UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA - CCET
DEPARTAMENTO DE ENGENHARIA ELÉTRICA - DEE

Fabricio Vellone

Um estudo sobre consumo de energia em aplicações JavaScript: Identificação dos custos energéticos para a etapa de Integração Continua

SÃO CARLOS - SP
2023

Fabricio Vellone

Um estudo sobre consumo de energia em aplicações JavaScript: Identificação dos custos energéticos para a etapa de Integração Continua

Trabalho de conclusão de curso apresentada ao Departamento de Engenharia Elétrica da Universidade Federal de São Carlos, para obtenção do titulo de bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. André Takeshi Endo

SÃO CARLOS - SP
2023

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Cienêcias Exatas e de Tecnologia - CCET
Programa de Graduação em Engenharia Elétrica

Comissão avaliadora

Membros da comissão examinadora que avaliou e aprovou a Defesa do Trabalho de Conclusão de Curso do candidato Fabricio Vellone, realizada em 22/03/2023

Prof. Dr. André Takeshi Endo
Instituição: Universidade Federal de São Carlos

Prof. Dr. Helder Vinicius Avanço Galeti
Instituição: Universidade Federal de São Carlos

Prof. Dr. Auri Marcelo Rizzo Vincenzi
Instituição: Universidade Federal de São Carlos

Dedicatória

Dedico este trabalho à minha esposa Amanda, meus pais Carlos e Rosa, minha irmã Eloha e à todos meus amigos que participaram desta minha jornada.

AGRADECIMENTO

Agradeço primeiramente aos meus pais Carlos e Rosa, minha irmã Eloha, minhas avós e minhas tias, que sempre me apoiaram em realizar a graduação em Engenharia Elétrica na UFSCar, me dando todo o suporte e ajudando em todos os momentos durante a caminhada.

Agradeço a minha esposa Amanda, por todo o suporte e motivação para realização do Trabalho de Conclusão de Curso e suporte na reta final desta jornada.

Agradeço aos amigos da república (Ale, Komono e Ryan), pela convivência diária, ajuda com as atividades da graduação e parceria pelo tempo em que estivemos morando juntos.

Agradeço aos colegas da turma 016 de Engenharia Elétrica (Leonardo N., Komono, Gross, Ryan, Pedro K., Vinícius B. e Heitor), pelo suporte nas matérias, convivência nos trabalhos em grupo e parceria para a vida.

Agradeço ao Prof. Dr. André Takeshi Endo, por toda paciência e todo o suporte no desenvolvimento do trabalho.

Agradeço ao Departamento de Engenharia Elétrica, por todo os ensinamentos passados e papel na minha caminhada durante esta graduação.

Agradeço a CAPES por tornar possível o programa de dupla diplomação realizado durante a faculdade.

Também agradeço por todos que fizeram parte da minha caminhada durante a realização do curso.

Ninguém ignora tudo. Ninguém sabe tudo. Todos nós sabemos alguma coisa. Todos nós ignoramos alguma coisa. Por isso aprendemos sempre - Paulo Freire

RESUMO

Dado o contexto climático que o mundo se encontra, controlar o consumo de energia que os seres humanos realizam diariamente é indispensável para o bem-estar do planeta e da sustentabilidade. Para o *software* isso não é diferente. Este trabalho teve como objetivo analisar o consumo de energia no processo de Integração Contínua de aplicações JavaScript. Particularmente, estudou-se cada etapa do processo, identificando como se distribui o consumo de energia, traçando uma relação entre o tempo de execução das etapas e o consumo energético, em adição a realizar paralelos entre estes consumos no mundo do *software* e os consumos de energia do dia-a-dia das pessoas.

Primeiro selecionou-se a ferramenta de captura de consumo energético *perf*. Em seguida, foram selecionadas de projetos JavaScript para análise seguindo alguns critérios específicos, como a existência de diferentes testes e CI, além de padronizar todas as nomenclaturas existentes nesses projetos. Por fim, combinou-se a ferramenta escolhida, scripts para obter os dados brutos e realizou-se as análises por meio de planilhas.

Foi possível constatar que as etapas de testes foram as mais custosas dentre as etapas avaliadas. Para a relação entre o tempo e o consumo de energia, observou-se que os consumos energéticos tendem a serem lineares com o tempo para processamentos diretos e os não diretos tendem a apresentar um padrão logarítmico de comportamento. Ademais, viu-se que um projeto pode ter, em média, o mesmo consumo de energia que um secador de cabelo e pode consumir mais que um roteador ou um modem de Internet.

