Plataforma de Baixo Custo Baseada em Contêineres para Disponibilização de Cenários de Aulas de Computação em *Single Board Computers*

Augusto Cesar da Silva, Rodolfo Francisco de Oliveira

[1]Instituto Federal de Educação, Ciência e Tecnologia de São Paulo – Campus Hortolândia.
Avenida Thereza Ana Cecon Breda, s/n - Vila São Pedro. Hortolândia-SP – Brasil

augusto.silva@zoho.com, rodolfo.oliveira@ifsp.edu.br

Abstract. *On behalf of the reduced budget for educational institutions and the increase in price of electronic equipment, it is necessary to optimize academic costs. This article proposes the use of Single Board Computers, as an alternative for optimizing the costs of deploying electronic equipment on computer labs in schools, as well as providing more practical classroom scenarios in computer labs. The proposed architecture aims to use a local repository, in which class scenarios will be made available as containers, for future access by students. The behavior of this proposal was analyzed in two different scenarios.*

Resumo. *Com a redução do orçamento para instituições de ensino juntamente ao aumento do preço de equipamentos eletrônicos, se torna cada vez mais necessário otimizar os custos acadêmicos. Este artigo propõe a utilização de computadores de placa-única em conjunto com contêineres como alternativa para a otimização de custos de implantação de equipamentos eletrônicos em laboratórios escolares. A arquitetura proposta visa a utilização de um repositório local, no qual os cenários de aula serão disponibilizados como contêineres, para futuro acesso pelos alunos. O comportamento desta proposta foi analisada em dois cenários distintos.*

1. Introdução

Com a redução orçamentária do Instituto Federal de Educação, Ciência e Tecnologia do Estado de São Paulo [IFSP 2021], juntamente ao aumento do preço de equipamentos eletrônicos [Perrin 2020] [Baptista 2021], se torna cada vez mais necessário otimizar os custos de implantação dos laboratórios de informática de tal instituição de ensino, e outras instituições de ensino com baixa capacidade orçamentária. Além disso, com o crescente aumento do número de ferramentas (linguagens de programação, *frameworks*, bancos de dados, etc.), bem como a complexidade e tempo para a instalação das mesmas, se torna necessário uma maneira de otimizar a disponibilização de cenários de aula entre a comunidade acadêmica, como por exemplo entre professores e alunos durante as aulas nos laboratórios de informática.

Este trabalho propõe a utilização de computadores de placa-única (*Single Board Computers*) ao invés dos tradicionais computadores de mesa, visando reduzir o gasto com a implantação dos laboratórios; além disso, a proposta apresentada visa a utilização de um repositório local onde os professores poderão disponibilizar cenários de aula baseados em contêineres, para que sejam disponibilizados e utilizados pelos alunos nos *Single*

Board Computers. Também será utilizado um gerenciador de contêineres em cada máquina, possibilitando que apenas os contêineres sejam instalados na máquina e, a cada aula, apenas as imagens e serviços da disciplina serão transferidas no início. Para isso, o código-fonte e o arquivo para iniciar os contêineres de cada aula serão armazenados no *Git* [Loeliger and McCullough 2012], uma ferramenta de versionamento de código-fonte, a qual possibilita que o professor compartilhe o código-fonte e realize o versionamento dentro da ferramenta. O aluno, por sua vez, irá executar um *script shell*, no qual serão selecionadas a matéria e respectiva aula que se deseja acessar.

Por fim, será avaliado como a proposta impacta no desempenho computacional do *Single Board Computer*, levando em consideração cenários de aula criados pelos professores de matérias do curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Ciência e Tecnologia de São Paulo - Campus Hortolândia.

O restante do trabalho é organizado como se segue: No Capítulo 2 é apresentado uma revisão bibliográfica com os principais conceitos abordados no trabalho; No Capítulo 3 é apresentado os trabalhos correlatos; No Capítulo 4 é apresentado a arquitetura proposta para realizar os ambientes de aula; No Capítulo 5 é apresentada a metodologia utilizada para execução dos testes de performance; No Capítulo 6 é apresentado os resultados obtidos dos testes; No Capítulo 7 é apresentada a conclusão.

2. Revisão Bibliográfica

Para prosseguir com a proposta de implantação desse modelo de laboratório, foi necessário compreender algumas ferramentas: *Docker*, como o orquestrador de contêineres, *Git* como o gerenciador de versionamento de código, e o *Raspberry Pi* que foi o *Single Board Computer* escolhido para essa implantação. A escolha de cada uma dessas ferramentas será fundamentada e explicada ao longo deste capítulo.

2.1. Docker

O *Docker* permite a instalação de uma aplicação de forma isolada dentro de um sistema operacional, de acordo com o que for definido em cada imagem. Uma imagem *Docker* cria um contêiner *Docker*, que contém tudo o que é necessário para a aplicação ser executada de forma isolada. Por exemplo: executando a imagem com *Ubuntu* 16.04 com a aplicação *PHP 7* tem-se o *PHP 7* sendo executado em um contêiner utilizando a imagem do *Ubuntu* 16.04 como base.

Cada imagem *Docker* é como um molde de quais camadas de serviço executar para que seja possível rodar a aplicação. O intuito das imagens é de serem leves e rápidas, por isso geralmente são arquivos pequenos. As imagens são executadas durante a construção do contêiner *Docker*.

Os contêineres são como uma instância de execução de uma imagem *Docker*, e podem ser executados um ou mais contêineres para cada imagem, de acordo com a necessidade. Os contêineres são executados dentro de um mesmo Sistema Operacional ou *Kernel*, os fazendo consumir CPU, RAM e armazenamento de um SO mesmo que sejam executados múltiplos contêineres [Goasguen 2015].

Em comparação com as máquinas virtuais, os contêineres consomem menos recursos de *hardware* por não precisarem executar um sistema operacional completo para cada

contêiner, como acontece com as máquinas virtuais. Os mesmos também são inicializados mais rapidamente que estas por não necessitarem executar um Sistema Operacional [Poulton 2020].

Pode-se armazenar essas imagens em um servidor *Docker* privado ou público, chamado de *Docker Registry*, na nuvem ou dentro de uma rede privada compartilhada. Para que sempre que um cliente *Docker* faça a requisição de uma imagem, ela seja requisitada para este servidor, responsável por armazenar a informação da imagem. Dessa forma pode-se executar vários ambientes diferentes na mesma máquina cliente, sem necessidade de instalar serviços além do *Docker*.

