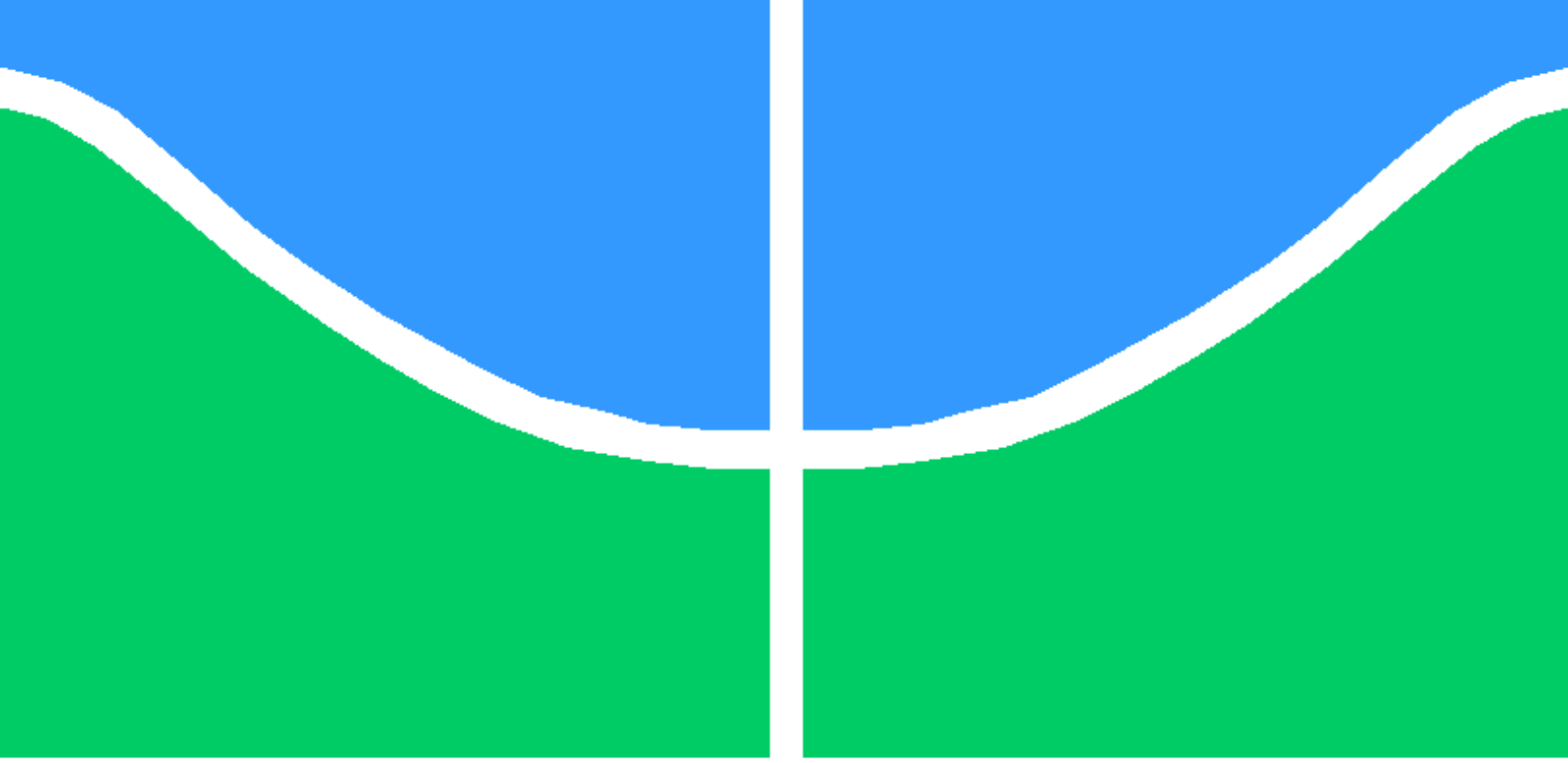


TRABALHO DE GRADUAÇÃO

Aprendizado de máquina aplicado em cluster Kubernetes para dispositivos de baixo custo

Gabriel Lins

Brasília, Maio de 2021

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

Aprendizado de máquina aplicado em cluster Kubernetes para dispositivos de baixo custo

Gabriel Lins

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro de Redes e Comunicação

Banca Examinadora

Prof. Alexandre Nery, ENE/UnB _____
Orientador

Prof. Daniel Chaves Café, ENE/UnB _____
Examinador Interno

Prof. Georges Daniel Amvame Nze, ENE/UnB _____
Examinador Interno

Civilization advances by extending the number of important operations which we can perform without thinking of them.

Alfred North Whitehead

Agradecimentos

Com o final desse ciclo extenso regado de aprendizados, experiências e sentimentos, devo lembrar de todos aqueles que estiveram ao meu lado em todos esses momentos e que me ajudaram de forma única. Antes de mais nada, agradeço pela minha vida, pela saúde e pela capacidade plena de correr atrás dos meus objetivos.

É com imensa gratidão que agradeço todos os meus colegas de curso que dividiram parte desse caminho comigo e, com tamanha amizade, fraternidade e auxílio, fizeram dos momentos de estudos mais dinâmicos. Em especial, agradeço ao João Lucas V. Assis, que desde a infância me ofereceu a melhor amizade que tenho. Agradeço também ao João Paulo Fernandes, Rafael Zerbini e Paulo, que me ajudaram conjuntamente no momento acadêmico em que mais precisei. Valeu 4Teto!

Agradeço à minha família, o alicerce da minha história, que sempre esteve ao meu lado, em especial aos meus pais Elizabeth Peixoto T. Lins e Edson Ferreira Lins, que renunciaram às suas próprias conquistas para permitir que eu concluísse meus estudos. Jamais esquecerei da preocupação e do carinho em, sem se importarem, buscar as melhores escolas públicas para a minha construção pessoal. Agradeço também ao meu grande amor e companheira de vida, Caroline de Melo Ferreira, por todo o apoio fornecido durante a realização deste curso. Desde que descobri o porto seguro que é o seu abraço, jamais me senti desamparado. Estendo os agradecimentos à Luísa Lins e Guilherme Zakarewicz de Aguiar que como irmãos de sangue e de coração, me proporcionaram alegria nos momentos difíceis. À Lara Lins eu agradeço imensamente pois, como irmã mais velha, foi a primeira a explorar o caminho pelo qual eu adentrei mais facilmente. Sua trajetória foi ímpar e sou grato por ter nos ensinado o valor das coisas.

Agradeço a todos cujos caminhos já se encontraram com o meu, a quem me ajudou direta ou indiretamente nesta pesquisa, em especial aos meus professores.

Agradeço à Universidade de Brasília, por me receber jovem e com muitos sonhos, e me devolver para a sociedade um adulto instruído e com sonhos ainda maiores. Ao meu professor orientador, Alexandre S. Nery, sou eternamente grato pelos ensinamentos e apoio oferecido durante esse último ano de minha graduação.

Gabriel Lins

RESUMO

Diante da crescente aplicação do *machine learning* para auxilio em diversas áreas e a quantidade de dados cada vez maior, barreiras relacionadas ao poder computacional foram construídas e hoje, tornam todo o processo bastante oneroso. O grande custo e a sustentabilidade têm sido impulsionadores para que novas alternativas sejam desenvolvidas e estudadas. Nesse contexto, pesquisas e aplicações relacionadas a *Model Parallelism* e *Data Parallelism* ganharam uma notável importância e uma valiosa contribuição para tornar a aplicação do aprendizado de máquina presente em um maior número de ambientes e por um maior número de pessoas.

Neste trabalho, métricas estatísticas que permitem analisar o desempenho e o resultado obtido entre a solução usual e a solução proposta serão apresentadas, tornando possível uma avaliação profunda do *trade off* realizado entre os parâmetros intrinsecamente ligados. Para avaliar o consumo energético por parte dos dispositivos, execuções serão realizadas em dois tipos de dispositivos: em dispositivos *Raspberry Pi 4* modelo B e em máquinas virtuais.

A solução desenvolvida, baseado na construção de um *cluster* Kubernetes utilizando *Raspberry Pi*, busca além de oferecer menor custo no processo de treinamento e inferência do *machine learning*, uma maior eficiência de energia. A distribuição de Kubernetes utilizada nesse estudo foi a oferecida pela Rancher Labs, K3s, voltada para dispositivos de baixo custo. Com o intuito de ser mais leve que as demais distribuições, é possível utilizar o benefício oferecido pelo Kubernetes sem que haja uma sobrecarga nos dispositivos.

Durante o trabalho, *pipelines* de aprendizado de máquinas foram implementados e testados. Utilizando a ferramenta *Kubeflow*, foi construído um ambiente integrado ao Kubernetes voltado para o aprendizado de máquinas. Com essa arquitetura construída, foi possível mensurar elementos como uso de memória, CPU, temperatura e consumo energético e assim, estabelecer métodos para avaliar a utilização de dispositivos *Raspberry Pi* com processamento ARM em etapas de treinamento e inferência de um modelo genérico.

Para compor a pesquisa, foram utilizados ferramentas *open-source* que auxiliam em todo as etapas do aprendizado de máquinas. A solução de *storage* escolhida foi o *LongHorn*, para monitoramento foi utilizado painéis do Grafana facilmente instalados através do Rancher. Para prover o *cluster*, foi utilizado o K3s e o ambiente de aprendizado de máquinas dá-se na ferramenta Kubeflow.

ABSTRACT

In view of the increasing of machine learning in different areas and the increasing amount of data, barriers related to computational power were built and today, the whole process is quite costly. The high cost and sustainability have been boosters to study and develop new alternatives. In this context, research and applications related to "Model Parallelism" and "Data Parallelism" gained notable importance and a valuable contribution to making the application of machine learning present in a greater number of environments and by a greater number of people.

In this work, statistical metrics that allow to analyze the performance and the result obtained between the usual solution and the proposed solution will be presented, making possible an in-depth assessment of the trade off carried out between the intrinsically linked parameters.

The developed solution, based on the construction of a Kubernetes cluster using Raspberry Pi, seeks in addition to offering lower cost in the training process and inference from machine learning, greater energy efficiency.

During the work, machine learning pipelines were implemented and tested. Using the Kubeflow tool, an integrated environment for Kubernetes aimed at machine learning was built. With this built architecture, it was possible to measure elements such as memory usage, CPU, temperature and energy consumption and, thus, establish methods to evaluate the use of Raspberry Pi ARM processing devices in stages of training and inference of a generic model.

To build the research, were used open-source tools that assist in all stages of machine learning. The storage solution chosen was *LongHorn*, for monitoring, Grafana panels were used, easily installed through the Rancher. To provide the cluster, K3s were used and the machine learning environment takes place in the Kubeflow tool.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

Acrônimos

CD	*Continuous Delivery*
CI	*Continuous Integration*
CNCF	*Cloud Native Computing Foundation*
CRISP-DM	*Cross-Industry Standard Process for Data Mining*
DL	*Deep Learning*
ML	*Machine Learning*
NOC	*Network Operations Center*
P&D	Pesquisa e Desenvolvimento
TI	Tecnologia da Informação
UI	*User Interface*
UnB	Universidade de Brasília
VM	*Virtual Machine*

Capítulo 1

Introdução

Durante muitos séculos, a humanidade esteve submetida a diferentes cenários e fizeram parte de constantes mudanças. Diferentes momentos históricos foram registrados e hoje, com a análise dos registros deixados por nossos ancestrais, é possível entender e assimilar como viviam e o que os motivavam. A forma em que os seres pré-históricos caçavam, como descobriam e acompanhavam as mudanças do tempo foram compreendidas através das pinturas rupestres gravadas nas paredes, no interior de uma caverna. Esses registros trouxeram ensinamentos para as próximas gerações. Posteriormente, com a ascensão de uma nova era histórica, a partir de uma simbiose dada através de registros hereditários, foi possível desenvolver a primeira forma de linguagem oral. Desde aqui, é possível já verificar a importância do registro, de armazenar os dados obtidos e torna-los disponíveis para futuras gerações e necessidades.

Com a sociedade moderna, diante todas as políticas sociais e a ascensão do capitalismo, a necessidade de caçar para sobreviver foi substituída pela necessidade do trabalho. Houve uma inversão de valores e uma terceirização de diversas atividades. Iniciou-se uma era onde a busca por pesquisas, desenvolvimento e aprimoramento de técnicas que facilitassem o trabalho foi incentivada. Nessa época o desenvolvimento tecnológico e o ritmo de vida foram acelerados, criando novas necessidades de consumo.

Passados alguns anos, agora com uma percepção diferente, a Idade Contemporânea foi marcada pela invenção do primeiro computador eletrônico digital, o ENIAC - *Eletronic Numerical Integrator And Computer*. Desenvolvido para o laboratório de pesquisa do exército americano, o primeiro computador facilitou a forma em que os cálculos da época eram realizados mesmo possuindo uma capacidade de operação menor que a mais simples calculadora atual. Desde então, a importância e a necessidade dos computadores aumentaram. Dá-se início a uma busca incansável por recursos computacionais.

A constante busca por conhecimento computacional é o retrato mais próximo que, quando comparado com tempos de guerra, recebeu o nome de corrida armamentista. Se no período da Grande Guerra, seguido pelo momento histórico conhecido como Revolução Industrial, a computação já mostrou ser elemento importante, hoje ela se destaca e toma o papel de protagonista. De acordo com a atual situação da sociedade moderna, estudos já comprovam que o poder computacional é

um motor importante na economia e no desenvolvimento de um país.

Registros conhecidos sobre os primeiros computadores deixam claro que os mesmos surgiram para auxiliar e facilitar atividades importantes. Com funções diferentes, todos tinham em comum o processamento de entradas e retorno de resultados como saída. Atendiam ao que era requisitado na época, mas tiveram que ser adequados com o passar dos anos.

A evolução dos computadores, e consequentemente dos seus *hardwares*, por muito tempo acompanhou o desenvolvimento humano. A ciência possibilitou a descoberta de novas matérias primas para compor os chips, novas formas de dispor os processadores e protocolos de comunicação mais inteligentes. A afirmação de Gordon Moore, que ficou conhecida como a Lei de Moore, acerca do aumento da capacidade de processamento dos computadores (DEVANATHAN, 2016) veio sendo fidedigna até onde o processo litográfico dos dispositivos semicondutores permitiu.

Com a atual composição dos componentes dos computadores, os fabricantes estão com dificuldades em manter o desenvolvimento. A demanda ultrapassou as barreiras do poder computacional e o tempo de processamento deve ser cada vez menor. Chegamos a um momento histórico na computação. Espectamos avanços em torno da computação quântica, chips biológicos e nanotubos de carbono. Vale ressaltar que a evolução dos computadores é de extrema importância para toda a humanidade e deve permanecer em constante progresso.

Foi somente com os computadores que todas as ciências puderam evoluir ao mesmo tempo, com a mesma velocidade. Novas aplicações, protocolos, *softwares* e principalmente a rede mundial de internet foram agregadas ao imenso catálogo que a computação hoje fornece.

Na atual conjuntura, a adoção da computação para auxilio no campo profissional foi mundialmente realizada. Hoje é fácil encontrar indústrias dividindo processos da produção com máquinas e robôs. Na área da saúde, por exemplo, o emprego de aprendizado de máquina para verificação de raio-x é realizado possibilitando especificar com precisão possíveis riscos em pacientes, como descreve (ESTEVES; LEA; ALVES, 2019). Na área da segurança pública, o reconhecimento facial é assunto fortemente debatido e busca mapear, com o auxílio técnicas avançadas, crimes e áreas com grande incidência de criminalidade e uso de arma de fogo (GOIN; RUDOLPH; AHERN, 2018).

Com base no panorama de evolução e da adoção da ciência nos últimos anos, há indícios fortes de que decisões e descobertas importantes terão início em bancos de *big data*[1](SICULAR, 2013, p.1). Com o desenvolvimento de métodos cada vez mais genéricos e potentes, pode-se ainda prever que a adoção do aprendizado de máquinas ditará a forma como cientistas e diversos profissionais das áreas mais diversas levarão seu processo de descoberta e avaliação.

O desafio é inato ao desenvolvimento. A eficácia e a consequente implementação do aprendizado de máquina em suas mais diversas aplicações, trouxeram a necessidade de especializações por parte dos operadores que buscaram capacitar-se na ferramenta. Os cursos de aprendizado de máquina e o "*Learning How to Learn*" foram os cursos mais procurados no ano de 2017 na plataforma de ensino *online* COURSERA. Esse resultado mostra um interesse crescente no tema de aprendizado

[1]Big data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation.

de máquina. Não somente no meio acadêmico, o *ML* também está presente dentro dos códigos de grandes empresas como Google, Waze, Netflix, Amazon e Facebook.

O século atual demanda alternativas que entreguem resultados rápidos, práticos e confiáveis. Analisando a utilização do aprendizado de máquina no mundo, é perceptível que o tema encontra-se presente nas mais simples atividades de qualquer indivíduo, seja de forma direta ou não, e oferece de forma rápida e eficaz o que é requisitado. Seu crescimento deve-se muito pelo sucesso e pela vasta possibilidade oferecida.

> *"A grande meta da ciência é abordar o maior número possível de fatos experimentais por dedução lógica a partir do menor número de hipóteses ou axiomas.- Albert Einstein*

1.1 Motivação

O cenário tecnológico atual demanda opções que atendam as reais necessidades dos usuários. Não somente ditado pelo poder computacional, as empresas da área sentem-se na obrigação de resolver as demais dependências relacionadas. A preocupação com eficiência energética também é um fator priorizado em estudos e futuros projetos.

Em novembro de 2020, a Apple passou a adotar processadores próprios fabricados em arquitetura ARM em seus novos notebooks, antes equipados com processadores da Intel (PORTER, 2020). Essa mudança impactou o mercado pois, por se tratar de uma grande empresa, outras empresas também estão pensando em adotar tal mudança. A Microsoft, por exemplo, já realizou duas tentativas de lançar seus próprios *chips* baseado nessa arquitetura (WARREN, 2020).

A NVIDIA, outra grande representante da tecnologia mundial, também mostrou estar interessada no processamento *ARM*. Muitas vezes servindo como elemento de comparação, a NVIDIA desenvolveu um projeto chamado "NVIDIA Grace" onde propõe a utilização de chips de processamento baseado em *ARM* projetado especificamente para *data centers* de IA.(HOLLISTER, 2021). Com o objetivo de tornar-se a maior referência em inteligência artificial, o projeto entrou na lista de prioridades da empresa e tem lançamento marcado para o ano de 2025.

Dessa forma, como os processadores *ARM* possuem uma baixa potência podendo poupar até 85% de energia sem comprometer o desempenho (CHANDRASEKAR et al., 2015), é possível apresentar-se como proposta também para os problemas relacionados a consumo energético.

Motivado pelas pesquisas e debates acerca do conteúdo, a empresa Raspberry Pi Foundation, fundada em 2006, mostrou-se apta para oferecer infraestrutura voltada para a "internet das coisas", IoT. Com o desenvolvimento da tecnologia, criação de novas ferramentas e serviços, modelos mais robustos foram lançados. Indo em direção à motivação inicial da empresa de criar um "dispositivo barato e acessível para todos" (UPTON et al., 2016), em 2021, a empresa encontra soluções viáveis para empregar seus dispositivos em função do aprendizado de máquina e o que toca o assunto de inteligência artificial.

Com a mesma motivação que fomenta o desenvolvimento do assunto por parte dos cientistas e

engenheiros, o presente trabalho se faz necessário como parte de contribuição para o assunto muito debatido.

1.2 Objetivo Geral

Analisar soluções alternativas para contornar com eficiência os problemas ocasionados pela alta demanda de poder computacional durante processamentos de redes neurais, de forma a tornar o processo menos custoso e oneroso. Como parte fundamental do trabalho, tem-se a comparação entre duas implementações diferentes de um sistema de aprendizado de máquinas com base em métricas de consumo de energia, uso do processador, uso de memória e tempo de processamento.

Uma rede neural paralela e distribuída é o elemento de estudo e pesquisa. Conceitos de *clusterização* e a ferramenta Kubernetes são abordados. Dados analíticos que comprovam o estudo serão apresentados e analisados a fim de obter-se um resultado plausível e criterioso a respeito do tema.

1.3 Justificativa

Dado que os valores e as informações descritas explicitam problemas como a quantidade de dados aumentando drasticamente e o grande valor gasto ocasionado pela necessidade de adequação computacional, uma análise mais profunda permitiria entender e propor possíveis formas de amenizar as consequências desses fatores. Dessa forma, o proposto nesse trabalho é, com o auxílio de um *cluster* composto por dispositivos *Raspberries*, realizar as formas de paralelismo conhecidas para averiguar e, posteriormente, concluir com as consequências encontradas, a melhor forma de se realizar a divisão computacional. Durante o processo, é pretendido buscar compreender os detalhes envolvidos dentro de um algoritmo de aprendizado de máquina e como as camadas comunicam entre si, buscando desenvolver uma alternativa eficaz para o problema de escassez de recursos computacionais.