Por fim, foi identificado que em grandes escalas, estes consumos de energia podem ser significantes e se deve ter atenção com a quantidade de vezes que as *pipelines* de CI são executadas ao longo de um desenvolvimento de projeto. Também pode-se observar que quanto maior a complexidade de um tipo de testes, maior seu custo em termos de energia, indo de acordo com a pirâmide de testes.

Palavras-chave: *Green Computing*. Consumo de energia em software. Testes automatizados. Integração Contínua. *pipelines*. JavaScript. Node.js.

ABSTRACT

Given the climate context that the world finds itself in, controlling the energy consumption that humans perform on a daily basis is indispensable for the well-being of the planet and sustainability. For software this is no different. This work aimed to analyze the energy consumption in the Continuous Integration process of JavaScript applications. In particular, each step of the process was studied, identifying how energy consumption is distributed, tracing a relationship between the execution time of the steps and energy consumption, in addition to making parallels between these consumptions in the software world and the energy consumption of everyday life.

First the energy consumption capture tool perf was selected. Next, JavaScript projects were selected for analysis following some specific criteria, such as the existence of different tests and CI, as well as standardizing all existing nomenclatures in these projects. Finally, the chosen tool was combined with scripts to obtain the raw data and the analysis was performed using spreadsheets.

It was possible to verify that the testing stages were the most costly among the stages evaluated. For the relationship between time and energy consumption, it was observed that the energy consumption tends to be linear with time for direct processing, and the non-direct processing tends to present a logarithmic pattern of behavior. Furthermore, it was seen that a project can have, on average, the same energy consumption as a hair dryer and can consume more than a router or an Internet modem.

Finally, it was identified that at large scales, these energy consumptions can be significant and attention should be paid to the number of times IC pipelines are run throughout a project development. It can also be observed that the higher the complexity of a test type, the higher its cost in terms of energy, going according to the testing pyramid.

Keyword: Green Computing. Power consumption in software. Automated testing. Continuous Integration. pipelines. JavaScript. Node.js.

Lista de Figuras

Lista de Tabelas

Conteúdo

1 Introdução

Nos últimos anos, fica cada vez mais evidente o impacto do aquecimento global ao redor do mundo, desde situações de extremo calor em regiões antes nunca vistas ou de lugares que começam a deixar de serem habitáveis para o ser humano (Reis, 2022). Este grande crescimento nas temperaturas médias se dá por diversos fatores e muitas entidades ao longo do globo terrestre começam a tomar medidas em direção à diminuição ou redução total de gases poluentes, sendo um assunto que gerou e gera contribuição inter-fronteiras, unindo diversos países em diferentes acordos como por exemplo, o Acordo de Paris e o acordo de Glasgow (Dewan e Cassidy, 2021).

Seguindo este preceito, a Engenharia e a Ciência caminham juntas envolvendo diversos campos com um objetivo em comum: o desenvolvimento da humanidade cientificamente, tecnologicamente e a preservação do planeta. Neste contexto, assuntos e temas de pesquisas começam a aparecer com mais frequência no meio acadêmico da Engenharia Elétrica, como o aumento das pesquisas por desenvolvimento de energias renováveis e sustentáveis, a aplicação de redes inteligentes (*smart-grids*) para diminuição de consumo e o reaproveitamento de energia excedente (Coram e Katzner, 2018).

Neste mesmo cenário, o avanço das ferramentas e processos por meio de sistemas de *software* crescem demasiadamente; a migração de serviços que antes eram realizados manualmente se tornam digitais (Terra, 2022). Toda essa demanda gera uma inclinação contrária no consumo de energia relacionada a este tipo de tecnologia, e nos últimos anos a parcela de gastos relacionados à áreas de computação - como por exemplo, em *data centers* - só tem projeções de crescimento (Kurp, 2008).

Logo, vários estudos começaram a ser conduzidos para levantar medidas para contornar esses aumentos e aumentar a eficiência energética destes segmentos, desde soluções para melhorar a performance em sistemas de refrigeração de grandes centros de infraestrutura até o aumento da eficiência dos ciclos de vida de baterias de dispositivos portáteis (Rieger e Bockisch, 2017). Para isso, um avanço na tecnologia de como desenvolver e manter aplicações que são executadas em diferentes dispositivos e um aumento no conhecimento geral dos profissionais destas áreas se torna tão importante para o equilíbrio da parte física e lógica deste ecossistema.

