UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de São José do Rio Preto

VICTOR FERNANDES GARDINI

AUTOMATIZAÇÃO DE PROCESSOS DE MACHINE LEARNING DO FRAMEWORK DNS PARA A DETECÇÃO DE DOMÍNIOS MALICIOSOS

São José do Rio Preto
2021

VICTOR FERNANDES GARDINI

AUTOMATIZAÇÃO DE PROCESSOS DE MACHINE LEARNING DO FRAMEWORK DNS PARA A DETECÇÃO DE DOMÍNIOS MALICIOSOS

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do título Bacharel em Ciência da Computação, junto ao Departamento de Ciências de Computação e Estatística, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.
Financiador: NIC.br - Proc. 2764/2018

Orientador: Prof. Dr. Adriano Mauro Cansian

São José do Rio Preto
2021

VICTOR FERNANDES GARDINI

AUTOMATIZAÇÃO DE PROCESSOS DE MACHINE LEARNING DO FRAMEWORK DNS PARA A DETECÇÃO DE DOMÍNIOS MALICIOSOS

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do título Bacharel em Ciência da Computação, junto ao Departamento de Ciências de Computação e Estatística, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Comissão Examinadora

Prof. Dr. Adriano Mauro Cansian
UNESP – Câmpus de São José do Rio Preto
Orientador

Prof. Dr. Geraldo Francisco Donegá Zafalon
UNESP – Câmpus de São José do Rio Preto

Profa. Dra. Rogéria Cristiane Gratão de Souza
UNESP – Câmpus de São José do Rio Preto

São José do Rio Preto
2021

“Seja forte e corajoso! Não fique desanimado, nem tenha medo, porque eu, o SENHOR, seu Deus, estarei com você em qualquer lugar para onde você for.”
Josué 1:9

G224a

Gardini, Victor Fernandes
Automatização de processos de machine learning do framework DNS para a detecção de domínios maliciosos / Victor Fernandes Gardini. -- São José do Rio Preto, 2021
82 p.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto
Orientador: Adriano Mauro Cansian

1. Computadores Medidas de segurança. 2. Nomes de domínio na Internet. 3. Software de aplicação. 4. Ciência da computação. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Sirlei e Roberval, por todo apoio, carinho e ensinamentos ao longo desses anos até hoje; aos meus avôs e avós Noemia, Sebastião, Guiomar e Antônio; a minha tia Sandra; aos meus irmãos, Vinicius e Giovanna, por sempre estarem ao meu lado e por oferecerem todo apoio independente de tudo.

Agradeço ao meu orientador Adriano Mauro Cansian, pelas oportunidades, apoio e por todo aprendizado ao longo desses anos, o que contribuiu de forma significativa para a minha formação acadêmica e pessoal. Agradeço aos integrantes do Laboratório ACME!, em especial aos membros Amanda, Bruno, João, Leandro e Marcos, por todos os conhecimentos compartilhados e por toda ajuda. Agradeço aos meus amigos Leandro Marcos da Silva e Paulo André Pimenta Aragão, pela amizade ao longo desses anos de graduação e que levarei por toda minha vida.

Agradeço aos professores do Departamento de Ciências de Computação e Estatística pelos ensinamentos passados.

Este trabalho foi financiado por intermédio do Convênio de Pesquisa e Inovação com o NIC.BR - Núcleo de Informações e Coordenação do Ponto BR, Processo Fundunesp número 2764/2018 - NIC. BR.

RESUMO

A utilização de nomes de domínio para a prática de atividades maliciosas na internet é um problema enfrentado em escala global, com destaque para o Brasil que está no ranking dos países mais afetados por ciberataques do tipo *phishing*. Para resolver isso, diversas abordagens são estudadas pela academia, e entre elas destaca-se a utilização de aprendizado de máquina para a classificação de domínios como maliciosos ou legítimos. Para lidar com isso, foi proposta a classificação de domínios em três estágios, onde cada um deles está ligado à um único sistema denominado *framework* DNS. O sistema permite fazer o treinamento de novos modelos e submeter novos conjuntos de dados para a etapa de treinamento, entretanto os modelos e listas anteriores são descartados durante o processo. Diante disso, foram estudadas abordagens utilizadas pela comunidade acadêmica que culminaram em um conjunto de técnicas e abordagens para gerenciar modelos de aprendizado de máquina, e essas práticas são comumente agrupadas e definidas pelo termo MLOps. A partir disso, foi possível construir um novo sistema com a capacidade de armazenar, versionar e monitorar modelos, listas e *logs* do sistema, que é posteriormente integrado com o *framework*. Dessa forma, cada um dos estágios pode ter, de forma independente, os seus conjuntos de treinamento construídos de forma incremental a partir de operações bem definidas, sem ocasionar a perda do processo anterior. Também é possível criar novos modelos por meio de um *pipeline* automatizado, para que o mesmo seja disponibilizado em ambiente de produção.

Palavras-chave: Cibersegurança, MLOps, Machine Learning, Automatização, *Framework* DNS.

ABSTRACT

The use of domain names for the practice of malicious activities on the internet is a problem faced on a global scale, with emphasis on Brazil, which is in the ranking of countries most affected by phishing-type cyberattacks. To solve this, several approaches are studied by academia, among them the use of machine learning to classify domains as malicious or legitimate stands out. To deal with this, it was proposed to classify domains in three stages, where each one of them is interconnected through a single system called DNS framework. The system allows training new models and submitting new datasets for the training step, however the previous models and lists are lost. Therefore, approaches used by the academic community were studied that culminated in a set of techniques and approaches to manage machine learning models, these practices are commonly grouped and defined by the term MLOps. From that, it was possible to build a new system with the capacity to store, version and monitor models, lists and system logs, being later integrated with the framework. In this way, ensuring that each of the stages can have, independently, their training sets built incrementally from well-defined operations, without causing the loss of the previous process, in addition, allowing the creation of new models through an automated pipeline, so that it is made available to the production environment.

Keywords: Cybersecurity, MLOps, Machine Learning, Automation, DNS Framework.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API: *Application Programming Interface*

AUC: *Area Under The ROC Curve*

ccTLD: *Country Code Top Level Domain*

CD: *Continuous Deploy*

CERN: European Center for Nuclear Physics

CI: *Continuous Integration*

CSS: *Cascading Style Sheets*

CT: *Continuous Training*

CT: *Continuous Training*

DNS: *Domain Name System*

FN: Falso Negativo

FP: Falso Positivo

FQDN: *Fully Qualified Domain Name*

gTLD: *Generic Top Level Domain*

HTML: *HyperText Markup Language*

HTTP: *Hypertext Transfer Protocol*

ICANN: *Internet Corporation for Assigned Names and Numbers*

IP: *Internet Protocol*

ISP: *Internet Service Provider*

JS: JavaScript

JSON: *JavaScript Object Notation*

MG: Módulo de Gerenciamento

ML: *Machine Learning*

MLOps: Machine Learning Operations

REST: *Representational State Transfer*

RFC: *Request for Comments*

ROC: *Receiver Operating Characteristic*

TCP: *Transmission Control Protocol*

TLD: *Top-Level Domain*

TLD: *Top-Level Domain*

URI: *Uniform Resource Identifier*

URL: *Uniform Resource Locator*

URL: *Universal Resource Locator*

VN: Verdadeiro Negativo

VP: Verdadeiro Positivo

SUMÁRIO

Capítulo 1 - Introdução

Com a pilha de protocolos da Internet, cada dispositivo conectado a ela possui em sua interface de rede uma identificação denominada endereço IP (STEVENS, 1993). Este nome vem do protocolo IP (*Internet Protocol*), que é essencial para a rede mundial de computadores, uma vez que realiza a troca dos datagramas entre dispositivos por meio de um endereço de origem e destino (POSTEL, 1981). Esses endereços, dada a sua composição numérica, são facilmente manipulados pelos computadores, porém são difíceis de serem lembrados por seres humanos.

Para isso, foi especificado o protocolo Sistema de Nomes de Domínio (DNS - *Domain Name System*), introduzido no RFC 1034 (MOCKAPETRIS, 1987) e tendo sua implementação detalhada no RFC 1035 (MOCKAPETRIS, 1987). Trata-se de um banco de dados distribuído que fornece o mapeamento entre nomes de *host*[1] e endereços IP ou vice-versa (STEVENS, 1993). O DNS é amplamente utilizado pois, uma vez que esse mapeamento é definido, o protocolo consegue realizar a tradução dos nomes de domínio, comumente utilizados pelos usuários da Internet, em endereços IP, que por sua vez são essenciais para os protocolos utilizados na rede mundial de computadores.

No primeiro trimestre de 2021 foram contabilizados 363,5 milhões de nomes de domínios de topo (ccTLD) registrados pelo mundo todo, e, somente neste intervalo de tempo, foram registrados cerca de 6,3 milhões de novos domínios (VERISIGN, 2021).

[1] Também denominado sistema final, são dispositivos conectados na Internet que hospedam programas de aplicações.

No Brasil, nesse mesmo período, foram registrados aproximadamente 100 mil novos domínios .br, atingindo a marca de aproximadamente 4,6 milhões de domínios (REGISTRO.BR, 2021). Esses nomes de domínios são adquiridos por pessoas ao redor do mundo e no Brasil com diversos objetivos, de lojas para a venda de produtos, prestação de serviços online e até mesmo para atacar outros usuários da Internet.

Entre os principais ataques realizados por usuários na Internet, um dos que mais se destacam é o *phishing*. Neste tipo de ataque, pessoas mal intencionadas se aproveitam do registro de nomes de domínios para emitir uma falsa identidade, com o objetivo de roubar as informações de outros usuários da Internet. Um relatório da Kaspersky aponta a situação do Brasil no segundo trimestre de 2020 com 12,91% de todos os usuários de Internet no país afetados (SECURELIST, 2020), ressaltando a sua elevada colocação entre os países com mais vítimas. Dado o grande volume de domínios registrados mensalmente, é difícil monitorar e verificar o propósito dos novos domínios, uma vez que outros tipos de ataques, como por exemplo *adwares* e *botnets*, podem ser realizados causando prejuízos pessoais e financeiros imensuráveis.

Para resolver esse problema, técnicas baseadas em aprendizado de máquina foram empregadas com a finalidade de classificar domínios como maliciosos ou legítimos. Dentre as pesquisas que utilizam essa abordagem, destacam-se os modelos capazes de realizar a classificação de um domínio por meio da sua composição léxical antes do domínio se tornar operacional (SPOOREN *et al*., 2019) e capaz de classificar domínios que praticam *phishing* (BASNET *et al*., 2014). Além disso, destacam-se os modelos que são capazes de classificar domínios com base na sua configuração DNS: utilizando o tráfego das consultas solicitando a resolução de nomes de domínio, também denominado DNS passivo (BILGE *et al*., 2014; SILVEIRA *et. al*, 2020) e, por fim, utilizando a configuração do respectivo domínio nos servidores DNS, também denominado tráfego ativo (KOUNTOURAS *et al*., 2016; DAN *et al*., 2019).

As abordagens descritas anteriormente propõem a utilização de modelos de aprendizado de máquina para a classificação de domínios como maliciosos ou legítimos. Com base nisso, em trabalhos anteriores do laboratório ACME! (BATISTA, 2020), foi desenvolvida uma aplicação intitulada *framework* DNS, responsável por interligar três modelos de aprendizado de máquina distintos disponibilizados por meio de APIs (*Application Programming Interfaces*), que realizam a classificação de domínios com

base nas suas características léxicas, DNS passivo e tráfego ativo, sendo que cada modelo recebeu o nome de módulo I, módulo II e módulo III, respectivamente.

Identificou-se que o estágio atual do *framework*, em relação ao gerenciamento dos modelos de aprendizado de máquina, pode ser definido como centralizado em modelos, em que o seu principal objetivo consiste em construir os primeiros modelos e disponibilizá-los para produção (MAKINEN *et al*., 2021). A próxima etapa do projeto seria a centralizada em *pipeline*, na qual o objetivo é gerenciar e versionar diversos modelos, além de conjuntos de dados por meio de *pipelines* automatizados e controle de logs, práticas que recebem o nome de MLOps (GOOGLE, 2021). Essa questão torna-se relevante uma vez que cada um dos modelos em produção é treinado com um certo conjunto de dados que pode, por sua vez, se tornar obsoleto com o tempo, desta forma, os modelos em produção também se tornam obsoletos.

Diante disso, mostra-se necessária a implementação de técnicas e abordagens definidas pelo MLOps para que o *framework* DNS atinja o próximo estágio de maturidade. Esta etapa será atingida por meio do desenvolvimento de uma ferramenta capaz de versionar e construir modelos por meio de *pipelines* automatizados, versionar os conjuntos de dados rotulados, além de permitir a visualização de possíveis erros ou *logs* para cada um dos módulos.

1.1 Justificativa

Identificou-se que o *framework* DNS (BATISTA, 2020) encontra-se em transição do estágio centralizado em modelos para o centralizado em *pipelines* (MAKINEN *et al*., 2021), ou seja, os primeiros modelos foram gerados e alcançou-se resultados significativos em suas abordagens (SILVEIRA et. al, 2020). Para que o *framework* e os respectivos módulos que o integram possam compor uma aplicação plenamente capaz de classificar domínios em diferentes situações, devem-se empregar as práticas comumente utilizadas no MLOps como capacidade de armazenar e versionar listas de domínios rotulados e os modelos sem que conteúdo anterior seja perdido (MAKINEN *et al*., 2021).

Nesse contexto, a implementação de uma API que possibilite o gerenciamento dos módulos com as principais práticas do MLOps foi desenvolvida e integrada ao *framework*. Garantindo que os conjuntos de treinamento possam ser incrementais, viabilizando a criação de novos modelos por meio de um *pipeline* automatizado para que

cada um deles seja disponibilizado para o ambiente de produção a qualquer momento pelo operador do sistema. A proposta desenvolvida neste trabalho contribui com o avanço da classificação de domínios como maliciosos ou legítimos, por meio de um sistema que gerencia os modelos integrando-os a interfaces gráficas, o que por sua vez contribui para o aumento da capacidade em tratar de diversos modelos de aprendizado de máquina. Isso possibilita a eficiência do *framework* em manter os seus modelos atualizados na medida que o conjunto de treinamento também é atualizado, o que por fim apoia a capacidade de detectar e mitigar comportamentos maliciosos na rede.

1.2 Objetivos

O objetivo deste trabalho é implementar um novo conjunto de funcionalidades nos diferentes módulos que integram o *framework* DNS, de tal forma que viabilize o gerenciamento dos modelos disponíveis para o ambiente de produção. Este objetivo principal foi alcançado por meio de sete objetivos secundários, conforme segue:

1. Construir um *pipeline* automatizado que viabilize a geração de modelos de aprendizado de máquina;
2. Implementar um sistema de versionamento para os conjuntos de treinamento de modelos, bem como o armazenamento das informações a respeito dos modelos;
3. A API deve possuir a capacidade de alternar entre modelos prontos no ambiente de produção, bem como um sistema de monitoramento de *logs* e erros;
4. A API deve possuir um sistema que possibilite monitorar domínios que, posteriormente, sejam analisados, rotulados e introduzidos nos conjuntos de treinamento selecionados;
5. Realizar a integração das novas funcionalidades da API no *framework* DNS por meio de novas páginas;
6. Permitir aos usuários do *framework* acesso a todos os dados relevantes relacionados às listas e modelos;
7. Permitir que o analista de dados consiga manipular algumas informações armazenadas na API pelo *framework* DNS, por meio da interface gráfica.

1.3 Organização da Monografia

Esta monografia está dividida em quatro capítulos principais. No Capítulo 2, é realizada uma revisão bibliográfica a fim de contextualizar o leitor sobre os conceitos e tecnologias utilizadas para o desenvolvimento desta pesquisa. No Capítulo 3, é apresentando como será realizado o desenvolvimento do sistema proposto por meio da descrição do sistema, sua arquitetura e componentes, como será realizada a integração com o *framework* DNS. No Capítulo 4, são apresentados os testes realizados e os resultados obtidos para o modelo proposto. Por fim, no Capítulo 5, são apresentadas as conclusões obtidas, as dificuldades encontradas e propostas de trabalhos futuros.

Capítulo 2 - Revisão Bibliográfica

2.1 Protocolo DNS

Os diversos dispositivos conectados à Internet comumente utilizam a pilha de protocolos TCP/IP em sua interface de rede (STEVENS, 1993). Desta forma, cada um deles possui uma identificação denominada endereço IP, onde a sua composição varia de acordo com a versão do protocolo utilizado. No caso da quarta versão, em que o protocolo também é conhecido como IPv4, o endereço é dividido em quatro grupos de 8 bits cada, totalizando 32 bits, representados por números decimais e separados por pontos, como por exemplo 192.168.0.1. Por outro lado, temos a sexta versão do protocolo IP, comumente chamado de IPv6, em que a quantidade de grupos e de bits aumentam quando comparados com o IPv4, passando para oito grupos de 16 bits cada, totalizando 128 bits, representados no sistema hexadecimal[2] e separados por dois pontos, como por exemplo 2001:0DB8:AD1F:25E2:0000:0000:0000:0000 (IPv6.BR, 2021).

Os endereços IP são difíceis de serem memorizados para os seres humanos, até mesmo os endereços IPv4 que são menores e mais simples que o IPv6. Por isso, nos primórdios da Internet, antes do protocolo DNS e IPv6 serem definidos, era comum realizar o mapeamento entre os *hosts* de uma rede local por meio de uma tabela com os nomes e os respectivos endereços no próprio dispositivo, o que era feito fazendo uso de

[2] A notação hexadecimal (base 16) possui os dígitos: 0, 1, 2, ..., 8, 9, A, B, C, D, E, F.

um simples arquivo de texto chamado “hosts.txt”, cuja implementação é descrita no RFC 952 (FEINLER *et al*., 1985). A problemática desta implementação está na necessidade do arquivo ser configurado manualmente em cada um dos computadores da rede local, dada a demanda de se manter sempre atualizado, exigindo muita coordenação da equipe e pelo fato da implementação não ser escalável.

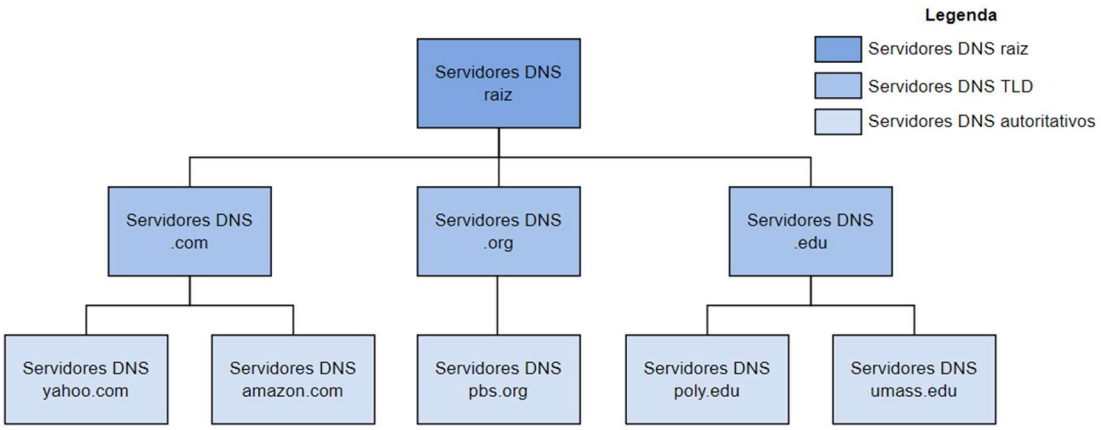
A partir da exigência de se resolver problemas como esse, foi introduzido o protocolo Sistema de Nomes de Domínio (DNS - *Domain Name System*) no RFC 1034 (MOCKAPETRIS, 1987) como um banco de dados distribuído executado em uma hierarquia de servidores DNS, e também como um protocolo de camada de aplicação capaz de permitir consultas no seu banco de dados, onde estão os mapeamentos entre nomes de *host* e endereços IP, ou vice-versa (KUROSE; ROSS, 2012).