São inúmeros os elementos que podem tornar um algoritmo de aprendizado de máquina oneroso. A quantidade de camadas convolucionais contidas no modelo, ou a quantidade de parâmetros e até como as camadas estão relacionadas as outras devem ser levadas em consideração. A realidade é que modelos atuais contam com vinte e duas camadas, quatro milhões de parâmetros e várias camadas totalmente conectadas, como é o caso da (SZEGEDY et al., 2015). Nesse sentido, estamos tratando de *big data* e o tratamento deve ser realizado com minuciosidade. Fora do contexto da *big data*, empecilhos que acarretam no mal funcionamento de um modelo também são encontrados.

Em diversas áreas do conhecimento a ciência de dados, o aprendizado de máquina e outras técnicas avançadas para tomada de decisões já são utilizadas. É um recurso que está sendo amplamente adotado por conta da sua capacidade de generalização e serve para atender inúmeras necessidades tornando possível ser moldado e adequado a necessidade atual.

Dessa forma, a produção de um estudo buscando alternativas benéficas para o melhor uso de

modelos computacionais, que envolvem o aprendizado de máquina, torna-se significante uma vez que o seu crescimento é positivo e tende a estar presente cada vez mais na rotina de qualquer profissional, independente de sua área acadêmica. Esse trabalho visa contribuir simplificando o entendimento de problemas reais, enfrentados em diversos momentos e em diversas localidades, de profissionais da área da computação.

1.4 Estrutura do Trabalho

1. Capítulo 1: Introdução sobre a importância do aprendizado de máquina para a humanidade e o problema encontrado em relação a recurso computacional. A motivação bem com o objetivo do estudo estão presentes nesse capítulo.

2. Capítulo 2: Contextualização dos assuntos necessários para o entendimento das ferramentas utilizadas no estudo. Nesse capítulo, uma breve apresentação sobre os elementos são apresentados bem como a metodologia utilizada para a realização do estudo.

3. Capítulo 3: Implementação da solução explicitada em forma de gráficos e imagens. Complementando o capítulo anterior, esse capítulo tem por objetivo mostrar os passos necessários para uma futura replicação do estudo.

4. Capítulo 4: Análise dos resultados e suas conclusões. Discussões acerca dos resultados e suas apresentações são explanados afim de buscar conclusões interessantes para o estudo.

5. Capítulo 5: Conclusão

Capítulo 2

Fundamentação Teórica

2.1 Cenário nacional

Desde a década de 1990, pesquisadores do país encontravam-se em uma constante busca por domínio tecnológico. Percebendo que havia uma relação direta entre o domínio tecnológico e a geração de vantagens competitivas, a busca por conhecimento sempre esteve em ascensão e investida com prioridade alta. O conhecimento é uma competência essencial, cuja priorização traz resultados benéficos duradouros que serão permeados durante boa parte da década seguinte.

A competitividade pelo conhecimento geraram benefícios para os países que largaram na frente e tiveram uma política forte voltada para o tema. Por outro lado, os países que demoraram para entender a importância da educação e, consequentemente, da ciência foram prejudicados. No Brasil, estudos realizados na década de 1990 indicavam a superação de dependência tecnológica como um desafio competitivo da indústria nacional.

É a partir da abertura econômica da década de 1990 que as empresas brasileiras sentem com mais ênfase a necessidade de ajustar-se à condição de competitividade internacional, resultando na alteração nos métodos produtivos e gerenciais, bem como em mudanças de priorização para os investimentos. No trabalho realizado para o simpósio de Gramado (R. et al., 2006), o tema é discutido e uma análise a respeito da evolução do investimento em pesquisa e desenvolvimento no Brasil é debatida. Nesse mesmo artigo, os autores concluem que existe uma forte relação direta entre o investimento em pesquisa e desenvolvimento, P&D, e a geração de equipes capacitadas, gerando conhecimento e tecnologia para o país.

Para entender o cenário atual, uma vez que as consequências são alastradas e permanecem pelos próximos anos, é necessário visualizar um pouco antes do ano de 2021. A figura 2.1 trata sobre um indicador tradicional para avaliar a posição de um país em relação a pesquisa e desenvolvimento, P&D, realizada em 2019. Trata-se de uma relação entre o dispêndio nacional em P&D e o produto interno bruto, PIB.

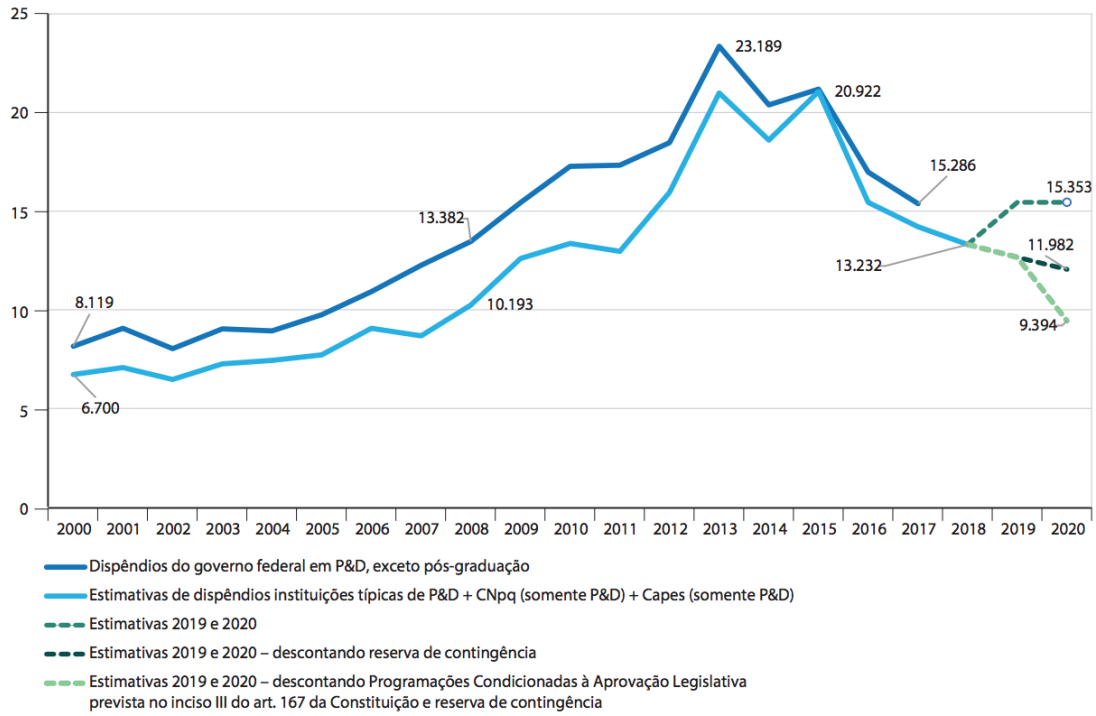


Figura 2.1: Dispêndios e estimativas de investimento do governo federal em P&D em milhões de reais (2000-2020) - *Retirado de MCTIC, 23 out. 2019*

O gráfico presente na figura 2.1 é interessante por abordar o começo do novo milênio e trazer uma projeção para o ano de 2020. Analisando o gráfico, é possível enxergar que entre os anos de 2012 e 2013, o Brasil obteve o seu maior momento do dispêndio chegando ao valor aproximado de R$23.2 milhões. De acordo com (CARRARA; FERREIRA, 2020), nesse momento histórico os investimentos em instituições de ensino superior representavam 58,72%, seguido pelo desenvolvimento tecnológico industrial com 10,60%.

Entretanto, com uma inversão no crescimento depois do ano de 2013, tem-se uma constante queda no gráfico. Essa análise visual que a figura 2.1 possibilita é traduzida no entendimento de que a verba destinada a esse fim veio sendo reduzida, representando um impacto direto na falta de capital para investimento em novos equipamentos e para manutenção dos já adquiridos.

O investimento em pesquisas mostra-se essencial e de extrema importância quando analisamos a recente pandemia que acometeu a população mundial. A união de cientistas e pesquisadores de diversas nacionalidades trabalhando com um objetivo comum trouxe agilidade e esperança. Com isso, países como China e França prometeram aumentar o orçamento destinado à pesquisa nos próximos anos (NORMILLE, 2020). Por sua vez, o Brasil possui uma visão diferente.

Diante da crise educacional que aflige o Brasil, há uma contenção de gastos que impossibilita o avanço da pesquisa no país. Pesquisadores e cientistas são submetidos a encontrarem formas de continuarem seu papel fundamental e importante com o auxílio de materiais ultrapassados ou até mesmo sucateados. Apesar do avanço tecnológico ter alcançado resultados incríveis, o custo é alto e a realidade da grande maioria dos centros tecnológicos nacionais é primitiva. Com a

liberação apenas do básico e do essencial, quanto mais dispendioso for o investimento, maior a chance do projeto não ser aprovado e a verba ser negada. Dessa forma, substitutos mais viáveis economicamente e que possibilitem alcançar a eficacia desejada tendem a ser procurados por ambas as partes.

2.1.1 Raspberry Pi

O conceito do dispositivo *Raspberry Pi* começou a ser arquitetado em 2006, inspirado pelos microcontroladores Atmel ATmega644. Com o intuito de desenvolver um computador acessível e que possibilitasse o desenvolvimento criativo das crianças, foram desenvolvidos os primeiros modelos da marca.

O último lançamento de *hardware* foi realizado no ano de 2019, e recebeu o nome de *Raspberry Pi* 4 Model B. Com a oferta de três opções diferentes, o modelo foi atualizado com suporte a saída 4K e 8GB de memória RAM mantendo o mesmo modelo de CPU, um quad-core da Broadcom com velocidade de até 1,5GHz em cada um dos seus núcleos Cortex-A72. Mais informações acerca do modelo são encontradas no site[1] da *Raspberry*.

No Brasil, por conta da taxa de importação e da moeda, os dispositivos chegam com um preço mais alto. Ainda assim, é uma boa oportunidade para a comunidade rural e para as escolas públicas, que contam com mais de 5 milhões de pessoas sem acesso ao computador em 2020 (CASTIONE et al., 2021). Adotando o dispositivo como uma alternativa para suprir os *déficits* relacionado ao acesso a um computador, é esperado que com um investimento bem menor, todos os estudantes brasileiros possam ter contato com esse item essencial e primordial para uma boa educação. Possibilitar o acesso a um microcomputador de baixo custo significa ter novos benefícios educacionais fazendo com que as pessoas estabeleçam uma nova forma de relação com o conhecimento (PINTO, 2010).

Como essa realidade assola não somente o Brasil, empresas do Reino Unido mostraram interesse em patrocinar escolas públicas e privadas oferecendo dispositivos *Raspberry* para cada aluno. Países do oriente médio também estão ofertando os microcomputadores para os alunos de suas escolas a fim de melhorar suas perspectivas de emprego e mitigar o grave problema incondicional do predacionismo da concentração de renda de grandes grupos econômicos.

Inúmeros estudos acerca da aplicação do dispositivo são realizados anualmente. Apesar do futuro estar tendenciosamente voltado para o uso de GPUs para realizarem operações de aprendizado de máquina, no mundo acadêmico análises sobre o desempenho dos *Raspberry* são considerados uma vez que há vantagens energéticas e menor custo relacionado. (BESIMI et al., 2020)

No presente estudo, foram utilizados 4 *Raspberry Pi* 4 modelo B. Conectados através do cabo de rede *ethernet* Cat5E, é alcançada uma largura de banda máxima de 10Mbps. Como o cartão *microSD* presente em cada dispositivo é de apenas 64GB, foi necessário adicionar um disco removível sendo montado como *storage* físico. Esse armazenamento externo oferecido pelo HD externo é ponto centralizado em operações de leitura e escrita em disco, sendo também utilizado

[1] https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/

pelo LongHorn. A figura 2.2 mostra parte da estrutura real utilizada no estudo.

Figura 2.2: Arquitetura real dos dispositivos *Raspberry Pi*

A outra parte da arquitetura é composta por máquinas virtuais que auxiliam na orquestração e administração de todo o *cluster*.

2.2 Machine Learning

Muito tem-se ouvido falar a respeito do aprendizado de máquina e suas mais diversas aplicações. Como (EL NAQA; MURPHY, 2015) trata em seu livro, o aprendizado de máquina é um ramo em constante evolução de algoritmos computacionais projetados para emular a inteligência humana aprendendo com o ambiente circundante. É uma sub-área da Inteligência Artificial (IA) e está fortemente relacionado a estatísticas computacionais que envolvem métodos, teorias e aplicações matemáticas, para aprender. O aprendizado pode ser realizado de forma supervisionada como também de forma não supervisionada (XIN et al., 2018).

Tendo como inspiração o funcionamento do corpo humano, abrange técnicas de cognição a fim de treinar e reconhecer padrões com o objetivo de aplicá-los em previsões futuras (AZUAJE; WITTEN; E, 2006). Arthur Samuel, o pioneiro do aprendizado de máquina no mundo, propõe a seguinte definição: “ é um campo de estudo que dá aos computadores a capacidade de aprender sem serem explicitamente programados”.

Técnicas baseadas no aprendizado de máquina têm sido aplicada nas mais vastas áreas variando entre padrões de reconhecimento, aplicações médicas e engenharia aeroespacial. Dentre os campos de aplicação, a técnica é recomendada para aquelas que envolvem reconhecimento de padrões e interpretações técnicas. Com o seu alto desempenho e sua alta eficácia, trabalhando em conjunto com humanos, os erros e intervenções humanas tendem a ser eliminados. O emprego da técnica gera inúmeras contribuições para a ciência.

2.2.1 *Deep learning*

A diferenciação entre aprendizado de máquina e *deep learning* é discutida dentro da esfera da Inteligência Artifical. O *deep learning* é um novo campo dentro do aprendizado de máquina que simula o cérebro humano para interpretar dados (LECUN; BENGIO; HINTON, 2015). Como principais diferenças entre ambos encontra-se a quantidade de dados que devem ser fornecidos para obter-se um nível aceitável de resposta. Uma vez que o aprendizado de máquina adapta-se bem com poucos dados de entrada, o *deep learning* apresentaria um resultado ineficaz se a mesma quantidade de dados fossem mantidas. Dessa forma, o desempenho é diretamente afetado de acordo com a quantidade de dados fornecidos para cada um. Outro fator que os diferenciam, principalmente pelo primeiro motivo apresentado, está a dependência de GPU e a diferença no tempo de execução. Com uma grande quantidade de dados sendo apresentada para o modelo, múltiplas operações e parâmetros devem ser testados, resultando na dependência de componentes específicos e aumentando o tempo de análise.

Considerando que a produção e o armazenamento de dados vêm crescendo consideravelmente ao longo do tempo e analisando o avanço do desenvolvimento de algoritmos muito mais complexos e rápidos, é esperado que o conceito do *deep learning* (DL) ganhe o lugar do aprendizado de máquina (ML) e se adéque mais ao cenário dos próximos anos. Essa nova fase depende diretamente das inovações tecnológicas. Para que seja possível estar pronto para esse momento, todo a arquitetura e o ambiente relacionado deve passar por reformulações e moldar-se de acordo com os pré-requisitos da nova tecnologia, trazendo elementos cada vez melhores. É sabido que atualizações de *hardwares* é oneroso e envolvem procedimentos burocráticos, que podem interromper uma esteira inteira de pesquisa e estudos por tempo indeterminado. A depender do projeto, anos de trabalho são desperdiçados caso seja necessário haver uma interrupção inesperada e sem previsão para retomada.

Na esfera do aprendizado de máquina, grandes modelos demandando um longo tempo para treinamento estão motivando o desenvolvimento de algoritmos de treinamento distribuído. Dispostos a contornar o problema que envolve grande uso de recursos computacionais ligados ao processo de aprendizado de máquina (CPU, memória RAM e GPU), cientistas por muito tempo buscam por alternativas que resolvam o problema computacional de maneira prática e inteligente, sem demandar muito investimento em super computadores. Em seu trabalho, (DEAN et al., 2012) e (ZHANG; CHOROMANSKA; LECUN, 2015) debatem essa questão e propõem soluções.

Pensando em trazer uma alternativa viável e que atenda as necessidades dos usuários que não possuem condições para investir em super máquinas, mas fazem do aprendizado de máquina ferramenta primordial e essencial de suas atividades, o seguinte estudo é de extrema valia. Usando a técnica do paralelismo, a carga de trabalho, antes sendo realizada em apenas uma única máquina, pode ser dividida em inúmeros *hosts* independentes que realizam etapas do trabalho.

O uso de GPU foi um avanço importante que se desenvolveu no passar dos últimos anos tornando possível o treino de uma grande rede neural - *deep networks*. (DAHL et al., 2012) (CIREŞAN et al., 2010)

2.3 Paralelismo

Buscando contornar o balanceamento entre custo e recurso computacional, estudos acerca de métodos alternativos para aplicar o aprendizado de máquina têm ganhado importância. O número de parâmetros nos modelos modernos de *deep learning* está se tornando cada vez maior, e o tamanho do conjunto de dados também está aumentando drasticamente. Para treinar um modelo de aprendizado profundo moderno e sofisticado com um grande conjunto de dados, é necessário usar o treinamento distribuído, caso contrário, o tempo de processamento tornará todo o projeto inviável. Nesse caso, sempre é possível ver o uso do *data parallelism* e *model parallelism* no treinamento de aprendizado profundo distribuído.

Por conta do desenvolvimento de métodos que facilitam o escalonamento e a capacidade das redes neurais, o *ML*, assim como o *DL*, obtiveram um bom progresso durante a última década. Classificações de imagens e processamento de linguagem natural, como as apresentadas por (MCCANN et al., 2018) e (PETERS et al., 2018) em seus trabalhos, trazem os benefícios que a ferramenta oferece para a população. Todavia, para ter a possibilidade de ser amplamente utilizado, deve ser abordado de abrangente de forma a se encaixar com todo o cenário global já que a realidade da maioria dos centros de pesquisa do país é diferente da realidade dos países de primeiro mundo. O tema de aprendizado de máquina paralelizado e distribuído mostra ser valoroso e necessário para os demais centros de pesquisas mundiais que possuem baixo investimento por parte do governo.

Para discussão, será proposto nesse presente documento duas abordagens diferentes para o processamento paralelo. Apesar de ambos possuírem o objetivo em comum, a forma em que são realizados bem como seus resultados são diferentes. (DAHL et al., 2012) (CIREŞAN et al., 2010). A busca é encontrar formas de aplicar ambas as técnicas para que trabalhem juntas e tornem o processo otimizado de forma inteligente e distribuída.

2.3.1 Data paralelismo

O processamento paralelo de dados é uma técnica que divide um único *job* em diferentes *tasks*, executando-os simultaneamente, de forma paralela. Dessa maneira, um único modelo é replicado em vários dispositivos e cada um processa frações dos dados mesclando os resultados encontrados.

Nesse modo, um único modelo é utilizado para fazer a *forward propagation*. Com o modelo presente em cada um dos nós, pequenos lotes de dados são enviados para cada um deles e o gradiente é calculado. Ao final do processo, cada gradiente calculado é enviado de volta para o nó principal. Com todos os gradientes calculados, a média ponderada é calculada e então utilizada como parâmetro para atualizar o modelo. Por possuírem velocidades de processamento diferentes, o procedimento é feito de forma assíncrona.

O procedimento se descreve como um paradigma de aprendizagem onde várias réplicas de um único modelo são usados para otimizar um único objetivo. Na expressão demonstrada em 2.1, a técnica é descrita mostrando que a mesma pode ser sustentada matematicamente.

$$
\begin{aligned}
\frac{\partial \text{ Loss}}{\partial w} &= \frac{\partial \left[\frac{1}{n}\sum_{i=1}^{n} f\left(x_i, y_i\right)\right]}{\partial w} \\
&= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial f\left(x_i, y_i\right)}{\partial w} \\
&= \frac{m_1}{n}\frac{\partial\left[\frac{1}{m_1}\sum_{i=1}^{m_1} f\left(x_i, y_i\right)\right]}{\partial w} + \frac{m_2}{n}\frac{\partial\left[\frac{1}{m_2}\sum_{i=m_1+1}^{m_1+m_2} f\left(x_i, y_i\right)\right]}{\partial w} + \cdots + \frac{m_k}{n}\frac{\partial\left[\frac{1}{m_k}\sum_{i=m_k+1}^{m_k+m_k} f\left(x_i, y_i\right)\right]}{\partial w} \\
&= \frac{m_1}{n}\frac{\partial l_1}{\partial w} + \frac{m_2}{n}\frac{\partial l_2}{\partial w} + \cdots + \frac{m_k}{n}\frac{\partial l_k}{\partial w},
\end{aligned}
\tag{2.1}
$$

onde:

w é o parâmetro do modelo, $\frac{\partial \text{Loss}}{\partial w}$ é o gradiente real do *batch* de tamanho n, $\frac{\partial l_k}{\partial w}$ é o gradiente de cada *batch* em cada nó k, x_i e y_i são, respectivamente, *features* e *labels* do dos dados i, $f(x_i, y_i)$ é o *loss* de cada ponto calculada a partir da propagação direta, n é o total de *data points* no *dataset*, k é o total de nós, m_k é o número de *data points* atribuído para cada nó, $m_1 + m_2 + \cdots + m_k = n$.