Este trabalho faz um estudo referente à criação e manutenção de *software*, analisando a parte de integração contínua (ou *Continuos Integration - CI - em inglês*) de seu desenvolvimento. A CI é uma série de etapas e processos automatizados aplicados no dia-a-dia de trabalho de times de desenvolvimento. Neste ciclo, quando alguém verifica o código durante o processo de revisão em algum repositório (visto que todo código precisa ser mantido em um), um sistema automatizado analisa as mudanças e aplica essa série de etapas inspecionando todo o código para ver se as mudanças realizadas foram boas e não quebraram o sistema (Meyer, 2014). Assim, este trabalho mede a quantidade de energia

referente a este processo, estabelecendo um paralelo com quais etapas são mais custosas durante a fase de um projeto e entre outras comparações energéticas.

1.1 Justificativa

Desde 2008, alguns departamentos de energia já haviam notado um crescente aumento no consumo de energia na área da tecnologia na onde, por exemplo, o aumento esperado para o consumo em *data-centers* - local onde estão concentrados os sistemas computacionais de uma ou várias empresas - mostrava-se inclinado a aumentar em média 10% ao ano (Kurp, 2008). O gasto energético com desenvolvimento de software em ambientes remotos já acumulavam gastos de $7,4 bilhões de dólares nos EUA em 2011 (Kurp, 2008).

Hoje em dia, segundo uma matéria no *Deutshe Welle* (2022), esses mesmos negócios na Europa já consomem mais energia elétrica do que alguns países inteiros, como por exemplo Colômbia, Argentina e Egito, com um consumo médio de 250 Terawatt-hora por ano.

Manotas et al. (2016) mostram uma pesquisa feita com profissionais da área da TI de grandes empresas como ABB e Google, na onde muitas perguntas são referentes à relevância que o consumo energético tem dentro dos projetos. Novamente fica evidente que existe uma preocupação por parte dos profissionais nesse assunto, principalmente os que desenvolvem aplicações/serviços para dispositivos móveis, ainda que esse tópico seja pouco explorado e que exista uma falta de conhecimento sobre o assunto.

Após citado esses casos, foi estipulado como tema deste trabalho analisar o desenvolvimento de projetos para se ter uma visibilidade maior com relação ao consumo de certas etapas de um projeto, nesse caso analisando estritamente o processo de CI (Seção 2.3) do software, para obter-se mais informações sobre o assunto e conseguir adquirir mais visibilidade deste tema que cresce com o passar dos anos, dado que o consumo de energia tende a aumentar ao longo do tempo nesse setor (Kurp, 2008), assim sendo importante compreender cada vez mais sobre o assunto.

1.2 Objetivos

O objetivo deste trabalho foi analisar projetos que utilizam como base o JavaScript (Node.js), nos quais buscou-se medir o consumo de energia nas diversas etapas de Integração Contínua (CI). O intuito era verificar as maiores etapas de gasto energético, verificar a relação entre o consumo da energia e o tempo para estes projetos, além de realizar paralelos entre estes consumos no mundo do *software* com consumos de energia do dia-a-dia das pessoas.

1.3 Estrutura do Trabalho

Este trabalho está dividido na seguinte forma: o Capítulo 2 apresenta o conceito das ferramentas, expõe alguns dados científicos além de enunciar os trabalhos relacionados; o Capítulo 3 apresenta a metodologia utilizada para a condução do estudo; no Capítulo 4 tem-se a apresentação e análise dos resultados; por fim, no Capítulo 5 as conclusões a respeito do que foi estudado e levantado neste trabalho são apresentadas.