2.1.1 Hierarquia DNS

Os servidores DNS são utilizados de forma constante e a nível global, por isso um grande número de servidores DNS estão organizados respeitando uma determinada hierarquia e estão distribuídos pelo mundo todo permitindo que os mapeamentos estejam distribuídos entre eles (KUROSE; ROSS, 2012). Diante disso, a hierarquia na qual os servidores estão configurados é baseada em árvore, sendo comparável com a estrutura do sistema de arquivos utilizada nos sistemas Unix (STEVENS, 1993).

Essa hierarquia se dá por meio de três classes de servidores DNS: raiz, de domínio de alto nível (TLD) e servidores DNS autoritativos. Os servidores DNS raiz estão no topo da hierarquia, existindo 13 servidores e 1402 réplicas operadas ao redor do mundo (SERVERS.ORG, 2021), e sua função é possuir as informações dos domínios de alto nível ou TLDs. Os servidores TLD são responsáveis pelos domínios de alto nível, sendo eles: “com”, “org”, “edu”, “pt”, “br”, entre outros, e esses nomes são categorizados em: domínios geográficos (*Country Code Top Level Domain* - ccTLD), e domínios genéricos (*Generic Top Level Domain* - gTLD). Por fim, é possível destacar que os servidores DNS autoritativos são os responsáveis por possuir os endereços dos *hosts* que estão sendo consultados, ou seja, nestes servidores os mapeamentos dos nomes de domínio são definidos. Na Figura 1 é ilustrado um exemplo de como a hierarquia dos servidores DNS está organizada.

Figura 1: Hierarquia dos servidores DNS



Fonte: Adaptado de Kurose; Ross (2012)

Há um tipo de servidor, denominado servidor DNS local, que normalmente é disponibilizado por um *Internet Service Provider* (ISP) para os seus clientes, e sua função é retransmitir consultas DNS auxiliando no processo de tradução, o que será discutido na seção 2.1.2. Cabe destacar que este servidor é central para a arquitetura DNS, porém não pertence, estritamente, à hierarquia (KUROSE; ROSS, 2012).

Destaca-se que cada nó da hierarquia possui o mapeamento respectivo ao seu nó filho, também chamado de subdomínio, tornando possível a resolução do nome de domínio. Por fim, é possível definir o FQDN (*Fully Qualified Domain Name*), como sendo um nome de domínio construído por meio do percurso entre os servidores inferiores até o servidor raiz (STEVENS, 1993).

2.1.2 Tradução de nomes de domínio

O seguinte exemplo foi adaptado do livro de Kurose; Ross, (2012) e retrata como é realizada a tradução de um nome de domínio. Supondo um computador denominado cliente, que por sua vez deseja saber qual é o endereço IP do domínio “example.com”, a consulta DNS para o processo é detalhada nos seguintes passos, conforme também ilustrado na Figura 2:

Passo 1: O cliente consulta o servidor DNS local, que normalmente é o servidor mais próximo do cliente;

Passo 2: Caso o servidor DNS local não tenha essa informação, ele será o responsável por encaminhá-la para um servidor DNS raiz.

Passo 3: O servidor DNS raiz responde a consulta com o endereço do servidor DNS TLD responsável pelo domínio "`.com`";

Passo 4: Então a consulta é repetida para o servidor DNS responsável pelo domínio "`.com`";

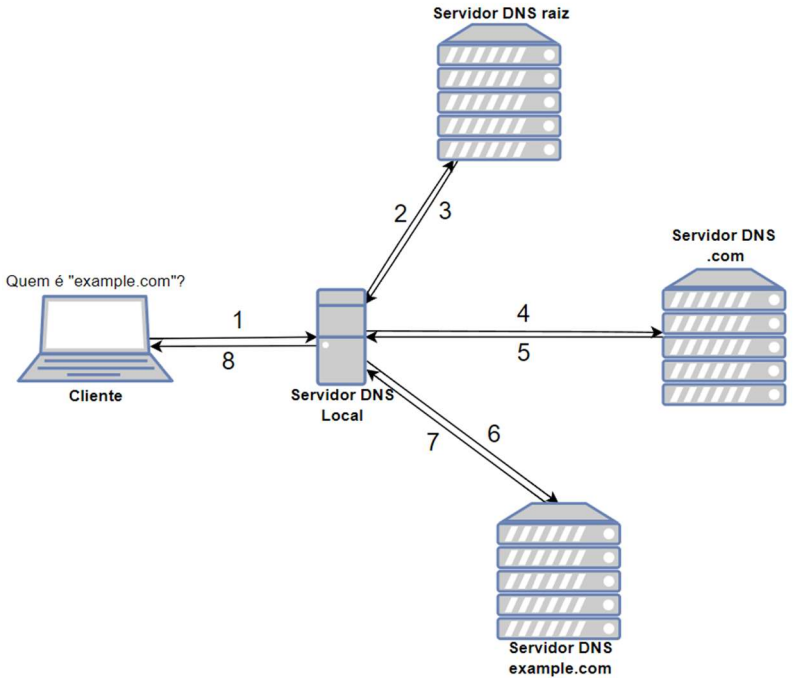
Passo 5: O servidor responde com o endereço do servidor autoritativo responsável pelo domínio "`example.com`";

Passo 6: A consulta é repetida para o servidor DNS autoritativo responsável pelo domínio "`example.com`";

Passo 7: O servidor possuindo o mapeamento configurado corretamente, responde com o respectivo endereço IP de "`example.com`";

Passo 8: O servidor DNS local agora possui o endereço resultante da consulta, e o encaminha para o cliente.

Figura 2: Diagrama exemplo de uma consulta DNS



Fonte: Elaborado pelo autor

2.1.3 Registro de Domínios e Coleta de Dados DNS

Registrar um domínio é o nome dado a uma sequência de etapas na qual um domínio será disponibilizado para ser mapeado nos servidores DNS globais por uma pessoa física ou jurídica. Isso pode ser realizado por meio de três agentes: *registry, registrar* e *registrant* (ICANN, 2021).

Um *registry* é uma entidade responsável por administrar e armazenar registros de segundo nível dos TLDs que foram delegados pela ICANN (*Internet Corporation for Assigned Names and Numbers*)[3]. Sua administração dá-se por meio da criação de extensões de nome de domínio e suas respectivas regras, além de poder trabalhar com *registrars* para a alocação de nomes de domínios. O *registrar* é entidade credenciada por um *registry* responsável por realizar o registro de nomes de domínio requeridos por um *registrant*, armazenando as principais informações técnicas e administrativas. Por fim, o *registrant* é a pessoa ou entidade que deseja realizar a alocação do nome de domínio desejado, sendo o responsável por fornecer os dados ao *registrar*, como o nome e os mapeamentos desejados (KUROSE; ROSS, 2012).

Uma vez que o nome de domínio é registrado, é possível utilizar técnicas de coletas de dados para armazenar as configurações registradas publicamente nos servidores DNS autoritativos, podendo ocorrer de duas formas distintas: passiva ou ativa. A coleta ativa de DNS consiste em realizar uma consulta e armazenar o respectivo resultado (KOUNTOURAS *et al*., 2016). Por outro lado, a coleta passiva exige que um monitoramento da rede seja implementado, com o objetivo de armazenar as respostas das consultas do servidor DNS que está sendo monitorado (ANTONAKAKIS *et al*., 2010).

2.2 Aprendizado de Máquina

Em 1959 (WEISS, 1992), foi apresentada uma definição sobre o aprendizado de máquina, como sendo “o campo de estudo que dá aos computadores a habilidade de aprender sem serem explicitamente programados”. Destaca-se que esta área surgiu como um método para o desenvolvimento de *softwares* para visão computacional, reconhecimento de fala, processamento de linguagem natural, controle de robôs, entre outras (JORDAN; MITCHELL, 2015). Porém, sua aplicabilidade também foi demonstrada como promissora quanto a classificação de domínios (SILVEIRA et. al, 2020).

Segundo Géron (2019), os sistemas de aprendizado de máquina podem ser separados em: supervisionados, cuja principal característica é a presença de rótulos durante a etapa de treinamento (é possível fazer uma analogia de que o algoritmo é como

[3] Corporação internacional sem fins lucrativos e responsável pela alocação de endereços IP, gerenciamento dos TLDs e dos servidores DNS raiz (ICANN, 2021).

um aluno aprendendo com a ajuda de um professor), e o aprendizado não supervisionado, cuja principal característica está na ausência dos rótulos durante a etapa de treinamento (utilizado a mesma analogia anterior, é como um aluno aprendendo sem a ajuda do professor).

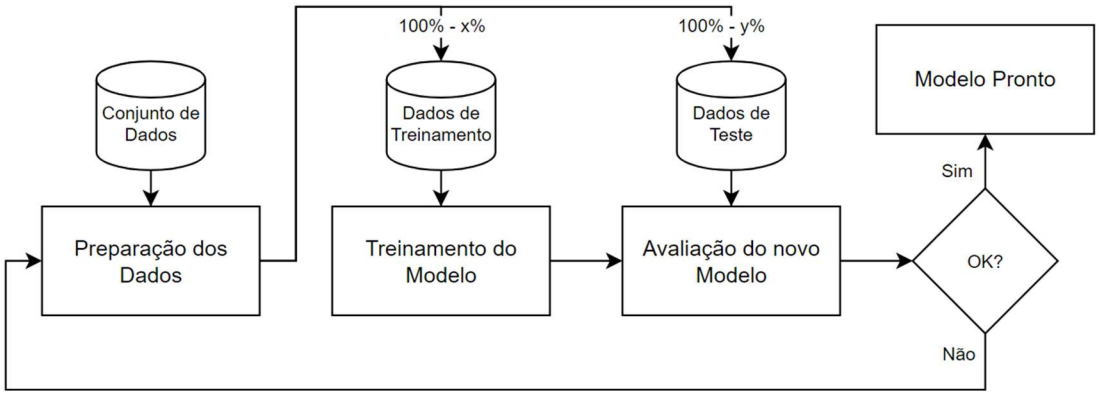
2.2.1 Aprendizado Supervisionado

O objetivo do aprendizado de máquina é obter um classificador como produto final que, quando concluído, é o responsável por classificar os novos dados submetidos. Todavia, para obter um modelo final, é necessário que uma sequência de etapas seja realizada de forma sequencial, também denominada *pipeline*. Conforme ilustrado na Figura 3, baseando-se em Kotsiantis *et al*. (2007), o aprendizado de máquina supervisionado pode ser resumido no seguinte *pipeline*:

- Preparação dos dados: A primeira etapa é coletar o conjunto de dados, para serem tratados. Nesta etapa, métodos podem ser empregados para lidar com campos vazios ou ruídos (dados que poderiam atrapalhar o treinamento do modelo). Ao final dessa etapa os dados do conjunto podem ser reduzidos, restando apenas os mais relevantes para a próxima etapa funcionar de forma eficaz;
- Treinamento do modelo: Antes mesmo dessa etapa ser iniciada, os dados são divididos em duas partes, para que uma parcela seja utilizada durante o treinamento do modelo e a outra para testá-lo. Por exemplo, 70% ($x = 70\%$) do conjunto será utilizado para o treinamento e 30% ($y = 30\%$) para testá-lo[4];
- Avaliação do modelo: A outra parcela do conjunto de treinamento é submetida para ser classificada pelo novo modelo. Como os dados são rotulados, é possível obter as métricas do modelo (serão aprofundadas na seção 2.2.2). Se a taxa de erro for insatisfatória, o novo modelo pode ser descartado e o ciclo começa novamente com base na decisão do cientista de dados responsável pelo processo, caso a taxa de erro seja satisfatória, o modelo pode ser considerado pronto para classificar dados não rotulados.

[4] Nesse contexto, as variáveis x e y foram escolhidas arbitrariamente apenas para representar dois conjuntos distintos.

Figura 3: Etapas do aprendizado de máquina supervisionado



Fonte: Elaborado pelo autor

2.2.2 Validando classificadores

Validar um classificador é uma etapa importante quando se está trabalhando com aprendizado de máquina. Após a etapa de treinamento, é relevante saber qual o desempenho do modelo resultante, por isso, uma parte do conjunto rotulado é submetido para a classificação e, a partir disso, é possível realizar algumas prospecções.

Em Kotsiantis *et al.* (2007), é destacado que a matriz de confusão é uma ótima maneira de avaliar o desempenho de um classificador. É possível obtê-la logo após a etapa de treinamento, comparando os resultados da classificação submetida ao modelo com dados rotulados, e, por meio dela, é possível saber quantas vezes o modelo classificou o que é "A" como "B", ou vice-versa. Tomando apenas como exemplo, supondo um modelo que classifique como positivos os domínios malignos, e negativos os domínios legítimos, seguem as definições abaixo:

- VP (verdadeiro positivo): número de vezes que o modelo previu como positivo corretamente. No contexto desse trabalho, o modelo classificou o domínio como malicioso corretamente;
- FP (falso positivo): número de vezes que o modelo previu como positivo incorretamente. No contexto desse trabalho, o modelo classificou o domínio como malicioso incorretamente;
- VN (verdadeiro negativo): número de vezes que o modelo previu como negativo corretamente. No contexto desse trabalho, o modelo classificou o domínio como legítimo corretamente;

- FN (falso negativo): número de vezes que o modelo previu como negativo incorretamente. No contexto desse trabalho, o modelo classificou o domínio como legítimo incorretamente.

Com as definições anteriores, é possível ilustrar a matriz de confusão e os seus respectivos campos, conforme segue na Figura 4.

Figura 4: Ilustração de uma matriz de confusão

		Previsto	
		Negativo	Positivo
Real	Negativo	VN	FP
	Positivo	FN	VP

Fonte: Adaptado de Kotsiantis *et al*. (2007)

Uma vez que a matriz de confusão é definida, é possível obter métricas importantes para a avaliação de um modelo, como à acurácia, *recall,* precisão e *f1-score* (KOTSIANTIS *et al*., 2007).

A partir da matriz de confusão, é possível obter outra métrica denominada Área Sob a Curva ROC (AUC), que tem sido cada vez mais adotada pelas comunidades de aprendizagem de máquina, dada a sua capacidade de avaliar classificadores binários (FAWCETT, 2004). Pode ser representada na forma decimal, com valores no intervalo entre 0 e 1, e na forma gráfica. Quanto mais próximo de 1, mais o modelo resultante é capaz de diferenciar entre as duas classes, isto é, melhorar a sua *performance*.

2.3 Serviços Web

A *World Wide Web*, conhecida como Web, foi inventada por Tim Berners-Lee e seus companheiros no ano de 1990, por meio de uma pesquisa no CERN (*European Center for Nuclear Physics*). A equipe também é responsável pelas versões iniciais dos protocolos HTML (*HyperText Markup Language*) e HTTP (*Hypertext Transfer Protocol*), por um servidor Web e um navegador, os quatro componentes fundamentais da Web (KUROSSE, 2012). A Web foi, inicialmente, proposta como um sistema de hipertexto para compartilhamento de dados e informações entre cientistas, e se tornou uma plataforma incomparável para o desenvolvimento de sistemas de informação distribuídos (KOPECKÝ, 2014).

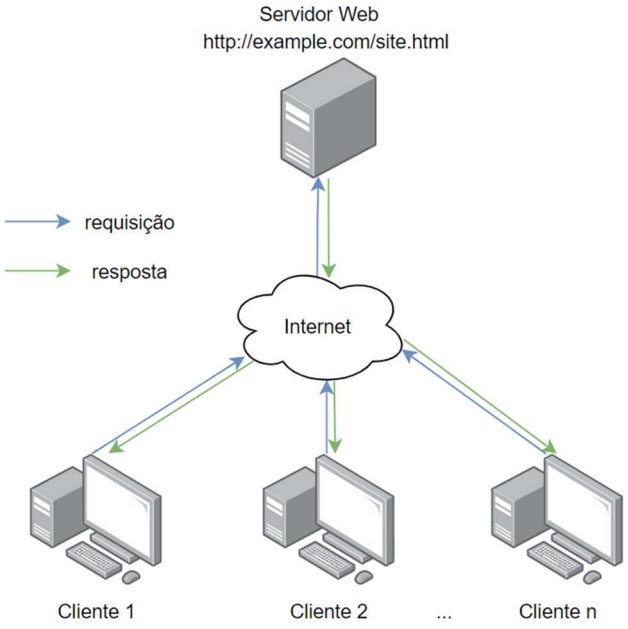
No final do milênio, a Internet dava suporte a centenas de aplicações Web oferecendo diversos serviços populares (KUROSE, 2012), como por exemplo:

- Serviços de e-mail, oferecendo suporte a anexos e provendo a utilização do correio eletrônico por meio de uma interface Web;
- Servidores Web que permitiram a navegação pela Web (como sistemas de busca) e sistemas de comércio pela Internet;
- Servidores com serviços de mensagens instantâneas entre os usuários;
- Serviços que permitiam o compartilhamento *peer-to-peer* de arquivos de extensão MP3.

A Web funciona numa arquitetura denominada cliente-servidor, e sua principal característica é possuir um *host* (servidor) com um serviço sempre em funcionamento. Ele é o responsável por atender às requisições originadas por outros *hosts* (clientes). Quando um servidor recebe a requisição do cliente, ele então responde enviando o recurso solicitado.

A comunicação entre o cliente e servidor ocorre por meio do endereço IP do servidor, que comumente é fixo. Na Figura 5 é ilustrada a arquitetura em questão. É importante ressaltar que os clientes enviam a requisição para o servidor mediante o seu endereço IP, que nesse caso foi identificado pelo nome de *host* example.com, que será traduzido por meio do protocolo DNS; além disso, para cada requisição recebida dos seus clientes, o servidor irá processar e responder cada um de forma individual.