Quando $m_1 = m_2 = ... = m_k = \frac{n}{k}$, a expressão acima pode ser reduzido na formula 2.2, que é apresentada abaixo.

$$
\frac{\partial \,\text{Loss}}{\partial w} = \frac{1}{k}\left[\frac{\partial l_1}{\partial w} + \frac{\partial l_2}{\partial w} + \cdots + \frac{\partial l_k}{\partial w}\right] \tag{2.2}
$$

Com isso, chega-se ao fato comprovado de que a soma de cada gradiente calculado por cada *host* dividido pelo número de *hosts* resulta-se no gradiente total de todo o *dataset* comprovando matematicamente que a paralelização não traz consequências destrutivas ou até mesmo incorretas para todo o modelo.

2.3.2 Modelo paralelizado

Nesse tipo de processamento, diferentes partes de um único modelo são executados em diferentes dispositivos, processando um único lote de dados conjuntamente. Essa técnica é apropriado para distribuir o modelo entre diferentes GPUs dentro de um mesmo dispositivo, ou mesmo espalhado em outros dispositivos.

A fim de exemplificar a técnica, a figura 2.3 mostra de forma visual um procedimento realizado onde um modelo composto por quatro camadas foram divididos entre quatro máquinas diferentes.

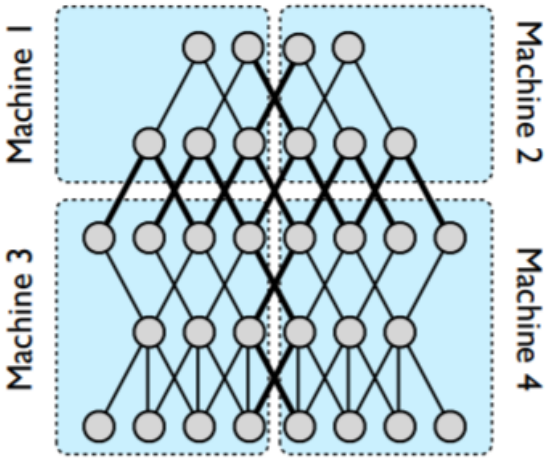


Figura 2.3: Representação de um modelo paralelizado e suas camadas distribuídas - *Retirado de (DEAN et al., 2012)*

Utilizando CPUs presente em cada *host*, é possível reduzir drasticamente o tempo de processamento utilizando essa técnica descrita (DEAN et al., 2012). A figura 2.3 mostra uma relação entre quatro máquinas conectadas dividindo um mesmo modelo onde cada máquina torna-se responsável por algumas camadas.

Para descrever um exemplo, imagine que na situação da figura 2.3 onde tem-se 4 CPUs, deseja-se treinar um modelo simples. Em uma divisão, as primeiras 2 camadas poderiam ser atribuídas à CPU 1, as 5 segundas camadas à CPU 2 e assim por diante, até a as últimas camadas serem atribuídas à última CPU 4. Durante o treinamento, em cada iteração, a propagação direta deve ser feita. Em forma de cadeia, cada CPU espera pelo retorno da anterior para realizar a sua operação. Assim que a propagação direta for concluída, os gradientes para as últimas camadas serão calculados e os parâmetros atualizados. Em seguida, os gradientes retrocedem para as camadas anteriores, voltando de CPU em CPU. No exemplo descrito, cada CPU pode ser visto como um *host* pertencente a um espaço na linha de produção onde cada um espera pelos produtos de seu compartimento anterior e envia seus próprios produtos para o próximo compartimento.

Retratado como uma forma básica de alocar recurso computacional dividindo as camadas para mais de um dispositivo, a técnica se difere ainda do *data parallelism* por não ser embasada em um modelo matemático (MAO, 2019). De qualquer forma, a diminuição de consumo computacional utilizado em cada máquina e a diminuição do tempo total do treinamento são pontos evidentes e fáceis de serem notados dentro dessa técnica.

2.4 Introdução ao Kubernetes

A constante evolução e adequação da computação, da tecnologia da informação e tudo o que tange a infraestrutura da TI, é notória. De forma comparativa, analisando a composição do fluxo de trabalho de décadas passadas e a do atual momento (2021), a mudança é perceptível.

Desde o ano de 2013, marcado pelo lançamento do Docker, houve um período de transformação digital. Os serviços, antes totalmente voltados para *on-premise*, passaram a ser desenvolvidos de forma isoladas e possuem a sua infraestrutura em provedoras de *cloud*. Essa mudança mostrou ser eficaz por oferecer uma maior portabilidade dos serviços assim como maior escalabilidade, controle e menor indisponibilidade relacionada a dependências. De acordo com a página (DOCKER, 2020), o conceito do Docker já era utilizado por mais de 5 milhões de usuários no ano de 2020. De acordo com uma pesquisa realizada pela Cloud Native Computing Foundation (CNCF, 2020b), 92% das empresas estão utilizando conteineres em produção em 2020. Isso representa um aumento de 23% em relação a 2016. Tamanha adoção trouxe consigo a importância de uma plataforma que fosse capaz de gerenciar todos esses *conteineres*[2] e tornar o trabalho dos desenvolvedores e dos administradores mais fácil nessa cultura DevOps.

Com o propósito de realizar a centralização, gerenciamento e monitoramento dos serviços em *conteineres* Docker, surgiu o Kubernetes. Sendo uma solução de código aberto, a adoção da ferramenta proporciona tarefas automatizadas que além de auxiliar na criação de novos serviços *conteiner*, também oferece uma orquestração. O *cluster* Kubernetes é um conjunto de máquinas dedicadas a trabalharem em conjunto para executar as aplicações em *conteineres*.

De acordo com a mesma pesquisa realizada pela CNCF (CNCF, 2020b), 91% dos entrevistados utilizam Kubernetes, sendo 83% em ambiente de produção. Dessa maneira, é notável a preferência e crescente adoção desses conceitos para as atuais arquiteturas e infraestrutura de TI.

Com o avanço dos provedores desse tipo de orquestração, foi de extrema importância a criação de um projeto que viabilizasse a integração das atuais soluções existentes e tornasse a indústria alinhada com a evolução da ferramenta. Esse projeto recebeu o nome de *Cloud Native Computing Foundation* - CNCF, e hoje contribui de forma ativa para tornar a computação nativa em nuvem universal e sustentável com as mais variadas soluções *open-source*.

Os clusters Kubernetes podem ser construídos através de inúmeras soluções. É possível utilizar solução proprietárias da Google, Amazon e Microsoft mas também são encontradas soluções *open-source* prontas para atender a demanda, como é o caso da solução ofertada pela Rancher, a *Rancher Kubernetes Engine*, RKE[3] e K3s, uma solução também da Rancher voltada para arquitetura ARM e dispositivos IoT, para uma abordagem de *edge devices*. Para a realização desse projeto, foi escolhido utilizar a distribuição Kubernetes provida pelo K3s. Na seção 3.2, detalhes sobre a implementação e especificações do *cluster* serão abordados.

Trabalhando de uma forma independente e isolada, um *cluster* recebe o adjetivo de "*distroless*" por possuírem a característica de não ser dependente a nenhuma distribuição de sistema operacional. De acordo com a figura 2.4, é possível identificar os principais componentes que compõem um *cluster* Kubernetes. Uma breve descrição acerca de cada elemento faz-se necessária e será feita logo abaixo na seção 2.4.1.

[2]Termo utilizado para tratar os serviços implementados pelo Docker
[3]Disponível em https://rancher.com/docs/rke/latest/en/

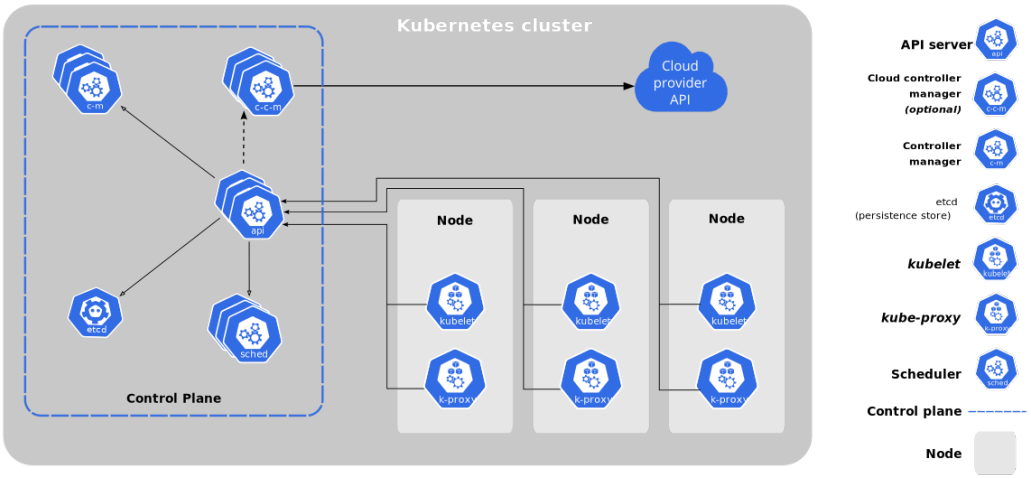


Figura 2.4: Representação dos componentes de um *cluster* Kubernetes - *Retirado de (SIMOES, 2021)*

2.4.1 Elementos do cluster

- Control plane: O componente recebe esse nome por ser o responsável por tomar decisões globais sobre todo o *cluster*. Os *hosts* definidos com a legenda "*control plane*", tornam-se os responsáveis por, por exemplo, iniciar um novo *pod*[4], migrar serviços para outras máquinas e deletar um serviço. Cada *control plane* recebe os seguintes componentes:
  - kube-apiserver: Responsável por executar as operações REST e fornecer o *front-end* para que os componentes do *cluster* interajam.
  - etcd: Componente de armazenamento de valores-chave de alta disponibilidade usado como repositório do Kubernetes para todos os dados do cluster.
  - kube-scheduler: Sua função é definir cargas de trabalho para nodes específicos levando em consideração recursos individuais, requisitos de qualidade de serviço e localidade dos dados.
  - kube-controller-manager: Controlador responsável por definir os estados das réplicas, controlar os *endpoints*, *namespaces* e *service accounts*.
- Node: Dentro de cada nó com a legenda "*worker*", os componentes abaixo são executados em cada nó, sendo responsáveis por manter os *pods* em execução.
  - kubelet: Presente em todos os nós, é responsável por ler o *manifest* de cada *conteiner* e certificar que cada *conteiner* está ativo e saudável.
  - kube-proxy: Componente responsável por manter regras de rede nos *nodes*, permitindo a comunicação com seus *pods* a partir de sessões de rede dentro ou fora do *cluster*.
  - Conteiner runtime: Responsável pela execução dos *conteineres*.

[4]Grupo de um ou mais *conteineres* implementado em um único nó

Dessa maneira, o nó intitulado como *master*, é o ponto central de todo o *cluster* Kubernetes onde os administradores podem interagir, realizando as operações de forma centralizada. O nó *master* guarda o estado e as configurações de de todo o *cluster* em seu componente *etcd*. Por ser um espaço acessível por todos os nós, os *workers* conseguem ler o arquivo de configuração e aprender como cada *conteiner* deve ser criado. Toda a comunicação entre os nós do *cluster* é realizada através do componente "*kube-apiserver*", que serve como um ponto de acesso central.

2.4.2 Kubeflow

A indústria desenvolveu técnicas e ferramentas para compor uma esteira de desenvolvimento e prover uma estrutura baseada em metodologia ágil. Com um *pipeline* automatizado onde testes, desenvolvimentos e homologação são realizados constantemente, tem-se o que recebe o nome de *continuous delivery* e *continuous integration*. Com a entrega contínua de *softwares*, houve um aumento considerável nos resultados e nos lucros consequentemente. Esse método tem sido cada vez mais comum e, por conta de seus benefícios, tende a ser aplicado e diversificado em outros ambientes.

Nesse contexto, com a utilização do CI/CD em esteiras de desenvolvimento, e com o aumento da utilização do aprendizado de máquina, é esperado que ambos sejam interligados. Muitas vezes, um fluxo de treinamento e sua entrega a um ambiente de produção contém muito trabalho manual. Dependendo do modelo que está sendo treinado, os números de intervenções manuais aumentam. Esse fator interfere diretamente no tempo de trabalho do operador que, se envolverá nas minuciosidades do processo resultando em uma sobrecarga desnecessária.

Muito se tem desenvolvido nessa área de aplicação a fim de integrar ferramentas para compor o CI/CD de um treinamento de máquinas. Apesar de existirem para entrega de *softwares*, os mesmos não podem ser aplicados para os modelos do aprendizado de máquina.

Com o objetivo de fornecer um *pipeline* de desenvolvimento envolvendo um modelo de ML surgiu o Kubeflow. Desenvolvido pela Google e outras contribuidoras, o propósito da ferramenta é facilitar a entrega de *workloads* de ML, tanto *on-premise* quanto em *cloud*. Uma das suas características é ter sido desenvolvida para compor o catálogo de serviços do Kubernetes, tornado-se uma ferramenta que integra diversos *frameworks* de ML usuais, tais como TensorFlow, Keras e PyTorch, e os torna acessíveis dentro de um *cluster* Kubernetes.

Durante o desenvolvimento de um modelo de aprendizado de máquina, são necessárias seguir algumas rotinas e etapas para o bom andamento do projeto. Sem que haja uma pesquisa profunda sobre o tema e sem uma definição prévia das ferramentas que serão utilizadas, por exemplo, o processo torna-se dificultoso e uma inferência com resultados aceitáveis torna-se inacessível. Etapas como preparação dos dados, treinamento do modelo e teste de integração são itens essenciais nesse tipo de desenvolvimento.

A figura 2.5 tem por objetivo mostrar as etapas usualmente respeitadas na construção de um modelo e relacioná-las aos componentes ofertados no Kubeflow. Cada etapa possui seu nome e, em seguida, o tipo de ferramenta usualmente utilizada para realiza-la. Essas figuras evidenciadas

representam ícones de *softwares*, linguagens e *frameworks* que possuem compatibilidade com o Kubeflow. A análise da figura permite compreender que a ferramenta Kubeflow está presente desde a fase de treino do modelo.

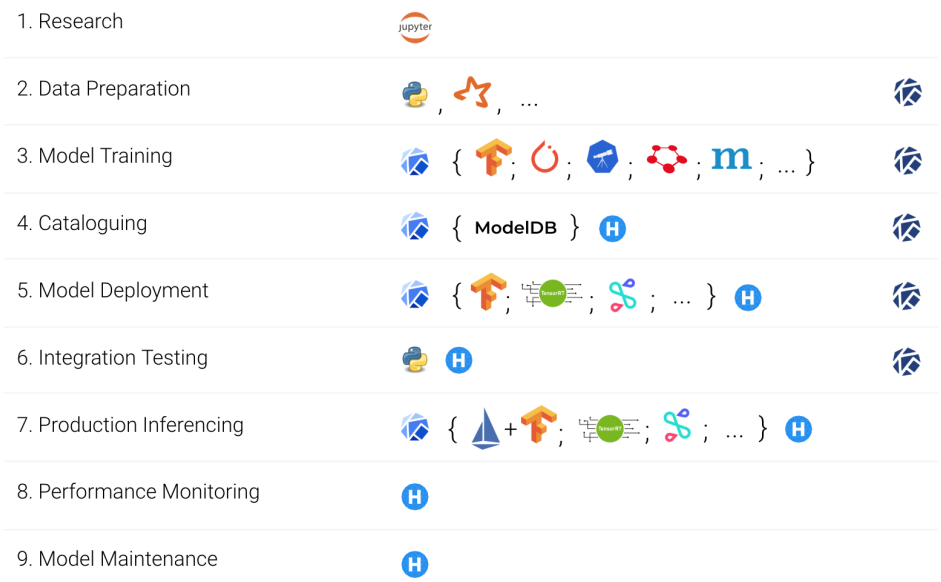


Figura 2.5: Etapas do aprendizado de máquina implementadas dentro do Kubeflow - *Retirado de (GARIFULLIN, 2020)*

Dentro da ciência dos dados, os princípios do CRISP-DM (*Cross Industry Standard Process for Data Mining*) é amplamente utilizado. Diante da necessidade dos profissionais, a metodologia foi criada e vêm guiando o processo de mineração de dados desde então. Com suas etapas bem definidas, como mostra a figura 2.6, vantagens como facilidade para resolução de problemas e implementação de novos modelos são adquiridos.

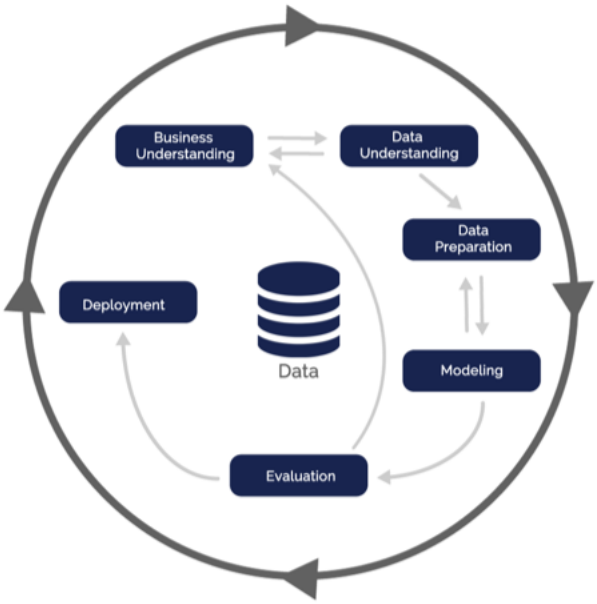


Figura 2.6: Etapas da metodologia CRISP-DM para mineração de dados

Quando analisadas e comparadas as figuras 2.5 e 2.6, é possível encontrar formas de implementar todas as etapas da metodologia CRISP-DM dentro do Kubeflow, sem a necessidade de ferramentas externas para tal. Essa percepção mostra uma grande abrangência por parte da ferramenta escolhida.

Uma vez entendido a respeito das etapas do desenvolvimento de um aprendizado de máquina, o valor da ferramenta Kubeflow é evidenciado. Para o intuito desse trabalho, onde é desejado paralelizar o treinamento do modelo em diversos dispositivos *Raspberry Pi*, a aplicação da ferramenta será de grande valia. O foco principal do ecosistema Kubeflow é treinar modelos de aprendizado de máquina construídos em diferentes *frameworks* de maneira distribuída fazendo uso do *cluster* Kubernetes.

A partir de um modelo genérico composto de algumas camadas e pesos aleatórios, o escopo do trabalho permeará entre a criação de um *pipeline* de aprendizado de máquina e o processo de divisão de todas as etapas. Uma análise acerca desse estudo está presente no capítulo 4.

2.4.3 Rancher

A empresa Rancher Labs, criada no primeiro semestre de 2014, tem como função auxiliar no gerenciamento de *cluster* Kubernetes em escala. Com o objetivo de atender as demandas que o novo estilo de desenvolvimento baseado em *conteineres* trouxeram, já em 2021, a empresa conta com diversas ferramentas que são ofertadas gratuitamente como opção para compor o *cluster* do usuário.

Motivada pela ansiedade e pelos novos horizontes que a introdução do Docker trouxe, a empresa americana incorporou a necessidade de auxiliar em questões de segurança, orquestração e

gerenciamento dos *conteineres*. Somente em 2016, adotando o Kubernetes como orquestrador, a ferramenta passou a ser adotada por milhares de times por todo o mundo.

Em 2021, como parte da história da empresa, a Rancher foi comprada pela empresa alemã SUSE. Diante dessa movimentação, para alinhar os interesses de ambas as empresas, novas ferramentas *open-source* estão sendo desenvolvidas e o tema de aprendizado de máquina é uma das questões que devem ser apresentadas em breve para a toda a comunidade.

Como ferramentas ofertadas pela Rancher Labs têm-se o RKE e K3s - ferramentas de criação de *cluster* Kubernetes -, a ferramenta para *storage* dedicado para o funcionamento de *conteineres* chamada LongHorn e, como item principal, o próprio Rancher, que torna-se uma plataforma unificada para gerenciar todos os *clusters* Kubernetes presentes no ambiente.

Dentro de um *cluster* Kubernetes, o usuário familiarizado com o terminal pode usar a ferramenta *kubectl* para ver informações sobre todo o *cluster*, bem como o *status* dos *nodes*, *status* de cada *pod* e saber, por exemplo, em qual nó cada um está hospedado. A instalação dessa ferramenta pode ser feita através do próprio site[5] ou, no caso do K3s, esse procedimento não se faz necessário uma vez que o mesmo já é automaticamente instalado com a criação do *cluster*. Pensado em atender os mais diversos tipos de usuários, a ferramenta Rancher facilita o processo de administração do *cluster* provendo uma página gráfica capaz de realizar os mesmos procedimentos realizados pelo *kubectl* de forma transparente, retornando para o usuário final um resultado mais fácil de ser visualizado. Essa característica será utilizada nesse trabalho uma vez que vai de encontro com o tema principal que é facilitar o uso do Kubernetes para provisionar um *pipeline* distribuído de aprendizado de máquina.