2 REVISÃO DE LITERATURA

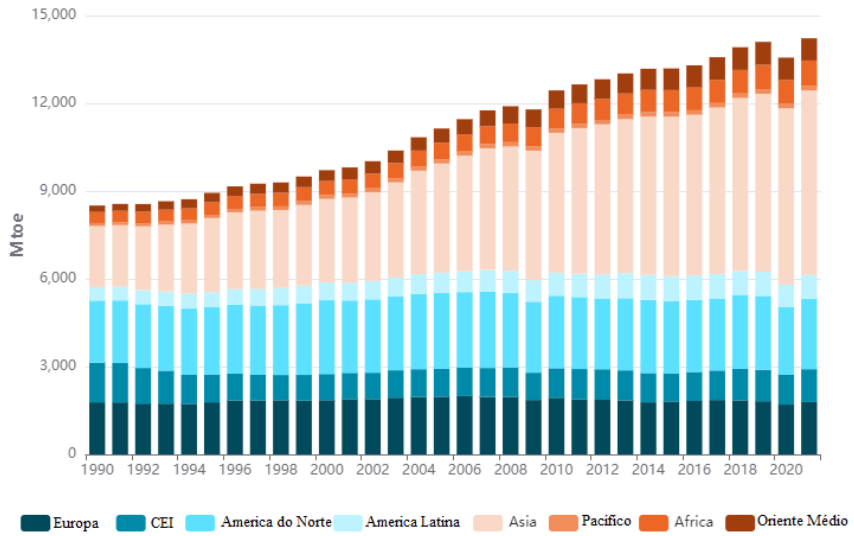
Neste capítulo são apresentadas as ferramentas para se compreender o trabalho em questão, além de introduzir conceitos chaves para o bom desenvolvimento do mesmo. Assim, o capítulo fica estruturado da seguinte forma: a Seção 2.1 aborda *Green Computing*; a Seção 2.2 descreve ferramentas de *software* que são utilizados para medir o consumo de energia por processos em computadores; a Seção 2.4 é dedicada para apresentar trabalhos relacionados à este projeto.

2.1 Green Computing e o consumo de energia em Software

Ao se falar de consumo de energia em software, um tema que se desponta é o de *Green Computing* (GC). De modo geral, GC é a prática relacionada à preocupação no desenvolvimento de ferramentas para T.I. (e seu uso) visando o seu impacto no Ambiente, além de sempre buscar a maior eficiência dos componentes/projetos desenvolvidos, ou seja, desenvolver sistemas mais robustos e rápidos usando menos recursos energéticos (Li e Zhou, 2011).

Como mencionado na Seção 1.1, o consumo de energia global aumenta drasticamente ano após ano, como é mostrado na Figura 1. Nesta, é possível identificar no eixo horizontal os anos em que o consumo foi medido, começando de 1990 até 2021 e no eixo vertical tem-se os valores em Mega Tonelada equivalente de petróleo (Mtoe) para o consumo de energia, onde 1 Mtoe equivale a 11,63 Terawatt-hora (TWh). Por fim, é possível identificar cores diferentes nas barras, apresentando a distribuição por macro regiões geográficas. Observa-se que os maiores consumidores de energia são a Asia e a Europa.

Figura 1: Consumo de energia em tonelada equivalente de petróleo de 1990 - 2021.



Fonte: Adaptado de https://yearbook.enerdata.net/total-energy/world-consumption-statistics.html

Segundo Ahmed, Bollen e Alvarez (2021), em meados de 2018 a demanda estimada anual de *data-centers* ao redor do mundo era de 205 TWh. Dado que *data-centers* são apenas um dos componentes da energia deste setor, estima-se que os gastos gerais sejam ainda maiores, o que confirma a relevância desse seguimento na parcela de consumo global de energia e da importância de entendê-la.

Li e Zhou (2011) dizem que para medir o consumo de um sistema computacional, tem-se que pensar no sistema como um todo: desde o consumo do hardware, até se estimar o gasto dos *softwares* - buscando medir o gasto de cada processo rodando dentro do Sistema Operacional (SO). Logo, existem vários modos de se calcular e avaliar este consumo. Por exemplo, pode-se usar ferramentas de medição de tensão e corrente na saída desses equipamentos, com o uso de multímetros e amperímetros especializados para este tipo de medição, para captar o consumo da máquina como um todo. Outra possibilidade é de se estimar via teoria computacional, o que acaba deixando algumas lacunas e não sendo tão exata, visto que se baseia em estimar os ciclos de *clock* de uma CPU e sua tensão para realizar determinadas tarefas. E o último caminho mais comum, e que será o foco deste trabalho, é utilizar ferramentas computacionais que medem a energia consumida por processos ou mesmo por partes do SO.

2.2 Ferramentas para capturar consumo de energia em software

Quando o assunto é consumo de energia, normalmente encontra-se nos aparelhos eletroeletrônicos uma especificação, contendo o consumo por hora deste aparelho em questão, medidos em Quilowatt-hora (kWh), sendo esta uma unidade de medida muito comum no meio comercial - outra unidade de consumo de energia muito usada na literatura é a unidade Joules (J).