Para realizar a requisição do *download* de uma imagem, é utilizado o comando *pull* do *Docker*, que busca no *registry* padrão (*DockerHub*) ou no *registry* privado, caso seja passada o endereço desse *registry*. Também é possível alterar o conteúdo de uma imagem, ou publicar imagens novas no *registry* através do comando *push* do *Docker*, que envia a imagem construída localmente para o *registry* designado.

O processo de obter uma imagem de um *registry* para o repositório local da máquina é executado pelo comando *docker image pull*. Por padrão, o comando *image pull* sempre buscará imagens do *DockerHub*, a menos que seja especificado o *registry* a ser realizado o *download*.

Para criar e registrar novas imagens ou alterações em imagens já existentes é utilizado o comando *docker push*, que enviará para o *registry* a imagem já construída de acordo com o *Dockerfile*. É possível criar novas imagens, com configurações de acordo com o que for necessário para a imagem em questão, através da criação de um novo arquivo *Dockerfile*, que contém as instruções do que é necessário rodar no contêiner para executar a imagem [Turnbull 2014].

É possível executar mais de uma imagem em diferentes contêineres na mesma rede, montando, por exemplo, uma rede com *Wordpress* e *MySQL* se comunicando através do orquestrador *Docker Compose* que permite através de um único comando iniciar um ambiente de desenvolvimento com todas as imagens necessárias [Foundation 2021].

Na Figura 1 é apresentada a arquitetura do *Docker* envolvendo os comandos *push* e *pull*.

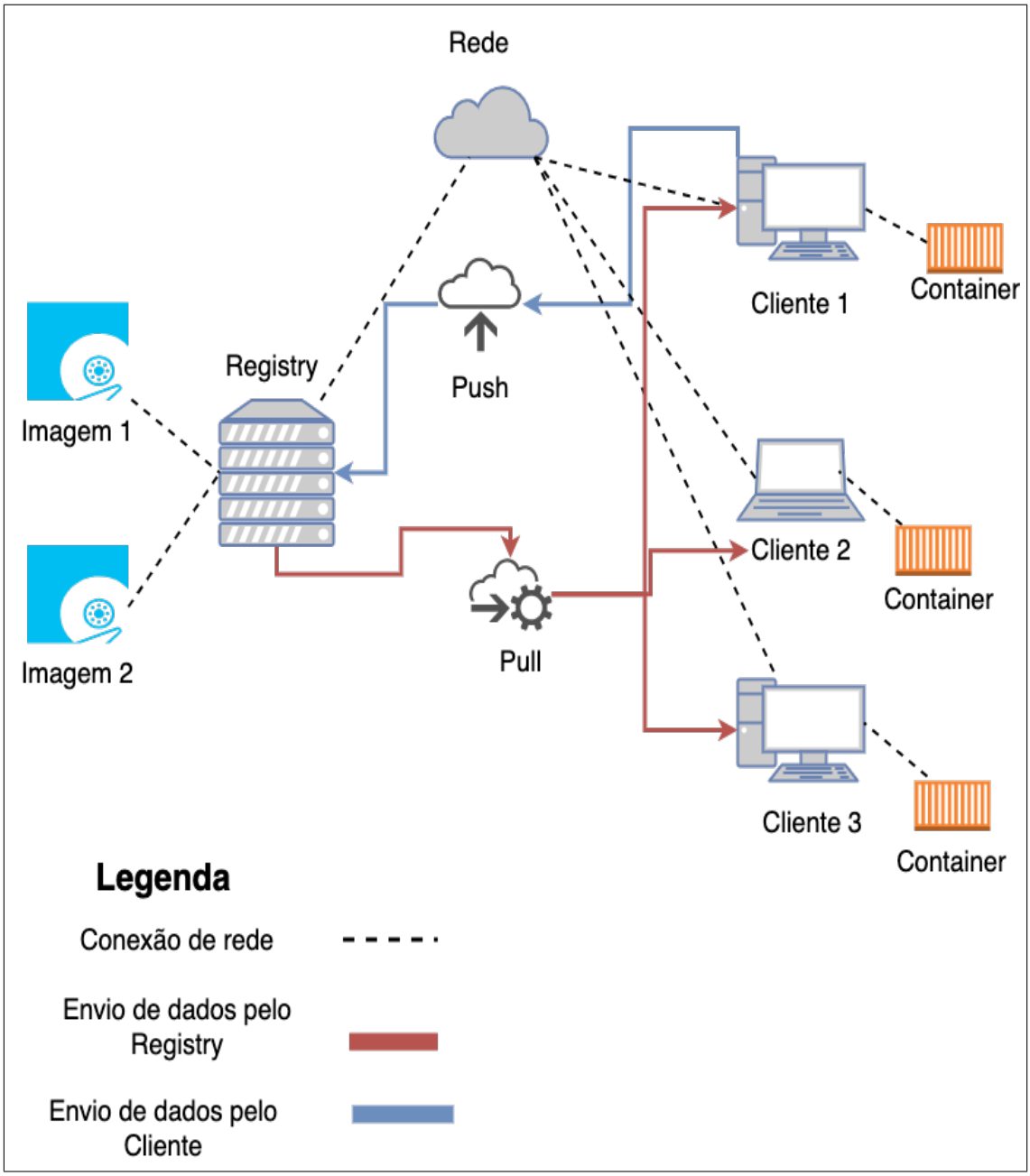

Figura 1: Arquitetura Docker

2.2. Git

O *Git* é uma ferramenta de versionamento de código, que permite a distribuição de código através de um servidor para múltiplos clientes. O *Git* armazena uma versão base do arquivo, e a cada mudança adiciona apenas a alteração que foi realizada [Loeliger and McCullough 2012]. Através do comando *git push* é possível publicar todas as alterações feitas localmente para um servidor *Git* que pode ser instalada em uma máquina local, ou na Internet [Chacon 2014]. Qualquer usuário com acesso ao servidor, poderá realizar o *download* dos arquivos através do comando *git pull*. Na Figura 2 é apresentada a arquitetura do Git.

