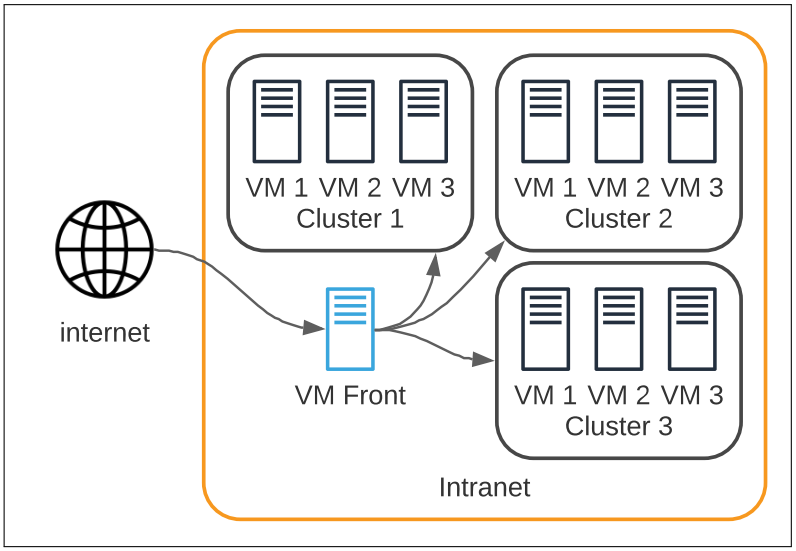

Figura 4.1: Organização das máquinas virtuais.

Apenas uma máquina virtual possui um endereço de IP alcançável pela internet. As demais máquinas virtuais são apenas acessáveis pela rede local, evitando que os serviços hospedados sejam explorados ou atacados diretamente. Todas as requisições devem passar pela máquina virtual de recepção antes de ser repassada.

Cada um dos clusters criados contém instâncias de bancos de dados. Assim, caso uma das máquinas virtuais apresente falhas, outras máquinas do mesmo cluster podem garantir o funcionamento do banco de dados. Dentro de um cluster, todos os dados do banco são replicados.

Os clusters são destinados a aplicações específicas. Há três serviços que fornecem as funcionalidades principais dentro da arquitetura, sendo eles a API de gerenciamento dos recursos da unidade de saúde, a API responsável pela aprendizagem de máquina e o serviço de autenticação. Dessa forma, cada cluster contém as instâncias e réplicas do

banco de dados para suprir as necessidades de um serviço. Dentro dos clusters formados pelas máquinas virtuais também encontra-se os serviços de monitoramento e serviços de proxy reverse.

Dentro do servidor de recepção, encontra-se os serviços integrados ao Kubernetes. Todas as requisições recebidas são redirecionadas aos serviços gerenciados pelo Kubernetes. Os serviços utilizam os dados contidos nos clusters, criando assim a separação entre os hosts de armazenamento de dados e o host de execução das APIs.

4.3 Deploy da solução

Nessa seção são explicadas as etapas utilizadas para realizar o deploy da solução proposta, representada pela Figura 3.5. Durante a seção, é abordado o processo de configuração de réplicas assíncronas do Postgres e sua integração com o HAProxy. É também explicado o processo de configuração utilizado para as réplicas do MongoDB.

Para monitorar os clusters de dados, é apresentado o processo de configuração do serviço ClusterControl. Para a configuração ser realizada, também é necessário utilizar conexão SSH.

Esta seção também apresenta as estratégias adotadas para a integração dos serviços de APIs conteinerizados com o Kubernetes, explicando o uso do ConfigMap, Secrets, Deployment e Service dentro da solução proposta.

4.3.1 PostgresSQL com replicação assíncrona

O Postgres permite realizar o desenvolvimento de um conjunto de servidores replicados, conhecido como replicação Postgres. Para acelerar as operações de escrita, o Postgres define as operações como assíncrono por padrão. Para a implementação, foi criado dois clusters Postgres, cada um contendo três instâncias conteinerizadas com docker rootless. É representado na Figura 4.2 a estrutura de cada cluster Postgres.

Para cada uma das instâncias, foram expostas três portas, contendo o serviço SSH para integração do ClusterControl, uma porta para realizar a recepção das requisições

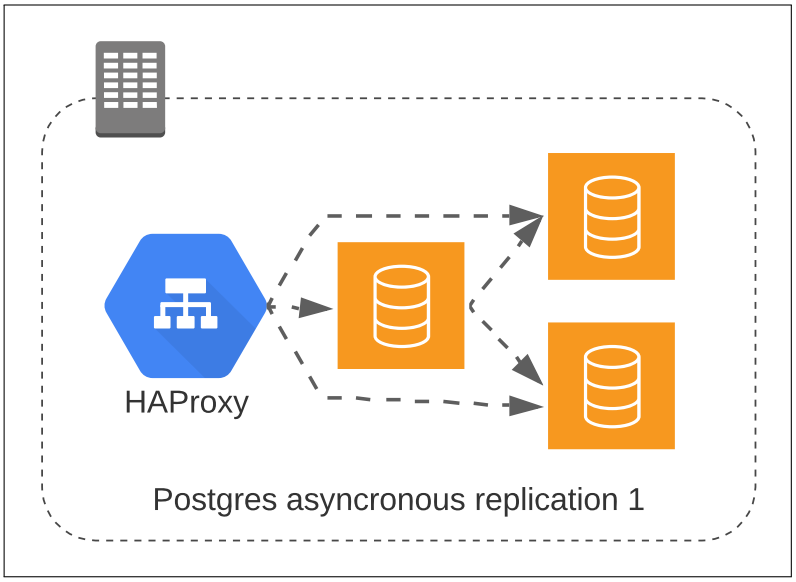

Figura 4.2: Estrutura do Cluster Postgres.

de operações no banco de dados, e uma porta para realizar a checagem do servidor. A estratégia comumente utilizada consiste em definir nó sendo o master, capaz de realizar leituras e escritas, enquanto que os outros nós são definidos para leitura apenas. Há outras abordagens permitidas dentro do Postgres, focadas em múltiplas gravações, por exemplo.

Para realizar a implementação do cluster, primeiramente foi necessário realizar as configurações da instância master. Os comandos utilizados é documentado no Anexo A.2. A configuração seguiu as etapas:

- Criação do contêiner rootless contendo a imagem do Postgres e expondo três portas, uma vez que não será possível expor novas portas sem recriar e reconfigurar os contêineres;

- Configuração do nó, informando as definições do cluster como a quantidade máxima de nós permitidos e quantidade de anotações de registro;

- Configuração do acesso, indicando através do arquivo `pg_hba.conf` qual utilizador estará permitido para acessar o banco como réplica, e qual endereço de IP o utilizador deve conter;

- Criação dos utilizadores, para habilitar o acesso dos nós de cada réplica, além da criação dos utilizadores para que outros serviços utilizem o banco;
- Criação dos slots, utilizado pelo Postgres como replicação lógica, assim é possível mapear cada nó de réplica a um dos slots declarados;
- instalação do extended internet service deamond (xinetd) no contêiner, para integração com o HAProxy.

A criação dos nós de réplica do cluster ocorrem de modo diferente. Após criar a instância master, é necessário realizar um backup do banco de dados, atribuindo a um determinado slot e utilizador. Durante o processo, o Postgres cria arquivos onde é especificada a conexão com a instância master, além dos dados de backup. Os arquivos obtidos na operação devem substituir o diretório de dados das réplicas, e dessa forma o Postgres detecta as réplicas criadas. O procedimento é realizado para cada slot, de modo que cada réplica possua um slot próprio.

A conexão entre as réplicas e o nó master podem não estar corretas devido ao uso de contêineres. Ao gerar as informações para conexão, o nó master reconhecerá o IP e porta interna do contêiner, que pode não ser alcançável por outros contêineres em diferentes hosts. Para resolver o problema, foi necessário editar as configurações da conexão.

Uma vez que todas as requisições de escrita devem chegar apenas à instância master, é necessário realizar a descoberta de qual das instâncias satisfará esse critério. Diferentemente do MongoDB, o Postgres não fornece automaticamente o redirecionamento para o instância master. Para realizar o roteamento da solicitação, foi utilizado o HAProxy em conjunto com o xinetd.

HAProxy

O HAProxy provê balanceamento de carga e realiza o roteamento de requisições. Seu uso em conjunto com o postgres permite redirecionar as solicitações para qualquer um dos nós contidos no cluster, dividindo assim a carga entre os nós. Caso uma das instâncias

apresente mal funcionamento, o HAProxy redireciona a requisição para os servidores em funcionamento.

Durante uma solicitação de leitura, não há dificuldades extras em redirecionar para qualquer instância do Postgres, já que todas contém os mesmos dados e permissão de leitura. Porém, o cenário é diferente quando se trata de escritas, pois apenas o nó master será capaz de realizá-las.

Implementar formas para detecção do tipo de operação (leitura ou escrita no banco de dados), e estratégias para descobrir qual instância poderá escrever são os desafios adicionais que surgem ao optar pelo uso do Postgres. Geralmente, a diferenciação das operações é feita através da definição de portas ou URLs diferentes para cada objetivo.

Para a diferenciação das operações dentro da arquitetura proposta, foram expostas duas portas, sendo a primeira responsável por enviar a solicitação para qualquer instância capaz de realizar leitura, enquanto que a segunda porta exposta é capaz de redirecionar as solicitações apenas para a instância master, capaz de realizar leitura e escrita.

Para que a instância seja identificada pelo HAProxy, foi necessário criar um procedimento capaz de emitir diferentes respostas de acordo com o resultado obtido por uma consulta local no banco Postgres. O processo é a criação da checagem de saúde do servidor e simultaneamente define o tipo do servidor. Através do serviço xinetd contido em cada contêiner do cluster Postgres, o procedimento consulta o tipo de instância configurada e retorna ao HAProxy a informação de que o banco está a funcionar como réplica, está a funcionar como master, ou não está a funcionar. Como se trata de uma consulta local, foi criado um utilizador com baixas permissões de acesso. Os comandos referentes a implementação estão disponíveis no no Anexo A.2.

Ao receber a resposta, o HAProxy consegue identificar se a instância é a desejada, ou se ainda deverá buscar por outra. Ao finalizar, os serviços podem se conectar as instâncias do Postgres por intermédio do HAProxy.

4.3.2 Mongodb Replicaset

O banco de dados mongoDB permite realizar o deploy de um conjunto de servidores replicados, conhecido como MongoDB replicaSet. As principais vantagem da abordagem são promover uma instância capaz de substituir a principal em caso de falha, e ainda permitir aos utilizadores acessarem a diferentes instâncias como se fossem o mesmo banco, dessa forma a quantidade de requisições e processamentos pode ser dividida entre os hosts.

Para a criação do cluster, foram utilizadas três instâncias do docker rootless contendo a imagem do mongo. Cada um dos contêineres funcionam em diferentes hosts, a fim de evitar que o mal funcionamento de um único host não afete o sistema. A estrutura de adotada para o cluster é representada na Figura 4.3.

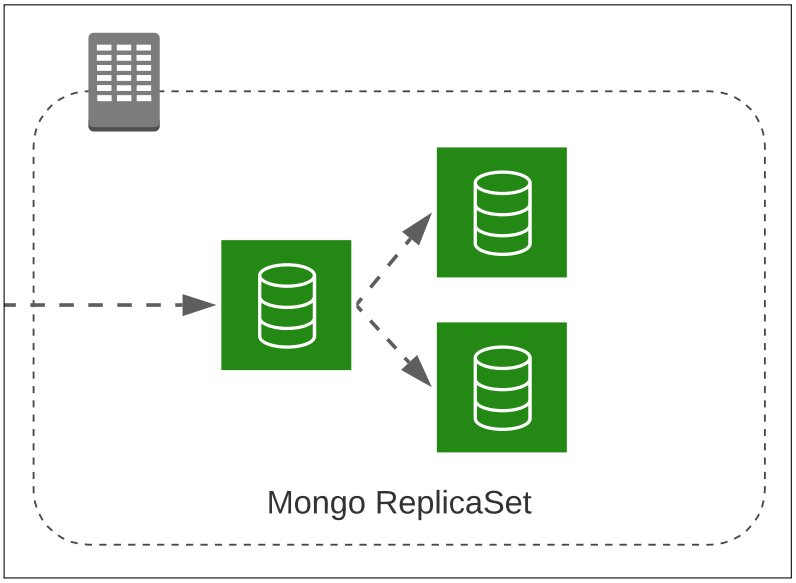

Figura 4.3: Estrutura do Cluster MongoDB.

Para garantir a consistência dos dados, geralmente apenas uma das instâncias contém permissão de escrita, nomeada como master. Então, caso um serviço deseja realizar uma operação de escrita, sua requisição deverá chegar à instância que tem essa permissão. Durante uma requisição de leitura, qualquer servidor poderá responder à solicitação. Existe estratégias para permitir múltiplas escritas como o mongodb Sharding, mas não é o foco desta seção.

A descoberta da instância master é feita automaticamente pelo mongoDB, evitando assim a adição de serviços para proxy reverses e soluções semelhantes. Em outras palavras, caso uma requisição de escrita seja enviada a um serviço do mongo que não é o master, a solicitação automaticamente será repassada ao master. Caso a instância master seja encerrada, o serviço também possui procedimentos para eleger uma das instâncias restantes como a nova instância master.

Durante a criação de cada contêiner do MongoDB, foram expostas e duas portas, uma para a conexão com o banco, e outra para o serviço de SSH. Através da porta de conexão com o banco, os serviços são capazes de enviar requisições para obtenção, exclusão ou alteração dos dados enquanto que a porta do serviço SSH é dedicada apenas à conexão com o ClusterControl.

A segurança entre as instâncias foram configuradas através de um arquivo denominado como keyfile. Os comandos referentes a implementação foram disponibilizados no Anexo A.1 O arquivo fornece configurações mínimas para segurança do cluster através da autenticação interna, mais recomendados durante os períodos de o desenvolvimento e testes [47].

As etapas seguidas para a criação da instância master foram:

- Criação do arquivo keyfile ;
- Criação do contêiner do MongoDB;
- Criação das credenciais de acesso do MongoDB;
- Habilitar o uso do keyfile;
- Iniciar o conjunto de replicas através da interface de linha de comando do MongoDB.

Para que uma aplicação tenha permissão para realizar consultas e escritas nas coleções do MongoDB, é desejável configurar os utilizadores do banco. Ao definir utilizadores, podemos controlar as permissões e limites de cada um, além de barrar a conexão com aplicações que não possuem credenciais cadastradas. O mongoDB não obriga a criação

de usuários root e administradores, mas fazê-los é uma boa prática para a proteção do banco.

Foram definidos quatro credenciais para o cluster, sendo o root e o administrador capazes de gerenciar o conjunto de réplicas e as credenciais de acesso, o utilizador que permite leitura e escrita em um banco específico e um utilizador para o ClusterControl realizar o monitoramento do estado dos nós.

Após a configuração da instância master, foram criadas as instâncias de réplica utilizando a autenticação via keyfile, além de informar os hosts participantes e nome do conjunto de réplicas.

4.3.3 Serviço ClusterControl

O serviço ClusterControl oferece operações automatizadas para clusters de dados além de ser uma ferramenta capaz de monitorar o estado de cluster e de cada host. Dentro do sistema proposto, o ClusterControl realiza o monitoramento de dois clusters Postgres e um cluster mongoDB, onde cada cluster contém três instâncias conteinerizadas, como representado pela Figura 4.4.