Figura 5: Arquitetura cliente-servidor



Fonte: Elaborado pelo autor

2.3.1 O protocolo HTTP

No RFC 2616 (Fielding, 1999), está definida a primeira versão do protocolo HTTP. Trata-se de um protocolo de camada de aplicação que viabiliza a troca de mensagens HTML entre dois *hosts*. Na etapa de requisição feita do cliente para um servidor, é possível especificar alguns parâmetros, entre eles:

- Versão do protocolo: responsável por especificar a versão do protocolo a ser utilizada na requisição auxiliando o servidor a interpretar a mensagem e o conteúdo do pacote, por exemplo: "1.1";
- URI (*Uniform Resource Identifier*): especificar a localização global do recurso. Trata-se de uma cadeia de caracteres que especifica o nome de domínio (*host*), a localização do recurso acessado (*path*) e, dependendo do método HTTP, os parâmetros da requisição (*query*). Além disso, existem as URLs (*Uniform Resource Locator*) que, adicionando-se a URI, especifica o protocolo utilizado (*schema*) no início da cadeia de caracteres, indicando qual protocolo utilizar para acessar o recurso. Na
- Figura 5, é possível encontrar um exemplo de URL com os parâmetros: *schema* = "http", *host* = "example.com", *path* = "/site.html".
- Método HTTP: indica o método que deve ser executado no recurso especificado pela URI. Este parâmetro auxilia o servidor a entender que tipo de informação o cliente deseja. Foram selecionados os seguintes identificadores entre os possíveis:
    - "OPTION": indica que o cliente está solicitando quais opções de comunicação estão disponíveis pelo servidor naquela URI;
    - "GET": indica que o cliente está solicitando a informação oferecida pela URI;
    - "HEAD": idêntico ao método anterior, porém não deve ser retornado nenhum conteúdo além das informações contidas no cabeçalho da resposta para a requisição;
    - "POST": indica que o cliente está solicitando ao servidor que aceite os dados fornecidos no corpo da requisição;

- “PUT”: indica que o cliente está solicitando ao servidor que o conteúdo fornecido na requisição seja armazenado, caso não exista, ou que seja sobrescrito, caso exista;
- “DELETE”: indica que o cliente está solicitando ao servidor que o recurso seja deletado.

O conteúdo apresentado anteriormente é referente a uma requisição feita pelo cliente a um servidor. Como o protocolo é implementado com base na arquitetura cliente-servidor, é esperado que, para cada requisição, haja uma resposta específica. A resposta é sempre acompanhada de um código de *status*, que por sua vez são valores inteiros classificados em cinco classes e intervalos, sendo eles:

1. Classe de informação: representada pelos códigos no intervalo de 100 a 199, sua função é indicar uma resposta provisória ao cliente;
2. Classe de sucesso: representado pelos códigos no intervalo de 200 a 299, sua função é indicar que a solicitação foi recebida, compreendida e aceita com sucesso pelo servidor;
3. Classe de redirecionamento: representada pelos códigos no intervalo 300 a 399, sua função é indicar que outras ações precisam ser executadas pelo cliente para atender à solicitação. Dependendo da ação, pode ser realizada sem a interação do usuário;
4. Classe de erros por parte do cliente: representada pelos códigos no intervalo 400 a 499, sua função é indicar que houve um erro de requisição por parte do cliente;
5. Classe de erro por parte do servidor: representada pelos códigos no intervalo 500 a 599, sua função é indicar que ocorreu um erro para aquela requisição no servidor.

Na Tabela 1, é possível encontrar alguns exemplos com as principais classes de respostas e o seu respectivo significado.

Tabela 1: Principais códigos de status para requisições HTTP

Nome do Código	Código	Descrição
Continue	100	Tudo OK com a requisição, o cliente deve continuar com a solicitação ou ignorá-la caso já esteja concluída.
OK	200	Indica que a requisição foi concluída com sucesso.
Created	201	A solicitação para a criação do recurso foi um sucesso.
No Content	204	Não existe conteúdo a ser retornado.
Moved Permanently	301	A localização do recurso solicitado foi alterada permanentemente, a resposta contém a nova URL.
Found	302	O recurso solicitado foi alterado temporariamente.
Bad Request	400	Não foi possível atender à solicitação devido a um erro de sintaxe pelo usuário.
Unauthorized	401	Indica que o usuário não está autenticado para acessar o recurso.
Forbidden	403	Indica que o usuário não tem autorização para acessar o recurso.
Not Found	404	O servidor não conseguiu encontrar o recurso associado à requisição.
Method Not Allowed	405	O método HTTP especificado na requisição não foi encontrado.
Internal Server Error	500	Ocorreu um erro interno ao servidor e a requisição não pode ser completada.
Not Implemented	501	O método HTTP especificado não é suportado.
Bad Gateway	502	Enquanto o servidor completava a requisição por meio de um *gateway*, recebeu uma resposta inesperada.
Service Unavailable	503	O servidor não está pronto para atender a requisição.

Fonte: Adaptado de Fielding; Reschke (2014)

2.3.2 JSON

O JSON (*JavaScript Object Notation*) é um formato de texto para a serialização de dados estruturados independentemente do idioma (RFC 8259, 2017). Seu principal objetivo é viabilizar a troca de dados entre diferentes sistemas (JSON.ORG, 2021). A técnica só é possível por meio de estruturas de dados simples e comuns entre diversas linguagens, tornando-o fácil de ser lido e construído por seres humanos e diferentes sistemas. Mesmo que cada sistema seja executado com linguagens de programação diferentes, o JSON possui extensa compatibilidade com diversas linguagens de forma nativa, caso contrário, existem diversas bibliotecas externas que podem ser acopladas para permitir a sua compatibilidade.

2.3.3 API RESTful

Segundo Redhat (2021), API (*Application Programming Interface*) é um conjunto de definições e protocolos usados no desenvolvimento e na integração de *software* de aplicações. Seu objetivo é permitir que soluções e serviços se comuniquem entre si, sem a necessidade de saber como foram implementados.

Serviços da Web podem adotar a arquitetura REST (*Representational State Transfer*) para servidores HTTP. Introduzido por Roy Fielding (FIELDING, 2000), trata-se de um conjunto de características fundamentais para o desenvolvimento das aplicações. O seu objetivo é orientar boas práticas durante a implementação de APIs, aproveitando-se das definições estabelecidas pelo protocolo HTTP para definir a composição das URLs, além da utilização dos códigos de status e métodos do protocolo.

Para uma API ser considerada RESTful (REDHAT, 2021), as seguintes restrições precisam ser satisfeitas:

1. *Client-server*: separar interface do usuário e do servidor, responsável pelo armazenamento e processamento dos dados;
2. *Stateless*: as solicitação do cliente não devem possuir estado, para isso, a requisição deve conter todas as informações necessárias para que o servidor consiga processá-la;
3. *Cache*: o servidor precisa ser capaz de armazenar dados em cache, com a finalidade de otimizar as interações entre o cliente e o servidor;
4. *Uniform Interface*: é preciso que a interface entre os componentes seja uniforme, permitindo que as informações sejam transferidas de forma padronizada. Isso é possível por meio das seguintes restrições:
    - Identificação dos recursos: os recursos solicitados pelo cliente devem ser identificáveis;
    - Manipulação de recursos: representações devem ser adotadas para a manipulação dos recursos, desde que as informações sejam suficientes;
    - Mensagens auto descritivas: as mensagens retornadas aos clientes precisam conter informações suficientes para serem processadas;
    - Hipertexto e hipermídia: após o cliente receber um recurso, é possível encontrar as demais ações disponíveis no momento.

5. *Layered System*: sistemas em camadas hierárquicas, o cliente não será capaz de ver o comportamento da API, isso permite que esses sistemas possam ser implementados em camadas que restrinjam o comportamento dos componentes.

2.4 DevOps

Muitos departamentos de TI possuem uma divisão clara de responsabilidades entre o time de desenvolvimento, responsável por criar novas aplicações, adicionar novas funcionalidades ou corrigir *bugs*, trata-se de um time que é incentivado para introduzir mudanças no *software*; e também o time de operações, responsável por cuidar dos produtos e das aplicações em produção prezando pela estabilidade do sistema como um todo (SATO, 2013).

Em Fitzgerald *et al*. (2015), é destacado que o surgimento do conceito DevOps deu-se por meio da crescente desconexão entre as funções de desenvolvimento e de operações que surgem em grandes empresas de *software*. Isso fica mais evidente uma vez que as organizações estão em constante crescimento e expansão, ocorrendo o mesmo com as suas equipes. Embora elas possam estar localizadas próximas e possuam bons vínculos de comunicação, o crescimento da equipe e a separação mais estrita das responsabilidades podem enfraquecê-las. Para evitar isso, é necessária uma conexão mais estreita entre o desenvolvimento e a execução para garantir que os erros sejam detectados e corrigidos o mais rápido possível, resultando no aprimoramento da qualidade e resiliência de um *software*.

Diante das tendências de mudanças de perspectivas sobre as responsabilidades e medições de um sistema, além da presença cada vez maior de ferramentas de automação (FITZGERALD *et al*., 2015), foram introduzidos os quatro princípios do DevOps:

- Cultural: o DevOps requer a aceitação da responsabilidade conjunta das equipes prezando pela entrega e alta qualidade do *software*;
- Automação: o DevOps preza pela implementação de automações com o objetivo de atingir prazos de entrega curtos, obtendo o *feedback* dos usuários finais mais rapidamente;
- Medição: é preciso que o time tenha compreensão de sua capacidade de entrega atual e que busque definir metas para melhorá-la por meio de

medições, como cobertura de testes e o tempo para implantar uma nova versão do *software*;

- Compartilhamento: compartilhamento de conhecimento, ferramentas e infraestrutura, visando a aproximação das equipes.

Diante dos princípios anteriormente definidos, pode-se observar a sua influência em diversas atividades contínuas, em Fitzgerald *et al*. (2015) foram definidas algumas delas, com destaque para as descritas na

Tabela 2.

A partir das atividades apresentadas, tem-se um exemplo de um *pipeline* DevOps automatizado, conforme ilustrado na Figura 6.

2.5 MLOps

Em Makinen *et al.* (2021), é destacado o recente interesse em implantar, de forma rápida, os recursos de aprendizado de máquina (ML), e tal prática recebe o nome de *Machine Learning Operations* (MLOps). Para este fim, o MLOps refere-se à defesa da automação e monitoramento em todas as etapas referentes ao desenvolvimento e implantação de um sistema de ML, isto é, executar a *Continuous Integration* (CI), *Continuous Delivery* (CD) e o *Continuous Training* (CT) para sistemas de ML.

No contexto de MLOps, algumas coisas mudam (GOOGLE, 2021). Nesse caso, o CI deve testar e validar dados, esquemas de dados e modelos, além de continuar testando e validando códigos e componentes, já o CD irá contemplar um *pipeline* de treinamento ML que deve implantar o serviço de predição dos modelos, por fim, o CT deve se preocupar em treinar e exibir automaticamente os modelos.

Em Tamburri (2020), é destacado que para que um sistema seja desenvolvido baseando-se em MLOps, o mesmo precisa oferecer um conjunto de ferramentas para serem utilizadas com a finalidade de cumprir, pelo menos, 5 funções essenciais, sendo elas:

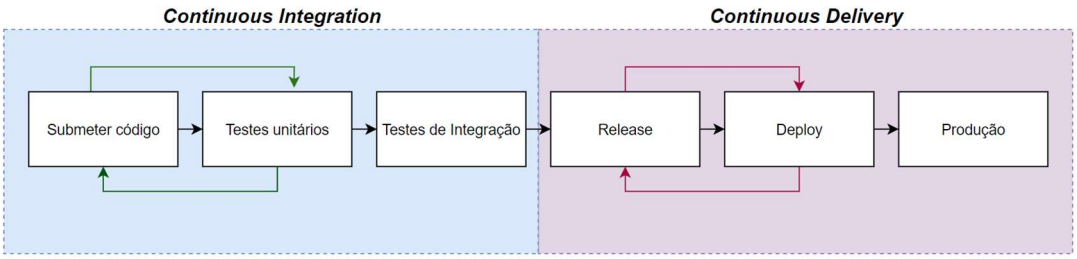
1. Ingestão/transporte dos dados;
2. Transformação de dados;
3. Treinar e retreinar continuamente um modelo;
4. Implantar e reimplantar um modelo de forma contínua;
5. Produzir e apresentar a saída do modelo para o usuário final.

Tabela 2: Descrição das atividades contínuas

Time Afetado	Atividade	Descrição
Desenvolvimento	*Continuous integration* (CI)	Esta etapa contempla o acionamento de um mecanismo de forma automática, este que será responsável pela execução de determinadas etapas encadeadas, como a compilação do código, execução de testes de unidade e de aceitação, relatórios de cobertura de códigos, verificação de erros de sintaxe ou a criação de pacotes de implantação. O objetivo é obter um rápido *feedback* para os desenvolvedores. Nesta etapa podem ser encontrados problemas ou falhas, esses eventos ajudam a garantir que a integração do sistema seja feita de forma correta.
	Continuous Deployment	Esta etapa contempla a implantação do sistema desenvolvido de forma contínua para algum ambiente, não necessariamente para usuários finais.
	Continuous Delivery (CD)	Esta etapa contempla a disponibilização do sistema a qualquer momento para os usuários reais de forma automatizada.
	Continuous Verification	Esta etapa contempla a adoção de atividades de verificação, como métodos formais e inspeções ao longo do processo de desenvolvimento de forma contínua.
	Continuous Testing	Esta etapa normalmente envolve a utilização de testes automatizados do sistema, como ajudar a reduzir o tempo entre a introdução dos erros e sua detecção.
Operações	Continuous Run-Time Monitoring	Esta etapa contempla monitorar o comportamento do sistema durante sua execução, o objetivo é permitir a detecção precoce de problemas de qualidade de serviço, como degradação de desempenho.

Fonte: Adaptado de Fitzgerald *et al*. (2015)

Figura 6: Exemplo de *pipeline* DevOps



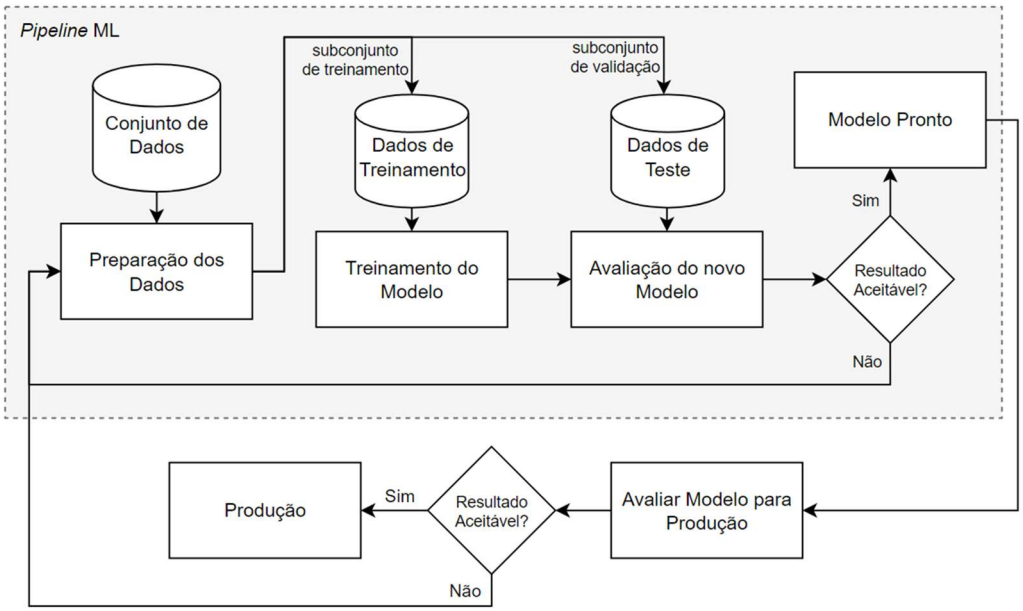
Fonte: Elaborado pelo autor

As três primeiras funções essenciais definidas anteriormente são referentes ao *pipeline* de treinamento de um modelo, o seu objetivo é encontrar e fornecer os conjuntos de dados que serão utilizados durante o treinamento de um modelo, para a etapa de preparação, onde serão processados e as informações mais relevantes serão extraídas e direcionadas para a etapa seguinte. Esta próxima etapa é responsável por subdividir o conjunto de dados em duas partes complementares, sendo um deles utilizado durante o treinamento e o outro utilizado durante a validação do modelo.

As duas últimas etapas se referem ao gerenciamento de um modelo para o ambiente de produção, e sua principal responsabilidade é garantir que um modelo seja utilizado de forma contínua e que seja facilmente substituível caso o resultado da validação seja consideravelmente relevante. Caso necessário, o processo de treinamento pode ser submetido novamente, repetindo as etapas anteriores, com outros conjuntos de dados ou parâmetros diferentes para o modelo de aprendizado afim de obter resultados diferentes.

A partir das discussões anteriores foi possível desenvolver uma adaptação de MLOps em um *pipeline* de treinamento supervisionado, cujo resultado pode ser observado na Figura 7.

Figura 7: Exemplo de um pipeline MLOps



Fonte: Elaborado pelo autor

2.6 Trabalhos Correlatos

Em Akiba *et al*. (2019), é apresentado um *framework* de otimização de hiperparâmetros. Um dos fatores a serem destacados nesse projeto é a questão da arquitetura do sistema escolhida para a implementação e reutilização das propriedades e estruturas da linguagem Python, com ênfase no desenvolvimento de forma altamente modularizada. O *framework* resultante permite que, mesmo em cenários mais complexos, a legibilidade e interpretação do código permaneça em alto nível. Por meio do desenvolvimento modular, as classes e métodos desenvolvidos são facilmente reaproveitados em qualquer trecho do código de forma independente.

Em Arnold *et al*. (2020), é apresentado um conjunto de tecnologias que podem ser utilizadas com a finalidade de aumentar o nível de automação nas operações com ML, consequentemente reduzindo o esforço humano necessário. Os autores destacam o estágio de monitoramento dos modelos em produção, verificando a variação de desempenho e alertando um operador de que a precisão dos modelos caiu, e o mecanismo pode ser adicionado manualmente ou em determinados períodos. Além disso, destacou-se a realização de testes antes do lançamento do modelo para produção, denominado testes

pré-lançamento, e também o diagnóstico de problemas e possíveis melhorias dos modelos de forma contínua.

Em Fursin *et al*. (2020), é desenvolvido o CodeReef, uma plataforma colaborativa para o compartilhamento de modelos ML portáveis com a possibilidade de automatização dos treinamentos, realizar *benchmarks*[5] e permitir a portabilidade MLOps. A linguagem de programação Python foi utilizada no projeto, com o auxílio de outras tecnologias como o JSON, responsável por armazenar a configuração do sistema, permitindo a configuração, observação e interação com os resultados por meio de uma API.

Em Makinen *et al.* (2021), os autores realizaram uma pesquisa com o objetivo de entender o grau de maturidade dos cientistas de dados que trabalharam com ML em 2020. Com um total de 331 respostas de 63 países, os autores propõem que as organizações sejam agrupadas em três categorias, sendo que cada uma delas representa a sua maturidade:

i) Centralizada em dados: o objetivo atual da organização está concentrado em descobrir e aprimorar a melhor forma de utilizar os seus dados;
ii) Centralizada em modelos: o objetivo atual da organização está concentrado em construir seus primeiros modelos e disponibilizá-los para produção;
iii) Centralizada em *pipeline*: o objetivo atual da organização está concentrado no gerenciamento dos diversos classificadores e conjuntos de dados que possui, além da reciclagem e implantação frequente de novos modelos retreinados.

Os autores destacaram que, no geral, as organizações começam como centralizadas em dados e, em seguida, avançam para o modo centralizado em modelos, quando solucionam problemas relacionados à engenharia de dados. Assim que a organização domina a construção de modelos, ela pode transformar o ML em um procedimento operacional padrão, que requer automação na forma de *pipelines*. Além disso, foi destacado que a maior parte das tarefas giram em torno de dados relacionais e de séries temporais, reforçando a necessidade de trabalhar com diversos conjuntos de dados e modelos diferentes. Com relação aos benefícios de implantar MLOps em uma organização, os autores destacam que eles surgem apenas para a categoria (iii), uma vez que existe uma necessidade de reciclar e redistribuir os modelos regularmente.