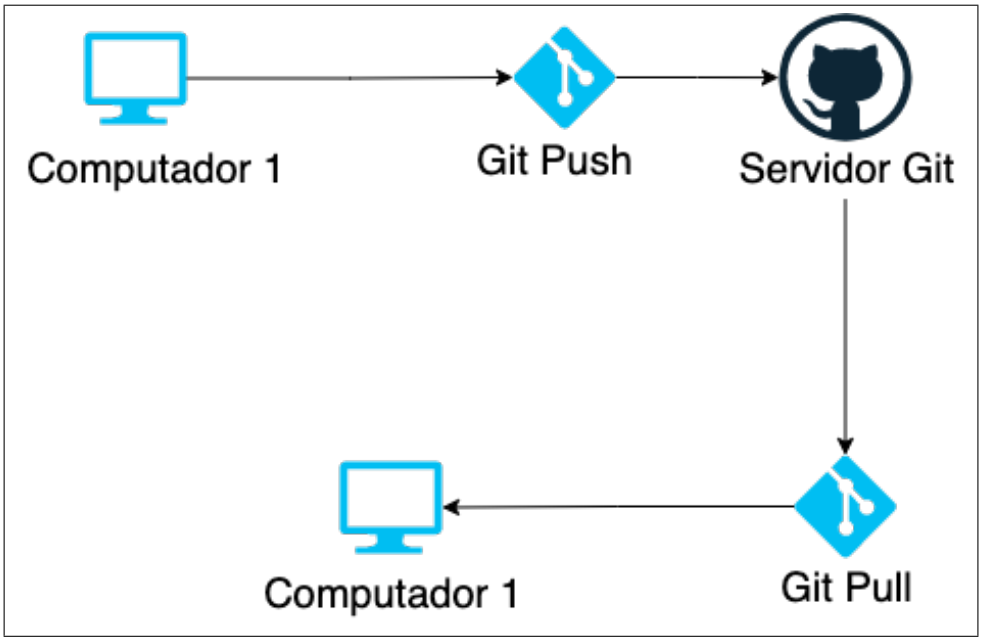

Figura 2: Arquitetura Git

2.3. Raspberry Pi

O *Raspberry Pi* foi criado pela *Raspberry Pi Foundation* para auxiliar o ensino de computação, através de um computador de baixo custo capaz de substituir um computador de mesa para tarefas básicas e de programação [Foundation 2012]. Na Tabela 1 se encontra as especificações do *Raspberry Pi 3 B+*, que será o modelo utilizado neste trabalho.

Tabela 1. Especificações do *Raspberry Pi 3 B+*

Processador	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
Memória RAM	1GB LPDDR2 SDRAM
Conexão de internet	2.4GHz e 5GHz padrão IEEE 802.11.b/g/n/ac wireless LAN
Conexões	Ethernet Gigabit, e Bluetooth 4.2
Conector de video	HDMI
USB 2.0	4 portas
Entrada de energia	5V/2.5A
Memória permanente	Micro SD

Em comparação com outros *Single Board Computers*, como o *BeagleBone Black*, o *Raspberry Pi 3 B+* é um *Single Board Computer* de maior distribuição no Brasil, possui uma documentação bem acessível na Internet, e foi bem aceito como computador de baixo custo pela comunidade, possuindo vários exemplos de projetos disponíveis. Pode-se utilizar o *Raspberry Pi* também para controlar dispositivos externos como motores servo e luzes [Balon and Simic 2019]. Sendo assim, o *Raspberry Pi* é uma opção completa, possibilitando o uso no laboratório para as matérias de computação e, eventualmente, com oficinas de robótica.

A escolha do *Raspberry Pi* como plataforma principal de computação se dá pela possibilidade de ser utilizado como um computador de baixo custo. O mesmo custa cerca de 20 à 35 euros, possui uma compatibilidade com periféricos como teclado e mouse, a disponibilidade de um sistema operacional criado especificamente para o *Raspberry Pi*, e a possibilidade de rodar linguagens de programação e o *Docker* [Valov and Valova 2019].

3. Trabalhos Correlatos

Durante a pesquisa, não foi encontrado nenhum trabalho correlato que tenha como foco utilizar o *Docker* juntamente a um computador de baixo-custo para o ensino de computação. No entanto, utilizando o *site IEEE Explore*, e os seguintes termos: *Raspberry Pi, Education, Computer Science education*, foram encontrados dois artigos que apresentam um cenário onde se utiliza um computador de baixo custo para as aulas que envolvem programação.

O artigo *Make Space for the Pi* [Bruce et al. 2015] explora a utilização de um modelo antigo do *Raspberry Pi* (a 1º versão) em disciplinas da área de Ciência da Computação do Ensino Superior, como administração de banco de dados e arquitetura da computação. O *Raspberry Pi* foi utilizado em um laboratório de forma exclusiva. Já em outro laboratório, o mesmo foi utilizado em conjunto com um computador do tipo desktop. Os testes se mostraram promissores, demonstrando que o *Raspberry Pi* é uma ferramenta valiosa para o estudo de assuntos relacionados a *hardware* e *software*.

O artigo *Raspberry Pi as a Tool to Combine Different Courses Part of University Education* [Valov and Valova 2019] abrange o uso do *Raspberry Pi* como computador principal para os alunos aplicarem o que foi aprendido nas matérias de Desenvolvimento *Web*, Administração de Banco de Dados, e Programação de Sistemas. O uso do Raspberry foi bem aceito pelos alunos, e 95% deles ficaram satisfeitos com o que foi desenvolvido como resultado do projeto, e inclusive apoiam o uso do *Raspberry Pi* para outras disciplinas.

Em nenhum dos trabalhos correlatos foi utilizado uma estrutura semelhante à da proposta presente neste trabalho, juntamente ao *Raspberry Pi*.

4. Metodologia Utilizada

Foi utilizada a metodologia ágil *Kanban* para o desenvolvimento deste trabalho, a mesma consiste em separar cada tarefa em um cartão, e inseri-las em uma das três colunas: A fazer, Em progresso, e Feito. O método foi escolhido por ser simples de implementar, e gerar entregas contínuas do trabalho e permitir a revisão de cada item após a entrega. [Hammarberg and Sundén 2014]

5. Arquitetura proposta

A arquitetura proposta por este trabalho consiste em um servidor na rede local que será responsável por hospedar a ferramenta de versionamento, na qual o professor é responsável por criar o código da aula em sua máquina e enviar esse código de aula para o servidor na rede local, através da ferramenta de versionamento. Cada aluno é responsável por acessar o código disponível na rede local através da ferramenta de versionamento, e

executar o código da aula em seu computador, e todos os serviços necessários serão executados em contêineres de acordo com o código da aula. Na Figura 3 é apresentada uma representação visual da arquitetura proposta.

