Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Framework para Controle de Acesso em Function-as-a-Service (FaaS)

Matheus de Sousa Lemos Fernandes

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof.a Dr.a Aletéia Patrícia Favacho de Araújo Von Paumgartten

Brasília
2022

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Framework para Controle de Acesso em Function-as-a-Service (FaaS)

Matheus de Sousa Lemos Fernandes

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.a Dr.a Aletéia Patrícia Favacho de Araújo Von Paumgartten (Orientadora)
CIC/UnB

Prof. Dr. João José Costa Gondim
CIC/UnB

Prof. Dr. Flávio de Barros Vidal
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 10 de Maio de 2022

Dedicatória

Dedico este trabalho a todas as pessoas que contribuíram, seja com conhecimento ou palavras de encorajamento, para que este trabalho viesse a ser concluído. Em especial, dedico este trabalho à minha namorada Bárbara Castro, que em todos os momentos esteve ao meu lado, fornecendo todo o seu apoio e suporte. Dedico-o também aos meus irmãos Cecília, Raphael e Daniela, que contribuíram para a estruturação deste trabalho, cada um à sua maneira. Por fim, dedico este trabalho aos meus queridos pais Dilson e Edileusa, que me ensinaram o valor da persistência e da perseverança, e sempre me apoiaram a buscar conhecimento e nunca parar de aprender.

Agradecimentos

Agradeço principalmente à Prof.a Dr.a Aletéia Patrícia Favacho de Araújo Von Paumgartten por ter aceitado me orientar neste desafiador trabalho, e por ter aberto meus olhos para as infinitas possibilidades deste incrível universo da computação em nuvem. Agradeço também ao Leonardo Rebouças de Carvalho, por todo o auxílio prestado para a elaboração deste trabalho ao longo do último ano. Agradeço ao Prof. Dr. João José Costa Gondim, que me auxiliou a direcionar meus estudos sobre segurança computacional, tão crucial para este trabalho. Gostaria, por fim, de agradecer a todos os professores do Departamento de Ciência da Computação da Universidade de Brasília.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Controle de acesso é uma parte importante do uso de serviços em ambientes de nuvem. É necessário que o ambiente limite o acesso de usuários de acordo com a sua identidade e as permissões existentes. Essa tarefa é executada, em ambientes de nuvem, por ferramentas de Gestão de Identidade e Acesso (do inglês *Identity and Access Management* ou IAM). Entretanto, o uso de ferramentas de IAM limita o usuário aos serviços providos pelo provedor de nuvem em questão. Este trabalho apresenta um *framework* de controle de acesso para ambientes de Função como um Serviço (do inglês *Function-as-a-Service* ou FaaS) em nuvem pública. A segurança e o desempenho do *framework* são avaliados por meio de requisições de rede, de modo a validar que a ferramenta garante, de maneira eficaz, a autenticação e autorização dos usuários de forma segura em ambientes de FaaS. Os resultados iniciais indicam que o *framework* proposto oferece uma alternativa ao uso limitado de ferramentas tradicionais de IAM, as quais limitam o acesso de recursos a usuários cadastrados em provedores de nuvem pública.

Palavras-chave: Computação em Nuvem, Controle de Acesso, FaaS, Autenticação, Autorização

Abstract

Access control is an important part of using services in cloud environments. The environment must limit user access according to their identity and existing permissions. This task is performed, in cloud environments, by Identity and Access Management (IAM) tools. However, the use of IAM tools limits user access to the services provided by the cloud provider in question. This work presents an access control *framework* for Function as a Service (FaaS) environments in public cloud. The security and performance of the *framework* are evaluated through network requests, in order to validate that the tool effectively guarantees secure authentication and authorization of users in FaaS environments. Initial results indicate that the proposed *framework* offers an alternative to the limited use of traditional IAM tools, which limit resource access to users registered with public cloud providers.

Keywords: Cloud Computing, Access Control, FaaS, Authentication, Authorization

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

2FA Two-Factor Authentication.

ABAC Attribute Based Access Control.

API Application Programming Interface.

AS Authorization Server.

AWS Amazon Web Services.

CEO Chief Executive Officer.

CSRF Cross Site Request Forgery.

CWE Common Weakness Enumeration.

DAC Discretionary Access Control.

EC2 Elastic Compute Cloud.

FaaS Function-as-a-Service.

FTP File Transfer Protocol.

GCE Google Compute Engine.

GCP Google Cloud Platform.

HSTS HTTP Strict Transport Security.

HTTPS Hypertext Transfer Protocol Secure.

IaaS Infrastructure-as-a-Service.

IAM Identity and Access Management.

IdP Identity Provider.

JWT JSON Web Token.

MAC Mandatory Access Control.

NAT Network Address Translation.

NIST National Institute of Standards and Technology.

OASIS Organization for the Advancement of Structured Information Standards.

OC OAuth Consumer.

OIDC OpenID Connect.

OWASP Open Web Application Security Project.

PaaS Platform-as-a-Service.

RBAC Role Based Access Control.

RO Resource Owner.

RS Resource Server.

SaaS Software-as-a-Service.

SAML Security Assertion Markup Language.

SFA Single-Factor Authentication.

SMTP Simple Mail Transfer Protocol.

SSH Secure Shell.

SSO Single Sign-On.

TLS Transport Layer Security.

Capítulo 1

Introdução

Computação em nuvem é um paradigma surgido no início do século XXI, que veio para oferecer computação como um serviço. Esta nova forma de prestação de serviço apresenta uma nova solução para hospedagem de computacionais na Internet, dado que retira do desenvolvedor ou usuário final a necessidade de manter seus produtos ou serviços em sua própria infraestrutura. Isso é feito permitindo que um provedor de serviço, o chamado provedor de nuvem, realize a hospedagem e a manutenção da infraestrutura subjacente. O provedor, em contrapartida, permite o acesso a essa infraestrutura, cobrando taxas de utilização dos serviços de nuvem por parte do usuário final, de acordo com o consumo do serviço.

Um dos tipos de serviço disponibilizados pelos provedores de nuvem é o chamado Função como um Serviço (do inglês, *Function-as-a-Service* - FaaS). Nesse tipo de serviço, o usuário executa código, escrito em alguma linguagem de programação, na infraestrutura do provedor, sem se preocupar com a infraestrutura subjacente ou com a configuração do ambiente, visto que isso é totalmente gerido pelo provedor de nuvem. Esse código é encapsulado em unidades atômicas chamadas funções, que definem as estruturas necessárias para a execução do código em um ambiente de nuvem. O uso de funções permite que um desenvolvedor teste de forma rápida a viabilidade de um programa, visto que o esforço é totalmente voltado para o desenvolvimento do mesmo [11].

Contudo, serviços de nuvem, ainda que cômodos e eficientes na prestação de serviço, também precisam lidar com o necessidade de manter, de forma segura, dados dos usuários que os utilizam. Esses dados podem muitas vezes ser de caráter privado e confidencial, como é o caso de credenciais de acesso (email, nome de usuário, senhas). A necessidade de armazenar e trafegar esses dados de forma confiável levou à criação de ferramentas especializadas, presentes na maioria dos provedores de nuvem existentes atualmente. Essas ferramentas são chamadas ferramentas de Gestão de Identidade e Acesso (do inglês, *Identity and Access Management* - IAM). Essas ferramentas de IAM utilizam técnicas

para proteger o acesso de determinados recursos de usuários não autorizados a acessá-los. Dessa forma, é necessário que um usuário do provedor não só ateste sua identidade, mas que também possua as permissões necessárias para utilizar os recursos. Esses mecanismos de IAM, entretanto, exigem que um usuário utilize apenas os serviços daquele provedor, dado que suas credenciais não são compartilhadas entre diferentes provedores de nuvem, de modo que o usuário fica "preso"aos serviços de um mesmo provedor.

Diante do exposto, este trabalho apresenta um *framework* que oferece uma alternativa à utilização de ferramentas de IAM dos provedores de nuvem para acessar serviços de FaaS. Esse *framework* implementa um conjunto de módulos para executar funções em ambientes de FaaS, realizando todo o processo de Autenticação (verificação da identidade) de um usuário. Após o usuário ser autenticado, o *framework* utiliza permissões definidas pelo administrador da ferramenta de IAM Keycloak para geração de *tokens* de acesso. Um usuário do *framework* pode então executar funções, de forma segura, em um ambiente de FaaS, utilizando o *token* gerado, visto que as verificações necessárias para permitir seu acesso são feitas antes da execução de uma função.

O trabalho proposto possui o objetivo de implementar um *framework* que promova a execução segura de funções em um ambiente de FaaS, por meio de verificação de identidade e permissões de usuários do *framework*. Para isso, é implementada uma interface com usuário orientada a requisições de rede, utilizando uma API RESTful. Além disso, para assegurar a comunicação segura entre as partes, foram implementadas recomendações de segurança para aplicações web.

Para apresentar o *framework* proposto, este trabalho foi dividido em mais cinco capítulos. O Capítulo 2 trata das definições e contextualização de computação em nuvem, desde sua concepção conceitual até as definições formais surgidas posteriormente. Além disso, são apresentados os conceitos de provedor de nuvem, assim como apresentação dos provedores de nuvem mais relevantes da atualidade, modelos de serviço e modelos de implantação. O Capítulo 3 trata das definições de controle de acesso, assim como seus dois componentes principais, Autenticação e Autorização. Também aborda os principais métodos de Autenticação e Autorização utilizados em ambientes de nuvem e em aplicações web, além de ferramentas de IAM dos provedores de nuvem e ferramentas de código aberto que podem ser utilizadas por usuários finais. Em seguida, o Capítulo 4 apresenta a descrição do *framework* de controle de acesso, e descreve a arquitetura e o fluxo de execução do mesmo. O Capítulo 5 apresenta os experimentos realizados para averiguar a segurança e o desempenho do *framework* proposto, e apresenta os resultados dos testes executados. Por fim, o Capítulo 6 apresenta as conclusões atingidas neste trabalho, e os passos futuros para aprimoramento do *framework* proposto, assim como novos experimentos a serem realizados.

Capítulo 2

Computação em Nuvem

Computação em nuvem é um modelo computacional surgido no início do século XXI, como uma releitura de um antigo conceito, a computação remota [12]. Enquanto em anos anteriores a computação era feita em infraestrutura local ou remota, de posse dos detentores das máquinas, a computação em nuvem emergiu com a proposta de oferecer computação como serviço. O conceito de fornecer serviços de computação de forma remota para um cliente não é sem precedentes: ao surgirem os primeiros *mainframes*, máquinas clientes utilizavam os recursos de uma máquina central compartilhada, utilizando o conceito de compartilhamento de tempo, ou *timesharing* [12]. Assim, ainda há usuários que necessitam de poder computacional e recursos não disponíveis em suas máquinas locais [13]. Nesse contexto, A computação em nuvem surgiu para suprir essa necessidade.

A origem do conceito da computação em nuvem data de meados de 1996, e é originalmente atribuído a duas pessoas que trabalharam na empresa Compaq Computers, George Favaloro e Sean O'Sullivan [12]. Contudo, passou-se mais uma década antes que o conceito fosse formalizado em um termo próprio para descrever esse novo modelo. Em 2006, o CEO da Google Eric Scmidt introduziu o termo em uma conferência [12], e a partir de então o termo ganhou considerável relevância. Em meados de 2011, o NIST (National Institute of Standards and Technology), órgão que regulamenta o setor de computação nos Estados Unidos e que tem relevância global, definiu computação em nuvem como [14]:

> A computação em nuvem é um modelo para permitir acesso de rede onipresente, conveniente e sob demanda a um conjunto de recursos de computação configuráveis (por exemplo, redes, servidores, armazenamento, aplicativos e serviços) que pode ser rapidamente provisionado e liberado com o mínimo esforço de gerenciamento ou interação com o provedor de serviços.

Essa definição engloba uma gama de serviços que envolvem tanto infraestrutura (armazenamento, processamento) quanto programas disponibilizados pela rede como substitutos

para recursos locais, acessíveis remotamente. Além disso, computação em nuvem aborda uma série de características arquiteturais [14]:

- Serviço sob demanda e self-service: o usuário pode provisionar serviços de computação, a exemplo de armazenamento e servidores;
- Acesso ubíquo: o usuário pode acessar os recursos da nuvem por meio de dispositivos conectados à Internet;
- Agrupamento de recursos: recursos de computação disponibilizados pelo provedor de nuvem são agrupados para servir uma grande quantidade de usuários;
- Elasticidade rápida: recursos podem ser alocados e desalocados, muitas vezes de forma automática, para satisfazer a demanda pelos serviços;
- Serviço medido e tarifado: ambientes de computação em nuvem controlam e otimizam os recursos utilizados, por meio de métricas para avaliar o uso dos serviços disponibilizados.

Essas características muitas vezes não podem ser alcançadas por organizações sem equipamento especializado. A computação em nuvem surge então para disponibilizar serviços computacionais com mínimo esforço gerencial por parte dos usuários, suprindo uma necessidade cada vez mais presente. Dada essa necessidade, empresas e organizações especializadas surgiram para satisfazer essa demanda de mercado, os chamados provedores de nuvem. Um provedor de nuvem é uma pessoa, organização ou entidade responsável por tornar um serviço disponível às partes interessadas [15]. Dessa forma, essa entidade passa a oferecer serviços de forma terceirizada a outras entidades e ornanizações. Provedores de nuvem serão abordados em detalhe na Seção 2.2.

O NIST também define serviços de nuvem em quatro modelos de implantação e três modelos de serviço [14]. Modelos de implantação serão definidos na Seção 2.1 e modelos de serviço serão abordados na Seção 2.3.

2.1 Modelos de Implantação

O tipo de ambiente de computação em nuvem podem ser classificado de acordo com o modo como é disponibilizado para os usuários. Os tipos de ambiente são definidos como modelos de implantação. Atualmente, o NIST separa os modelos de implantação para computação em nuvem em quatro categorias [14]:

- Nuvem privada: a infraestrutura disponibilizada é acessível apenas a um grupo de usuários pertencentes a uma organização. Esse tipo de infraestrutura pode ser gerido pela própria organização ou por terceiros;

- Nuvem comunitária: a infraestrutura disponibilizada é acessível a uma comunidade de usuários, e os recursos são voltados para uso por essa comunidade. Tal infraestrutura pode ser gerida por uma ou mais entidades pertencentes à organização;
- Nuvem pública: a infraestrutura de nuvem é acessível pelo público em geral, e pode ser gerida por organizações públicas ou privadas (ou ambas);
- Nuvem híbrida: tal modelo pode ser visto como uma mescla de infraestruturas privada, comunitária e pública, de forma a oferecer serviços específicos à organização que a utiliza.

Atualmente, devido à alta demanda por serviços de nuvem, tem surgido empresas e plataformas voltadas a oferecer serviços de nuvem pública sob demanda, os chamados provedores de nuvem pública, que serão discutidos em detalhe na próxima seção.

2.2 Provedores de Nuvem Pública

Em um âmbito de disponibilização de serviço é necessário que uma das partes ofereça esses serviços de computação em nuvem. O provedor de nuvem é quem detém os recursos computacionais e de rede, e disponibiliza os mesmos para usuários, também chamados consumidores de nuvem [15]. Devido à disponibilização de serviços de forma pública por parte dos provedores, os mesmos são chamados de provedores de nuvem pública, em contraste com nuvens privadas, que não são disponibilizadas para o público geral e são voltadas para uso particular de empresas e outras entidades.

Atualmente existem sete principais provedores de nuvem pública em atuação, de acordo com o relatório anual de 2021 do Gartner [1], que é uma companhia de consultoria e pesquisa em tecnologia da informação. Na Figura 2.1, é possível ver os maiores competidores no cenário de nuvem pública. Amazon Web Serices (AWS) [16], Google Cloud Platform (GCP) [17] e Microsoft Azure [18] lideram o mercado de nuvem pública. Além deles, existem também o Alibaba Cloud [19], que oferece serviços de nuvem majoritariamente para países asiáticos, e que atendem necessidades mais específicas. Por fim, tem-se os operadores de nicho, Oracle Cloud [20], IBM Cloud [21] e Tencent Cloud [22], que oferecem produtos especializados. Este ano, o relatório não incluiu provedores na categoria de Desafiadores. Esses provedores oferecem serviços em cada uma das categorias principais enunciadas pelo NIST (IaaS, PaaS e SaaS), descritos na Seção 2.3. As próximas seções descrevem os principais provedores públicos de nuvem.

Figura 2.1: Quadrante mágico do Gartner de julho de 2021 [1].

