Dockerização do Sofware EpiBuilder-2.0

Richard Matheus Miranda[1], **Vinícius Rossi Waltrick**[1], **Renato Simões Moreira**[1]

[1]Instituto Federal de Santa Catarina (IFSC) - Câmpus Lages
Curso Superior de Ciência da Computação
R. Heitor Villa Lobos, 225 - São Francisco, Lages - SC, 88506-400

Abstract. *Epitopes are portions of a protein that are reconignized by antibodies. For the study and analysis of these epitopes bioinformatics tools are used, and sometimes more than one is needed, therefore generating many steps of data manipulation to achieve satisfying results. This article aims to contextualize the reader about epitopes and tools for analysis, also to present a solution for the use of diverse tools required for the analyzes.*

Resumo. *Epítopos são porções de proteínas reconhecidas por anticorpos. Para o estudo e análise desses epítopos faz-se o uso de ferramentas da bioinformática, por vezes envolvendo o uso de mais de uma ferramenta, e com isso gerando várias fases de manipulação de dados para ter os resultados desejados. Este artigo visa contextualizar o leitor sobre epítopos e ferramentas para análise, bem como apresentar uma solução para o uso das variadas aplicações requeridas para análises.*

1. Introdução

A bioinformática é uma área que busca integrar ferramentas e tecnologias da informática com os métodos e conhecimentos da biologia, tendo por objetivo facilitar os processos de análise de genes[1], proteínas[2], aminoácidos[3] e outros. Dentro desse escopo, ocorrem análises que possibilitam a melhoria na vacinologia, ampliando a capacidade de investigar e avaliar predições (Pereira et al., 2021).

Durante as análises de dados com bioinformática, testes são realizados para validar e comprovar a eficiência de novas vacinas, no qual os epítopos (pequenas porções imunogênicas de proteínas) são fundamentais durante esse processo pois, eles são reconhecidos por anticorpos e, seu estudo, representa um grande avanço na área de bioinformática chamada imunoinformática. Os epítopos são amplamente utilizados no desenvolvimento de vacinas e testes de diagnóstico, possibilitando verificar a resposta imune dos anticorpos e atuar como uma forma de conexão entre o sistema de defesa do corpo contra micro-organismos invasores (Larsen et al., 2006). A identificação de epítopos de células B é de grande interesse em aplicações biotecnológicas e clínicas, como desenhos de vacinas atenuadas ou de subunidades e desenvolvimento de anticorpos terapêuticos. Sua identificação, no entanto, é um processo caro e demorado que requer uma extensa triagem de ensaios experimentais (Clifford et al., 2022). Conforme demonstrado pela Figura 1 que demonstra um fluxograma do processo para desenvolvimento de um medicamento.

[1]O gene é um segmento de uma molécula de DNA (ácido desoxirribonucleico), responsável pelas características herdadas geneticamente.

[2]Composto orgânico composto por aminoácidos.

[3]Moléculas orgânicas compostas por uma carboxila e um amino.

Durante a etapa de análises utilizando ferramentas computacionais, existe uma gama de programas que apoiam na predição e análise de epítopos de *células B* (BCE), como o *ABCPred*, *SVMTrip*, *EpiDope*, *EpitopeVec*, *BepiPred-2.0* e *BepiPred-3.0*. Entretanto, o processo de predição ainda é composto por etapas complexas que demandam conhecimento técnico e esforço de seus usuários. Para ajudar a sanar este problema surgiu o *EpiBuilder*, um software que tem como objetivo auxiliar na montagem, classificação e busca de epítopos usando os resultados de saída do *BepiPred-3.0*. O *EpiBuilder 1.0* traz melhorias para os processos que dependiam da utilização de outras aplicações de predição, otimizando a execução de diversas características que anteriormente só eram obtidas individualmente, o que demandava um esforço maior para chegar a um resultado (Moreira et al., 2022).

Visto que o software demanda alguns conhecimentos técnicos em tecnologias de predição, o objetivo deste trabalho tem como ponto principal a diminuição da complexidade da instalação do preditor *BepiPred-3.0*. Será aplicado a utilização de *Docker* para que que o processo de instalação seja facilitado e seu funcionamento integrado a nova versão do *EpiBuilder*.

Para que esse objetivo seja alcançado foram definidos três objetivos específicos:

- Desenvolver *Docker Images* com *Docker*;
- Integrar o *Epibuilder* e suas dependências na Docker Image; e
- Automatizar a execução do Epibuilder com *Scripts*.

Em relação à metodologia, este trabalho se classifica como uma pesquisa aplicada e foi dividida em quatro partes. A primeira parte é um estudo a cerca da utilização do *EpiBuilder 2.0*, do estudo do contexto sobre epítopos e também do estudo de ferramentas para conteinerização. A segunda parte é o levantamento dos requisitos e ferramentas necessárias para a elaboração de todos os objetivos descritos acima. A terceira parte é a realização da integração com o *Bepipred-3.0*. A quarta é a inclusão ao *Docker* e criação do *Dockerfile*.

2. Referencial Teórico

Nessa seção são apresentados os conceitos que fundamentam a teoria referente ao sistema proposto, bem como os componentes utilizados em seu aprimoramento. A subseção 2.1 apresenta uma explicação sobre bioinformática e suas subseções conceitualizam tópicos importantes na compreensão do artigo. A subseção 2.2 aborda o software *Epibuilder 1.0* e sua funcionalidade. A subseção 2.3 e suas subseções apresentam *softwares* de bioinformática e tecnologias relevantes ao desenvolvimento do trabalho. Por fim, a subseção 2.4 apresenta as *strings de busca.*

2.1. Bioinformática

Segundo de Araújo et al. (2008), a bioinformática integra conhecimentos em diferentes áreas do conhecimento com objetivo de analisar o código genético existente nas biomoléculas pelo estabelecimento de modelos lógico-matemáticos e estatísticos. Com as pesquisas nesta área é possível resolver diversos eventos biológicos, tendo como o primeiro passo o sequenciamento de genes e proteínas, podendo-se gerar bases de dados que acumulem essas informações e permitam que se pratique melhores modos de exibir os dados coletados e estudos sobre os dados dispostos com o objetivo de avanços na

áreas biológicas. O fluxograma apresentado na Figura 1 demonstra a presença da bioinformática na primeira etapa da pesquisa, sinalizando a importância da área para o mundo atual.

Figura 1. Fluxograma do desenvolvimento de um medicamento para combate a um antígeno

2.1.1. Epítopo

Um epítopo é um peptídeo, também chamado de determinante antigênico, é uma porção de um proteína reconhecida pelos anticorpos do corpo, células B ou células T (Kerpan et al., 2020). Pode se classificado como o agente que realiza a ligação entre os antígenos e o anticorpos, utilizando como conexão uma molécula do anticorpo chamado parátopo (Garrett, 2003).

Conforme citado no parágrafo acima, através da Figura 2, é possível visualizar uma representação gráfica da ligação entre o epítopo e o parátopo.

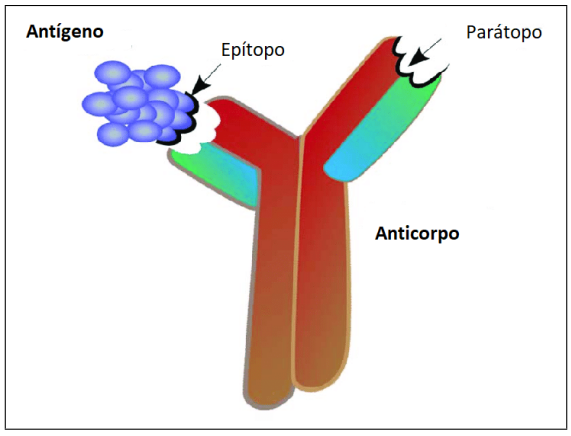

Figura 2. Epítopo se conectando ao Parátopo

A Figura 2 apresenta o processo de conexão entre o epítopo e o parátopo, onde o anticorpo entra em contato com o antígeno para realizar a sua neutralização. Essa conexão possui como característica a tentativa do anticorpo se anexar ao agente invasor, que pode falhar caso a quantidade de epítopos não seja suficiente para neutralizá-lo.

2.1.2. Células B

As células B, ou linfócitos B, são glóbulos que atuam como um componente no sistema imune adaptativo, atuando através da liberação de anticorpos (Kilsen, 2020). O resultado da liberação dos anticorpos se caracteriza pelo início da neutralização do organismo invasor, chamado de antígeno.

2.1.3. Proteínas

As proteínas são compostos de elevada massa molecular (5.000 a vários milhões de mols) produzidas pelas células vivas de todas as formas de vida. São polímeros naturais complexos de aminoácidos, unidos entre si por um tipo específico de ligação covalente – a ligação peptídica (MOTTA, 2010).

2.1.4. Aminoácidos

Aminoácidos são as unidades básicas da composição de uma proteína. Em humanos, nove aminoácidos são considerados essenciais, uma vez que não podem ser sintetizados endogenamente (Rogero e Tirapegui, 2008).

2.2. Epibuilder 1.0

O *Epibuilder* é um software *user friendly* que auxilia na montagem de epítopos, classificando e pesquisando-os através dos resultados de entrada do *BepiPred-2.0*. O software pode pesquisar toda a sequência de epítopos em vários arquivos *FASTA* e permite usar o *BLASTP* para identificar epítopos que eventualmente tenham variações de sequência. O *EpiBuilder 1.0* foi desenvolvido usando a linguagem de programação *Java* (suportada pela versão 1.8) com uma interface gráfica do usuário (GUI) e opções de linha de comando (CLI), sem nenhum pré-requisito adicional (além do *Java Runtime Environment*) para executar todas as funções, exceto *BLASTP*, se estiver definido. O *EpiBuilder* está preparado para ser executado em qualquer sistema operacional que suporte a tecnologia *Java*, como *Windows* e sistemas baseados em *Unix* (Moreira et al., 2022). Através do *Epibuilder*, é possível automatizar o trabalho manual de tratar individualmente cada resultado do *BepiPred-2.0*, que é salvo em uma planilha única para cada predição realizada.

2.3. Linguagens e Tecnologias Utilizadas

2.3.1. Java

A linguagem de programação utilizada no desenvolvimento do *Epibuilder* é o *Java*. A escolha se deu pelo conhecimento da equipe e por sua escalabilidade em sistemas operacionais, sendo possível executar a aplicação em qualquer sistema por conta de ser uma linguagem de programação híbrida.

O *Java* é uma linguagem de programação de alto nível *orientada a objetos* criado pela *Sun Microsystems* no ano de 1995 (Nery, 2017). Essa linguagem possibilita o programador desenvolver tanto aplicações *Desktop* e *Web* robustas, possuindo um bom desempenho em aplicações de qualquer tamanho e arquitetura.

Uma linguagem de programação híbrida é uma aplicação compilada e interpretada onde o código-fonte gerado pelo programador é executado em um ambiente de execução através de uma conversão de *bytecodes* em códigos executáveis em nosso ambiente (Claro e Sobral, 2008). O *Java* utiliza a *Java Virtual Machine* (JVM) para executar o código na máquina do usuário, realizando a criação de um ambiente virtual responsável por interpretar e executar o código compilado. A *Java Virtual Machine* é uma máquina de computação abstrata. Similar a uma máquina de computação real, ela possui um conjunto de instruções e manipula várias áreas da memória em tempo de execução (Yellin e Lindholm, 1996).

Na Figura 3 está ilustrado uma representação de um simples código em Java sendo compilado e interpretado em diversos Sistemas Operacionais através da *JVM*.

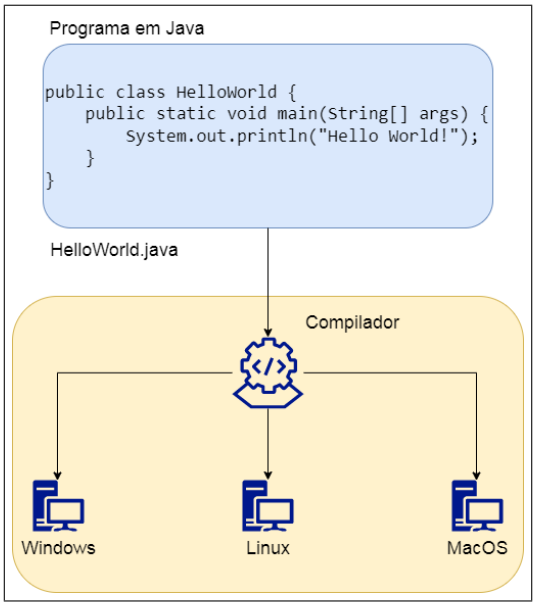

Figura 3. Compilação e execução através da JVM

2.3.2. Conteinerização

De acordo com Turnbull (2014), a conteinerização é um modelo de virtualização que implementa um conjunto de processos ou recursos de ambiente isolados do Sistema Operacional. Os contêineres implementam isolamento lógico e físico, onde há processos, usuários e outros recursos sendo executados de forma independente de uma camada intermediária de hardware. A execução lógica de um contêiner pode ser representado como uma isolação de uma parte da máquina *host*, onde será somente executado processos referentes a ele, ou seja, um contêiner não possui acesso a outros, executando suas tarefas de forma independente e evitando quaisquer conflitos com outros contêineres em um servidor. O isolamento físico de um contêiner determina os parâmetros referentes aos recursos que a máquina que está executando-os irá fornecer, seja a distribuição de núcleos de um

CPU, quantidade de *memória RAM* e prioridade de execução de cada contêiner. A Figura 4 apresenta o esquema da conteinerização, demonstrando como cada processo é isolado em um *Userspace*.