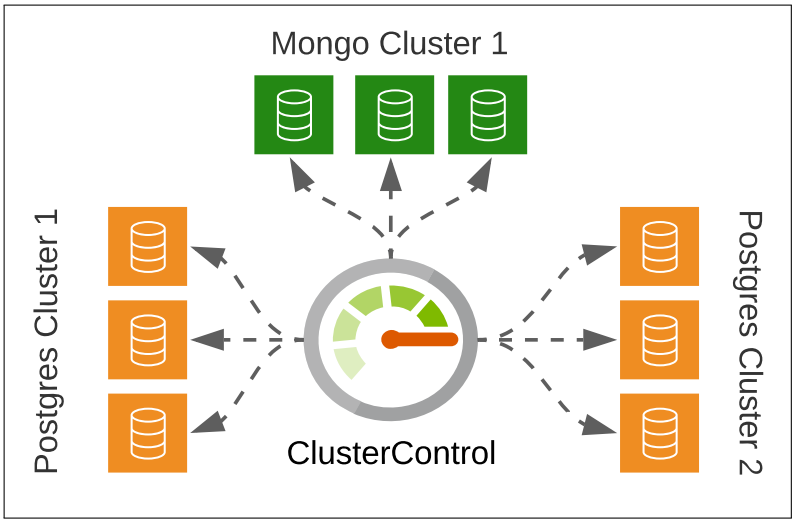

Figura 4.4: Conexões do ClusterControl.

O serviço é capaz de automatizar as operações para a criação de clusters, porém traz

algumas limitações durante o processo. As operações automatizadas para implementação dos clusters não possuem um suporte adequado para bancos conteinerizados. Para realizar a integração, foi necessário desenvolver os clusters manualmente, e incluí-los ao serviço para utilizar as outras funcionalidades fornecidas, como é o caso do monitoramento dos nós.

Para que a integração entre o ClusterControl e as instâncias de serviços de bancos de dados ocorresse, foi necessário expor uma porta para o serviço de SSH em cada um dos contêineres, além de realizar as configurações para permitir que o ClusterControl envie instruções a cada um dos contêineres via SSH.

Configuração de acesso SSH

O ClusterControl exige que os hosts pertencentes a um cluster tenham as mesmas características e configurações. Por isso, os contêineres de banco de dados devem conter imagens com a mesma versão. As portas mapeadas para acesso ao banco e para SSH também deverão ser as mesmas, para que não ocorra dificuldades durante a integração.

Para realizar as configurações do serviço de SSH, em cada contêiner foi seguido as etapas:

- Exposição da porta do serviço SSH durante a criação do contêiner, uma vez que o Docker não permite a abertura de novas portas após a criação do contêiner;
- Criação de utilizador e senha padrão, para que todos os contêineres possuam as mesmas configurações de credenciais;
- Instalação do servidor de SSH, para que o contêiner permita o acesso via SSH;
- Criação das chaves públicas e privadas, que serão utilizadas para autenticação sem a necessidade do uso de senhas;
- Configuração de autenticação passwordless;
  - Habilitar autenticação passwordless, permitindo que os hosts sejam conectados através do reconhecimento de suas chaves;

  - Configurar arquivo `authorized_keys`, onde são listadas todas as chaves públicas dos serviços que têm permissão de se realizar o acesso.

O local onde é salvo as chaves de acesso via SSH devem ser iguais ao utilizado no ClusterControl. Caso um dos contêineres não esteja utilizando o mesmo endereço de diretório para as credenciais, a conexão não acontecerá.

Incluindo um cluster ao ClusterControl

Uma vez que os clusters já tinham sido criados e configurados manualmente, foi possível realizar a integração ao serviço do ClusterControl através da inclusão de clusters já existentes. A integração dos clusters Postgres e MongoDB estão documentadas nos Anexos A.1 e A.2. Os processos ocorrem de modo semelhante, seguindo as etapas:

- Configuração de acesso SSH;
  - Informar nome do utilizador do SSH, para que seja possível a conexão;
  - Informar o diretório padrão onde são salvas as chaves publicas e privadas , iguais para todos os nós do cluster e o ClusterControl;
  - Informar a porta de conexão via SSH mapeada nos hosts, sendo iguais entre todos nós do cluster;
- Configuração de acesso ao banco;
  - Informar credenciais para o acesso ao banco, sendo composto por nome de utilizador, senha e tabela ou coleção utilizada para autenticação;
  - Informar a porta mapeada no host para a conexão com o banco de dados;
- Configuração dos hosts;
  - Informar cada endereço dos hosts pertencentes ao cluster.

Após preenchido todas as informações, o ClusterControl utilizará a lista de endereços, porta SSH, utilizador e chaves para validar as conexões via SSH. Em caso de sucesso da

conexão, fará a tentativa de acesso aos serviços do banco através das credenciais e da porta fornecida. No final do processo, fará a inclusão do cluster.

Pode ocorrer falhas durante o processo devido a configuração do SSH, rejeitando a conexão por não reconhecer a chave pública, configuração da firewall, bloqueando alguma das portas utilizadas, definições do Domain Name System (DNS), não reconhecendo o domínio dos servidores, proxy reverse, realizando erroneamente o roteamento para o host, etc. É importante verificar se a estrutura está configurada corretamente, permitindo a troca de mensagem entre os serviços

4.3.4 Conteinerização de APIs

Para permitir a integração entre as APIs desenvolvidas com o Kubernetes, é necessário conteinerizar a aplicação. As APIs desenvolvidas utilizam python como linguagem, e foram implementadas através do mesmo framework, Flask. Por isso, o arquivo Dockerfile é capaz de gerar a imagem da APIs de aprendizagem de máquina e o Dockerfile para a APIs de gerenciamento são similares. Os arquivos e comandos necessários para realizar o deploy das APIs foram documentados nos Anexos A.4 e A.5.

O arquivo Dockerfile realiza as seguintes etapas:

- Download da imagem do Python 3.8 para executar a aplicação;
- Definição do diretório onde contém todos os arquivos da API;
- Criação de cópias dos arquivos da API dentro do contêiner;
- Instalações das bibliotecas necessárias para o funcionamento da API através do gerenciador de pacotes PIP;
- Execução do script python responsável por iniciar a API.

A imagem será criada ao utilizar o comando build do docker. Caso ocorra algum erro durante o processo, o docker informará quais os motivos de falha durante a construção da imagem.

Durante o processo de conteinerização, não foi inserido valores para as variáveis de ambiente diretamente no Dockerfile. As informações são adicionadas posteriormente, através de serviços específicos do kubernetes.

4.3.5 Implementação do Kubernetes

A implementação do Kubernetes foi realizada através do minikube, com o objetivo de criar um ambiente no servidor local. O kubernetes foi utilizado para realizar o deploy dos serviços de API de gerenciamento da unidade de saúde, API de aprendizado de máquina e o deploy do Keycloak como serviço de SSO.

Para cada um dos serviços principais mencionados foi seguido o mesmo padrão de integração, representado na figura 4.5. O kubernetes fornece diversos serviços voltados à arquitetura, facilitando a escalabilidade, manutenção e contendo práticas voltadas a segurança. Para cada um dos serviços principais do sistema (APIs e SSO), foram adicionados os serviços do Kubernetes, sendo eles: Secrets, ConfigMap, Service, Deployment e Pods. As configurações utilizadas foram disponibilizadas no Anexo A.5, A.4 e A.6.

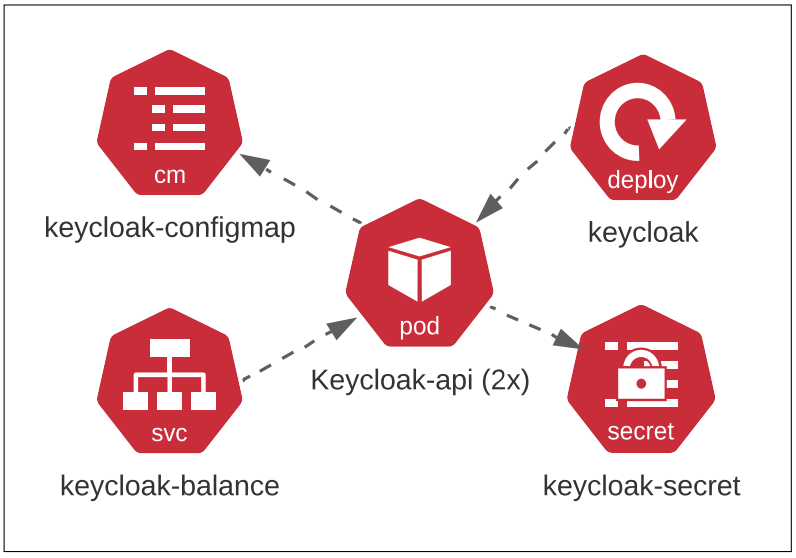

Figura 4.5: Padrão de integração de serviços do Kubernetes.

ConfigMap

O serviço configMap foi utilizado para fornecer as chaves publicas de cada serviço e configurações de variáveis de ambiente que podem ser compartilhadas. Dessa forma, os serviços podem consumir os dados sem que hajam dificuldades.

Ao realizar atualizações nos serviços ConfigMap, todos os outros contêineres terão acesso às novas informações. O ConfigMap é um serviço utilizado apenas internamente, não tendo acesso diretamente pela internet. Para que não haja mudanças de configuração, suas informações foram definidas como imutáveis. Dessa forma, apenas a exclusão e criação de um novo configMap permitirá novos valores.

Secrets

Semelhante ao configMap, o serviço secrets foi utilizado para fornecer variáveis de ambiente ao Keycloak, API de gerenciamento e API de aprendizagem de máquina. Porém, os valores armazenados não podem ser compartilhados entre os demais serviços.

O secrets contém informações necessárias para realizar autenticação no banco de dados, além de conter as chaves privadas de cada um dos serviços principais. Suas informações foram definidas como imutável, e dessa forma as mudanças de informações necessitam de exclusão e recriação do serviço.

Deployment

O serviço de Deployment foi implementado para o Keycloak, a API de gerenciamento e a API de aprendizagem de máquina. O serviço gerencia a quantidade de Pods executados pelo sistema, além de fornecer facilidades para a manutenção.

Os Pods são criados através do serviço de Deployment. Durante a configuração do Deployment, é informada a quantidade de Pods e as informações necessárias para que o Pod funcione. O Deployment é capaz de verificar os estados dos Pods, destruí-los e recriá-los automaticamente.

As configurações dos Pods contém os dados necessários para iniciar os contêineres. É

necessário informar a imagem que será utilizada para a criação dos contêineres, além das flags e variáveis de ambiente. Durante a configuração, foi referenciado os outros serviços para complementar os dados, obtendo as informações contidas no configMap e no Secret.

Imagens personalizadas descritas com o arquivo Dockerfile em que foram realizas operação de build localmente não são encontradas pelo kubernetes, pois o minikube utiliza uma máquina virtual tornando a imagem inexistente dentro de seu ambiente. Para utilizar builds locais é necessário configurar o minikube para utilizar o Docker Deamon. A configuração não é permanente, afetando apenas a sessão que está a ser utilizada.

Serviços caracterizados como stateless não são afetados durante a recriação dos pods, já que não guardam estados, sendo completamente substituíveis. O Keycloak armazena em cache informações sobre as sessões de utilizadores, então é um serviço que é afetado em caso de recriação, exclusão ou falha. Durante as configurações do Keycloak é possível definir a quantidade de instâncias para compartilhar os dados em cache, e dessa forma diminui a possibilidade de um utilizador ter sua sessão encerrada devido ao encerramento de um Pod, impactando sua velocidade.

Service

Os Pods são efêmeros, podendo ser destruídos e recriados a qualquer momento. Por conta disso, seus endereços de IP tornam-se extremamente dinâmicos. É inviável referenciá-los através de seus endereços de IP. Tornar o uso de pods uma estratégia viável é necessário recorrer ao Service.

O Service fornecido pelo kubernetes é capaz de selecionar um conjunto de Pods, a fim de realizar o balanceamento de carga entre eles. O Service não costuma ser destruído ou recriado com tanta frequência ao ser comparado com os Pods, fornecendo assim um endereço mais estável.

Foi declarado três Services, o primeiro para satisfazer o conjunto de Pods da API de aprendizagem de máquina, o segundo para satisfazer o cluster dos Pods da API de gerenciamento, e o terceiro sendo responsável pelos Pods do Keycloak.

Ao atualizar a quantidade de Pods através do serviço de Deployment, automaticamente o balanceador de carga contido no Service é modificado, tornando o processo de escalabilidade menos susceptível a erro humano. Caso ocorra falhas de um Pod, o Service é capaz de rotear a requisição a outros Pods pertencentes ao mesmo cluster.

Ingress

O ingress permite realizar o roteamento de requisições para dentro da máquina virtual criada pelo minikube. Diferentemente das soluções padrões de proxy reverse, o Ingress foi elaborado para ser utilizado diretamente com os serviços do Kubernetes, representado pela Figura 4.6.

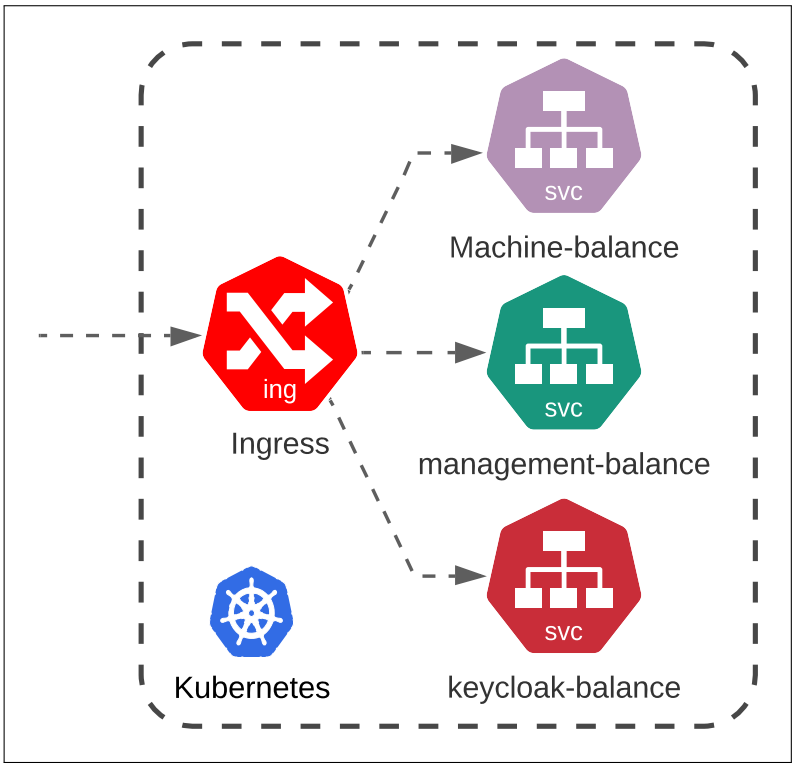

Figura 4.6: Funcionamento do Ingress na solução proposta.

O Ingress necessita de um controlador para seu funcionamento. No projeto foi utilizado o Nginx controller para gerenciar as requisições, pois o minikube já possui integração com o controlador, abastando apenas habilitá-lo.

Para realizar o roteamento das requisições, foi criado um caminho especificado na URL para cada Kubernetes Service. Dessa forma, o Ingress é responsável por reconhecer o padrão descrito na URL e enviar ao Service correto. Por sua vez, o Service realizará o balanceamento de carga entre os Pods disponíveis.

Arquitetura do Kubernetes

Ao reunir todos os serviços implementados no ambiente do kubernetes, é obtida a arquitetura representada na Figura 4.7. É demonstrado o serviço Ingress, capaz de enviar requisições para os Services, que por sua vez realizam o balanceamento de carga para os Pods. Os demais serviços são utilizados para gerenciamento das informações e gerenciamento de Pods.

4.4 Pseudonimização

O uso de pseudonimização é feito para disponibilizar as informações de tratamentos sem divulgar os indivíduos envolvidos no processo. Para o caso adotado, foi optado por fazer uso de um único pseudônimo por utilizador, já que são armazenados múltiplos registros de tratamentos para um único paciente, e o serviço de aprendizado de máquina deve ser capaz de reconhecê-los.

Os registros de um paciente servem como uma base de dados histórica para as reações que um paciente demonstra mediante a um tratamento, sendo investigado por um profissional de saúde e pelo serviço de aprendizagem de máquina. Utilizar múltiplos pseudônimos por utilizador afeta os processos realizados pela inteligência artificial, por identificar todos os tratamentos de um utilizador como sendo realizados por pessoas distintas, uma vez que o serviço não possui permissão para obter as identidades reais.