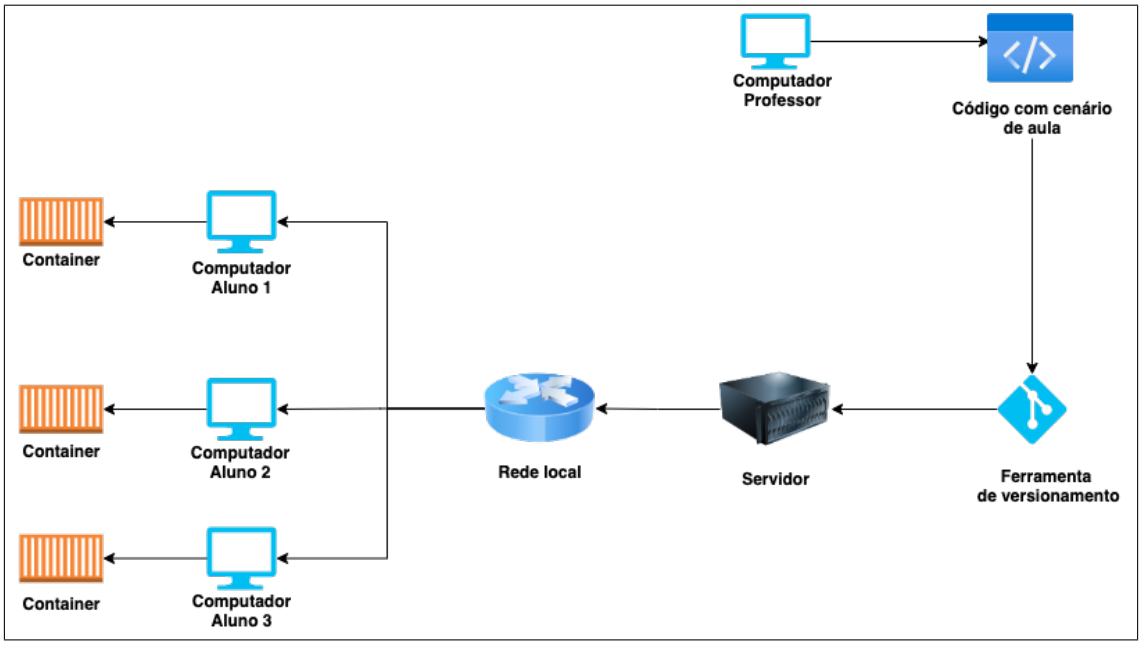

Figura 3: Arquitetura Proposta

6. Materiais e Métodos

O Sistema Operacional a ser executado no *Raspberry Pi* para executar os testes foi o *Ubuntu Server* para ARM64 versão 20.04.3 LTS [Canonical 2021] em conjunto com a interface gráfica *KDE Plasma* [KDE 2021], para facilitar o acesso aos documentos e visualização do resultados. Para o monitoramento de consumo de processamento computacional, memória, rede, e uso de disco foi utilizado a ferramenta gratuita NMON [IBM 2021], e os dados gerados pela ferramenta foram analisados utilizando o *pyNmonAnalyzer* [Lee 2021] para a geração dos gráficos de consumo de memória e processamento computacional.

Os testes realizados foram de consumo de processamento computacional e memória, cujo objetivo é medir consumo de processamento computacional, e verificar se o consumo se mantém estável durante a execução dos cenários, também foi realizado o teste de consumo de rede, com o objetivo de monitorar o uso da placa de rede durante a execução dos cenários. Por fim, foi realizado o teste de uso de disco, a fim de identificar se o espaço em disco é suficiente para a execução dos cenários.

Os testes foram realizados utilizando como base dois cenários distintos: o primeiro cenário utilizando uma aplicação em PHP que realiza o cadastro de clientes, a qual possui páginas para realizar a inserção, alteração e remoção de clientes utilizando a integração com um banco de dados em PostgreSQL, e o segundo cenário utilizando uma aplicação em Java com a biblioteca *Spark* para hospedar o mesmo serviço de cadastro de clientes, e o banco de dados MariaDB. Ambos cenários foram baseados em aulas da

matéria de Desenvolvimento de Sistemas *Web*.

O objetivo dos testes é analisar como o *Raspberry Pi* se comporta ao ser utilizado nessas condições. As métricas utilizadas foram o consumo de memória RAM ao longo do tempo de execução, o consumo de processamento computacional, a porcentagem de utilização do disco, e o consumo de rede.

7. Resultados Obtidos

Os resultados obtidos através da análise de performance utilizando a ferramenta NMON, foram analisados através de gráficos gerados pela ferramenta *pyNmonAnalyzer* para uma representação visual dos dados obtidos.

7.1. PHP e PostgreSQL

Os resultados obtidos utilizando o cenário de aula com PHP e PostgreSQL mostram que, conforme a Figura 4, em nenhum momento a porcentagem consumo de processamento ultrapassou 65%, durante o processo de execução do código e dos contêineres. Fora os picos causados pela instalação dos contêineres e primeira execução do código, o consumo de processamento se manteve abaixo dos 30%. Conforme a Figura 5, neste cenário, durante os momentos de instalação dos contêineres e de primeira execução do código, a porcentagem de ocupação do disco de memória física teve picos, chegando em 100% em alguns momentos. Durante o período de intervalo desses procedimentos, a porcentagem se manteve abaixo dos 60%. O consumo de memória se manteve estável entre 60% e 80% durante todo o processo, conforme mostra a Figura 6. A rede só foi utilizada durante o *download* do código, conforme mostra a Figura 7, pois o código roda localmente após o primeiro *download*.