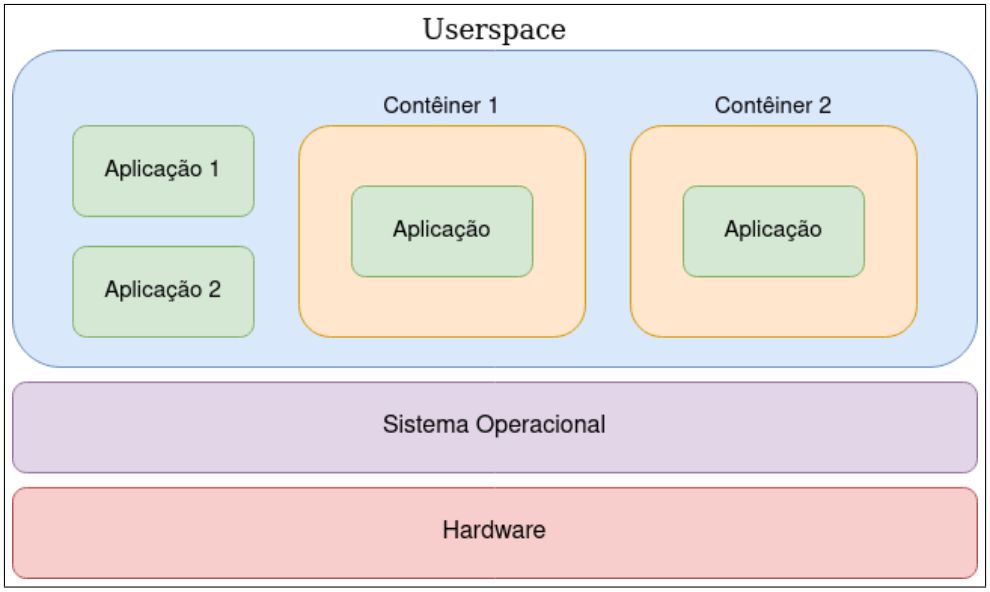

Figura 4. Esquema de uma Conteinerização

A Figura 4 demonstra a característica do *Userspace*, sendo responsável por criar um ambiente isolado para a execução de contêineres, realizando a comunicação com o Sistema Operacional e o *hardware* para implementar a execução lógica e física. O *Userspace* foi introduzido em 1992 com objetivo em realizar o isolamento de recursos baseado em processo. Os *Userspaces* fornecem ferramentas para isolar a visualização de recursos globais, como detalhes sobre sistemas de arquivos, processos, interfaces de rede, *Inter Process Communication* (IPC), nomes de *host* e *IDs* de usuário (Gholami e Laure, 2016).

A criação dos contêineres tem como base o comando *Chroot* do Sistema Operacional *Unix V7*, introduzido como uma forma de alterar o diretório raiz de um processo e seus processos filhos para um novo local separado, isolando-os de forma que uma invasão ou acesso de fora desse diretório seja complicado de conseguir (Gholami e Laure, 2016).

Comparado aos contêineres, temos um outro recurso comum: Máquinas virtuais. A Máquina Virtual (Virtual Machine) foi desenvolvida em 1960 pela IBM, no qual se aprimorou durante o tempo, sendo muito utilizada em ambiente de testes e desenvolvimento (Chen e Wagner, 2002). A Máquina Virtual é um recurso que possibilita executar um ou mais sistemas operacionais em uma máquina *Host*, simulando outros computadores em apenas um *Hardware*. Essa execução se caracteriza pelo uso de um *Hypervisor Virtualization*, um recurso responsável por realizar o controle das Máquinas Virtuais, atuando através de uma camada intermediária de *hardware*, permitindo emular completamente um computador. Eder (2016) define que a virtualização baseada em *hipervisor* permite emular totalmente outro computador, portanto, é possível emular outros tipos de dispositivos (por exemplo, um *smartphone*), outras arquiteturas de *CPU* ou outros sistemas operacionais. A Figura 5 apresenta uma comparação entre os dois modelos de virtualização.

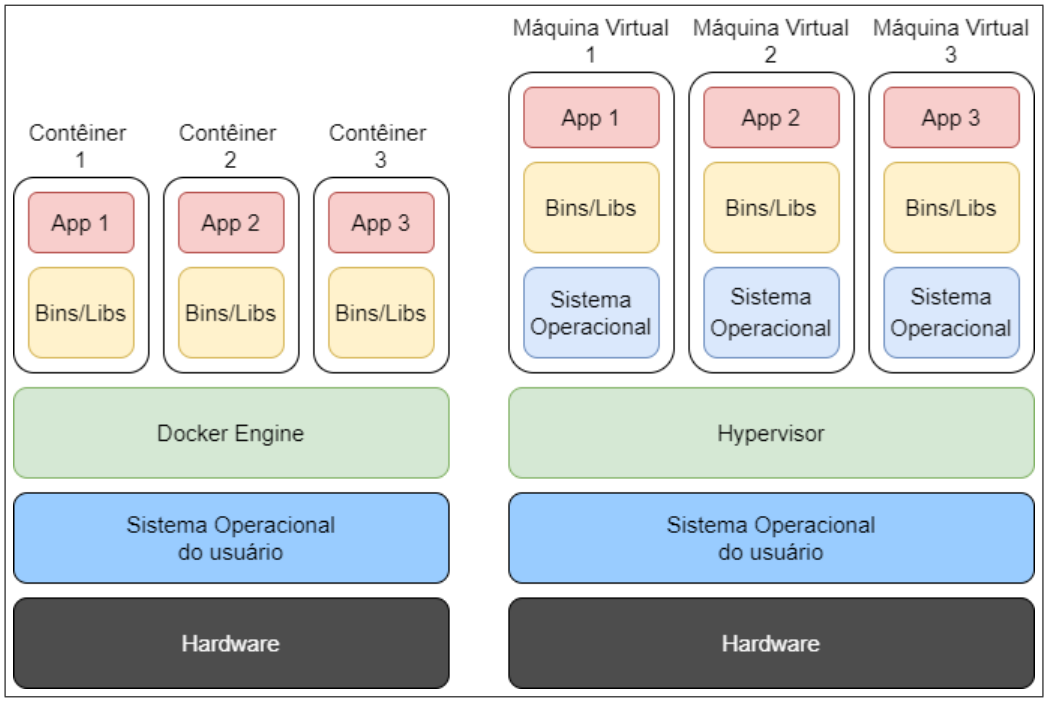

Figura 5. (a) Arquitetura baseada em Docker Contêiner (b) Arquitetura baseada em Hypervisor

A Figura 5 apresenta a principal diferença entre um contêiner e uma máquina virtual, o *Hypervisor Virtualization*, no qual o contêiner não emula uma máquina completa, utilizando apenas alguns recursos do Sistema Operacional atual para realizar a execução dos contêineres. A principal vantagem de um contêiner não utilizar um *Hypervisor* é a possibilidade de utilizar um hardware de baixo custo, isso porque não há uma emulação completa de um novo Sistema Operacional. Para Mattos (2008), essa vantagem impacta na de redução do *overhead* computacional, escalabilidade de aplicações e o custo necessário para rodar contêineres, permitindo uma grande quantidade a serem executados no host, alocando uma quantidade mínima de recursos para cada execução individual.

A subseção 2.3.3 retrata uma plataforma para criação e gerenciamento de contêineres, aplicando conceitos e inovações do modelo apresentado acima.

2.3.3. *Docker*

O *Docker* é um projeto de código aberto que se baseia em muitas tecnologias familiares de longa data da pesquisa de sistemas operacionais: *Linux Containers* (*LXC*), virtualização do sistema operacional e um sistema de versão e diferenciação baseado em *hash* ou *git*, entre outros (Boettiger, 2015). O *Docker* fornece uma plataforma que executa aplicações e facilita o processo de desenvolvimento e distribuição. As aplicações desenvolvidas no *Docker* são empacotadas com todas as dependências de suporte em um formulário padrão chamado contêiner. Esses contêineres continuam rodando de forma isolada em cima do *kernel* do Sistema Operacional (Rad et al., 2017).

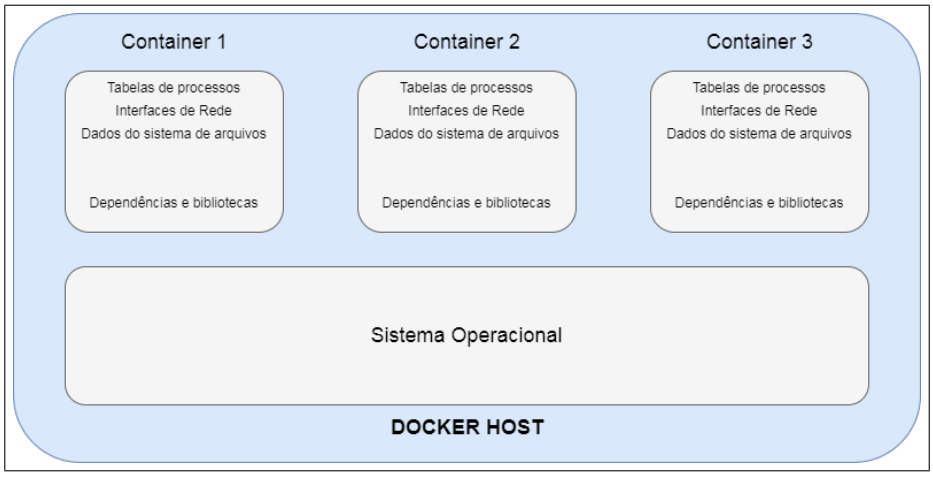

Figura 6. Contêiner em um *Docker* no *host*

A Figura 6 exemplifica como funciona o *Docker Host* e sua relação com os contêineres. Temos na base o *kernel* do sistema que receberá os contêineres, e em cima temos três contêineres que compartilharão os processos de um Sistema Operacional. Cada contêiner é independente dos outros, tem seus arquivos específicos, pré-requisitos necessários e executa a aplicação definida em sua *Docker Image*.

Como citado anteriormente, o *Docker* implementa o conceito de contêineres baseado no *LXC*. A subseção 2.3.4 retrata o modelo no qual o *Docker* se baseia.

2.3.4. Contêineres Linux (LXC)

O conceito de contêineres no Docker implementa sua própria versão com base nos *Contêineres Linux*, abstraindo conceitos de virtualização a nível de processos do Sistema Operacional. Contêineres *Linux* (LXC) é uma tecnologia de virtualização no nível do sistema operacional que permite criar e executar vários Sistemas Operacionais (SO) *Linux* simultaneamente em uma única máquina *Linux* (host LXC). O *LXC* fornece um conjunto de ferramentas para gerenciar seu contêiner, bem como modelos para criar um ambiente virtual do sistema operacional *Linux* mais comum (Kahuha, 2022). Para Ivanov (2017), a implantação de aplicações em um *LXC* permite diminuir problemas de pacotes em sistemas operacionais diferentes, no qual cada um impacta na funcionalidade e resultado final.

2.3.5. *BepiPred-2.0*

O *BepiPred-2.0* é um software *Desktop* e um servidor *Web* para predição de epítopos de células B a partir de sequências de antígenos, disponibilizado pela *DTU Health Tech*. O software foi desenvolvido através da linguagem de programação *Python*, sendo baseado em um algoritmo de árvore de regressão treinado em epítopos anotados a partir de estruturas de proteínas de anticorpos-antígenos. O método novo apresentado pelo *BepiPred-2.0* foi desenvolvido para superar outras ferramentas disponíveis para predição de epítopos baseada em sequência, tanto em dados de epítopos derivados de estruturas 3D resolvidas, quanto em uma grande coleção de epítopos lineares baixados do banco de dados do

IEDB (Jespersen et al., 2017). O *BepiPred-2.0* implementa duas versões: *Desktop* e *Web*. Através da aplicação *Desktop*, é possível instalar em qualquer Sistema Operacional, sendo necessário seguir diversos passos que exigem um pouco de experiência técnica do usuário para instalar o *software*. A versão *Web* possui facilidade na utilização, sendo necessário apenas disponibilizar o arquivo com a sequência da proteína, comumente chamada de arquivo *FASTA*, no formulário de submissão e iniciar a predição. A Figura 7 demonstra a página Web do *BepiPred-2.0* e seu formulário de submissão.