[5] Nome dado a um conjunto de técnicas com o objetivo de avaliar a performance do mesmo por meio de métricas como velocidade, latência e precisão.

Em Renggli *et al*. (2021), é destacado que a qualidade de um modelo de ML é, muitas vezes, um reflexo da qualidade do conjunto de treinamento. Como um dos objetivos do MLOps é buscar formas de se aprimorar os modelos, os autores propõem que uma das maneiras mais promissoras de se melhorar a precisão, justiça e robustez de um modelo de ML é melhorar o conjunto de treinamento regularmente, o que pode ser feito por meio da limpeza dos dados, integração e aquisição de novos rótulos. Além disso, é destacado que a tarefa de limpeza do conjunto pode ser realizada de forma semiautomática, isto é, por seres humanos com o auxílio de ferramentas automáticas. Nesse contexto, o objetivo do MLOps é justamente minimizar a quantidade de esforço humano no processo.

Os autores também destacam a questão de como definir um novo modelo para produção, denominado adaptação automática de domínio, como sendo comum em empresas que trabalham com coleções de modelos. A questão é que, uma vez que cada modelo é gerado a partir de um conjunto de dados diferentes, cada classificador pronto no sistema tem o potencial de ser utilizado em produção. Diante disso, para que a adaptação automática funcione, metainformações dos modelos como o dia e horário no qual os treinamentos foram submetidos são destacadas como dados que são comumente utilizados nas decisões das empresas.

Por fim, em Granlund *et al*. (2021), são exemplificados dois casos em que o MLOps foi implementado em duas organizações distintas. Com destaque para o primeiro caso, é relatado como foi realizada a implementação de MLOps de um sistema que possui dados sensíveis e confidenciais a respeito de trinta mil, e posteriormente quarenta mil, pacientes. O modelo foi construído como uma forma de interface entre o hospital e a provedora de serviços responsável pelo *software*, de forma que uma organização ficou responsável pelo treinamento enquanto a outra poderia consumir os modelos, sem que os dados de treinamento fossem expostos. O ritmo de treinamento é definido pelos experimentos realizados no hospital, ou manualmente, onde a cada etapa de treinamento, é criada uma nova versão do modelo, que pode ou não ser entregue à provedora de serviços.

2.7 Considerações Finais

Em Makinen *et al.* (2021) ficou evidente o estágio no qual uma organização passa até atingir maturidade suficiente para implementar o MLOps, o que pode ser observado com exemplos reais nos dois casos relatados por Granlund *et al*. (2021). Em todos esses casos fica nítida a presença de elementos no time de operações, desenvolvimento e treinamento. O último time, adicionado pelo MLOps, é importante para o desenvolvimento e manutenção dos modelos em produção, uma vez que são as pessoas mais capacitadas para a orientação de quais alterações deverão ser realizadas a fim de melhorar o *pipeline* de MLOps.

A implementação de MLOps envolvendo dados sensíveis foi relatada em Granlund *et al*. (2021), em que a desenvolvedora e responsável pelo sistema não poderia ter acesso aos dados de treinamento fornecidos pelo hospital responsável pelos pacientes. Nesse contexto, a implementação de MLOps ainda foi possível uma vez que o modelo resultante do treinamento, que é acionado pela equipe do hospital, age como intermediário entre a empresa, que tem acesso ao modelo pronto, e o hospital, que possui os dados para o treinamento, como uma forma segura para o caso de relacionar organizações e dados sensíveis.

Em Akiba *et al*. (2019), observou-se a importância de desenvolver um sistema com ótima documentação e com os seus componentes modularizados. Essa implementação permite a localização mais rápida dos recursos do *framework,* além de facilitar a implementação e manutenção dos mesmos. A linguagem também pode ser observada no projeto de Fursin *et al*. (2020), em que foi empregada no desenvolvimento da plataforma que suporta modelos portáteis e a realização de *benchmarks*, além disso, pode ser observada a utilização da tecnologia JSON para a comunicação com a API e configuração do sistema. Essas tecnologias puderam ser bem exploradas por serem um ponto forte da linguagem de programação Python, permitindo integração com várias outras tecnologias e mantendo-se o alto nível de código.

Por fim, em Renggli *et al*. (2021), fica destacada a necessidade de implementar ferramentas de suporte para o conjunto de treinamento do *pipeline*, uma vez que a qualidade dos estados de treinamento está associado aos resultados do modelo final, e é preciso que ferramentas para o gerenciamento do conjunto estejam disponíveis em alto nível para o cientista de dados responsável pelo treinamento. É importante destacar que

o papel do MLOps é automatizar o máximo possível as ferramentas com a finalidade de facilitar a tarefa para o usuário do sistema.

Alinhando-se aos trabalhos correlatos, o presente trabalho implementa um *pipeline* MLOps para gerenciar modelos que realizam a classificação de domínios como maliciosos ou legítimos na forma de uma API. Com base nas observações realizadas nessa seção, foram selecionadas as tecnologias e metodologias mais relevantes para o desenvolvimento e integração com o *framework* DNS.

Capítulo 3 - Desenvolvimento

3.1 Considerações Iniciais

Os estudos teóricos levantados no capítulo 2 ajudaram a fornecer uma base de conhecimento sólido sobre as principais dificuldades e abordagens utilizadas para a implementação de MLOps. Assim sendo, foi definido um conjunto de tecnologias e procedimentos que foram implementados neste estudo.

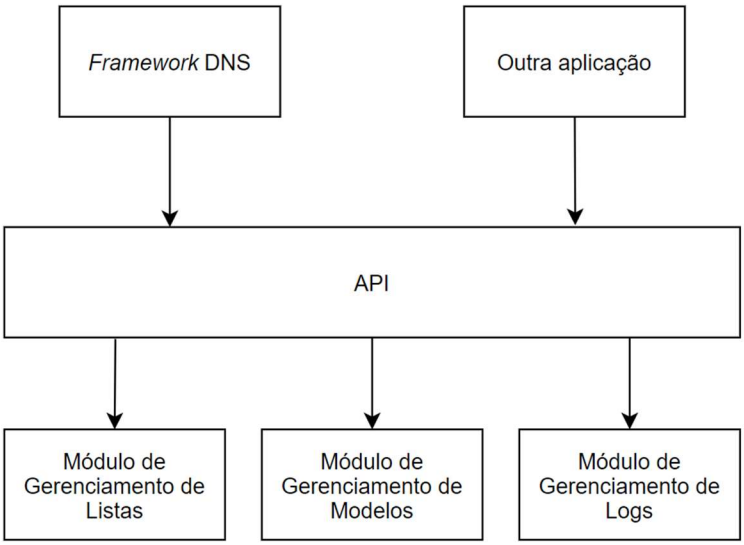
Diante disso, este capítulo tem como objetivo apresentar a metodologia utilizada durante a elaboração do presente trabalho. Na seção 3.2 é apresentada a arquitetura geral do sistema, enquanto que nas seções 3.3, 3.4 e 3.5 são apresentados e definidos todos os módulos que compuseram este trabalho; na seção 3.6 são apresentadas as telas responsáveis pela integração da API com o *framework* DNS e, por fim, na seção 3.7 são apresentadas as considerações finais a respeito do desenvolvimento do sistema.

3.2 Arquitetura do Sistema

Na Figura 8 é apresentada a arquitetura geral do sistema em que é possível observar o papel da API como uma conexão entre o mundo externo, que contempla as aplicações que consumirão os seus recursos, como o *framework* DNS ou outras aplicações; e o mundo interno, que estão representados na figura como sendo três módulos. Cada um desses módulos agrupam um conjunto de funcionalidades suportadas pela API: o gerenciamento de listas, o gerenciamento de modelos, e o gerenciamento de *logs*.

Inicialmente, a API é um servidor Web permanecendo em constante espera por uma requisição para interpretá-la e executar os recursos disponíveis, como a classificação de domínios, exibição das listas, modelos e *logs*, executar o treinamento de um novo modelo; entre outros. Como a API está sendo desenvolvida baseando-se na arquitetura REST, cada recurso é acionado com base nas informações disponíveis na requisição, como a URL e método HTTP e serão retornadas informações, normalmente contendo uma mensagem no formato JSON, referentes ao recurso em questão, e um código HTTP, sinalizando que a requisição foi respondida com sucesso.

Figura 8: Esquema de arquitetura geral do sistema



Fonte: Elaborado pelo autor

Para cada um dos recursos descritos na sequência, será apresentada a URL base do módulo, isto é, todos os recursos disponíveis serão variações da mesma, e o método HTTP que ajuda a identificá-lo, como por exemplo: o módulo de gerenciamento de listas possui a URL base "*lists*", significa que, para acessar todos os recursos do módulo, será necessário acessar a URL "http://api.com.br/lists/"[6].

3.3 Módulo de Gerenciamento de Listas

A URL base desse módulo é a "lists", e o seu objetivo é listar, gerenciar e versionar as listas disponíveis para que sejam utilizadas no processo de treinamento de

[6] Apenas para fins didáticos, neste trabalho será considerado que a API possui o nome de host como "api.com.br".

um novo modelo. Destaca-se que, no contexto desse trabalho, a lista foi definida como uma forma de identificar cada conjunto de domínios de forma única os quais, por sua vez, podem ser utilizados durante o treinamento dos modelos de aprendizado de máquina.

3.3.1 Listagem e Metadados de uma Lista

As informações descritas na Tabela 3 são armazenadas e ficam disponíveis para serem requisitadas a qualquer momento por meio da API. Quanto à edição dos dados, apenas os campos descrição e *blocklist*[7] estarão disponíveis para serem editados pelo usuário, uma vez que a descrição foi elaborada com esta finalidade. Para a última opção, estará disponível caso o cientista de dados tenha salvado uma lista, por engano, como *blocklist* verdadeiro e deseja reverter para falso, ou vice-versa. Desta forma, os demais campos são controlados e preenchidos pela própria API automaticamente, para fins de controle.

Tabela 3: Metadados das listas

Dados	Descrição
Identificador único	Identificador, pode ser utilizado para definir sua versão.
Descrição	Descrição curta, nota ou observação a respeito da lista.
Data criado	Data na qual a lista foi criada.
Data Atualizado	Data na qual ocorreu a última modificação da lista.
Tamanho	Quantidade de domínios armazenados.
Blocklist	Também referenciado como tipo de lista, sinaliza se possui domínios rotulados como maliciosos (verdadeiro) ou legítimos (falso).

Fonte: Elaborado pelo autor

3.3.2 Armazenamento e Versionamento

Para adicionar uma nova lista na API, é necessário que alguns dados, como por exemplo os domínios pertencentes àquela lista e seu tipo, sejam informados na requisição. Para isso, foram definidas três operações de inserção de conjuntos de treinamento no sistema. Ao final de cada operação, é atribuído um identificador único para o novo

[7] Termo comumente utilizado para definir o conjunto de dados cujos domínios são rotulados como maliciosos.

conjunto de treinamento, que pode ser armazenado juntamente com a data da operação e a quantidade de domínios que foram adicionados.

É importante ressaltar que, para todas as possibilidades de inserção de domínios na API, a mesma precisa verificar se não existe nenhum dado repetido nos conjuntos que estão sendo operados. Essa etapa é muito importante, uma vez que a sua ocorrência pode interferir na qualidade do modelo que irá utilizá-la durante a etapa de treinamento (RENGGLI *et al*., 2021).

3.3.3 Operação de Adição de uma nova lista

A primeira operação de listas foi identificada como a de adição de uma nova lista na API, sua principal característica é o recebimento de um conjunto contendo domínios que precisam ser armazenados no sistema. Para isso, foi necessário realizar a implementação de um algoritmo capaz de remover domínios repetidos no conjunto informado. Além disso, é necessária a remoção de domínios que possam estar presentes nos domínios do conjunto de testes.

O conjunto resultante após a remoção das duplicidades é então submetido a uma etapa denominada *Hold-Out* (RASCHKA, 2018), onde os domínios são subdivididos de forma aleatória em outros dois subconjuntos. Uma divisão comum utilizada nessa técnica é a 80/20 (RASCHKA, 2018), que consiste em obter um subconjunto de menor tamanho para os testes, com uma parcela de 20% do total de domínios, enquanto que o subconjunto de maior tamanho, com os demais 80% de domínios, são utilizados no treinamento.

Supondo que um conjunto de domínios seja submetido ao sistema com a operação de adição, conforme descrito anteriormente, será possível obter uma nova lista não existente no sistema com os seus dados devidamente filtrados.

3.3.4 Operação de Adição em uma lista existente

A segunda operação de listas definidas para a API é a de adição de domínios a uma lista já existente. Assim como na primeira operação, sua principal característica é a de introduzir um conjunto de dados rotulados para a API, e o diferencial é que os dados devem ser acrescentados a uma lista que já existe no sistema. Eles são analisados e as entradas duplicadas são identificadas e removidas do conjunto, e isso ocorre também para os dados em comum com o conjunto de testes reservados e os que já pertencem à lista,

isto é, como produto final dessa etapa de filtragem inicial, têm-se os dados únicos que não pertencem tanto ao conjunto de testes quanto ao conjunto alvo.

Após a etapa de filtragem, assim como na primeira operação, os dados são separados em outros dois subconjuntos de forma aleatória, em que 80% serão adicionados à lista já existente, enquanto que os demais 20% serão adicionados ao conjunto de testes.

Supondo uma lista pré-existente no sistema, ao ser submetida à operação descrita anteriormente, no fim do processo a lista resultante possui, além dos dados que já possuía, os novos domínios filtrados da operação.

3.3.5 Operação de União entre Listas

A terceira e última operação definida para a API é a de união entre duas listas. Destaca-se que, no contexto dessa operação, os conjuntos de dados já pertencem ao sistema; desta forma, não é necessário que um novo conjunto de domínios seja disponibilizado pelo operador do sistema.

Diante disso, mediante a identificação de dois conjuntos de dados previamente armazenados no sistema, a etapa inicial consiste em formar um conjunto de dados a partir da captura de todos os dados que pertencem aos conjuntos que serão unidos. Depois disso, é possível aplicar uma filtragem com a finalidade de identificar entradas duplicadas entre os dois conjuntos, uma vez que, pelas operações definidas anteriormente, os dados que pertencem aos conjuntos não se encontram no conjunto de testes. Dessa forma, é possível formar a nova lista diretamente a partir do conjunto resultante da união entre os dois.

Supondo duas listas pré-existentes no sistema, ao serem submetidas à operação descrita anteriormente, no fim do processo é criada uma nova lista, onde o seu conteúdo é o mesmo que as duas utilizadas no processo.

3.4 Módulo de Gerenciamento de Modelos

A URL base desse módulo é a "*classifiers*", e o seu objetivo é listar, gerenciar, versionar, trocar o modelo que se encontra em produção, além de submeter um novo modelo para a etapa de treinamento a partir de duas listas de domínios rotulados. Destaca-se que, no contexto desse trabalho, o modelo foi definido como uma forma de identificar cada um dos classificadores da API.

3.4.1 Listagem e Metadados de um Modelo

As informações descritas na Tabela 4 são armazenadas e ficam disponíveis para serem listadas a qualquer momento por meio da API. Quanto à edição dos dados, apenas o campo padrão está disponível para ser editado pelo usuário; isso se deve ao fato de que o modelo que é utilizado para classificar os domínios submetidos à API pode ser alterado a qualquer momento. Desta forma, os dados restantes serão controlados e preenchidos pela própria API, para fins de controle.

Tabela 4: Metadados dos modelos

Dados	Descrição
Identificador único	Identificador, pode ser utilizado para definir sua versão
Blocklist	Identifica a blocklist utilizada durante o seu treinamento
Allowlist[8]	Identifica a allowlist utilizada durante o seu treinamento
Matriz de Confusão	Resultado da matriz de confusão obtido após o término do treinamento, na etapa de validação do modelo
Padrão	Se o modelo será utilizado para classificação de domínios da API
Data criado	Data na qual o treinamento foi submetido
Data Finalizado	Data na qual o treinamento do modelo foi finalizado
Status	Status atual do treinamento, as opções são explicadas na seção 3.4.2

Fonte: Elaborado pelo autor

3.4.2 *Pipeline* de Treinamento Automatizado

Considerando um trabalho anterior do laboratório ACME! (POMPEU, 2018), foi possível obter os códigos que contemplam a extração de características (referente à etapa de preparação dos dados de um modelo) e os códigos para o treinamento de modelos utilizando o algoritmo *Random Forest* com seus respectivos parâmetros já otimizados para a classificação de domínios como legítimos ou maliciosos. Diante disso, neste presente trabalho será implementado um *pipeline* de treinamento automatizado baseando-se na extração de características em nomes de domínios utilizando DNS ativo e o

[8] Termo comumente utilizado para definir o conjunto de dados cujos domínios são rotulados como legítimos.

treinamento de modelos utilizando o algoritmo *Random* Forest, respectivos ao módulo II do *framework* DNS.