2.2.1 Amazon Web Services (AWS)

Amazon Web Services [16] é o braço tecnológico da empresa Amazon. Ele foi fundado em 2006 como um meio da empresa evitar que sua robusta infraestrutura computacional ficasse ociosa. Atualmente é o maior provedor de nuvem pública, e possui uma quantidade considerável do mercado de nuvem mundial, estando à frente de todos os outros provedores [23].

O AWS oferece serviços de nuvem em diversas localizações geográficas, de modo a oferecer qualidade de serviço em qualquer lugar do mundo. Seus datacenters estão distribuídos em 26 regiões e 84 zonas de disponibilidade [2], como apresentado na Figura 2.2. Zonas de disponibilidade representam áreas fisicamente isoladas em *datacenters*, consistindo de diversos servidores [24]. Além disso, o AWS oferece diversos serviços de computação, armazenamento, rede, entre outros. Algumas das funcionalidades oferecidas incluem [23]:

- Elastic Compute Cloud (EC2): Serviço de computação em nuvem [25];
- Simple Storage Service (S3): Serviço de armazenamento em objetos [26];
- Elastic Kubernetes Service (EKS): Serviço gerenciado de contêineres para desenvolvimento e implantação de aplicações em Kubernetes [27].

Figura 2.2: Mapa de datacenters da AWS [2].

2.2.2 Google Cloud Platform (GCP)

Google Cloud Platform (GCP) [17] é a plataforma de nuvem da Google, lançada em 2008 para competir no crescente mercado de serviços de nuvem. O GCP, assim como o AWS, surgiu para fornecer serviços de computação sob demanda na grande infraestrutura já existente utilizada pela empresa. O GCP também oferece serviços que se encaixam nos três modelos de serviço definidos pelo NIST.

O GCP possui diversos datacenters distribuídos em diversos países do mundo, de modo a prover o melhor serviço para clientes em qualquer parte do planeta. Atualmente existem 29 regiões e 88 zonas de disponibilidade disponíveis para clientes do GCP. A Figura 2.3 apresenta a localização dessas zonas. O GCP também oferece uma gama de serviços de nuvem, que abrangem computação, armazenamento, contêineres, entre outros. Alguns dos serviços oferecidos são [28]:

- Google Kubernetes Engine (GKE): Serviço de contêineres para implantação de aplicações em Kubernetes [29];
- Cloud Storage: Serviço de armazenamento de objetos em nuvem [30];
- Cloud SQL: Serviço de bancos de dados MySQL, PostgreSQL e SQL Server em nuvem [31].

Figura 2.3: Regiões de disponilidade do Google Cloud Platform [3].

2.2.3 Microsoft Azure

Microsoft Azure [18] é a plataforma de nuvem da Microsoft, lançada em 2010 para concorrer com as duas grandes plataformas, AWS e GCP. O grande diferencial do Azure, no entanto, foi oferecer na nuvem produtos existentes desenvolvidos pela Microsoft. Um exemplo é o Azure Active Directory, cujo equivalente é o bem consolidado Microsoft Active Directory, utilizado para gestão de usuários em empresas.

O Azure possui diversos datacenters distribuídos geograficamente em 61 regiões de disponibilidade ao redor do globo [4]. A Figura 2.4 apresenta a localização de cada região oferecida. O Azure também oferece diversos serviços de nuvem dos mais diversos tipos, que incluem computação, armazenamento, gestão de identidade, entre outros. Alguns dos serviços oferecidos pelo Azure são:

- Azure Active Directory: Serviço de gestão de identidades em nuvem [32];
- CosmosDB: Serviço de banco de dados não-relacional gerenciado em nuvem [33];
- Azure Kubernetes Service (AKS): Serviço de contêineres para desenvolvimento de aplicações em Kubernetes [34].

2.2.4 Outros Provedores de Nuvem Pública

Além dos grandes provedores de nuvem pública citados anteriormente, outros provedores também se destacam no cenário de nuvem global:

Figura 2.4: Regiões do Microsoft Azure [4].

- Alibaba Cloud: também conhecida por Aliyun [19], é uma companhia subsidiária do conglomerado Alibaba Group, que fornece serviços de nuvem majoritariamente ao mercado chinês. Fundada em 2011, o Alibaba Cloud oferece serviços de nuvem a partir de 25 centros de dados (datacenters) geograficamente distribuídos na China continental e em outros países.
- Oracle Cloud [20]: serviço de nuvem oferecido pela empresa Oracle. O serviço foi disponibilizado inicialmente em 2016, e conta com serviços disponíveis por meio de um ampla infraestrutura, com datacenters presentes em diversos locais pelo mundo.
- IBM Cloud [21]: plataforma de serviços de nuvem da IBM, uma empresa consolidada no mercado mundial de computadores. Fundada em 2013, sob o nome de Bluemix, surgiu como um serviço de nicho, com produtos voltados para inteligência artificial e aprendizado de máquina.
- Tencent Cloud [22]: divisão de computação do conglomerado chinês Tencent, empresa de tecnologia com sede na cidade de Shenzhen, na China. É o mais novo grande competidor no mercado de nuvem pública, não tendo aparecido no relatório do Gartner de 2020. O provedor oferece serviços já consolidados no mercado de nuvem, desde serviços de computação e armazenamento a serviços de inteligência artificial.

A Tabela 2.1 apresenta um comparativo dos provedores de nuvem pública apresentados, de acordo com ano de criação e quantidade de zonas de disponibilidade oferecidas.

Provedor	Ano de Criação	Zonas de Disponibilidade
Amazon Web Services	2006	84
Google Cloud Platform	2008	88
Microsoft Azure	2010	113
Alibaba Cloud	2009	84
Oracle Cloud	2016	38
IBM Cloud	2013	19
Tencent Cloud	2013	70

Tabela 2.1: Comparativo de provedores de nuvem pública.

2.3 Modelos de Serviço

O NIST classifica os modelos de serviço de nuvem de acordo com o nível de gestão e controle que o provedor tem sobre os recursos computacionais. Ele define três modelos principais, em ordem crescente de gestão por parte dos provedores, os quais são Infraestrutura como um Serviço (IaaS), Plataforma como um Serviço (PaaS) e Software como um Serviço (SaaS) [15]. A Figura 2.5 mostra como esses três modelos se comparam a utilizar máquinas *on-premises*, definida na imagem como "Traditional IT". Cada modelo de serviço tem suas particularidades, que serão definidas a seguir.

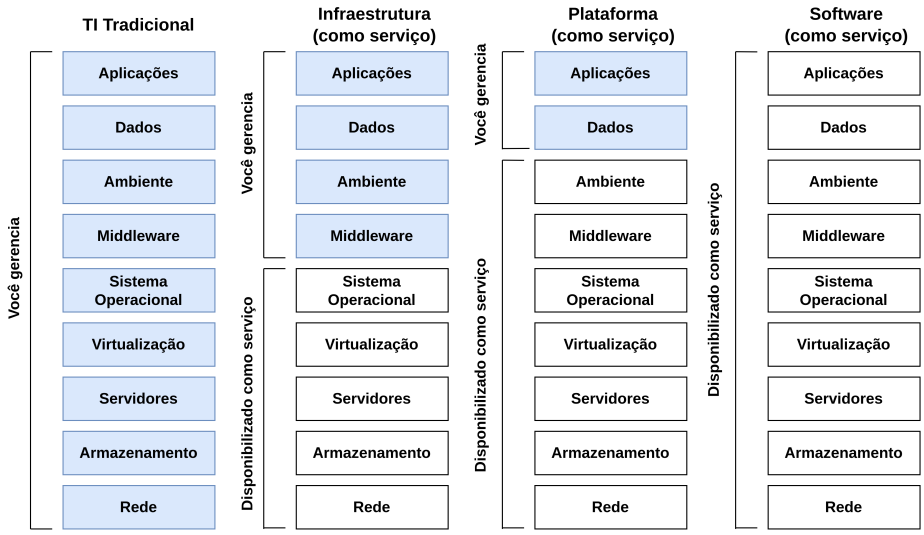

Figura 2.5: Níveis de gestão para cada modelo de serviço, adaptado de [5].

2.3.1 Infraestrutura como um Serviço

Na Infraestrutura como um Serviço (Infrastructure-as-a-Service ou IaaS), o provedor provisiona redes, servidores, armazenamento ou sistemas operacionais [14] e disponibiliza

estes recursos ao usuário por meio de uma série de interfaces e abstrações. Serviços de IaaS são caracterizados pelo conceito de virtualização de hardware, no qual um consumidor de nuvem pode executar suas aplicações no hardware hospedeiro do provedor de nuvem [28]. Dessa forma, esse modelo oferece mais liberdade ao consumidor do serviço, visto que o mesmo tem acesso menos restrito aos recursos disponibilizados [15].

Produtos neste modelo de serviço envolvem máquinas (máquinas virtuais compartilhadas e dedicadas, máquinas *bare-metal*), armazenamento (discos rígidos virtuais, sistemas de arquivos) e serviços de rede (firewall, NAT). Alguns exemplos de produtos de IaaS oferecidos pelos provedores na Seção 2.2 são:

- Amazon Elastic Compute Cloud (EC2): serviço de máquinas virtuais do Amazon Web Services [25];
- Google Compute Engine (GCE): serviço de máquinas virtuais do Google Cloud Platform;
- Azure managed disks: serviço de armazenamento em bloco para máquinas virtuais oferecido pelo Microsoft Azure [35];
- File Storage NAS: Serviço de armazenamento em sistema de arquivo do Alibaba Cloud [36];
- Virtual Cloud Network: Serviço de redes privadas oferecido pelo Oracle Cloud [37];
- IBM Cloud Virtual Servers: Serviço de máquinas virtuais públicas dedicadas do IBM Cloud [38];
- Cloud File Storage: Serviço de armazenamento em arquivos do Tencent Cloud [39].

2.3.2 Plataforma como um Serviço

Na Plataforma como um Serviço (Platform-as-a-Service ou PaaS), o provedor gerencia a infraestrutura e oferece ao usuário ferramentas, linguagens de programação, bibliotecas e serviços adequados para o desenvolvimento de aplicações [14]. Neste modelo de serviço o consumidor de nuvem não tem acesso à infraestrutura subjacente, mas tem acesso às aplicações desenvolvidas utilizando o ambiente disponibilizado [15]. Alguns produtos de PaaS oferecidos pelos provedores de nuvem pública são:

- Amazon Elastic Beanstalk: plataforma para desenvolvimento e implantação de aplicações web [40];
- Google Kubernetes Engine: ambiente para implantação e gestão de aplicações em *containers* utilizando Kubernetes [29];

- Azure Cloud Services: plataforma de criação e implantação de aplicações web do Microsoft Azure [41];
- Alibaba Cloud Container Service for Kubernetes: serviço do Alibaba Cloud para implantação de aplicações em *containers* utilizando Kubernetes [42];
- Application Development: plataforma do Oracle Cloud para desenvolvimento integrada para aplicações de nuvem [43];
- IoT Hub: plataforma do Tencent Cloud para auxiliar desenvolvedores na implementação de soluções para Internet das Coisas [44].

2.3.3 Software como um Serviço

No Software como um Serviço (Software-as-a-Service ou SaaS) o provedor oferece produtos de software voltado para o usuário final, acessível por meio de alguma interface, que pode ser um navegador de Internet ou interface de programa [14]. Dessa forma, o provedor de nuvem realiza configuração, manutenção e gestão do ciclo de vida dessas aplicações, de modo a gerenciar o *software* como um todo. Algumas categorias de produtos de SaaS são *webmail*, *webchat* e lojas *online* (*e-commerce*). Alguns produtos de SaaS existentes atualmente são:

- Google Workspace: ferramentas de colaboração profissional da Google, que inclui *webmail*, armazenamento de arquivos, gestão de calendário, entre outras aplicações [45];
- Office 365: ferramentas de escritório, semelhante ao pacote Office distribuído pela Microsoft
- Media Processing Service: serviço de processamento de mídia oferecido pelo Tencent Cloud [46].

2.3.4 *Anything-as-a-Service*

Com o tempo, novos modelos de serviço surgiram para atender demandas específicas. Assim, e a medida que novos serviços eram criados, foi necessário categorizar novamente o paradigma de computação em nuvem, visto que as definições previstas pelo NIST [15] já não abrangiam a complexidade dos produtos oferecidos. Atualmente, há uma grande quantidade de serviços em nuvem, e isso marcou a criação do termo *Anything-as-a-Service*, ou Xaas, que em tradução livre pode ser entendido como "Qualquer coisa como um serviço". Logo, o termo *Anything-as-a-Service* é definido como [47]:

`Qualquer tecnologia disponibilizada através da Internet que costumava ser disponibilizada localmente.`

Dada essa definição abrangente, surgiram vários tipos de categoriação para os mais variados produtos que, antes utilizados localmente, passaram a ser ofertados como serviços em nuvem. Alguns desses serviços são [47]:

- Banco de dados como um serviço, *Database-as-a-Service* (DBaaS/DaaS);
- Processos de negócio como um serviço, *Business-as-a-Service* (BPaaS);
- *Malware* como um serviço, *Malware-as-a-Service* (MaaS);
- Função como um serviço, *Function-as-a-Service* (FaaS).

Entre os modelos citados, o modelo de serviço *Function-as-a-Service* (FaaS) tem ganhado proeminência nos últimos anos. Assim sendo, dada a importância deste modelo para este trabalho, ele será melhor definido na Seção 2.4.

2.4 Função como um Serviço

Função como um Serviço (do inglês Function-as-a-Service ou FaaS) é um modelo de serviço que surgiu em 2014, com o surgimento do AWS Lambda [48], que é uma ferramenta que revolucionou a maneira como as aplicações são implantadas em ambientes de nuvem. Assim, o AWS Lambda iniciou o paradigma conhecido como Computação sem Servidor, do inglês *Serverless Computing*. O paradigma de Computação sem Servidor apresenta um novo modo de executar aplicações em nuvem, onde o usuário é tarifado apenas por tempo de execução, em contraste com pagamento por alocação de recursos [49], presente no paradigma de IaaS.

O principal componente do modelo FaaS são as funções, unidades atômicas de código que são interpretadas e executadas pelo ambiente de FaaS. Essas funções podem ser escritas em diversas linguagens de programação, desde que o provedor de nuvem ofereça um ambiente de execução para a linguagem. O modelo FaaS difere dos modelos tradicionais de computação de diversas formas, principalmente, em seu modelo arquitetural [48]:

- O ciclo de vida do programa é gerido completamente pelo ambiente de FaaS, sem necessidade de configuração por parte do desenvolvedor;
- Funções independem das características da máquina hospedeira (arquitetura, sistema operacional), em geral por meio de tecnologias de virtualização;

- Funções oferecem elasticidade automática, permitindo que o código do usuário se adapte à carga de execução;
- Usuários pagam apenas pelos recursos utilizados em um intervalo de tempo de execução das funções, não sendo taxado o tempo ocioso.

O modelo de FaaS dá ao desenvolvedor muita liberdade para implementar suas aplicações, enquanto o ambiente de nuvem realiza a gestão de todos os recursos computacionais. Além disso, com o FaaS, o aumento dos recursos necessários para executar funções computacionalmente intensivas (*auto-scaling*), e a distribuição da carga para execução paralela são tarefas exercidas automaticamente pelo provedor.

Enquanto modelos tradicionais de computação, como máquinas virtuais, podem realizar as mesmas tarefas que um ambiente de FaaS, do ponto de vista técnico, tais ferramentas, não são feitas para reagir a eventos, como um envio de arquivo para um sistema de armazenamento ou um pico de utilização em outro recurso do ambiente de nuvem [11], o que é uma característica fundamentais de FaaS.

2.4.1 Produtos de Função como Serviço

Atualmente, devido ao crescimento deste modelo, os grandes provedores de nuvem têm disponibilizado esse tipo de serviço. O produto que inaugurou esse novo modelo foi o AWS Lambda [50], desenvolvido pela AWS. O AWS Lambda é um ambiente que executa funções de usuário, e que permite interação com outros produtos da AWS. É muito utilizado para criação de pequenos procedimentos, de modo a reagir a eventos enviados pelo usuário ou outras ferramentas [51].

Um exemplo é o envio de imagens para um serviço de armazenamento de arquivos, como o Amazon S3 [51, 26]. O envio de uma imagem pode gerar algum tipo de gatilho que realizar diversas verificações: se a imagem é do formato correto ou se inclui *malware*, por exemplo [51]. Uma função do AWS Lambda pode ser desenvolvida para reagir a um evento enviado pelo S3, e realizar as verificações citadas acima sempre que um usuário realiza um envio de imagens. Após feitas as devidas verificações, a imagem pode então ser armazenada. Além do AWS, outros provedores de nuvem pública oferecem seus próprios produtos de FaaS. Alguns desse serviços são:

- Google Cloud Functions, oferecido pelo Google Cloud Platform [52];
- Azure Functions, oferecido pelo Microsoft Azure [53];
- Function Compute, oferecido Alibaba Cloud [54];
- Oracle Cloud Functions, oferecido Oracle Cloud [55];

- IBM Cloud Functions, oferecido pelo IBM Cloud [56];
- Serverless Cloud Function, oferecido pelo Tencent Cloud [57].