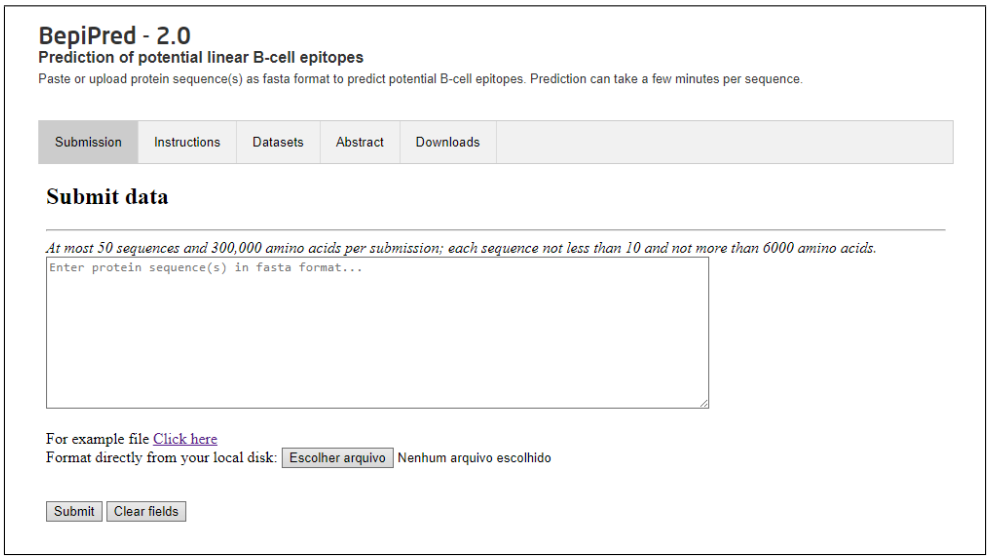

Figura 7. Formulário de submissão

Como demonstra a Figura 7, é possível processar até 50 sequências e 300.000 aminoácidos a cada requisição. Essas requisições do *BepiPred-2.0* funcionam através de uma fila, onde cada requisição de todos os usuários é tratada sobre uma solicitação realizada ao servidor *Web* e enfileirada. Cada usuário tem que esperar o recurso do servidor *Web* ser liberado até ser possível o seu arquivo *FASTA* ser processado, gerando atrasos caso o servidor esteja ocupado. Outro ponto a ser avaliado é o tempo que o servidor *Web* processa a sequência, levando horas ou até dias baseado no tamanho e quantidade das sequências fornecidas no arquivo.

A versão *Desktop* do *BepiPred-2.0* necessita de conhecimento técnico, sendo aplicado na instalação de outros *softwares* necessários para utilizá-lo. Cada programa necessário para o *BepiPred-2.0* tem seus pré-requisitos e, por isso, é necessário tempo e pesquisa de dependências necessárias para cada um, tornando-se um problema para usuários que não possuem conhecimento para instalá-los. O ponto positivo para a versão *Desktop* do *BepiPred* é evitar filas e longos períodos de processamento do servidor *Web*, já que o processamento é realizado pela máquina do usuário, tornando-se ainda mais prática em hardwares com alto nível de processamento.

O *BepiPred-2.0* fornece sequenciamentos de teste para validar o comportamento da *aplicação*. A Figura 8 demonstra o resultado de um sequenciamento de exemplo processado através do servidor *Web*, contendo um trecho da predição de um anticorpo terapêutico para tratamento de diabetes tipo 2.

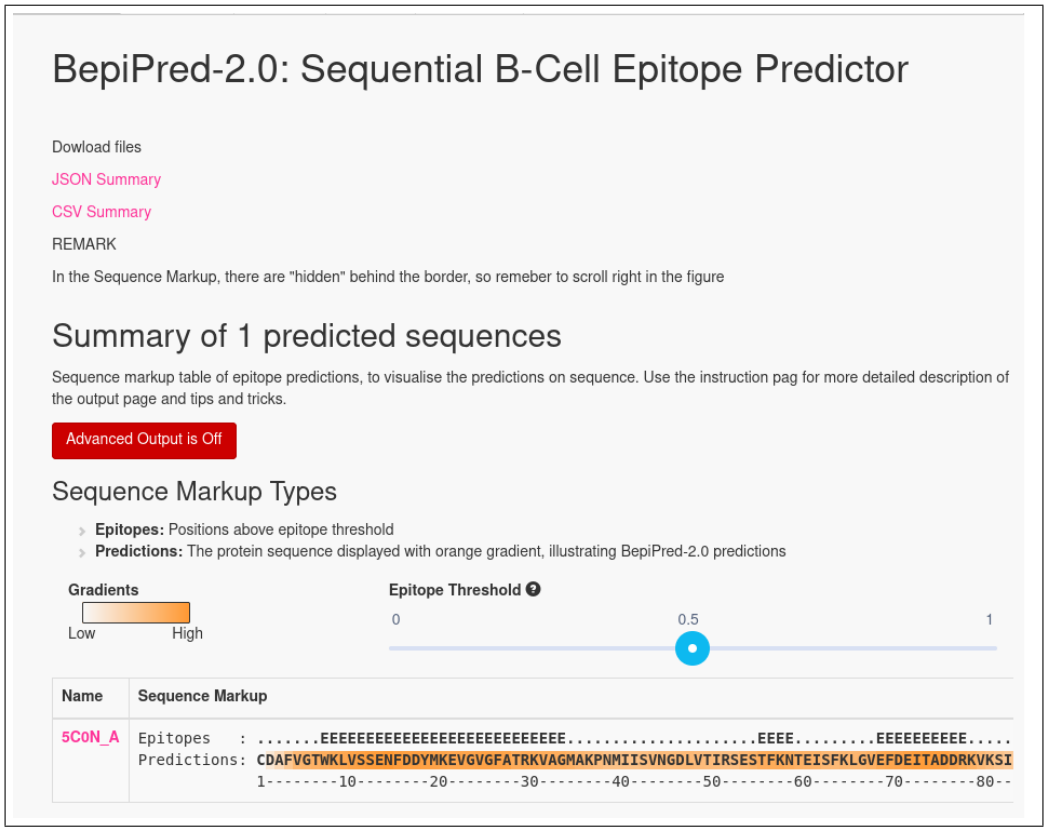

Figura 8. Trecho do resultado de uma execução do BepiPred-2.0 Web.

O resultado do servidor *Web* é apresentado através de um quadro simples, possuindo a seguinte estrutura: *Name* e *Sequence Markup*. O *Name* contém a descrição do processo realizado, retornando como resultado uma *Sequence* (Sequenciamento) no exemplo da nossa predição. O *Sequence Markup* apresenta o resultado do sequenciamento, através de três linhas: A primeira linha, identificada como *Epitopes*, é responsável por indicar a presença de Epítopos em cada parte da sequência analisada. A linha subsequente, chamada de *Predictions*, apresenta o resultado do sequenciamento completo do anticorpo processado, é possível observar que a linha possui uma coloração gradiente e variável, responsável por indicar a presença da proteína presente na sequência. Por fim, a última linha apenas apresenta apenas a quantidade de caracteres da sequência, indicando cada dezena. É possível também ajustar o grau de confiança do limite do epítopo (*threshold*), utilizando o *scroll* entre valores maiores e maiores que 0.5, até o minímo de 0 e máximo de 1. Esses valores representam uma classificação da presença do epítopo em altos e baixos graus. Diminuir o *threshold* impacta na presença de mais epítopos, porém uma menor confiança na presença concreta deles. Aumentar o *threshold* realiza uma classificação mais profunda na busca de epítopos, tornando-os raros durante a sequencia toda, porém, fornecendo mais certeza de sua presença no sequenciamento todo.

2.3.6. BioLib

BioLib é uma biblioteca de aplicativos de ciência de dados biológicos. As aplicações do *BioLib* vão desde pequenos utilitários de bioinformática até algoritmos de aprendi-

zado de máquina de última geração para prever características de moléculas biológicas (BioLib, 2022), permitindo a execução direta no *website* ou por meio de uma instalação na máquina do usuário através de um *pacote* da linguagem de programação *Python*. As aplicações possuem uma ferramenta de controle de versões, onde, é possível proporcionar ao usuário escolher previamente uma versão específica antes de executar uma aplicação. A simplicidade do *BioLib* traz benefícios tanto para o usuário quanto ao desenvolvedor, possibilitando ambos de utilizar todos os recursos fornecidos pela plataforma e evitar o trabalho manual de documentar um processo de instalação e execução, o qual pode, inclusive, desestimular o usuário a instalar o *software*. Por ser disponibilizado na linguagem *Python*, é possível também utilizar os pacotes em qualquer sistema operacional.

Desenvolvedores podem publicar suas aplicações de bioinformática no *BioLib*, criando uma conta e um projeto na plataforma, possibilitando a integração do seu *software* e todos os pré-requisitos necessários para o usuário executá-lo. A integração de um software no *BioLib* ocorre através da criação um projeto no *website*, onde o desenvolvedor recebe instruções para implantar sua aplicação na plataforma e como configurá-la. Após o desenvolvedor realizar uma publicação na plataforma, a aplicação será incluída na biblioteca *Python* oficial do *BioLib*, conhecida como *pybiolib*. A biblioteca *pybiolib* é responsável por disponibilizar o acesso à aplicações localmente (através de terminal do sistema operacional), centralizando as aplicações do *BioLib* instaladas no servidor. Esta biblioteca aplica uma técnica de execução similar a um gerenciador de pacotes, atuando de forma a reduzir o uso de armazenamento na máquina do usuário, isto significa que quando o *pybiolib* é instalado localmente, todos os projetos publicados na plataforma não são baixados junto com a biblioteca, sendo instalados quando uma tentativa de execução de um projeto específico é realizado. Quando o usuário executa um comando no terminal através do *pybiolib*, é verificado se a aplicação solicitada está presente na máquina local e, caso esteja, será executado, porém se não estiver instalado, o *pybiolib* realiza o *download* dessa aplicação publicada na plataforma e insere diretamente na biblioteca, tornando possível durante a tentativa de execução atual e futuras.

A criação da camada *Web* de uma aplicação no *BioLib* é simplificado e desenvolvido de forma automática, utilizando uma automação de elementos de interface gráfica, através de um arquivo no formato *YAML* (YAML Ain't Markup Language). Isso significa que o desenvolvedor não necessita de conhecimento em desenvolvimento *Web*, no qual o *HTML* e *CSS* são produzidos automaticamente através do direcionamento de quais parâmetros de entrada e comando de execução serão utilizados. Essa automatização permite aos programadores focarem no desenvolvimento do seu código e trabalhar de froma ágil, controlando a camada *Web* de sua aplicação por meio de arquivos de configuração e ajustes automáticos do *BioLib*.

2.4. Strings de Busca

Para embasar as informações apresentadas, foi feito uso da ferramenta *Google Scholar*, em que, utilizando de *strings* de buscas sobre o tema do projeto, foram coletados e revisados os materiais bibliográficos obtidos visando filtrar os trabalhos mais relevantes para o projeto.