Durante o armazenamento, os dados são separados antes de serem armazenados. Em um banco de dados, é feito o armazenamento dos utilizadores, papéis dentro do sistema, consultas agendadas. Em outra base de dados é feito o armazenamento das informações

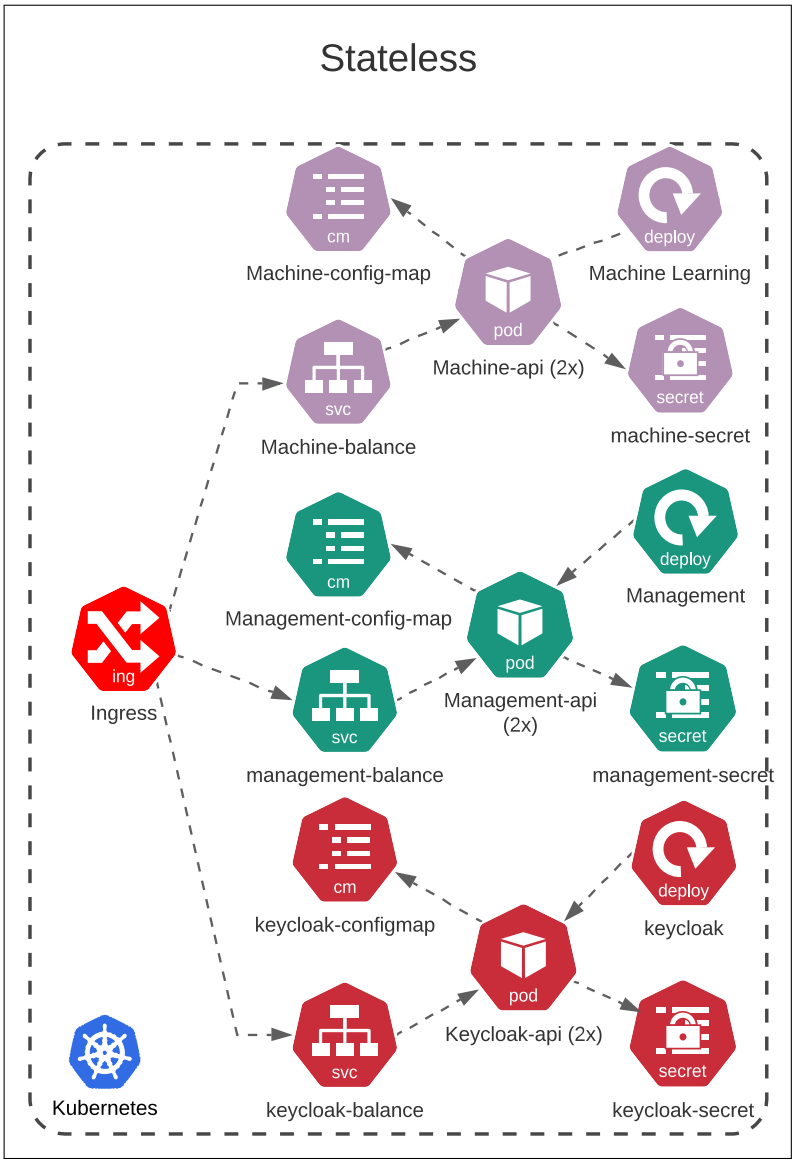

Figura 4.7: Arquitetura final do Kubernetes.

referentes aos tratamentos de modo pseudonimizado, contendo as referências para os utilizadores reais. Os dados de tratamentos são formados pela coleta do dispositivo vestível, comentários do profissional de saúde e do paciente sobre o tratamento.

Para gerar o pseudônimo de um utilizador, foram seguidas as etapas:

- Gerar uma cadeia de caracteres aleatória e de tamanho fixo, sendo esse o possível valor para ser utilizado como pseudônimo;
- Verificar se o valor gerado já pertence a algum outro utilizador, evitando colisões de pseudônimos;
- Inserir o pseudônimo e o identificador real em uma tabela, para ser possível realizar o mapeamento dos valores quando necessário.

Ao possuir o identificador real e as devidas permissões, é possível utilizar a tabela de mapeamento para obter o pseudônimo correspondente ao utilizador. O processo contrário também é possível, e ainda necessita de permissões específicas para fazê-lo.

Por se tratar de uma cadeia de caracteres aleatórios, as sequências são improváveis. Utilizar contadores incrementais ou padrões durante a geração da cadeia de caracteres torna mais fácil para um atacante sugerir pseudônimos válidos durante uma tentativa de ataque.

A função de pseudonimização ocorre internamente no serviço de gerenciamento de utilizadores. Assim, não é possível interagir diretamente com as opções de pseudonimização. Dessa forma torna-se mais difícil a exploração da funcionalidade.

4.5 Interações do sistema

A interação do sistema é feita através da troca de mensagens utilizando o protocolo HTTP, e o formato JSON. O sistema também faz uso de JWT quando a informação deve ser transmitida através de um meio inseguro.

Os JWT emitidos pelo sistema contém as informações de qual serviço o gerou e tempo de expiração, além dos demais dados para cada funcionalidade. Os tokens funcionam como

uma autorização temporária assinada por um serviço, garantindo que outros serviços serão capazes de reconhecer a autoria para confiar nos dados.

Foi utilizado dois tipos de tokens para utilizadores. O primeiro token trata-se dos dados de autenticação emitidos pelo keycloak, informando que o utilizador está autenticado, contendo algumas informações pessoais, como nome, identificador e papéis no sistema. O segundo token contém o pseudônimo do utilizador e suas permissões, sendo emitido e assinado pela API de gerenciamento de recursos hospitalares, para ser utilizado nos serviços em que a identificação ocorre através do pseudônimo.

O token real emitido pelo keycloak permite que o utilizador realize operações na API de gerenciamento, obtendo as consultas agendadas, informações pessoais enquanto que o token com o pseudônimo permite a execução das funções provenientes do serviço de aprendizado de máquina.

Para exemplificar o funcionamento, é mostrado na Figura 4.8 um diagrama de sequência que representa as etapas necessárias para realizar a autenticação com o *Keycloak*. É possível perceber que as informações transitam entre múltiplos serviços de roteamento antes de chegarem ao seu destino.

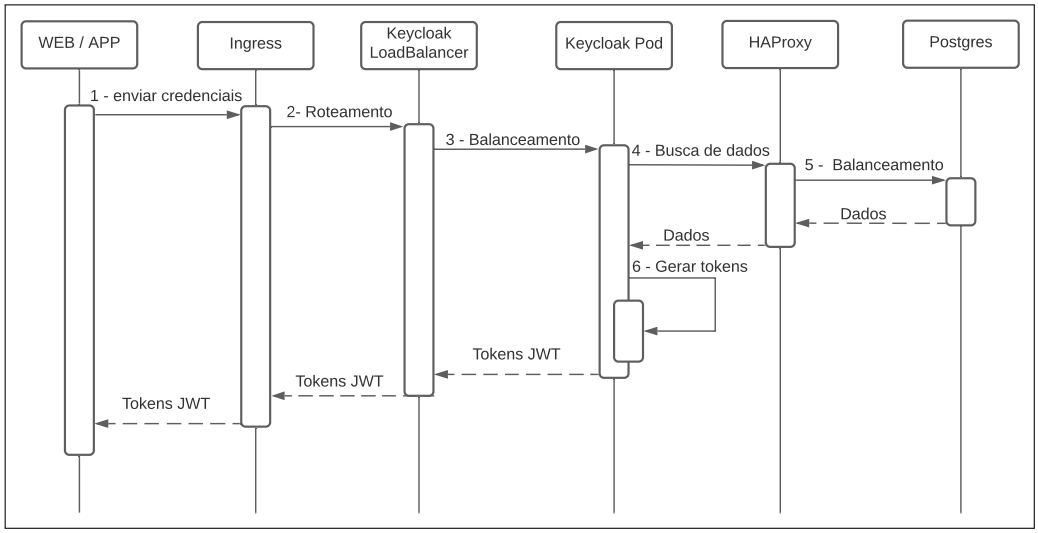

Figura 4.8: Diagrama de sequência para realizar autenticação

O processo demonstrado na Figura 4.8 é necessário para identificar um utilizador do sistema. No token contém algumas informações sobre o utilizador, como função no

sistema, nome e identificador. Apenas o *Keycloak* é capaz de gerar o token, mas todos os serviços podem verificar a autenticidade do token. Na Figura 4.9 é mostrado como a validação de um token é feita dentro da plataforma.

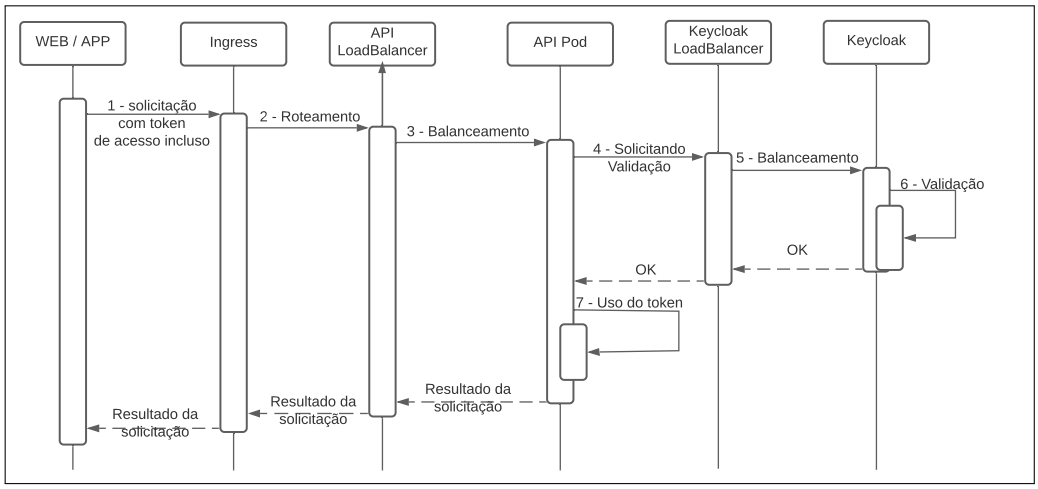

Figura 4.9: Diagrama de sequência para validação e uso do token de acesso

O *keycloak* utiliza a criptografia RSA para gerar e validar tokens. Na estratégia adotada, a validação do token está centrada nos Pods do próprio *Keycloak*, porém também é possível optar por descentralizar a verificação, já que a chave publica do *keycloak* pode ser utilizada para esse fim. A estratégia de verificação descentralizada ocorre com o token que contém o pseudônimo, e seu diagrama de sequência é apresentado na Figura 4.10.

A descentralização da validação de tokens possui algumas desvantagens ao ser aplicado com o serviço de autenticação. O *keycloak* permite revogar tokens utilizados na plataforma mesmo que não estejam expirados. Porém, ao utilizar a validação descentralizada, é levado em consideração apenas o tempo de validade do token e sua autenticidade. Dessa forma os tokens continuam a funcionar em outros microsserviços, mesmo depois de revogados pelo *Keycloak*.

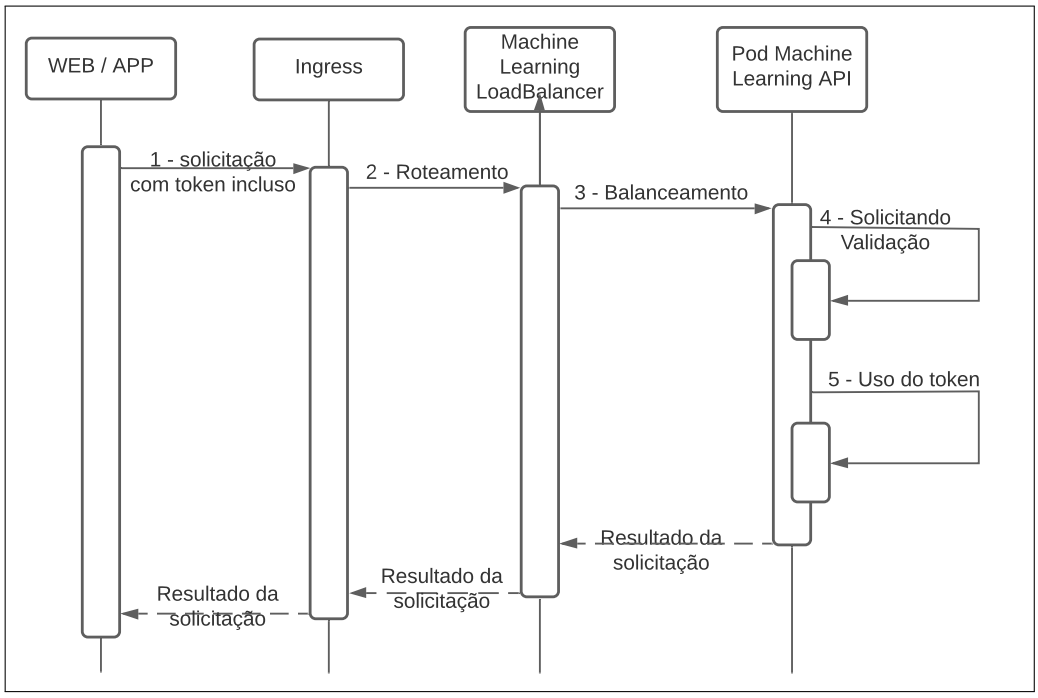

Figura 4.10: Diagrama de sequência para validação e uso do token com o pseudônimo

Capítulo 5

Análise

O capítulo atual descreve as análises realizadas do projeto NanoStim e arquitetura. As análises são utilizadas para garantir estratégias de segurança funcionais ao NanoStim e sua arquitetura, de modo a suprir os requisitos de desejados. Algumas das análises foram realizadas mediante testes, que também encontram-se descritos ao decorrer do capítulo.

Inicialmente, é abordado o teste de disponibilidade dos principais recursos da plataforma, que são as APIs contidas nos Pods do Kubernetes, e os serviços para gerenciamento de dados. A seguir são aparesentados os testes referentes à funcionalidade das estratégias de comunicação autenticada entre múltiplos serviços em um cenário onde ocorre picos de solicitações a um serviço específico. No final, é apresentado a modelagem de ameaças STRIDE que envolvem o projeto.

5.1 Teste de Disponibilidade

Nesta seção são apresentados os testes de disponibilidade dos principais componentes da arquitetura. Em cada subseção é descrito o que está sendo avaliado no teste, e suas etapas de execução.

Inicialmente é apresentado o comportamento da arquitetura diante da ocorrência de falhas dos Pods definidos dentro do kubernetes, onde contém os serviços relacionados ao aprendizado de máquina, autenticação e gerenciamento de recursos hospitalares. Em

seguida é apresentado o teste realizado para simular a ocorrência de falhas na instância primária do MongoDB e seu comportamento. No final, é descrito o teste referente à disponibilidade do cluster formado por instâncias do Postgres, informando também as consequências da falha ao ter em mente a integração com o serviço HAProxy.

5.1.1 Falha de Pods

Os serviços de gerenciamento de informações da unidade de saúde, a API relacionada ao serviço de aprendizagem de máquina e o serviço responsável pela autenticação encontram-se dentro do ambiente do kubernetes, implementados em Pods. Ao ocorrer falhas graves em um Pod, o mesmo pode ser automaticamente excluído e recriado, por serem serviços efêmeros.

O objetivo de testar o comportamento da arquitetura ao ocorrer uma falha em um Pod é detectar o tempo que um serviço poderá ficar indisponível, tendo que a falha de um serviço também poderá prejudicar os serviços vizinhos. Para simular um erro grave, foi seguido as etapas:

- Iniciar todos os serviços, deixando todos já funcionais;
- Assistir aos eventos disparados pelo Pods;
- Realizar a uma exclusão de um Pod diretamente.

Na configuração da plataforma, os Pods são gerenciados pelo serviço de Deployment oferecido no ambiente do Kubernetes. Ao excluir um dos Pods para simular um erro, o serviço de Deployment detectará que a quantidade de instâncias está incorreta, tomando as providências para criar uma nova.