Porém, ao abordar a energia consumida pelo *software* o desafio é maior, visto que o valor da potência discriminada na fonte de um computador é o valor máximo fornecido pela mesma e não reflete apenas o consumo das aplicações em si, ela também estará alimentando todo o *hardware* necessário para o computador funcionar. É por isso que muitas empresas fabricantes de processadores, como a *Intel* e a *AMD*, ou fabricantes de placa de vídeo, como a *Nvidia*, possuem ferramentas para medir este consumo de energia para usuários, de uma forma abstraída e isolada, sendo assim de mais fácil compreensão.

Essas ferramentas são disponibilizadas em geral como Interface de Linha de Comando (do inglês: *command-line interface* - CLI). Muitas vezes se baseiam no processador usado, e computa a quantidade de ciclos de *clock* realizados pelas unidades de processamento, para conseguirem gerar uma relação entre o tempo e o gasto energético. Assim, tendo isso em vista, têm-se duas classes de software que medem o consumo de energia: as ferramentas que gerenciam o consumo global da máquina; e as ferramentas de consumo por processo isolado (Cruz, 2021). Vale ressaltar que, a quantidade de processos sendo executados

na máquina impacta diretamente no valor final medido, tal qual o SO utilizado e outros fatores.

Segundo a lista de ferramentas apresentadas por Cruz (2021), para as ferramentas analisadas de consumo geral normalmente se tem um número de amostras recolhidas, um tempo entre cada amostra, e assim obtém-se um relatório da máquina, enunciando seu consumo global.

Na Figura 2 tem-se um exemplo deste relatório. Aqui foi escolhido realizar 10 capturas do sistema, então é possível verificar em cada uma das primeiras linhas antes da divisão as informações referentes à estas amostras. Cada coluna contém uma informação sobre o SO, onde a primeira e a última são as mais importantes para a análise deste trabalho, sendo a primeira o horário de começo da medição e a última o valor em Watts da potência média desta amostra do sistema. É possível identificar que o programa dá as informações sobre os valores médios, mínimos e máximos dessas amostras. Assim, é possível observar que o consumo geral (apenas da CPU) do computador utilizado no estudo é de 6,37 Wh sem somar outros periféricos do sistema.

Figura 2: Exemplo de saída do Powerstat.

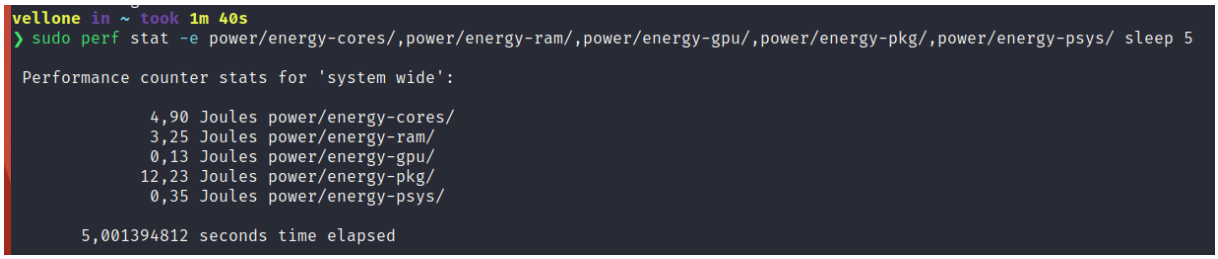
```
vellone in ~ took 16s
❯ sudo powerstat -R 10 10
Running for 100.0 seconds (10 samples at 10.0 second intervals).
Power measurements will start in 0 seconds time.

  Time    User  Nice   Sys  Idle    IO  Run Ctxt/s  IRQ/s Fork Exec Exit  Watts
19:50:57   2.2   0.0   0.7  97.1   0.1    2   2536    938    2    0   25   5.31
19:51:07   2.4   0.0   0.8  96.7   0.1    1   2821   1042   22    0    3   5.96
19:51:17   2.1   0.0   0.7  97.2   0.1    1   2494    864    2    0    1   5.24
19:51:27   2.0   0.0   0.7  97.3   0.1    2   2535    869    0    0    2   5.34
19:51:37   2.1   0.0   0.6  97.1   0.1    1   2500    904    1    0   23   5.04
19:51:47   1.9   0.0   0.5  97.5   0.1    1   2307    731    1    0    0   4.61
19:51:57   2.3   0.0   0.7  96.9   0.1    1   2481    895    1    0    2   5.04
19:52:07   2.6   0.0   0.7  96.6   0.1    1   2959   1012   35    0    3   6.09
19:52:17   6.2   0.0   1.3  92.4   0.2    2   8083   2527   77    0   31  14.59
19:52:27   3.2   0.0   0.8  95.7   0.3    1   4080   1227    8    0   25   6.48
-------- ----- ----- ----- ----- ----- ---- ------ ------ ---- ---- ---- ------
 Average   2.7   0.0   0.7  96.4   0.1  1.3 3279.7 1100.8 14.9  0.0 11.5   6.37
 GeoMean   2.5   0.0   0.7  96.4   0.1  1.2 3020.4 1031.3  0.0  0.0  0.0   5.99
  StdDev   1.2   0.0   0.2   1.4   0.1  0.5 1671.6  491.6 23.4  0.0 12.0   2.79
-------- ----- ----- ----- ----- ----- ---- ------ ------ ---- ---- ---- ------
 Minimum   1.9   0.0   0.5  92.4   0.1  1.0 2307.4  730.9  0.0  0.0  0.0   4.61
 Maximum   6.2   0.0   1.3  97.5   0.3  2.0 8083.3 2527.4 77.0  0.0 31.0  14.59
-------- ----- ----- ----- ----- ----- ---- ------ ------ ---- ---- ---- ------
Summary:
CPU:   6.37 Watts on average with standard deviation 2.79
Note: power read from RAPL domains: package-0, uncore, package-0, dram, core, dram, psys.
These readings do not cover all the hardware in this device.
```