Outro ponto oferecido pelo Rancher que será explorado nesse trabalho é o uso de um *marketplace* interno para instalação facilitada de serviços. Em um projeto real, rodando em produção ou não, diversos elementos estão presentes para compor uma única solução. Um desenvolvedor *web*, por exemplo, desejando entregar uma aplicação *web* faz uso de um balanceador de cargas, um *database* e ferramentas de monitoramento, para alertar sobre o *status* do projeto. Cada uma dessas ferramentas precisam estar conectadas e em pleno funcionamento. Com o Rancher, o usuário pode optar por rapidamente instalar a aplicação pelo *marketplace*, instalando por exemplo um *Grafana* e *Prometheus* para o monitoramento da aplicação, o *LongHorn* para tratar o armazenamento dos dados resolvendo as dependências do projeto. Nesse trabalho, para servir como ponto central de armazenamento, o serviço *LongHorn* será utilizado assim como o monitoramento provido pelo *Prometheus* e *Grafana*.

Por fim, ainda fazendo uso de alguns dos recursos oferecidos pela solução *open-source*, será utilizado a visualização de *logs* de cada *conteineres* criado em cada etapa do *pipeline* gerado pelo *KubeFlow*. Dessa forma, será possível analisar os resultados e obter uma análise acerca do projeto.

Por fazerem parte da solução estudada, as seções 2.5 e 2.6 têm por objetivo apresentar uma introdução a respeito da solução LongHorn e do monitoramento.

[5]Disponível em https://kubernetes.io/docs/tasks/tools/

2.4.3.1 Principais componentes do Rancher

- *Administração pela UI*: Todo o *cluster* pode ser administrado através da interface intuitiva oferecida pelo Rancher. Com o objetivo de facilitar o manuseio e tornar possível a integração de usuários inexperientes, todas os comandos antes realizados em um terminal, podem ser facilmente acessados através de uma *user interface*.

- *Ambientes segmentados*: Para organizar os diferentes ambientes que podem ser criado dentro do Kubernetes, o Rancher *management* oferece uma abstração diferente do Kubernetes, que só faz uso de separação por *namespaces*. Com a possibilidade de criar projetos, o Rancher faz com que cada *node* presente no *cluser* possa ser alocado para um projeto específico, aumentando a segurança e a disponibilidade de todo o *cluster*. Dessa forma, o usuário não corre o risco de ter uma máquina dedicada ao ambiente de produção desperdiçando recursos hospedando um projeto de teste.

- *Autenticação*: O controle de acesso pode ser realizado através de ferramentas conhecidas e amplamente utilizadas pelos administradores como, por exemplo, o *Active Directory*, da Microsoft ou o GitHub.

- *Integração monitoramento*: Para monitorar todos os seus serviços é possível adicionar as *dashboards* e coletar as métricas através dessas duas ferramentas *open source*. Ao instalar o serviço, o usuário automaticamente conta com gráficos de monitoramente pré-configurados que conseguem mensurar e monitorar boa parte dos serviços presentes no *cluster*.

2.5 LongHorn

Para entender a ferramenta em questão, inicialmente deve-se entender os conceitos de uma aplicação *stateful* e *stateless* bem como as suas diferenças.

Em uma perspectiva ampla, as aplicações podem, ou não, precisar de um banco de dados para salvar dados. Algumas aplicações têm a natureza *stateless* por não dependerem de um *storage*, sendo chamadas de aplicações efêmeras. Dessa forma, quando forem reiniciadas, terão os seus dados perdidos. Essa característica pode não ser um problema caso a aplicação seja construída para agir dessa forma. Aplicações que processam *requests* baseados apenas em informações fornecidas, sem depender de *request* anteriores é um exemplo de aplicação desenvolvida para ser *stateless*.

Por outro lado, as chamadas aplicações *stateful* contam com um banco de dados para armazenar todos os dados. Assim, elas conseguem manter o eu estado e não perder nenhum dado armazenado em um necessário *restart* ou até mesmo após serem deletados. Esse tipo de aplicação é essencial para, por exemplo, um serviço que lida com transações bancárias, compras *on-line*.

Dessa forma, uma vez entendido que aplicações podem e precisam ser sustentadas por um *storage*, a funcionalidade do *LonhHorn* pode ser apresentada.

Com o intuito de suprir as necessidades das aplicações *stateful* dentro do *Kubernetes*, o *LongHorn* surgiu. Desenvolvido para ser uma ferramente de fácil *deploy* e gerenciamento, sua adoção torna

possível e facilita o processo de distribuição de blocos de *storage* para cada *pod* do *cluster*, sem a necessidade de contratar um serviço externo, *cloud providers*, para realizar a função. De forma simplificada e rápida, um usuário final pode contar com particionamento do *block storage* em volumes menores para serem usados nos *pods*, replicar cada bloco do *storage* em diversas quantidades para aumentar a disponibilidade dos dados e até mesmo criar regras de *disaster discovery* que auxiliam no processo de recuperação dos dados (LUSE; YANG, 2021). Por esses diferenciais, a solução vem sendo amplamente utilizada e foi adotada como solução suportada dentro do Rancher. Em sua versão v1.1, o suporte da arquitetura ARM64 em fase de *beta* tornou possível a adoção da ferramenta para a pesquisa.

Para descrever o funcionamento da ferramenta, a figura 2.7 mostra um gráfico descrevendo o processo de leitura/escrita envolvendo os volumes, a própria ferramenta LongHorn e as réplicas das instâncias e de discos. Dentro de cada nó do *cluster* Kubernetes roda uma instância chamada "*LongHorn Manager*". Esse *Pods* é responsável por não somente criar e gerenciar os volumes dentro do *cluster* Kubernetes, mas também lidar com as chamadas da API a partir da UI. O *LongHorn Manager* comunica então com a API do servidor Kubernetes para que seja criado um volume CRD (*custom resource).* Somente então, ao receber a resposta da API informando que o novo volume CRD foi criado, o *LongHorn* cria o novo volume.

Todo esse procedimento, incluindo a característica de subir 3 réplicas por padrão em *nodes* distintos para prover uma maior disponibilidade, é feita de forma automática sem precisar de nenhuma interferência manual.

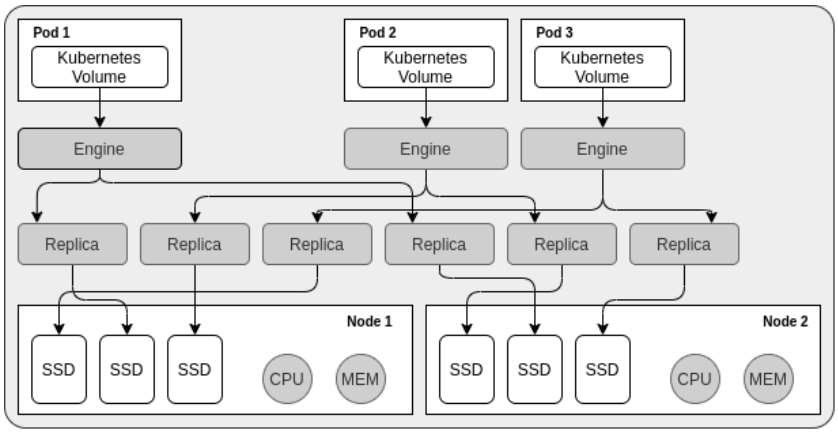


Figura 2.7: Esquemático representando o funcionamento da ferramenta de *storage* LongHorn - *Adaptado de (LUSE; YANG, 2021)*

Tendo a figura 2.7 como referência, pode-se então verificar algumas características da ferramenta:

- Possui três instâncias de volumes *LongHorn*
- Cada volume tem o seu próprio *controller*, chamado de *LongHorn Engine*, rodando como processo Linux

- Cada volume *LongHorn* possui duas réplicas, e cada réplica é uma processo Linux

As setas na figura representam o processo de leitura/escrita entre o volume, o *controller*, as réplicas e os discos. Com a mesma figura 2.7, é possível verificar que, mesmo se um dos *controllers* falharem, o funcionamento dos demais discos não são afetados.

Dessa forma, a utilização da ferramenta traz benefícios para o estudo uma vez que, ao realizar um procedimento de aprendizado de máquina não é desejável que haja uma perda de dados ou que os mesmos venham a ser corrompidos. Pode-se fazer uso da característica e da funcionalidade de aceitar múltiplos acessos de uma única vez aos discos, opção que o *storage class* padrão do Rancher não oferece. No capítulo 3, onde será descrita a implementação da solução, mais detalhes sobre a ferramenta serão escritos.

2.6 Monitoramento

Durante esse estudo, com a finalidade de não trazer nenhum *lock-in* relacionado a ferramentas prioritárias, ferramentas de código aberto foram preferencialmente escolhidas. Essa atenção também foi levada em consideração para escolher a ferramenta de monitoramento. Essa seção possui uma relevância alta dentro do trabalho pois, é através do monitoramento que todos os dados referente aos resultados serão levantados e uma posterior análise será realizada.

A ferramenta escolhida para gerar os gráficos de desempenho dos dispositivos durante a execução das cargas de trabalho foi o Grafana. Essa ferramenta é amplamente utilizada dentro dos NOCs, *network operations center*, por entregar ao usuário uma ampla visão acerca dos *hosts* (físicos, virtuais e *on-premise*) e possibilitar a modificação e criação de novas telas de monitoramento, chamadas de *dashboard*. Com o seu auxílio, é possível entender e analisar as possíveis falhas de todo um ambiente e trabalhar de forma incisiva e direcionada em cima do agente causador.

Para que as métricas sejam adquiridas, cada dispositivo deve possuir um tipo de *exporter* instalado. Dentro do Grafana por padrão, é utilizado o Prometheus, que consegue de forma automática adquirir dados sobre *hardware* e *kernel*. Caso o usuário necessite adquirir informações que não são oferecidas pelo Prometheus, escolhas entre diversos outros *exporters* existentes podem ser realizadas, ou até mesmo customizações próprias.

Trabalhando integrado com diferentes *exporters*, a solução de monitoramento escolhida é também parte do *marketplace* do Rancher, fazendo parte assim do seu catálogo de serviços. Dessa forma, o usuário que deseja utiliza-lo tem a sua instalação simplificada e pode fazer uso das diversas *dashboards* pré-construídas.

Voltado ao Kubernetes, a solução de monitoramento irá provisionar as métricas, bem como criar gráficos de cada elemento monitorado de forma automática, e para todos os dispositivos presente no *cluster*. Cada *pod* bem como seus *conteineres* e serviços serão monitorados, tornando possível verificar questões como tráfego gerado, recursos utilizados e estado do *node*.

Capítulo 3

Implementação da solução

3.1 Metodologia

Nesse estudo foram utilizados elementos comuns dentro de um ambiente de desenvolvimento moderno. A adoção de ferramentas *open-source* foi realizada para que o uso do aprendizado de máquina fosse simplificado, tornando-o prático e acessível a todos.

Com o objetivo de apresentar uma alternativa para a distribuição da carga envolvida em um *pipeline* de treinamento de máquina, a ferramenta Kubernetes fez-se essencial. Sua presença enriquece o estudo pois encontra-se em conformidade com o caminho em que o mercado está seguindo. Uma vez que a presença do aprendizado de máquina está em crescimento constante, faz-se necessário que usuários comuns - aqueles que não possuem muita familiaridade no assunto - consigam desenvolver seus próprios *scripts* de forma simples e rápida.

Os *scripts* utilizados foram retirados de testes realizados localmente e desenvolvidos em conjunto pelo grupo de pesquisa da UnB (Universidade de Brasília). Também foram utilizados modelos retirados de artigos e testes publicados. Apesar do objeto de treinamento não ser um item importante para o estudo, durante as execuções foi utilizado um modelo de teste MNIST para reconhecer dígitos escritos manuais. O uso da ferramenta não está dependente desse modelo, podendo receber outros *scripts* prontos ou até mesmo construídos pelo usuário.

Para esse estudo foi utilizado um único modelo de teste afim de obter dados homogêneos e métricas que condissessem com o mesmo modelo. Testes utilizando outros exemplos MNIST também foram efetuados mas apenas para caráter de verificação, sendo totalmente possível ser realizado novamente, com as mesmas implementações.

Para as medições de consumo energético, foram utilizados duas ferramentas diferentes. Para mensurar o consumo realizado pela *Raspberry Pi* foi feito o uso de um dispositivo desenvolvido pela Ruideng chamado UM24C. O *hardware* é acoplado na fonte de energia entre a tomada e o cabo que alimenta a *Raspberry Pi*. Através de um *software* e da conexão *bluetooth*, é possível adquirir de forma automática os valores de consumo a cada meio segundo, sendo esse o seu tempo de amostragem. Já para mensurar o consumo energético da máquina virtual, foi utilizado o *software*

HWMonitor fornecido pela empresa CPUID. Com um tempo de amostragem maior, foi adquirido dados de consumo energético a cada um segundo.

3.2 Cluster Kubernetes

Como primeiro elemento do estudo tem-se a definição do *cluster* Kubernetes que será utilizado. Nesse estudo, todas as ferramentas utilizadas são instaladas dentro do *cluster*, tornando-o como elemento principal e fundamental da arquitetura. Diante desse fato, a escolha deve ser realizada de forma rigorosa e detalhada, para não correr o risco de criar uma camada desnecessária que demandará recurso demasiado de toda a sua topologia e contribuirá de forma negativa no desempenho final.

Diante inúmeras possibilidades existentes para a criação de um *cluster* Kubernetes, alguns elementos devem ser levados em consideração para a escolha da opção correta. Para atender as mais diversas necessidades, existem ferramentas que provisionam o *cluster* em uma topologia baseada em *cloud*, *on-premises* e até mesmo em dispositivos voltados para dispositivos de borda (*edge devices*). A forma que cada opção é construída e como ela opera interfere no funcionamento do *cluster*, afetando diretamente no desempenho e na estabilidade.

Com o intuito de provisionar um *cluster* composto por dispositivos *Raspberry Pi*, foi utilizado o *K3s*, solução própria da Rancher. Essa alternativa mostra-se eficaz para o estudo uma vez que seu instalador é um binário redimensionado para possuir um tamanho de apenas 40 MB. Comparado com a solução tradicional oferecida pelo *RKE*, o *K3s* teve parte dos *drivers* removidos e outros substituídos por *add-ons*. Através dessas alterações, é possível provisionar um *cluster* de forma mais rápida, e em qualquer dispositivo que possua pelo menos 512MB de memória RAM. Essa opção é aprovada pela CNCF como uma opção certificada de Kubernetes, o que traz benefícios de segurança e suporte.

A ferramenta *k3s* escolhida possui uma pequena desvantagem encontrada durante o provisionamento do *cluster*. Diferente de outras opções, essa não possibilita nativamente a opção de definir mais de um "mestre" dentro da topologia construída. Dessa maneira, pontos importantes como redundância e alta disponibilidade são deixados de lado caso o usuário acabe por instalar a opção padrão. Se esse fato for um problema para a sua aplicação, é possível conectar o *cluster* com uma plataforma externa de banco de dados e definir mais de um nó servindo a API do Kubernetes, por exemplo. Essa instalação é explicada na página oficial da Rancher[1] onde maiores detalhes podem ser verificados. Realizando essa instalação, um balanceamento interno dentro do Rancher é garantida, assim como uma maior tolerância em termos da disponibilidade dos nós.

Antes de iniciar o processo de instalação, é necessário seguir alguns pré-requisitos com o intuito de preparar as máquinas para o *cluster*[2]. Após, com um único comando, a instalação é iniciada. A partir de um *script* pronto, onde algumas variáveis podem ser definidas, o *cluster* é criado e a ferramenta *kubectl* para gerir o *cluster* é automaticamente instalada. O comando abaixo mostra

[1]Disponível em https://rancher.com/docs/k3s/latest/en/installation/ha/
[2]Disponível em https://rancher.com/docs/k3s/latest/en/installation/installation-requirements/

como foi instalado o *cluster*.

```
curl -sfL https://get.k3s.io | INSTALL_K3S_VERSION=v1.19.8+k3s1
    K3S_KUBECONFIG_MODE=0644 sh -
```

Listing 3.1: Comando realizado para provisionar o *cluster*

Como parâmetros do comando, foram definidas a versão do Kubernetes a ser utilizada e a permissão do seu respectivo arquivo de configuração. Com esses dois parâmetros, é garantido que a versão do *cluster* seja compatível com as administradas pelo Rancher.

Já para adicionar os nós, um outro comando é realizado. A partir de uma chave codificada armazenada dentro do nó *server*, é feita a adição dos agentes. O comando abaixo identifica o procedimento executado. Doravante, o processo de criação do *cluster* foi realizado e o mesmo encontra-se pronto para receber a carga.

```
curl -sfL https://get.k3s.io | K3S_URL=https://k3s-server:6443 K3S_TOKEN=
    mynodetoken sh -
```

Listing 3.2: Comando realizado para adicionar *workers* no *cluster*

Para exemplificar a arquitetura base estudada, o diagrama da figura 3.1 abaixo é apresentado. Para testar os resultados, quantidades diferentes de *Raspberry Pi* foram usados. Dessa forma, os números de *agent nodes* foram variados.

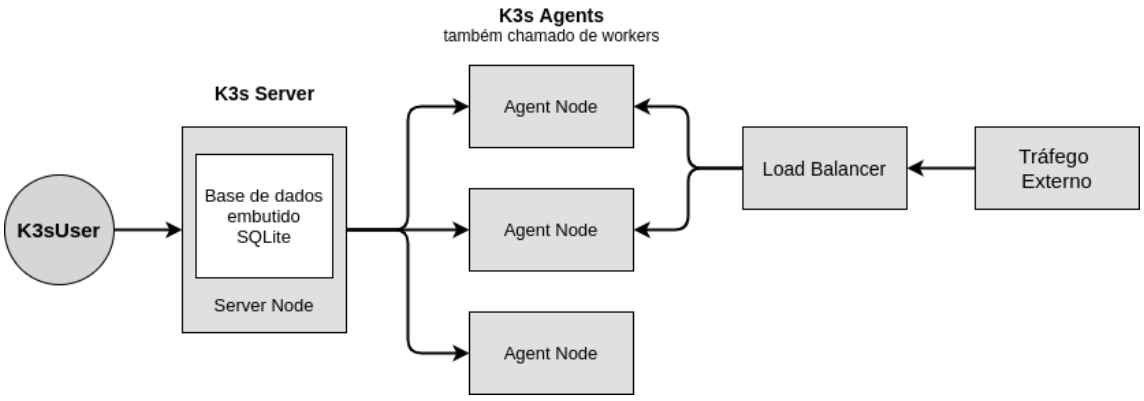


Figura 3.1: Exemplo de arquitetura construída no *cluster* K3s - *Adaptado de (RANCHER, 2021a)*

A partir de uma máquina virtual criada exclusivamente para operar como *K3s server*, o *cluster* foi provisionado. Assim como mostrado na figura 3.1, foi implementado um *database* integrado SQLite para armazenar imagens e outros elementos comuns da estrutura. Diretamente ligado ao *K3s server* estão os *Raspberry Pi*, os *agent nodes* da estrutura. Cada um recebe as cargas comportadas dentro de *conteineres* Docker estando sob gerenciamento de um *load balancer* que balanceia os acessos recebidos externamente.

Para melhor entender o funcionamento do *cluster* K3s, a seção abaixo explicitará mais a fundo os elementos presentes em cada nó agente bem como os elementos presentes no nó servidor. Será possível também verificar pequenas diferenças entre o *cluster* Kubernetes chamado de K8s e o *cluster* Kubernetes chamado de K3s.

3.2.1 Elementos do K3s

Na figura 3.2, os elementos presentes em um *cluster* Kubernetes provisionado pelo K3s são destacados e uma ligação entre os mesmos é apresentada.

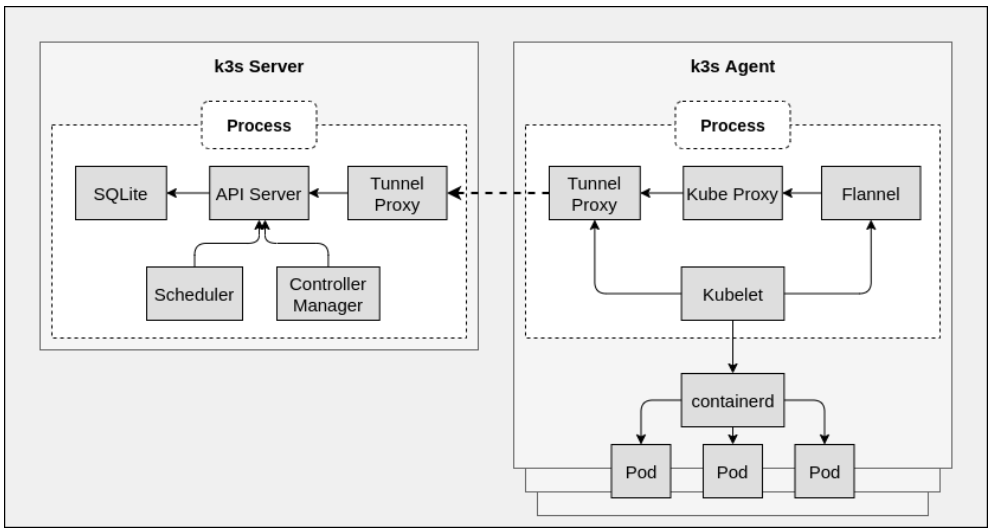


Figura 3.2: Representação dos componentes de um *cluster* K3s - *Adaptado de (CNCF, 2020a)*

Analisando a figura 3.2, diferentemente da figura 2.4 que foi utilizada para descrever o *cluster* Kubernetes K8s, os termos "mestre" e "trabalhadores" foram substituídos por "server" e "agent". Dentro do K3s, o nó *server* possui o papel de *control plane* e também, assim como os nós *agent*, são encarregados de executar as tarefas. Essa característica evidencia uma primeira diferença em relação ao *cluster* Kubernetes K8s que, por padrão, isola o nó mestre de toda a carga fazendo com que ele seja responsável apenas por administrar o *cluster*. Caso o usuário queira que o nó mestre também seja trabalhador do *cluster*, o título de "worker" deve ser atribuído a ele.