Título	Link artigo	Autor	Ano	Resumo	String de busca	Base de Busca
An Introduction to B-Cell Epitope Mapping and In Silico Epitope Prediction	https://www.hindawi.com/journals/jir/2016/6780830/	Potocnakova, Lenka, Mangesh Bhide, and Lucia Borszek "An introduction to B-cell epitope m	2016	Neste trabalho é apresentado uma breve introdução a predição de epitopos e a sua importância,	b cell epitope prediction	Google Scholar
Epitopia: a web-server for predicting B-cell epitopes	https://bmcbioinformatics.biomedcentral.com/articles	Rubinstein, N.D., Mayrose, I., Martz, E. et al. Epitopia: a web-ser for predicting B-cell epitopes.	2009	Exemplo de site para predição	epitope b cell prediction server	Google Scholar
Bioinformatics Resources and Tools for Conformational B-Cell Epitope Prediction	https://www.hindawi.com/journals/cmmm/2013/94363	Pingping Sun, Haixu Ju, Zhenbang Liu, Qiao Ning, Jian Zhang, Xiaowei Zhao, Yanxin Huang,	2013	Exemplos de ferramentas para predição d	epitope prediction server	Google Scholar
Bioinformatics as a tool in the analysis of antig design of vaccines against Anaplasma margina	https://www.brazilianjournals.com/index.php/BRJD/ar	Pereira, Nayara Gonçalves, et al. "E	2021	Apresentação de ferramentas computacio	bioinformatic epitopes	Google Scholar
Bioinformática: da Biologia à Flexibilidade Molecular	https://www.lume.ufrgs.br/bitstream/handle/10183/16	Verli, Hugo. "Bioinformática: da biol à flexibilidade molecular." (2014).	2014	Apresenta uma visão sobre a Bioinformáti sua história, desenvolvimento e definição	bioinformatica	Google Scholar
A ERA DA BIOINFORMÁTICA: SEU POTENC	https://periodicos.pucpr.br/estudosdebiologia/article/v	Araújo, N. D. de, Farias, R. P. de, F	2008	Apresentação sobre Bioinformática simple	bioinformatica	Google Scholar
In silico Identification and Validation of a Linear and Naturally Immunogenic B-Cell Epitope of the Plasmodium vivax Malaria	https://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0146051&type=printa		2016	Exemplo de trabalho similar	In Silico Identification and Validation	Google Scholar
Epitopos	https://www.google.com.br/books/edition/C%C3%A9	Russom Kilsen	?	Definição de células B	epitopo	Google Books
Microbiologia_III_sistema_imunológico	https://www.google.com.br/books/edition/Microbiologi	Isidore Kerpan, Franklin Walzem, F	2020	Definição de Epítopos	epitopo	Google Books
Linguagem de Programação JAVA	https://intranet.ifs.ifsuldeminas.edu.br/michelle.nery/2	Michelle Nery	desconhecido	Definição da linguagem Java	linguagem Java	Google Scholar
Programação em JAVA	http://www.lasid.ufba.br/teste/pessoal/danielaclaro/do	Daniela Barreiro Claro, João Bosco	2008	Definição da linguagem Java	linguagem Java	Google Scholar
Documentação da Oracle sobre Java	https://docs.oracle.com/javase/tutorial/getStarted/intr	Oracle	?	Fluxogramas do Oracle	ENCONTRADO EM UMA REFERENCIA NA WE	Doc. Oficial
AMINOÁCIDOS E PROTEÍNAS	https://d1wqtxts1xzle7.cloudfront.net/58447838/Bioq	VALTER T. MOTTA	2010	Explicação completa sobre aminoacidos e	proteínas o que são	Google Scholar
Epibuilder 1.0	https://journals.sagepub.com/doi/full/10.1177/1177932	Moreira, Renato Simões and Filho,	2022	Artigo completo sobre o Epibuilder 1.0	epibuilder	Google Scholar
Aspectos atuais sobre aminoácidos de cadeia	https://www.scielo.br/j/rbcf/a/VBDDK7jfV9Gb7dDxk7s	Rogero, Marcelo Macedo and Tirapc	2008	Artigo sobre aminoácidos	aminoácido	Google Scholar
The Docker Book: Containerization Is the New	https://books.google.com.br/books?hl=pt-BR&lr=&id	Turnbull, James	2014	Definição de contêineres e Docker	containerization	Google Scholar
Virtualização: VMWare e Xen	https://www.researchgate.net/profile/Diogo-Menezes-	Diogo Menezes Ferrazani Mattos	2008	Mais definições de virtualização	virtualization	Google Scholar
Proceedings of the Seminars Future Internet (FI) and	https://www.net.in.tum.de/fileadmin/TUM/NET/NET-20	Chen, Hao and Wagner, David	2016	Hypervisor vs container	chroot	Google Scholar
Security and Privacy of Sensitive Data in Clou	https://arxiv.org/abs/1601.01498	Ali Gholami, Erwin Laure	2016	Definição Chroot Unix	chroot unix	Google Scholar
Aprendendo JavaScript	http://www.ouka.com.br/carol/e-book/javascript-para-i	Grillo, Filipe Del Nero and Fortes, R	2008	Introdução ao JavaScript	JavaScript o que é	Google Scholar
Uma reintrodução ao JavaScript	https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/A_re-introduction_to_JavaScript	?		JavaScript e detalhes	JavaScript o que é	Blog Mozilla
Version Control System: A Review	https://www.sciencedirect.com/science/article/pii/S18	Zolkifli, Nazatul Nurlisa and Ngah, A	2018	Conceito sobre versionamento de código	Version Control System	Google Scholar
An introduction to Docker for reproducible rese	https://arxiv.org/pdf/1410.0846.pdf	Carl Boettiger	2015	Resumo sobre docker	Docker	Google Scholar

Figura 9. Strings de busca usadas para embasar a seção de Referencial Teórico

2.5. Versionamento

Os sistemas de controle de versão (VCS) têm sido usados por muitos desenvolvedores de software durante o desenvolvimento de projetos, isso porque esses sistemas auxiliam a gerenciar os códigos-fonte e permite que eles mantenham todas as versões do projeto em que trabalharam. É o caminho para gerenciar, organizar e coordenar o desenvolvimento de objetos (Zolkifli et al., 2018). Aplicações desenvolvidas no mundo moderno tendem a gerar diversas versões por conta de novas funcionalidade e correções de *bugs*, onde a necessidade de controlar todo o código produzido é prioridade. Novas funcionalidades desenvolvidas podem possuir problemas, portanto o versionamento atua como um controle onde é possível gerenciar o código completo de um software e garantir que, em caso falhas, é possível fazer *rollback*.

A subseção 3.5.1 apresenta uma ferramenta responsável por gerenciar o versionamento de código.

2.5.1. GitHub

O *GitHub* foi utilizado para manter o controle de versionamento dos recursos que forem sendo criados e adicionados a aplicação durante o decorrer do desenvolvimento do *EpiBuilder 2.0*.

GitHub é uma plataforma para o controle de versionamento de código, além de servir como repositório para diversos projetos que são desenvolvidos em grupo, fazendo controle dos arquivos e gestão dos conflitos de código através de *pull requests* e *merge*. Outro fator é que é possível o uso de forma gratuita da plataforma, apenas com o requisito de que o repositório fique público (Shen e Spruit, 2019). O *GitHub* oferece vários recursos exclusivos para facilitar a colaboração do usuário. Sua característica mais importante é o mecanismo *Pull Request (PR)*, que é uma forma de iniciar a discussão com outros usuários e compartilhar ou comentar sobre os vários artefatos em um projeto (normal-

mente alterações no conteúdo do projeto) (Zagalsky et al., 2015).

Em nosso trabalho usamos a plataforma para inserir as novas informações referentes ao desenvolvimento do com *Docker*, adicionando documentação e instruções para que outras pessoas consigam fazer uso dessas funcionalidades e ferramentas.

3. Desenvolvimento

Nessa seção são apresentados os conceitos referentes ao desenvolvimento do *EpiBuilder 2.0*, abordando tecnologias que serão utilizadas e pré-requisitos do sistema.

3.1. Pré-requisitos

Como pré-requisitos, é necessário ter conhecimento em análise de proteínas e imunologia, utilizados para validar os resultados posteriores e durante o uso do *Epibuilder*. Em nível computacional, é necessário um computador com *Java* e *Docker* instalado, utilizando as versões mais recentes de ambos. Faz-se necessário também ter no mínimo dois *cores* de processamento no computador do usuário.

3.2. *Softwares* de Bioinformática

Durante essa subseção, serão apresentados *softwares* de bioinformática com foco em análises de epítopos. Os programas abaixo usam de diferentes métodos para executar a predição de epítopos e obter o resultado de forma eficaz e ágil.

3.2.1. *BepiPred-3.0*

O *BepiPred-3.0* é uma ferramenta de predição de epítopo baseada em sequência que, explorando incorporações de aprendizado de máquina, otimiza e aperfeiçoa a predição de epítopos lineares e conformacionais em vários conjuntos de teste independentes (Clifford et al., 2022). Essa nova versão, lançada em maio de 2022, implementa melhorias na instalação do software, um novo *dataset* e novos *outputs* para os resultados dos arquivos *FASTA* processados. É possível utilizar versões atuais do *Python* para instalar e utilizar o *BepiPred-3.0*, se tornando uma melhoria comparado a sua versão antecessora que era limitada a utilizar a versão 2.7 do *Python* e inferiores. A principal melhoria na instalação é a simplicidade em instalar o software, isso porque não há a necessidade de configurar aplicações de terceiros, variáveis de ambiente e lidar com versões antigas do *Python*. De acordo com Clifford et al. (2022), essa nova versão também é capaz de prever epítopos em centenas de sequências em meros minutos.

O *BepiPred-3.0* utiliza *datasets* estruturais para realizar as predições, reutilizando o modelo do *BepiPred-2.0* e adicionando um novo para essa nova versão, chamado de *BP3*. Para Clifford et al. (2022), ambos os *datasets* possuem um comportamento similar, realizando inicialmente a identificação de estruturas cristalinas através do uso de um banco de dados de proteínas. O resultado das estruturas cristalinas são utilizadas com uma identificação de resíduo de epítopos e a uma cadeia de antígenos pré-treinada, no qual foi possível obter ao total 1466 antígenos e adicioná-los ao novo *dataset* para realizar as predições.

Essa versão mantém o uso da versão *Web*, adicionando o novo *dataset* para as predições. O comportamento permanece em um formato de fila, no qual é necessário

aguardar a predição de outros usuários a cada requisição. Cada requisição é chamada de *Job*, sendo um identificador único de cada arquivo processado.

O diferencial nessa nova versão *Web* são os novos recursos adicionados, entre eles a exibição de gráficos dos resultados processados, exibidos de forma interativa e de fácil visualização, um arquivo com os 15 melhores resultados da predição. A Figura 10 apresenta o resultado de um sequenciamento na versão *Web* do *BepiPred-3.0*.

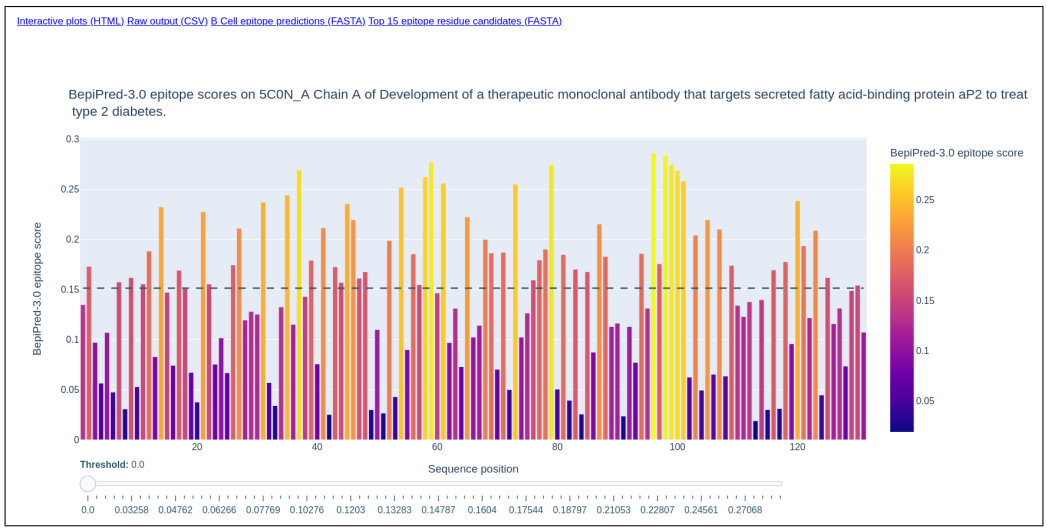

Figura 10. Resultado de um sequenciamento no BepiPred-3.0.

Foi realizado uma predição com um arquivo *FASTA* de exemplo do *BepiPred-3.0*, sendo esse sequenciamento de uma cadeia de desenvolvimento de um anticorpo para tratar diabetes tipo 2. A Figura 10 apresenta o resultado do sequenciamento de exemplo e também o novo formato de resultado do *BepiPred* em sua versão 3.0, no qual é possível visualizar um gráfico em barras com o *score* de cada epítopo, permitindo também alterar o *Threshold* desse gráfico. O *BepiPred-3.0* permite também realizar o *download* do gráfico em um arquivo *HTML*, o arquivo *FASTA* original, um novo arquivo *FASTA* que contém os 15 melhores resultados da predição e também um arquivo *CSV* com os resultados dessa predição. No canto superior esquerdo da Figura 10, há 4 *links* clicáveis, no qual cada um realiza o *download* específico desses resultados.