Durante os eventos de ocorrência de erro, exclusão do contêiner, detecção de quantidades não compatíveis com as declaradas no Deployment, inicio do novo Pod e finalização dos processos iniciais do contêiner, ocorre um gasto de tempo. Durante esse tempo, a arquitetura deve funcionar com uma quantidade reduzida de Pods. O problema se agrava caso mais de um Pod do mesmo serviço apresente falha dentro do período, tornando a

plataforma incapaz de suprir as requisições até que ao menos uma das instâncias esteja pronta.

Os dados obtidos ao realizar os testes na API de gerenciamento de informações hospitalares é demonstrado na Tabela 5.1. Durante os dois testes de exclusão demonstrados observados na tabela, é possível perceber que os Pods demoraram cerca de dois segundos para se reestabelecerem. Mesmo assim, durante o período indisponível, ainda havia mais um Pod capaz de fornecer as funcionalidades. As mesmas características ocorreram durante o teste com a API de aprendizado de máquina, representado na Tabela 5.2, porém apresentando o tempo de aproximadamente quatro segundos para se estabilizar.

O aumento de Pods reduziria a chance de que os serviços fiquem indisponíveis, porém ainda há as limitações de hardware, uma vez que os pods estão sendo executados na única máquina virtual disponibilizada para isso. Tendo acesso a mais máquinas, é possível utilizar as mesmas configurações, e dessa forma o aumento de serviços pode fazer uso de mais recursos, como processamento e memória RAM. O aumento de quantidades de pods em uma única máquina virtual consequentemente aumentará o uso de recursos dos recursos.

Os testes realizados com o Keycloak, apresentados na Tabela 5.3, demostram que a recriação dos serviços ocorre de modo mais demorados. Os processos de levantamento de uma instância envolvem a detecção e sincronia com as outras instâncias do keycloak disponível na mesma rede, além dos processos de sincronia com o banco de dados. Os testes mostraram que a plataforma demorará cerca de cinquenta segundos para reestabelecer o número de instâncias.

5.1.2 Falha em instância do MongoDB

Pode ocorrer algum momento em que a instância que realiza o papel de master dentro do cluster pare de funcionar. A instância master é a única capaz de realizar a escrita em clusters, enquanto que as outras instâncias são capazes de realizar leitura.

Hora	**Pod ID**	**Status**	**Hora**	**Age**
09:30:47.123	management-api-6b8467fd4f-cf4n6	READY	Running	7m9s
09:30:47.124	management-api-6b8467fd4f-qtlpt	READY	Running	16h
09:31:28.729	management-api-6b8467fd4f-qtlpt	READY	Terminating	16h
09:31:28.764	management-api-6b8467fd4f-49w69	STOP	Pending	0s
09:31:28.798	management-api-6b8467fd4f-49w69	STOP	Pending	0s
09:31:28.877	management-api-6b8467fd4f-49w69	STOP	ContainerCreating	0s
09:31:30.650	management-api-6b8467fd4f-49w69	READY	Running	2s
09:31:59.934	management-api-6b8467fd4f-qtlpt	STOP	Terminating	16h
09:31:59.958	management-api-6b8467fd4f-qtlpt	STOP	Terminating	16h
09:32:00.020	management-api-6b8467fd4f-qtlpt	STOP	Terminating	16h
09:32:42.700	management-api-6b8467fd4f-49w69	READY	Terminating	74s
09:32:42.750	management-api-6b8467fd4f-ncwkj	STOP	Pending	0s
09:32:42.758	management-api-6b8467fd4f-ncwkj	STOP	Pending	0s
09:32:42.773	management-api-6b8467fd4f-ncwkj	STOP	ContainerCreating	0s
09:32:44.315	management-api-6b8467fd4f-ncwkj	READY	Running	2s
09:33:13.607	management-api-6b8467fd4f-49w69	STOP	Terminating	105s
09:33:13.624	management-api-6b8467fd4f-49w69	STOP	Terminating	105s
09:33:13.679	management-api-6b8467fd4f-49w69	STOP	Terminating	105s

Tabela 5.1: Tempo de recriação do pod com a API de gerenciamento de informações hospitalares

Hora	**Pod ID**	**Status**	**Hora**	**Age**
09:36:20.626	machine-api-7b6cf4f867-755rx	READY	Running	6m33s
09:36:20.627	machine-api-7b6cf4f867-mpxr9	READY	Running	17h
09:38:17.724	machine-api-7b6cf4f867-755rx	READY	Terminating	8m30s
09:38:17.760	machine-api-7b6cf4f867-kssqb	STOP	Pending	0s
09:38:17.784	machine-api-7b6cf4f867-kssqb	STOP	Pending	0s
09:38:17.897	machine-api-7b6cf4f867-kssqb	STOP	ContainerCreating	0s
09:38:20.232	machine-api-7b6cf4f867-kssqb	READY	Running	3s
09:38:49.495	machine-api-7b6cf4f867-755rx	STOP	Terminating	9m2s
09:38:49.519	machine-api-7b6cf4f867-755rx	STOP	Terminating	9m2s
09:38:49.523	machine-api-7b6cf4f867-755rx	STOP	Terminating	9m2s
09:38:56.733	machine-api-7b6cf4f867-kssqb	READY	Terminating	39s
09:38:56.850	machine-api-7b6cf4f867-469bc	STOP	Pending	0s
09:38:56.858	machine-api-7b6cf4f867-469bc	STOP	Pending	0s
09:38:56.874	machine-api-7b6cf4f867-469bc	STOP	ContainerCreating	0s
09:39:00.064	machine-api-7b6cf4f867-469bc	READY	Running	4s
09:39:28.362	machine-api-7b6cf4f867-kssqb	STOP	Terminating	71s
09:39:28.407	machine-api-7b6cf4f867-kssqb	STOP	Terminating	71s
09:39:28.457	machine-api-7b6cf4f867-kssqb	STOP	Terminating	71s

Tabela 5.2: Tempo de recriação do pod com a API de aprendizagem de máquina

Hora	Pod ID	Status	Hora	Age
09:41:23.445	keycloak-698695f58c-hpqmt	READY	Running	15h
09:41:23.447	keycloak-698695f58c-xcnqp	READY	Running	15h
09:41:47.657	keycloak-698695f58c-hpqmt	READY	Terminating	15h
09:41:47.701	keycloak-698695f58c-76rf6	STOP	Pending	0s
09:41:47.709	keycloak-698695f58c-76rf6	STOP	Pending	0s
09:41:47.769	keycloak-698695f58c-76rf6	STOP	ContainerCreating	0s
09:41:49.571	keycloak-698695f58c-hpqmt	STOP	Terminating	15h
09:41:49.700	keycloak-698695f58c-hpqmt	STOP	Terminating	15h
09:41:49.865	keycloak-698695f58c-hpqmt	STOP	Terminating	15h
09:41:49.877	keycloak-698695f58c-76rf6	STOP	Running	2s
09:42:37.748	keycloak-698695f58c-76rf6	READY	Running	50s
09:42:59.724	keycloak-698695f58c-76rf6	READY	Terminating	72s
09:43:00.084	keycloak-698695f58c-gfhnx	STOP	Pending	0s
09:43:00.097	keycloak-698695f58c-gfhnx	STOP	Pending	0s
09:43:00.154	keycloak-698695f58c-gfhnx	STOP	ContainerCreating	0s
09:43:01.234	keycloak-698695f58c-76rf6	STOP	Terminating	74s
09:43:01.265	keycloak-698695f58c-76rf6	STOP	Terminating	74s
09:43:01.309	keycloak-698695f58c-76rf6	STOP	Terminating	74s
09:43:02.269	keycloak-698695f58c-gfhnx	STOP	Running	2s
09:43:40.708	keycloak-698695f58c-gfhnx	READY	Running	40s

Tabela 5.3: Tempo de recriação do pod com Keycloak

O objetivo do teste atual é verificar o comportamento da arquitetura quando a instância de escrita do MongoDB pára de funcionar. Para simular a ocorrência de um erro grave, foi finalizado o contêiner que havia permissões de escrita. Dessa forma, os outros membros do cluster deveriam realizar a votação para eleger um novo master.

Para a análise, foram seguidas as etapas:

- Iniciar todos os nós para que o cluster já esteja em funcionamento antes da realização do teste;
- Verificar o status do cluster, coletando as informações antes de realizar o teste;
- Finalizar o contêiner onde se encontra a instância master do cluster, forçando a ocorrência de votação entre os nós subjacentes;
- verificar através de um dos nós restantes quais foram as alterações ocorridas no status do cluster, para garantir que há uma nova instância master.

O status do cluster antes da realização do teste é demonstrado na Figura 5.1. Para simplificar a compreensão, foi removido outros atributos devolvidos pelo MongoDB. É possível perceber que todos os nós estão funcionando, indicado por `health : 1`, além de informar qual a instância é a master, indicado por `stateStr : "PRIMARY"`, e a sua data de eleição.

Ao finalizar o contêiner que executava a instância master, foi verificado novamente as configurações da réplica. Os dados obtidos são demonstrados na Figura 5.2. É possível perceber que a instância primária foi atribuída a outro nó, e que a instância um dos nós não está mais saudável, uma vez que não está sendo executado o contêiner. Também ocorreu a mudança na data de última votação.

Ao executar o contêiner que havia sido parado, não há garantia de que se torne o master novamente durante seu processo de inclusão no cluster. Normalmente, a eleição de um novo nó primário só ocorre quando o cluster não encontra uma instância master saudável ou alcançável. Porém, o mongoDB permite definir uma instância como primária através de sua linha de comando.

Figura 5.1: Informações do conjunto de Réplica MongoDB antes da realização do teste.

Os contêineres de todos os bancos foram configurados para auto-reiniciar em caso de falhas reais. Um contêiner não será reiniciado ao ser parado manualmente. Se o erro ocorre apenas no contêiner, a instância master se reabilita após reiniciar automaticamente. A integração ao clusterControl, serviço que monitora o funcionamento do cluster, continua configurada, capaz de identificar o momento que a instância está pronta novamente.

5.1.3 Falha em instância do Postgres

O objetivo do teste atual está em identificar os comportamentos ocorridos na arquitetura como consequência das configurações escolhidas. Para o teste, são considerados três fatores principais para o funcionamento, sendo eles o HAProxy, o xinetd e o próprio postgres.

Para realizar o teste, foram seguidas as etapas:

- Iniciar o cluster completamente, de modo que todas as instâncias, processos e o HAProxy estejam funcionando devidamente;

Figura 5.2: Informações do conjunto de Réplica MongoDB após a realização do teste.

- Parar manualmente o contêiner onde se encontra a instância master, para simular a ocorrência de um erro grave;
- Testar a conexão para escritas através do HAProxy;
- Testar a conexão para leituras através do HAProxy;

Um cluster Postgres também realiza escritas através de uma instância master, e leituras em quaisquer instâncias, como ocorre com o MongoDB. As configurações utilizadas no cluster foram definidas para o funcionamento em modo hot standby, onde as réplicas não são capazes de substituir a master. Dessa forma, não ocorre votação para estabelecer uma nova instância capaz de escrita, em caso de não funcionamento da master. Para permitir a votação automática, o cluster deve ser configurado como warm standby.

O HAProxy foi configurado para utilizar duas portas, uma para redirecionar as operações de leitura à instância master, enquanto que a outra é capaz de realizar o balanceamento de carga para todas as instâncias quando é requisitado para as operações de

leitura. Para que seja procurado a instância master, o HAProxy se comunica com um processo do xinetd que informa qual o status da instância. O xinetd não é incluso na imagem oficial do Postgres, então o mesmo foi instalado e iniciado manualmente em cada instância.

Cada contêiner foi configurado para reiniciar em caso de erros. Porém, uma vez que o xinetd não foi configurado para iniciar juntamente com o contêiner, a integração com o HAProxy é desfeita, fazendo com que o HAProxy reconheça a instância como inativa ou não saudável. Para resolver o problema deve-se incluir a inicialização por padrão do xinetd em um arquivo Dockerfile.

Ao parar a instância master, foi feito uma tentativa de conexão remota com alguma instância de leitura através do HAProxy. A conexão foi estabelecida com sucesso, demonstrando que o HAProxy foi capaz de identificar uma das instâncias através do processo do xinetd.

Em seguida, foi testada a conexão remota com a instância master através do HAProxy, que se encontrava desativada. A tentativa falhou, uma vez que as configurações de votação estavam desabilitadas no cluster. Em outras palavras, o HAProxy não encontrou uma instância que havia permissão para escrita dentro do cluster, e dessa forma foi notificado o seguinte erro: **psql: error: server closed the connection unexpectedly. This probably means the server terminated abnormally before or while processing the request.** Ao reiniciar a instância, o mesmo erro continuará sendo exibido enquanto que o processo xinetd estiver desabilitado.

O cluster pode ficar indisponível temporariamente para escrita, porém as funcionalidades que utilizam apenas leitura continuarão funcionando enquanto houver uma instância disponível no cluster. O cluster se torna inacessível ao desativar o HAProxy, uma vez que o serviço é utilizado como intermediador entre as APIs e o banco de dados. Também sofrem de ataques físicos, desligamento da máquina virtual ou dos servidores do IPB, entre outros.

5.2 Teste funcional

Esta sessão aborda o teste funcional que valida as estratégias de comunicação da plataforma. É analisado o processo necessário para autenticação através do *keycloak*, representado pela Figura 4.8, a validação e uso do token de acesso emitido pelo *Keycloak*, representado na Figura 4.9, e a validação e uso do pseudônimo, Figura 4.10.

Para isso, foram escolhidos três *endpoints*, um de cada API, sendo eles:

- o *endpoint* responsável pela realização de autenticação de um paciente no serviço *Keycloak*, que recebe as credenciais e retorna o token de acesso;
- o *endpoint* disponível na API de gerenciamento responsável por gerar o token de acesso com o pseudônimo incluso, sendo uma rota autenticada;
- o *endpoint* que utiliza o token com o pseudônimo para realizar a consulta de tratamentos de um paciente, disponível no serviço de aprendizagem de máquina.

Além da validação das estratégias de comunicação, também foi testado o comportamento da plataforma ao sofrer picos de solicitação em cada uma das APIs. Os recursos computacionais da plataforma precisa suprir grandes quantidades de requisições para se precaver de ataques do tipo Distributed Denial of Service (DDoS).

Para simular o fluxo de informação, foi utilizado um *script* desenvolvido em python com o objetivo de enviar diversas vezes a mesma requisição, fazendo com que o servidor tenha maiores gastos de recursos para responder. O *script* está disponível no A.7. Para que o fluxo de trabalho do servidor se torne maior, todas as requisições enviadas foram configuradas como assíncronas, permitindo ao *script* uma vazão de dados maior. O processo foi executado em um portátil com acesso remoto aos serviços.

Antes de iniciar o teste, a plataforma já consumia recursos devido ao uso de contêineres, kubernetes, e os demais serviços habilitados. Para exemplificar o consumo de recursos, foi registrado o estado do servidor antes e durante os testes, exibido nas Figuras 5.3 e 5.4 respectivamente.

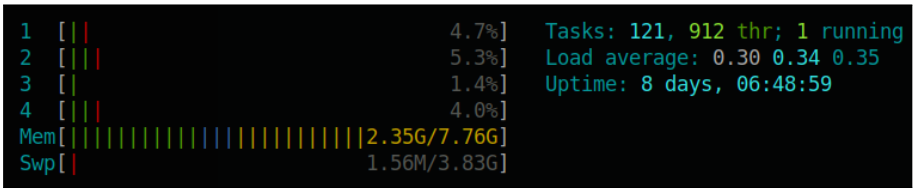

Figura 5.3: Recursos antes do início do teste.

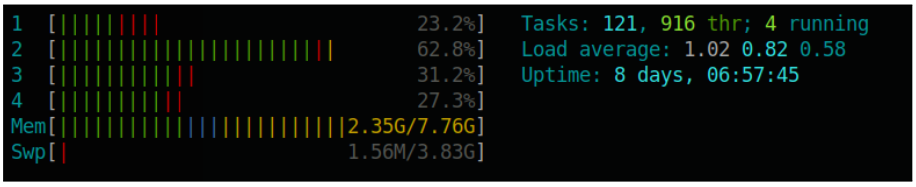

Figura 5.4: Recursos durante a execução do teste.