Os serviços do Kubernetes são criados dentro dos elementos denominados *pods* e executados nas máquinas disponíveis que compõem o *cluster*. Seguindo uma orquestração baseada em recursos disponíveis, cada serviço é direcionado para um *host* e ele torna-se responsável por hospedar e servir esse serviço. Caso esse venha a ficar sobrecarregado, o serviço pode ser redistribuído para um outro *host*. Toda essa administração é feita através da rede que o próprio Kubernetes cria. Dessa maneira, o Kubernetes fornece a infraestrutura e as ferramentas necessárias para manter a conectividade de rede entre as suas aplicações e serviços.

3.2.2 Rede do *cluster*

Dentro da rede, cada *pod* recebe um endereço IP automaticamente. Isso significa que o processo de mapeamento de portas e criação de *links* é invisível para o usuário e que cada *pod* pode ser tratado como uma máquina virtual ou até mesmo *host* físico em relação a alocação de portas, descoberta de serviços e balanceamento de cargas.

De forma geral e simplificada, o Kubernetes segue os seguintes requisitos para implementar a rede:

- Todos os *pods* em um nó podem comunicar-se com outros *pods* em todos os outros nós, sem o uso de NAT.
- Os componentes do Kubernetes podem comunicar-se com todos os *pods* dentro do nó.

Dessa maneira, os *pods* possuem endereços IP providos pelo Kubernetes e os contêineres dentro de um *pod* compartilham seus *namespaces* de rede - incluindo seus endereços IP e MAC. Isso significa que todos os contêineres podem alcançar as portas uns dos outros no *localhost*. Os contêineres devem coordenar o uso da porta, assim como é realizado em uma máquina virtual. Isso recebe o nome de modelo "*IP-per-pod*".

Para exemplificar o funcionamento da rede dentro de um *cluster*, um exemplo será realizado. Com o intuito de simular a comunicação entre dois *pods* que estão hospedados em dois dispositivos diferentes, na figura 3.3 foi desenhado um diagrama.

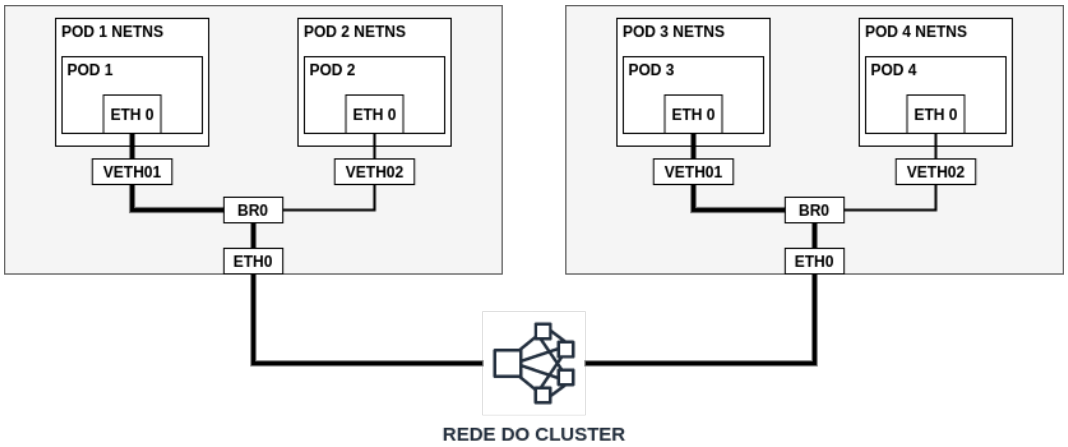


Figura 3.3: Rede interna do Kubernetes

A figura 3.3 possui a representação de dois nós dentro de um único *cluster* Kubernetes. Em negrito, há uma rota descrevendo a tentativa de comunicação entre os *pods*. Para entendimento, o processo terá início no nó 1 e terminará no nó 2, em um fluxo da esquerda para a direita.

No cenário de exemplo, o *Pod 1* inicia a comunicação criando um pacote com o IP do *pod* de destino, o *Pod 3*. Como ele não está dentro do nó de origem, o pacote deve ser encaminhado para a rota padrão *eth0* através da rede virtual *veth01* e da *bridge BR0*. Nesse momento, o pacote pode ser encapsulado em um pacote VXLAN antes de ir para a rede, ou a própria rede pode ser configurada com as rotas estáticas e sair de *eth0* inalterado. Nesse momento a rede do *cluster* é encarregada de direcionar o pacote para o nó correto.

No momento da criação do *cluster*, pode-se escolher qual tipo de rede o usuário deseja. São os chamados "*CNI Providers*". Esses provedores são *frameworks* que facilitam a configuração da rede

tornando o processo dinâmico. Algumas opções padrão são ofertadas - Flannel, Calico, Canal e Weave. Cada uma cumpre com os pré-requisitos ditados pela necessidade do Kubernetes e pode oferecer algumas outras vantagens para o usuário, como é o caso do Flannel, que configura uma rede de sobreposição IPv4 de camada 3. Nessa rede, cada nó recebe uma sub-rede para alocar endereços IP internamente. Para mais informações a respeito dos principais provedores existentes e suas diferentes aplicações, é possível acessar o *blog* da Rancher[3].

Dentro do K3s utilizado para provisionar o *cluster* estudado, o CNI padrão é o Flannel (RANCHER, 2021b). No estudo realizado o padrão foi seguido, porém o usuário está livre para alterar o CNI de acordo com a sua necessidade e preferência.

3.3 Kubeflow

A instalação da ferramenta é feita dentro do *cluster* Kubernetes, através de um terminal. Com o acesso ao servidor *master*, aquele que gerencia todo o *cluster*, o processo de instalação e configuração pode ser realizado.

Para que a instalação seja bem-sucedida, há o pré-requisito mínimo de existir pelo menos um *host* dentro do *cluster* com 12GB de memória ram e 4vCPUs. Para atender essa demanda, uma VM com essa configuração foi provisionada e adicionada ao *cluster*. Para maior estabilidade, duas novas máquinas, com 8GB de ram cada, foram também adicionadas ao *cluster*.

Por possuir o *cluster* Kubernetes já em funcionamento, foi escolhida a opção de instalação em um *cluster* Kubernetes já existente. Esse tipo de instalação é realizada através de um único arquivo de configuração pré-preparado, que pode ser totalmente customizado. Mais detalhes acerca da instalação podem ser vistos no site oficial do Kubeflow (KUBEFLOW; VASCONCELOS, 2021).

Para adequação e compatibilidade da ferramenta com o Kubeflow, foi necessário alterar a versão do elemento "argoexec", que por padrão vem com a versão v2.3.0. Durante testes executados, foi encontrado uma incompatibilidade desse versão padrão instalada com os dispositivos *Raspberry Pi*. Por se tratar de um *conteiner* usando uma imagem definida, foi possível substituir a versão padrão por uma compatível com *ARM*, listada no repositório oficial do *argoproj* presente no Docker Hub. Com isso, o primeiro desafio foi superado e a integração de dispositivos *ARM* passou a ser possível.

Próximo passo, possuindo o *cluster* Kubernetes provisionado e o Kubeflow em funcionamento, foi realizar a instalação da ferramenta de provisionamento de disco *LongHorn*. Sua presença foi de extrema importância pois é necessário que haja um *storage* compatível com múltiplos acessos de leitura e escrita, o que não há no *cluster* Kubernetes padrão.

Para elucidar o funcionamento do LongHorn junto com o Kubeflow, a figura 3.4 foi adicionada ao documento. Representando a *dashboard* do serviço, visto a partir da interface do usuário, é possível verificar a criação dos volumes após a execução de um *pipeline* de aprendizado de máquina bem como o total de disco disponível e o estado dos nós presente no *cluster* Kubernetes.

[3]Disponível em https://rancher.com/blog/2019/2019-03-21-comparing-kubernetes-cni-providers-flannel-calico-canal-and-weave/

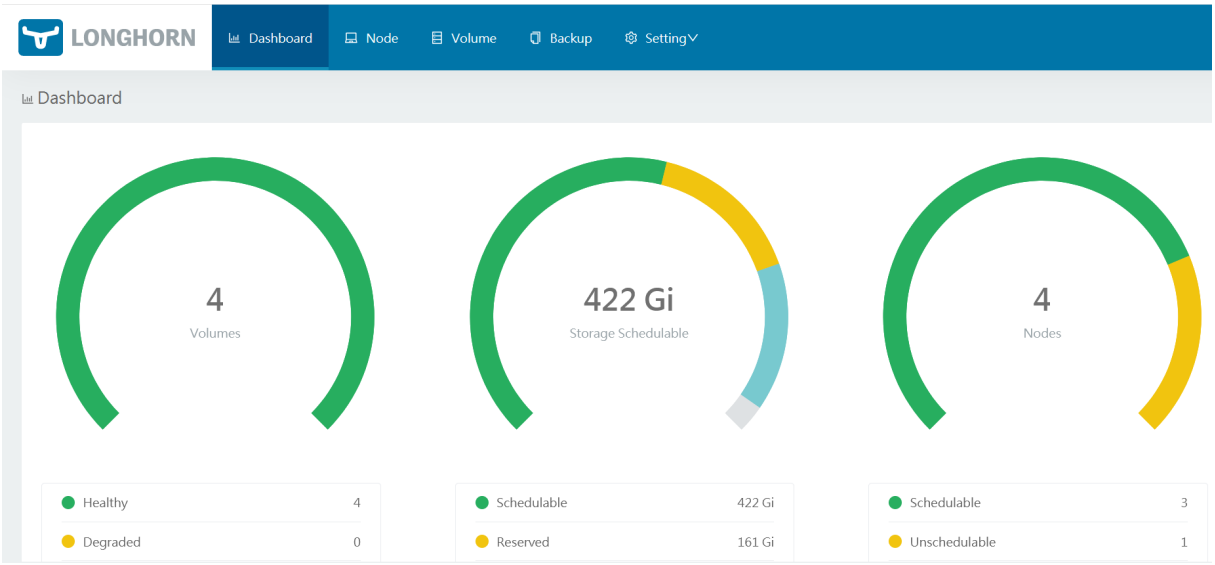


Figura 3.4: *Dashboard* do serviço *LongHorn*

A figura 3.4, assim como a figura 3.5, serve para mostrar como o *LongHorn* facilita não somente a visualização do usuário, mas também o gerenciamento dos discos dentro do Kubernetes. Esse procedimento pode ser complexo caso o usuário possua apenas o CLI *kubectl* do Kubernetes e um terminal porém, torna-se automático com o auxílio da ferramenta.

Ainda na figura 3.5, há a especificação de cada réplica criada e seus *holders*. Percebe-se, pela imagem, que cada uma das três replicas foi destinada a um *host* diferente. Essa característica põe em prática o funcionamento do LongHorn de ser robusto contra falhas e propor um plano de recuperação dos dados.

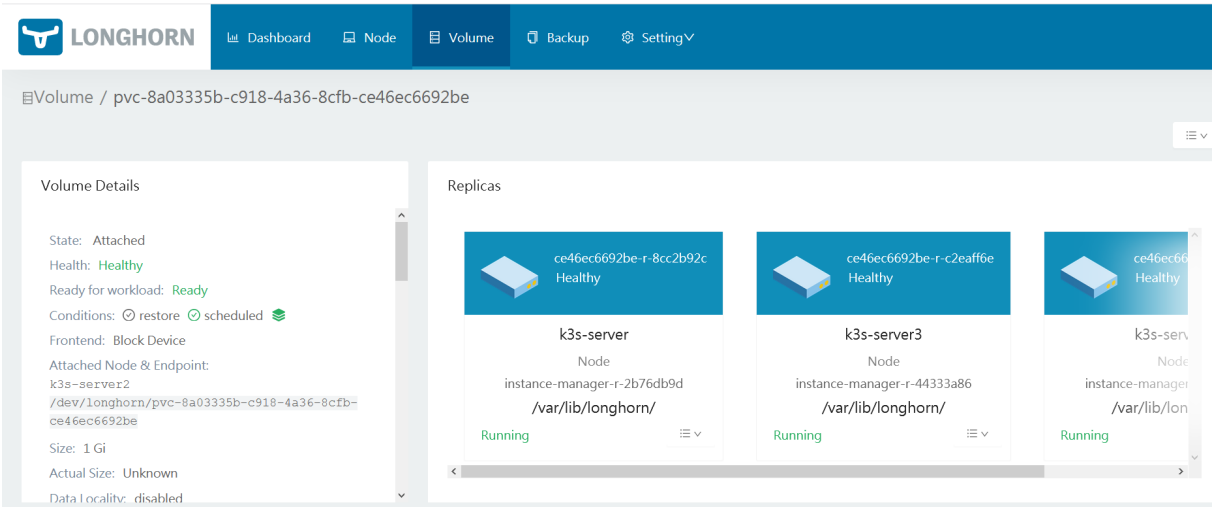


Figura 3.5: Volumes criados pelo LongHorn

Uma vez realizada o procedimento de criação dos discos, não é necessário repetir o procedimento para as demais etapas do *pipeline*. O espaço em bloco criado é armazenado e compartilhado por

todas as etapas sucessoras, sendo possível exportar dados salvos de uma etapa para outra.

Cada *pipeline* pode ser construída de acordo com a necessidade do usuário. Em alguns dos testes executados, foram utilizados exemplos MNIST para avaliar a performance. Na figura 3.6 um procedimento executado foi evidenciado, bem como suas etapas realizadas. Em cada etapa um *pod* é criado e atribuído a um dos dispositivos do *cluster* que esteja em condições de recebe-lo[4]. Com isso, cada *Raspberry Pi* fica responsável por uma ou mais etapas, recebendo o papel de executar aquela etapa do *pipeline*.

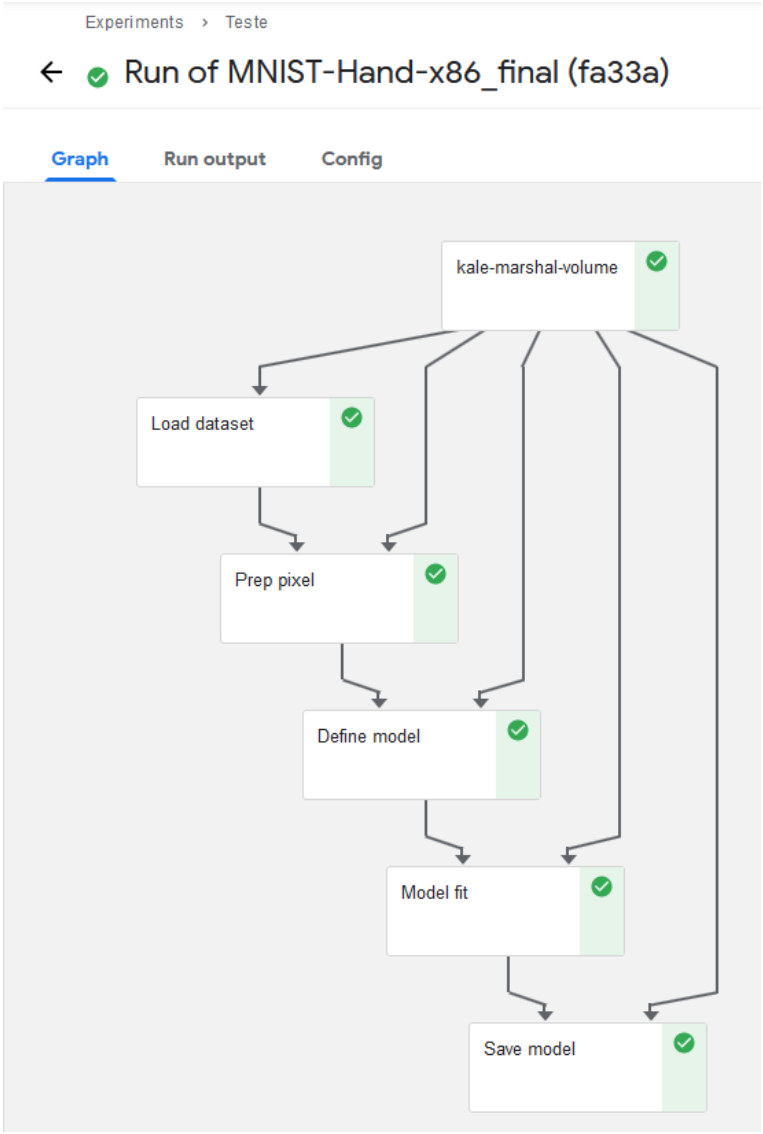


Figura 3.6: *Pipeline* exemplo - MNIST Written Digit

Para visualizar os resultados de cada etapa, o Kubeflow possibilita que o usuário selecione qualquer uma das etapas executadas e abra o *log* daquela execução. Essa opção permite que o usuário abstraia a complexidade do *cluster* e consiga enxergar os *logs* de uma maneira mais prática

[4]Segundo os critérios próprios do Kubernetes

e facilitada. A imagem 3.7 mostra os *logs* da etapa "*Model fit*", responsável por treinar o modelo.

Figura 3.7: Acurácia do treinamento do modelo de teste

Com o auxílio da figura 3.7, é possível verificar o resultado do treinamento, ou seja, sua acurácia, bem como o objeto "*model*" sendo criado, salvado e pronto para ser passado adiante, para a etapa responsável por salvar os pesos e os parâmetros obtidos após o treinamento.

Após a última etapa, por possuir um ponto comum de *storage*, o usuário pode verificar a presença do arquivo de formato ".h5" em qualquer *host*, no diretório especificado na figura 3.5.

3.3.1 Criação de um *pipeline*

Uma vez elucidado o procedimento de execução de um *pipeline*, é necessário explicitar como foram criados os *pipelines* de teste. Dentro do Kubeflow, assim como a figura 2.5 mostra como elemento da sua primeira etapa - a etapa de "*Research*"- , é possível adicionar um *notebook* Jupyter[5] e desenvolver o seu código dentro do Kubeflow.

Utilizando a linguagem *Python*, foram desenvolvidos os exemplos de teste para esse estudo. De forma a possibilitar a criação de mais de um *conteiner*, o código foi dividido estrategicamente para que cada etapa pudesse ser realizada em um *host* diferente. A estratégia utilizada pode ser verificada nas figuras 3.8 e 3.9. Com o auxílio do projeto *open-source Kale* (FIORAVANZO, 2019), foi possível atribuir legendas para cada uma das células. Esse passo tornou-se fundamental para

[5]Interface *online* utilizada para desenvolvimento de códigos em diversas linguagens.

o projeto uma vez que é dessa forma que o *pipeline* é montado e as relações entre as diferentes etapas são definidas.

A figura 3.8 mostra a primeira parte do *script* montado. As etapas de importação das bibliotecas e a definição das funções receberam *labels* específicas que, diferente das demais, são tratadas como variáveis globais. Assim, as demais células do *notebook* podem utilizar-se das funções definidas pois as mesmas têm conhecimento sobre elas.

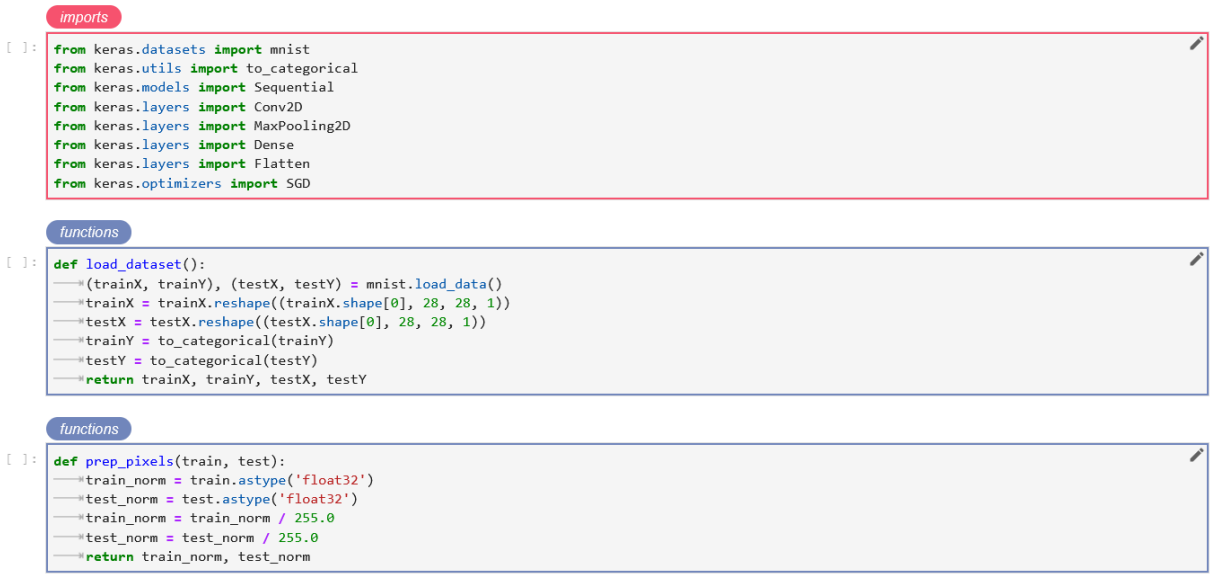


Figura 3.8: Composição do *pipeline* - Parte 1

Já na figura 3.9, estão representadas as etapas que são executadas apenas uma vez dentro de todo o *pipeline*. Ainda nessa figura, pode-se verificar a relação das etapas da figura 3.6 com o nome de cada uma das células criadas. Em cada uma daquelas etapas é executado o comando descrito na célula correspondente.