O *pipeline* inicia-se quando a API recebe uma requisição para gerar um novo classificador pelo usuário, onde é informada a versão da *blocklist* e *allowlist* que devem ser utilizados durante o treinamento do novo modelo. Para cada etapa do *pipeline* automatizado, o valor do *status* do respectivo modelo é atualizado automaticamente pelo sistema, uma vez que ele é o responsável por sinalizar o estado onde o modelo se encontra no momento. A seguir, é apresentado cada possibilidade que o campo pode assumir durante o treinamento, bem como o seu significado:

Estágio 1 - Esperando submeter treinamento: o novo modelo foi cadastrado na base de dados da API e o treinamento ainda não foi submetido, porém o mesmo está pronto e poderá ser iniciado a qualquer momento;

Estágio 2 - Preparando os dados: as listas de domínios legítimos e malignos foram separadas, também foi iniciada a etapa de extração de suas respectivas características que serão utilizadas na etapa de treinamento do modelo. Ainda nesse estágio, na medida em que as características são separadas, os resultados estão sendo construídos e armazenados em *DataFrames*[9];

Estágio 3 - Treinamento submetido: nesta etapa o treinamento do modelo foi submetido. Após isso, a validação do modelo é realizada onde a matriz de confusão e a área sob a curva ROC (AUC) são extraídas e armazenadas pela API;

Estágio 4 - Classificador pronto: esta etapa representa o estágio final do treinamento do modelo, isto é, o treinamento foi finalizado e o modelo está pronto para realizar as classificações dos domínios;

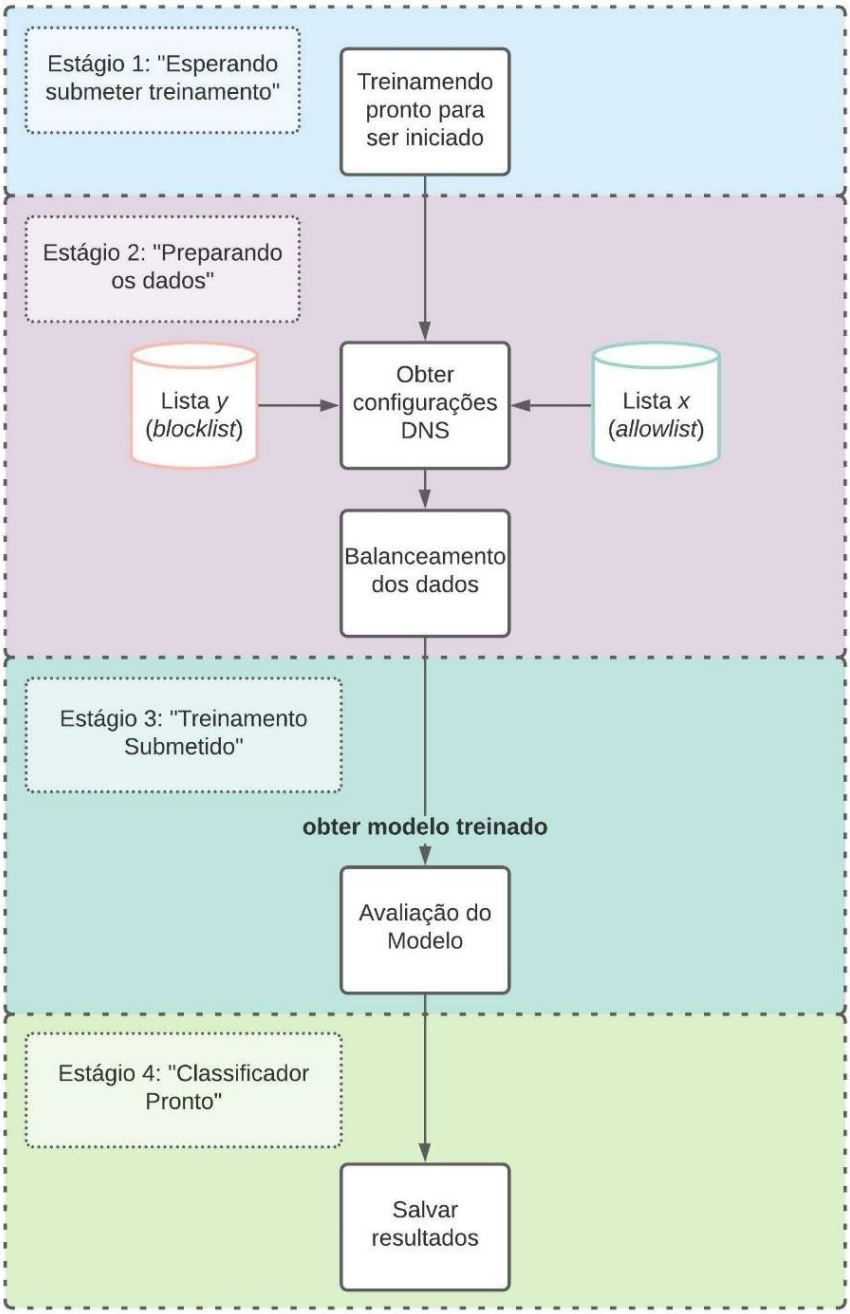
Estágio 5 - Erro na classificação: *status* reservado para modelos que não tiveram os seus treinamentos finalizados devido à ocorrência de erros durante o processo, o servidor DNS recursivo não está com acesso à Internet, inviabilizando a etapa de extração de características das configurações DNS dos domínios. É importante destacar que o modelo não

[9] Estrutura de dados em forma de tabela comumente utilizada durante o processo de treinamento de um modelo.

pode ser utilizado para a classificação, porém continua armazenado juntamente com o erro na API.

Com base nos estágios anteriores, é possível ilustrar o funcionamento do *pipeline* automatizado da API, o que pode ser observado na Figura 9.

Figura 9: *Pipeline* de Treinamento de um Classificador



Fonte: Elaborado pelo autor

A qualquer momento, a API é capaz de exibir os metadados dos modelos que se encontram nos estágios definidos anteriormente, quando solicitado por meio de uma

simples requisição HTTP. Além disso, destaca-se que, como a etapa de treinamento de um modelo pode demandar de minutos ou até mesmo horas para ser finalizado, após um treinamento ser solicitado pelo usuário, é enviada uma resposta para a requisição contendo o identificador único do modelo que acabou de ser submetido, para que a conexão cliente-servidor não fique ativada durante todo o processo.

3.4.3 Classificação de Domínios

Supondo que um modelo esteja pronto para realizar a classificação e que seja o padrão da API, o sistema deve ser capaz de exibir, para cada domínio da requisição, o resultado e a probabilidade do respectivo domínio ser legítimo ou malicioso, bem como informar qual modelo foi utilizado para a classificação. É possível especificar um modelo na requisição HTTP, mas, caso ele não esteja com o *status* definido como "Classificador Pronto", uma resposta deve informar o usuário que não é possível completar a classificação dos domínios.

3.5 Módulo de Gerenciamento de Logs

A URL base desse módulo é a "*logs*", e o seu objetivo é coletar e listar os erros que podem ocorrer durante a etapa de treinamento dos modelos com o seu respectivo *traceback*[10] completo. Separadamente, a API deve fornecer o *log* completo da aplicação para ser consultado, onde será possível encontrar dados gerais do sistema e contemplar se erros ocorreram em outros pontos da aplicação.

As informações descritas na Tabela 5 são armazenadas e ficam disponíveis para serem listadas a qualquer momento por meio da API. Quanto à edição dos dados, nenhum deles pode ser editado em momento algum, sendo estes disponíveis apenas para a visualização. Desta forma, todos os dados referentes a este módulo são controlados e preenchidos pela própria API, para fins de controle.

[10] Recurso oferecido pela linguagem de programação Python no qual é listado um relatório com toda a pilha de chamadas de função e respectivos erros que ocorreram durante a execução de um programa.

Tabela 5: Metadados dos logs

Dado	Descrição
Identificador único	Número que será utilizado para identificar cada um dos *logs* de forma individual e única
Descrição	Traceback completo do erro
Data criado	Data na qual o erro ocorreu
Modelo	Modelo no qual o erro ocorreu

Fonte: Elaborado pelo autor

3.6 Integração com o *Framework* DNS

Uma vez que, após o desenvolvimento da API ser concluído, é necessário que a mesma seja integrada com o *framework* DNS. Importante ressaltar que cada uma delas está hospedada em dispositivos diferentes, desta forma, cada máquina é acessada por meio de um IP distinto. Logo, foi necessário utilizar um formato leve para a troca de informações entre os dois sistemas por meio do protocolo HTTP. Nesse contexto, destaca-se a utilização da tecnologia JSON, que é capaz de estruturar e conter as informações da requisição, além da biblioteca *requests* disponível na linguagem de programação Python, como responsável por realizar e interpretar as requisições HTTP.

Após a implementação da comunicação entre os dois sistemas, algumas páginas inexistentes anteriormente também foram implementadas, sendo elas:

- Página de listas: página responsável por exibir as listas disponíveis na API, separadas por seu identificador. Também exibe os metadados mais relevantes, descritos na Tabela 4, em uma interface gráfica para o operador do *framework*. Por fim, a página oferece a possibilidade de gerenciar as listas alterando sua descrição ou excluindo-a;
- Página de modelos: página responsável por exibir todos os modelos disponíveis na API, separados por seu identificador. Além disso, os metadados mais relevantes, descritos na Tabela 4, são exibidos por meio de uma interface gráfica para o operador do *framework*. Por fim, existe a possibilidade de alterar o modelo padrão da respectiva API ou deletar algum modelo específico;

- Página de *logs*: página responsável por exibir todos os *logs* disponíveis na API, separados por seu identificador. Além disso, os metadados mais relevantes, descritos na Tabela 5, também são exibidos por meio da interface gráfica para o operador do *framework*. Por fim, a possibilidade de solicitar que um determinado *log* seja deletado;
- Página para submeter treinamento: página onde o usuário do *framework* é capaz de acionar o mecanismo de treinamento da API selecionando uma *blocklist* e uma *allowlist* disponível para o treinamento do modelo. Por fim, após acionar o mecanismo, a página exibe o identificador do modelo que acabou de ser submetido para o treinamento;
- Página para enviar uma nova lista: página responsável por receber os domínios informados pelo usuário do *framework* e enviá-los para a API mediante o fornecimento do tipo de lista e o arquivo contendo os domínios pelo usuário;
- Página para fundir duas listas: página responsável por exibir as listas disponíveis para serem fundidas, mediante interação do usuário do *framework* por meio de uma interface gráfica. A página possui uma validação para que apenas listas do mesmo tipo sejam selecionadas. Por fim, a lista resultante é informada ao usuário.

Além das páginas descritas anteriormente, a página de listagem de informações de um módulo foi modificada, foram adicionadas informações gerais sobre o respectivo módulo. Para cada página descrita nessa seção, o desenvolvimento seguiu as tecnologias que foram utilizadas no *framework* DNS até então, como: HTML, CSS e JavaScript.

3.7 Considerações Finais

Neste capítulo descreveu-se o funcionamento do projeto e das suas principais funcionalidades. Aproveitando-se de trabalhos anteriores do laboratório (POMPEU, 2018; BATISTA, 2020), foi possível destacar o funcionamento do *pipeline* de treinamento por meio de uma ilustração mostrando a interação entre o usuário e o *pipeline* por meio da API. Destacou-se quais metadados são armazenados, como o usuário deve interpretá-los e como foi realizada a integração com o *framework* DNS utilizando-se o protocolo HTTP e a tecnologia JSON.

Capítulo 4 - Resultados

Neste capítulo são apresentados os resultados obtidos com os experimentos realizados a partir da abordagem proposta neste trabalho. A fim de validar o método proposto, foram realizados testes em dois conjuntos distintos de dados contendo informações sobre configurações DNS maliciosas e legítimas. Nas seções 4.1, 4.2 e 4.3 são detalhadas as questões quanto aos recursos e operações que cada um dos três módulos: de listas, modelos e *logs* possuem, respectivamente. Na Seção 4.4 são apresentados e discutidos os resultados da integração da API com o *framework* DNS.

4.1 Módulo de Gerenciamento de Listas (MG de Listas)

Conforme destacado na seção 3.3, o objetivo deste módulo é listar, gerenciar, extrair as métricas dos domínios presentes nos conjuntos, e também versionar os diferentes conjuntos disponíveis para que sejam utilizadas no processo de treinamento de novos modelos.

É importante destacar a definição de um tipo de lista especial denominada conjunto de testes globais, que deve ser utilizada para a avaliação global dos modelos do sistema. Sua existência é única para cada tipo de conjunto rotulado, isto é, existe apenas um único conjunto de teste global para os domínios rotulados como maliciosos, e outro único para domínios rotulados como legítimos. Sua principal característica é a de possuir nomes de domínios rotulados e únicos no sistema, ou seja, os domínios que pertencem a esse conjunto de dados não estarão presentes em nenhum outro conjunto.

O sistema possui a capacidade de adicionar novos domínios a conjuntos pré-existentes. Para isso, um conjunto de novos domínios devem ser fornecidos para que sejam adicionados na operação. Ao final do processo, a API atualiza os metadados referentes à lista em questão, preenchendo a nova data de última atualização do conjunto.

Também é possível realizar a fusão de duas listas. Para isso, é preciso fornecer o identificador único das duas listas que serão fundidas, e é importante que as listas possuam o mesmo valor para o metadado *blocklist* e que estejam cadastradas na API. A partir disso, é gerada uma nova lista com um identificador único, de mesmo tipo e com os mesmos domínios das duas listas utilizas no processo. Por fim, na descrição do novo conjunto é adicionado um texto inicial destacando que se trata de uma fusão das anteriores.

Durante as operações de adição de novos domínios e o cadastro de uma nova lista, os novos dados são separados em dois conjuntos: o primeiro deles é a lista em si, e o segundo trata-se de um tipo de lista especial denominada conjuntos de testes. Ao todo, são duas listas especiais, sendo uma versão para *blocklist* verdadeiro e a outra versão para *blocklist* falso. Cada lista do conjunto de testes armazena domínios utilizados especialmente para a validação dos modelos que já foram treinados e, por isso, não contém domínios em comum com nenhuma das outras listas da API.

Por fim, como resultado das operações anteriores, torna-se possível armazenar e gerenciar diferentes conjuntos de dados que serão utilizados no treinamento de classificadores pela API.

4.1.1 Gerenciamento de domínios através do *pool*

Durante o desenvolvimento deste módulo, identificou-se a necessidade de um mecanismo capaz de gerenciar domínios sob um estado intermediário de observação. Neste estado, os domínios não estão rotulados como maliciosos ou legítimos, porém pretende-se anexá-los às listas disponíveis da API. Para que isso seja realizado com sucesso, o mecanismo precisa possuir a capacidade de armazenar as configurações DNS do respectivo domínio.

Foi desenvolvido um recurso no sistema capaz de realizar consultas DNS ativas para um determinado domínio armazenando os seus respectivos resultados. Durante os testes foram identificadas duas situações possíveis, sendo elas: o domínio ainda não foi

cadastrado no *pool*[11], neste caso a consulta DNS é realizada e a primeira entrada do domínio é adicionada na API; no segundo caso, o domínio já se encontra em observação no *pool*, para esta situação o resultado da consulta DNS é adicionada às consultas anteriores.

Uma vez que o sistema é capaz de armazenar os resultados de consultas DNS para um determinado domínio, é possível analisar e, posteriormente, definir se a configuração armazenada pode ser rotulada como práticas legítimas ou maliciosas. Na Figura 10, é possível observar o resultado para a solicitação de listar as consultas DNS de um determinado domínio. Destacam-se as datas nas quais as consultas foram realizadas, e os seus respectivos resultados nos campos “A”, “MX” e “SOA” que foram ocultados para melhorar a visualização.

Figura 10: Resultado da listagem de um domínio sob observação


```
[
  {
    "id": 6,
    "history": [
      {
        "date": "2021-10-08T12:45:02.299383Z",
        "A": [↔],
        "MX": [↔],
        "SOA": [↔]
      },
      {
        "date": "2021-10-08T12:45:36.329637Z",
        "A": [↔],
        "MX": [↔],
        "SOA": [↔]
      },
      {
        "date": "2021-10-08T12:45:38.038966Z",
        "A": [↔],
        "MX": [↔],
        "SOA": [↔]
      },
```


Fonte: Elaborado pelo autor

[11] Termo comumente utilizado para se referir a uma coleção de objetos.

Uma vez que um domínio presente no *pool* já pode ser rotulado, um algoritmo será o responsável por extrair as métricas das últimas consultas DNS presentes no seu histórico, salvando-as no respectivo conjunto de dados. O usuário pode solicitar que uma configuração específica, diferente da última coletada, seja armazenada no conjunto de dados rotulados por meio de uma requisição HTTP.

4.1.2 Resultados para o módulo de listas

Conforme detalhado na seção 3.3, com o armazenamento das listas, a API deve ser capaz de responder às solicitações de recursos do usuário. Durante os testes, foi possível utilizar-se do módulo de gerenciamento das listas para criar, listar e deletar conjuntos de dados rotulados pelo sistema. Na Figura 11 é possível observar o resultado de uma consulta das listas disponíveis pela API, no qual observa-se a presença dos metadados de uma lista, conforme foi detalhado na Tabela 3.

Figura 11: Resultado do JSON Retornado da Listagem de Listas


```
{
    "id": 18,
    "description": "",
    "created": "2021-01-31T11:11:01-03:00",
    "size": 40,
    "is_blocklist": true
},
{
    "id": 17,
    "description": "List that contains reserved domains for testing classifiers",
    "created": "2021-01-31T11:11:01-03:00",
    "size": 10,
    "is_blocklist": true
},
{
    "id": 16,
    "description": "",
    "created": "2021-01-31T11:10:45-03:00",
    "size": 40,
    "is_blocklist": false
},
{
    "id": 15,
    "description": "List that contains reserved domains for testing classifiers",
    "created": "2021-01-31T11:10:45-03:00",
    "size": 10,
    "is_blocklist": false
}
```


Fonte: Elaborado pelo autor

Além disso, o módulo possui operações bem definidas, que fornecem novas formas de introduzir conjuntos de dados sem perder os anteriores, o que pode ser feito por meio de três operações de listas bem definidas. O módulo também possui a capacidade

de adicionar domínios para um status intermediário de observação para que as suas configurações DNS sejam capturadas de forma ativa, o que oferece ao cientista de dados que opera o sistema uma nova forma de analisar e enriquecer os conjuntos de treinamento com cada vez mais dados.

O módulo é capaz de coletar ativamente as configurações DNS dos domínios que foram cadastrados na API, com destaque para a execução automática dessa funcionalidade quando novos domínios são cadastrados nas operações de cadastro de uma nova lista e adição de novos domínios em uma lista pré-existente.

Realizar a captura das configurações DNS previamente se deve ao fato de um domínio poder ser editado ou removido dos servidores DNS globais a qualquer momento, principalmente para domínios utilizados para práticas maliciosas que, a partir do momento em que as práticas são identificadas, são rapidamente removidos. Desta forma, mesmo que um domínio seja removido, a coleta foi realizada e armazenada anteriormente.

4.2 Módulo de Gerenciamento de Modelos (MG de Modelos)

Conforme destacado na seção 3.4, o módulo de gerenciamento de modelos é o responsável polos diversos modelos disponibilizados pela API, o que inclui selecionar os dados rotulados, extrair as suas respectivas métricas, submeter um modelo para o treinamento, extrair os resultados após a finalização do treinamento de um modelo, salvá-lo para que seja utilizado na classificação de domínios futuramente e, por fim, realizar a classificação dos domínios submetidos à API.

4.2.1 Análise de Desempenho de Classificadores

Uma vez que o *pipeline* torna possível a capacidade da API gerar diversos classificadores a partir de diferentes conjuntos de dados, no entanto, foi identificada a necessidade de avaliar o desempenho dos diferentes modelos presentes na API de forma contínua e automatizada. Uma vez que a matriz de confusão e a AUC são as principais formas de avaliação dos classificadores do sistema, foi desenvolvido um mecanismo responsável por obter essas métricas a qualquer momento para os classificadores que estejam com o *status* de “classificador pronto”.

Considerando que os conjuntos de testes possuem dados que nunca serão utilizados no treinamento de qualquer modelo e estão sempre evoluindo de tamanho na medida que novos conjuntos de dados são fornecidos para a API, torna-se válida a

utilização dos mesmos para a avaliação do desempenho de todos os modelos disponíveis pela API. Esse mecanismo é o responsável por identificar os melhores classificadores que o sistema possui naquele momento, por meio da obtenção das métricas AUC e matriz de confusão para o operador da API.

Uma vez que é possível consultar as métricas de cada um dos classificadores prontos, outras duas funcionalidades estão disponíveis para o operador da API:

- Troca de classificador em produção: trata-se de um recurso que consiste em alterar qual classificador, entre os disponíveis, deverá ser utilizado para classificar os domínios solicitados;
- Classificação de domínios: trata-se de um recurso responsável por responder a uma requisição com uma lista de domínios que devem ser classificados, nesta resposta encontra-se, para cada domínio, se o mesmo é malicioso ou legítimo e a probabilidade de ser malicioso.

4.2.2 Resultados para o módulo de modelos

Com a implementação da *pipeline* automatizada, conforme detalhado na seção 3.4, o módulo é capaz de gerar diversos classificadores a partir das listas disponíveis na própria API, de coletar e armazenar *queries* DNS ativamente, classificar domínios conforme solicitado o recurso e trocar facilmente o classificador que se encontra em produção. Além disso, durante os testes, foi possível utilizar-se do módulo para listar e deletar classificadores que estão cadastrados no sistema.

Na Figura 12 é possível observar o resultado de uma solicitação de listagem dos classificadores disponíveis pela API, no qual observa-se a presença dos metadados de um modelo, conforme detalhado na Tabela 4.