O envio das requisições ocorreram em ciclos. Foi estabelecido como valor inicial o envio de dez requisições assíncronas a um serviço específico. Ao finalizar, foi enviado a mesma quantidade de requisições aos restantes serviços, um de cada vez. Quando todos os serviços foram testados com a quantia, o primeiro ciclo foi encerrado. A cada ciclo de testes, a quantidade de requisições foi incrementada em 100 unidades, fazendo que o segundo ciclo realizasse 110 solicitações, o terceiro com 220, seguindo o mesmo padrão.

Para identificar a ocorrência de atraso no tempo de resposta, foi registrado o momento do envio da solicitação, e o momento do recebimento da resposta. Em seguida a resposta foi validada, para garantir que a resposta está correta, e calculado o tempo gasto entre o envio e o recebimento. No final, foi calculado o tempo médio de resposta em cada ciclo. Os resultados obtidos são exibidos na Figura 5.5.

Os resultados mostram que para a quantidade de requisições testadas não houve problemas com o tempo médio de resposta. Porém, ainda existem fatores que impõe limites no teste aplicado, como a internet, o hardware e a quantidade de computadores emissores. Ataques DDoS utilizam vários dispositivos para realizar diversas requisições. Também foi possível confirmar o funcionamento das estratégias de comunicação.

É possível perceber que os *endpoints* possuem tempos diferentes para realizar o processamento da resposta. O tempo médio é influenciado diretamente pela quantidade de instruções realizadas pela rota, que podem incluir validação, chamadas a bancos de dados

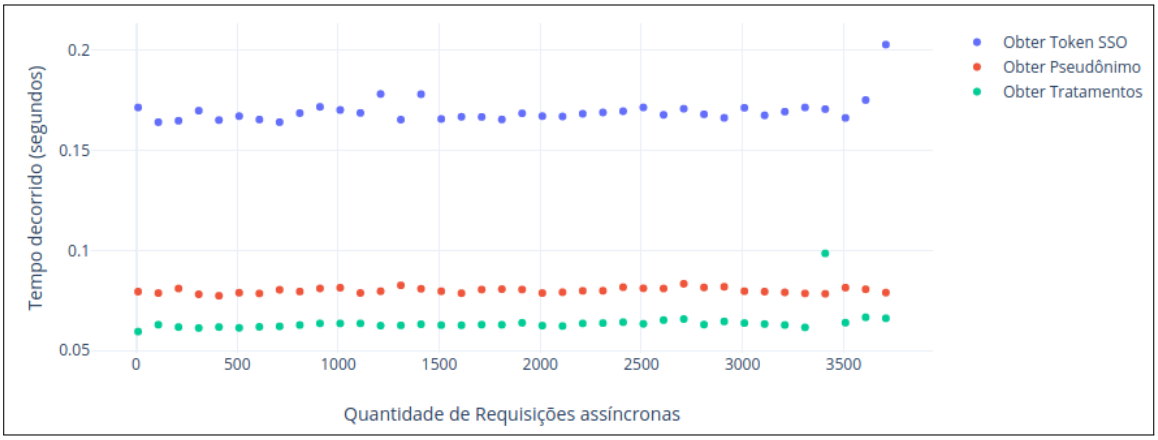

Figura 5.5: Tempo médio de resposta à solicitações.

e chamadas à outros serviços da plataforma. Garantir *endpoints* com poucas cargas de trabalho evita o acúmulo de operações, tornando menos susceptível à DDoS. O uso de cache também auxilia no processo.

A qualidade do acesso à internet, distância do servidor e componentes de hardware do dispositivo também podem causar variações durante o teste. A arquitetura está hospedada nos servidores do instituto politécnico, podendo sofrer alterações em picos de uso de outros serviços, que não estão conectados ao NanoStim.

5.3 Analise de ameaças

Nesta seção é abordado as ameaças dos ativos que compõem o projeto. Para a realização da análise, foi utilizado a modelagem de ameaças STRIDE. Esta análise aborda aspectos como ameaça de falsificação, alteração de dados, repúdio, divulgação indevida de dados, negação de serviço e elevação de privilégios.

A primeira característica abordada é a falsificação, referida no modelo STRIDE como *Spoofing*. As ameaças com foco em falsificação são informadas na Tabela 5.4.

Ameaça	Descrição	Motivação	Formas de Mitigação
Roubo do JWT	O JWT é utilizado para como forma de autorização e autenticação. Ao roubar o JWT, o atacante possui acesso não autorizado à funcionalidades do sistema.	Ao roubar o JWT de um utilizador, o atacante pode realizar operações dentro do sistema tendo acesso á todas as funcionalidades permitidas ao utilizador.	Utilizar tempo de expiração para impedir o reuso do JWT, proteger JWT durante o transporte de pacotes.
Roubar a Identidade Digital	Para que o utilizador receba o token com suas devidas autorizações, é necessário realizar o login dentro do sistema utilizando credenciais, como email e senha.	Ao roubar as credenciais de acesso, o atacante obtém acesso não autorizado à funcionalidades do sistema, realizando as operações em nome do utilizador roubado. Para esse caso, o tempo de validade do JWT não trará proteção adicional, uma vez que o atacante é capaz de obter novos tokens à qualquer momento.	Proteção de senhas e chaves durante o transporte de pacotes, salvar senhas criptografadas, autenticação de dois fatores, estratégias de criptografia em repouso para banco de dados, proteger servidores contra DUMPs de memória RAM.

Continua na próxima página

Tabela 5.4 – Continuação da página anterior

Ameaça	Descrição	Motivação	Formas de Mitigação
Servidor Falso	Todas as requisições são enviadas a um servidor. O atacante pode oferecer um servidor semelhante.	O atacante passa a receber todas as requisições, obtendo dados sensíveis, além de ser capaz de retornar informações falsas.	Verificando a identidade do servidor através do uso de certificados virtuais, dificultando acesso aos servidores físicos.
Aplicativo Falso	O aplicativo móvel faz a conexão entre o dispositivo vestível e o sistema na nuvem. O atacante pode oferecer um aplicativo semelhante.	O atacante tem controle sobre os dados enviados para o vestível e para a nuvem, sendo capaz de alterar os dados antes do envio pela aplicação, redirecionar o tráfego das informações, ter acesso á dados sensíveis, roubo de credenciais e chaves.	Verificando a autenticidade do aplicativo e do desenvolvedor, não utilizando versões alternativas.

Continua na próxima página

Tabela 5.4 – Continuação da página anterior

Ameaça	Descrição	Motivação	Formas de Mitigação
vestível Falso	A solução proposta utiliza um dispositivo vestível para realizar tratamentos. O atacante pode oferecer uma versão alternativa do Wearable.	O atacante tem controle sobre as configurações do vestível e requisições, podendo controlá-lo remotamente, negar o tratamentos a um paciente, promover tratamentos prejudiciais ao utilizador ou realizar espionagem.	A arquitetura em nuvem deve reconhecer os dispositivos confiáveis. O vestível deve utilizar criptografia RSA, ou equivalente, de modo que o aplicativo mobile consiga verificar a autenticidade do vestível através de uma autenticação própria. Para desenvolvimentos mais simples, pode-se utilizar *whitelists* para filtrar endereços Media Access Control Address (MAC) de dispositivos vestíveis , porém não é uma estratégia tão robusta quanto o uso de chaves RSA.
Continua na próxima página			

Tabela 5.4 – Continuação da página anterior

Ameaça	Descrição	Motivação	Formas de Mitigação
Websítio Falso	O websítio contém a interface com a integração de funções permitidas apenas por funcionários do hospital. O atacante pode oferecer uma versão alternativa, ou muito similar ao websítio.	O atacante tem controle sobre as ações realizadas na interface, sendo capaz de alterar os dados antes de realizar uma requisição, forjar ações do funcionário, alterar o tráfego dos dados e ter acesso a informações sensíveis.	Verificar o domínio do websítio, certificado de segurança, identidade do servidor.
Micro serviço Falso	O atacante pode inserir um falso serviço na plataforma.	Coletar informações, obter interações com outros serviços e servidores, envio de dados falsos ou mal intencionados, parada do sistema, uso indevido de recursos	Garantir configurações corretas nos serviços e servidores, controle de acesso á ficheiros e arquivos, configuração de permissões, monitoramento de serviços. O uso de certificados em cada microsserviço pode ser utilizado para identificação de serviços oficiais dentro da arquitetura.

Continua na próxima página

Tabela 5.4 – Continuação da página anterior

Ameaça	Descrição	Motivação	Formas de Mitigação

Tabela 5.4: Ameaças que envolvem falsificação.

Em seguida é apresentado a analise das ameaças que tem como foco a alteração de informações, referido pelo modelo STRIDE como *Tampering*. Na tabela 5.5 é mostrado as possíveis motivações que envolvem a alteração não permitida dos dados.

Ameaça	Descrição	Motivação	Formas de Mitigação
Alterar Banco de Dados	O atacante altera informações do banco de dados utilizado por algum componente do sistema.	O atacante ganha acesso ao banco de dados, com o objetivo de realizar alterações para afetar alguma aplicação ou serviço. O ataque pode acarretar em mudança de permissões de um utilizador, exclusão de dados e informações falsas.	registrar atividades do banco de dados (log), configurar utilizadores e permissões para operar no banco de dados, realizar backups, remover acesso direto via internet.
Continua na próxima página			

Tabela 5.5 – Continuação da página anterior

Ameaça	Descrição	Motivação	Formas de Mitigação
Alterar Conteúdo	O atacante utiliza informações mal formadas em entradas do sistema.	O atacante edita um conteúdo para realizar um *script* no servidor ou no cliente. Também pode utilizar o ataque para alterar o funcionamento padrão do sistema.	Validação de dados, utilizar JWT e hash para detectar alterações de informação.
Alterar dados em trânsito	O atacante intercepta a comunicação e realiza alterações nos dados que foram enviados pelo utilizador.	O atacante pode modificar o serviço requisitado, os parâmetros enviados para uma rotina do servidor, o destino da mensagem, além de conseguir forçar a perda ou rejeição de pacotes.	A utilização de hashs e JWT para validação dos dados pode detectar alterações durante a transmissão das informações, uso de SSL/Transport Layer Security (TLS) para criptografar a comunicação.

Tabela 5.5: Ameaças que envolvem alteração de informações.

A modelagem STRIDE também aborda a repudiação. Esse caso ocorre quando um utilizador afirma que não realizou determinada ação no sistema e não é possível utilizar provas para chegar a uma conclusão. As ameaças referentes à repudiação são mostradas na Tabela 5.6.

Ameaça	Descrição	Motivação	Formas de Mitigação
Criar conta falsa	O atacante pode utilizar dados de pessoas reais para criar contas, ou ainda criar uma conta com os dados de uma pessoa inexistente.	Utilizar os serviços do sistema se passando por outro utilizador que ainda não foi cadastrado.	Permitir que apenas funcionários com as devidas permissões registrem um utilizador, verificando a autenticidade das informações por meio de documentos, ou ainda que contém informações de cadastro.
Negar interação	Um utilizador realiza interações com o sistema. Em um determinado caso, é possível que o utilizador informe que não é responsável por determinadas atitudes.	Tomar atitudes errôneas ou prejudiciais através das funcionalidades do sistema e se ausentar de culpa.	Armazenar logs para as operações ocorridas dentro do sistema.

Tabela 5.6: Ameaças que envolvem repudiação de ações.

Há também ameaças relacionadas na divulgação indevida de dados. Esse risco afeta a privacidade dos utilizadores, tanto pacientes quanto profissional de saúde. As ameaças são descritas na Tabela 5.7.

Ameaça	Descrição	Motivação	Formas de Mitigação
Acessar dados secretos ou privados	O atacante pode extrair informações armazenadas pelos serviços através da exploração de fraquezas. Dessa forma, obterá acesso a dados secretos ou privados.	O atacante poderá utilizar as informações para obter vantagens sobre os utilizadores do sistema.	Bloquear o acesso direto via internet aos serviços de bancos de dados, habilitar autenticação nos bancos, fazer uso de pseudônimos para separar dados de identificação de dados de sessão
espionagem	O atacante pode realizar a escuta dos meios de transmissão dos dados (bluetooth, wifi, etc) para obter informações.	Obter acesso aos dados privados dos utilizador.	Utilizar criptografia durante as comunicações em meios inseguros.

Continua na próxima página

Tabela 5.7 – Continuação da página anterior

Ameaça	Descrição	Motivação	Formas de Mitigação
controle sobre dispositivos	O atacante pode ter acesso físico ou remoto sobre o dispositivo de um utilizador.	Obter acesso aos dados privados dos utilizador.	O acesso remoto aos dispositivos deve ser tratado através de práticas seguras de desenvolvimento. Meios de comunicação devem ser criptografados. Cabe ao utilizador garantir a segurança de seus dispositivos, através de senhas, leitor de digitais, e estratégias semelhantes.

Tabela 5.7: Ameaças que envolvem divulgação de informação.

Também foi documentado as ameaças que envolvem a negação de serviços. Esse tipo de ameaça ocorre quando um atacante é capaz de parar temporariamente ou definitivamente algum dos serviços contidos na solução. As ameaças são apresentadas na Tabela 5.8.

Ameaça	Descrição	Motivação	Formas de Mitigação
Danificar ou Alterar vestível	O atacante pode danificar o vestível ou modificar seu funcionamento.	Garantir que o utilizador que tem a posse do dispositivo não realize o tratamento.	Verificar o equipamento com frequência, possuir rotinas automatizadas para certificar seu funcionamento.
Interferir no sinal bluetooth	O vestível recebe comandos através do bluetooth. Caso as requisições sejam invalidadas ou barradas durante a transmissão, o vestível não receberá o comando para iniciar o tratamento ou seus parâmetros.	Garantir que os utilizadores próximos não consigam utilizar vestível, inutilizar dados enviados via bluetooth.	Garantir distância de aparelhos que possam causar interferência.
inutilizar mobile temporariamente ou definitivamente.	O vestível é controlado através de um aplicativo mobile. Ao inutilizar o mobile que contém a aplicação, não será possível se comunicar com o vestível.	O atacante atrasa ou impossibilita o tratamento do paciente.	Garantir dispositivos reserva quando possível, com carga e acesso aos os recursos necessários, como internet e bluetooth.

Continua na próxima página

Tabela 5.8 – Continuação da página anterior

Ameaça	Descrição	Motivação	Formas de Mitigação
Interferir na internet	A aplicação necessita de internet para obter as informações na nuvem, assim como realizar autenticação e consultar sessões. Ao impedir a busca das informações, o tratamento não poderá ser iniciado.	O atacante impossibilita o utilizador de realizar as operações dentro da aplicação, e de iniciar o tratamento.	Garantir internet segura para realizar as operações iniciais. A aplicação pode armazenar dados localmente, para utilizar posteriormente.
apagar ou parar banco de dados	O banco de dados contém informações como credenciais de autenticação, sessões de tratamentos agendados, parâmetros de configuração do vestível para cada sessão. Ao apagar dados do banco, o utilizador pode perder suas credenciais de autenticação, informações pessoais e dados médicas.	O atacante pode impossibilitar um ou mais pacientes de utilizarem a aplicação e vestível, realizar autenticação no sistema e realizarem os tratamentos agendados.	Criação de backups regulares, sistemas de log, controle de acesso ao banco de dados configurado com as devidas permissões, conjunto de réplicas ativas do banco, barrar acesso direto aos bancos de dados.

Continua na próxima página

Tabela 5.8 – Continuação da página anterior

Ameaça	Descrição	Motivação	Formas de Mitigação
tornar serviço indisponível ou extremamente lento	O atacante pode realizar grande demanda de requisições aos servidores ou dispositivos, os sobrecarregando. Desse modo, um utilizador pode não ser atendido.	O atacante pode causar lentidão ou parar um ou mais serviços, impossibilitando o funcionamento da plataforma ou aplicação.	Utilizar estratégias de balanceamento de carga, cache, e promover capacidade para escalabilidade durante o desenvolvimento da arquitetura.