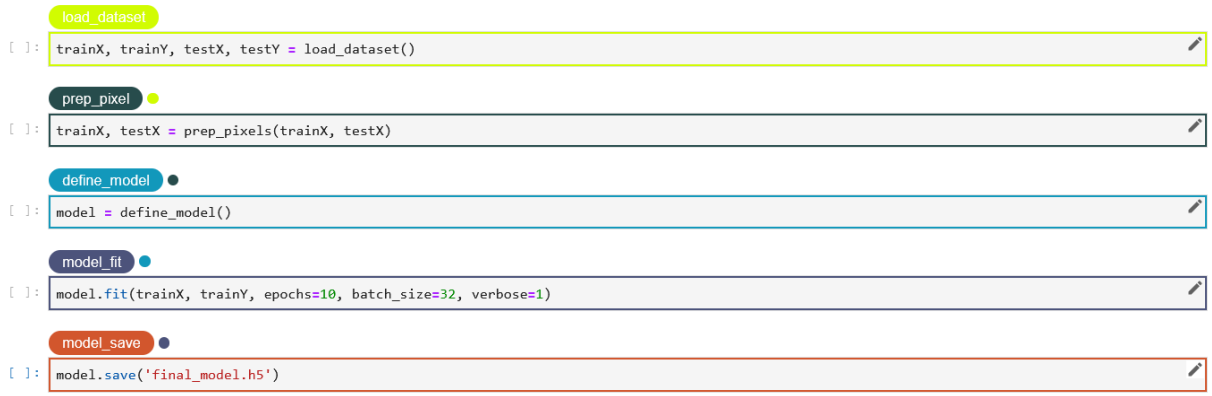


Figura 3.9: Composição do *pipeline* - Parte 2

Cada célula definida recebe um encapsulamento onde bibliotecas próprias do Kubeflow são

adicionadas e a imagem Docker a ser utilizada é definida. Dessa forma, o Docker consegue entender como cada *conteiner* deve ser criado. Nesse momento é importante verificar que a imagem escolhida é compatível com a arquitetura de processamento dos *hosts* adicionados no *cluster* Kubernetes. Nesse estudo foi necessário criar um próprio *Dockerfile* para criar uma imagem própria, compatível com a arquitetura *ARM*. Na subseção 3.3.2, mais informações acerca do Dockerfile criado serão evidenciados.

Para finalizar a evidencia da execução do procedimento, tem-se a figura 3.10 que mostra uma visão dentro do Rancher do *namespace* chamado de "*distributed-ml*".

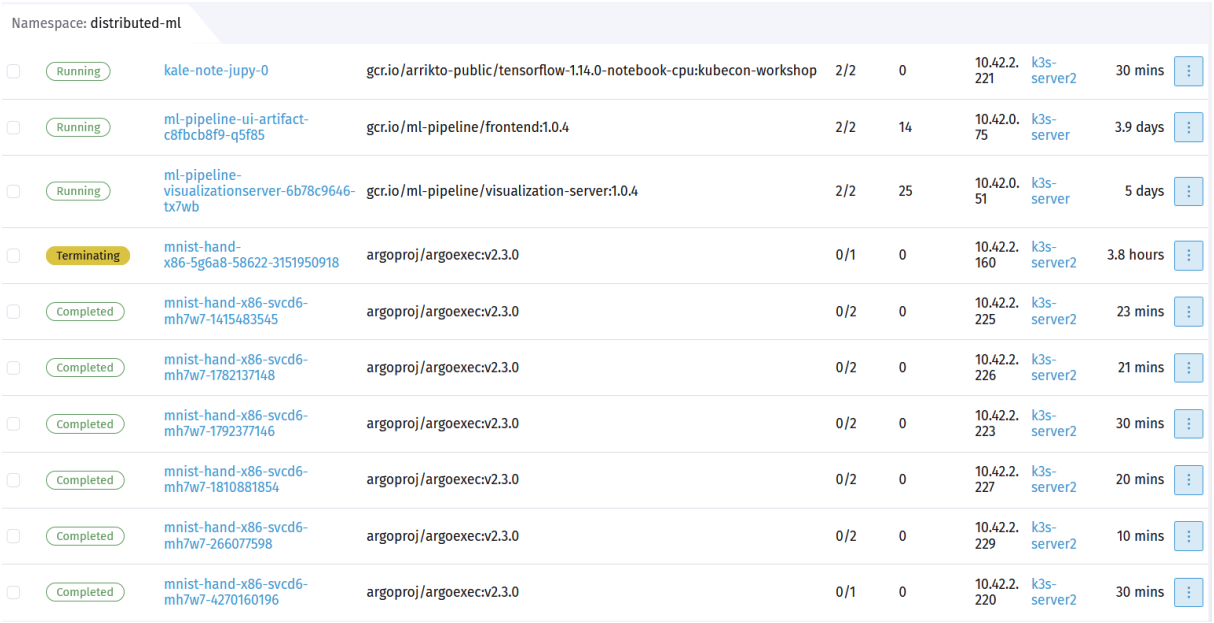


Figura 3.10: Lista dos PODs criados dentro de cada etapa

Para apresentar a figura 3.10 é necessário evidenciar cada coluna apresentada. Iniciando da esquerda para a direita, é possível notar nas três primeiras colunas, respectivamente, informações sobre o estado de cada *pod*, o seu nome e a imagem que o mesmo está utilizando. Nas próximas três colunas são apresentadas informações sobre a quantidade de *containers* existentes e que estão executando tarefas, quantidades de *restarts* do *pod* bem como o seu endereço IP dentro da rede Kubernetes. Já nas duas últimas colunas, o usuário pode verificar em qual *host* o *pod* foi atribuído e a quanto tempo o mesmo foi implementado.

Na figura 3.10 há a descrição de cada *pod* criado bem como o seu nome e a versão da imagem que foi utilizada para criar o *pod*. Os três primeiros *pods* da imagem são conteiners criados para possibilitar o uso do *notebook* Jupyter dentro do Kubeflow e possibilitar a visualização do *pipeline* da forma que é apresentada na figura 3.6. Os demais *pods* são referentes a execução do *pipeline* de exemplo.

A medida que cada etapa vai sendo executada dentro do Kubeflow, é possível verificar os *logs* referentes a criação dos *pods* onde existem informações como, por exemplo, imagem sendo baixado e o tempo que levou para baixa-la, o volume sendo anexado ao *pod*, a criação dos *conteiners*

definidos nesse *pod*. Porém, os demais *logs* são apresentados somente dentro do Kubeflow, da forma que mostra a figura 3.7. A junção desses dois pontos de coleta de *logs* permite um entendimento e acompanhamento completo sobre todo o processo de criação e execução do *pod* trabalhando de forma complementar.

3.3.2 Dockerfile

Para solucionar o problema de compatibilidade com as imagens padrão e oferecidas pela comunidade, foi necessário criar uma imagem própria. Dentro do conceito do *Docker*, cada imagem precisa de um *dockerfile* para ser contruída. Com um arquivo composto por linhas de instruções, a imagem é construída e enviada para o repositório *docker hub*, onde cada *conteiner* criado pelo Kubeflow irá ler.

Tratando-se de uma imagem que representa um sistema operacional, todas as bibliotecas e ferramentas devem ser instaladas. Tal instruções estão logo no início do Dockerfile, nas linhas 3 à linha 18 da imagem 3.11.

```
3  RUN dpkg --add-architecture armhf && dpkg --add-architecture arm64 \
4      && apt-get update && apt-get install -y \
5          openjdk-11-jdk automake autoconf libpng-dev \
6          curl zip unzip libtool swig zlib1g-dev pkg-config git wget xz-utils \
7          python3-numpy python3-pip python3-mock python-pip python-setuptools build
       -essential python-dev \
8          libpython3-dev libpython3-all-dev \
9          libpython3-dev:armhf libpython3-all-dev:armhf \
10         libpython3-dev:arm64 libpython3-all-dev:arm64 g++ gcc
11
12 RUN apt install software-properties-common -y
13 RUN add-apt-repository ppa:deadsnakes/ppa
14 RUN apt install python3.7 -y
15 RUN /bin/bash -c "rm /usr/bin/pip /usr/bin/python /usr/bin/python3"
16 RUN ln -s /usr/bin/pip3 /usr/bin/pip
17 RUN ln -s /usr/bin/python3.7 /usr/bin/python
18 RUN ln -s /usr/bin/python3.7 /usr/bin/python3
```

Figura 3.11: Trecho do *dockerfile* responsável por instalar as dependências

Para atender os pré-requisitos do Kubeflow, Kale e do TensorFlow, as bibliotecas utilizadas devem ser instaladas. Sendo assim, o restante do Dockerfile fica responsável por instala-los. A figura 3.12 evidencia a parte descrita.

```
RUN pip3 install -U --user keras_applications==1.0.8 --no-deps \
    && pip3 install -U --user keras_preprocessing==1.1.0 --no-deps \

RUN pip3 install -U --user docs\tring_parser \
    && pip3 install deprecated

RUN /bin/bash -c "update-alternatives --install /usr/bin/python python /usr/bin/
    python3 150"

RUN /bin/bash -c "pip3 install --upgrade pip"

WORKDIR /root
RUN git clone https://github.com/lhelontra/tensorflow-on-arm/
WORKDIR /root/tensorflow-on-arm/build_tensorflow/
RUN git checkout v2.3.0
CMD ["/bin/bash"]
```

Figura 3.12: Trecho do *dockerfile* responsável por instalar ferramentas necessárias

Dessa forma, utilizando uma imagem própria com todos as dependências resolvidas e TensorFlow instalado, os *conteiners* podem agora serem hospedados nos *Raspberry Pi*. O *Dockerfile* criado pode ser visto por completo no Anexo II.

Capítulo 4

Análise dos resultados

Este capítulo descreve os testes realizados utilizando a ferramenta, detalhando os resultados, as principais vantagens bem como suas limitações.

4.1 Solução estudada

Na seção 2.3, duas estratégias utilizadas para realizar o paralelismo foram explanadas. É apresentado a aplicação do aprendizado de maquina em dispositivos *Raspberry Pi* utilizando *Kubernetes* com a finalidade de propor uma solução compatível com a tendencia de arquitetura atual e colaborar com o andamento do assunto.

Através deste estudo, é esperado que uma alternativa para o aprendizado de máquina seja apresentada e informações acerca do seu desempenho traga conhecimento para os leitores. Combinando duas ferramentas fortemente adotas, há a possibilidade de entregar uma solução condizente e preparada para compor o cenário da maioria dos ambientes.

Com o objetivo de mostrar um dos empregos do dispositivo *Raspberry Pi*, foi desenvolvido o estudo adicionando-os em um *cluster* Kubernetes. É objetivo então levar essa abordagem mostrando que, independente do resultado, a arquitetura *ARM* mostra estar pronta para o aprendizado de máquinas. Porém, para validar algumas teorias, também serão apresentados comparativos com o método tradicionalmente utilizado. Assim, os mesmos testes executados no *cluster* Kubernetes também foram executados em uma única máquina, fora do contexto do Kubernetes. Dessa forma, é possível verificar resultados de ambas as execuções.

É importante ressaltar que a solução faz uso de uma composição nova de arquitetura e que o intuito também é abranger dispositivos de baixo consumo de energia. Assim, tem-se um duplo benefício sendo entregue, trazendo mais mais vantagem para o usuário.

4.2 Métricas aplicadas durante treinamento

Todo o comparativo foi montado em cima da ferramenta de monitoramento Grafana, a partir de métricas exportadas das máquinas, tanto a nível de *hardware* quanto a nível de serviço. A utilização de *dashboards* permite que inúmeras métricas personalizadas possam auxiliar no processo de monitoramento da rotina de aprendizado de máquina. Através dessa possibilidade, é possível exportar gráficos que auxiliam no levantamento de dados e inferência dos resultados.

Para validar a técnica e verificar se a implementação da mesma trazem benefícios, foram extraídos informações sobre uso de CPU, memória RAM, temperatura e consumo de energia com o objetivo de analisar o recurso computacional utilizado durante a execução de todo o *pipeline* de aprendizado. Em posse dessas informações, foi construído uma análise possibilitando a conclusão do estudo.

Durante a execução de um exemplo de teste, foram nítidos os efeitos que a execução trouxe para os dispositivos. As imagens oferecidas logo abaixo são referentes às métricas obtidas e cada uma tem o objetivo de explicitar os principais elementos analisados.

Iniciando a análise pela imagem 4.1, que evidencia a utilização de processamento de um dos *pods* criados, pode-se concluir algumas informações. Durante o período específico do treinamento, foi observado um grande crescimento e um valor alto no gráfico. Dentro de uma escala de 0 a 1, onde 0 representa o dispositivo em ociosidade e 1 para o máximo da capacidade de processamento disponível, foi possível verificar que o dispositivo trabalhou por alguns minutos dentro do seu máximo de desempenho.

Figura 4.1: Consumo de processamento

Esse resultado foi esperado uma vez que essa informação foi retirada do *pod* criado na etapa de treinamento do modelo chamada "*model_fit*" (figura 3.9). Por ser o *pod* responsável por executar o processamento de todo o aprendizado de máquinas, é compreensível o desprendimento de todo o recurso computacional solicitado.

A informação ajudou a entender que por um momento os recursos da *Raspberry Pi* foram requisitados a ponto do dispositivo ficar exclusivamente dedicado a esse único processo. Essa análise nos leva a concluir que uma rede um pouco mais complexa, que gere mais processamento,

pode resultar em uma escassez de recurso. Seguindo a análise dos resultados, é interessante também verificar o uso de memória RAM do dispositivo.

Na infraestrutura da aplicação, durante o processo de instalação e provisionamento do *cluster*, foi considerado dedicar as *Raspberry Pi* somente para serem trabalhadores. Processos não necessários para o funcionamento dos dispositivos foram desativados, bem como serviços e aplicações. Dessa maneira, os componentes do *cluster* Kubernetes eram os únicos elementos hospedados nesses *hosts* que contribuíam com a carga. Isso tornou o gráfico da figura 4.2 um item fidedigno que retrata perfeitamente o consumo de memória RAM durante a execução do treinamento.

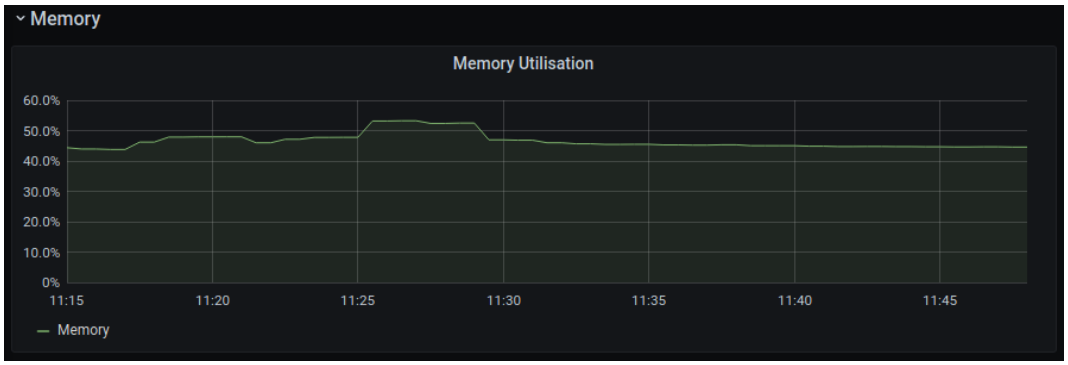


Figura 4.2: Consumo de memória RAM do *namespace*

A partir do gráfico apresentado na figura 4.2, é analisada a memória RAM de todo o *namespace* dedicado para o aprendizado de máquina dentro do *cluster*. Durante o período de execução de teste, houve um consumo médio de 50% da memória total. Como o *cluster* foi montado com *Raspberry Pi* 4 modelo B, cada dispositivo conta com 8GB de memória. Com esse gráfico, é possível ver que a memória foi suficiente e ficou longe de ser um gargalo. Em compensação, analisando a figura 4.3 também referente a memória RAM, tem-se uma visão mais ampla sobre todo o *cluster*.

Com a figura 4.3 em evidência, o panorama muda. Saindo da visão de um único *namespace* para a visão global de todo o *cluster*, é possível verificar que o dispositivo chegou ao limiar de 7GB de memória RAM. Esse resultado é encarado como um problema pois deixou o *host* trabalhando em sua quase capacidade máxima.

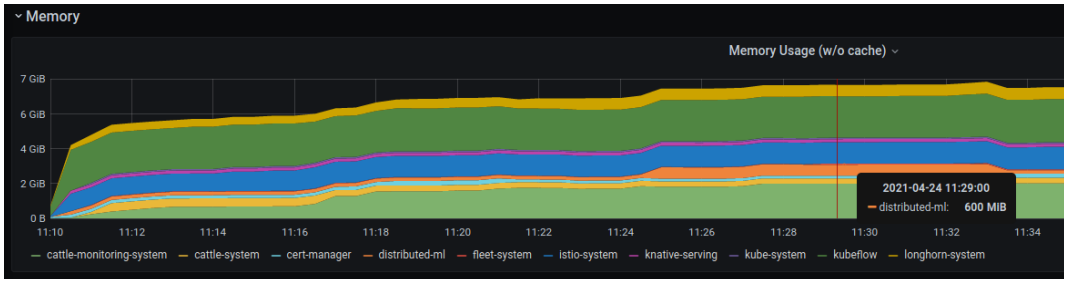


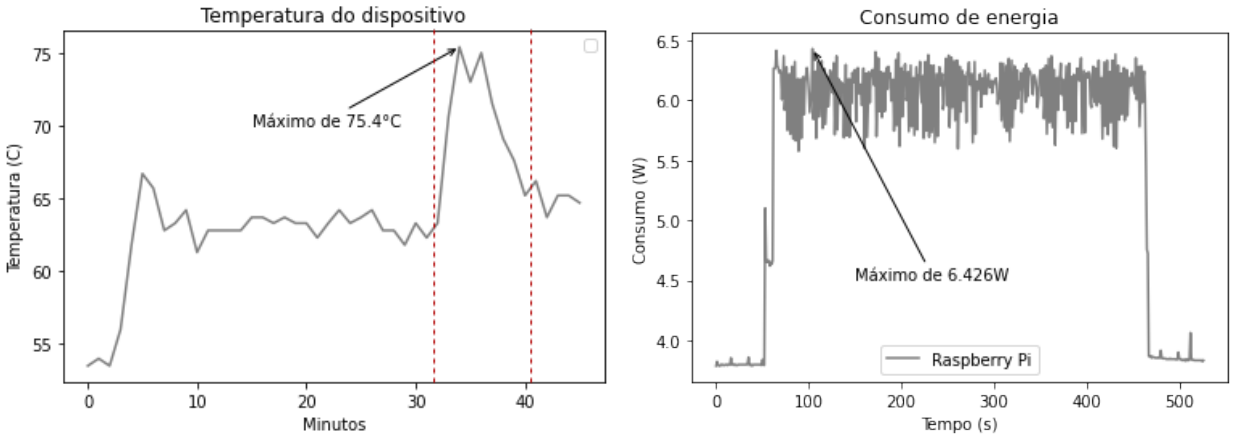
Figura 4.3: Consumo de memória RAM do *cluster*

Apesar de ter escolhido uma opção de *cluster* mais leve, exclusivo para rodar em dispositivos de

borda - *edge devices* - todos os componentes necessários do Kubeflow elevaram o uso do *host*. Ainda na figura 4.3, é possível verificar em cor laranja a contribuição que o "*distributed-ml*", *namespace* responsável pelos *pods* de treinamento, traz para todo o *cluster*. Apenas 600MiB foram utilizados de fato pelo treinamento.

Com a análise das duas últimas figuras - figura 4.2 e figura 4.3 - faz perceber que, apesar do *pipeline* não depender de muito recurso computacional, os demais elementos vitais para o funcionamento do *cluster* e da ferramenta Kubeflow contribuem drasticamente para a carga.

Para finalizar a análise dos dados adquiridos, têm-se dois novos gráficos retirados manualmente durante a execução do teste. Com o auxílio da figura 4.4, pode-se avaliar dados referente a temperatura e consumo de energia do dispositivo durante o período de processamento. Os gráficos apresentados foram construídos a partir do mesmo período de execução, durante o mesmo treinamento.