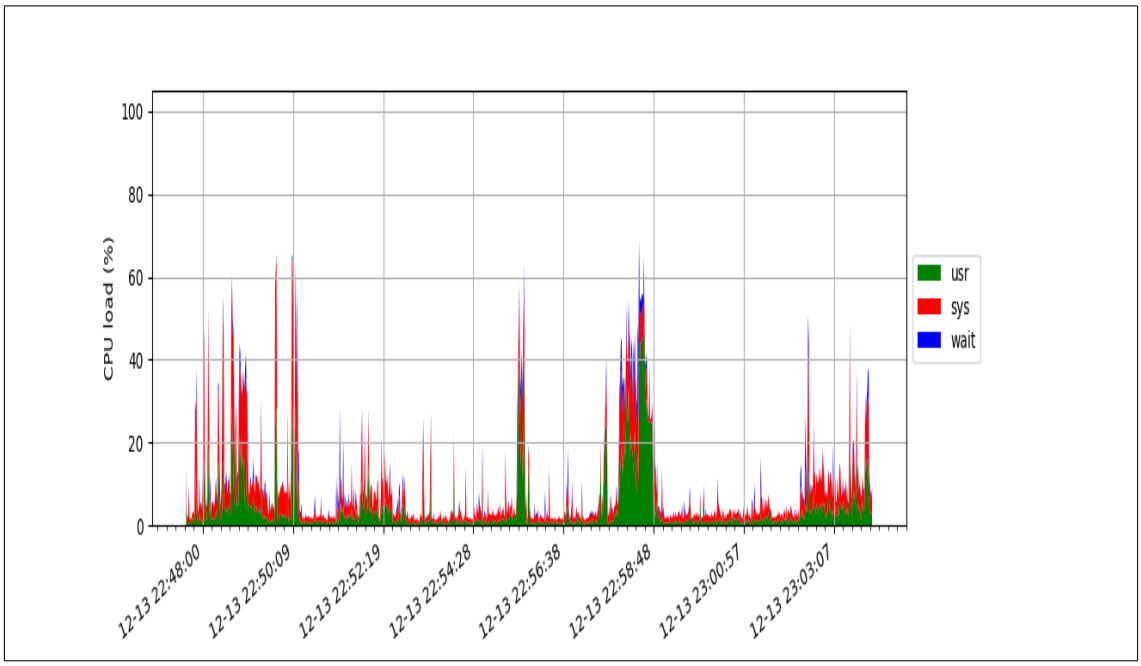

Figura 4: Porcentagem de consumo de processamento computacional ao longo do tempo de execução do cenário com PHP

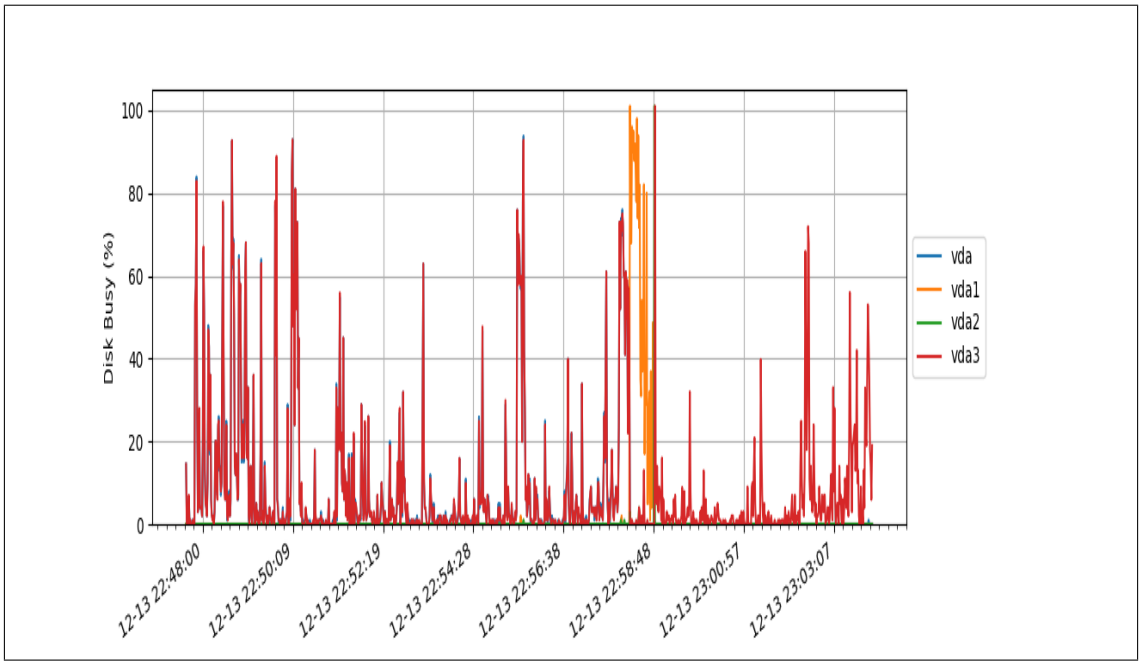

Figura 5: Porcentagem de ocupação do disco de memória física ao longo do tempo de execução do cenário com PHP

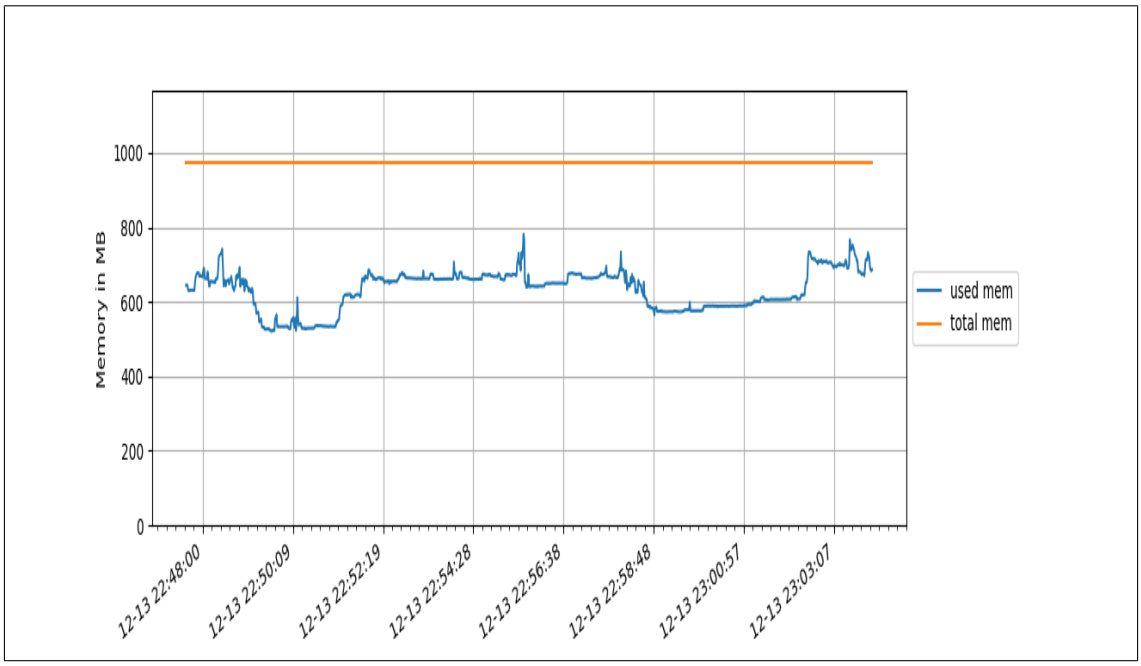

Figura 6: Consumo de memória RAM ao longo do tempo de execução do cenário com PHP