Tabela 5.8: Ameaças referentes à negação de serviços.

Ao final, o modelo de ameaças faz análise aos possíveis problemas relacionados à elevação de privilégio, mostrado na Tabela 5.9. O problema acontece quando um atacante consegue obter permissões dentro do sistema de modo indevido.

Ameaça	Descrição	Motivação	Formas de Mitigação
Burlar controle de acesso	O atacante utiliza formas para burlar o controle de acesso, através de entradas mal formadas, redirecionamentos.	Permite a utilização não autorizada de recursos	Utilizar estratégias e serviços robustos durante o desenvolvimento, ter políticas bem definidas

Continua na próxima página

Tabela 5.9 – Continuação da página anterior

Ameaça	Descrição	Motivação	Formas de Mitigação
acesso a ações administrativas	O atacante obtém acesso a ações administrativas, sendo capaz de alterar as permissões de utilizadores.	Permite a atribuição incorreta de permissões a um utilizador, dando maior capacidade de interação dentro do sistema.	As políticas de acesso devem ser bem consolidadas, para minimizar erros humanos e barrar a atribuição indevida de privilégios.
controle sobre dispositivos	O atacante obtém o controle de um dispositivo com acesso aos serviços de gerenciamento.	Permite a utilização e configuração não autorizada de recursos do sistema.	Dificultar o acesso físico ou remoto aos serviços, armazenar corretamente os segredos que permitem acesso
Utilizar token não expirado	Um JWT não é facilmente revogado. Caso o atacante obtenha acesso a um JWT que não excedeu a validade, poderá ser utilizado para interagir com o sistema.	Obter acesso a recursos protegidos, fazer mal uso dos serviços, explorar fraquezas.	Manter curta duração para a troca de JWT entre serviços, proteger tokens de atualização, utilizar serviços robustos que permitem a revocação de tokens.

Tabela 5.9: Ameaças envolvendo elevação de privilégios.

Todos os ativos do sistema podem ser alvos de ataques a qualquer momento, tornando grande a superfície de ataque. A análise dos pontos levantados é necessária para identificar e reduzir a chance de ocorrência das ameaças, para tornar a plataforma mais segura. A cada instante do ciclo de vida do projeto, é importante perceber as vulnerabilidades que ainda não foram detectadas e tomar medidas de mitigação.

De forma resumida, é possível verificar que as vulnerabilidades de um ativo abrem meios para a exploração de outros ativos de confiança. Os ativos que se encontram em meios inseguros, como a aplicação, a página web e o dispositivo vestível, são os meios de interação que permitem o início de um ataque, por serem facilmente obtidos e explorados. Estabelecer formas robustas de reconhecimento de serviços, dispositivos, utilizadores e informações não modificadas dificulta a exploração mal intencionada das funcionalidades da plataforma, afetando positivamente a privacidade.

Capítulo 6

Conclusões

O capítulo atual aborda as conclusões obtidas durante as etapas de análise da plataforma e desenvolvimento da plataforma. Inicialmente são apresentados as conclusões, e em seguida é descrito os trabalhos futuros.

6.1 Conclusão

A disponibilidade de serviços é alcançada através da clusterização dos serviços. Assim há novas dificuldades para o sistema, como o gerenciamento de dados guardados em cache, roteamento de informações, acesso às configuração do cluster, comunicação assegurada, monitoramento e automatização. Logo, a disponibilidade de um microsserviço implica o consumo de novos recursos que também podem ficar indisponíveis em algum momento.

Configurações de serviços conteinerizados com caraterísticas statefull não auxiliam durante o processo de desenvolvimento e manutenção. Para serviços statefull, como bancos de dados, é viável utilizar máquinas virtuais dedicadas. Contêineres são normalmente focados em serviços efêmeros, entrando em conflito com as características statefull.

Uma arquitetura orientada a microsserviços tende a torna-se complexa. Com o dinamismo da arquitetura, é difícil de acompanhar as configurações, alterações e comportamentos suspeitos. É necessário investir tempo na elaboração para facilitar a manutenção durante todo o ciclo de vida da arquitetura.

Um microsserviço pode ser testado individualmente. Isso garante que o serviço funcionará, porém não há garantia que a integração com outros serviços também ocorrerá. Encontrar erros de integração é uma tarefa cansativa, pois pode ocorrer a qualquer momento por detalhes não percebidos, e afetar múltiplos serviços por propagação de erros.

Todas as decisões tomadas durante a analise de ameaça interagem com as características de funcionalidade, usabilidade e segurança. O sistema deve ser seguro, porém ainda necessita ser funcional e usável. É necessário ponderar, identificando até qual ponto é viável proteger determinadas funcionalidades.

Ao utilizar a pseudonimização, os dados podem ser distribuídos para outros serviços com melhorias na privacidade dos utilizadores. Porém, ao permitir que os utilizadores do sistema sejam capazes de obter os pseudônimos de outros utilizadores e identificá-los, surge uma maior complexidade de implementação do sistema que gerencia as permissões de re-identificação.

Além de múltiplos serviços, pode ser utilizado múltiplos servidores em uma arquitetura de microsserviços para hospedar parte das funcionalidades. A quantidade de detalhes de implementação aumentam como endereços os de IPs, estratégias de comunicação, credenciais de acessos e segredos, configurações, podendo aumentar a superfície de ataque devido a erro humano ou desconhecimento de falhas.

Microsserviços podem ter permissões de chamadas exclusivas a outros serviços, por estarem no *localhost*, ou por configurações. Manipular um serviço pode dar poder a um atacante de utilizar os privilégios cedidos a um microsserviço.

Manter configurações de chaves dentro de uma plataforma de microsserviço é uma tarefa que exige muita atenção. Utilizar serviços para gerenciamento de certificados facilita nos processos de manutenção da arquitetura.

Como os microsserviços trabalham em conjunto para atingirem seu propósito, o tempo de resposta é afetado. Os serviços realizam chamada a outros, e dessa forma o tempo de processamento e consumo de rede aumentam.

O uso da criptografia RSA foi muito utilizado para garantir o reconhecimento entre dois serviços. Os serviços se reconhecem através de suas chaves públicas, e dessa forma

podem se confiar, mesmo que seus dados usem meios de transmissão inseguro. O RSA é utilizado por muitos sistemas, mas sofre ameaças com o avanço da computação quântica.

Os dispositivos de entrada dos dados são as principais fontes de informação. Mesmo que uma plataforma seja capaz proteger contra a alterar ou exclusão indevida, os dados ainda necessitam ser emitidos por dispositivos confiáveis e íntegros.

Os objetivos referentes a escalabilidade, disponibilidade e manutenção dos serviços da arquitetura foram atingidas principalmente devido ao uso do kubernetes. A inclusão do kubernetes facilitou a automatização e monitoria dos serviços stateless da plataforma, contribuindo também a disponibilidade e segurança das configurações da arquitetura. Porém, o kubernetes geralmente não possui integração simples para serviços statefull.

Os objetivos voltados a disponibilidade de dados foram alcançados através das configurações de replicas dos bancos de dados. A estratégia garante instâncias iguais caso ocorra falha, mas não provê backup por padrão.

As estratégias de segurança com foco na autenticação de utilizadores em múltiplos serviços foram mitigadas através da inclusão de um serviço de autenticação centralizada. A utilização do Keycloak facilita a configuração de funções de autenticação, além de prover ao utilizador as funcionalidades de duplo fator de autenticação, controle de permissões de aplicações, confirmação de e-mail e recuperação de senha.

A autenticação entre APIs por meio de chaves RSA atingem o objetivo de reconhecimento de serviços. Cada serviço é capaz de reconhecer a autoria e integridade das informações de outros serviços da plataforma.

As possíveis ameaças obtidas a partir da modelação STRIDE guiaram o desenvolvimento da plataforma, a fim de controlar a possibilidade de ocorrência de ameaças, balanceando também com viabilidade da resolução. Como dito anteriormente, foi necessário obter o equilíbrio entre funcionalidade, segurança e usabilidade da arquitetura.

6.2 Trabalhos Futuros

Como trabalho futuro pretende-se iniciar os testes reais com toda a solução em funcionamento, ou seja com o telemóvel que controla o vestível e efetuar sessões de testes que façam uso de toda as funcionalidades da solução proposta.

Para suprir grandes quantidades de requisições, é vantajoso que a plataforma seja capaz de escalar automaticamente a quantidade de serviços. Como trabalho futuro, deve-se adicionar a elasticidade à plataforma, poupando recursos e maximizando o desempenho quando necessário. Deverão também ser efetuados testes de carga.

Para aumento da segurança de chaves da criptografia RSA e dos dados do sistema, é desejado habilitar a criptografia em repouso. Dessa forma, os dados permanecem criptografados no disco, dificultando o acesso indevido em caso de roubo de hardware.

Em relação aos dados da plataforma, é desejado realizar a melhoria na comunicação entre os bancos de dados, e melhoria no processo de autenticação entre os serviços, além de habilitar a votação automática do postgres para eleger novas instâncias primárias automaticamente.

A detecção e mitigação de novas ameaças é um processo que deve estar presente em todo o ciclo de vida da plataforma. Para auxiliar na detecção de falhas, é desejado incluir um serviço de blockchain para centralizar os logs da plataforma de modo a facilitar a busca e leitura, protegendo contra mudanças e acessos indevidos. Também deve-se realizar a adição de um serviço de email integrado ao keycloak para habilitar as funções de confirmação de email e recuperação de senha. O serviço de email também deverá ser utilizado pelo clusterControl para que os mantenedores da arquitetura recebam os alertas emitidos quando é detectado um problema em um cluster de dados, como queda de instâncias ou lentidão.

Bibliografia

[1] NANOSTIM, *Projeto NanoStim*, 2020. URL: https://nanostim.pt/projeto-nanostim/.

[2] NIST, "The NIST Definition of Cloud Computing", rel. téc. BBVA Research, 2016.

[3] P. J. Sun, "Privacy Protection and Data Security in Cloud Computing: A Survey, Challenges, and Solutions", *IEEE Access*, vol. 7, 2019, ISSN: 21693536. DOI: 10.1109/ACCESS.2019.2946185.

[4] J. Zhang, H. Liu e L. Ni, "A Secure Energy-Saving Communication and Encrypted Storage Model Based on RC4 for EHR", *IEEE Access*, vol. 8, 2020, ISSN: 21693536. DOI: 10.1109/ACCESS.2020.2975208. URL: https://ieeexplore.ieee.org/document/9004574.

[5] B. Fabian, T. Ermakova e P. Junghanns, "Collaborative and secure sharing of healthcare data in multi-clouds", *Information Systems*, vol. 48, 2015, ISSN: 03064379. DOI: 10.1016/j.is.2014.05.004.

[6] A. Gandomi e M. Haider, "Beyond the hype: Big data concepts, methods, and analytics", *International Journal of Information Management*, vol. 35, n.º 2, 2015, ISSN: 02684012. DOI: 10.1016/j.ijinfomgt.2014.10.007.

[7] S. Selvaraj e S. Sundaravaradhan, "Challenges and opportunities in IoT healthcare systems: a systematic review", *SN Applied Sciences*, vol. 2, n.º 1, 2020, ISSN: 2523-3963. DOI: 10.1007/s42452-019-1925-y. URL: https://link.springer.com/

article/10.1007/s42452-019-1925-y?utm_source=getftr&utm_medium=getftr&utm_campaign=getftr_pilot.

[8] Y. Ai, M. Peng e K. Zhang, "Edge computing technologies for Internet of Things: a primer", *Digital Communications and Networks*, vol. 4, n.º 2, 2018, ISSN: 23528648. DOI: 10.1016/j.dcan.2017.07.001.

[9] D. Namiot e M. Sneps-sneppe, "On micro-services architecture", *International Journal of Open Information Technologies*, vol. 2, n.º 9, 2014, ISSN: 2307-8162.

[10] A. Hoang, "Analysis of Micro Services and Serverless Architecture for Mobile Application Enablement", tese de doutoramento, 2017.

[11] M. A. Scholl, K. M. Stine, J. Hash, P. Bowen, L. A. Johnson, C. D. Smith e D. I. Steinberg, "NIST SP 800-66 Revision 1 An Introductory Resource Guide for Implementing the Health Insurance Portability and Accountability Act (HIPAA) Security Rule", *An Introductory Resource Guide for Implementing the Health Insurance Portability and Accountability Act (HIPAA) Security Rule*, vol. 800, n.º October, 2008.

[12] A. Mishra, " Amazon DynamoDB ", em *Machine Learning in the AWS Cloud*, 2019. DOI: 10.1002/9781119556749.ch11.

[13] T. Xin e B. Xiaofang, "Online banking security analysis based on STRIDE threat model", *International Journal of Security and its Applications*, vol. 8, n.º 2, 2014, ISSN: 17389976. DOI: 10.14257/ijsia.2014.8.2.28.

[14] L. O. Nweke e S. D. Wolthusen, "A review of asset-centric threat modelling approaches", *International Journal of Advanced Computer Science and Applications*, n.º 2, 2020, ISSN: 21565570. DOI: 10.14569/ijacsa.2020.0110201.

[15] M. Tatam, B. Shanmugam, S. Azam e K. Kannoorpatti, *A review of threat modelling approaches for APT-style attacks*, 2021. DOI: 10.1016/j.heliyon.2021.e05969. URL: https://reader.elsevier.com/reader/sd/pii/S2405844021000748.

[16] P. Yang, N. Xiong e J. Ren, *Data Security and Privacy Protection for Cloud Storage: A Survey*, 2020. DOI: 10.1109/ACCESS.2020.3009876.

[17] F. Maqsood, M. Ahmed, M. Mumtaz e M. Ali, "Cryptography: A Comparative Analysis for Modern Techniques", *International Journal of Advanced Computer Science and Applications*, vol. 8, n.º 6, 2017, ISSN: 2158107X. DOI: 10.14569/ijacsa.2017.080659.

[18] Sutriman e B. Sugiantoro, "Analysis of password and salt combination scheme to improve hash algorithm security", *International Journal of Advanced Computer Science and Applications*, vol. 10, n.º 11, 2019, ISSN: 21565570. DOI: 10.14569/IJACSA.2019.0101158.

[19] J. Zhang, B. Chen, Y. Zhao, X. Cheng e F. Hu, "Data Security and Privacy-Preserving in Edge Computing Paradigm: Survey and Open Issues", *IEEE Access*, vol. 6, 2018, ISSN: 21693536. DOI: 10.1109/ACCESS.2018.2820162.

[20] TreeSolution, *Plan a successful security awareness campaign*, jun. de 2019. URL: https://www.treesolution.com/en/security-awareness-behaviour-culture.

[21] C. Lauradoux, K. Limniotis, M. Hansen, M. Jensen e P. Eftasthopoulos, "DATA PSEUDONYMISATION: ADVANCED TECHNIQUES & USE CASES", 2021. DOI: 10.2824/860099. URL: https://www.enisa.europa.eu/publications/data-pseudonymisation-advanced-techniques-and-use-cases.

[22] T. Melamed, "An active man-in-The-middle attack on bluetooth smart devices", em *International Journal of Safety and Security Engineering*, vol. 8, 2018. DOI: 10.2495/SAFE-V8-N2-200-211.

[23] Pallets organization, *Welcome to Flask – Flask Documentation (2.0.x)*, 2010. URL: https://flask.palletsprojects.com/en/2.0.x/#.

[24] Auth0, *Get Started with JSON Web Tokens - Auth0*, 2015. URL: https://auth0.com/learn/json-web-tokens/.

[25] ——, *JSON Web Tokens*, 2021.

[26] Imperva, "Man in the Cloud (MITC) attacks", 2015. URL: https://www.imperva.com/docs/hii_man_in_the_cloud_attacks.pdf.

[27] Docker Inc., “Docker Documentation”, *Docker Docs*, 2018. URL: https://docs.docker.com/get-started/overview/.

[28] ——, *Swarm mode overview | Docker Documentation*, 2020. URL: https://docs.docker.com/engine/swarm/.

[29] MongoDB Inc., *O banco de dados para aplicativos modernos*, 2021. URL: https://www.mongodb.com/pt-br.