Figura 12: Resultado JSON de uma Consulta de Modelos pela API


```
{
    "id": 5,
    "blocklist_version": 19,
    "allowlist_version": 20,
    "confusion_matrix": "[[1, 0], [0,1]]",
    "auc": "1.000",
    "default": true,
    "date_created": "2021-03-19T19:14:06-03:00",
    "date_finished": "2021-03-19T19:23:44-03:00",
    "status": "Ready classifier"
},
{
    "id": 1,
    "blocklist_version": 18,
    "allowlist_version": 15,
    "confusion_matrix": null,
    "auc": null,
    "default": false,
    "date_created": "2021-03-01T10:17:24-03:00",
    "date_finished": "2021-03-31T10:20:33-03:00",
    "status": "Classification error"
},
```


Fonte: Elaborado pelo autor

4.3 Módulo de Gerenciamento de *Logs* (MG de Logs)

Conforme destacado na seção 3.5, o desenvolvimento do módulo de gerenciamento de *logs* deu-se em dois escopos, sendo eles: *logs* de treinamento de modelos, e *logs* da API como um todo, também denominado *logs* globais. Destaca-se que a linguagem Python possui bibliotecas nativas que viabilizam o rastreamento e tratamento das chamadas de funções realizadas pelo sistema.

Os *logs* referentes ao treinamento de modelos têm como objetivo capturar o *traceback* completo dos erros que podem ocasionalmente ocorrer durante o processo. Durante os testes, foi possível observar a captura de erros variados, com destaque para erros de má configuração do *pipeline* de treinamento. Além disso, foi possível extrair e armazenar cada erro de forma automática no banco de dados do sistema.

Os erros que ocorrem durante a etapa de treinamento de um modelo podem inviabilizar a sua construção ou até mesmo comprometer a sua utilização na classificação de domínios. Desta forma, o erro é vinculado ao modelo em questão e o mesmo tem o seu *status* ajustado para “Erro na classificação”, fazendo com que o modelo não seja utilizado

para possíveis classificações de domínios. Por fim, outras informações como a data e tempo no qual ocorreu também são armazenadas.

Em termos dos registros considerados na API de modo geral foi desenvolvida a funcionalidade de armazenar os principais acontecimentos do sistema. Durante o seu teste, foi possível capturar e armazenar informações relevantes que não são necessariamente erros do sistema, com destaque para os dados das requisições que estão sendo recebidas pela API. Tais dados são armazenados em um único arquivo, acessível apenas por meio da máquina na qual o sistema está sendo executado, uma vez que esses dados devem ser utilizados apenas para compreender e entender o melhor funcionamento da mesma.

Com a implementação do sistema que captura e armazena os *logs* do *pipeline* de treinamento, foi possível contemplar o objetivo detalhado na seção 3.5, uma vez que, ao ocorrer um erro durante o treinamento de qualquer modelo, é possível armazenar a pilha de execução completa, conforme suportado pela linguagem de programação Python. Além disso, ao adicionar o escopo de *logs* global da aplicação, foi possível capturar informações extras além de erros, como quais recursos estão sendo acessados, o código de *status* da resposta, bem como o tamanho do conteúdo devolvido para a requisição.

Na Figura 13, é possível observar um erro que ocorreu durante o treinamento de um determinado modelo, com destaque para os metadados de um *log*, conforme detalhado na Tabela 5.

Figura 13: Resultado JSON da Listagem de Erros


```
{
    "id": 3,
    "description": "'ListModel' object has no attribute 'filter' Traceback (most
    "date_created": "2021-10-21T10:44:47.504837-03:00",
    "classifier_version": 3
},
```


Fonte: Elaborado pelo autor

4.4 Resultado da Integração com o *Framework* DNS

A integração entre o *framework* DNS e a API foi feita por meio da linguagem de programação Python, uma vez que possui uma biblioteca nativa para os processos de comunicação entre diferentes serviços por meio da formulação, execução, interpretação e armazenamento de requisições HTTP. O objetivo da integração é permitir que as

funcionalidades apresentadas nas subseções anteriores sejam acionadas por intermédio de outra aplicação que possui uma interface gráfica, que no caso é o *framework* DNS.

O *framework* recebeu novas funcionalidades, bem como um mapeamento das operações internas para algoritmos que recebem, realizam e analisam requisições HTTP. O desenvolvimento deste mapeamento foi baseado em concordância com as demais funcionalidades que o mesmo já possuía, mantendo a alta modularização e legibilidade de código para as funções de comunicação, bem como a identidade visual dos componentes gráficos que o sistema já possui.

É importante destacar que o *framework* DNS já possuía a responsabilidade de armazenar os dados de usuários e dos módulos, autenticar os usuários, consumir APIs e exibir as suas interfaces. Durante os testes, foi possível observar que além das funcionalidades anteriores, o sistema agora possui a responsabilidade de realizar, armazenar e exibir os resultados para as solicitações dos novos recursos que a API possui, respectivos aos módulos de gerenciamento. Diante disso, foi definida uma ordem de execução para as novas operações em conjunto com as responsabilidades anteriores do *framework*, a qual deu-se da seguinte forma:

1. O usuário solicita a exibição de uma página Web;
2. O usuário solicita um recurso da API;
3. O *framework* identifica o recurso solicitado pelo usuário, bem como os seus dados, iniciando um processo de validação da requisição. Depois disso, é iniciada a comunicação com a API por meio de uma requisição HTTP;
4. A resposta para a requisição é recebida e interpretada pelo *framework*;
5. O resultado da operação é exibido na tela do usuário.

Com as novas responsabilidades definidas, foi possível iniciar a etapa de planejamento para a exibição das novas interfaces do sistema.

4.4.1 Estratégias de Exibição das Interfaces

Foram identificadas duas abordagens para a estratégia de exibição das páginas: a primeira consiste em exibir a página de uma vez, assim que o *framework* obtiver o resultado da API com todos os dados; e a segunda consiste em exibir a página com as informações mais importantes enquanto o conteúdo solicitado à API não chega. Durante

os testes, foi possível identificar as seguintes situações na qual a primeira abordagem é mais confortável para o operador do *framework*:

- Enquanto a solicitação do recurso é realizada pelo *framework* que permanece aguardando a resposta da API, a página solicitada pelo operador está sendo montada e exibida; desta forma, o resultado da operação será exibido assim que a resposta for enviada pela API;
- Dada a situação anterior, considerando o caso em que a API não respondeu à solicitação do recurso, a página solicitada é exibida ao usuário normalmente, assim que o *framework* detectar o erro da mensagem, uma tela de erro pode ser exibida para o operador informando-o sobre o problema que ocorreu na comunicação;
- Uma vez que a operação solicitada pelo usuário pode demandar de segundos ou até mesmo minutos para ser respondida, alguns elementos da página, que não dependem da requisição da API, podem ser exibidos para o operador, permitindo a visualização da página enquanto a resposta para a requisição é aguardada.

4.4.2 Interface Inicial das APIs

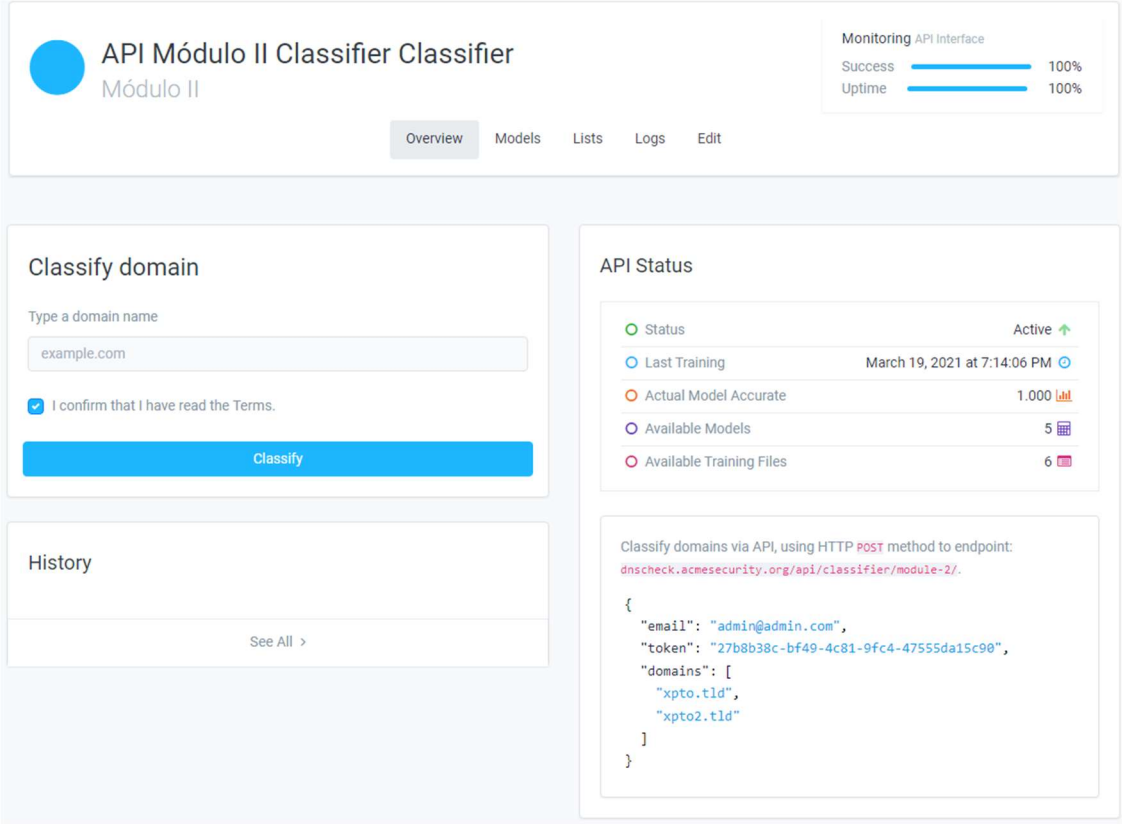
A interface inicial das APIs no *framework* DNS, recebeu esse nome dado o fato de ser a primeira página exibida para o usuário após selecionar a API que deseja gerenciar. Nesta interface, encontram-se um resumo das principais informações a respeito da respectiva API, além de possuir a funcionalidade de classificar um domínio. Conforme ilustrado na Figura 14, as seguintes informações são encontradas nesta interface:

- Status: indica se a API se encontra ativa, isto é, se a mesma está disponível para classificar domínios, gerenciar listas, modelos e logs;
- Último treinamento: indica quando foi realizado o último treinamento desta API, exibindo a data exata na qual o recurso foi solicitado pela última vez;
- Acurácia atual do modelo ativo: informa uma das principais métricas utilizadas para a avaliação de desempenho de um modelo de aprendizagem de máquina, métrica referente ao modelo que se encontra em produção, isto é, o modelo responsável pela classificação dos domínios;

- Modelos disponíveis: informa quantos modelos de aprendizagem de máquina a API possui e estão disponíveis para assumirem o modo de produção, isto é, a classificação de domínios;
- Arquivos de treinamento disponíveis: informa o resultado da soma de *allowlists* e *blocklists* que a API possui atualmente, isto é, quantos arquivos de treinamento estão disponíveis para serem utilizados nos treinamentos de novos modelos.

Caso a comunicação com a API não tenha sido completada, isto é, não foi possível obter as informações solicitadas, é exibida uma mensagem na tela do usuário notificando que não foi possível completar a requisição e as informações de *status* não são listadas.

Figura 14: Resultado da Interface da Página inicial da API



Fonte: Elaborado pelo autor

4.4.3 Interface de Gerenciamento dos Modelos

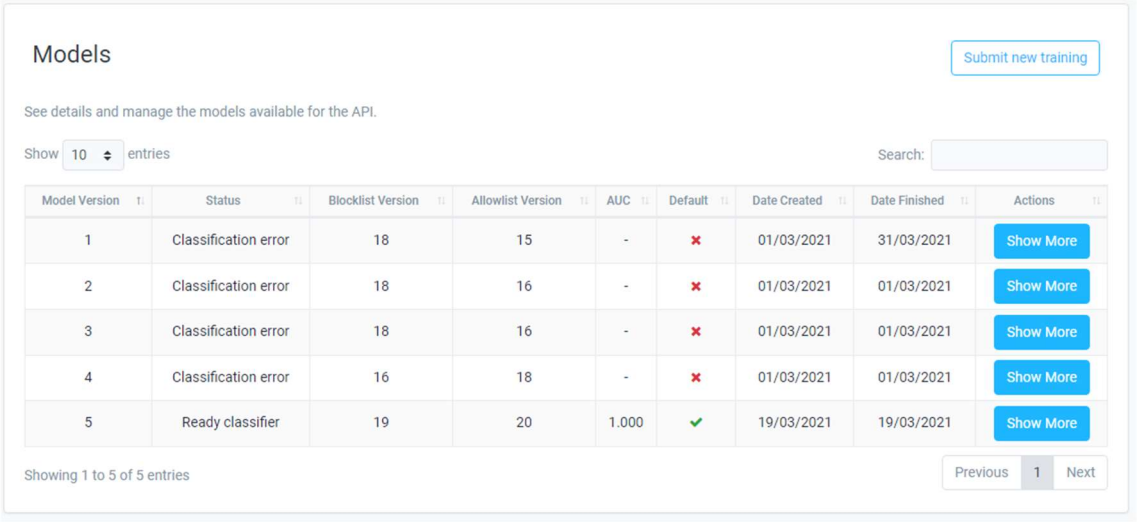
Foi desenvolvida uma interface responsável por tratar dos recursos do módulo de gerenciamento de modelos da API, com destaque para a capacidade de listar os metadados

mais importantes referentes aos modelos que foram cadastrados e estão disponíveis no sistema.

Após receber o resultado da requisição da API contendo os dados a respeito dos modelos disponíveis, os mesmos são filtrados e organizados em uma tabela, onde cada classificador é representado por uma linha. Nela estão localizados os dados considerados mais relevantes a respeito de um modelo, como a sua identificação, qual o seu *status* atual, quais conjuntos de dados rotulados foram utilizados durante o seu treinamento, o resultado da AUC obtida durante o seu treinamento, se o modelo é o padrão utilizado para a classificação dos domínios daquela API, em qual data o treinamento foi submetido e a data na qual foi concluído.

Desta forma, o conteúdo JSON observado na Figura 12 proveniente da consulta de modelos pela API foi distribuído na interface, conforme observado na Figura 15.

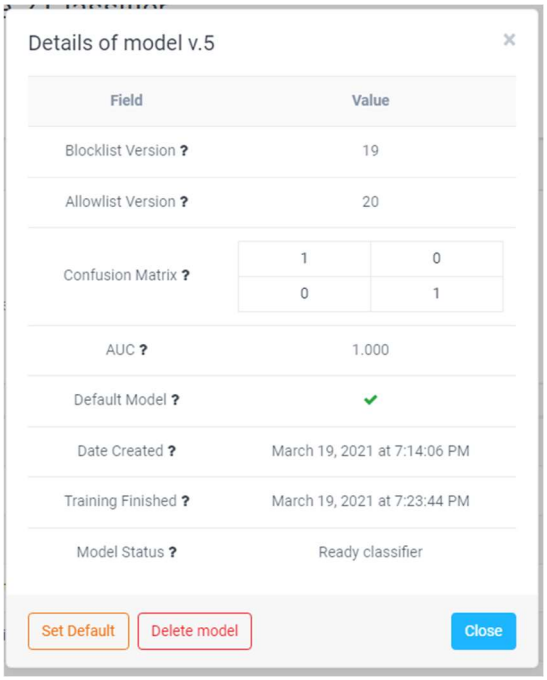
Figura 15: Resultado da Página de Listagem de Modelos



Fonte: Elaborado pelo autor

Uma vez que, para evitar excesso de conteúdo, apenas as principais informações do conjunto são exibidas diretamente na tabela, as informações secundárias foram separadas. Assim, foi adicionada uma janela adicional com as demais informações e recursos para cada entrada da tabela. Na Figura 16 pode ser observado o resultado obtido com a implementação da janela de detalhes de um modelo, com destaque para a representação da matriz da confusão obtida durante o seu treinamento.

Figura 16: Resultado da Janela de Detalhes do Modelo

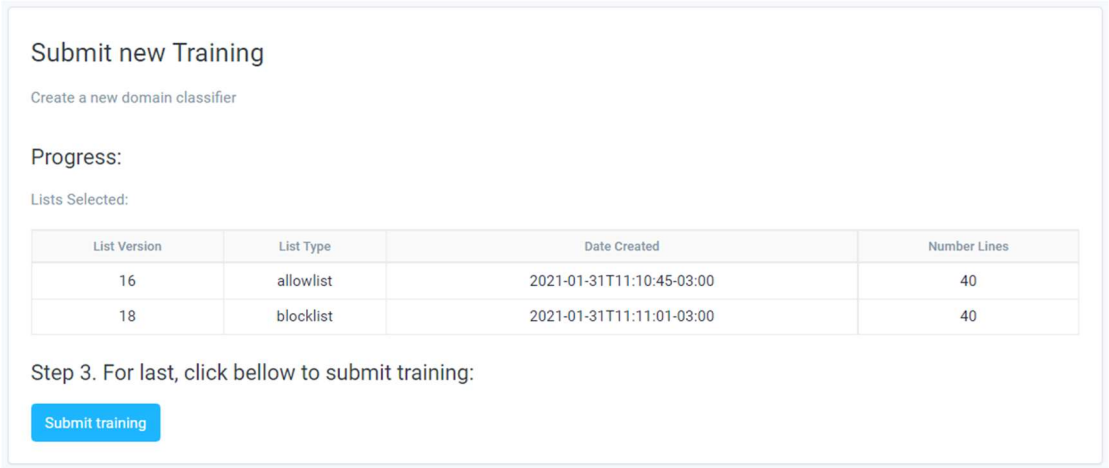


Fonte: Elaborado pelo autor

Ainda na Figura 16 é possível observar a presença de outros dois botões, o “Set Default” e o “Delete model” que, respectivamente, aciona o mecanismo de alterar o classificador que se encontra em produção para este modelo em específico, e aciona o mecanismo responsável por deletar aquele classificador.

Por meio do botão “Submit new training”, localizado na interface de listagem de modelos da Figura 15, é possível acessar a interface responsável pela submissão de treinamentos para a obtenção de novos modelos para a respectiva API.