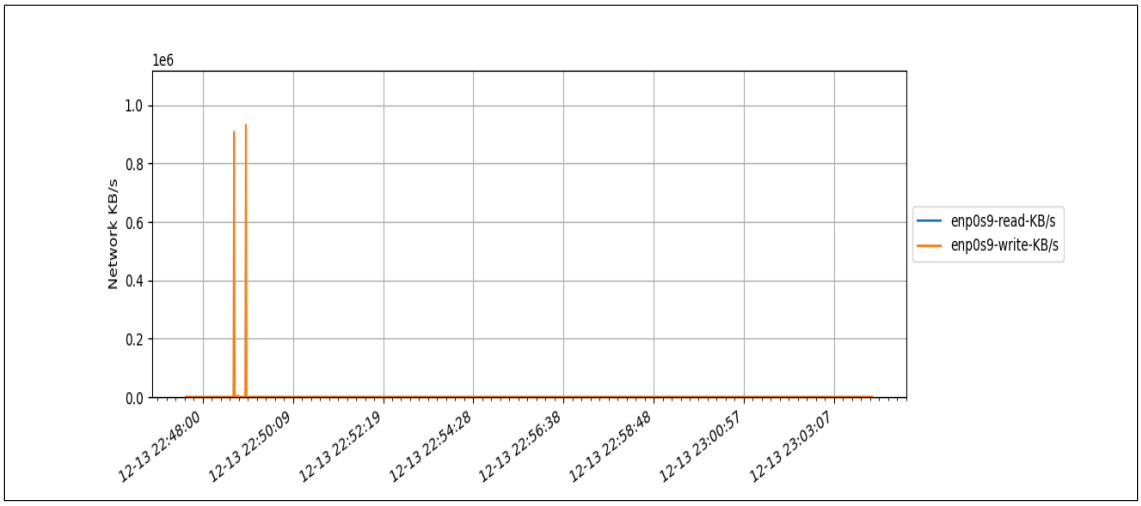

Figura 7: Consumo de rede ao longo do tempo de execução do cenário com PHP

7.2. Java, *Spark*, e MariaDB

Os resultados obtidos utilizando o cenário de aula com Java, Spark, e MariaDB mostram que, conforme a Figura 8, em nenhum momento a porcentagem consumo de processamento ultrapassou 65% durante o processo de execução do código e dos contêineres. Fora os picos causados pela instalação dos contêineres e primeira execução do código, o consumo de processamento se manteve abaixo dos 20%. Conforme a Figura 9, neste cenário, durante os momentos de instalação dos contêineres e de primeira execução do código, a porcentagem de ocupação do disco de memória física teve picos, chegando perto de 100% em alguns momentos. Durante o período de intervalo desses procedimentos, a porcentagem se manteve abaixo dos 40%. O consumo de memória se manteve estável entre 55% e 75% durante todo o processo, conforme mostra a Figura 10. A rede foi utilizada durante o *download* do código, conforme mostra a Figura 11, e com certa frequência no final do teste, pois as requisições do exemplo de código utilizado são realizadas pela rede local.

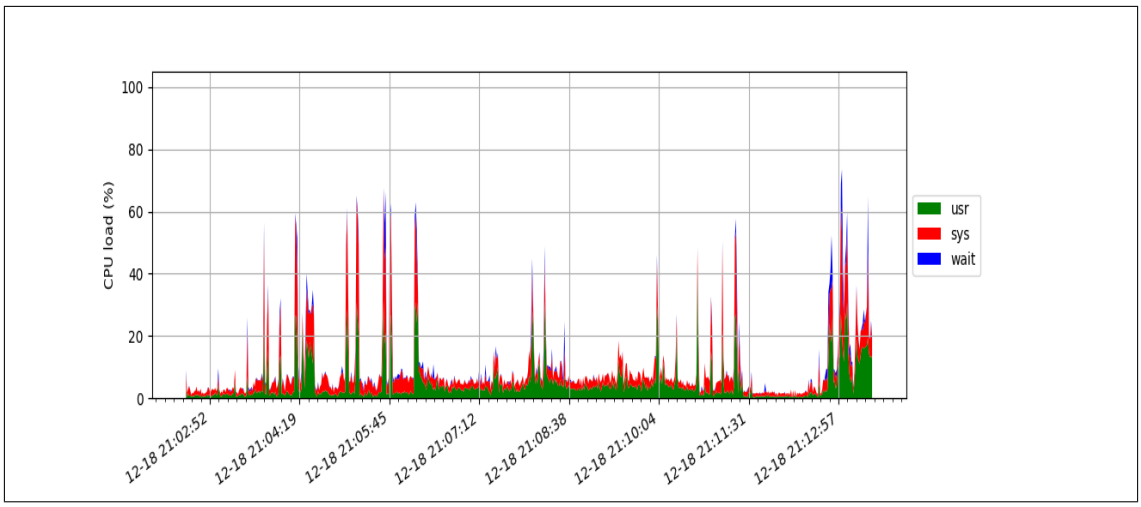

Figura 8: Porcentagem de consumo de processamento computacional ao longo do tempo de execução do cenário com Java