3.3. Desenvolvimento utilizando o Docker

O principal objetivo da utilização do *Docker* é facilitar o uso das ferramentas *BepiPred-3.0* e *Epibuilder*, que podem ser bastante simplificadas para qualquer tipo de usuário. Durante o desenvolvimento, é necessária a composição completa do *Docker*, que inclui três componentes principais para seu uso: *Docker Engine*, *Docker Desktop* e *Docker Hub*. Após a definição desse objetivo, cada ferramenta e processo executado na etapa de desenvolvimento utilizando as ferramentas do *Docker* é apresentado separadamente. Esta seção está dividida em 4 subseções que apresentam e definem as ferramentas utilizadas pelo *Docker*, descritas nas subseções 3.3.1, 3.3.2 e 3.3.3 respectivamente.

3.3.1. Docker Engine

O *Docker Engine* implementa todas as funcionalidades necessárias para gerenciar contêineres e *Docker Images*, sendo a aplicação principal. Para utilizar o *Docker Engine*, é necessário operá-lo através de linhas de comando pelo terminal do Sistema Operacional ou utilizando o *Docker Desktop*. O *Docker Engine* implementa um componente responsável pela comunicação para gerenciar o *download* e *upload* de Docker Images no *Docker Hub*: *Docker Daemon*. De acordo com Khandhar e Shah (2019), esse componente escuta solicitações da *API* do Docker e gerencia objetos como *Docker Images*, contêineres, redes e *volumes*. Um *daemon* também pode se comunicar com outros *daemons* para gerenciar os serviços do *Docker*. Essa ferramenta foi utilizada para realizar todas as implementações de automatização do *Epibuilder* e *BepiPred-3.0*.

3.3.2. *Docker Desktop*

O *Docker Desktop* ajuda os desenvolvedores a codificar e conteinerizar seus aplicativos. Este software também permite fácil coordenação entre os principais componentes do *Docker*, como o *Docker Engine* para várias plataformas, por exemplo, sistemas operacionais *Windows*, *Linux* e *MacOS* (Haque et al., 2020). Essa ferramenta oferece uma interface gráfica para o usuário trabalhar com o Docker Engine, possibilitando gerenciar todas as imagens e contêineres. Outra característica do *Docker Desktop* é utilizar uma conta de usuário do *Docker Hub* para realizar a autenticação e login no ambiente de trabalho do desenvolvedor, ou seja, é necessário para utilizar as ferramentas do *Docker* e vincular todos os artefatos desenvolvidos através do *Docker Engine*. Essa funcionalidade permite evitar o trabalho manual de gerenciar os contêineres e facilitar o envio de *Docker Images* para a plataforma *Docker Hub*.

3.3.3. *Docker Hub*

A plataforma *Docker Hub* é um repositório central de imagens (públicas e privadas), no qual os usuários podem compartilhar suas *Docker Images* personalizadas em repositórios e torná-las disponíveis para outros desenvolvedores utilizarem. Os usuários também podem pesquisar imagens publicadas e baixá-las com o cliente *Docker* (Bui, 2015). Repositório refere-se ao armazenamento de imagens do *Docker* onde os desenvolvedores podem fazer *upload* ou *download* de imagens para uso em suas áreas de desenvolvimento (Haque et al., 2020). O *Docker Hub* foi utilizado para disponibilizar uma *Docker Image* pública com o *Epibuilder* e *BepiPred-3.0* pré-instalados.

3.3.4. Desenvolvimento de um Docker Image

Para Kwon e Lee (2020) o *Docker Image* é formato de arquivo quantitativo, onde o *Docker* empacota todos os arquivos necessários para o aplicativo, biblioteca, *middleware*, Sistema Operacional, configuração de rede etc. *Docker Images* permitem ao desenvolvedor utilizá-las como base para criar os seus contêineres. Alterações internas no contêiner como a adição de software e modificações nos processos anteriormente disponibilizados pela base

do *Docker Image* incrementam a experiência do usuário e proporciona um ambiente configurado para ele suprir suas necessidades. Com base nas alterações de um contêiner, uma nova *Docker Image* pode ser gerada a partir dele, possuindo todas as alterações realizadas anteriormente salvas na nova *Docker Image*.

Durante o desenvolvimento, foram criadas duas *Docker Images*, sendo uma baseada na distribuição *Linux Debian* e outra no *Alpine*. Nessa seção, será apresentado o desenvolvimento completo da *Docker Image* baseada no Alpine. Para Alles et al. (2018), o *Alpine* é uma distribuição *Linux* leve que busca eficiência, isolamento e simplicidade nos processos, utilizando o mínimo de recursos possíveis. O *Alpine*, no *Docker Hub*, é uma *Docker Image* com tamanho de apenas 5MB, sendo uma das mais utilizadas nessa plataforma. A *Docker Image* do *Alpine* oferece o uso baixo de recursos, se comparada a *Docker Images* de outras *distribuições Linux* disponibilizadas no *Docker Hub*, como o *Debian* e *Ubuntu*, que possuem um tamanho maior de 100MB.

A implementação de um contêiner *Docker* teve como objetivo preparar um ambiente configurado com os pré-requisitos necessários para utilizar o *Epibuilder* e *BepiPred-3.0*, posteriormente gerando uma *Docker Image* através desse contêiner e disponibilizando-o no *Docker Hub*. É possível gerar uma *Docker Image* através de um contêiner ou utilizar um *Dockerfile*. Um *Dockerfile* é um documento de texto que contém todos os comandos que um usuário pode chamar na linha de comando para montar uma *Docker Image* (Docker, 2022). Inicialmente foi gerado um contêiner através da *Docker Image Alpine*, no qual foi configurado manualmente todas as configurações do *BepiPred-3.0* e *Epibuilder*. Todos os processos serão apresentados a seguir através de Figuras, as quais foram executadas em um terminal de um Sistema Operacional *Linux*. A Figura 11 retrata a execução de um comando no terminal para criar um contêiner novo a partir da *Docker Image* do *Alpine*.

Figura 11. Comando do Docker para criar um contêiner Alpine através do Terminal.

O comando apresentado pela Figura 11 demonstra um padrão utilizado para executar tarefas do *Docker Engine* através de um terminal. Em um terminal, qualquer comando utilizando o parâmetro *docker* no início produz uma chamada para o *Docker Engine* realizar uma ação, no qual a Figura 11 demonstra a criação de um novo contêiner. É indicado o parâmetro *container* após o parâmetro *docker*, designando que esse comando realizará uma ação diretamente para um contêiner. O parâmetro *run* aponta que será criado e executado um novo contêiner. Em sequência, o parâmetro *-it* indica dois sub parâmetros para o *Docker* executar: interatividade e conexão com o terminal do contêiner. Esses sub parâmetros solicitam ao *Docker Engine* que, o acesso do usuário, ao executar o contêiner, seja direcionado ao terminal interno a cada execução, pois os processos dos *Epibuilder* são executados através desse recurso. Por fim, o último indica a *Docker Image* a ser utilizada como base desse novo contêiner. A Figura 12 apresenta o resultado da execução do comando *Docker*.

Figura 12. Execução de um novo contêiner através do Docker.

A Figura 12 apresenta as seguintes mensagens no Terminal após a executar o comando: *Unable to find image 'alpine:latest' locally* e *latest: Pulling from library/alpine*. Essas mensagens informam que não há a *Docker Image* do Alpine presente na máquina *host* que está sendo executado o comando, então o *Docker* inicia um processo de *download* dessa *Docker Image*, posteriormente apresentando uma nova mensagem de sucesso de *download* no terminal: *Status: Downloaded newer image for alpine:latest*. Esse comportamento do *Docker* é similar a biblioteca *pybiolib*, que realiza o *download* para a máquina *host* somente quando o recurso necessário é solicitado, poupando uma alocação não necessária de recursos. Como demonstra a Figura 12, a execução do comando criou um novo contêiner baseado na *Docker Image* do *Alpine* e automaticamente realizou uma conexão ao seu Terminal. Essa conexão se caracteriza pela representação visual dos caracteres */ #*, demonstrando que o terminal do contêiner está sendo acessado. Estando conectado ao terminal do contêiner, podem ser utilizados comandos para preparar o contêiner para os *programas BepiPred-3.0* e *Epibuilder*. A primeira etapa com o contêiner é utilizar o *gerenciador de pacotes* do *Alpine*, conhecido como *apk*, para baixar todas as dependências necessárias para utilizar os programas no contêiner. Conforme Ozga et al. (2020) definem, os sistemas operacionais usam os gerenciadores de pacotes para simplificar a instalação, atualização e remoção de aplicações. A sintaxe para utilizar o *apk* tem uso o parâmetro *apk* e a ação que deseja realizar com ele, baixando, atualizando ou até removendo um software. Outros gerenciadores de pacotes mantém a mesma sintaxe ou similar para realizar as mesmas operações. A Figura 13 mostra um caso de uso do gerenciador de pacotes do *Alpine*.

Figura 13. Atualização de repositório do Apk.

Como demonstra a Figura 13, o comando *apk update* tem como resultado a atualização do repositório local de aplicações disponíveis para *download* através do *apk*.

Caso um novo software ou uma nova versão de algum já instalado tenha sido lançada, esse comando permite que o gerenciador de pacotes tenha ciência da existência desse recurso. Essa atualização de repositório tem como objetivo permitir ao *apk* ter visibiidade das versões mais recentes do *Java*, *Python* e a dependências do *pybiolib* para possibilitar a realização do *download* desses pacotes. A Figura 14 apresenta o comando utilizado para baixar os pacotes citados anteriormente, sendo eles necessários para o contêiner funcionar corretamente.

Figura 14. Instalação de Software necessários.

O comando *apk add* demonstrado na Figura 14 proporciona a possibilidade de instalar um ou mais pacotes por uma única linha de comando. Os pacotes instalados são: *python3*, *py3-pip*, *gcc*, *g++* e *openjdk11-jre*. O pacote *python3* contém a linguagem de programação *Python* e suas dependências para a execução de aplicações em *Python*. O pacote subsequente instalado é o *py3-pip*, conhecido como *pip*, se define como o gerenciador de pacotes da linguagem *Python*, sendo utilizado para realizar a instalação do *pybiolib*. Os pacotes *gcc* e *g++* são pacotes responsáveis por instalar dependências necessárias para o *pybiolib*. Por último, é instalado o pacote *openjdk11-jre*, sendo o ambiente de execução do Java (*Java Runtime Environment*) responsável por executar o *Epibuilder*. Após o comando ser executado e concluído com sucesso, é possível instalar a biblioteca *pybiolib*. A Figura 15 exibe a instalação dessa biblioteca através do gerenciador de pacotes *pip*.

Figura 15. Instalação da biblioteca pybiolib.

A Figura 15 demonstra o *pip* sendo utilizado para baixar a biblioteca *pybiolib*, que, diferente do *apk*, utiliza o parâmetro *install* para realizar a instalação de pacotes e o parâmetro *-U* para atualizar pacotes antigos do *Python*. As instalações do *pip* utilizam o repositório *PyPI*. De acordo com (Abdalkareem et al., 2020), o *PyPI* é a plataforma oficial de gerenciamento de pacotes para a linguagem de programação *Python*. Após concluir a instalação, podemos utilizar o *BepiPred-3.0* através do *pybiolib*. A Figura 16 apresenta a criação de um diretório para realizar o processamento de predições.

Figura 16. Criação de um diretório de predição.

O diretório de predição será utilizado como entrada e saída para o processamento de arquivos entre o *BepiPred-3.0* e *Epibuilder*. É necessário um arquivo *FASTA* para iniciar o processo de predição. Conforme a Figura 17 exibe, foi realizado o download de um arquivo *FASTA* para testar a predição do *BepiPred-3.0*.

Figura 17. Download de um arquivo FASTA.

O comando *wget* apresentado na Figura 17 tem como funcionalidade realizar o download de um arquivo com base no parâmetro passado, sendo esse parâmetro um *link*. Foi realizado o download de um arquivo *FASTA* através da execução do comando *wget*, seguido do link de um arquivo *FASTA* de exemplo do servidor Web do *BepiPred-3.0* como parâmetro. O arquivo é baixado com base no link utilizado e armazenado no diretório onde o comando foi executado. A Figura 17 também apresenta os *logs* do *wget* ao realizar o download do arquivo *FASTA*. Utilizando o *BepiPred-3.0*, esse arquivo *FASTA* foi processado. Esse processamento é demonstrado pela Figura 18.