Fonte: Autoria própria.

A outra classe de ferramentas que analisa o consumo isolado por processo, dando seu relatório baseado em um comando a ser executado (como por exemplo executar uma aplicação web) tendo um relatório como na Figura 3. Neste exemplo, é possível identificar o comando usado para se iniciar a captura, onde selecionou-se todo o pacote disponível da ferramenta Perf relacionado à energia - medindo o consumo de energia dos núcleos,

placa de vídeo, sistema e aplicações. A medição foi feita em cima do comando "*sleep 5*" que realiza uma pausa no sistema por 5 segundos. E como saída da medição, obteu-se o consumo de energia em Joule para cada um dos módulos escolhidos e o tempo total de execução do comando.

Figura 3: Exemplo de saída do Perf.



Fonte: Autoria própria.

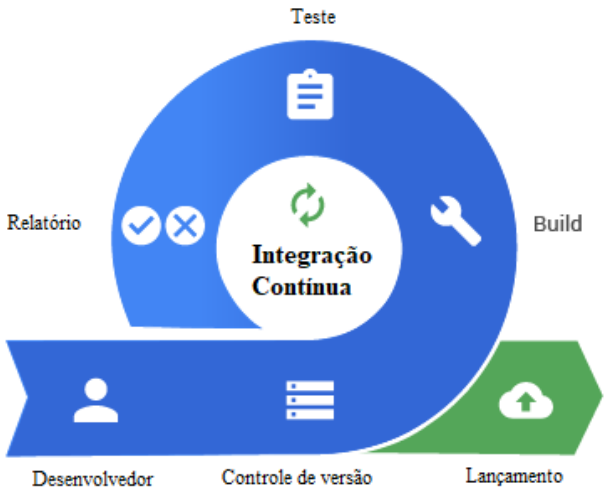
Este será o tipo de ferramenta escolhida para ser o enfoque deste trabalho, onde se tem uma seção abordando mais profundamente a decisão de escolha das ferramentas (Seção 3.1). A ferramenta Perf foi escolhida dado a sua versatilidade de se concatenar comandos no terminal, aceitando até scripts inteiros, facilitando as capturas necessárias nesse estudo.

2.3 Integração Continua (CI)

Integração Contínua é o nome dado para uma das práticas chave do desenvolvimento de *software*, onde adicionam-se automatizações ao processo de desenvolvimento do mesmo como a etapa de montagem de um projeto (*build*), checagem de padrões de código, testes automatizados e outras verificações julgadas necessárias pela equipe. Estes passos são executados sempre que uma nova alteração é adicionada por um desenvolvedor garantindo maior qualidade, aumento de produtividade e ajudando na diminuição de lançamentos de novas versões de projetos (Shahin, Babar e Zhu, 2017).