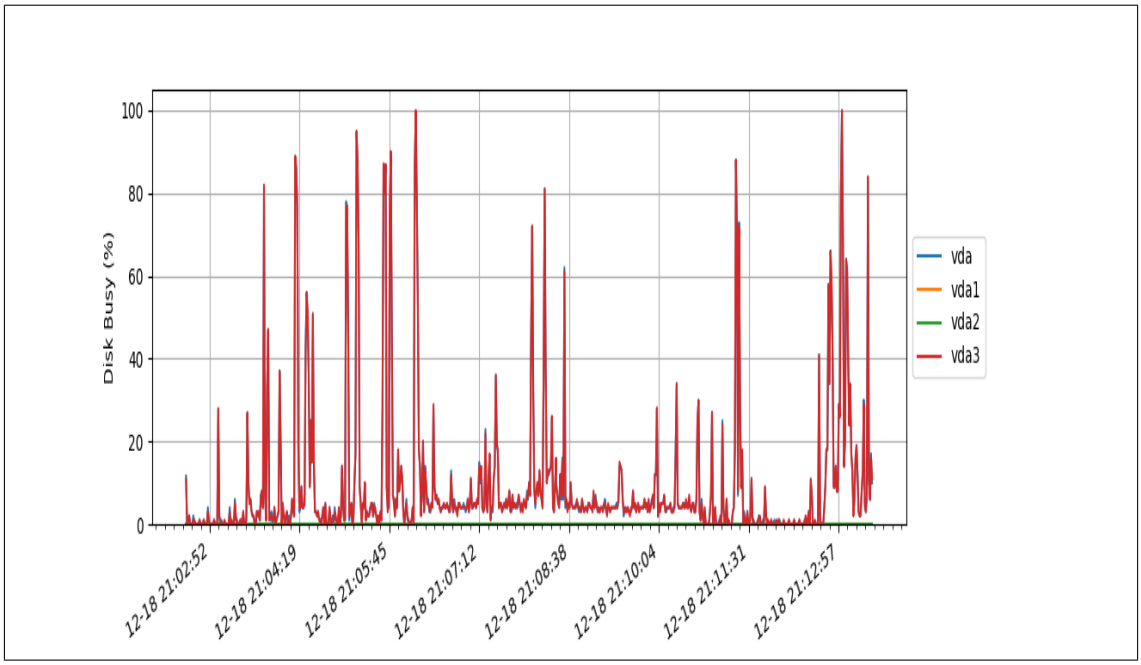

Figura 9: Porcentagem de ocupação do disco de memória física ao longo do tempo de execução do cenário com Java

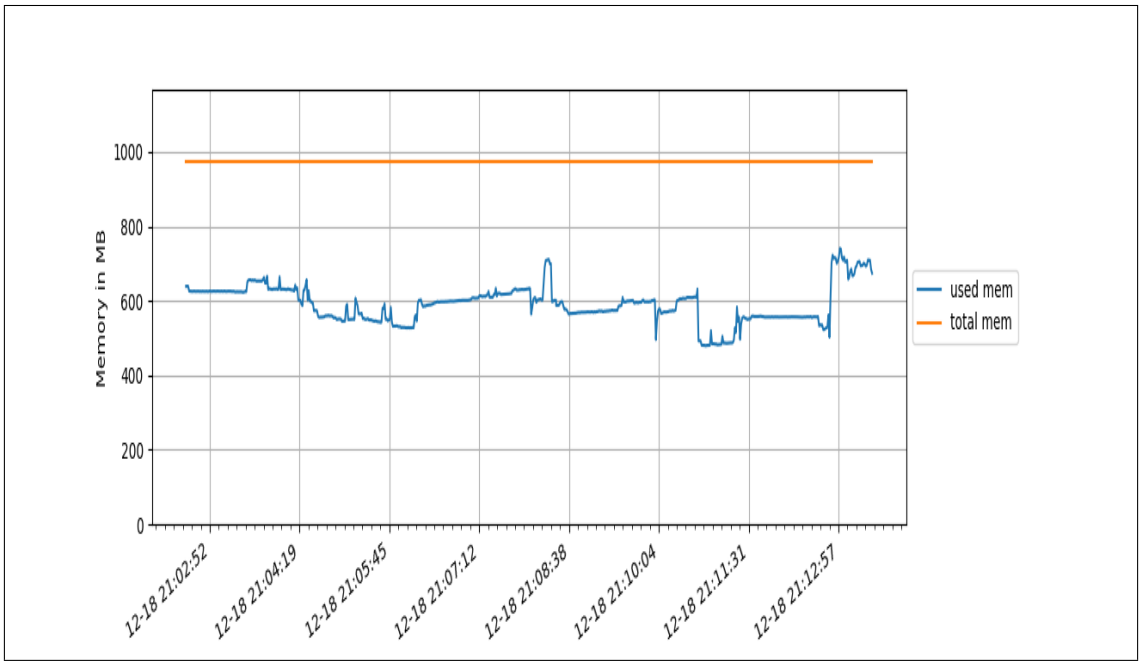

Figura 10: Consumo de memória RAM ao longo do tempo de execução do cenário com Java

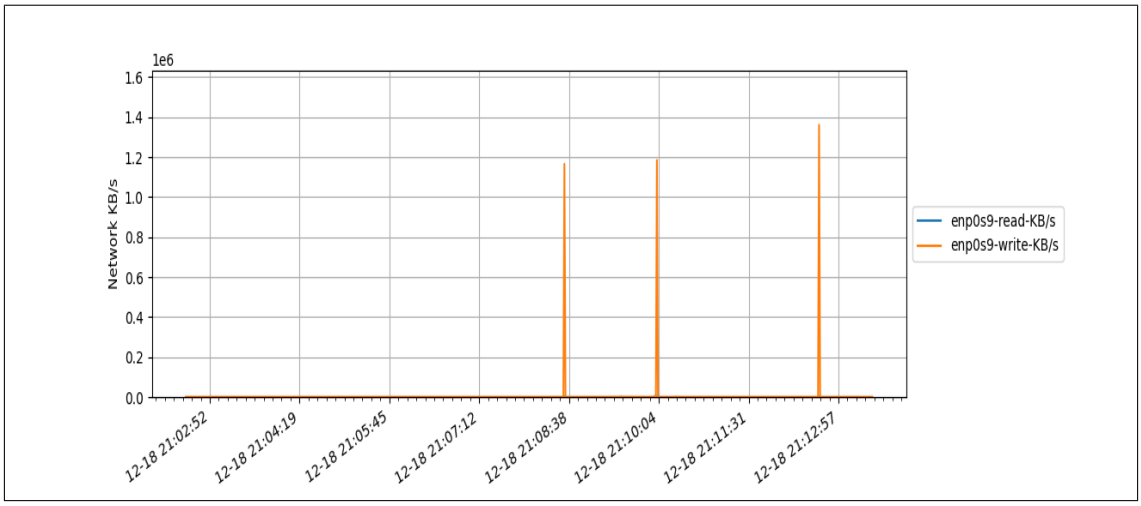

Figura 11: Consumo de rede ao longo do tempo de execução do cenário com Java

7.3. Comparação de custo

Cada kit de computador de alto desempenho Lenovo ThinkCentre M75s foi adquirido por R$6570,00 no ano de 2020 pelo IFSP de Hortolândia, já incluso teclado, mouse e monitor.