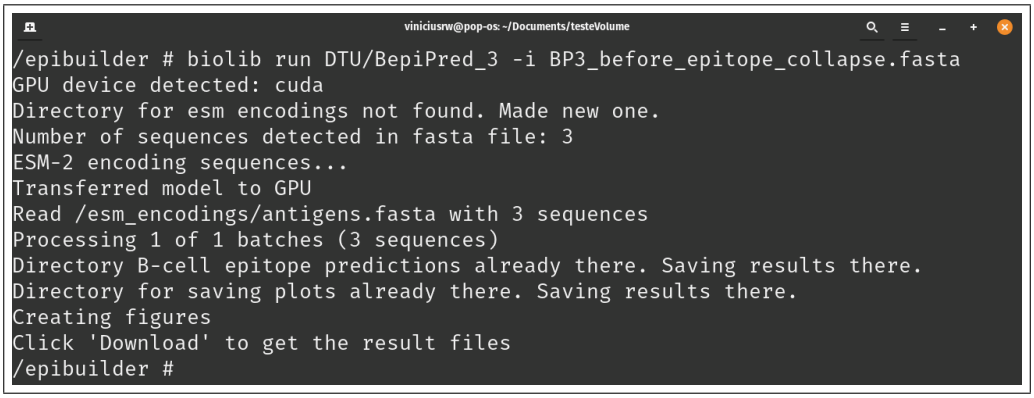

Figura 18. Processamento do arquivo FASTA com o BepiPred-3.0.

Para utilizar as funcionalidades do *pybiolib*, é necessário utilizar o Terminal com o comando inicial *biolib*, seguido de seus parâmetros. O comando utilizado para realizar o processamento foi o *biolib run DTU/BepiPred_3*, aplicando também o parâmetro *-i* para informar o arquivo *FASTA* a ser processado. A Figura 18 fornece um *feedback* visual de todos os processos que o *BepiPred-3.0* está executando para realizar o processamento do arquivo *FASTA*. Com base no *hardware* gráfico, o processamento é realizado de forma mais rápida. O *BepiPred-3.0* separa cada sequência em lotes de processamento, no qual cada um é processado em ordem. Concluindo o processamento, o *BepiPred-3.0* gera os arquivos de resultado e os salva em um subdiretório criado no diretório do *FASTA* executado, com o nome de *biolib_results*. Esse diretório armazena todos os resultados e arquivos originais que o *BepiPred-3.0* utilizou durante o processamento. Navegando até esse diretório e utilizando o comando *ls* no Terminal, é possível visualizar os arquivos gerados pelo *BepiPred-3.0*. A Figura 19 mostra os arquivos gerados.

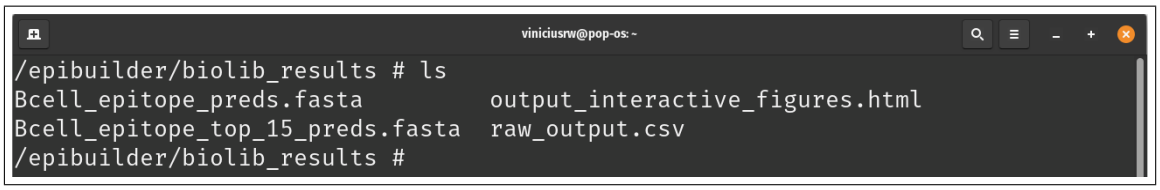

Figura 19. Arquivos gerados através do processamento do FASTA.

Como explicado anteriormente, o BepiPred-3.0 produz 4 arquivos após processar um sequenciamento: o gráfico em um formato *HTML*, o arquivo *FASTA* original, e um arquivo com os 15 melhores resultados e um arquivo *CSV* com esses resultados. Por fim, o último processo é utilizar o *Epibuilder* para processar o arquivo *CSV* e produzir um novo resultado com os dados montados e classificados. A Figura 20 apresenta o *Download* do *Epibuilder* e, posteriormente, a predição do arquivo *CSV*.

Figura 20. Download do Epibuilder.

Realizado o download do arquivo *Java* do *Epibuilder* com o comando *wget* e apontando para o link do *GitHub*, foi realizado a predição do arquivo *CSV*. A Figura 21 exibe o processamento com o software *Epibuilder 2.0*.

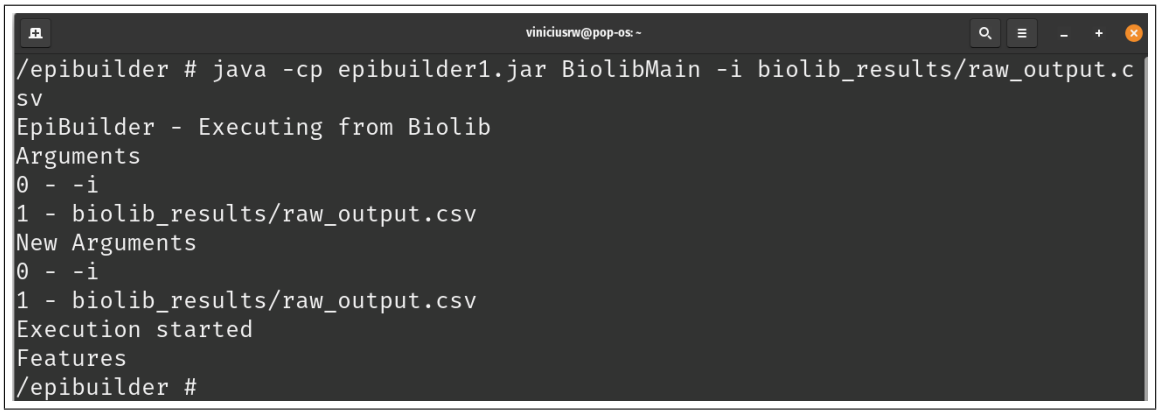

Figura 21. Processamento do arquivo CSV.

A Figura 21 apresenta o resultado ao executar a predição do *Epibuilder*. O comando utilizado é executado através do *Java*, apresentado como o primeiro parâmetro no Terminal com o parâmetro *java*. O *Epibuilder* é executado através de *Classpath* do Java, como indica o parâmetro *-cp*. De acordo com Oracle (2022), o *Classpath* é uma maneira de informar aos aplicativos, incluindo as ferramentas *JDK*, onde procurar classes de usuário. O uso do *Classpath* especifica a *classe Java* que será utilizada para processar o arquivo *CSV*, no qual utilizamos a classe *BiolibMain*. O parâmetro *-i* define que o próximo texto adicionado ao comando será o arquivo *CSV* para ser processado, que na Figura 21 é apontado o arquivo dessa extensão gerado pelo *BepiPred-3.0*. Utilizando um *software* capaz de trabalhar com planilhas eletrônicas, é possível visualizar os resultados produzidos. A Figura 22 apresenta o arquivo *CSV* gerado pelo processamento do *Epibuilder*.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Id	Position	Residue	Bepipred2	Emini	Kolaskar	Chou Fosman	Karplus Schulz	Parker	MW	IP	Hydropathy
2	Tb927.5.4270	1	M	0.08	0.00	0.00	0.00	0.00	0.00	149.21	5.28	1.90
3	Tb927.5.4270	2	D	0.10	0.00	0.00	0.00	0.00	0.00	133.10	3.80	-3.50
4	Tb927.5.4270	3	E	0.13	1.52	0.00	0.00	0.00	0.00	147.13	4.00	-3.50
5	Tb927.5.4270	4	R	0.18	2.67	0.87	0.87	1.04	3.91	174.20	9.75	-4.50
6	Tb927.5.4270	5	Q	0.14	2.67	0.88	0.99	1.03	5.94	146.15	5.52	-3.50
7	Tb927.5.4270	6	M	0.13	3.08	0.89	0.93	1.02	5.33	149.21	5.28	1.90
8	Tb927.5.4270	7	E	0.14	3.08	0.89	0.96	1.03	4.81	147.13	4.00	-3.50
9	Tb927.5.4270	8	D	0.13	2.57	0.90	0.96	1.04	4.96	133.10	3.80	-3.50
10	Tb927.5.4270	9	K	0.13	2.62	0.90	0.91	1.05	4.40	146.19	8.75	-3.90
11	Tb927.5.4270	10	R	0.15	2.96	0.91	0.96	1.05	5.60	174.20	9.75	-4.50
12	Tb927.5.4270	11	T	0.11	3.55	0.92	1.00	1.05	5.30	119.12	5.19	-0.70
13	Tb927.5.4270	12	A	0.09	3.48	0.92	0.93	1.05	4.47	89.09	5.57	1.80
14	Tb927.5.4270	13	R	0.14	3.48	0.91	0.92	1.05	4.26	174.20	9.75	-4.50
15	Tb927.5.4270	14	K	0.12	2.53	0.92	0.92	1.04	2.23	146.19	8.75	-3.90
16	Tb927.5.4270	15	R	0.14	2.53	0.94	0.88	1.02	1.79	174.20	9.75	-4.50
17	Tb927.5.4270	16	R	0.13	2.24	0.91	0.89	1.01	2.60	174.20	9.75	-4.50
18	Tb927.5.4270	17	W	0.11	2.19	0.91	0.89	0.99	2.60	204.23	5.52	-0.90
19	Tb927.5.4270	18	A	0.10	1.94	0.92	0.88	1.00	2.64	89.09	5.57	1.80
20	Tb927.5.4270	19	E	0.14	1.71	0.92	0.85	1.02	3.16	147.13	4.00	-3.50
21	Tb927.5.4270	20	R	0.12	3.19	0.92	0.85	1.05	3.16	174.20	9.75	-4.50
22	Tb927.5.4270	21	Q	0.13	4.24	0.93	0.92	1.08	5.51	146.15	5.52	-3.50
23	Tb927.5.4270	22	E	0.13	4.79	0.91	0.96	1.09	5.81	147.13	4.00	-3.50
24	Tb927.5.4270	23	R	0.14	2.02	0.96	0.94	1.08	3.39	174.20	9.75	-4.50
25	Tb927.5.4270	24	S	0.10	1.87	0.95	1.03	1.07	3.79	105.09	5.24	-0.80
26	Tb927.5.4270	25	R	0.15	2.12	0.93	1.02	1.05	3.53	174.20	9.75	-4.50

Figura 22. Arquivo CSV gerado pelo *Epibuilder*.

A planilha exibida na Figura 22 demonstra o resultado do processamento do *Epibuilder* no *software LibreOffice Calc*. Cada coluna apresentada nessa figura é ordenada al-

fabeticamente, denotando individualmente cada resultado gerado pelo *Epibuilder*. Como resultado, a coluna A (*Id*) demonstra o identificador da proteína. A coluna B (*Position*), apresenta a posição de processamento da proteína. A coluna C (*Residue*) denota o resíduo da proteína. A coluna D (*Bepipred2*) e subsequentes denotam os *scores* dos cálculos de predição para cada algoritmo utilizado pelo *BepiPred-3.0*. O resultado gerado pelo *Epibuilder* oferece uma representação visual e completamente compreensível de cada resultado gerado pelo *BepiPred-3.0*. Para realizar a comparação, a Figura 23 demonstra o resultado inicial gerado pelo *BepiPred-3.0*.

1	Accession,Residue,BepiPred-3.0 score
2	1A2Y_BA C 2,k,0.15557675063610077
3	1A2Y_BA C 2,v,0.15754003822803497
4	1A2Y_BA C 2,f,0.15010391175746918
5	1A2Y_BA C 2,g,0.20969705283641815
6	1A2Y_BA C 2,r,0.18997323513031006
7	1A2Y_BA C 2,c,0.12845809757709503
8	1A2Y_BA C 2,e,0.20289626717567444
9	1A2Y_BA C 2,l,0.06815382093191147
10	1A2Y_BA C 2,a,0.07319370657205582
11	1A2Y_BA C 2,a,0.19596649706363678
12	1A2Y_BA C 2,a,0.1222163662314415
13	1A2Y_BA C 2,m,0.07239404320716858

Figura 23. Arquivo CSV gerado pelo BepiPred-3.0.

A Figura 23 demonstra o arquivo *CSV* gerado pelo *BepiPred-3.0*, no qual torna-se claro que o resultado do processamento de um arquivo *FASTA* tem como resultado uma única coluna com todos os valores concatenados e difíceis de serem interpretados. Como apresentado na Figura 22, o *Epibuilder* processa o *CSV* gerado pelo *BepiPred-3.0*, ajusta os valores, realiza o cálculo de predições através de outros algoritmos e gerou um novo arquivo *CSV* simplificado e legível.

Todos os processos apresentados concluem o desenvolvimento do contêiner, execução dos software *BepiPred-3.0* e *Epibuilder 2.0*. O contêiner gerado foi baseado no Alpine, no qual consome poucos recursos do *host*. Foi desenvolvido também um segundo contêiner, porém baseado na distribuição *Linux Debian*. É possível também simplificar a criação desses componentes através de um *Dockerfile*. A seção 3.3.5 apresenta esse conceito.