[30] ——, *The MongoDB 5.0 Manual*, 2021. URL: https://docs.mongodb.com/manual/introduction/.

[31] Postgres, *About Postgres*, 2015.

[32] Red Hat inc., *Keycloak - About*. URL: https://www.keycloak.org/about.html.

[33] ——, *Open Source Identity and Access Management For Modern Applications and Services*. URL: https://www.keycloak.org/.

[34] Several Nines, *Cluster Control Free Community Edition*, 2014. URL: https://severalnines.com/product/clustercontrol/clustercontrol-community-edition.

[35] The Docker Community, *HAProxy Oficial Image*. URL: https://hub.docker.com/_/haproxy.

[36] Imperva, *HAProxy configurations vs on-edge load balancing – use case comparisons*. URL: https://www.imperva.com/learn/availability/haproxy/.

[37] The Kubernetes Authors, “What is Kubernetes?”, jul. de 2021. URL: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.

[38] ——, *minikube start*, set. de 2021. URL: https://minikube.sigs.k8s.io/docs/start/.

[39] ——, *Using Minikube to Create a Cluster*, fev. de 2021. URL: https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/.

[40] ——, *Ingress*, jun. de 2021. URL: https://kubernetes.io/docs/concepts/services-networking/ingress/.

[41] F5 Networks Inc, *What Is an Ingress Controller?*, 2021. URL: https://www.nginx.com/resources/glossary/kubernetes-ingress-controller/.

[42] The Kubernetes Authors, *Pods*, set. de 2021. URL: https://kubernetes.io/docs/concepts/workloads/pods/.

[43] ——, *Service*, set. de 2021. URL: https://kubernetes.io/docs/concepts/services-networking/service/.

[44] ——, *Deployments*, set. de 2021. URL: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/.

[45] ——, "ConfigMaps", jun. de 2021. URL: https://kubernetes.io/docs/concepts/configuration/configmap/.

[46] ——, "Secrets", ago. de 2021. URL: https://kubernetes.io/docs/concepts/configuration/secret/.

[47] MongoDB Inc., *Update Replica Set to Keyfile Authentication*. URL: https://docs.mongodb.com/manual/tutorial/enforce-keyfile-access-control-in-existing-replica-set/.

Apêndice A

Anexos

Neste capítulo contém as instruções utilizadas para configuração da arquitetura. Os *scripts* contém endereços de IP, portas, usuários e senhas fictícias.

A.1 MongoDB

```
# definicao dos IPs

export mongo1=192.93.94.5
export mongo2=192.93.94.6
export mongo3=192.93.94.7

# criando arquivo keyfile

mkdir -p ~/mongo/core/mongo-files ~/mongo/ssh_conf
openssl rand -base64 741 > mongodb-keyfile
mv mongodb-keyfile ~/mongo/core/mongo-files
```

```
# configurando keyfile

docker run  --rm \
    --name mongo_node \
    -v ~/mongo/core/mongo-files/data:/data/db \
    -v ~/mongo/core/mongo-files:/opt/keyfile \
    -v ~/mongo/ssh_conf:/root/.ssh \
    --hostname="mongo1.com" \
    -p 27017:27017 \
    -p 17394:17394 \
    -d mongo

docker exec -it mongo_node bash
chmod 600 /opt/keyfile/mongodb-keyfile
chown 999 /opt/keyfile/mongodb-keyfile
exit

# criando credenciais de acesso

docker exec -it mongo_node mongo

use admin

db.createUser( {
      user: "mongo_adm",
      pwd: "senha",
      roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
    });
```

```
db.createUser( {
      user: "mongo_root",
      pwd: "senha",
      roles: [ { role: "root", db: "admin" } ]
    });

db.createUser(
  {
    user: "mongo_user",
    pwd: 'senha',
    roles: [
      { role: "readWrite", db: "database" },
    ]
  }
);

exit

docker stop mongo_node ;

# criando instancia final que utiliza keyfile

docker run -d \
-p 27017:27017 \
-p 17394:22 \
--name mongo_node \
-v ~/mongo/core/mongo-files/data:/data/db \
-v ~/mongo/core/mongo-files:/opt/keyfile:ro \
-v ~/mongo/ssh_conf:/root/.ssh \
```

```
--restart always \
--hostname="mongo1.com" \
--add-host mongo1.com:${mongo1} \
--add-host mongo2.com:${mongo2} \
--add-host mongo3.com:${mongo3} \
 mongo \
--keyFile /opt/keyfile/mongodb-keyfile \
--replSet "rs0"

docker exec -it mongo_node mongo

use admin
db.auth("mongo_root","senha")
rs.initiate()
exit

# Apos enviar keyfile para outros servidores
# iniciando conteiner para configuracao de cada replica

docker run --rm \
    --name mongo_node \
    -v ~/mongo/core/mongo-files/data:/data/db \
    -v ~/mongo/core/mongo-files:/opt/keyfile \
    -v ~/mongo/ssh_conf:/root/.ssh \
    --hostname="mongo2.com" \
    -p 27017:27017 \
    -d mongo

# alterando permissoes do keyfile
```

```
docker exec -it mongo_node bash
chmod 600 /opt/keyfile/mongodb-keyfile
chown 999 /opt/keyfile/mongodb-keyfile
exit

docker stop mongo_node;

# iniciando container de replica, utilizando keytile

docker run -d \
--name mongo_node \
-v ~/mongo/core/mongo-files/data:/data/db \
-v ~/mongo/core/mongo-files:/opt/keyfile \
-v ~/mongo/ssh_conf:/root/.ssh \
--restart always \
--hostname="mongo2.com" \
--add-host mongo1.com:${mongo1} \
--add-host mongo2.com:${mongo2} \
--add-host mongo3.com:${mongo3} \
-p 17394:22 \
-p 27017:27017 mongo \
--keyFile /opt/keyfile/mongodb-keyfile \
--replSet "rs0" \
--bind_ip 0.0.0.0

# Ativar todos os servidores na instancia primaria

docker exec -it mongo_node mongo
```

```
use admin
db.auth("mongo_root","senha")

rs.add("mongo2.com:27017");
rs.add("mongo3.com:27017");

# integrando ao clustercontrol
# instalando configurando ssh

docker exec -i mongo_node apt-get update &&
docker exec -i mongo_node apt-get install -y openssh-server \
openssh-client  && \
docker exec -i mongo_node sed -i \
's|^PermitRootLogin.*|PermitRootLogin yes|g' /etc/ssh/sshd_config
&& \
docker exec -i mongo_node sed -i \
's|^#PermitRootLogin.*|PermitRootLogin yes|g' /etc/ssh/sshd_config
&& \
docker exec -i mongo_node ssh-keygen -t rsa -q -f \
"/root/.ssh/id_rsa" -N ""  && \
docker exec -i mongo_node service ssh start

# exemplo de troca de chaves entre os containeres

ssh nanostim-clustercontrol 'docker exec -i clustercontrol \
cat root/.ssh/id_rsa.pub' > authorized_keys && \
scp authorized_keys nanostim-mongo-0:~/mongo/ssh_conf && \
scp authorized_keys nanostim-mongo-1:~/mongo/ssh_conf && \
```

```
scp authorized_keys nanostim-mongo-2:~/mongo/ssh_conf && \

rm authorized_keys && \
ssh nanostim-mongo-0
'cat mongo/ssh_conf/id_rsa.pub' >> authorized_keys && \
ssh nanostim-mongo-1 \
'cat mongo/ssh_conf/id_rsa.pub' >> authorized_keys && \
ssh nanostim-mongo-2 \
'cat mongo/ssh_conf/id_rsa.pub' >> authorized_keys && scp \
authorized_keys \
nanostim-clustercontrol:~/cluster/ssh_conf/append && \
ssh nanostim-clustercontrol \
'cat cluster/ssh_conf/append >>
cluster/ssh_conf/authorized_keys &&
rm cluster/ssh_conf/append'

# habilitando servico ssh

ssh nanostim-mongo-0 \
'docker exec -i mongo_node service ssh start' && \
ssh nanostim-mongo-1 \
'docker exec -i mongo_node service ssh start' && \
ssh nanostim-mongo-2 \
'docker exec -i mongo_node service ssh start'
```

A.2 Postgres

```
# Variaveis de configuracao do script
```

```
# necessario para criacao e autenticacao de instancias do postgres

export database_name=postgres_database
export database_user=postgres_user
export database_pass=postgres_passw
export pg_password_replicator=replicador_senha

# necessario para definiccao de portas, ips, e nome dos
# containers postgres

export pg_container_name=container_pg
export pg_port_map=36445
export pg_ssh_map=17394
export pg_psqlchk=31643
export host_master_ip=192.93.95.20
export host_replica_1_ip=192.93.95.21
export host_replica_2_ip=192.93.95.22

# portas abertas no host, que fazem o mapeamento do haproxy

export haproxy_ro_port=9394
export haproxy_rw_port=9395
export ha_container_name=ha_postgres

# configuracoes de SSH, para transferencia de arquivos
# entre hosts

export host_master=nanostim-postegres-0
export host_replica_1=nanostim-postegres-1
```

```
export host_replica_2=nanostim-postegres-2

# variaveis do clusterControl
export host_cc=nanostim-postegres-0
export cc_container_name=clustercontrol
export cc_port_map=9395

# configurando cluster
# criando instancia de nodes postgres


docker run -d \
    --name $pg_container_name \
    -p $pg_port_map:5432 \
    -p $pg_ssh_map:22 \
    -p $pg_psqlchk:31643 \
    --restart always \
    -e POSTGRES_DB=$database_name \
    -e POSTGRES_USER=$database_user \
    -e POSTGRES_PASSWORD=$database_pass \
    -v ~/postgres/data:/var/lib/postgresql/data \
    -v ~/postgres/ssh_conf:/root/.ssh \
    postgres

sleep 3

# editando configuracoes do no

(cat << eof
```

```
# Replication
wal_level = replica
hot_standby = on
max_wal_senders = 10
max_replication_slots = 10
hot_standby_feedback = on
eof
) > postgres.conf

docker cp postgres.conf
$pg_container_name:/var/lib/postgresql/data/postgres.conf
rm postgres.conf

# permissao para acesso

docker exec -it $pg_container_name bash -c \
'echo "host replication replicator 172.17.0.1/16 trust" >> \
/var/lib/postgresql/data/pg_hba.conf' ;
sleep 1

# criacao de replica

docker exec -it $pg_container_name psql \
$database_name -U $database_user -c \
"CREATE USER replicator WITH REPLICATION ENCRYPTED PASSWORD \
'$pg_password_replicator';";

sleep 1
```

```
# criacao de slots para cada replica

docker exec -it $pg_container_name psql \
$database_name -U $database_user -c "SELECT * FROM \
pg_create_physical_replication_slot('replication_slot_slave_1')";
sleep 1

docker exec -it $pg_container_name psql $database_name \
-U $database_user -c "SELECT * FROM \
pg_create_physical_replication_slot('replication_slot_slave_2')";
sleep 1

# realizando backup de cada replica

docker exec -it $pg_container_name pg_basebackup -D /tmp/data_1 \
-S replication_slot_slave_1 -X stream -P -U replicator -Fp -R ;
sleep 1

docker exec -it $pg_container_name pg_basebackup -D /tmp/data_2 \
-S replication_slot_slave_2 -X stream -P -U replicator -Fp -R ;

# retirando dados da replica do docker

mkdir postgres/temporario ;
sleep 1
docker cp $pg_container_name:/tmp/data_1 postgres/temporario/data_1 ;
docker cp $pg_container_name:/tmp/data_2 postgres/temporario/data_2 ;

docker restart $pg_container_name ;
```

```
# Em um computador com acesso aos hosts via SSH
# copiando dados de replica

scp -r $host_master:~/postgres/temporario/data_1/ . ;
scp -r $host_master:~/postgres/temporario/data_2/ . ;

# corrigindo arquivos de configuracao de conexao do postgres

echo restore_command = \'cp /var/lib/postgresql/data/pg_wal/%f
\"%p\"\' >> data_1/postgresql.auto.conf ;
echo restore_command = \'cp /var/lib/postgresql/data/pg_wal/%f \
\"%p\"\' >> data_2/postgresql.auto.conf ;

cat data_1/postgresql.auto.conf | sed \
"s/port\=5432/port\=$pg_port_map/g" > data_1/temp_config ;
cat data_2/postgresql.auto.conf | sed \
"s/port\=5432/port\=$pg_port_map/g" > data_2/temp_config ;

cat data_1/temp_config | sed \
"s/user\=replicator/host\=$host_master_ip\ user\=replicator\ \
password\=$pg_password_replicator/g" > data_1/postgresql.auto.conf ;
cat data_2/temp_config | sed "s/user\=replicator/host\=$host_master_ip\ \
user\=replicator\ password\=$pg_password_replicator/g" > \
data_2/postgresql.auto.conf ;

rm data_*/temp_config ;

# repassando informacoes para cada replica
```

```
ssh $host_replica_1 "mkdir postgres" ;
ssh $host_replica_2 "mkdir postgres" ;

scp -r data_1/ $host_replica_1:~/postgres/data ;
scp -r data_2/ $host_replica_2:~/postgres/data ;

# criando cada instancia de replica

docker run -d \
    --name $pg_container_name \
    -p $pg_port_map:5432 \
    -p $pg_ssh_map:22 \
    -p $pg_psqlchk:31643 \
    --restart always \
    -e POSTGRES_DB=$database_name \
    -e POSTGRES_USER=$database_user \
    -e POSTGRES_PASSWORD=$database_pass \
    -v ~/postgres/data:/var/lib/postgresql/data \
    -v ~/postgres/ssh_conf:/root/.ssh \
    postgres

# verificando status final (no master)

docker exec -it $pg_container_name psql $database_name \
-U $database_user -c 'SELECT * FROM pg_replication_slots;'

# configurando xinetd para obter no servidor ou no replica
# (healthchek para integracao com o haproxy) Em cada instancia
```

```
( cat << eof
#!/bin/bash

VALUE=\$(psql $database_name -t -U $database_user -c \
"select pg_is_in_recovery()")

if [ \$VALUE == "t" ]
then
    /bin/echo -e "HTTP/1.1 206 OK\r\n"
    /bin/echo -e "Content-Type: Content-Type: text/plain\r\n"
    /bin/echo -e "\r\n"
    /bin/echo "Standby"
    /bin/echo -e "\r\n"
elif [ \$VALUE == "f" ]
then
    /bin/echo -e "HTTP/1.1 200 OK\r\n"
    /bin/echo -e "Content-Type: Content-Type: text/plain\r\n"
    /bin/echo -e "\r\n"  echo "" > run.sh ; nano run.sh ; \
        chmod +x run.sh ; ./run.sh ;
    /bin/echo "Primary"
    /bin/echo -e "\r\n"
else
    /bin/echo -e "HTTP/1.1 503 Service Unavailable\r\n"
    /bin/echo -e "Content-Type: Content-Type: text/plain\r\n"
    /bin/echo -e "\r\n"
    /bin/echo "DB Down"
    /bin/echo -e "\r\n"
fi
```

```
eof
) > pgsqlchk

(cat << eof
service pgsqlchk
{
        flags           = REUSE
        socket_type     = stream
        port            = 31643  xinetd -f /etc/xinetd.conf -d
        wait            = no
        user            = nobody
        server          = /opt/pgsqlchk
        log_on_failure  += USERID
        disable         = no
        only_from       = 0.0.0.0/0
        per_source      = UNLIMITED
}
eof
) > pgsqlchk_conf ;

# inserndo script no conteiner e ativacao

docker cp pgsqlchk $pg_container_name:/opt/pgsqlchk ;
docker exec -it $pg_container_name bash -c 'chmod 755 /opt/pgsqlchk' ;
docker exec -it $pg_container_name apt-get update ;
docker exec -it $pg_container_name apt-get install xinetd telnet -y ;
docker cp pgsqlchk_conf $pg_container_name:/etc/xinetd.d/pgsqlchk ;
docker exec -it $pg_container_name bash -c 'echo \
"pgsqlchk 31643/tcp # pgsqlchk" >> /etc/services'
```

```
docker exec -it $pg_container_name service xinetd restart

sleep 3

# testando funcionamento do healthcheck
docker exec -it $pg_container_name telnet localhost 31643

# configurando haproxy

rm -r haproxy ;
docker rm -f $ha_container_name ;

mkdir haproxy

(cat << eof
global
    maxconn 100

defaults
    log global
    mode tcp
    retries 2
    timeout client 30m
    timeout connect 4s
    timeout server 30m
    timeout check 5s

listen stats
    mode http
```

```
    bind *:17394
    stats enable
    stats uri /

listen ReadWrite
    bind *:7879
    option httpchk
    http-check expect status 200
    default-server inter 3s fall 3 rise 2
        on-marked-down shutdown-sessions
    server pg0 $host_master_ip:$pg_port_map
        maxconn 100 check port $pg_psqlchk
    server pg1 $host_replica_1_ip:$pg_port_map
        maxconn 100 check port $pg_psqlchk
    server pg2 $host_replica_2_ip:$pg_port_map
        maxconn 100 check port $pg_psqlchk

listen ReadOnly
    bind *:9395
    option httpchk
    http-check expect rstatus ^20[06]
    default-server inter 3s fall 3 rise 2
        on-marked-down shutdown-sessions
    server pg0 $host_master_ip:$pg_port_map
        maxconn 100 check port $pg_psqlchk
    server pg1 $host_replica_1_ip:$pg_port_map
        maxconn 100 check port $pg_psqlchk
    server pg2 $host_replica_2_ip:$pg_port_map
        maxconn 100 check port $pg_psqlchk
```

```
eof
) > haproxy/haproxy.cfg

# criando Dockerfile

(cat << end
FROM haproxy:2.3
COPY haproxy.cfg /usr/local/etc/haproxy/haproxy.cfg
end
) > haproxy/Dockerfile


docker build -t haproxy-postg haproxy

docker run -d \
        --name $ha_container_name \
        -p $haproxy_rw_port:7879 \
        -p $haproxy_ro_port:9395 \
        haproxy-postg ;

# testando configuracao do load balance
# OBS: supondo que o ip esta em ens3 e que o host hospeda o haproxy

export my_ip=$(ip -4 -br a s ens3 | awk -F" " '{print $3}'|cut -d'/' -f1)
docker run --rm -it --name temp-pg postgres psql $database_name -h \
$my_ip -p $haproxy_rw_port -U $database_user -c \
"select pg_is_in_recovery()"
docker run --rm -it --name temp-pg postgres psql $database_name -h \
$my_ip -p $haproxy_ro_port -U $database_user -c "select pg_is_in_\
```

```
recovery()"

# ClusterControl integracao
# instalando ssh em cada host

docker exec -i $pg_container_name apt-get update ;
docker exec -i $pg_container_name apt-get install -y \
openssh-server openssh-client ;
docker exec -i $pg_container_name sed -i \
's|^PermitRootLogin.*|PermitRootLogin yes|g' /etc/ssh/sshd_config ;
docker exec -i $pg_container_name sed -i \
's|^#PermitRootLogin.*|PermitRootLogin yes|g' /etc/ssh/sshd_config ;
docker exec -i $pg_container_name ssh-keygen \
-t rsa -q -f "/root/.ssh/id_rsa" -N "" ;
docker exec -i $pg_container_name service ssh start ;
sleep 3

# Configurando o cluster SSH
# obtendo as chaves publicas de todos os componentes de um cluster
echo "\n" > authorized_keys;
ssh $host_master 'cat postgres/ssh_conf/id_rsa.pub' >> authorized_keys ;
ssh $host_replica_1 'cat postgres/ssh_conf/id_rsa.pub' >> authorized_keys ;
ssh $host_replica_2 'cat postgres/ssh_conf/id_rsa.pub' >> authorized_keys ;

# incluindo no cluster control
scp authorized_keys $host_cc:~/cluster/ssh_conf/append && \
ssh $host_cc 'cat cluster/ssh_conf/append >> \
cluster/ssh_conf/authorized_keys && rm cluster/ssh_conf/append'
```

```
# adicionando chave do cluster control na lista de hosts autorizados
ssh $host_cc "docker exec -i \
$cc_container_name cat root/.ssh/id_rsa.pub" > authorized_keys

# distribuindo authorized file para os outros servicos
scp authorized_keys $host_master:~/postgres/ssh_conf ;
scp authorized_keys $host_replica_1:~/postgres/ssh_conf ;
scp authorized_keys $host_replica_2:~/postgres/ssh_conf ;

rm authorized_keys

# reiniciando servico_ssh de cada no
ssh $host_master \
"docker exec -i $pg_container_name service ssh start" ;
ssh $host_replica_1 \
"docker exec -i $pg_container_name service ssh start" ;
ssh $host_replica_2 \
"docker exec -i $pg_container_name service ssh start" ;

# Criando o servico ClusterControl

docker run -d --name $cc_container_name \
    -p $cc_port_map:443 \
    --restart always \
    -v ~/cluster/cmon.d:/etc/cmon.d \
    -v ~/cluster/datadir:/var/lib/mysql \
    -v ~/cluster/ssh_conf:/root/.ssh \
    -v ~/cluster/cmonlib:/var/lib/cmon \
    -v ~/cluster/backups:/root/backups \
```

```
    -v ~/cluster/prom_data:/var/lib/prometheus \
    -v ~/cluster/prom_conf:/etc/prometheus \
    severalnines/clustercontrol
```