(a) Temperatura do dispositivo

(b) Consumo de energia do dispositivo

Figura 4.4: Tempo de execução e consumo de energia durante a execução de um treinamento

Começando pela imagem da esquerda (4.4a), é possível ver duas linhas verticais tracejadas na cor vermelha. Essas linhas delimitam exatamente o período em que o treinamento foi realizado. Dentro dos período delimitado, entre os minutos 30 e 40, aproximadamente, é possível verificar uma maior elevação da temperatura do dispositivo, chegando ao máximo de 75,4°C. Com uma média de 64°C, houve uma variação máxima de 11,4°C. Esse valor é explicado pela imagem 4.4b, que representa o consumo de energia durante o treinamento. É importante ressaltar que os dispositivos estavam sobre um banco de madeira, ao lado de uma janela de vidro onde a incidência de raios solares eram moderados durante o período da tarde, como mostra a Figura 2.2. Esse fator explica o motivo das *Raspberry Pi* começarem com um valor de 50°C como mostra no início do gráfico mostrado através da Figura 4.4a. Para analisar a Figura 4.4a é recomendado concluir apenas a diferença entre a temperatura máxima e a temperatura mínima.

Através de um aparelho de coleta de dados desenvolvido pela RUIDENG, o UM24C, foi possível obter os dados plotados na figura 4.4b. Com o auxílio de um *software* foi possível analisar as métricas de energia coletadas e gerar uma planilha com as informações. O tratamento dos dados

foi realizado e o gráfico apresentado pela imagem foi gerado. Da mesma forma que foi visualizado na figura 4.4a, é possível verificar um aumento no gráfico representado na figura 4.4b durante um período bem específico. Entre a octogésima (80ª) e a quadringentésima octogésima (480ª) leitura, o consumo de energia subiu de forma característica, mostrando uma média de 6.426W. Para analisar o resultado obtido, esse valor será colocado em comparação logo em breve, na seção 4.2.2.

Ambos os dados apresentados foram de grande valia para o estudo pois através deles foi possível comprovar a baixa temperatura do *Raspberry Pi* bem como sua capacidade de realizar o processamento utilizando baixa energia.

4.2.1 Considerações gerais sobre o Kubernetes

Devido ao funcionamento nativamente distribuído que o *Kubernetes* aplica, os *conteiners* respectivos de cada procedimento criado devem realizar uma comunicação com o disco provisionado a fim de realizar o armazenamento dos dados criados. Dessa forma, necessitando ficar um tempo ocioso esperando que o disco seja criado, montado e pronto para a carga, foi acrescido em média 5 minutos em cada teste. Também é possível verificar uma comunicação de rede intensa durante todas as etapas, muitas vezes entrando em estado de erro quando ocorre uma oscilação na rede durante esse momento. Diante dos dois pontos comentados acima, é possível apontar então desvantagens no uso do Kubernetes quando comparada com o procedimento realizado em uma única máquina, fora do contexto de *cluster*. Por estar diretamente atrelada a um disco dedicado, nenhuma comunicação de rede é necessária.

A utilização de *conteiners* faz com que uma imagem contendo o aparato ferramental seja necessária. Como foi comentado na seção 3.3.2, na página 34, o *dockerfile* criado deu origem a uma imagem que deve estar presente em cada *conteiner*. Essa característica intrínseca ao funcionamento do *docker* cria mais um ponto de desvantagem. O usuário agora dependente da rede para baixar uma imagem que pode chegar a mais de 5GB, 10GB. Em um teste realizado, foi alcançado um máximo de 25 minutos para efetuar o *download* da imagem criada. Entretanto, por ficar salva nos metadados do *cluster* Kubernetes, é necessário realizar o procedimento de *download* apenas uma vez.

A figura 4.4 evidencia duas métricas coletadas manualmente durante a execução de um dos testes realizados. A imagem 4.5 mostra um gráfico gerado com dez tempos de execução obtidos, no qual o primeiro apresentado mostra a primeira execução do *cluster*. Retirando a primeira execução, foi obtido uma média de 17 minutos. Essa média já sobe para 18 minutos caso seja considerada a primeira execução. Esse dado revela uma informação importante que auxilia no comparativo entre os dois tipos diferentes de dispositivos que foram utilizados - *Raspberry Pi* e máquina virtual -, como será mostrado na próxima seção.

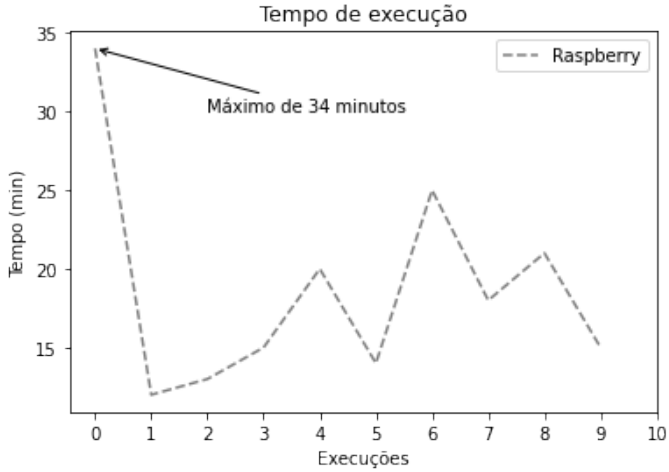


Figura 4.5: Tempo de execução do treinamento em Raspberry

4.2.2 Execução em máquina virtual

Com o objetivo de obter informações que servissem como base para as comparações, foi criada uma máquina virtual com a mesma característica de uma dispositivo *Raspberry Pi* 4 com apenas a diferença do processador. A máquina hospedeira possui como processador um *Intel Core i7-6820HQ*, com frequência de 2.7GHz. Esse processador é aproximadamente 2x mais rápido que o do *Raspberry Pi* 4 modelo B, que possui um processador de frequência de 1.5GHz.

Com posse do mesmo código executado nas *Raspberrys*, presente em sua totalidade no Anexo I, foram criados mais dois cenários de testes: máquinas virtuais fora do *cluster* e, posteriormente, máquinas virtuais dentro do *cluster*.

A figura 4.6 evidencia o mesmo gráfico presente na figura 4.5, agora com a adição dos resultados obtidos pela máquina virtual. Nesse primeiro momento, o teste foi executado fora do ambiente de Kubernetes, apenas para criar um panorama de um cenário fora do *cluster*.

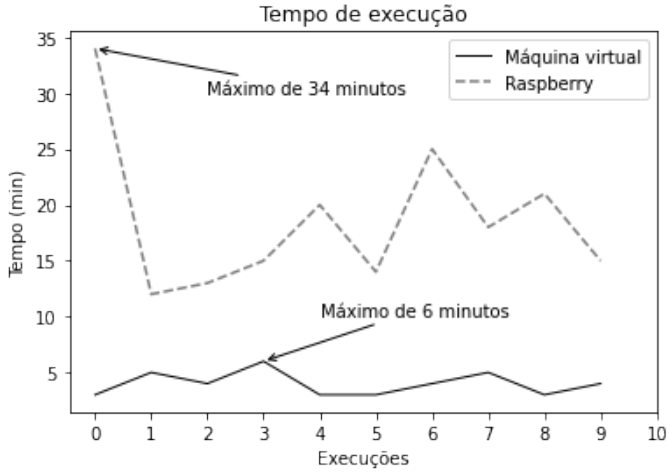


Figura 4.6: Comparação entre tempos de execução do treinamento na máquina virtual e no Raspberry

A comparação que a figura 4.6 proporciona mostra um tempo de execução máximo de 6 minutos obtido pela máquina virtual frente a um tempo máximo de 34 minutos, obtido pelo dispositivo *Raspberry Pi*. Em face do resultado, tendo conhecimento sobre o processo de obtenção da imagem dentro do *cluster* Kubernetes - explicado em sua totalidade na seção 4.2.1 - , faz-se necessária uma comparação com o Kubernetes sendo elemento comum aos dois cenários.

À vista disso, foi criado um *cluster* secundário composto somente por máquinas virtuais, com as mesmas configurações de *hardware*, diferindo somente no processamento, como mencionado acima. O mesmo teste foi executado e a figura 4.7 apresenta os resultados obtidos.

Com um *cluster* composto somente por máquinas virtuais, foi possível verificar um comportamento diferente da imagem 4.6, e com resultados mais próximos dos obtidos no cenário composto somente por *Raspberry Pi*.

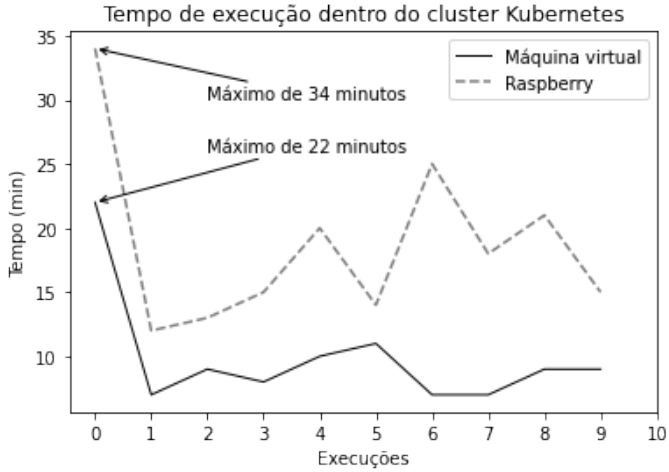
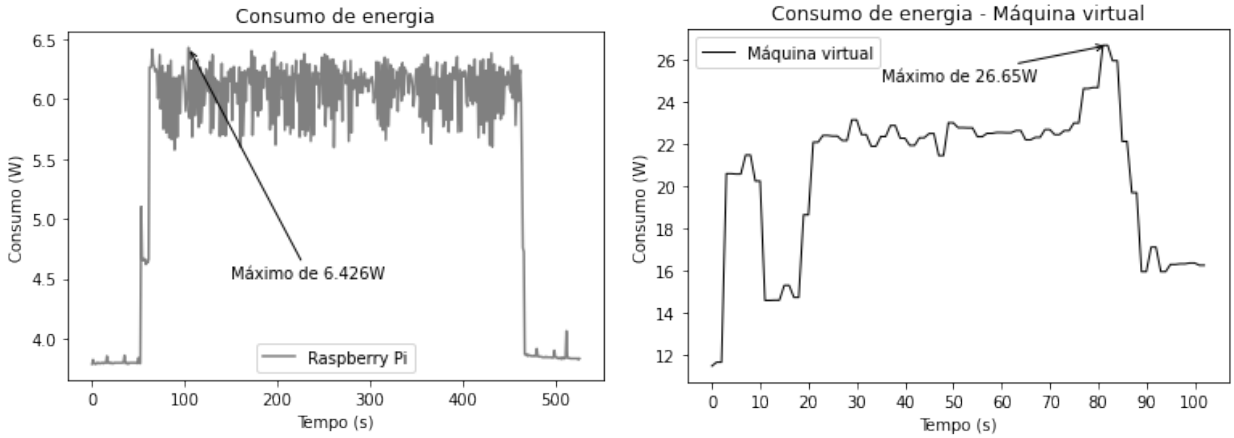


Figura 4.7: Comparação entre tempos de execução do treinamento na máquina virtual e no Raspberry, no *cluster* Kubernetes

O comportamento inevitável ocorrido nas primeiras execuções repetiu-se e o tempo médio das execuções subiram consideravelmente, chegando até o valor máximo de 22 minutos. Isso representa um aumento de 27,2% em relação ao tempo máximo obtido anteriormente. Com essa análise, é permitido visualizar os efeitos do funcionamento do *cluster* Kubernetes. Quando os dois cenários são evidenciados, há uma conclusão de que o *cluster* criou camadas que atrasam o procedimento.

Em relação a consumo de energia, é possível verificar uma grande desvantagem por parte da máquina virtual. Analisando a figura 4.8, que mostra dois gráficos lado-a-lado - figura 4.8a e 4.8b -, é possível verificar o comparativo referente ao consumo de energia entre os dois cenários.

A figura 4.8a é a mesma utilizada na seção 4.2. Ela foi reapresentada com a finalidade de ser melhor comparada com a imagem 4.8b, que mostra a energia consumida pela máquina virtual.



(a) Consumo de energia da Raspberry Pi

(b) Consumo de energia da máquina virtual

Figura 4.8: Tempo de execução e consumo de energia durante a execução de um treinamento

A partir da análise da figura 4.8b, é possível verificar um maior desprendimento de energia por parte da máquina virtual. Para executar o mesmo treinamento, foi alcançado valores médios de 24W, chegando até ao valor máximo de 26.65W. Esse valor representa em média 4 vezes mais consumo que o dispositivo Raspberry Pi, que mostrou executar as mesmas operações com média de 6W.

4.3 Resultado do treinamento

Com a finalidade de construir todo o *pipeline* e mostrar que todos os passos são possíveis de serem realizados nos dispositivos *Raspberry Pi*, o modelo treinado foi servido através de uma página *web*. No *pipeline* referente ao "*MNIST Handwritten Digit*", o usuário pode analisar o resultado do treinamento desenhando um número e verificando a previsão, como mostra a figura 4.9.

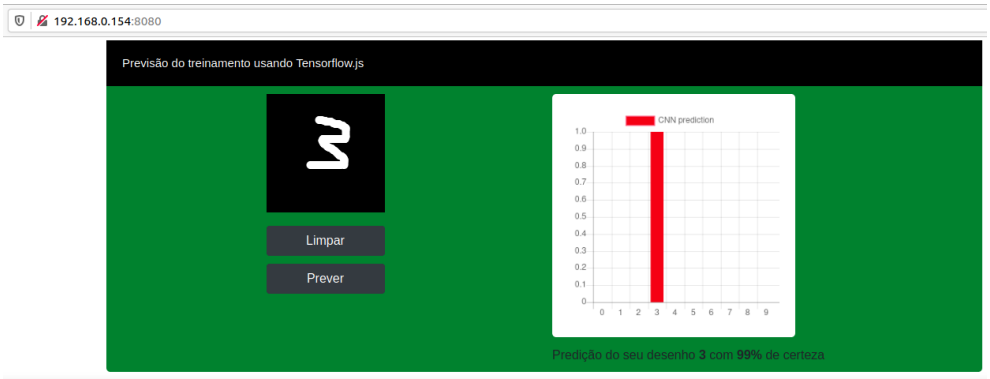


Figura 4.9: Página *web* para servir o modelo treinado

Essa predição foi realizada utilizando uma biblioteca *javascript* presente no Tensorflow, o *tensorflowjs*. Com o modelo treinado salvo, item presente no último passo do *pipeline*, é possível transformá-lo em formato *json* e servir como entrada para o *tensorflowjs*.

Fazendo uso do NodeJS para servir o projeto, a página de teste foi hospedada em um dispositivo *Raspberry Pi*. Dessa forma, foi certificado de que todo o processo fosse possível de ser executado em cima da arquitetura *ARM*, escopo principal do estudo. Para mostrar o acesso sendo realizado, a figura 4.10 mostra o *log* do momento do acesso assim como o nome do *node*.

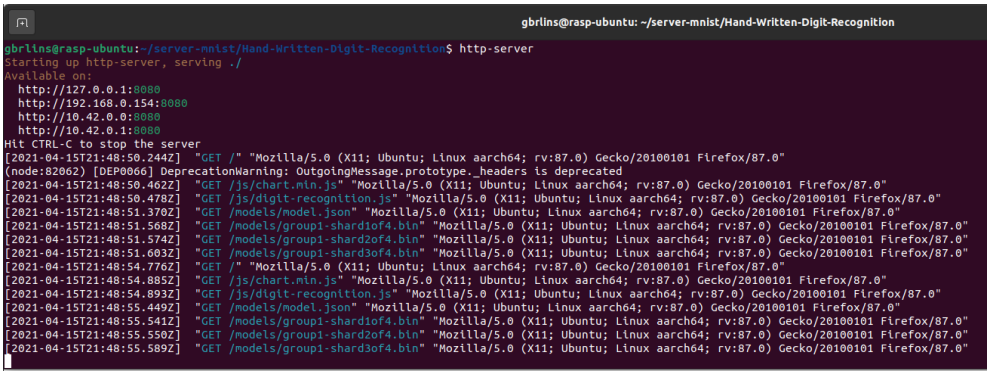


Figura 4.10: *Log* contendo registros de acesso à página *web*

É possível originar uma imagem a partir da criação de um novo *dockerfile* e escrever essa solução em um *conteiner*, criando uma aplicação *conteinerizada*. Dessa maneira, o usuário pode criar mais uma etapa em seu procedimento de aprendizado de máquina fazendo com que ela fique totalmente rodando dentro do Kubernetes.

4.4 Métricas aplicadas durante inferência

Para trazer ao estudo o emprego dos dispositivos *Raspberry Pi* na etapa de inferência do modelo treinado, foram realizados testes de carga, com o objetivo de estressar o servidor e adquirir

informações dessa ação. O objetivo é apresentar os resultados e criar um novo panorama relativo as métricas apresentadas na seção 4.2.

Trazendo uma mudança na arquitetura, é proposto utilizar as *Raspberry Pi* somente para realizar a inferência do treinamento. Dessa maneira, por deixar a etapa de treinamento ser realizada por placas de unidade de processamento visual (GPU), recomendadas para tal finalidade, o estudo encontra-se mais condizente com a abordagem que os desenvolvedores estão utilizando.

Dito isso, foi utilizado a ferramenta de código aberto *Apache JMeter*[1] para realizar o teste de carga com a finalidade de simular mais de um usuário requisitando a página de inferência apresentada na seção 4.3. A ferramenta possibilita o teste em diferentes tipos de protocolos, aplicações e servidores, incluindo o NodeJs, que foi a ferramenta escolhida para servir a página de inferência no presente estudo.

Através do *BlazeMeter*, extensão instalado no próprio navegador, foi possível realizar a gravação de todos os passos executados dentro da página bem como os parâmetros das requisições e a ordem de execução. Através do arquivo de extensão *jmx* criado, é possível carrega-lo no *software JMeter* e dar inicio aos testes.

A figura 4.11 abaixo mostra a tela do *software Apache JMeter* com o teste carregado e em execução.

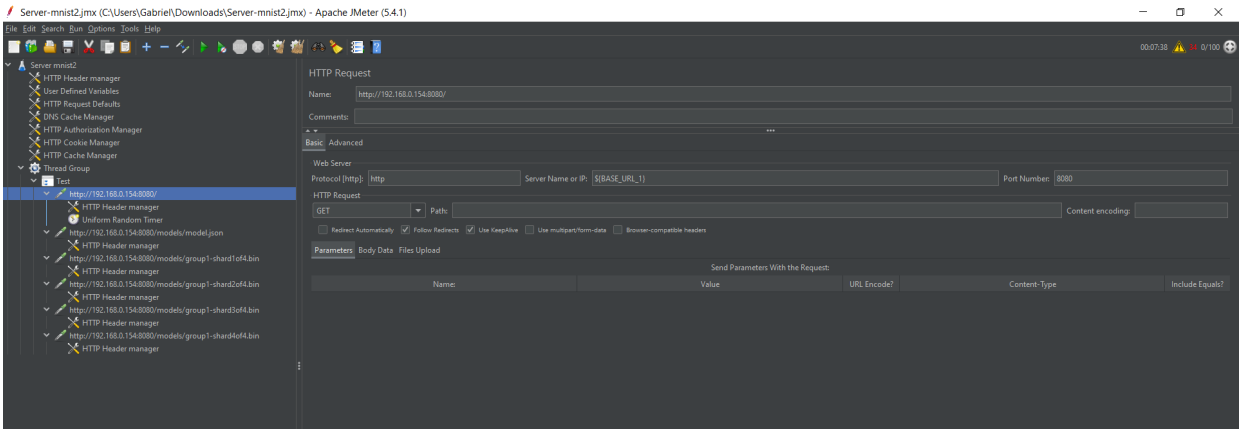
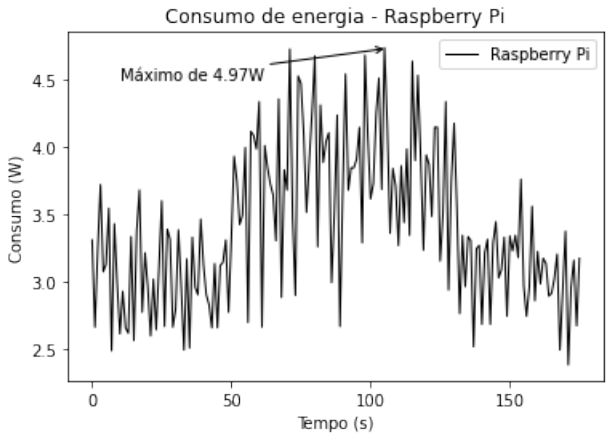


Figura 4.11: Tela do *software* Apache JMeter com o teste de carga em execução

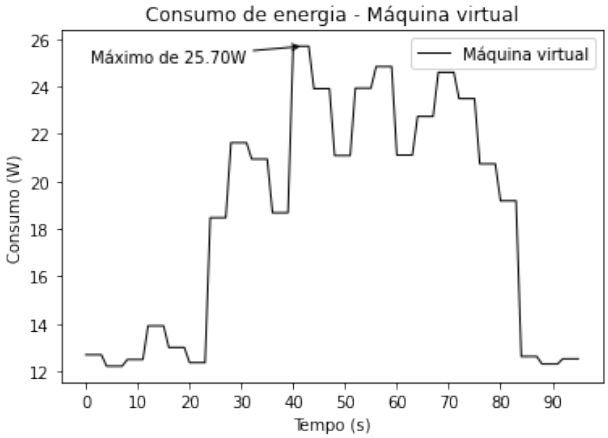
Analisando o canto superior direito da figura 4.11, é possível verificar que o teste simulou cem usuários acessando a página de inferência simultaneamente. Todos os usuários simulados executam os passos de carregar o site, enviar uma imagem pré-definida para a inferência e recebe a resposta nos demais quatro passos. Todos os passos são evidenciados no menu do canto esquerdo da figura 4.11, mostrado pela seleção em azul.