Figura 17: Resultado da Interface de Submeter Treinamento



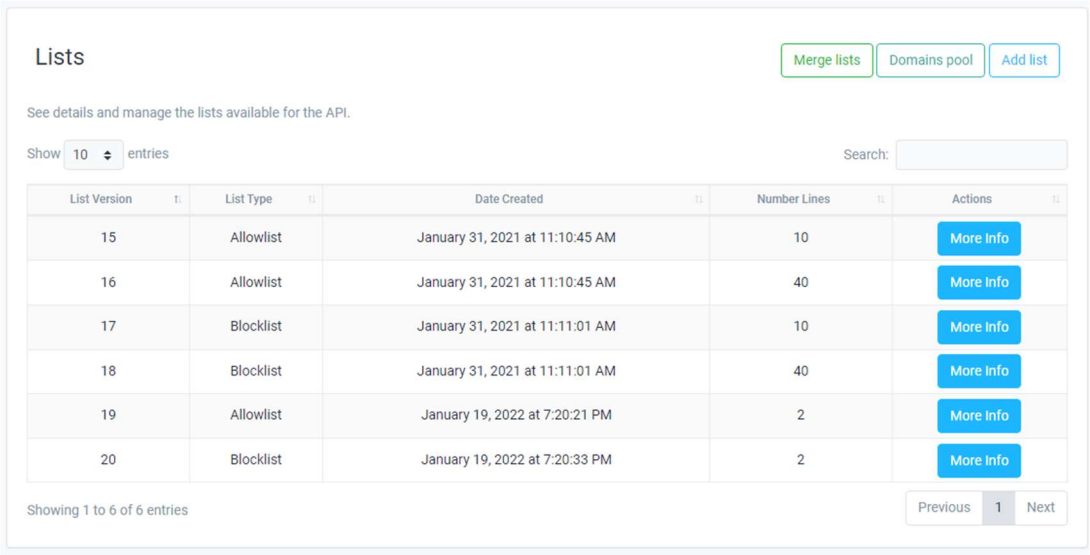
Fonte: Elaborado pelo autor

4.4.4 Interface de Gerenciamento das Listas

Foi desenvolvida uma interface responsável pelos recursos do módulo de gerenciamento de listas da API, com destaque para a capacidade de listar as informações mais importantes referentes aos conjuntos de dados utilizados para o treinamento de modelos disponíveis.

Os conjuntos são exibidos em uma tabela, onde cada um deles é representado por uma linha da tabela exibida na interface. Por meio dela, é possível identificar qual a versão do conjunto, o tipo de lista, a data na qual o conjunto foi criado e a quantidade de registros que ele possui. Desta forma, as informações contidas no JSON retornado pela API, que foi ilustrado na Figura 11, foram distribuídos pela interface de listagem de listas, conforme ilustrado na Figura 18.

Figura 18: Resultado da Interface de Listagem de Listas



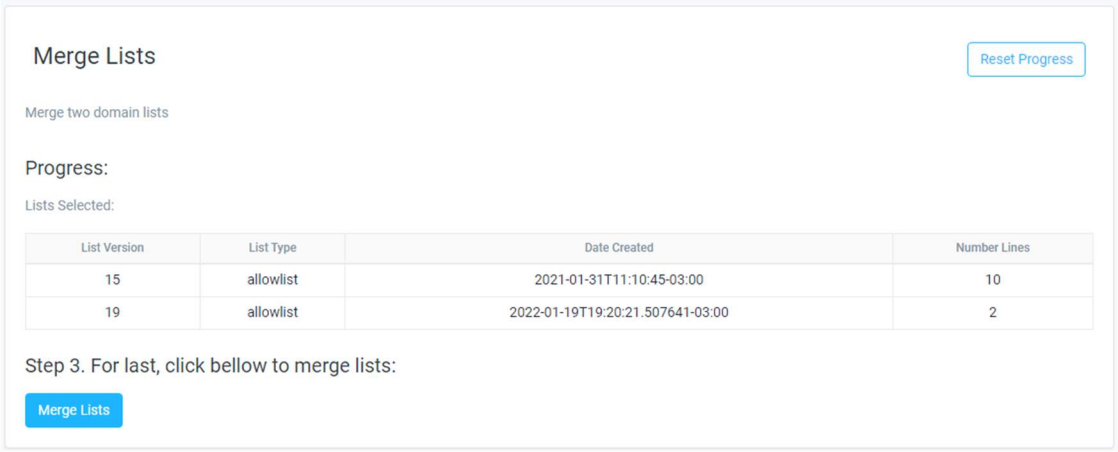
Fonte: Elaborado pelo autor

Ainda nessa interface, é possível observar o botão “Merge lists”; ele é responsável por redirecionar o operador para a interface que aciona a operação de combinar duas listas de mesmo tipo da API. Quando acessar essa interface, o operador deve selecionar o tipo de lista que deseja realizar a operação; depois disso, é aplicado um filtro para que uma tabela, antes oculta, seja exibida com apenas o tipo de lista selecionada pelo operador. Então, o operador poderá selecionar duas listas distintas que estão na tabela para serem unidas, durante todo o progresso um elemento é exibido mais acima indicando o progresso das listas selecionadas. Após selecionar as duas listas, apenas a tabela de

progresso permanece na tela, destacando a escolha do operador e um novo botão “Merge lists”, antes oculto, é exibido para confirmar a operação, conforme pode ser observado na Figura 19.

A operação de adicionar uma nova lista é selecionada por meio do botão “Add list” que, ao ser clicado, exibe uma janela para que o usuário selecione o arquivo contendo os novos domínios e o tipo de lista, sendo *allowlist* e *blocklist* as únicas opções disponíveis. Após o operador confirmar a operação fornecendo o arquivo e o tipo de lista, o *framework* DNS inicia um processo de extração de seu conteúdo para a forma de uma requisição HTTP para a API com todos os dados e, ao fim, exibindo uma seção com as informações da requisição, se foi concluída com êxito ou se houve algum erro.

Figura 19: Resultado da Interface de Combinar Listas

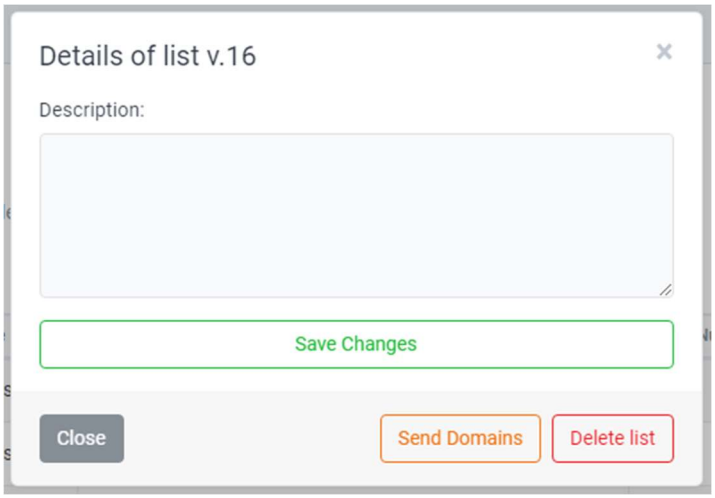


Fonte: Elaborado pelo autor

Uma vez que apenas as principais informações do conjunto são exibidas, as funcionalidades secundárias foram separadas para permitir a melhor visualização pelo operador de *framework*, por isso foi adicionada uma janela adicional com as demais informações e recursos para cada entrada da tabela. Conforme ilustrado na Figura 20, é possível observar a janela que se abre quando o operador aciona o botão “More info”, exibindo a descrição da respectiva lista e a opção de alterá-la, após clicar no botão “Save changes”.

Além disso, o botão “Delete list” pode ser observado, e, ao ser acionado, uma nova janela de confirmação é exibida para que o usuário confirme a ação de deletar a respectiva lista pela API. Ainda nesta janela, o botão “Close” é disponibilizado para caso o usuário deseje abortar a exclusão da lista.

Figura 20: Resultado da Janela de Detalhes de uma Lista



Fonte: Elaborado pelo autor

A terceira operação do módulo de gerenciamento de listas é acessada por meio da janela de detalhes de uma lista, onde observa-se o botão “Send domains”. Ao clicar nesse botão, é aberta uma nova janela na qual o operador do *framework* pode enviar um arquivo contendo os novos domínios que serão adicionados na lista por meio do recurso de adicionar novos domínios a uma lista existente da API.

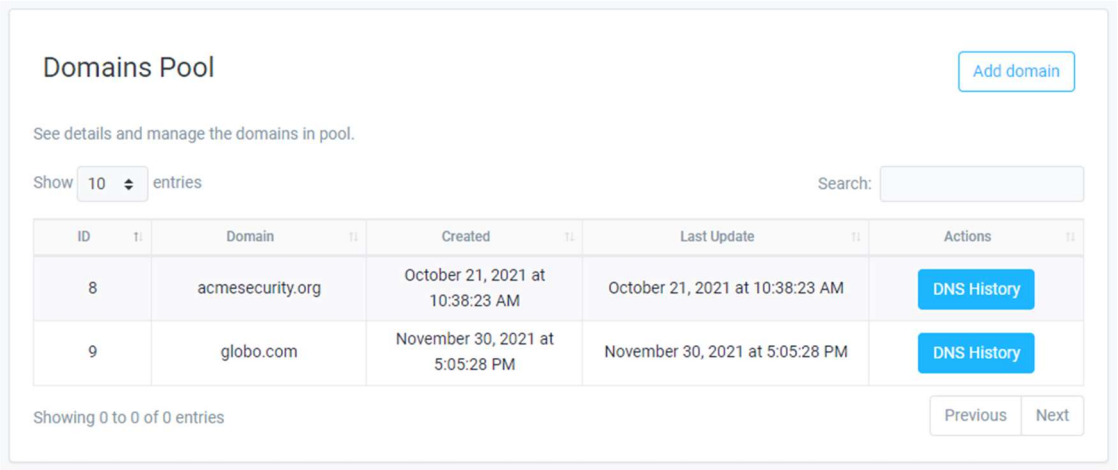
Durante os testes, foi possível fazer o upload de duas extensões de arquivos distintas, sendo elas “csv” e “txt”. Após o arquivo suportado ser selecionado e enviado, o *framework* é capaz de extrair os novos domínios e fazer a requisição para a API com os dados para serem adicionados e, ao completar a requisição, exibir o resultado da resposta da operação.

Na interface de listagem de listas, ilustrada na Figura 18, encontra-se o botão “Domains pool” responsável por direcionar o usuário até a interface de gerenciamento do *pool* de domínios. Ao clicar neste botão, o operador é redirecionado até uma página de listagem de domínios, contendo apenas os domínios que estão sob observação. Conforme ilustrado na Figura 21, na página de listagem de domínios no *pool* é possível observar uma tabela onde cada linha representa um domínio em observação naquele momento, bem como suas respectivas informações como o seu nome, data na qual a observação foi iniciada, data da última atualização que o mesmo recebeu e o botão “DNS history”, responsável por redirecionar o operador para um *dashboard* no qual é possível analisar o histórico de configurações DNS daquele domínio. Ainda nessa interface, também é possível observar o botão “Add domain”. Ao acioná-lo, é aberta uma janela com um

formulário onde o operador pode escrever o nome do domínio que deseja acionar ao *pool.* Após confirmada a operação, o mecanismo de captura de configurações DNS é acionado.

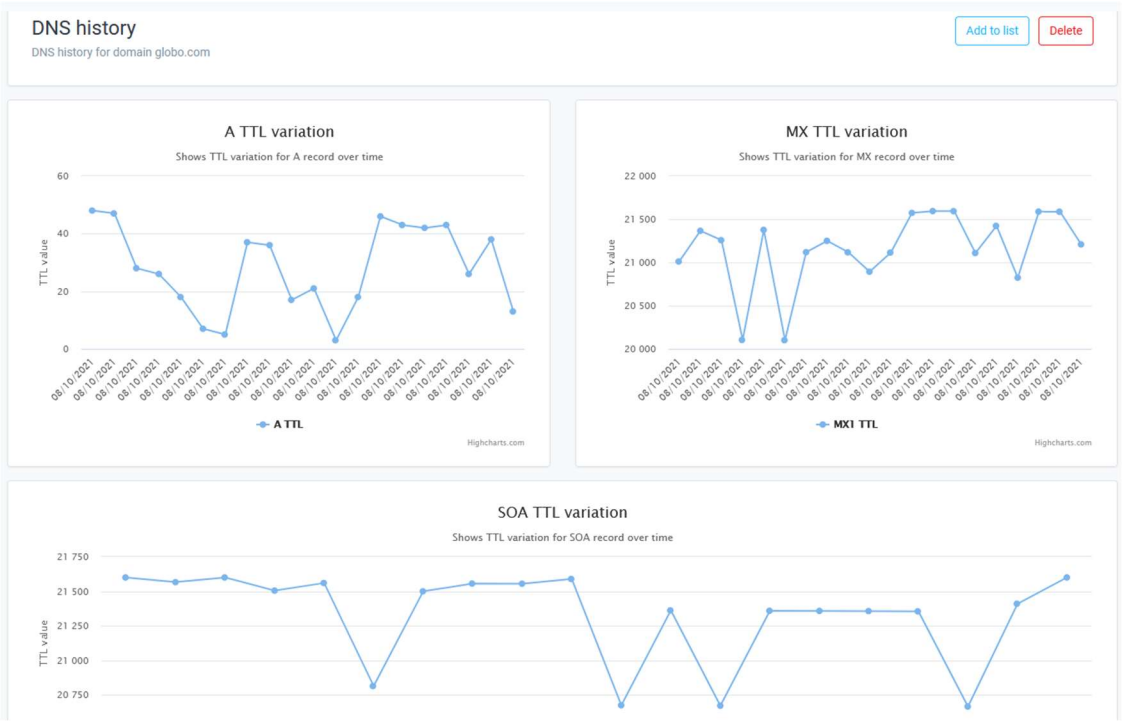
Inicialmente, no *dashboard* de histórico DNS é possível encontrar três gráficos de linha, que são responsáveis por auxiliar na visualização e compreensão da variação do campo TTL do domínio para cada uma das três *queries* DNS. No eixo horizontal, é possível observar a data daquela configuração; já no eixo vertical encontra-se o valor para o TTL registrado, desta forma os gráficos representam a variação da configuração de um domínio de acordo com o tempo, o que pode ser observado na Figura 22 e na Figura 23.

Figura 21: Interface de Listagem do Pool de Domínios



Fonte: Elaborado pelo autor

Figura 22: Interface Dashboard Pool de Domínios



Fonte: Elaborado pelo autor

Figura 23: Continuação Interface Dashboard Pool de Domínios

MX Records

Show 10 entries Search:

Date	TTL	Name	Type	Class	Answer	Preference
October 8, 2021 at 9:45:02 AM	21011	globo.com	MX	IN	mx.globo.locaweb.com.br.	5
October 8, 2021 at 9:45:02 AM	21011	globo.com	MX	IN	mx3.globo.locaweb.com.br.	20
October 8, 2021 at 9:45:02 AM	21011	globo.com	MX	IN	mx2.globo.locaweb.com.br.	10
October 8, 2021 at 9:45:36 AM	21366	globo.com	MX	IN	mx.globo.locaweb.com.br.	5
October 8, 2021 at 9:45:36 AM	21366	globo.com	MX	IN	mx3.globo.locaweb.com.br.	20
October 8, 2021 at 9:45:36 AM	21366	globo.com	MX	IN	mx2.globo.locaweb.com.br.	10
October 8, 2021 at 9:45:38 AM	21258	globo.com	MX	IN	mx.globo.locaweb.com.br.	5
October 8, 2021 at 9:45:38 AM	21258	globo.com	MX	IN	mx3.globo.locaweb.com.br.	20
October 8, 2021 at 9:45:38 AM	21258	globo.com	MX	IN	mx2.globo.locaweb.com.br.	10
October 8, 2021 at 9:45:40 AM	20104	globo.com	MX	IN	mx.globo.locaweb.com.br.	5

Showing 1 to 10 of 60 entries Previous 1 2 3 4 5 6 Next

SOA Records

Show 10 entries Search:

Date	NS	TTL	MBOX	Name	Type	Class	Retry	Expire	Serial	Min TTL	Refresh
October 8, 2021 at 9:45:02 AM	ns01.globo.com	21600	fapesp.corp.globo.com	globo.com	SOA	IN	3600	604800	2021100705	86400	10800
October 8, 2021 at 9:45:36 AM	ns01.globo.com	21566	fapesp.corp.globo.com	globo.com	SOA	IN	3600	604800	2021100705	86400	10800
October 8, 2021 at 9:45:38 AM	ns01.globo.com	21600	fapesp.corp.globo.com	globo.com	SOA	IN	3600	604800	2021100705	86400	10800

Fonte: Elaborado pelo autor

Na parte superior da *dashboard* é possível observar dois outros botões, conforme ilustrado na Figura 22. Ao acionar o botão “Add to list”, é aberta uma janela onde o operador pode selecionar qual das listas disponíveis na API receberá esse domínio; este botão aciona o mecanismo de inserção de um domínio do *pool* em uma lista. Já o botão “Delete”, ao ser acionado, exibe uma janela de confirmação para que o respectivo domínio seja excluído do *pool*, juntamente com o seu histórico.

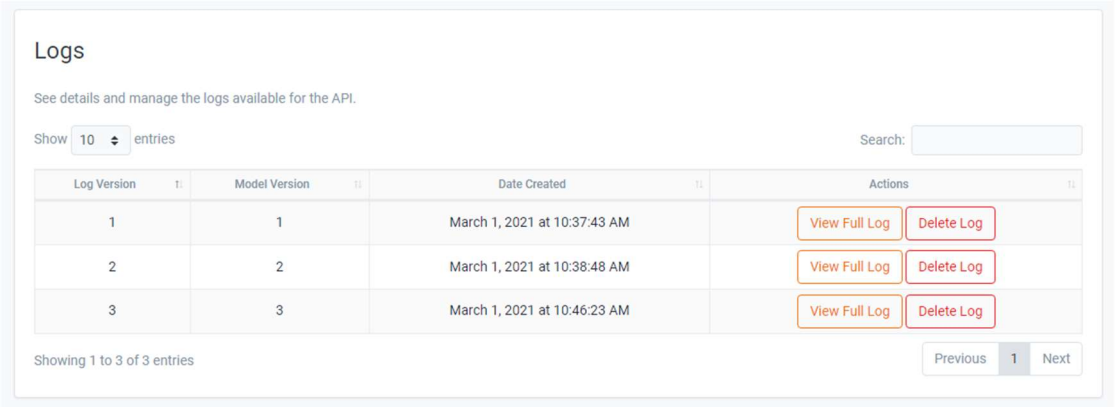
Após os gráficos do *dashboard*, são exibidas três tabelas para cada tipo de *query* DNS. Cada uma delas é representada por uma linha da mesma, onde são exibidos os resultados da consulta, bem como a sua data devidamente formatada e com maior precisão, se comparado aos gráficos.

4.4.5 Interface de Gerenciamento dos Logs

Foi desenvolvida uma interface de gerenciamento dos *logs* responsável por tratar dos recursos do módulo de gerenciamento de *logs* da API, com destaque para a capacidade de visualizar os *logs* do escopo de treinamento por meio do *framework* DNS.

Adotou-se a representação de cada um dos *logs* na forma de uma tabela, onde em cada linha estão representados os dados do erro, como o seu identificador único, o modelo que foi afetado e a data no qual ocorreu, referentes ao respectivo *log* que se encontra na resposta da API. Desta forma, os dados no formato JSON observados na Figura 13 foram distribuídos pela interface, conforme ilustrado na Figura 24.

Figura 24: Resultado da Página de Listagem de Logs



Fonte: Elaborado pelo autor

Ainda na Figura 24, observa-se que o *traceback* do erro não está sendo exibido na mesma tela. Isso deve-se ao fato de que os erros podem ser extensos, podendo levar a

problemas como a dificuldade de visualização e, consequentemente, de interpretação do seu conteúdo. Por conta disso, o erro completo pode ser visualizado em uma interface dedicada; para acessá-la, o operador deve acionar o botão “View full log” e então será direcionado. O resultado da interface de erro completo está ilustrado na Figura 25.