3.3.5. Dockerfile

Dentro do contexto do *Docker*, é possível utilizar de recursos que automatizem até mesmo a criação de *Docker Images*. É o caso do *Dockerfile*, um arquivo de texto que contém instruções do que é necessário para se construir a imagem pretendida. Para executar o arquivo recomenda-se que o arquivo esteja isolado em um diretório, pois ao executá-lo ele criará arquivos e informações da imagem no caminho em que se encontra. A criação

da *Docker Image* através do *Dockerfile* deve ser executado pelo comando *docker build .* via Terminal, sendo que o parâmetro "."(ponto) indica que será usado o diretório atual. Por padrão, o *Dockerfile* chamado ao executar o processo irá procurá-lo no diretório em que está sendo executado o terminal. De acordo com Rosa et al. (2022), uma *Docker Image* do *Docker* é definida por meio de um *Dockerfile*, que contém instruções para construir a imagem que contém uma aplicação. A Figura 24 exibe o *Dockerfile* do *Epibuilder* configurado com suas instruções.

```
1  FROM alpine
2  LABEL app="Epibuilder"
3  LABEL version="1.0.0"
4  LABEL description="Bio tools. Small Docker Image for running BepiPred-3.0 on BioLib with Epibuilder 2.0."
5
6  WORKDIR /epibuilder/
7
8  COPY epibuilder-2.jar /usr/local/bin
9  COPY epibuilder.sh /usr/local/bin
10
11 RUN apk update && apk add python3 py3-pip gcc g++ openjdk11-jre
12 RUN pip3 install -U pybiolib && biolib run DTU/BepiPred_3 --help
13 RUN export PATH=$PATH:/usr/local/bin
14 RUN chmod +x /usr/local/bin/epibuilder-2.jar
15 RUN chmod +x /usr/local/bin/epibuilder.sh
16
17 CMD ["sh"]
```

Figura 24. Exemplo de script de Dockerfile.

A Figura 24 apresenta o *Dockerfile* utilizado para automatizar a instalação de todas as dependências e ferramentas para utilizar o *Epibuilder 2.0* e *BepiPred-3.0*. Esse processo permite que contêineres sejam criados a partir dessa *Docker Image*, não sendo necessário instalar todos os itens novamente a cada contêiner gerado. O início do *Dockerfile*, na linha 1, apresenta a *Docker Image* no qual a nossa nova *Docker Image* utilizou como base. As linhas 2, 3 e 4 que utilizam o parâmetro *LABEL* funcionam como rótulos que indicam informações sobre a *Docker Image*. A linha 6 apresenta o parâmetro *WORKDIR*, que define o diretório que o usuário é encaminhado após se conectar ao contêiner. A linha 8 e 9 aplicam o comando *COPY*, sendo responsável por copiar arquivos da máquina *host* e encaminhá-los para um diretório específico na *Docker Image*, que, no *Dockerfile*, é informado que será copiado para o diretório */usr/local/bin*. Foi passado como parâmetro o arquivo *Java* compilado do *Epibuilder* e o *script* de execução do *software*. A linha 11 executa comandos para a *Docker Image* executar. Os comandos fornecidos são a atualização do repositório do *Alpine* e a instalação via pacote *apk* do *Python*, *pip*, *gcc*, *g++* e o *Java Runtime Enviroment*. A linha 12 realiza a instalação da biblioteca *pybiolib* e a execução de exemplo do *BepiPred-3.0*. A linha 13 define que o diretório *usr/local/bin* será adicionado a variável *PATH* da *Docker Image*, permitindo executar o *script* ou o arquivo *JAR* do *Epibuilder* em qualquer diretório do contêiner. As linhas 14 e 15 alteram a permissão do arquivo *JAR* do *Epibuilder* e seu *script* para que possam ser executados no contêiner sem nenhuma restrição. Por fim, na linha 17, é apresentado o comando *CMD*, no qual realiza uma função no terminal integrado a *Docker Image*. O comando executado nesse terminal é o *sh*, tendo como funcionalidade direcionar o usuário ao *terminal* padrão do Alpine, podendo interagir com o contêiner. O *Dockerfile* desenvolvido para o *Debian* possui a instalação *desktop* do *BepiPred-3.0*, tornando-se mais complexa que o *Dockerfile* do *Alpine*. A Figura 25 apresenta o *Dockerfile* desenvolvido para o *Debian*.

```
1  FROM debian
2
3  LABEL app="EpiBuilder"
4  LABEL version="2.0"
5  LABEL description="Bioinformatic tools"
6
7  WORKDIR /epibuilder/
8
9  #install python and pip package manager
10 RUN apt update
11 RUN apt install --no-install-recommends python3 python3-pip git -y
12 #install java jre
13 RUN apt install openjdk-11-jre -y
14 #install NCBI-Blast (for blastp and makeblastdb)
15 RUN apt install ncbi-blast+ -y
16 RUN apt clean -y
17 RUN rm -rf /var/lib/apt/lists/*
18
19 #install all requirements
20 COPY requirements.txt .
21 RUN pip3 install --no-cache-dir -r requirements.txt
22
23 #create alias for python executable to be python3 executable
24 RUN ln -s $(which python3) /usr/bin/python
25
26 #copy CLI script and model esm1b model into image
27 COPY bepipred3_CLI.py .
28 COPY models.tar.gz .
29 COPY esm_encodings.tar.gz .
30 RUN tar -xzf models.tar.gz
31 RUN tar -xzf esm_encodings.tar.gz
32 RUN rm models.tar.gz
33 RUN rm esm_encodings.tar.gz
34 RUN mkdir epibuilder/
35 COPY example_antigens.fasta .
36 COPY raw_output.csv .
37 COPY epibuilder-2.jar epibuilder/
```

Figura 25. Dockerfile versão Debian.

O *Debian* utiliza o seu gerenciador de pacotes *apt* para realizar o gerenciamento de *software* instalados no sistema, possuindo diferenças na sintaxe de seus comandos, se comparado ao *Alpine*. Similar a estrutura do Dockerfile desenvolvido para o *Alpine*, o *Dockerfile* da Figura 25 realiza a instalação da linguagem de programação *Python*, o gerenciador de pacotes *pip*, *Java Runtime Environment* e o pacote de predição *NCBI-Blast+*, utilizado para a predição do *BepiPred-3.0*. A linha 10 do *Dockerfile* do realiza a atualização do *Debian*. A linha 11 apresenta a instalação do *Python*, *pip* e *git*, utilizado para instalar pacotes necessários do *BepiPred-3.0*. As linhas 13 e 14 realizam a instalação do *Java* e *NCBI-Blast+* e, por fim, é realizado a limpeza de pacotes utilizados somente na instalação, na linha 16, utilizando o comando *apt clean*. Os comandos entre a linha 17 e 37 realizam a integração do *BepiPred-3.0* a *Docker Image*, sendo esses comandos baseados no *Dockerfile* do *BepiPred-3.0*.

Para criar a *Docker Image* utilizando o *Dockerfile*, foi utilizado o comando apresentado na Figura 26.

```
viniciusrw@pop-os: ~
viniciusrw@pop-os:~$ docker build -t epibuilder:2.0 .
```

Figura 26. Build da Docker Image.

Para realizar o *build* de uma *Docker Image*, é necessário chamar o comando diretamente do diretório onde está presente o Dockerfile. O comando apresentado na Figura 26 realiza o *build* da *Docker Image* e torna-o disponível para criar contêineres com base na imagem construída.

3.4. Comparação entre Docker Images base

A *Docker Image* base escolhida para o primeiro contêiner é o *Alpine*, que é uma distribuição *Linux* que tem por objetivo ser pequena e simples. Ela foi escolhida visando reduzir o tamanho da *Docker Image*, a fim de que o processo de *download* da imagem e dos conteúdos nela presentes seja mais rápido e possua somente a funcionalidade de interagir com o *BepiPred-3.0* do *Biolib*. O diferencial do *Alpine* é o tamanho da *Docker Image*, se comparado a outras disponibilizadas no *Docker Hub*. O *Alpine* possui uma *Docker Image* de apenas 5 MB, que, se comparado a outras *Docker Images*, elas possuem mais de 50 MB de tamanho, gerando uma grande vantagem para o Alpine. É uma das *Docker Images* mais utilizadas no *Docker Hub*, por conta de seu tamanho e ótima funcionalidade.

Para efeito de comparação, instalando o *Epibuilder* e suas dependências, quando utilizado como imagem base a distribuição *Debian* o tamanho da imagem era de cerca de 972 MB, enquanto a imagem usando como base o *Alpine* tem o tamanho de 527 MB conforme apresentado na Figura 27. Ambas as *Docker Images* comparadas possuem os mesmos *software* instalados.

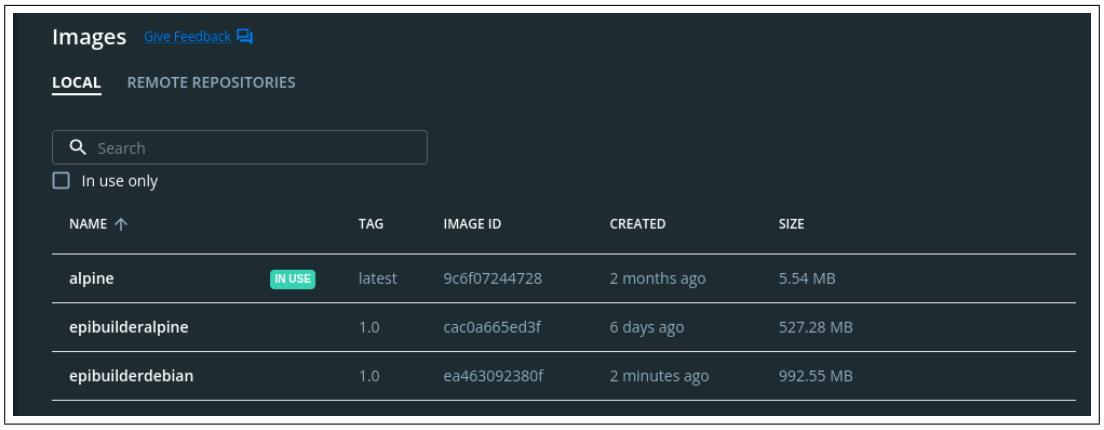

Figura 27. Comparação entre Docker Images.

O segundo *Dockerfile* desenvolvido utiliza a distribuição *Debian*. O *Debian* é uma das distribuições *GNU/Linux* mais populares, não apenas entre os usuários finais, mas também como base para outros sistemas. Além de ser popular, também é uma das maiores compilações de *software* (Amor-Iglesias et al., 2005). A escolha dessa *Docker Image* tem como objetivo utilizar mais funcionalidades e recursos que essa imagem provê, se comparado ao *Alpine*, que entrega o mínimo de recursos para ter uma *Docker Image* muito leve. A *Docker Image* gerada com esse Dockerfile possui uma instalação local do *BepiPred-3.0*. A instalação local permite o uso total de recursos da máquina do *host*, não necessitando de comunicação com os serviços do *BioLib*, agilizando o processamento de arquivos *FASTA*. A Figura 28 apresenta as *Docker Images* geradas para ambas as versões.

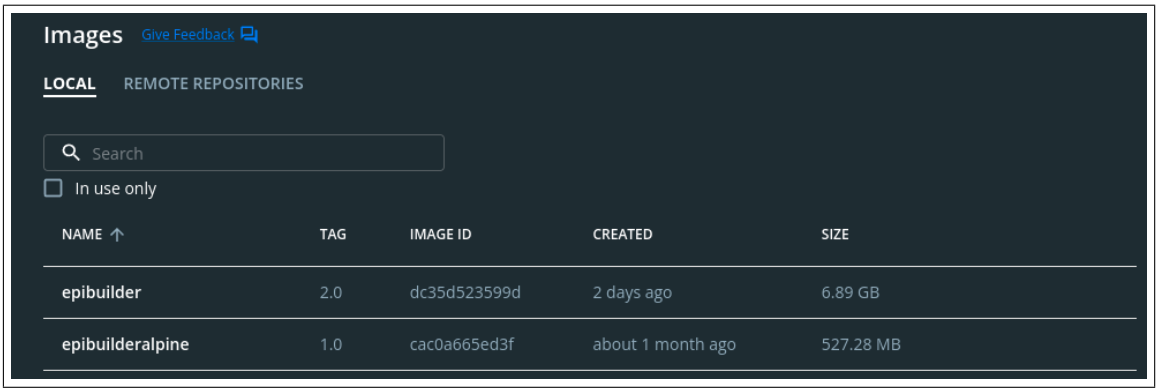

Figura 28. Docker Images desenvolvidas.

Como ponto negativo, essa *Docker Image* possui um tamanho notavelmente maior que a gerada do *Alpine*, possuindo 6.89 GB. Mesmo sendo uma *Docker Image* mais pesada, essa versão possui a completa funcionalidade dos *software Epibuilder* e *BepiPred-3.0*, onde o *Epibuilder* possui uma funcionalidade de executar o *BepiPred-3.0* diretamente pelo seu arquivo executável. Essa integração permite ao usuário executar somente um comando para executar todo o fluxo entre ambos os *software*.