2.5 Considerações Finais

Neste capítulo foram apresentadas as definições e origens da computação em nuvem, desde seu contexto teórico inicial até as categorizações em modelos de serviço. Além disso, foram apresentados os conceitos de provedor de nuvem, e os principais provedores de nuvem no mercado de nuvem pública.

No próximo capítulo serão apresentados os desafios enfrentados por provedores de nuvem pública para manter a segurança de dados e usuários. Além disso, serão introduzidos os diferentes meios utilizados por desenvolvedores e administradores de sistema, para autenticar e autorizar usuários em ambientes de nuvem pública.

Capítulo 3

Controle de Acesso

Controle de acesso pode ser definido como a restrição de acesso a um local ou recurso específico [58]. É um conjunto de condições que determinam o acesso que uma pessoa tem a dados ou recursos, de forma a restringir o acesso desses recursos de acordo com privilégios determinados por um sistema [58].

Controle de acesso pode ser dividido em duas etapas distintas: autenticação e autorização. Essas duas etapas permitem que um sistema ofereça confidencialidade, integridade e disponibilidade dos recursos que elas protegem [58], e serão descritas em detalhe nas próxima seções.

3.1 Autenticação

Autenticação é o processo de aprovar o acesso de uma entidade a um determinado recurso por intermédio de outra entidade, verificando que a mesma tem as permissões necessárias [7]. Esse processo envolve diferentes meios de atestar a autenticidade dessa entidade, seja ela um usuário ou uma aplicação. Para realizar autenticação de usuários e aplicações, são utilizados mecanismos de autenticação. Tais mecanismos são divididos em dois grupos principais: mecanismos físicos e mecanismos digitais de autenticação, os quais serão descritos nas próximas seções.

Mecanismos físicos de autenticação são dispositivos, eletrônicos ou não, utilizados para atestar a identidade do portador e conferir (ou recusar) acesso a um recurso. Dessa forma, o acesso a recursos fica atrelado à posse do mecanismo. Alguns exemplos de mecanismos físicos são:

- Cartão inteligente ou *smartcard* [59];

- Biometria [60]: uso de informações fisiológicas ou comportamentais para identificação do usuário (impressão digital, reconhecimento de retina, reconhecimento de voz) [60].

Tais mecanismos, além de serem utilizados por si só, também podem ser utilizados junto a outros métodos de autenticação, na chamada autenticação em dois fatores, que será discutida em detalhe na Seção 3.1.2.

Mecanismos digitais, em contraste, são dispositivos computadorizados que verificam a identidade de usuários ou aplicações por meio de credenciais. Tais credenciais atestam que o usuário, ao conhecê-las, tem autoridade para acessar os recursos protegidos por elas [7]. Assim, mecanismos digitais de autenticação envolvem fornecer as credenciais a um autenticador, que verifica que as mesmas estão corretas e, assim, permitir acesso ao usuário. Entretanto, tais métodos de autenticação variam quanto à sua complexidade e grau de segurança fornecidos ao usuário. Os métodos mais comuns de autenticação existentes são autenticação por senha, Autenticação em dois fatores e chaves criptográficas [7].

3.1.1 Autenticação por Senha

No método de autenticação por senha (do inglês *password authentication*) o usuário utiliza um nome de usuário e uma senha [61]. Ambos são então verificados junto ao serviço de modo a verificar que o usuário existe e que as credenciais passadas estão corretas.

Esse método, entretanto, tem diversas vulnerabilidades, tanto do lado do cliente (usuário) quanto do lado do servidor, que mantém registro de todos os usuários que acessam seu serviço. Usuários podem, por exemplo, utilizar senhas muito simples e fáceis de lembrar, que podem ser facilmente descobertas [61]. Outro problema é que usuários costumam utilizar as mesmas credenciais para múltiplos serviços [62], o que aumenta a chance de ter suas credenciais disponibilizadas abertamente na Internet caso algum desses serviços seja comprometido.

Dadas as fragilidades de autenticação por senha, mecanismos foram criados para verificar a identitidade de um usuário de mais de uma forma, a chamada Autenticação em dois fatores.

3.1.2 Autenticação em Dois Fatores

Como visto na seção anterior, autenticação por senha é um método simples de verificação de identidade. Autenticação por senha trata-se de um exemplo de Autenticação de Fator Único (do inglês *Single-Factor Authentication* ou SFA), no qual apenas um mecanismo é utilizado para autenticar um usuário [63]. Entretanto, devido às vulnerabilidades de tais

métodos, verificou-se que Autenticação de Fator Único não era suficiente para oferecer segurança a aplicações, dada a grande quantidade de falhas de segurança [63], pois um atacante pode conseguir as credenciais de acesso em diversas partes da comunicação entre as partes [61].

Dessa forma, foi proposto o método de Autenticação em Dois Fatores (do inglês *Two-Factor Authentication* ou 2FA), um método de autenticação que acopla os chamados dados representativos (como é o caso de usuário e senha na autenticação por senha) com um fator de posse pessoal [63]. Esse fator de posse pessoal pode ser um mecanismo físico ou digital. Tais mecanismos fornecem uma camada extra de segurança ao prevenir que um atacante, de posse de uma das credenciais de um usuário, acessem um serviço se passando por ele [7]. Alguns dos mecanismos mais comuns de autenticação multifator são [7, 63]:

- Chave de Uso Único (*One-Time Password* ou OTP): código ou senha que expira após o primeiro uso, ou após um determinado período de tempo, como é o caso de aplicativos autenticadores, tais como o Google Authenticator [64] ou o FreeOTP [65];
- Captcha: código alfanumérico ou sequência de ações necessárias para permitir acesso a uma aplicação após digitadas as credenciais. *Captchas* são frequentemente utilizados para proteger aplicações web de *malware* automatizado direcionado às mesmas [7];
- Cartão inteligente ou *smartcard* [59]: Cartões físicos que oferecem acesso a um recurso em conjunto com uma credencial de usuário.

Ainda que o 2FA tenha apresentado uma melhora drástica na segurança de aplicações, existem vulnerabilidades que atacantes podem explorar de modo a conseguir acesso a recursos e aplicações. Tais vulnerabilidades vão de ataques de replay a ataques de adversário, e muitas vezes os ataques são direcionados ao dispositivo utilizado para realizar o segundo fator da autenticação [63].

3.1.3 Chaves Criptográficas

Além de autenticação por senha e 2FA, outro mecanismo muito utilizado, principalmente para acesso a máquinas remotas, é a autenticação utilizando chaves criptográficas, um dos métodos mais utilizados para acesso a serviços remotos. Esse mecanismo utiliza criptografia de chave pública para trafegar dados sensíveis por um meio inseguro, ou seja, entre a máquina do usuário e o servidor remoto. Um dos tipos mais comuns de criptografia de chave pública é o *Secure Shell* ou SSH [7].

SSH é um protocolo para login remoto e outros serviços de rede realizados em uma rede insegura [66]. Chaves SSH utilizam criptografia de chave pública para verificar, do lado do servidor, que uma chave enviada pelo usuário é válida. Para utilizar SSH é necessário que um par de chaves seja gerado, uma chave privada e uma chave pública. As chaves são mantidas na máquina cliente, e quando uma conexão SSH é aberta para um servidor, duas verificações são feitas:

- As chaves pública e privada do cliente são complementares;
- O usuário detentor da chave pública possui acesso à máquina remota.

A chave de usuário é então verificada por uma chave presente no servidor, chamada chave de hospedeiro (do inglês *host key*), de modo que a chave de usuário nunca é trafegada entre as partes [66]. Desde que o usuário esteja de posse da chave, é possível acessar o recurso remoto. Esse mecanismo, ainda que pouco utilizado em aplicações finais, é muito utilizado para acesso remoto a servidores por meio de desenvolvedores e administradores de sistema.

Ainda que chaves SSH ofereçam muita segurança para acesso remoto de aplicações, seu uso não é próprio para autenticação em aplicações web, da forma que autenticação por senha e 2FA o são. Além desses três métodos, outro método de autenticação tem ganhado proeminência em anos recentes, o *Single Sign-On*, que será descrito na próxima seção.

3.2 Single Sign-On (SSO)

Muitas vezes os mecanismos citados anteriormente não são suficientes para verificar permissões de um usuário. Todos os mecanismos citados se baseiam na gestão de credenciais e permissões por parte do provedor de serviço. Entretanto, pode ser desejável a um provedor de serviço terceirizar a gestão de usuários para outra entidade, de modo que verificação de identidade e permissões não é feita pelo mesmo.

Essa é a premissa do mecanismo *Single Sign-On*, ou SSO. Nesse mecanismo de autenticação, o usuário utiliza apenas um serviço de autenticação (e portanto, apenas uma credencial) para se autenticar em uma gama de serviços diferentes [7]. Esse método de autenticação, entretanto, apresenta a desvantagem de manter apenas uma credencial principal para o usuário, criando assim um único ponto de falha.

Mecanismos de SSO funcionam seguindo um mesmo processo, isto é, o usuário acessa um mecanismo de autenticação por meio de um *hyperlink* em uma aplicação ou *website*. Ao acessar o mecanismo, o usuário fornece suas credenciais para o serviço de SSO. Dado que as credenciais são válidas, o serviço retorna à aplicação algum tipo de resposta de

sucesso, de modo a indicar que o usuário possui as permissões necessárias. Por fim, de posse dessa resposta, a aplicação garante acesso ao usuário.

A entidade que fornece o serviço de SSO é chamado de Provedor de Identidade (Identity Provider ou IdP). Atualmente, existem três padrões principais de autenticação e gerenciamento de identidade por SSO: *Security Assertion Markup Language*, ou SAML, OAuth e OpenID Connect (OIDC), descritos a seguir.

3.2.1 SAML

O SAML é um mecanismo de autenticação desenvolvido pela Organização para o Avanço de Sistemas de Informação estruturados (*Organization for the Advancement of Structured Information Standards* ou OASIS) [6].

Esse método utiliza da noção de *Assertions*, estruturas que determinam uma identidade de usuário [67]. Em uma típica comunicação utilizando SAML, o usuário requisita acesso a uma aplicação. A aplicação por si gera uma requisição SAML. A requisição é encaminhada pelo navegador ao IdP, que verifica a requisição e autentica o usuário, retornado uma resposta SAML, que contém a *Assertion* necessária para a aplicação requisitante. Caso a resposta seja válida, a aplicação permite o acesso do usuário. A Figura 3.1 ilustra a requisição de acesso a um recurso utilizando SAML.

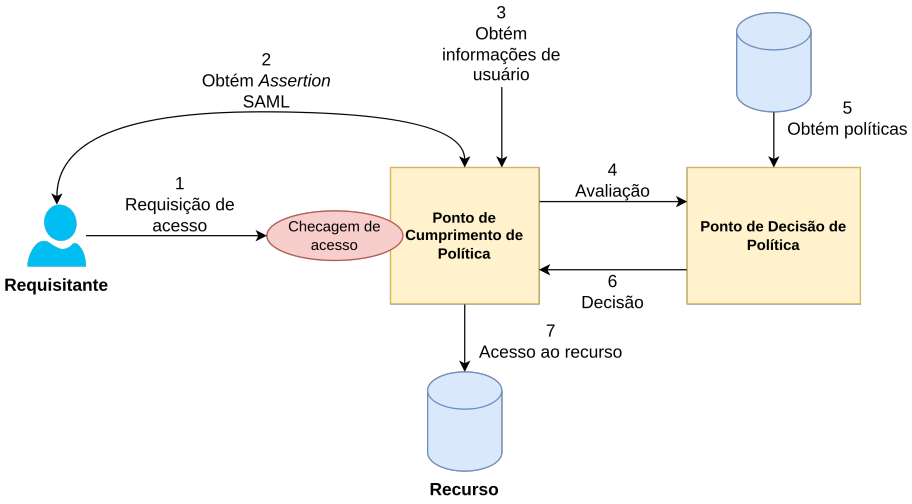

Figura 3.1: Funcionamento de uma requisição SAML, adaptado de [6].

3.2.2 OAuth

Outro padrão, que tem ganhado grande proeminência nos últimos anos, é o OAuth. O OAuth é um protocolo de delegação escalável, que envolve delegar um determinado processo a uma entidade terceira [67]. Dessa forma, o protocolo dá a uma aplicação o poder

de executar ações no lugar do usuário requisitante [67]. Em uma típica comunicação utilizando OAuth, são executados os seguintes passos:

1. Um usuário (denominado, na nomenclatura do OAuth, como *Resource Owner* ou RO) requisita acesso a recursos de posse de uma organização;
2. A organização (denominada *OAuth Consumer* ou OC), realiza uma requisição de um *token* de acesso e uma chave privada;
3. O Servidor de Autorização (*Authorization Server*, ou AS), retorna ao OC um *token* de acesso e a chave privada correspondente;
4. O OC retorna ao usuário uma URL, contendo o *token* de acesso e uma requisição de autorização por parte do usuário;
5. O RO então acessa o serviço do OC e clica na URL recebida;
6. o AS requisita ao usuário para permitir ou negar o OC;
7. RO permite acesso aos recursos
8. AS gera um *token* de acesso e redireciona-o para o OC;
9. OC envia o *token* de acesso e consegue os recursos para o usuário;
10. O Servidor de Recursos (*Resource Server* ou RS), detentor dos recursos protegidos, retorna os dados ao OC;
11. OC envia os recursos ao usuário.

A Figura 3 exemplifica as interações no processo de autenticação por OAuth.Assim, *tokens* OAuth podem ter diversos formatos, de modo a se adaptar a diferentes tipos de aplicação que necessitam desse tipo de processo de autenticação. Alguns dos mais comuns são o OAuth 2.0 JWT Spec, que descreve o uso de *tokens* em formato *JSON Web Token* (JWT); e OAuth 2.0 SAML Spec, que descreve o uso *token* de acesso como uma Assertion SAML [67].

3.2.3 OpenID Connect

Outro método de gestão de identidades digitais que tem obtido destaque é o *OpenID Connect*, ou OIDC. O OIDC é um protocolo aberto, que surgiu como uma evolução do *OpenID 2.0*, um protocolo de autenticação criado de forma independente dos dois protocolos citados anteriormente. Entretanto, o OIDC passou a ser construído a partir da especificação do OAuth 2.0 [7, 67], que já se mostrou como uma tecnologia robusta

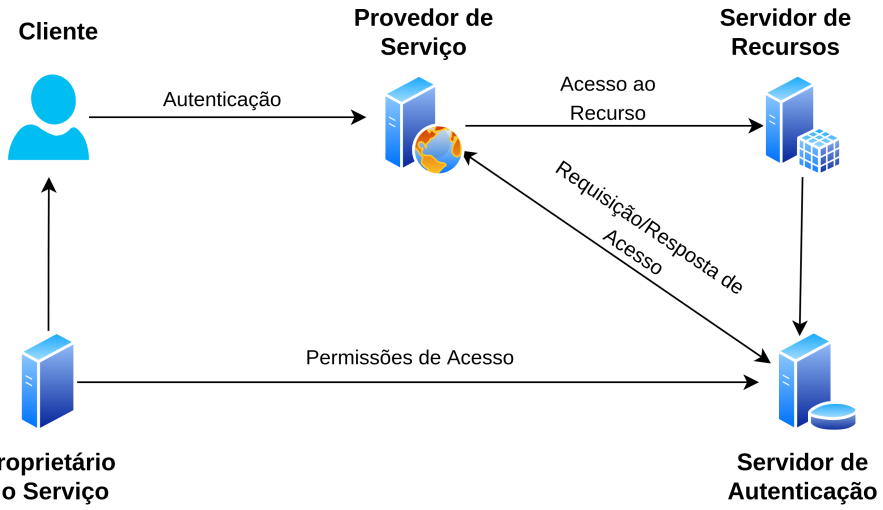

Figura 3.2: Processo de autenticação por OAuth, adaptado de [7].

de autenticação. Um IdP que implementa OIDC armazena uma lista de usuários, que criam suas contas em algum serviço de cadastro promovido pelo IdP. Essa conta, então, é utilizada para autenticar usuários junto a aplicações que utilizem esse IdP [7]. Dessa forma, basta que o usuário possua apenas um cadastro para que possa se autenticar em diversos serviços web.