Para a análise, foram então adicionadas as figuras 4.12a e 4.12b que mostram o novo consumo de energia da *Raspberry Pi* e da máquina virtual, respectivamente.

[1]Disponível em https://jmeter.apache.org/



(a) Consumo de energia da Raspberry Pi



(b) Consumo de energia da máquina virtual

Figura 4.12: Consumo de energia durante a fase de inferência dos dois dispositivos estudados

A figura 4.12a mostra uma imagem representando o gráfico de consumo de energia do dispositivo *Raspberry Pi*. Diferente da etapa de treinamento, nesse momento de inferência, o dispositivo *Raspberry Pi* utilizou apenas 3.35W de consumo médio, chegando a um valor máximo de 4.97W. Nessa imagem é possível verificar um comportamento diferente da imagem 4.8a, que mostra o consumo de energia durante a etapa de treinamento. Quando comparadas, é possível concluir que o procedimento de inferência demanda menos energia que o de treinamento.

Na mesma figura 4.12, no gráfico representado pela figura 4.12b, também é possível verificar o consumo de energia da máquina virtual na mesma fase de inferência. Com um máximo de 25.7W, a energia utilizada para servir o modelo treinado diferiu pouco da fase de treinamento, como mostra a figura 4.8b. Nesse momento, o gráfico apresentou aproximadamente o mesmo valor máximo de 26.65W da figura 4.8b mostrando que não houve muita diferença.

O objetivo de evidenciar a diferença entre o consumo de energia durante o procedimento de inferência foi concluída com as imagens comentadas logo acima. A grande diferença energética apresentada também nessa etapa traz para o leitor o entendimento de que o dispositivo *Raspberry Pi* pode também ser considerado como alternativa para servir o modelo treinado.

Capítulo 5

Conclusão

Em sua concepção, o trabalho tinha como objetivo principal criar um paralelismo, onde cada camada do modelo de aprendizado de máquina fosse atribuído para um dispositivo diferente. Porém, adentrando no mundo do Kubernetes e visualizando o seu crescimento e adoção, a proposta evoluiu para esse outro caminho. Instigado pela grande discussão do tema e pouco material disponível, foi desencadeado um grande interesse pelo assunto.

Em um cenário onde a conteinerização faz parte da arquitetura da maioria das empresas, levando em consideração também o grande uso do aprendizado de máquina nas mais vastas áreas de conhecimento, a motivação em desenvolver ferramentas para unir os dois mundos faz-se de extrema necessidade. Dessa maneira, haverá uma maior compatibilidade entre os diferentes métodos de implementação e, em um futuro próximo, surgirão mais recursos para atender as demandas.

A ferramenta estudada traz consigo um grande aparato para atender diferentes métodos de aprendizado de máquinas, e um acervo de recursos que facilita o dia a dia do usuário pois centraliza os ambientes de desenvolvimento, teste e produção. Pronta para atender todos as etapas de um teste, como descrito na seção 2.4.2, é ofertada uma facilidade nunca antes oferecida por outra ferramenta, o que se torna diferencial não somente para um usuário iniciante em aprendizado de máquinas e Kubernetes, mas também para os mais avançados.

O conjunto de soluções apresentado nesse projeto mostra-se compatível com dispositivos de arquitetura *ARM* e também *amd*. Encontrar soluções que fossem o mais abrangente possível, de código aberto e gratuito, foram itens sempre considerados desde o início. Dessa maneira, a adoção e o estudo podem ser facilitados, proporcionando ao usuário final um arranjo de soluções compatível e recomendado.

Apesar de possuir um caráter meramente expositivo e acadêmico, o tema é valoroso e chama atenção de inúmeros profissionais empolgados com o futuro promissor que a arquitetura *ARM* está proporcionando para a tecnologia. O processamento acessível aliado com o baixo consumo de energia traz vantagens econômicas que podem viabilizar o acesso para um maior número de pessoas de forma mais rápida. É pensado em cenários críticos onde economia de energia e menores investimentos são mais valorosos que execuções mais rápidas.

Como foi possível verificar nas análises dos resultados encontrados, o dispositivo *Raspberry Pi* não deve ser entendido como uma substituição das máquinas virtuais, físicas ou estruturada em *cloud*. Deve-se entender a proposta como um dispositivo em crescente adoção por possuir um potencial enorme e ser objeto de pesquisa de grandes peças do mercado. O baixo consumo de energia e o seu valor mais acessível são pontos importantes levados em consideração pelos usuários interessados em economizar recursos.

Em um *cluster* com o recurso de um acelerador gráfico disponível, é válido adicionar as *Raspberry Pi* e dedicá-las somente a inferência do resultado. Por ser a etapa duradoura do aprendizado de máquinas, a que de fato irá tornar possível a utilização do modelo treinado, quando hospedada em um dispositivo de menor consumo energético, trará o benefício de economia de energia.

De incentivo com a sustentabilidade, todos os recursos possíveis devem ser economizados e substituídos por alternativas mais econômicas e que gerem um menor nível de degradação ao ambiente. O estudo traz consigo essa alternativa, mostrando que, sem perda alguma, é possível dar início ao pensamento ecológico adicionando dispositivos de baixo custo em sua arquitetura de aprendizado de máquinas.

No mais, concluo que a ferramenta estudada aborda itens necessários que facilitam a rotina de um desenvolvedor dentro da área do aprendizado de máquinas. Também é possível concluir que o uso do conjunto ferramental descrito no decorrer do documento traz uma facilidade maior e, consequentemente, um menor tempo de adaptação por parte do usuários.

5.1 Trabalhos Futuros

Com base nos objetos de estudo utilizados e no desenvolvimento da solução para o projeto, é possível notar que o ferramental necessário para suportar dispositivos de arquitetura *ARM* ainda é escasso e, apesar de mostrar ser tema importante para diversas empresas e grupos de pesquisa, falta maturação e soluções viáveis.

Como sugestão de futuros trabalhos dentro da área tem-se o desenvolvimento de uma solução que melhor atenda a especificidade da ferramenta utilizada, em conjunto com o dispositivo escolhido, no quesito de recurso computacional demandado. Sendo desenvolvida uma versão mais leve, possuindo somente o básico, seria possível criar um *cluster* composto totalmente só por *Raspberrys*, o que hoje é impossível. Outro ponto também poderia ser a criação de uma ferramenta de provisionamento de disco embutido na própria ferramenta Kubeflow, que eliminasse a dependência de um ferramental externo e, consequentemente, o tempo de espera entre etapas de leitura e escrita de disco e etapa de provisionamento do mesmo.

Referências

AZUAJE, Francisco; WITTEN, Ian; E, Frank. Witten IH, Frank E: Data Mining: Practical Machine Learning Tools and Techniques. **Biomedical Engineering Online**, v. 5, p. 1–51, set. 2006. ISSN 1475-925X. DOI: <10.1186/1475-925X-5-51>.

BESIMI, Nuhi et al. Using distributed raspberry PIs to enable low-cost energy-efficient machine learning algorithms for scientific articles recommendation. **Microprocessors and Microsystems**, v. 78, p. 9, 2020. ISSN 0141-9331. DOI: <10.1016/j.micpro.2020.103252>. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0141933120304130>>.

CARRARA, A.; FERREIRA, G. A Economia em Revista. **Periodicos UEM**, p. 73–90, 2020. ISSN 751375151548. Disponível em: <<http://periodicos.uem.br/ojs/index.php/EconRev/article/view/55442/751375151548>>.

CASTIONE, R et al. Universidades Federais na Pandemia da Covid-19. **IPEA**, v. 1, p. 17–21, mar. 2021. ISSN 1415-4765. DOI: <10.38116/td2637>. Disponível em: <<https://www.ipea.gov.br/portal/images/stories/PDFs/TDs/210326_td_2637_web.pdf>>.

CHANDRASEKAR, Raghunath Raja et al. Power-Check: An Energy-Efficient Checkpointing Framework for HPC Clusters. **IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing**, p. 261–270, 2015. DOI: <10.1109/CCGrid.2015.169>. Disponível em: <<https://ieeexplore.ieee.org/document/7152492>>.

CIREŞAN, Dan Claudiu et al. Deep, Big, Simple Neural Nets for Handwritten Digit Recognition. **Neural Computation**, MIT Press - Journals, v. 22, n. 12, p. 3207–3220, dez. 2010. ISSN 0899-7667. DOI: <10.1162/NECO_a_00052>. Disponível em: <<https://ieeexplore.ieee.org/document/6797043>>.

CNCF. **Lightweight Kubernetes**. [S.l.: s.n.], abr. 2020. Published online. Downloaded 04.13.2021. Disponível em: <<https://k3s.io/>>.

_____. **Use of containers in Production jump 300 from our first survey**. [S.l.: s.n.], nov. 2020. Disponível em: <<https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf>>.

DAHL, G. E. et al. Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition. **IEEE Transactions on Audio, Speech, and Language Processing**, v. 20, n. 1, p. 30–42, 2012. DOI: <10.1109/TASL.2011.2134090>.

DEAN, Jeffrey et al. Large Scale Distributed Deep Networks. Edição: F. Pereira. Curran Associates, Inc., v. 25, p. 1223–1231, 2012. Disponível em: <<https://proceedings.neurips.cc/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf>>.

DEVANATHAN, Ram. Moore's Law: The Life of Gordon Moore, Silicon Valley's Quiet Revolutionary Arnold Thackray, David C. Brock, and Rachel Jones. **MRS Bulletin**, v. 41, n. 05, mai. 2016. ISSN 0883-7694. DOI: <10.1557/mrs.2016.107>. Disponível em: <<https://www.osti.gov/biblio/1253834>>.

DOCKER. [S.l.: s.n.], jan. 2020. Published online. Downloaded 13.01.2021. Disponível em: <<https://www.docker.com/company>>.

EL NAQA, Issam; MURPHY, Martin J. What Is Machine Learning? Edição: Issam El Naqa, Ruijiang Li e Martin J. Murphy. Springer International Publishing, Cham, p. 3–11, 2015. DOI: <10.1007/978-3-319-18305-3_1>. Disponível em: <<https://doi.org/10.1007/978-3-319-18305-3_1>>.

ESTEVES, Gabriel; LEA, Andrea; ALVES, Esther de Oliveira. Fadiga e Estresse como preditores do Burnout em Profissionais da Saúde. pt. **Revista Psicologia Organização e Trabalho**, scielopepsic, v. 19, p. 695–702, set. 2019. ISSN 1984-6657. Disponível em: <<http://pepsic.bvsalud.org/scielo.php?script=sci_arttext&pid=S1984-66572019000300008&nrm=iso>>.

FIORAVANZO, S. **Automating Jupyter Notebook Deployments to Kubeflow Pipelines with Kale**. [S.l.: s.n.], abr. 2019. Disponível em: <<https://medium.com/kubeflow/automating-jupyter-notebook-deployments-to-kubeflow-pipelines-with-kale-a4ede38bea1f>>.

GARIFULLIN, I. **Kubeflow Times Machine Learning**. [S.l.: s.n.], mar. 2020. Disponível em: <<https://medium.com/hydrosphere-io/train-and-deliver-machine-learning-models-to-production-with-a-single-button-push-a6f89dcb1bfb>>.

GOIN, Dana E.; RUDOLPH, Kara E.; AHERN, Jennifer. Predictors of firearm violence in urban communities: A machine-learning approach. **Health Place**, v. 51, p. 61–67, 2018. ISSN 1353-8292. DOI: <https://doi.org/10.1016/j.healthplace.2018.02.013>. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1353829217307621>>.

HOLLISTER, S. **NVIDIA is building new Arm CPUs again**. [S.l.: s.n.], abr. 2021. Published online. Downloaded 04.13.2021. Disponível em: <<https://www.theverge.com/circuitbreaker/2021/4/12/22380065/nvidia-grace-cpu-arm-data-center-ai-hpc>>.

KUBEFLOW; VASCONCELOS, R. **Kubeflow Deployment with kfctl k8s istio**. [S.l.: s.n.], abr. 2021. Published online. Downloaded 04.13.2021. Disponível em: <<https://www.kubeflow.org/docs/started/k8s/kfctl-k8s-istio/>>.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. **Nature**, v. 521, n. 7553, p. 436–444, mai. 2015. ISSN 1476-4687. DOI: <10.1038/nature14539>. Disponível em: <<https://doi.org/10.1038/nature14539>>.

LUSE, C.; YANG, S. **What is LongHorn?** [S.l.: s.n.], abr. 2021. Published online. Downloaded 04.08.2021. Disponível em: <<https://longhorn.io/docs/1.1.0/what-is-longhorn/>>.

MAO, Lei. **Data Parallelism vs Model Parallelism in Distributed Deep Learning Training**. [S.l.: s.n.], mai. 2019. Published online. Downloaded 04.01.2021. Disponível em: <<https://leimao.github.io/blog/Data-Parallelism-vs-Model-Paralelism/>>.

MCCANN, Bryan et al. **Learned in Translation: Contextualized Word Vectors**. [S.l.: s.n.], 2018. arXiv: <1708.00107> .

NORMILLE, D. China again boosts RD spending more 10. **Science Magazine**, 2020. DOI: <10.1126/science.abe5456>. Disponível em: <<https://www.sciencemag.org/news/2020/08/china-again-boosts-rd-spending-more-10>>.

PETERS, Matthew E. et al. **Deep contextualized word representations**. (S.l.: s.n.), 2018. arXiv: <1802.05365> .

PINTO, A. S. Scratch na aprendizagem da Matemática no 1 ciclo do ensino básico. **Universidade do Minho**, v. 1, p. 9, jun. 2010. Disponível em: <<https://repositorium.sdum.uminho.pt/bitstream/1822/14538/1/tese.pdf>>.

PORTER, J. **Apple says new Arm-based M1 chip offers the longest battery life ever in a Mac**. [S.l.: s.n.], nov. 2020. Disponível em: <<https://www.theverge.com/2020/11/10/21558095/apple-silicon-m1-chip-arm-macs-soc-charge-power-efficiency-mobile-processor>>.

R., Rathmann et al. Evolução dos Investimentos em Pesquisa e Desenvolvimento e o Registro de Patentes. **ANPAD**, p. 1–14, 2006. ISSN DCT53. Disponível em: <<http://www.anpad.org.br/admin/pdf/DCT53.pdf>>.

RANCHER. **K3s Architecture**. [S.l.: s.n.], abr. 2021. Published online. Downloaded 04.13.2021. Disponível em: <<https://rancher.com/docs/k3s/latest/en/architecture/>>.

_____. **Running on ARM**. [S.l.: s.n.], abr. 2021. Published online. Downloaded 04.01.2021. Disponível em: <<https://rancher.com/docs/rancher/v2.x/en/installation/resources/advanced/arm64-platform/>>.

SICULAR, Svetlana. **Gartner's Big Data Definition Consists of Three Parts, Not to Be Confused with Three "V"s**. Mar. 2013. Disponível em: <<http://www.forbes.com/sites/gartnergroup/2013/03/27/gartners-big-data-definition-consists-of-three-parts-not-to-be-confused-with-three-vs/>>.

SIMOES, P. **Componentes do Kubernetes**. [S.l.: s.n.], abr. 2021. Published online. Downloaded 04.13.2021. Disponível em: <<https://kubernetes.io/pt/docs/concepts/overview/components/>>.

SZEGEDY, Christian et al. Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], jun. 2015. p. 1–9. DOI: <10.1109/CVPR.2015.7298594>.

UPTON, Eben et al. **Learning Computer Architecture with Raspberry Pi**. 1st. [S.l.]: Wiley Publishing, 2016. ISBN 1119183936.

WARREN, T. **Microsoft reportedly designing its own ARM-based chips for servers and surface pcs**. [S.l.: s.n.], dez. 2020. Published online. Downloaded 04.13.2021. Disponível em: <<https://www.theverge.com/2020/12/18/22189450/microsoft-arm-processors-chips-servers-surface-report>>.

XIN, Y. et al. Machine Learning and Deep Learning Methods for Cybersecurity. **IEEE Access**, v. 6, p. 35365–35381, 2018. DOI: <10.1109/ACCESS.2018.2836950>.

ZHANG, Sixin; CHOROMANSKA, Anna; LECUN, Yann. **Deep learning with Elastic Averaging SGD**. [S.l.: s.n.], 2015. arXiv: <1412.6651> .

ANEXOS

ANEXO I

MNIST Model - Handwritten digit classification

```
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD

def load_dataset():
  (trainX, trainY), (testX, testY) = mnist.load_data()
  trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
  testX = testX.reshape((testX.shape[0], 28, 28, 1))
  trainY = to_categorical(trainY)
  testY = to_categorical(testY)
  return trainX, trainY, testX, testY

def prep_pixels(train, test):
  train_norm = train.astype('float32')
  test_norm = test.astype('float32')
  train_norm = train_norm / 255.0
  test_norm = test_norm / 255.0
  return train_norm, test_norm

def define_model():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform'
    , input_shape=(28, 28, 1)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'
    ))
  model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'
    ))
```

```
31    model.add(MaxPooling2D((2, 2)))
32    model.add(Flatten())
33    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
34    model.add(Dense(10, activation='softmax'))
35    opt = SGD(lr=0.01, momentum=0.9)
36    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['
        accuracy'])
37    return model
38
39  def run_test_harness():
40    trainX, trainY, testX, testY = load_dataset()
41    trainX, testX = prep_pixels(trainX, testX)
42    model = define_model()
43    model.fit(trainX, trainY, epochs=10, batch_size=32, verbose=0)
44    model.save('final_model.h5')
45
46  # entry point, run the test harness
47  run_test_harness()
```

Listing I.1: MNIST Model - Handwritten digit classification

ANEXO II

Dockerfile

```
FROM ubuntu:18.04

RUN dpkg --add-architecture armhf && dpkg --add-architecture arm64 \
    && apt-get update && apt-get install -y \
        openjdk-11-jdk automake autoconf libpng-dev \
        curl zip unzip libtool swig zlib1g-dev pkg-config git wget xz-utils \
        python3-numpy python3-pip python3-mock python-pip python-setuptools build-essential python-dev \
        libpython3-dev libpython3-all-dev \
        libpython3-dev:armhf libpython3-all-dev:armhf \
        libpython3-dev:arm64 libpython3-all-dev:arm64 g++ gcc

RUN apt install software-properties-common -y
RUN add-apt-repository ppa:deadsnakes/ppa
RUN apt install python3.7 -y
RUN /bin/bash -c "rm /usr/bin/pip /usr/bin/python /usr/bin/python3"
RUN ln -s /usr/bin/pip3 /usr/bin/pip
RUN ln -s /usr/bin/python3.7 /usr/bin/python
RUN ln -s /usr/bin/python3.7 /usr/bin/python3

RUN pip3 install -U --user keras_applications==1.0.8 --no-deps \
    && pip3 install -U --user keras_preprocessing==1.1.0 --no-deps \

RUN pip3 install -U --user docs\tring_parser \
    && pip3 install deprecated

RUN /bin/bash -c "update-alternatives --install /usr/bin/python python /usr/bin/python3 150"

RUN /bin/bash -c "pip3 install --upgrade pip"

WORKDIR /root
RUN git clone https://github.com/lhelontra/tensorflow-on-arm/
WORKDIR /root/tensorflow-on-arm/build_tensorflow/
RUN git checkout v2.3.0
CMD ["/bin/bash"]
```

```
35
36 WORKDIR /root
37 RUN wget https://files.pythonhosted.org/packages/5c/13/0
       d298ac709aecdc2e52806de70071ad71c5d3f16bbeaeafab3a0987b28c3/kfp-1.4.0.tar.gz
38 RUN tar -xvzf kfp-1.4.0.tar.gz
39 WORKDIR /root/kfp-1.4.0/
40 RUN python3 setup.py install
41
42 WORKDIR /root
43 RUN wget https://files.pythonhosted.org/packages/b6/00/16228
       a3dd048a06771a18caa2c9189beef543d5b2d84be8590e830458d30/kale-0.2.4.tar.gz
44 RUN tar -xvzf kale-0.2.4.tar.gz
45 WORKDIR /root/kale-0.2.4/
46 RUN python3 setup.py install
47 RUN /bin/bash -c "python3 --version"
48 RUN /bin/bash -c "python --version"
49
50 WORKDIR /root
51 RUN git clone https://github.com/gbrlins/edit-kubeflow/
52 WORKDIR /root/edit-kubeflow/kubeflow-kale-0.6.0/
53 RUN python3 setup.py install
```

Listing II.1: Dockerfile com a instalação de componentes para gerar a imagem compatível com *arm*