Na Tabela 2 se encontra as informações do modelo de computador utilizado nos laboratórios do IFSP de Hortolândia atualmente.

Tabela 2. Especificações dos computador Lenovo ThinkCentre M75s

Processador	AMD Ryzen 7-5700G
Memória RAM	16GB DDR4-3200 SODIMM
Conexão de internet	2.4GHz e 5GHz padrão IEEE 802.11.b/g/n/ac LAN
Conexões	Ethernet Gigabit
Conector de video	1 HDMI e 2 Displayport
USB	4 portas USB 2.0 e 4 portas USB 3.2
Entrada de energia	Fonte interna de 310W
Memória permanente	256GB SSD

O preço médio do *Raspberry Pi 3 Model B+* é de R$485,00 [RoboCore 2021], um monitor Lenovo de 19 polegadas custa R$1000,00 [Lenovo 2022b], e um kit de teclado e mouse Lenovo custa R$340,00 [Lenovo 2022a], totalizando um custo total de R$1825,00 para montar cada conjunto de computador para aula.

A diferença de preço entre os kits é de R$4745,00. Os preços podem ser alterados devido a recente escassez de chips que vem afetando o mercado de eletrônicos mundialmente. [Canaltech 2021]

8. Conclusão

Este artigo visou analisar o poder computacional da *Raspberry Pi* para executar cenários de aulas utilizando o *Docker*, através do *Ubuntu Server* para ARM64 versão 20.04.3 LTS

como sistema operacional em conjunto com a interface gráfica *KDE Plasma*, ao invés de realizar a instalação de cada requisito para aula específica. Isso possibilitou executar os cenários com linguagens de programação e sistema de bancos de dados diferentes, sem nenhuma outra instalação prévia além do *Docker*. Através dos dados coletados e apresentados no capítulo 7, é possível concluir que a arquitetura proposta é capaz de ser executada no *Raspberry Pi*, utilizando *Ubuntu Server* para ARM64 versão 20.04.3 LTS em conjunto com a interface gráfica *KDE Plasma*, pois a plataforma tem poder computacional suficiente para executar os cenários com PHP e Java, sem apresentar nenhum problema.

Foram utilizados os conhecimentos adquiridos durante as aulas das matérias de Arquitetura de Computadores, Redes de Computadores e Serviços de Rede, Metodologia Científica, e Projeto de Sistemas I e II e também conhecimentos extracurriculares sobre computação em contêineres e versionamento de *software*.

8.1. Sugestões de Trabalhos Futuros

Como trabalhos futuros pode ser realizado efetivamente a implementação do modelo sugerido pelo artigo em um laboratório de informática, a fim de validar o modelo.

Pode ser também realizado a pesquisa de viabilidade pedagógica a fim de examinar o impacto da implementação do modelo de laboratório proposto por este artigo.

Também pode ser realizado uma pesquisa que faça a comparação com outro modelo de laboratório que utilize um servidor principal mais potente e vários terminais ligados a esse servidor.

Referências

Balon, B. and Simic, M. (2019). Using Raspberry Pi Computers in Education. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 671–676, Opatija, Croatia. IEEE.

Baptista, R. (2021). Pandemia atrasa fabricação de peças, e isso vai encarecer novos eletrônicos.

Bruce, R. F., Brock, J. D., and Reiser, S. L. (2015). Make space for the Pi. In *SoutheastCon 2015*, pages 1–6, Fort Lauderdale, FL, USA. IEEE.

Canaltech (2021). Celulares e computadores ficarão ainda mais caros em 2022; entenda o motivo.

Canonical (2021). Ubuntu for arm: Download.

Chacon, S. (2014). *Pro Git*. The expert's voice in software development. Apress, New York, NY, second edition edition.

Foundation, D. (2021). Overview of Docker Compose.

Foundation, R. P. (2012). Raspberry Pi Foundation - About Us.

Goasguen, S. (2015). *Docker cookbook*. O'Reilly Media, Inc, Sebastopol, CA, first edition edition. OCLC: ocn899229266.

Hammarberg, M. and Sundén, J. (2014). *Kanban in action*. Manning.

IBM (2021). nmon for linux.

IFSP (2021). Painel Orçamentário IFSP.

KDE (2021). Home.

Lee, M. A. (2021). Madmaze/pynmonanalyzer: Python tool for reformatting and plotting/graphing nmon output.

Lenovo (2022a). Kit teclado e mouse sem fio lenovo essential.

Lenovo (2022b). Monitor thinkvision e20-1b de 19,5 polegadas.

Loeliger, J. and McCullough, M. (2012). *Version control with Git*. O'Reilly, Beijing, second edition edition.

Perrin, F. (2020). Preços de eletrodomésticos e eletrônicos sobem apesar de mais concorrência e procura menor.

Poulton, N. (2020). *Docker Deep Dive*. O'Reilly Media Company. OCLC: 1226442166.

RoboCore (2021). Raspberry pi 3 - model b+ anatel.

Turnbull, J. (2014). *The Docker Book*. O'Reilly Media Company. OCLC: 1137386966.

Valov, N. and Valova, I. (2019). Raspberry Pi as a Tool to Combine Different Courses Part of University Education. In *2019 18th International Conference on Information Technology Based Higher Education and Training (ITHET)*, pages 1–5, Magdeburg, Germany. IEEE.

Documento Digitalizado Restrito

Artigo Final

Assunto: Artigo Final
Assinado por: Rodolfo Oliveira
Tipo do Documento: Anexo
Situação: Finalizado
Nível de Acesso: Restrito
Hipótese Legal: Direito Autoral - conservar a obra inédita (Art. 24, III, da Lei nº 9.610/1998)
Tipo do Conferência: Documento Original

Documento assinado eletronicamente por:

- **Rodolfo Francisco de Oliveira**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 25/11/2022 16:31:34.

Este documento foi armazenado no SUAP em 25/11/2022. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1169323
Código de Autenticação: 2f233ebe3c