3.3 Autorização e Controle de Acesso

Além do processo autenticação, descrito em detalhe na Seção 3.1, outra etapa essencial para garantir a segurança de recursos em um ambiente de nuvem é a autorização. Autorização é o processo de garantir ou negar acesso a recurso, de acordo com as permissões de usuário autenticado [7]. Tal método verifica que, após averiguada a identidade de um usuário, o mesmo também tem as permissões necessárias para acesso a um determinado recurso.

Atualmente, também existem diversos mecanismos para autorizar usuários autenticados a acessar recursos. Tais mecanismos utilizam de métodos diferentes para verificar permissões de usuário junto a uma aplicação. Quatro dos mecanismos mais utilizados atualmente serão apresentados a seguir.

3.3.1 Controle de Acesso Obrigatório

Controle de Acesso Obrigatório (do inglês *Mandatory Access Control* ou MAC) é um mecanismo de autorização tradicional que define diretos de acesso para usuários [7]. o

MAC é utilizado primariamente para limitar acesso a recursos de sistemas operacionais e sistemas de arquivos.

Em um modelo de autorização que utiliza o mecanismo MAC, o acesso aos recursos é definido unicamente por atributos de sujeito e objeto para garantir acesso [68]. Além disso, o acesso é definido por um monitor, usualmente o administrador do sistema. Dessa forma, o administrador define as regras de um sistema MAC de forma hierárquica, partindo dos recursos de baixa sensibilidade até os recursos de alta sensibilidade [68].

Esse tipo de mecanismo é altamente eficiente para lidar com certos ataques do tipo Cavalo de Troia (do inglês *Trojan Horse* ou apenas *Trojan*), visto que esse mecanismo utiliza uma estrutura de dados específica para lidar com as permissões: um grafo direcionado. Ataques desse tipo requerem acesso aos recursos de modo acíclico, algo que não pode ser concretizado em um sistema MAC, que define regras hierárquicas de acesso [68] utilizando o grafo direcionado. Tentativas de acesso que entrem em conflito com a estrutura do grafo são prontamente rejeitadas. O uso de MAC entretanto, possui algumas desvantagens. Para que o uso desse mecanismo seja eficiente, é necessário que as regras de acesso sejam bem planejadas e que seja feito frequente monitoramento das mesmas, para mantê-las sempre em dia com os requisitos do sistema [7]. Uma alternativa que exige menos complexidade é o mecanismo de Controle de Acesso Discricionário.

3.3.2 Controle de Acesso Discricionário

Controle de Acesso Discricionário (do inglês *Discretionary Access Control* ou DAC) é um mecanismo de autorização que controla permissões a partir de usuários de dados . Em um sistema DAC, o controle de acesso aos recursos são averiguados no momento da autenticação, utilizando autenticação por senha, como descrito na Seção 3.1.1 [7]. Em um sistema DAC, o dono do objeto, usualmente seu criador, tem acesso discricionário, ou irrestrito, a um determinado recurso. Ou seja, a principal política do DAC é administração de recursos baseado em donos [69]. Um sistema DAC também pode ser classificado em três tipos [69]:

- DAC restrito;
- DAC liberal;
- DAC com mudança de posse.

Em um sistema de DAC restrito, a posse de um recurso é do criador do recurso, e essa posse não pode ser transferida para outros usuários. Já em um sistema DAC liberal, um usuário pode delegar a mudança de posse de um usuário para o outro. Por fim, um

sistema DAC com mudança de posse permite que um usuário transmita a posse de um objeto a outro usuário [69];

Devido ao esquema de permissão baseado na posse do recurso, O DAC se mostra mais flexível que o MAC, que necessita da intervenção direta do monitor ou administrador do serviço [68]. Entretanto, o DAC se mostra menos seguro que o MAC, visto que tal sistema é suscetível a ataques do tipo Cavalo de Troia [7]. Outra desvantagem do DAC é que ele permite que usuários definam políticas de acesso aos recursos de forma global, o que dificulta consistência nesse tipo de sistema [8], dado que cada usuário pode definir suas próprias permissões. Uma alternativa a essa granularidade é o mecanismo de Controle de Acesso Baseado em Papéis.

3.3.3 Controle de Acesso Baseado em Papeis

Controle de Acesso Baseado em Papeis (do inglês *Role Based Access Control* ou RBAC) é um mecanismo de autorização e controle de acesso baseado em papéis e privilégios de usuário [7]. Papéis são abstrações de comportamentos de usuário e suas responsabilidades associadas, de modo a associar permissões a recursos a grupos e seus respectivos membros [8]. Permissões de usuário são dadas por diferentes parâmetros do RBAC, como papeis de usuário (*user-roles*), permissões de papel (*role-permissions*) e relacionamentos entre papeis (*role-role relationships*).

O RBAC surge como uma alternativa para combinar o rígido esquema de permissões do MAC com a flexibilidade de autorizações explícitas do DAC [8]. Dessa forma, o RBAC limita o acesso a recursos de acordo com o papel onde um usuário ou aplicação está inserido. A Figura 4 exemplifica a relação entre usuários, papeis e recursos.

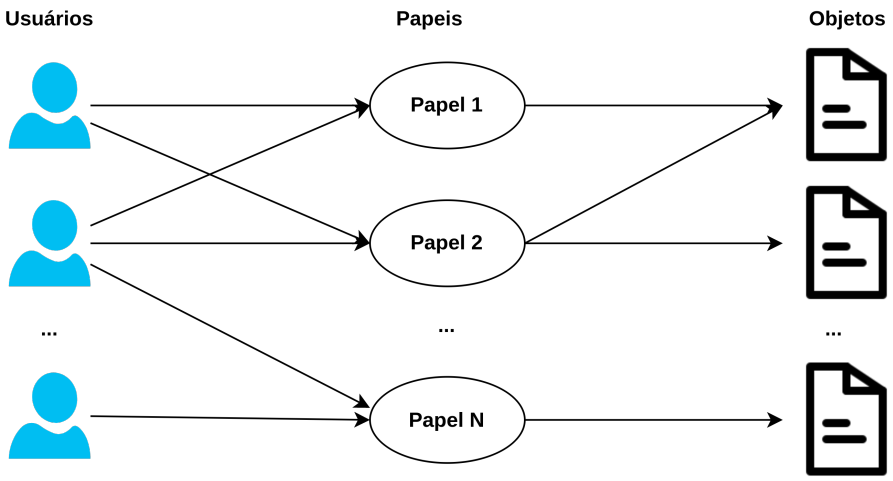

Figura 3.3: Relacionamento entre usuários, papeis e recursos no mecanismo RBAC, adaptado de [8].

O RBAC oferece dois tipos de papeis [7]:

- Papel de aplicação ou técnico;
- Papel organizacional ou técnico.

O papel de aplicação contém a combinação de diferentes atributos técnicos, e seu escopo é limitado a uma aplicação específica. Já o papel organizacional é gerado a partir de diferentes funções e direitos de acesso atribuídos a um usuário. Um papel organizacional é gerado a partir diferentes funções e direitos de acesso de um usuário [7].

O RBAC oferece uma série de vantagens em relação aos modelos anteriores, entre elas a independência entre usuário e permissão, dado que é possível associar usuários a papeis e papeis a objetos [8]. Além disso, é possível criar papeis hierárquicos, de modo a gerar estruturas mais complexas. Entretanto, devido a necessidade de associar usuários a grupos, muitas vezes o RBAC não é adequado para ambientes dinâmicos, onde as permissões são atribuídas em tempo real [7], dado que as atribuição de papeis é feita de forma estática [8]. Portanto, quando surge a necessidade dessa flexibilidade, um novo método se mostra mais apropriado, o Controle de Acesso Baseado em Atributos.

3.3.4 Controle de Acesso Baseado em Atributos

Controle de Acesso Baseado em Atributos (do inglês *Attribute Based Access Control* ou ABAC) é um mecanismo de controle de acesso que define as regras de controle de acesso por meio de políticas e atributos [7]. Tais políticas associam sujeitos a objetos. Sujeitos são atributos estáticos, como nome de usuário ou designação dentro de uma organização [8]. Objetos são metadados associados a um recurso, como por exemplo o assunto de um documento. Dessa forma, O ABAC utiliza essas associações de sujeitos e objetos para definir permissões de usuário [8].

O ABAC se mostra mais vantajoso que o RBAC para casos onde é necessário oferecer flexibilidade na atribuição de permissões, e evita a necessidade de associar usuários a papeis de modo a atribuir as permissões do mesmo ao usuário. Ao invés disso, é possível, são associados atributos ao usuário, que controlam de forma mais granular o acesso a um determinado recurso [8]. Além disso, no ABAC os privilégios de usuário são definidos previamente, o que evita problemas de autorização e oferece mais flexibilidade ao modelo [7]. Entretanto, devido à homogeneidade de utilizar os mesmos atributos para todos, é necessário manter um banco de dados central com as informações relativas a todos os atributos, o que pode gerar um único ponto de falha em um sistema ABAC [8].

Risco	No de CWEs	% de Aplicações Afetadas
Controle de Acesso Quebrado	34	55.97
Falhas Criptográficas	29	46.44
Injeção	33	19.09
Design inseguro	40	24.19
Má Configuração de Segurança	20	19.84
Componentes Desatualizados e Vulneráveis	3	27.96
Falhas de Identificação e Autenticação	22	14.84
Falhas de Software e Integridade de Dados	10	16.67
Falhas e Monitoramento	4	19.23
Cross Site Request Forgery (CSRF)	1	2.72

Tabela 3.1: 10 riscos de segurança mais comuns em aplicações web, adaptado de [9].

3.4 Segurança em Controle de Acesso

Como verificado nas seções anteriores, controle de acesso é um processo altamente complexo e que exige diversos processos de modo a funcionar de forma eficiente. Entretanto, falhas em controle de acesso e autenticação são duas das maiores vulnerabilidades de acordo com o relatório da Fundação Open Web Application Security Project (OWASP) [9]. A Fundação OWASP apresenta os dez maiores riscos de segurança para aplicações web, de acordo com fatores que incluem prevalência da vulnerabilidade, detectabilidade, explorabilidade e impacto técnico [70]. A Tabela 1 apresenta esses riscos, ordenados de acordo com os fatores citados anteriormente [9], assim como o número de Enumerações de Fraquezas Comuns (do inglês *Common Weakness Enumerations* ou CWEs) e a taxa de incidência dos riscos de segurança em aplicações.

Como pode ser verificado, falhas em controle de acesso e autenticação afetam 55.97% e 14.84% das aplicações web [9], com quantidades consideráveis de CWEs relacionados a cada um. A Fundação OWASP fornece soluções para cada um dos problemas. No escopo de controle de acesso quebrado, é sugerido [71]:

1. Negar acesso por padrão;
2. Implementar mecanismos de controle de acesso apenas uma vez e reusá-lo;
3. Mecanismos de controle de acesso devem reforçar posse dos recursos, ao invés de permitir criação, atualização ou remoção de recursos por usuários;
4. Requisitos de limite de negócios de aplicações exclusivas devem ser impostos por modelos de domínio;

5. Desabilitar listagem de diretórios web e garantir que metadados não se encontram na raiz de sistemas de arquivo;
6. Monitorar falhas de acesso a serviços e alertar administradores quando apropriado;
7. Limitar o acesso repetido (*rate limit*) a APIs para evitar ataques automatizados;
8. Identificadores de sessão devem ser removidos após logout no serviço ou aplicação.

Também são apresentadas soluções para evitar falhas de identificação e autenticação, que incluem [72]:

1. Implementar autenticação de dois fatores (2FA);
2. Não implantar credenciais padrão;
3. Implementar verificações de senhas fracas;
4. Alinhar complexidade e tamanho de senhas com os valores recomendados pelas políticas definidas pelo padrão 800-63b do NIST [73];
5. Verificar que cadastro, recuperação de credenciais e rotas de API estão protegidas contra ataques de enumeração de senha;
6. Limitar ou impedir repetidas tentativas mal-sucedidas de acesso a aplicações;
7. Utilizar um gestor de sessão junto ao servidor da aplicação, que gere um identificador de alta entropia para cada sessão de usuário.

Além dessas duas categorias, falhas criptográficas são responsáveis por 29 CWEs e afetam 46.44% das aplicações [9]. Essas falhas estão relacionadas principalmente ao uso de senhas padrão e algoritmos criptográficos defeituosos ou que apresentam riscos de segurança [74]. Algumas soluções para esses problemas são [74]:

1. Nao armazenzar dados não necessários para o funcionamento da aplicação;
2. Criptografar dados em trânsito com protocolos como Transport Layer Security;
3. Não usar protocolos antigos como Simple Mail Transfer Protocol (SMTP) ou File Transfer Protocol (FTP).

3.5 Controle de Acesso em Nuvem Pública

Gestão de usuários é uma parte integrante de todo provedor de nuvem. Em um ambiente de nuvem essa gestão é feita pelo provedor, e os dados de usuário se encontram em

posse do mesmo, em contraste com ambientes *on-premises*, nos quais os dados estão em posse do prestador de serviço [7]. Dessa forma, é necessário que o provedor mantenha e trafegue esses dados de forma segura. Entretanto, gerir credenciais e permissões para acesso a recursos é um problema complexo enfrentado por muitas organizações [75]. Além disso, regulações sobre o armazenamento e uso de dados têm se tornado cada vez mais prevalentes, o que levou desenvolvedores e gestores de sistema a adotar permissões cada vez mais granulares para seus sistemas [75].

Essa complexidade ocasionou no surgimento do que hoje é conhecido como sistema de Gestão de Acesso e Identidade (do inglês *Identity and Access Management* ou IAM).Um serviço de IAM é responsável por permitir acesso seguro por parte de usuários, além de gestão de usuários, verificação de credenciais e checagem de permissões, de modo a assegurar que apenas usuários e aplicações autorizados tenham acesso a um recurso [75]. Além disso, sistemas de IAM funcionam de forma a verificar que a mesma identidade está sendo utilizada em diferentes sistemas geridos por eles [7].

Atualmente, todos os provedores de nuvem citados na Seção 2.2 oferecem serviços de IAM, de modo a permitir que usuários utilizem uma ferramenta única para gestão de acesso em todos os serviços do provedor. Alguns dos serviços de IAM oferecidos pelos provedores são:

- AWS Identity and Access Management [76];
- Google Cloud Platform Identity and Access Management [77];
- Azure Active Directory [32];
- Alibaba Cloud Resource Access Management (RAM) [78];
- IBM Cloud Identity and Access Management Solutions [79];
- Oracle Identity Cloud Service [80];
- Tencent Cloud Access Management (CAM) [81].

Cada provedor de nuvem oferece diversos métodos de autenticação e autorização em seus serviços de IAM. As Tabelas 3.2 e 3.3 apresentam os mecanismos de autenticação e autorização disponibilizados pelos principais provedores de nuvem, onde recursos marcados por asterisco representam mecanismos oferecidos por outros serviços do provedor.

Serviços de IAM, entretanto, são próprios dos provedores que os implementam, e as especificidades de cada um pode ocasionar em *vendor lock-in*. Portanto, muitas vezes um administrador de sistema ou desenvolvedor pode desejar utilizar uma ferramenta própria, e hospedá-la em sua infraestrutura ou em ambientes de IaaS.

Provedor	**Autenticação por senha**	**Chaves criptográficas**	**2FA**	**SSO**
AWS	Sim	Sim*	Sim	Sim
GCP	Sim	Sim*	Sim	Sim
Azure	Sim	Sim*	Sim	Sim
Alibaba Cloud	Sim	Sim*	Sim	Sim
Oracle Cloud	Sim	Sim*	Sim	Sim
IBM Cloud	Sim	Sim*	Sim	Sim
Tencent Cloud	Sim	Sim*	Não	Sim

Tabela 3.2: Mecanismos de autenticação oferecidos pelos sistemas de IAM dos provedores de nuvem pública.

Provedor	MAC	DAC	RBAC	ABAC
AWS	Não	Não	Sim	Sim
GCP	Não	Não	Sim	Não
Azure	Não	Não	Sim	Sim
Alibaba Cloud	Sim*	Sim*	Sim	Sim
Oracle Cloud	Não	Não	Sim*	Não
IBM Cloud	Não	Não	Sim	Sim
Tencent Cloud	Não	Não	Sim*	Não

Tabela 3.3: Mecanismos de autorização oferecidos pelos sistemas de IAM dos provedores de nuvem pública.

3.5.1 Ferramentas de IAM

Como apresentado na seção anterior, ferramentas de IAM oferecem inúmeras utilidades para gestão e autenticação de usuários em ambientes de nuvem. Entretanto, como apresentado anteriormente, ferramentas de IAM oferecidas pelos provedores não costumam ser interoperáveis entre si. Devido à essa dificuldade de realizar autenticação e gestão de usuários entre diferentes provedores de nuvem, uma alternativa é utilizar uma ferramenta de IAM que não esteja associada diretamente a um provedor.