A.3 Ingress

A.3.1 Configuração do Namespace

```
apiVersion: v1
kind: Namespace
metadata:
  name: nanostim-namespace
```

A.3.2 Configuração do Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: server1
  namespace: nanostim-namespace
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  rules:
  - host: server1.com
    http:
      paths:
      - path: /machine(/|$)(.*)
        pathType: Prefix
        backend:
```

```
      service:
        name: machine-balance
        port:
          number: 5000
    - path: /management(/|$)(.*)
      pathType: Prefix
      backend:
        service:
          name: management-balance
          port:
            number: 5000
    - path: /keycloak(/|$)(.*)
      pathType: Prefix
      backend:
        service:
          name: keycloak-balance
          port:
            number: 5000
```

A.4 API de gerenciamento

A.4.1 Dockerfile

```
FROM python:3.8
WORKDIR /usr/src/app
COPY . .
RUN pip install --no-cache-dir -r requirements.txt
CMD ["python3","index.py"]
```

A.4.2 Configuração do Secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: mongo-secret
  namespace: nanostim-namespace
type: Opaque
data:
  MONGO_USER: base46==
  MONGO_DB: base46==
  MONGO_PASS: base46==
  FLASK_KEY: base46==
immutable: true
```

A.4.3 Configuração do ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: management-config-map
  namespace: nanostim-namespace
data:
  database_url: management-balance
immutable: true
```

A.4.4 Configuração do Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: management-api
  namespace: nanostim-namespace
```

```
spec:
  selector:
    matchLabels:
      app: management
  replicas: 2
  template:
    metadata:
      labels:
        app: management
    spec:
      containers:
      - name: management-api
        ports:
        # porta aberta do container
        - containerPort: 7879
        image: management
        imagePullPolicy: Never
        env:
          - name: ENV
            value: "production"
          - name: FLASK_APP
            value: "nanostim.app:initialize_app"
          - name: SECRET_KEY
            valueFrom:
              secretKeyRef:
                name: mongo-secret
                key: FLASK_KEY
          - name: PORT
            value: "8088"
```

```
        - name: MONGO_HOST
          value: "192.93.94.5"
        - name: MONGO_PORT
          value: "27017"
        - name: MONGO_DB
          valueFrom:
            secretKeyRef:
              name: mongo-secret
              key: MONGO_DB
        - name: MONGO_USER
          valueFrom:
            secretKeyRef:
              name: mongo-secret
              key: MONGO_USER
        - name: MONGO_PASS
          valueFrom:
            secretKeyRef:
              name: mongo-secret
              key: MONGO_PASS
```

A.4.5 Configuração do Service

```
apiVersion: v1
kind: Service
metadata:
  name: management-balance
  namespace: nanostim-namespace
spec:
  ports:
  - port: 7879
```

```
    protocol: TCP
    targetPort: 7879
    # entre 3000 e 32767
    nodePort: 32542
  selector:
    app: management
  type: LoadBalancer
```

A.5 API de aprendizado de maquina

A.5.1 Dockerfile

```
FROM python:3.8
WORKDIR /usr/src/app
COPY . .
RUN pip install --no-cache-dir -r requirements.txt
CMD ["python3","index.py"]
```

A.5.2 Configuração do Secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: mongo-machine
  namespace: nanostim-namespace
type: Opaque
data:
  # base 64
  MONGO_USER: base46==
  MONGO_DB: base46==
  MONGO_PASS: base46==
```

```
  FLASK_KEY: base46==
immutable: true
```

A.5.3 Configuração do ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: machine-config-map
  namespace: nanostim-namespace
data:
  database_url: machine-balance
immutable: true
```

A.5.4 Configuração do Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: machine-api
  namespace: nanostim-namespace
spec:
  selector:
    matchLabels:
      app: machine
  replicas: 2
  template:
    metadata:
      labels:
        app: machine
    spec:
```

```
containers:
- name: machine-api
  ports:
  # porta aberta do container
  - containerPort: 7879
  image: machine
  imagePullPolicy: Never
  env:
    - name: ENV
      value: "production"
    - name: FLASK_APP
      value: "nanostim.app:initialize_app"
    - name: SECRET_KEY
      valueFrom:
        secretKeyRef:
          name: mongo-machine
          key: FLASK_KEY
    - name: MONGO_DB
      valueFrom:
        secretKeyRef:
          name: mongo-machine
          key: MONGO_DB
    - name: PORT
      value: "8088"
    - name: MONGO_HOST
      value: "192.93.94.5"
    - name: MONGO_PORT
      value: "27017"
    - name: MONGO_USER
```

```
        valueFrom:
          secretKeyRef:
            name: mongo-machine
            key: MONGO_USER
      - name: MONGO_PASS
        valueFrom:
          secretKeyRef:
            name: mongo-machine
            key: MONGO_PASS
```

A.5.5 Configuração do Service

```
apiVersion: v1
kind: Service
metadata:
  name: machine-balance
  namespace: nanostim-namespace
spec:
  ports:
  - port: 7879
    protocol: TCP
    targetPort: 7879
    nodePort: 32000
  selector:
    app: machine
  type: LoadBalancer
```

A.6 Keycloak

A.6.1 Configuração do Secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: keycloak-secret
  namespace: nanostim-namespace
type: Opaque
data:
  # base 64
  POSTGRES_USER: base46==
  POSTGRES_PASS: base46==
  POSTGRES_DB: base46==
  POSTGRES_PORT: base46==
  #POSTGRES_PORT: base46==
immutable: true
```

A.6.2 Configuração do ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: keycloak-config-map
  namespace: nanostim-namespace
data:
  database_url: keycloak-balance
immutable: true
```

A.6.3 Configuração do Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: keycloak
  namespace: nanostim-namespace
  labels:
    app: keycloak
spec:
  replicas: 2
  selector:
    matchLabels:
      app: keycloak
  template:
    metadata:
      labels:
        app: keycloak
    spec:
      containers:
        - name: keycloak
          image: quay.io/keycloak/keycloak:15.0.2
          env:
            - name: PROXY_ADDRESS_FORWARDING
              value: "true"
            - name: DB_ADDR
              value: "haproxy.postgres.com"
            - name: DB_DATABASE
              valueFrom:
                secretKeyRef:
                  name: keycloak-secret
                  key: POSTGRES_DB
            - name: DB_PORT
              valueFrom:
```

```
        secretKeyRef:
          name: keycloak-secret
          key: POSTGRES_PORT
      - name: DB_USER
        valueFrom:
          secretKeyRef:
            name: keycloak-secret
            key: POSTGRES_USER
      - name: DB_PASSWORD
        valueFrom:
          secretKeyRef:
            name: keycloak-secret
            key: POSTGRES_PASS
      - name: DB_VENDOR
        value: POSTGRES
      - name: JGROUPS_DISCOVERY_PROTOCOL
        value: dns.DNS_PING
      - name: JGROUPS_DISCOVERY_PROPERTIES
        value: "dns_query=keycloak"
      - name: CACHE_OWNERS_COUNT
        value: "2"
      - name: CACHE_OWNERS_AUTH_SESSIONS_COUNT
        value: "2"
    ports:
      - name: jgroups
        containerPort: 7600
      - name: http
        containerPort: 8080
      - name: https
```

```
            containerPort: 8443
        readinessProbe:
          httpGet:
            path: /auth/realms/master
            port: 8080
```

A.6.4 Configuração do Service

```
apiVersion: v1
kind: Service
metadata:
  name: keycloak-balance
  namespace: nanostim-namespace
  labels:
    app: keycloak
spec:
  ports:
    - name: https
      port: 443
      targetPort: 8443
      nodePort: 32001
  selector:
    app: keycloak
  type: LoadBalancer
```

A.7 Teste de requisições assíncronas

```
import asyncio
import requests
import datetime
```

```
import jwt

from keycloak import KeycloakOpenID
requests.urllib3.disable_warnings()

task_inicial = 10
task_final = 100000
task_increase = 100
tasks = task_inicial

url_service = "url_service"

token_sso = None
token_pseudo = None

media_tempo= lambda lista:sum(map(
    lambda x:x['delta'].total_seconds(),lista))/len(lista)
count_sucess = lambda lista:sum(map(lambda x:x['result'],lista))

async def chronometer_async( function,*arg, **keys):
  info = {"start" : datetime.datetime.now()}
  info['result'] = await function(*arg,**keys)
  info["end"] = datetime.datetime.now()
  info['delta'] = info['end'] - info['start']
  return info

async def multiple_tasks( function, *arg, **keys):
  x = [asyncio.create_task(chronometer_async(function,*arg,**keys))
       for i in range(tasks)]
```

```
  y = [await y for y in x]
  return count_sucess(y), media_tempo(y) , y

def get_token(user="user_teste", passw="testepass"):
  url = f"https://{url_service}:"+
    "9395/auth/realms/patient/protocol/openid-connect/token"
  data = {
    "client_id" : "management_client",
    "client_secret" : "4secret_sample",
    "username" : "user_teste",
    "password" : "testepass",
    "grant_type" : "password",
  }
  return requests.post(url,data=data, verify=False).json()['access_token']

def get_pseudonimo():
  url = f"http://{url_service}:5002/api/pseudonymization/"
  header = {"SSO-TOKEN": token_sso}
  return requests.get(url, headers=header).json()['access_token']

def data_with_pseudo():
  url = f"http://{url_service}:7879/api/treatments/"
  json = {
    "date_start": "2021-10-26",
    "date_end": "2021-10-26",
    "pathology": "string",
    "patient_pseudonyme": "string",
    "professional_pseudonyme": "string"
  }
```

```
  header = {
    "TOKEN" : token_pseudo
  }
  return requests.post(url,json=json,headers=header).text

# teste de login de utilizadores
# retorna 1 caso sucesso ou 0 caso ocorra falha
async def login_patient( user="user_teste", passw="testepass" ):
  token = get_token(user,passw)
  try :
    jwt.get_unverified_header(token)
    return 1
  except: return 0

async def login_pseudonime():
  token = get_pseudonimo()
  try :
    jwt.get_unverified_header(token)
    return 1
  except: return 0

async def using_pseudonime():
  data = data_with_pseudo()
  return 1 if "treatment_id" in data else 0

def save_result(name, succ, media, result):
  print(f'{tasks}, {name}, {succ-tasks}, {media}')

def start_tests():
```

```
    global tasks, token_sso, token_pseudo
    while tasks <= task_final:

        save_result("login Keycloak",
            *asyncio.run(multiple_tasks(login_patient)))

        token_sso = get_token()
        save_result("rota auth SSO",
            *asyncio.run(multiple_tasks(login_pseudonime)))

        token_pseudo = get_pseudonimo()
        save_result("rota auth PSEU",
            *asyncio.run(multiple_tasks(using_pseudonime)))

        tasks += task_increase

start_tests()
```