3.5. Publicação no Docker Hub

O *Docker Hub* é um repositório de *Docker Images* e *Dockerfiles*, permitindo o compartilhamento público ou privado desses recursos. Foi disponibilizado nessa plataforma as *Docker Images* desenvolvidas, sendo criados repositórios para armazená-las individualmente. Para realizar o envio das *Docker Images* para o *Docker Hub*, foi necessário criar uma *tag* e anexá-la ao nome do repositório criado na plataforma, posteriormente realizando o *push* para o repositório. A Figura 29 exibe esses processos sendo executados no *Terminal*.

Figura 29. Tag e Push de uma Docker Image.

Realizado o *push*, as *Docker Images* estão disponíveis no *Docker Hub*. A plataforma disponibiliza recursos para descrever a *Docker Image* e um *README* para disponibilizar detalhes ou instruções. Foram definidos as instruções e parâmetros necessários para utilizar a *Docker Image* nesses campos de informação disponibilizados pela plataforma.

3.6. Trello

Trello é uma plataforma *Web* voltada para o gerenciamento de times, desenvolvimento e controle de projetos. Faz isso através do uso de quadros e cartões que simulam um

Kanban, permitindo a separação das atividades em fases e um controle de quem está responsável pela mesma.

No projeto usamos o *Trello* como ferramenta de controle das atividades em cada etapa de execução, tornando fácil a visualização do que está sendo feito no momento, o que está pronto e o que ainda precisa ser feito, como mostrado na Figura 30.

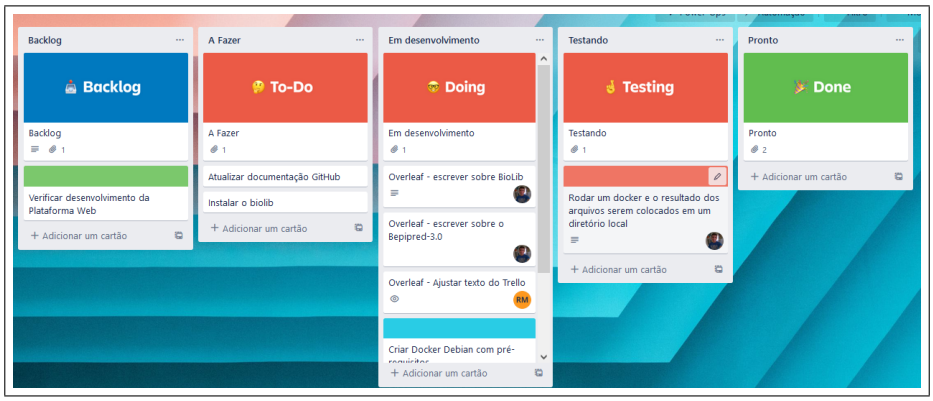

Figura 30. Interface do Trello.

Como exibido na Figura 30, separamos as etapas de desenvolvimento em 5 fases para controle. O quadro *Backlog* se define como o quadro inicial onde se registravam as tarefas a serem executadas para que fossem estudadas, sendo seguido pelo quadro *To-Do* que é o quadro com as atividades já analisadas e que precisam ser executadas. Os quadros *Em Desenvolvimento* e *Testando* nos permitiram controlar as atividades em suas etapas finais, que ao serem concluídas passavam para o quadro *Pronto*, facilitando a visualização do progresso do projeto.

4. Conclusões

Este trabalho teve como objetivo automatizar a instalação dos preditores de epítopo através de *scripts* de configuração. Estes software permitem gerar os resultados em planilhas e diversos outros formatos de arquivo texto, o que auxilia o usuário na manipulação e acesso aos resultados. Através da ferramenta *Docker*, foi desenvolvido uma *Docker Image* que proporciona ao usuário criar um contêiner com um ambiente *Linux* leve e pré-configurado com todas as dependências necessárias para utilizar os software de predição *Epibuilder 2.0* e *BepiPred-3.0*. Além disso, foi possível também utilizar *Scripts* que facilitaram a criação do contêiner *Docker* e o uso dos *software* para a realização dos processos de predição.

O software desenvolvido atingiu todos os objetivos estabelecidos, possuindo o funcionamento esperado e facilitando o uso dos *software* integrados. A complexidade de instalação e uso de *software* de bioinformática é um problema comumente encontrado por conta do foco em desenvolvimento dessas aplicações em sistema *Unix-like*, sendo Sistemas Operacionais um pouco mais técnicos e complexos se comparados ao Sistema Operacional *Windows* no quesito de gerenciamento de *software*. O uso de recursos e ferramentas externas desses *software* dificultam ainda mais a utilização, sendo necessário buscá-las em *websites* externos e usando versões específicas de pacotes obrigatórios durante a instalação, tornando assim mais restritos os acessos aos software de bioinformática.

Durante o desenvolvimento do trabalho, foram encontradas dificuldades na instalação do preditor *BepiPred-2.0*, que, inicialmente, foi utilizado no começo das implementações. O *BepiPred-2.0* utiliza uma versão não suportada mais pela linguagem de programação *Python*, tornando-se difícil de gerenciar a instalação dos pacotes necessários para o preditor inicialmente escolhido. Como solução, em maio de 2022, foi lançada a versão 3.0 do *BepiPred*, oferecendo uma instalação simplificada e utilizando versões atuais do *Python*, descartando a necessidade de realizar um *downgrade* para uma versão antiga. Após o lançamento do *BepiPred-3.0*, foi descontinuado o uso do *BepiPred-2.0* no desenvolvimento do *Epibuilder* diante das dificuldades apresentadas. Outro problema enfrentado foi o tamanho do arquivo da *Docker Image*, possuindo quase 1 *Gigabyte* na *Docker Image* que utiliza o mínimo de recursos, no qual foi necessário trocar a *Docker Image* base, substituindo o *Debian* para o *Alpine*, tendo como resultado uma redução significativa da *Docker Image* gerada.

A solução desenvolvida é linear, ou seja, torna-se necessário seguir os passos corretamente durante a configuração do contêiner e o uso dos *Scripts*. A configuração sem os *Scripts* necessita que o usuário interaja diretamente com conceitos do *Docker* um pouco mais avançados para configurar o seu contêiner manualmente, caso seja necessário configurar desta forma.

Como oportunidade de atividades futuras, melhorias na integração entre o *BepiPred* e *Epibuilder* tornam possível aprimorar a performance ou formar resultados mais detalhados para agregar valor ao *software*. O *BepiPred-3.0* possui uma versão inicial no repositório *PyPI*, sendo possível, futuramente, integrá-lo a uma versão em *Python* do *Epibuilder* e centralizar os processos de ambos em apenas um *software*.

Referências

Abdalkareem, R., Oda, V., Mujahid, S., e Shihab, E. (2020). On the impact of using trivial packages: an empirical case study on npm and pypi. *Empirical Software Engineering*, 25(2):1168–1204.

Alles, G. R., Carissimi, A., e Schnorr, L. M. (2018). Assessing the computation and communication overhead of linux containers for hpc applications. In *2018 Symposium on High Performance Computing Systems (WSCAD)*, pages 116–123. IEEE.

Amor-Iglesias, J.-J., González-Barahona, J. M., Robles-Martínez, G., e Herráiz-Tabernero, I. (2005). Measuring libre software using debian 3.1 (sarge) as a case study: Preliminary results. *UPGRADE The European Journal for the Informatics Professional*, 6(3):13–16.

BioLib (2022). Getting Started. [Online; acesso em 25 de Agosto de 2022].

Boettiger, C. (2015). An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79.

Bui, T. (2015). Analysis of docker security. *arXiv preprint arXiv:1501.02967*.

Chen, H. e Wagner, D. (2002). Mops: an infrastructure for examining security properties of software. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 235–244.

Claro, D. B. e Sobral, J. B. M. (2008). Programação em java. *Livro programando em Java 1ª edição*, page 12.

Clifford, J., Høie, M. H., Nielsen, M., Deleuran, S., Peters, B., e Marcatili, P. (2022).

Bepipred-3.0: Improved b-cell epitope prediction using protein language models. *bioRxiv*.

de Araújo, N. D., de Farias, R. P., Pereira, P. B., de Figueirêdo, F. M., de Morais, A. M. B., Saldanha, L. C., e Gabriel, J. E. (2008). A era da bioinformática: seu potencial e suas implicações para as ciências da saúde. *Estudos de biologia*, 30(70/72).

Docker (2022). Dockerfile reference. [Online; acesso em 12 de Outubro de 2022].

Eder, M. (2016). Hypervisor-vs. container-based virtualization. *Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)*, 1.

Garrett, S. M. (2003). A paratope is not an epitope: Implications for immune network models and clonal selection. In *International Conference on Artificial Immune Systems*, pages 217–228. Springer.

Gholami, A. e Laure, E. (2016). Security and privacy of sensitive data in cloud computing: a survey of recent developments. *arXiv preprint arXiv:1601.01498*.

Haque, M. U., Iwaya, L. H., e Babar, M. A. (2020). Challenges in docker development: A large-scale study using stack overflow. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11.

Ivanov, K. (2017). *Containerization with LXC*. Packt Publishing Ltd.

Jespersen, M. C., Peters, B., Nielsen, M., e Marcatili, P. (2017). Bepipred-2.0: improving sequence-based b-cell epitope prediction using conformational epitopes. *Nucleic acids research*, 45(W1):W24–W29.

Kahuha, E. (2022). LXC vs Docker: Which Container Platform Is Right for You? *Earthly Blog*.

Kerpan, I., Walzem, F., e Kilsen, R. (2020). *Microbiologia III: sistema imunológico*. Cambridge Stanford Books.

Khandhar, M. N. e Shah, M. S. (2019). Docker-the future of virtualization.

Kilsen, R. (2020). *Células B e anticorpos monoclonais*. Microbiologia III: sistema imunológico. Cambridge Stanford Books.

Kwon, S. e Lee, J.-H. (2020). Divds: Docker image vulnerability diagnostic system. *IEEE Access*, 8:42666–42673.

Larsen, J. E. P., Lund, O., e Nielsen, M. (2006). Improved method for predicting linear b-cell epitopes.

Mattos, D. M. F. (2008). Virtualização: Vmware e xen. *Universidade Federal do Rio de Janeiro*.

Moreira, R. S., Filho, V. B., Calomeno, N. A., Wagner, G., e Miletti, L. C. (2022). Epibuilder: A tool for assembling, searching, and classifying b-cell epitopes. *Bioinformatics and Biology Insights*.

MOTTA, V. T. (2010). Aminoácido e proteínas. ____. *Bioquímica*, 1.

Nery, M. (2017). Linguagem de programação java.

Oracle (2022). PATH and CLASSPATH (The Java™ Tutorials > Essential Java Classes > The Platform Environment). [Online; acesso em 18 de Outubro de 2022].

Ozga, W., Quoc, D. L., e Fetzer, C. (2020). A practical approach for updating an integrity-enforced operating system. In *Proceedings of the 21st International Middleware Conference*, pages 311–325.

Pereira, N. G., Oliveira, L. F., Ortega, J. R., Versiani, M. S., de Oliveira, A. M. E., de Oliveira Xavier, A. R. E., e de Sousa Xavier, M. A. (2021). Bioinformática como ferramenta na análise de epitopos antigênicos no design de vacinas contra anaplasma mar-

ginale, leishmania spp., sars-cov-2 e toxina de clostridium septicum. *Brazilian Journal of Development*, 7(4):41634–41650.

Rad, B. B., Bhatti, H. J., e Ahmadi, M. (2017). An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS)*, 17(3):228.

Rogero, M. M. e Tirapegui, J. (2008). Aspectos atuais sobre aminoácidos de cadeia ramificada e exercício físico. *Revista Brasileira de Ciências Farmacêuticas*, 44(4):563–575.

Rosa, G., Scalabrino, S., e Oliveto, R. (2022). Fixing dockerfile smells: An empirical study. *arXiv preprint arXiv:2208.09097*.

Shen, Z. e Spruit, M. (2019). A systematic review of open source clinical software on github for improving software reuse in smart healthcare. *Applied Sciences*, 9(1):150.

Turnbull, J. (2014). *The Docker Book: Containerization is the new virtualization*. James Turnbull.

Yellin, F. e Lindholm, T. (1996). The java virtual machine specification.

Zagalsky, A., Feliciano, J., Storey, M.-A., Zhao, Y., e Wang, W. (2015). The emergence of github as a collaborative platform for education. In *Proceedings of the 18th ACM conference on computer supported cooperative work & social computing*, pages 1906–1917.

Zolkifli, N. N., Ngah, A., e Deraman, A. (2018). Version control system: A review. *Procedia Computer Science*, 135:408–415.