Atualmente, existem diversas ferramentas de IAM de código aberto, e que podem ser utilizadas para trazer interoperabilidade no processo de autenticação. Algumas dessas ferramentas são Shibboleth [82], WSO2 Identity Server [83] e Keycloak [84]. O Shibboleth [82] é um sistema de acesso que utiliza SSO para redes de computadores e aplicações web, utilizando o mecanismo SAML para realizar gestão de identidade [85]. A arquitetura do Shibboleth define interações entre um IdP e um provedor de serviço para facilitar autenticação por SSO em navegadores web [85].

O WSO2 Identity Server é um servidor de IAM de código aberto que oferece diversas soluções de autenticação, sendo a maioria soluções de SSO [86]. Tais soluções são dis-

Característica	Shibboleth	WSO2	Keycloak
Código aberto	Sim	Sim	Sim
SAML	Sim	Sim	Sim
OpenID Connect	Não	Sim	Sim
Interface com provedor de serviço	Não	Sim	Sim
Middleware	Apache Tomcat	WSO2 Carbon	WildFly
Fácil instalação	Sim	Não	Sim
Fácil configuração	Não	Não	Sim
Interface gráfica moderna e amigável	Não	Não	Sim

Tabela 3.4: Características de cada ferramenta de IAM de código aberto, adaptada de [10].

Mecanismo	Shibboleth	WSO2	Keycloak
MAC	Não	Não	Não
DAC	Não	Não	Não
RBAC	Não	Sim	Sim
ABAC	Não	Sim	Sim

Tabela 3.5: Mecanismos de autenticação oferecidos por cada ferramenta de IAM.

ponibilizadas por meio de interfaces web, de forma a facilitar a configuração de IdPs e a gestão de usuários [86].

O Keycloak é uma ferramenta de IAM de código aberto [87], que oferece diversas soluções de autenticação, utilizando os métodos de SSO citados na Seção 3.2. Um administrador de sistemas pode hospedar sua própria instância do Keycloak e realizar a gestão e autenticação de usuários para aplicações web de uma determinada organização ou entidade. Um dos elementos principais do Keycloak são os chamados *realms*. Um realm do Keycloak gerencia um conjunto de usuários, credenciais, papeis e grupos, de modo que cada *realm* é separado dos outros, podendo apenas gerenciar usuários e grupos pertencentes a ele [88].

Cada ferramenta de IAM oferece características específicas, voltadas para certos tipos de aplicações. A Tabela 3.4 apresenta um comparativo quanto às unidades funcionais de cada ferramenta, e a Tabela 3.5 apresenta quais mecanismos de autorização são oferecidos por cada ferramenta.

3.6 Considerações Finais

Neste capítulo foram abordadas as definições de autenticação, além dos mecanismos físicos e digitais utilizados para autenticação. Além disso, foi abordado em detalhe o método

de *Single Sign-On*, um mecanismo de autenticação amplamente utilizado em aplicações web. Além disso, foram abordados os mecanismos mais comuns utilizados para oferecer autorização e controle de acesso a recursos na web, tendo sido apresentadosos quatro mecanismos de autorização mais utilizados, além de vantagens e desvantagens de cada um. Por fim, foram apresentados os empecilhos para gestão de usuário em um ambiente de nuvem pública, assim como os mecanismos criados para lidar com essas complexidades, as ferramentas de IAM. Também foram apresentadas as diferentes ferramentas de IAM de cada provedor e suas características quanto à oferta de diferentes mecanismos de autenticação apresentados.

No próximo capítulo será apresentado um framework de controle de acesso para ambientes de Função como um Serviço (FaaS), proposto nesse trabalho, que permite autenticar e autorizar usuários em um ambientes de FaaS utilizando infraestrutura própria.

Capítulo 4

Framework de Controle de Acesso

Como apresentado em seções anteriores, controle de acesso é um componente essencial para acesso seguro a ambientes de nuvem. Assim, verificou-se que ferramentas de controle de acesso, as chamadas ferramentas de IAM, permitem que um provedor de nuvem limite o acesso de usuários aos serviços oferecidos, de acordo com a identidade e as permissões do mesmo. Entretanto, como levantado na Seção 3.5, as ferramentas de IAM são particulares dos provedores de nuvem nos quais são oferecidos, e utilizar uma ferramenta de IAM pode ocasionar no chamado *vendor lock-in*, impedindo que um usuário utilize o mesmo serviço de IAM em outros provedores.

Este capítulo apresenta um *framework* de controle de acesso que pode ser utilizado para acessar serviços de Função como um Serviço em diversos provedores de nuvem. Este *framework* permite que um usuário execute uma função preexistente em um ambiente de FaaS, após passar pelo processo de controle de acesso, no qual são verificadas suas credenciais e suas permissões. As próximas seções irão tratar de uma descrição do *framework* proposto, além das funcionalidades implementadas e integrações com ferramentas de controle de acesso citadas na Seção 3.5.1.

4.1 Arquitetura do Framework

O *framework* de controle de acesso consiste de uma API RESTful, que permite que um usuário seja autenticado junto a uma ferramenta de IAM, na qual são verificadas suas credenciais e as permissões de acesso a um ambiente de FaaS pré-configurado. A API utiliza requisições HTTP para gerar um *token* OAuth válido, o qual permita assegurar que o usuário possui as devidas permissões para executar a função no ambiente de FaaS. O fluxo de execução do *framework* se dá por meio dos seguintes passos:

1. O usuário requisita autenticação junto ao *framework* de controle de acesso;

2. O *framework* requisita acesso à ferramenta de IAM, passando as credenciais do usuário;
3. A ferramenta de IAM retorna um *token* de acesso para o usuário;
4. O usuário requisita a execução de uma função no ambiente de FaaS, passando o *token* recebido anteriormente como um cabeçalho na requisição HTTP;
5. O *framework* utiliza o *token* recebido para realizar uma requisição HTTP para a função com as devidas permissões;
6. O ambiente de nuvem verifica as permissões passadas no *token* recebido, e envia um evento de execução para a função requisitada;
7. Após a função finalizar sua execução, o resultado é retornado para o *framework*. O resultado deve ser armazenado em um objeto JSON;
8. O resultado da função é repassado do *framework* para o usuário.

Dessa forma, é possível configurar o ambiente de FaaS no provedor de nuvem pública para aceitar apenas solicitações que definam um determinado cabeçalho recebido no *token* OAuth. Isso impede acesso vindo de clientes não autorizados. Para fins de implementação, a ferramenta de IAM selecionada foi o Keycloak [84], uma ferramenta de IAM de código aberto, descrito na Seção 3.5.1. Essa ferramenta foi selecionada por oferecer autenticação utilizando OpenID Connect e tokens OAuth, e autorização por meio de RBAC. O uso de RBAC permite mais facilidade na definição de papeis com permissão de acesso ao ambiente de FaaS. A Figura 4.1 ilustra o fluxo utilizado pelo *framework* para executar funções de forma segura.

O Keycloak é configurado de forma a utilizar apenas um *realm* [88], criado para fins de validação da API. Um *realm* é uma estrutura do Keycloak que mantém registro dos usuários, clientes, papeis e permissões necessários para funcionamento da API. O *realm* deve ser criado antes da execução do *framework*. Assim sendo, no *realm* é criado um cliente, que é uma entidade que pode requisitar autenticação de usuários por parte do Keycloak [88]. Esse cliente define as permissões dos usuários criados para este *realm* no padrão definido no mecanismo OpenID Connect, descrito na Seção 3.2.3.

As permissões do cliente são definidas utilizando o formato RBAC, descrito na Seção 3.3.4, e são definidos papeis, recursos e escopos para execução de funções no ambiente de FaaS. Para fins de validação do modelo foram definidos apenas um papel, um recurso e um escopo para execução de funções, como apresentado a seguir:

- Papel "*runner*": papel atribuído a usuários que tem permissão de execução de funções no ambiente de FaaS;

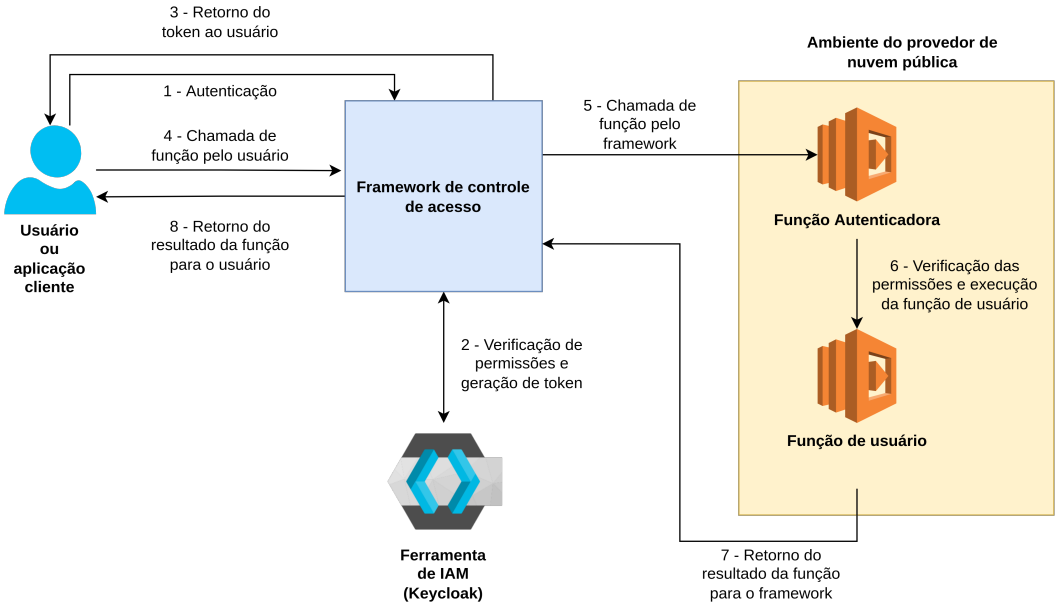

Figura 4.1: Fluxo de execução do framework de controle de acesso.

- Recurso "*res:function*": recurso que representa um mapeamento das funções do ambiente de FaaS dentro do cliente;
- Escopo "*scopes:execute*": permissão de execução para o recurso "*res:function*", que é atribuído ao papel "*runner*".

Dessa forma, a política RBAC acima pode ser interpretada como um papel "*runner*", que possui permissão de execução ("*execute*") em uma função (recurso "function"). Essa permissão é convertida em um *token* de acesso, que é enviado na requisição HTTP de execução da função. Essa requisição foi descrita na Seção 4.2.2. A política é então associada a usuários que possuem permissão de execução de funções em FaaS, e isso é definido pelo administrador da ferramenta de IAM.

Este trabalho utilizou como provedor de nuvem o Amazon Web Services [16]. O ambiente de FaaS deste provedor, o AWS Lambda, foi configurado utilizando um Autorizador Lambda (do inglês *Lambda Authorizer*) [89], um componente do serviço API Gateway da AWS [90] que permite controlar acesso a APIs RESTful da AWS por meio de funções do AWS Lambda [89]. O autorizador utilizado é um Autorizador Lambda baseado em *token*, no qual o API Gateway recebe um *token* do formato JSON Web Token (JWT) contendo informações necessárias para permitir ou negar acesso a uma função. Esse *token* corresponde ao *token* de acesso gerado com as permissões definidas anteriormente na política de autorização da ferramenta de IAM. A função autorizadora foi escrita na linguagem Javascript, uma das linguagens disponíveis no AWS Lambda, a qual realiza os seguintes passos:

1. Decifra o *token* recebido como *string*, codificada em formato base 64;
2. Converte o *token* para um objeto JavaScript [91], que é equivalente a um *hashmap* em outras linguagens;
3. Verifica que o objeto possui a chave "*access*", que corresponde ao papel definido na ferramenta de IAM. Caso tenha a chave, verifica o valor dessa variável. Caso contrário, a função gera um erro de validação, que é retornado ao usuário;
4. Caso o valor da variável "*access*"seja "*runner*", o papel de executor de funções definido, uma política de IAM de sucesso é gerada e então encaminhada para o AWS Lambda; caso contrário, o autorizador gera uma política de erro.

A Figura 4.2 exemplifica o fluxo de execução da função autorizadora, de acordo com os passos citados anteriormente. Após o fim da execução, o API Gateway gera uma política de IAM que consiste de uma chamada para o AWS Lambda. Uma política IAM do AWS é um objeto JSON, com o formato apresentado na Listagem 4.1. Nesta listagem o campo <efeito> pode ser "Allow"ou "Deny", de acordo com a validação da função, que pode permitir ou negar acesso à função de usuário, de acordo com as permissões recebidas no objeto. Caso o acesso seja permitido, o autorizador Lambda retorna a política de IAM de sucesso com o identificador do recurso definido no campo <recurso>. Esse identificador corresponde a um identificador global da função de usuário requisitada pelo *framework*, que é então executada.

Listagem 4.1: Política de IAM gerada pelo autorizador.

```
{
    "principalId":"user",
    "policyDocument":{
        "Version":"2022-04-17",
        "Statement":[
            {
                "Action":"execute-api:Invoke",
                "Effect": <efeito>,
                "Resource":<recurso>
            }
        ]
    }
}
```

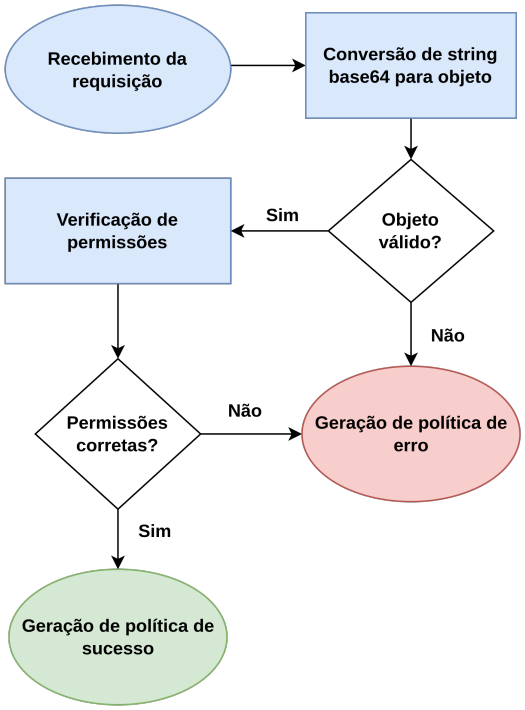

Figura 4.2: Fluxograma de execução do autorizador Lambda.

4.2 Implementação do Framework

O *framework* de controle de acesso foi implementado em um programa composto de diversos módulos acoplados. Cada componente do *framework* foi escrito na linguagem de programação Python [92], que será apresentada em detalhe na seção seguinte.

4.2.1 A Linguagem de Programação Python

A linguagem Python [92] é uma linguagem de programação interpretada de propósito geral. Foi criada por Guido von Rossum em 1991, em sua versão 0.9.0, e recebeu diversas atualizações desde então [93]. A linguagem Python é dinamicamente tipada com coletor de lixo (do inglês *garbage collector*), que suporta paradigmas de programação procedural, orientado a objetos e funcional.

A linguagem tem diversas aplicações, e pode ser utilizada para escrita de *scripts*, aplicações web e ferramentas voltadas para inteligência artificial e ciência de dados. No campo de aplicações web se destacam alguns *frameworks* altamente utilizados: o Flask [94], uma biblioteca para desenvolvimento rápido de APIs RESTful; e o Django [95], um *framework* web de alto nível, que incentiva desenvolvimento rápido e *design* pragmático [95]. Além desses dois grandes frameworks, outros concorrentes surgiram nos últimos anos, em especial o FastAPI [96], um framework web para construção de APIs em Python.

Atualmente, a linguagem Python é a linguagem de programação mais popular de acordo com o índice da Comunidade de Programação da TIOBE, uma companhia que é referência em avaliações de software [97]. O *ranking* avalia linguagens de programacão de considerável relevância para a comunidade.

4.2.2 Estrutura do Código

O *framework* de controle de acesso foi desenvolvido na linguagem Python [92]. Assim, para criação da API RESTful foi utilizada a biblioteca FastAPI [96], que permite implementação de rotas assíncronas em Python. Essa ferramenta foi selecionada em detrimento de outras como Flask [94] ou Django [95] por oferecer uma interface de API interativa, que permite testar as rotas definidas por meio de uma interface web, sem a necessidade de utilizar ferramentas externas. Em adição a essa biblioteca, foi utilizada a ferramenta Uvicorn [98] como servidor web, de modo a permitir o tratamento de diversas requisições simultâneas pelo *framework*. Essa ferramenta foi escolhida por integrar de forma simples com ferramentas relacionadas ao Python, como o FastAPI.