Figura 25: Resultado da Interface do Erro Completo

Fonte: Elaborado pelo autor

4.4.6 Testes Automatizados e Integração Contínua

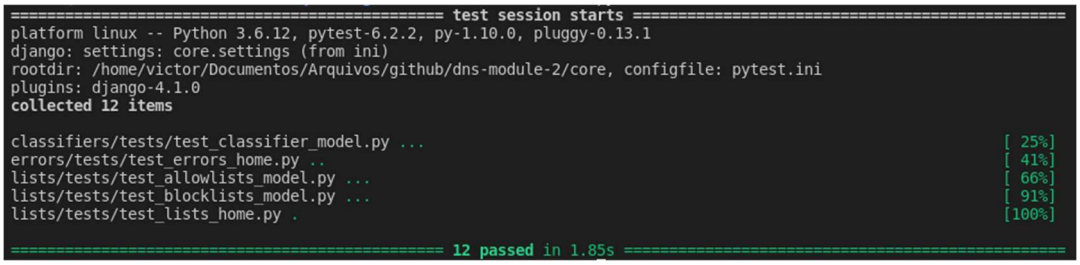
Baseando-se nos conceitos DevOps apresentados na seção 2.4, com destaque para a automação de processos, foram desenvolvidos testes automatizados para as principais funcionalidades descritas nas seções anteriores para serem executados juntamente com a API. Os seguintes testes foram implementados:

- Verificar se a listagem de listas, classificadores, erros, domínios do *pool* estão sendo exibidos com sucesso;
- Verificar se os *endpoints* estão retornando o código de status correto;
- Verificar se as listas estão sendo criadas ao receber uma requisição;
- Verificar se os dados estão sendo salvos no banco de dados após as solicitações.

Durante os testes do sistema, foi possível observar que os testes automatizados ajudaram a garantir que os recursos implementados estão funcionando da forma correta. Além disso, ao implementar novas funcionalidades no sistema, com o desenvolvimento dos testes antes das novas funcionalidades foi possível detectar erros previamente, como por exemplo erros de digitação que estavam interrompendo o processamento da requisição, e corrigi-los a tempo. Na Figura 26 está ilustrado a execução dos testes

executados no ambiente de desenvolvimento do projeto, onde é possível observar que foram realizados 12 testes com sucesso.

Figura 26: Resultado dos Testes Automatizados



Fonte: Elaborado pelo autor

Uma vez implementados, os testes foram configurados para serem executados juntamente com o repositório do código do projeto, que se encontra hospedado no site Github[12], por meio de uma ferramenta nativa disponibilizada pelo próprio site. Desta forma, a cada trecho de código adicionado à plataforma, os testes são executados automaticamente e os seus resultados são armazenados, permitindo a sua visualização por um curto período de tempo.

4.4.7 Considerações Finais

Com a descrição dos resultados apresentados nas subseções anteriores, pôde-se analisar a efetividade do sistema proposto nesta monografia. Nos testes realizados, o sistema de automatização de processos de treinamento na forma de API apresentou melhorias significativas em relação à realização automática de processos que antes seriam realizados de forma manual por um cientista de dados.

Desta forma, o sistema é capaz reduzir consideravelmente o esforço do cientista de dados na realização de atividades repetitivas, enquanto oferece ferramentas e dados relevantes para o processo de análise de resultados de modelos e de domínios.

[12] Plataforma online que permite a hospedagem e versionamento de códigos fonte de propósito geral.

Capítulo 5 - Conclusão

5.1 Conclusões Gerais

Diante da necessidade de classificar uma grande quantidade de domínios que são cadastrados mensalmente no Brasil, várias abordagens que utilizam modelos de aprendizado de máquina se mostraram promissoras. Diante disso, o laboratório ACME! propôs o desenvolvimento de uma aplicação intitulada *framework* DNS como a responsável por integrar três outros sistemas, onde cada um deles possui uma aplicação de algoritmos de aprendizado de máquina com parâmetros e configurações diferentes, em que todas elas possuem o objetivo em comum de classificar domínios como maliciosos ou legítimos.

Nesse contexto, identificou-se que o próximo grau de maturidade do *framework* DNS, segundo Makinen *et al*. (2021), quanto ao gerenciamento dos modelos de aprendizado de máquina, seria possuir a capacidade de gerenciar e versionar diversos modelos, bem como conjuntos de dados por meio de *pipelines* automatizados e controle de *logs*, práticas que recebem o nome de MLOps. Em outras palavras, no contexto deste trabalho, o *framework* DNS, que se encontrava anteriormente no grau centralizado em modelos, passou a operar no último grau de maturidade, que é o centralizado em *pipeline*.

Este trabalho empregou processos, técnicas e tecnologias atuais combinados de modo a desenvolver um sistema que suporte as novas práticas baseando-se nas estratégias propostas por Fursin *et al*. (2020), Akiba *et al*. (2020) e Arnold *et al*. (2020). Destaca-se

que os trabalhos relacionados foram fundamentais para o desenvolvimento do sistema, uma vez que as boas práticas destacadas foram utilizadas para atualizar o *framework* DNS, para que fosse capaz de gerenciar as APIs de forma remota, baseando-se na arquitetura API RESTful, em que cada recurso foi definido de acordo com as boas práticas adotadas e as especificações do protocolo HTTP.

Os resultados demonstram que os objetivos detalhados na seção 1.2 foram alcançados, pois por meio do módulo de gerenciamento de listas, é possível gerenciar múltiplas listas por meio de três operações definidas, sendo elas: adicionar um novo conjunto de treinamento, enriquecer um conjunto pré-existente a partir de novos dados, obter uma nova lista por meio de uma união entre dois conjuntos cadastrados no sistema e, por fim, realizando a consultas das configurações DNS com a finalidade de extrair as métricas de treinamento do respectivo domínio cadastrado no sistema durante as operações anteriores.

Por meio do módulo de gerenciamento dos modelos é acionado o *pipeline* de treinamento de novos classificadores e validação dos resultados obtidos dos mesmos. Além disso, este módulo é capaz de listar todos os metadados dos modelos e trocar o modelo utilizado no ambiente de produção. Com o módulo de gerenciamento de *logs* é possível obter de forma precisa qualquer erro que tenha ocorrido durante o *pipeline* de treinamento ou com o sistema de modo geral. Por fim, todos os recursos desenvolvidos no sistema proposto foram integrados ao *framework* DNS, onde o usuário é capaz de interagir com as APIs de forma remota e por meio de diversas interfaces gráficas.

Destaca-se que a implementação da API na forma do módulo II do *framework* DNS foi capaz de realizar, de forma automática, a coleta de configurações DNS de forma ativa, por meio do monitoramento de domínios no *pool* ou de novos domínios adicionados ao cadastro de um novo conjunto de dados. Com o monitoramento de domínios no *pool* é possível que mesmo domínios desativados naquele momento tenham a sua configuração DNS, que foi coletada anteriormente, mantida para fins de treinamento de novos modelos.

Além disso, destaca-se que o sistema pode ser facilmente adaptado para os demais módulos do sistema de classificação de domínios do laboratório ACME!, uma vez que a modularização e implementação dos componentes foi projetada para receber facilmente outros modelos de aprendizado de máquina bem como *pipelines* de maior complexidade, não exigindo necessariamente modificações no *framework* DNS.

5.2 Dificuldades Encontradas

A primeira dificuldade encontrada com o projeto foi o fato de o antigo módulo II que já integrava o sistema de classificações de domínios foi construído baseando-se em uma tecnologia específica, onde se identificou a existência de limitações quanto à modelagem, execução e a realização de consultas no banco de dados. Diante disso foi realizada uma pesquisa com a finalidade de trocar a tecnologia anterior por uma que possuísse suporte resolvendo os problemas encontrados até então e que, além disso, apresentasse uma ótima arquitetura para a modularização dos componentes posteriormente.

Outra dificuldade encontrada foi referente a execução do projeto no ambiente de testes. Realizou-se uma atualização da biblioteca de consultas DNS, em que havia a necessidade de passagem de um parâmetro obrigatório: o endereço IP do servidor DNS recursivo que será o responsável por realizar a consulta. Para resolver isso, localmente, especificou-se os endereços dos servidores de organizações como o Google e Cloudflare; já no laboratório uma máquina exclusiva foi configurada com o serviço *bind* como servidor DNS recursivo exclusivo do projeto.

Por fim, foram encontradas dificuldades relacionadas ao desenvolvimento do *pipeline* de treinamento automático, pois grande parte dos códigos foram retirados de arquivos notebooks[13], que não foram desenvolvidos para serem executados de forma automatizada. Para resolver essa dificuldade, foram realizados estudos juntamente com os demais companheiros de laboratório especialistas em trabalhar com aprendizado de máquina, juntamente com um aprofundamento maior das bibliotecas e a realização frequente de análises quanto ao resultado obtido em cada linha de código executada, foi possível alinhar com o que fora ilustrado na Figura 9.

5.3 Trabalhos Futuros

Como proposta para trabalhos futuros sugere-se a adaptação e utilização desta API para os demais módulos I e III dos sistemas de classificação de domínios utilizado no laboratório, uma vez que os resultados se mostraram sólidos e promissores quanto às funcionalidades implementadas e vantagens observadas em seu uso. Para isso, os sistemas

[13] Notebooks são arquivos que armazenam blocos de códigos para serem executados em uma determinada linguagem de programação

de extração de características, bem como as funções responsáveis pela construção do classificador, muito provavelmente, devem ser alterados para os novos contextos. É importante ressaltar que, mesmo diante dessa alteração, o *framework* DNS ainda será compatível com os módulos novos, sendo necessário adicioná-los por meio da interface de gerenciamento de APIs já disponível.

É esperado que no futuro as APIs possuam diversos modelos disponíveis para a classificação de domínios. Nesse contexto, visando uma melhoria do sistema proposto, é interessante que novas abordagens de gerenciamento dos modelos possibilitem o uso não apenas de um classificador em produção, como fora utilizado até este trabalho, mas sim de vários. Para isso, a abordagem sugerida é a de utilizar os *classifiers ensemble,* em que vários classificadores poderão ser configurados como "classifier default" e, por meio de uma abordagem estudada, aprimorar a qualidade de classificação de domínios.

Referências Bibliográficas

AKIBA, T.; SANO, S.; YANASE, T.; OHTA, T.; KOYAMA, M. Optuna: A next-generation hyperparameter optimization framework. **25th international conference on knowledge discovery & data mining**. 2019.

ANTONAKAKIS, M.; PERDISCI, R.; DAGON, D.; LEE, W.; FEAMSTER, N. Building a dynamic reputation system for dns. In: **Proceedings of the 19th USENIX Conference on Security**. Berkeley, CA, USA: USENIX Association, 2010. (USENIX Security'10), p. 18–18. ISBN 888-7-6666-5555-4.

ARNOLD, M.; BOSTON, J.; DESMOND, M.; DUESTERWALD, E.; ELDER, B.; MURTHI, A.; NAVRATIL, J.; REIMER, D. Towards automating the AI operations lifecycle. **arXiv preprint** arXiv:2003.12808. 2020.

BASNET, R. B.; SUNG, A. H.; LIU, Q. Learning to detect phishing urls. International **Journal of Research in Engineering and Technology**, v. 3, n. 6, p. 11–24, 2014.

BATISTA, V. B. **Framework Para Integração e Automação de Modelos de Aprendizado de Máquina Para Classificação de Nomes De Domínio**. Monografia (Bacharelado em Ciência da Computação). Universidade Estadual Paulista "Júlio De Mesquita Filho", São José do Rio Preto, 2020.

BILGE, L.; SEN, S.; BALZAROTTI, D.; KIRDA, E.; KRUEGEL, C. Exposure: A passive dns analysis service to detect and report malicious domains. **ACM Trans. Inf. Syst. Secur.**, ACM, New York, NY, USA, v. 16, n. 4, p. 14:1–14:28, abr. 2014. ISSN 1094-9224.

BRAY, T., ED. **The JavaScript Object Notation (JSON) Data Interchange Format.** STD 90, RFC 8259, DOI 10.17487/RFC8259. 2017. Disponível em: <https://www.rfc-editor.org/info/rfc8259>. Acesso em: 20 de jul. de 2021.

DAN, K.; KITAGAWA, N.; SAKURABA S.; YAMAI, N. Spam Domain Detection Method Using Active DNS Data and E-Mail Reception Log. **IEEE 43rd Annual Computer Software and Applications Conference**. pp. 896-899. doi: 10.1109/COMPSAC.2019.00133. 2019.

Endereçamento IPv6. IPv6.br. 2011. Disponível em: <http://ipv6.br/post/enderecamento-ipv6/>. Acesso em: 21 de jul. de 2021.

FAWCETT, T. **ROC Graphs: Notes and Practical Considerations for Researchers BT** - Tech Report HPL-2003-4, HP Laboratories. p. 1–38, 2004. Disponível em: <http://binf.gmu.edu/mmasso/ROC101.pdf>. Acesso em: 10 de set. de 2021.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. Doctoral dissertation, University of California, Irvine. 2000. Disponível em: <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>. Acesso em: 22 de jul. de 2021.

FIELDING, R. T.; RESCHKE, J. **Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content**. Disponível em: <https://www.rfc-editor.org/rfc/rfc7231.html>. Acesso em: 20 de jul. de 2021.

FIELDING, R.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, P.; BERNERS-LEE, T. **Hypertext Transfer Protocol -- HTTP/1.1** RFC 2616, DOI 10.17487/RFC2616. 1999. Disponível em: <https://www.rfc-editor.org/info/rfc2616>. Acesso em: 21 de jul. de 2021.

FITZGERALD, B.; STOL, K. Continuous software engineering: A roadmap and agenda. **Journal of Systems and Software**, v. 123, p. 176-189, 2015.

FURSIN, G.; GUILLOU, H.; ESSAYAN, N. CodeReef: an open platform for portable MLOps, Reusable Automation Actions and Reproducible Benchmarking. 2020. **The Workshop on MLOps Systems, MLSys'20**. arXiv:2001.07935.

GÉRON, A. **Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems**. O'Reilly Media, 2019.

GRANLUND, T.; KOPPONEN, A.; STIRBU, V.; MYLLYAHO, L.; MIKKONEN, T. MLOps Challenges in Multi-Organization Setup: Experiences from Two Real-World Cases. **arXiv preprint** arXiv:2103.08937. 2021.

HARRENSTIEN, K.; STAHL, M.K.; FEINLER, E.J. RFC-EDITOR. **DoD Internet host table specification**. Disponível em: <https://www.rfc-editor.org/rfc/rfc952.html>. Acesso em: 21 de jul. de 2021.

Information for Registrars. ICANN. Disponível em: <https://www.icann.org/resources/pages/registrars-0d-2012-02-25-en>. Acesso em: 20 de jul. de 2021.

Integração o que é api rest. REDHAT. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api 24/07/2021>. Acesso em: 24 de jul. de 2021.

Interface de programação de aplicações o que é API. REDHAT. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces>. Acesso em: 23 de jul. de 2021.

Introducing JSON. JSON.ORG. Disponível em: <https://www.json.org/json-en.html>. Acesso em: 20 de jul. de 2021.

JORDAN, M. I.; MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. **Science**, v. 349, n. 6245, p. 255-260, 2015.

KOPECKÝ, J.; FREMANTLE, P.; BOAKES, R. A history and future of Web APIs. **it-Information Technology**, v. 56, n. 3, p. 90-97. 2014.

KOTSIANTIS, S. B. Supervised Machine Learning: A Review of Classification Techniques. **Informatica**, v. 31, p. 249–268, 2007. ISSN 09226389.

KOUNTOURAS, A.; KINTIS, P.; LEVER, C.; CHEN, Y.; NADJI, Y.; DAGON, D.; ANTONAKAKIS, M.; JOFFE, R. Enabling network security through active DNS datasets. In: Research in Attacks, Intrusions, and Defenses - **19th International**

Symposium, RAID 2016, Paris, France, September 1921, 2016, Proceedings. [S.l.: s.n.], 2016. p. 188–208.

KULIKOVA T.; SIDORINA T.; SHCHERBAKOVA T. **Spam and phishing in Q2 2020**. SECURELIST. 2020. Disponível em <https://securelist.com/spam-and-phishing-in-q2-2020/97987/>. Acesso em: 29 de jul. de 2021.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a internet: uma abordagem top-down**. 6th. ed. [S.l.]: Pearson, 2012. ISBN 0132856204.

MAKINEN, S.; SKOGSTROM, H.; LAAKSONEN, E.; MIKKONEN, T. Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help?. **arXiv preprint** arXiv:2103.08942. 2021.

MLOps: Continuous delivery and automation pipelines in machine learning. GOOGLE. 2021. Disponível em: <https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>. Acesso em 26 de jul. de 2021.

MOCKAPETRIS. RFC-EDITOR. Domain names - concepts and facilities. Disponível em: <https://www.rfc-editor.org/rfc/rfc1034.html>. Acesso em: 21 de jul. de 2021.

POMPEU, A. **Detecção de Domínios Maliciosos Por Meio de Aprendizado de Máquina e Coleta Ativa de DNS**. Monografia (Bacharelado em Ciência da Computação). Universidade Estadual Paulista "Júlio De Mesquita Filho", São José do Rio Preto, 2018.

POSTEL, J. RFC-EDITOR. **Internet Protocol**. 1981. Disponível em: <https://www.rfc-editor.org/rfc/rfc791.html>. Acesso em: 18 de jul. de 2021.

RASCHKA, S. Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. **arXiv**, 2018. arXiv:1811.12808.

REGISTRO.BR. **Estatísticas**. Disponível em <https://registro.br/dominio/estatisticas/>. Acesso em: 19 de jul. de 2021.

RENGGLI, C.; RIMANIC, L.; GÜREL, N. M.; KARLAŠ, B.; WU, W.; ZHANG, C. A Data Quality-Driven View of MLOps. **arXiv preprint** arXiv:2102.07750. 2021.

SATO, D. **DevOps Na prática: entrega de software confiável e automatizada**. Casa do Código. 2013.

SERVERS.ORG root. **root-servers.org**. 2021. Disponível em: <https://root-servers.org/>. Acesso em: 21 de jul. de 2021.

SILVEIRA, M. R., CANSIAN, A. M.; KOBAYASHI, H. K. Detection of Malicious Domains Using Passive DNS with XGBoost. **IEEE International Conference on Intelligence and Security Informatics**. 2020. pp. 1-3, doi: 10.1109/ISI49825.2020.9280552.

SPOOREN, J; VISSERS, T.; JANSSEN, P; JOOSEN, W.; DESMET, L. Premadoma: an operational solution for DNS registries to prevent malicious domain registrations. **ACSAC '19: Proceedings of the 35th Annual Computer Security Applications Conference**. 2019. 557-567. 10.1145/3359789.3359836.

STEVENS, W. R. **TCP/IP Illustrated (Vol. 1): The Protocols**. USA: Addison-Wesley Longman Publishing Co., Inc., 1993. ISBN 0201633469.

TAMBURRI, D. A. Sustainable MLOps: Trends and Challenges. **22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing.** IEEE, 2020. p. 17-23. 2020.

The Domain Name Industry Brief. VERISIGN. 2021. Disponível em: <https://www.verisign.com/assets/domain-name-report-Q12021.pdf>. Acesso em: 17 de jul. de 2021.

WEISS, E. A. Biographies: Eloge: Arthur lee samuel (1901-90). **IEEE Annals of the History of Computing**, v. 14, n. 3, p. 55–69, 1992. ISSN 1058-6180.

What Does ICANN Do. ICANN. Disponível em: <https://www.icann.org/resources/pages/welcome-2012-02-25-en>. Acesso em: 22 de jul. de 2021.