O *framework* é composto de duas partes principais, as quais são o arquivo de definição da API e os módulos de comunicação com o Keycloak e com o AWS Lambda. O arquivo principal define as rotas da API, e os outros módulos são utilizados quando são feitas requisições HTTP. Assim sendo, a Figura 4.3 apresenta a estrutura do framework, assim como as conexões realizadas entre módulos e os arquivos utilizados para criptografar a comunicação.

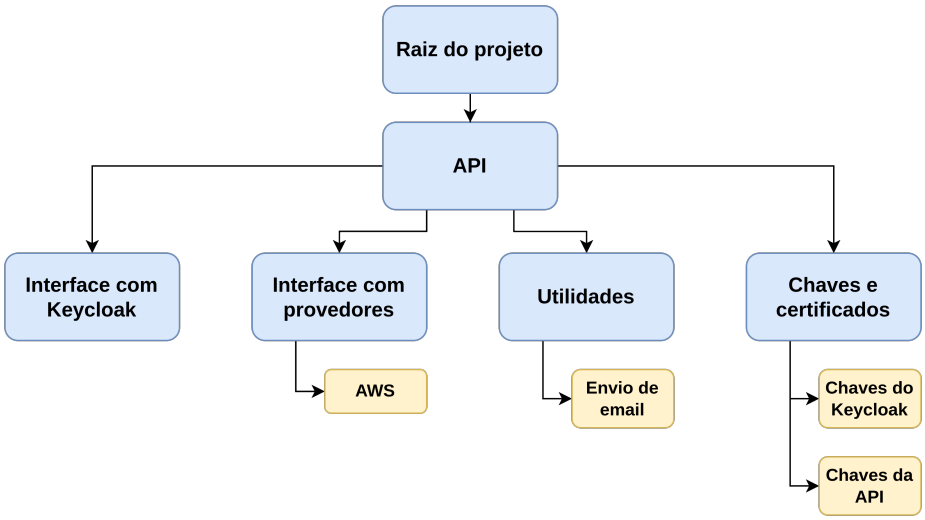

Figura 4.3: Estrutura do sistema de arquivos do *framework*.

Além disso, a API define quatro rotas principais:

- {realm}/user/new: cria um usuário no Keycloak, passando o *realm* de destino na variável {realm}. O usuário deve fornecer nome de usuário, email, primeiro nome e último nome no corpo da requisição. Após o usuário do Keycloak ser criado, o mesmo recebe um email para alterar sua senha junto à ferramenta de IAM. A função da rota recebe a *string* que representa o identificador de *realm* do Keycloak, assim como a variável *user*, uma instância da classe *User*, uma *model* que representa o usuário do *framework* de controle de acesso. A Listagem 4.2 apresenta o código relativo à função;

Listagem 4.2: Função de criação de usuário.

```
async def create_user(realm: str, user: User) -> dict:
    try:
        keycloak = Keycloak(keycloak_endpoint=KEYCLOAK_BASE_URL)
        response = keycloak.create_user(user, realm)
        full_name = f'{user.first_name} {user.last_name}'
        email.send_update_password_email(full_name,user.username, realm,
            user.email)
        response = True
        message = {
            'message': response
        }
    except Exception as e:
        message = {
            'message': str(e)
        }
    finally:
        return message
```

- {realm}/authorization/get: gera uma URL de autenticação que será passada ao usuário ou aplicação cliente, a qual deve ser acessada em um navegador para requisitar autenticação do usuário na ferramenta de IAM. A rota recebe como parâmetro o identificador do *realm* do Keycloak, e retorna um objeto JSON contendo a URL para geração de um *token* OAuth. A Listagem 4.3 apresenta o código relativo à rota;

Listagem 4.3: Função de geração de *token* OAuth.

```
async def get_authorization_url(realm: str) -> str:
    try:
```

```
        keycloak = Keycloak(keycloak_endpoint=KEYCLOAK_BASE_URL, realm=
            realm)

        # URL de autorizacao
        redirect_uri = f'{API_BASE_URL}/{realm}/authorization/token'
        authorization_url = keycloak.generate_oauth2_authorization_url(
            realm, redirect_uri=redirect_uri)
        message = {
            'message': authorization_url
        }
    except Exception as e:
        message = {
            'message': str(e)
        }
    finally:
        return RedirectResponse(message['message'])
```

- {realm}/authorization/token: retorna ao usuário um *token* OAuth codificado em um objeto do formato JSON Web Token (JWT), contendo as permissões de usuário. A função recebe como parâmetro o código de autenticação gerado pela rota de geração de URL, após o usuário realizar o processo de autenticação e autorização junto à ferramenta de IAM. Essa função então retorna um objeto JSON contendo o *token* OAuth codificado no formato JWT. A Listagem 4.4 apresenta o código relativo à rota;

Listagem 4.4: Função de geração de *token* OAuth.

```
async def get_bearer_token(realm: str, code: str) -> str:
    try:
        keycloak = Keycloak(keycloak_endpoint=KEYCLOAK_BASE_URL, realm=
            realm)
        authentication_code = code

        # Autentica o usu rio e gera um token de autoriza  o
        redirect_uri = f'{API_BASE_URL}/{realm}/authorization/token'
        token = keycloak.get_token(realm, authentication_code,
            redirect_uri=redirect_uri)

        message = {
            'message': f'Bearer␣{token}'
        }
    except Exception as e:
        message = {
```

```
            'message': str(e)
        }
    finally:
        return message
```

- {realm}/function/exec: executa uma função do AWS Lambda. A URL pública da função é passada como parâmetro, junto ao cabeçalho de autorização contendo o *token* OAuth. Caso a função execute com sucesso, a mensagem de retorno da função é devolvida ao usuário. Caso ocorra um erro, a mensagem de erro é retornada ao usuário, junto ao código de erro recebido. A Listagem 4.5 apresenta o código relativo à rota.

Listagem 4.5: Função de execução da função no ambiente de FaaS.

```
async def call_function(realm: str, function: Function,response:
   Response, authorization: str = Header(None)) -> dict:
    try:
        if not authorization:
            raise HTTPException(status_code=403, detail='No␣bearer␣token
               ␣found')

        aws = AWSClient()
        response = aws.call_lambda_function(function.function_url,
           authorization)
        message = {
            'message': {
                'provider_response_code': 200,
                'provider_response': response
            }
        }
        return message
    except HTTPException as e:
        message = {
            'message': {
              'provider_response_code': e.status_code,
              'provider_response': e.detail
            }
        }
        response.status_code = 400 # C digo de erro geral para erros do
            provedor
        return message
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Além das rotas principais descritas anteriormente, a API define uma rota /realm/authorization/debug, que funciona da mesma forma da rota /{realm}/authorization/get, e foi adicionada para simular a autenticação de usuário de forma automatizada, para fins de teste. Além disso, foi criada uma rota /health, utilizada para fins de verificação da saúde do *framework*. Também são definidos quatro componentes necessários para as conexões do *framework*, descritos a seguir.

- Interface com Keycloak: define a classe Keycloak, que realiza a conexão entre a API e o servidor Keycloak. Essa classe define métodos para criação de usuário e requisições para a API RESTful disponibilizada pelo Keycloak;
- Interface com provedores: define as funções que realizam a conexão com os provedores de nuvem. Atualmente, apenas as conexões com o AWS Lambda estão implementadas, em uma classe *AWSClient*, que define o método necessário para execução das funções presentes no AWS Lambda. O método recebe como parâmetros a URL pública da função e o *token* necessário para autorização pelo autorizador Lambda, descrito na Seção 4.1. A Listagem 4.6 apresenta o código da função utilizada para executar funções no ambiente de FaaS por meio de requisições HTTP;

Listagem 4.6: Método de execução de funções no AWS.

```
def call_lambda_function(self, lambda_uri: str, token: str) -> str:
    try:
        req = request.Request(lambda_uri)
        req.add_header('authorizationToken', token)

        with request.urlopen(req) as response:
            response_data = response.read().decode('utf-8')
            if response_data:
                return response_data
    except HTTPError as error:
        raise HTTPException(status_code=error.code)
    except Exception as e:
        logging.debug(e)
        raise HTTPException(status_code=403)
```

- Utilidades: define funções utilitárias necessárias para o funcionamento do processo de controle de acesso. Atualmente, apenas a função de envio de email, após cadastro de usuário, está implementada. A função recebe os parâmetros necessários para montagem e envio do email para o usuário, e envia uma mensagem criptografada utilizando as funcionalidades de criptografia do Python. A Listagem 4.7 apresenta o código relativo ao envio de emails após cadastro de usuário;

Listagem 4.7: Função de envio de email para usuários do *framework*.

```
def send_update_password_email(full_name: str, username: str, realm: str
    , recipient_email: str) -> bool:
    # Get email message
    message = get_reset_password_message(full_name, username, realm)

    # Send message to recipient
    response = send_email(message, recipient_email)
    if response:
        return True
    else:
        raise Exception("Could not send update password email")
```

- Chaves e certificados: armazena chaves e certificados SSH, descritos na Seção 3.1.3, para permitir comunicação segura entre o usuário e a API e entre API e o Keycloak. Não é necessário prover chaves para comunicação com os provedores de nuvem, visto que os servidores dos mesmos já são configurados utilizando HTTPS e TLS.

4.3 Considerações Finais

Neste capítulo foi apresentado o framework de controle de acesso para ambientes de FaaS, proposto neste trabalho. Ele foi desenvolvido utilizando a linguagem de programação Python, junto à ferramenta de IAM Keycloak, de modo a executar funções no AWS Lambda, o ambiente de função como um serviço da AWS. O *framework* foi apresentado em detalhe, juntamente com o fluxo de execução e a estrutura do sistema de arquivos que compõe a ferramenta.

No próximo capítulo serão apresentados os experimentos realizados para validar a segurança do *framework* proposto. Para isso, os testes realizados serão apresentados, de acordo com as vulnerabilidades apresentadas no relatório anual da OWASP [9], descrito na Seção 3.4.

Capítulo 5

Resultados

Como apresentado na Seção 3.4, duas das dez maiores vulnerabilidades em aplicações web estão relacionadas a falhas de identificação e problemas em controle de acesso [9]. Assim sendo, o *framework* desenvolvido neste trabalho se propõe a solucionar esses dois problemas. A Tabela 5.1 apresenta os problemas solucionados por meio do *framework* proposto, como apresentado na Seção 3.4.

O uso da ferramenta de IAM em infraestrutura *on-premises* ou hospedada em um serviço de IaaS oferece uma solução de implantação e reuso do mecanismo de controle de acesso, como recomendado no Tópico 2 das recomendações de segurança para controle de acesso da OWASP [9]. Já o uso de RBAC apresenta uma solução para definir a posse de recursos, como recomendado no Tópico 3, dado que a posse e o acesso aos recursos estão limitados aos usuários associados a um papel. Por fim, são propostas soluções para o problema de falhas criptográficas. O uso do protocolo TLS apresenta um solução de criptografia de dados em trânsito. Além disso, é utilizado HSTS e redirecionamento para páginas com Hypertext Transfer Protocol Secure (HTTPS) como solução de obrigatoriedade de criptografia.

5.1 Metodologia dos Testes

Para verificar a eficácia do uso de criptografia entre as partes, de modo a tornar efetivo o uso de HSTS, foram realizados testes de requisição entre o usuário e a API, e entre a API e a ferramenta de IAM. Esses testes buscaram verificar que a comunicação entre as partes ocorre sem vazamento de dados sensíveis, como credenciais de usuário ou *tokens* de acesso ao ambiente de FaaS. Dessa forma, é possível verificar se um atacante, que tem acesso à rede onde o *framework* se encontra, consegue capturar dados de usuário. Para verificar este caso, foi utilizada a ferramenta Apache JMeter [99], uma ferramenta para realização de testes funcionais desenvolvida e mantida pela Apache Software Foundation [99]. Essa

Problema	**Soluções**
Controle de acesso quebrado	- Implantação e reuso de um único mecanismo de controle de acesso; - Uso de RBAC para reforçar a posse dos recursos utilizando papeis.
Falhas de identificação e autenticação	- Obrigatoriedade da mundança de credenciais padrão de acesso ao *framework*.
Falhas criptográficas	- Criptografar dados em trânsito utilizando protocolos seguros como *Transport Layer Security* (TLS); - Reforçar obrigatoriedade de criptografia utilizando diretivas como *HTTP Strict Transport Security* (HSTS).

Tabela 5.1: Soluções para os problemas em plataformas web, como recomendado pela Open Web Application Security Project (OWASP) [9].

ferramenta é indicada para testar APIs RESTful de forma automatizada. Para fins de validar se dados são capturados ao utilizar criptografia de chave pública, foram realizadas duas categorias de testes. Foram realizados testes sem o uso de criptografia entre as partes, e testes com uso de criptografia de chave pública entre todos os componentes do *framework*. Cada um desses testes executa os dois seguintes passos:

1. Chamada do processo de autentica*ção* e execução da função;
2. Captura dos dados relativos à comunicação entre o usuário e a API, e entre a API e a ferramenta de IAM.

Utilizando essa estrutura, foram realizados testes de 10 requisições, com 10 usuários simultâneos, sendo definidos intervalos de 1 segundo por requisição. Assim, foi possível recuperar dados suficientes para verificar a eficácia do uso de criptografia entre as partes. Para fins de teste, foi criado um usuário a partir da rota *{realm}/user/new*, apresentada na Listagem 4.2, com os seguintes atributos passados no corpo da requisição:

- Primeiro nome: John;
- Último nome: Doe;
- Email: testuser@outlook.com;
- Nome de usuário: testuser.

Para recuperar os dados resultantes da comunicação dos componentes do *framework* em rede foi utilizada a ferramenta Wireshark [100]. O Wireshark é um analisador de

pacotes, que permite monitorar o tráfego em uma rede de computadores. A ferramenta captura pacotes trafegados pela rede e decompõe o pacote de acordo com a pilha de rede em utilização, permitindo exportação para diversos formatos [100]. Dessa forma, pode-se verificar se é possível capturar dados de usuário em rede durante a utilização do *framework*.

Além dos testes de segurança, foram realizados testes de desempenho para verificar quantos recursos o *framework* utiliza ao ser executado em um ambiente *on-premises* ou em um serviço de IaaS. Para verificar a utilização de recursos, o uso de memória, latência e tempo de conexão foram monitorados durante a execução dos testes do Apache JMeter, os quais foram realizados da mesma forma que os testes de segurança com criptografia. Vale ressaltar que os dados podem não ser representativos de um ambiente distribuído, no qual os componentes do *framework* e o ambiente de testes são executados em máquinas diferentes. Os testes foram realizados em ambiente local, com as seguintes especificações:

- Sistema Operacional: Arch Linux (GNU/Linux 64 bits) [101];
- Processador: Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz;
- Memória RAM: 4 Gib (Gibibytes).

Os experimentos de desempenho foram realizados utilizando como referência o uso de recursos ao executar uma função no AWS Lambda [50] sem o uso do *framework*. Para isso, foi definida uma regra que permite a execução da mesma função utilizada nos testes do *framework*, mas sem as regras de verificação de *token* no AWS Lambda. Dessa forma, foi possível executar a função por meio de requisições HTTP simples.

5.2 Experimentos de Segurança

Ao serem realizados os testes de segurança descritos anteriormente, os resultados foram compilados em gráficos de pizza, os quais apresentam a proporção de pacotes de rede de cada protocolo em relação à todos os pacotes capturados. A Figura 5.1 apresenta os gráficos gerados, com os dados relativos à execução dos testes em 4 casos:

1. Requisições sem nenhum uso de criptografia;
2. Requisições com uso de criptografia entre o usuário e a API;
3. Requisições com uso de criptografia entre a API e a ferramenta de IAM (Keycloak);
4. Requisições com uso de criptografia entre todas as partes.

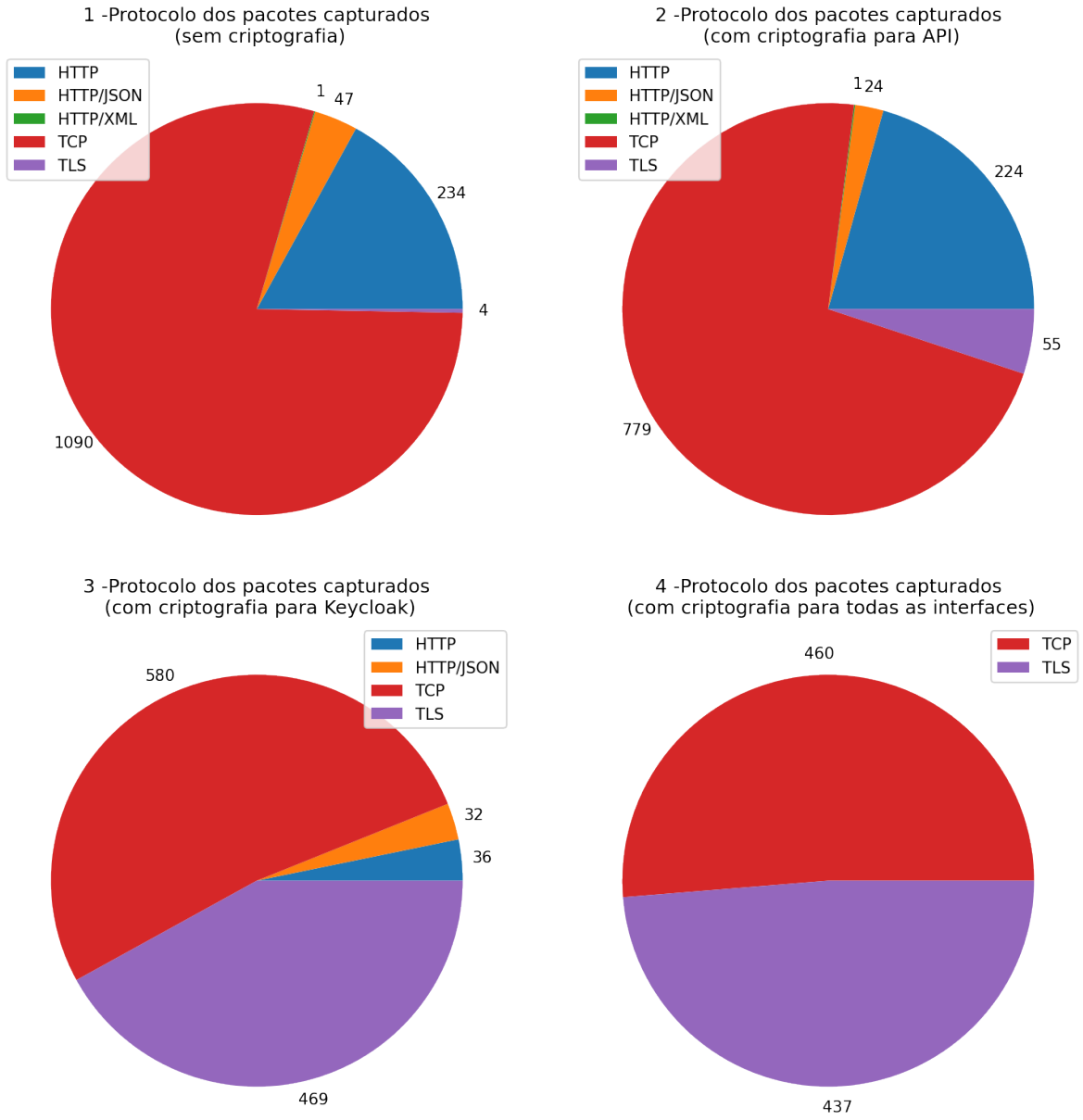

Figura 5.1: Pacotes capturados por protocolo, com e sem criptografia de chave pública entre as partes.

Para geração do gráfico foi utilizada a biblioteca Pandas [102], uma ferramenta de análise e manipulação de dados utilizando a linguagem Python [92].

Pode-se verificar no gráfico 1 da Figura 5.1 que as requisições HTTP realizadas entre as partes são capturadas pelo analisador de pacotes em texto plano nos testes sem criptografia. Ao verificar o conteúdo dos pacotes HTTP é possível encontrar informações de usuário passada para a URL de cadastro, utilizando a rota {realm}/user/new descrita na Seção 4.2.2, como pode ser verificado na Listagem 5.2. Além disso, pode-se verificar que também são capturadas as credenciais utilizadas para comunicação entre o *framework* e

a ferramenta de IAM, como verifica-se na Listagem 5.2. Além disso, é possível ver o tipo dos pacotes capturados e diferenciar pacotes que contém objetos JSON e XML.

Listagem 5.1: Mensagem de resposta da URL de criação de usuário sem uso de criptografia.

```
POST /tcc-test-realm/user/new HTTP/1.1
Host: localhost:8000
User-Agent: insomnia/2021.6.0
Content-Type: application/json
Accept: */*
Content-Length: 115

{
        "username": "testuser",
        "email": "testuser1234@outlook.com",
        "first_name": "John",
        "last_name": "Doe"
}
```

Listagem 5.2: Resposta contendo credenciais de administrador do cliente do Keycloak.

```
POST /auth/realms/master/protocol/openid-connect/token HTTP/1.1
Accept-Encoding: identity
Content-Length: 120
Host: localhost:8080
User-Agent: Python-urllib/3.9
Content-Type: application/x-www-form-urlencoded
Connection: close

username=admin&password=admin&grant_type=password&client_id=admin-cli&
    client_secret=05a6f2f4-7908-47a5-839c-b46080f895ce
```

Já quando é utilizada criptografia para comunicação com a API, o que é apresentado no gráfico 2 da Figura 5.1, a quantidade de pacotes HTTP/JSON diminuiu (de 47 para 24 pacotes), enquanto a quantidade de pacotes TLS aumentou (de 4 para 55 pacotes). Tal diminuição é indicativa da aplicação efetiva do uso das chaves SSH para criptografar a comunicação entre as partes, como apresentado na Seção 3.1.3. Quando é utilizada criptografia entre a API e o Keycloak, o número de pacotes TLS capturados aumentou consideravelmente em relação à execução sem o uso de criptografia (de 1 para 469 pacotes), o que mostra que a grande maioria dos dados sensíveis enviados é transmitido entre a API e a ferramenta de IAM.

Por fim, ao utilizar as chaves SSH para criptografar a comunicação entre todas as partes, como é mostrado no gráfico 4 da Figura 5.1, já não é possível diferenciar pacotes HTTP, visto que todos são encapsulados em pacotes criptografados utilizando o protocolo Transport Layer Security (TLS). Vale ressaltar que todos os pacotes TLS nas versões 1, 1.2 e 1.3 do TLS foram agrupados em um único grupo TLS para fins de visualização. A Tabela 5.2 apresenta a quantidade de pacotes capturados, com e sem utilização de criptografia de chave pública, entre as partes. Pode-se verificar que, ao utilizar chaves SSH para criptografar a comunicação em rede, foram capturados 457 pacotes TLS criptografados. Em contraste, foram capturados 47 pacotes HTTP contendo objetos JSON, representativos dos pacotes enviados para o *framework*. Portanto, pode-se concluir que o *framework* permite o acesso seguro por parte do usuário, dado que a comunicação foi devidamente configurada utilizando as recomendações de criptografia da OWASP.

Protocolo	**Sem criptografia**	**Criptografia (API)**	**Criptografia (Keycloak)**	**Criptografia (total)**
HTTP	234	224	36	0
HTTP/JSON	47	24	32	0
HTTP/XML	1	1	0	0
TCP	1090	779	580	460
TLS	4	55	469	437

Tabela 5.2: Pacotes capturados por protocolo, para os quatro casos de teste de requisição.

5.3 Experimentos de Desempenho

Como descrito anteriormente, foram verificados três aspectos da execução do *framework*: uso de memória em execução, latência das requisições e tempo de conexão. Para verificar o uso de memória, foi utilizado o comando *free* [103], disponibilizado por padrão em sistemas *UNIX-like*, entre eles, sistemas GNU/Linux. A Tabela 5.3 apresenta os resultados da análise de uso de memória. Pode-se verificar que o *framework*, em conjunto com a ferramenta de IAM Keycloak e o Apache JMeter, gera sobrecarga (do inglês *overhead*) no sistema de aproximadamente 500 Mib de utilização em um ambiente local.

Além disso, foi verificada a latência média das requisições ao longo da execução dos testes. Para isso, foi calculada a média da latência de requisição ao longo de 300 segundos (5 minutos), em intervalos de 30 segundos. Pode-se verificar na Figura 5.2 que a latência a cada 30 segundos costuma se manter entre 10 e 25 segundos, com algumas requisições chegando a 40 segundos de latência. Além disso, foi mensurada a média do tempo de

Memória	**Sem o *framework***	**Com o *framework***	**Diferença**
Utilizada	1,8	2,3	0.5
Livre	0.665	0.185	-0.48
Disponível	1,3	0.780	-0.52

Tabela 5.3: Uso de memória com e sem o uso do *framework* de controle de acesso em ambiente local (em Gib).

conexão para as requisições que, salvo raros casos, foi próxima de 0 segundos. Os dados relativos ao tempo de conexão são apresentados na Figura 5.3.

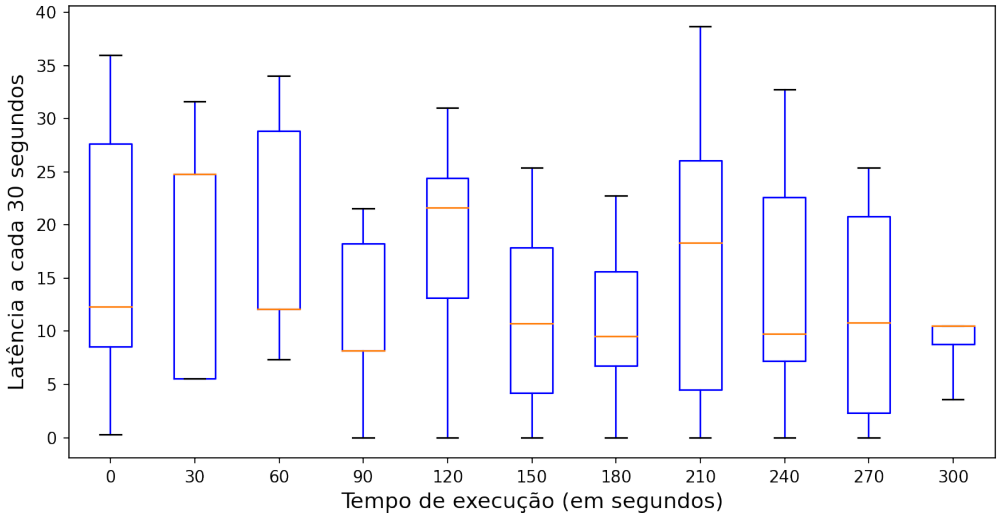

Figura 5.2: Latência média de resposta do *framework* para requisições do Apache JMeter.

Dado que a latência por vezes atingiu picos muito altos para algumas requisições, também verificou-se se essa alta latência configurava requisições com *timeout* ou que geraram algum tipo de erro. Entretanto, como pode ser verificado na Figura 5.4, 92% das requisições feitas foram bem-sucedidas, e apenas 8% das requisições gerou algum tipo de erro. Dessa forma, pode-se constatar que, ainda que as requisições tomem muito tempo para serem processadas, a alta latência e o tempo de conexão não necessariamente ocasionam em uma falha de requisição. Esses resultados mostram que o *framework* proposto permite acessos concorrentes, e pode processar requisições de múltiplos usuários. A Tabela 5.3 também mostra que é possível utilizar o *framework* em infraestrutura própria, sem criar um *overhead* impeditivo para hospedagem em infraestrutura de baixo custo.

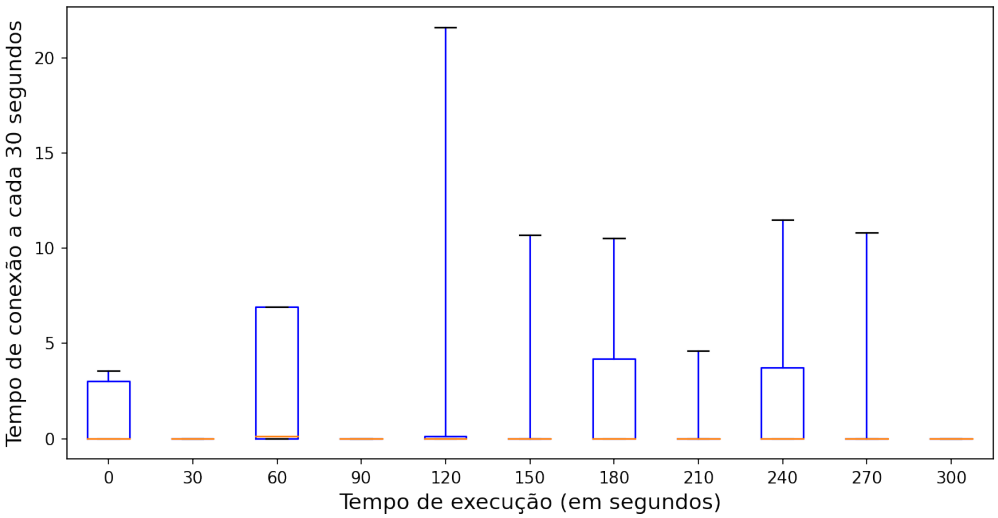

Figura 5.3: Tempo de conexão do *framework* para requisições do Apache JMeter.

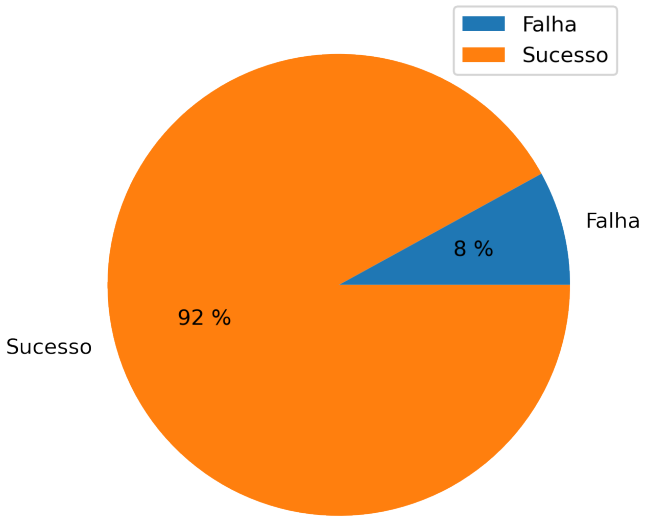

Figura 5.4: Porcentagem de requisições bem-sucedidas e mal-sucedidas no *framework*.

5.4 Considerações Finais

Neste capítulo foram apresentados os experimentos realizados para validar a segurança do *framework* proposto neste trabalho. Concluiu-se que o *framework* permite a execução segura de funções em um ambiente de FaaS, protegendo os dados de usuário em trânsito. Além disso, testou-se o *overhead* que o *framework* tem sobre um sistema computacional, e concluiu-se que é possível executar a ferramenta utilizando aproximadamente 500 Mib (mibibytes) de memória, o que não é impeditivo em sistemas computacionais modernos.

O próximo capítulo tratará das conclusões alcançadas neste trabalho e os passos futuros, de modo a aprimorar a ferramenta proposta.

Capítulo 6

Conclusão

Neste trabalho foram apresentados os conceito de computação em nuvem, e de como os modelos de serviço apresentados diferem dos modelos tradicionais de computação. Assim, foram apresentados os modelos previstos pelas definições do NIST, IaaS, PaaS e SaaS [14], assim como novos modelos que surgiram para preencher lacunas nas definições existentes, em especial o paradigma de FaaS [49]. O FaaS é um modelo de serviço de nuvem que permite a implementação de unidades atômicas de código, chamadas funções, em um ambiente de nuvem altamente gerenciado [49]. Além disso, foram apresentados os diferentes métodos utilizados para controlar o acesso de usuários a serviços de nuvem, em especial os mecanismos de *Single Sign-On*, muito utilizados para proteger recursos em aplicações web. Também foram abordados os contextos nos quais o SSO é utilizado e como é implementado pelos provedores de nuvem em ferramentas de IAM.

Em seguida foi descrito o *framework* de controle de acesso proposto neste trabalho, e como este *framework* pode atuar como uma ferramenta única de controle de acesso para uma multitude de serviços de FaaS em diversos provedores de nuvem pública. Assim, constatou-se que o *framework* oferece uma interface segura para realizar autenticação e autorização antes de executar uma função em ambiente de FaaS, de modo que é possível que um grupo ou organização mantenha sua própria instância do *framework* como uma alternativa segura e performática a uma ferramenta de IAM de algum provedor.

Em trabalhos futuros, serão implementadas novas interfaces com provedores de nuvem, de modo a permitir a execução de funções em outros ambientes de FaaS, a exemplo do Google Cloud Functions [52] e o Azure Functions [53]. Além disso, serão apresentadas novas métricas para validar a segurança do *framework* quanto às outras recomendações da OWASP [9] para resolução de vulnerabilidades de identificação [72], controle de acesso [71] e criptografia [74].

Referências

[1] Gartner: *Quadrante mágico para infraestrutura em nuvem e serviços de plataforma.* https://www.gartner.com/technology/media-products/reprints/AWS/1-271W1OT3-PTB.html, 2021. Acessado: Março, 2022. ix, 5, 6

[2] AWS: *Aws global infrastructure.* https://aws.amazon.com/pt/about-aws/global-infrastructure/?nc1=h_ls, 2022. Acessado: 2022. ix, 6, 7

[3] GCP: *Locais llobais: regiões e zonas.* https://cloud.google.com/about/locations, 2022. Acessado: 2022. ix, 8

[4] Azure: *Rede global da microsoft - azure.* https://docs.microsoft.com/pt-br/azure/networking/microsoft-global-network, 2022. Acessado: 2022. ix, 8, 9

[5] Chou, David: *Cloud service models (iaas, paas, saas) diagram.* https://dachou.github.io/2018/09/28/cloud-service-models.html, 2018. Acessado: Março, 2022. ix, 10

[6] Hughes, John e Eve Maler: *Security assertion markup language (saml) v2. 0 technical overview.* OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08, 13, 2005. ix, 20

[7] Indu, I., P.M.R Anand e V. Bhaskar: *Identity and access management in cloud environment: Mechanisms and challenges.* 2018. ix, 16, 17, 18, 19, 21, 22, 23, 24, 25, 28

[8] Ubale Swapnaja, A, G Modani Dattatray e S Apte Sulabha: *Analysis of dac mac rbac access control based models for security.* International Journal of Computer Applications, 104(5):6–13, 2014. ix, 24, 25

[9] OWASP: *Owasp top ten 2021.* https://owasp.org/Top10/, 2021. Acessado: 2022. x, 26, 27, 42, 43, 44, 52

[10] Divyabharathi, DN e Nagaraj G Cholli: *A review on identity and access management server (keycloak).* International Journal of Security and Privacy in Pervasive Computing (IJSPPC), 12(3):46–53, 2020. x, 30

[11] Shrestha, Sachin: *Comparing programming languages used in aws lambda for serverless architecture.* 2019. 1, 14

[12] Daylami, Nozar: *The origin and construct of cloud computing.* International Journal of the Academic Business World, 9(2):39–45, 2015. 3

[13] Wang, Lizhe, Gregor Von Laszewski, Andrew Younge, Xi He, Marcel Kunze, Jie Tao e Cheng Fu: *Cloud computing: a perspective study*. New generation computing, 28(2):137–146, 2010. 3

[14] Mell, Peter, Tim Grance *et al.*: *The nist definition of cloud computing*. 2011. 3, 4, 10, 11, 12, 52

[15] Liu, Fang, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, Dawn Leaf *et al.*: *Nist cloud computing reference architecture*. NIST special publication, 500(2011):1–28, 2011. 4, 5, 10, 11, 12

[16] AWS: *Cloud computing services - amazon web services (aws)*. `https://aws.amazon.com/`, 2022. Acessado: 2022. 5, 6, 34

[17] GCP: *Cloud computing services | google cloud*. `https://cloud.google.com/`, 2022. Acessado: 2022. 5, 7

[18] Azure: *Cloud computing services | microsoft azure*. `https://azure.microsoft.com/en-us/`, 2022. Acessado: 2022. 5, 8

[19] Alibaba: *Empower you business in usa canada with alibaba cloud*. `https://us.alibabacloud.com/`, 2022. Acessado: 2022. 5, 9

[20] Oracle: *Cloud infrastructure | oracle*. `https://www.oracle.com/cloud/`, 2022. Acessado: 2022. 5, 9

[21] IBM: *Ibm cloud | ibm*. `https://www.ibm.com/cloud`, 2022. Acessado: 2022. 5, 9

[22] Tencent: *Products and services | tencent cloud*. `https://intl.cloud.tencent.com/product`, 2022. Acessado: 2022. 5, 9

[23] Dutta, Pranay e Prashant Dutta: *Comparative study of cloud services offered by amazon, microsoft & google*. International Journal of Trend in Scientific Research and Development, 3(3):981–985, 2019. 6

[24] Xie, Xin, Chentao Wu, Junqing Gu, Han Qiu, Jie Li, Minyi Guo, Xubin He, Yuanyuan Dong e Yafei Zhao: *Az-code: An efficient availability zone level erasure code to provide high fault tolerance in cloud storage systems*. Em *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*, páginas 230–243. IEEE, 2019. 6

[25] AWS: *Secure and resizeable cloud compute - amazon ec2*. `https://aws.amazon.com/ec2/`, 2022. Acessado: 2022. 6, 11

[26] AWS: *Armazenamento s3 - simple storage service*. `https://aws.amazon.com/pt/s3/`, 2022. Acessado: 2022. 6, 14

[27] AWS: *Serviço gerenciado do kubernetes - amazon eks*. `https://aws.amazon.com/pt/eks`, 2022. Acessado: 2022. 6

[28] Prodan, Radu e Simon Ostermann: *A survey and taxonomy of infrastructure as a service and web hosting cloud providers*. Em *2009 10th IEEE/ACM International Conference on Grid Computing*, páginas 17–25. IEEE, 2009. 7, 11

[29] GCP: *Google kubernetes engine | google cloud*. `https://cloud.google.com/kubernetes-engine`, 2022. Acessado: 2022. 7, 11

[30] GCP: *Cloud storage | google cloud*. `https://cloud.google.com/storage`, 2022. Acessado: 2022. 7

[31] GCP: *Cloud sql para mysql, postgresql e sql server*. `https://cloud.google.com/sql/`, 2022. Acessado: 2022. 7

[32] Azure, Microsoft: *Azure active directory*. `https://azure.microsoft.com/en-us/services/active-directory/#overview`, 2022. Acessado: 2022. 8, 28

[33] Azure: *Azure cosmos db*. `https://docs.microsoft.com/en-us/azure/cosmos-db/`, 2022. Acessado: 2022. 8

[34] Azure: *Managed kubernetes service (aks)*. `https://azure.microsoft.com/en-us/services/kubernetes-service/#overview`, 2022. Acessado: 2022. 8

[35] Azure, Microsoft: *Azure disk storage overview | azure virtual machines*. `https://docs.microsoft.com/en-us/azure/virtual-machines/managed-disks-overview`, 2022. Acessado: 2022. 11

[36] Cloud, Alibaba: *File storage nas: Reliable data storage*. `https://www.alibabacloud.com/product/nas`, 2022. Acessado: 2022. 11

[37] Cloud, Oracle: *Virtual cloud network | oracle*. `https://www.oracle.com/cloud/networking/virtual-cloud-network/`, 2022. Acessado: 2022. 11

[38] Cloud, IBM: *Virtual private servers*. `https://www.ibm.com/cloud/virtual-servers`, 2022. Acessado: 2022. 11

[39] Cloud, Tencent: *Tencent*. `https://intl.cloud.tencent.com/ind/products/cfs`, 2022. Acessado: 2022. 11

[40] AWS: *Aws elastic beanstalk | deploy web applications*. `https://aws.amazon.com/elasticbeanstalk/`, 2022. Acessado: 2022. 11

[41] Azure, Microsoft: *Cloud services - deploy cloud apps apis | microsoft azure*. `https://azure.microsoft.com/en-us/services/cloud-services/`, 2022. Acessado: 2022. 12

[42] Cloud, Alibaba: *Alibaba cloud container service for kubernetes*. `https://www.alibabacloud.com/product/kubernetes`, 2022. Acessado: 2022. 12

[43] Cloud, Oracle: *Application development | oracle*. `https://www.oracle.com/application-development/`, 2022. Acessado: 2022. 12

[44] Tencent: *Iot hub | tencent cloud*. https://intl.cloud.tencent.com/products/iothub, 2022. Acessado: 2022. 12

[45] Google: *Google workspace | business apps collaboration tools*. https://workspace.google.com/, 2022. Acessado: 2022. 12

[46] Tencent: *Media processing service | tencent cloud*. https://intl.cloud.tencent.com/products/mps, 2022. Acessado: 2022. 12

[47] Miyachi, Christine: *What is "cloud"? it is time to update the nist definition?* IEEE Cloud computing, 5(03):6–11, 2018. 12, 13

[48] Shahrad, Mohammad, Jonathan Balkind e David Wentzlaff: *Architectural implications of function-as-a-service computing*. Em *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, páginas 1063–1075, 2019. 13

[49] Baldini, Ioana, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski *et al.*: *Serverless computing: Current trends and open problems*. Em *Research advances in cloud computing*, páginas 1–20. Springer, 2017. 13, 52

[50] AWS: *Aws lambda*. https://aws.amazon.com/lambda/url, 2021. Acessado: Outubro, 2021. 14, 45

[51] Wadia, Yohan e Udita Gupta: *Mastering AWS Lambda*. Packt Publishing Ltd, 2017. 14

[52] GCP: *Cloud functions*. https://cloud.google.com/functions/, 2022. Acessado: 2022. 14, 52

[53] Azure: *Azure functions - serverless apps and computing*. https://azure.microsoft.com/en-us/services/functions/#overview, 2022. Acessado: 2022. 14, 52

[54] Alibaba: *Function compute*. https://www.alibabacloud.com/product/function-compute, 2022. Acessado: 2022. 14

[55] Oracle: *Cloud functions | oracle cloud*. https://www.oracle.com/cloud-native/functions/, 2022. Acessado: 2022. 14

[56] IBM: *Ibm cloud functions*. https://cloud.ibm.com/functions/, 2022. Acessado: 2022. 15

[57] Tencent: *Serverless cloud function | tencent cloud*. https://intl.cloud.tencent.com/products/scf, 2022. Acessado: 2022. 15

[58] El Sibai, Rayane, Nader Gemayel, Jacques Bou Abdo e Jacques Demerjian: *A survey on access control mechanisms for cloud computing*. Transactions on Emerging Telecommunications Technologies, 31(2):e3720, 2020. 16

[59] Hwang, Min Shiang e Li Hua Li: *A new remote user authentication scheme using smart cards*. IEEE Transactions on consumer Electronics, 46(1):28–30, 2000. 16, 18

[60] Wayman, James, Anil Jain, Davide Maltoni e Dario Maio: *An introduction to biometric authentication systems*. Em *Biometric Systems*, páginas 1–20. Springer, 2005. 17

[61] Lal, Nilesh A, Salendra Prasad e Mohammed Farik: *A review of authentication methods*. vol, 5:246–249, 2016. 17, 18

[62] Chanda, Katha: *Password security: an analysis of password strengths and vulnerabilities*. International Journal of Computer Network and Information Security, 8(7):23, 2016. 17

[63] Ometov, Aleksandr, Sergey Bezzateev, Niko Mäkitalo, Sergey Andreev, Tommi Mikkonen e Yevgeni Koucheryavy: *Multi-factor authentication: A survey*. Cryptography, 2(1):1, 2018. 17, 18

[64] Play, Google: *Google authenticator - apps on google play*. `https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2`, 2022. Acessado: 2022. 18

[65] FreeOTP: *Freeotp*. `https://freeotp.github.io/`, 2022. Acessado: 2022. 18

[66] Ylonen, Tatu, Chris Lonvick *et al.*: *The secure shell (ssh) protocol architecture*, 2006. 19

[67] Naik, Nitin e Paul Jenkins: *Securing digital identities in the cloud by selecting an apposite federated identity management from saml, oauth and openid connect*. Em *2017 11th International Conference on Research Challenges in Information Science (RCIS)*, páginas 163–174. IEEE, 2017. 20, 21

[68] Nyanchama, Matunda e Sylvia Osborn: *Modeling mandatory access control in role-based security systems*. Em *Database Security IX*, páginas 129–144. Springer, 1996. 23, 24

[69] Osborn, Sylvia, Ravi Sandhu e Qamar Munawer: *Configuring role-based access control to enforce mandatory and discretionary access control policies*. ACM Transactions on Information and System Security (TISSEC), 3(2):85–106, 2000. 23, 24

[70] Li, Jinfeng: *Vulnerabilities mapping based on owasp-sans: a survey for static application security testing (sast)*. Annals of Emerging Technologies in Computing (AETiC), Print ISSN, páginas 2516–0281, 2020. 26

[71] OWASP: *Broken access control*. `https://owasp.org/Top10/A01_2021-Broken_Access_Control/`, 2022. Acessado: 2022. 26, 52

[72] OWASP: *A07 identification and authentication failures*. `https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/`, 2022. Acessado: 2022. 27, 52

[73] NIST: *Digital identity guidelines.* https://pages.nist.gov/800-63-3/sp800-63b.html, 2020. Acessado: 2022. 27

[74] OWASP: *A02:2021 – cryptographic failures.* https://owasp.org/Top10/A02_2021-Cryptographic_Failures/, 2022. Acessado: 2022. 27, 52

[75] Mohammed, Kabiru Hamza, Abubakar Hassan e Danlami Yusuf Mohammed: *Identity and access management system: a web-based approach for an enterprise.* 2018. 28

[76] AWS: *Aws identity and access management (iam).* https://aws.amazon.com/iam/, 2022. Acessado: 2022. 28

[77] GCP: *Identity and access management (iam) | iam | google cloud.* https://cloud.google.com/iam, 2022. Acessado: 2022. 28

[78] Cloud, Alibaba: *Resource access management.* https://www.alibabacloud.com/product/ram, 2022. Acessado: 2022. 28

[79] IBM: *Identity and access management (iam) solutions | ibm.* https://www.ibm.com/security/identity-access-management, 2022. Acessado: 2022. 28

[80] Oracle: *Oracle identity cloud service.* https://docs.oracle.com/en/cloud/paas/identity-cloud/index.html, 2022. Acessado: 2022. 28

[81] Tencent: *Cloud access management.* https://intl.cloud.tencent.com/products/cam, 2022. Acessado: 2022. 28

[82] Shibboleth: *Shibboleth consortium.* https://www.shibboleth.net/, 2022. Acessado: 2022. 29

[83] WSO2: *What is wso2 identity server?* https://wso2.com/library/articles/2017/08/what-is-wso2-identity-server/, 2022. Acessado: 2022. 29

[84] Keycloak: *Keycloak.* https://www.keycloak.org/, 2021. Acessado: Outubro,2021. 29, 33

[85] Cantor, Scott e T Scavo: *Shibboleth architecture.* Protocols and Profiles, 10:16, 2005. 29

[86] Fremantle, Paul, Benjamin Aziz, Jacek Kopeckỳ e Philip Scott: *Federated identity and access management for the internet of things.* Em *2014 International Workshop on Secure Internet of Things*, páginas 10–17. IEEE, 2014. 29, 30

[87] Christie, Marcus A, Anuj Bhandar, Supun Nakandala, Suresh Marru, Eroma Abeysinghe, Sudhakar Pamidighantam e Marlon E Pierce: *Using keycloak for gateway authentication and authorization.* 2017. 30

[88] Keycloak: *Core concepts and terms - server administration guide.* https://www.keycloak.org/docs/latest/server_admin/, 2022. Acessado: 2022. 30, 33

[89] AWS: *Use api gateway lambda authorizers*. https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-use-lambda-authorizer.html, 2022. Acessado: 2022. 34

[90] AWS: *Amazon api gateway*. https://aws.amazon.com/api-gateway/, 2022. Acessado: 2022. 34

[91] MDN: *Object - javascript | mdn*. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object, 2022. Acessado: 2022. 35

[92] Python: *Welcome to python.org*. https://www.python.org/, 2022. Acessado: 2022. 36, 37, 46

[93] Rossum, Guido van: *The history of python: A brief timeline of python*. https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html, 2009. Acessado: 2022. 36

[94] Flask: *Welcome to flask -flask documentation (2.1.x)*. https://flask.palletsprojects.com/en/2.1.x/, 2022. Acessado: 2022. 36, 37

[95] Django: *Django | the web framework for perfectionists with deadlines*. https://www.djangoproject.com/, 2022. Acessado: 2022. 36, 37

[96] Ramírez, Sebatián: *Fastapi*. https://fastapi.tiangolo.com/, 2018. Acessado: 2022. 36, 37

[97] TIOBE: *Tiobe index for april 2022*. https://www.tiobe.com/tiobe-index/, 2022. Acessado: 2022. 37

[98] Encode: *Uvicorn*. https://www.uvicorn.org/, 2022. Acessado: 2022. 37

[99] Foundation, Apache Software: *Apache jmeter*. https://jmeter.apache.org/, 2022. Acessado: 2022. 43

[100] Wireshark: *Wireshark*. https://www.wireshark.org/, 2021. Acessado: Outubro, 2021. 44, 45

[101] Linux, Arch: *Arch linux*. https://archlinux.org/, 2022. Acessado: 2022. 45

[102] pandas: *pandas - python data analysis library*. https://pandas.pydata.org/, 2022. Acessado: 2022. 46

[103] Free: *free - linux manual page*. https://man7.org/linux/man-pages/man1/free.1.html, 2022. Acessado: 2022. 48