Na Figura 4, tem-se uma representação visual deste processo, onde é possível identificar as etapas pertencentes a esta prática. Nela têm-se o desenvolvedor como primeiro passo do ciclo, e o lançamento/publicação do *software* como o final. Dentro do *looping* é possível encontrar a parte de repetição deste processo, que consiste em utilizar o controle de versão para a publicação de novas partes de um projeto, realiza-se a fase de "*build*"do mesmo, executa-se os testes e outras verificações e por fim se gera os relatórios, buscando certificar-se que todas as etapas deste ciclo foram bem sucedidas, possibilitando assim o prosseguimento para a publicação. Caso se haja falhas ou uma nova adição de conteúdo, o processo deve ser repetido, daí a ideia da continuidade do processo.

Figura 4: Ciclo de uma Integração Contínua.



Fonte: https://cloud.google.com/solutions/continuous-integration/images/hero-banner.png?hl=pt-br

Uma ferramenta importante que quase sempre está presente nesta etapa, são ferramentas de contêineres. Um contêiner é um ambiente isolado dentro de uma máquina, que contém um conjunto de processos que são executados a partir de uma imagem base com todos os arquivos necessários, compartilhando o mesmo *kernel* que o sistema operacional (SO) da máquina usada, porém com os processos isolados da mesma (14). Assim, se torna possível desenvolver aplicações e executá-las em outros ambientes e mesmo separar dependências entre projetos, ferramentas de testes, entre outras coisas do seu SO base, tornando o ambiente de trabalho isolado. Umas das ferramentas mais conhecidas e usadas para isto é o *Docker*, e foi essa a ferramenta presente em alguns dos projetos escolhidos para análise.

2.4 Trabalhos Relacionados

Amsel e Tomlinson (2010) expõem um caso de criação de uma aplicação visando medir o consumo de *softwares* de um dado dispositivo. A ferramenta em questão se chama "Green Tracker" e foi criado pensando em evidenciar os programas que são mais efetivos para determinada função (Ex.: Editar textos) ao fazer uma comparação de porcentagem de ocupação da CPU (visto que quanto mais uma aplicação consome recursos computacionais maior seu consumo energético). Segundo os autores, o objetivo do seu projeto é fazer com que os usuários escolham programas que são mais conscientes com relação ao Ambiente. Para isso, sua criação faz um *benchmark* do SO em si, bem como das ferramentas escolhidas e trás um relatório para o usuário poder melhor escolher a opção mais adequada, conseguindo apontar gráficos com relação à esta comparação. Como resultados, os autores constataram que a ferramenta proposta consegue apontar diferenças

entre aplicações da mesma categoria (diferenciar o consumo entre diversos navegadores, ou *players* de música), ajudando a evidenciar para o usuário quais aplicações consomem menos energia.

Para Schubert et al. (2012), os desafios quando se tratam de consumo de energia em *software* são mais difíceis. Dado que existem operações mais diretas, como os processos do SO e acessos á memórias, e outros processos mais complicados, como o uso de placas de rede, capturar o consumo de energia desses diferentes itens se torna um desafio. Isto posto, os autores também propuseram um projeto de ferramenta, chamado de "*eprof*" - *energy profile*. Esta ferramenta é bem mais robusta do que a proposta por Amsel e Tomlinson (2010). Ela conta com vários sistemas de seleção de consumo, desde chamadas do sistema (*syscall*), capturas em *drivers* e configurações feitas à nível de Kernel de um SO. Assim, eles conseguem somar os consumos de *hardware* aos dos processos computacionais (CPU) e apontar determinados gargalos no sistema. Como resultados, foi possível evidenciar diferenças entre arquiteturas de computadores e estratégias de performance em dispositivos diferentes, conseguindo constatar ganhos significativos de economia de energia ao, por exemplo, limitar a quantidade de I/O de um disco e aliviar à carga de uma CPU durante o processamento de dados.

Pang et al. (2016) diz que pesquisadores fizeram uma pesquisa com o objetivo de avaliar o conhecimento de programadores referente ao consumo de energia durante o desenvolvimento de *software*. O levantamento de dados foi feito de forma anônima, e com perguntas desde o nível do entrevistado em programação e sua linguagem de domínio até quais periféricos ou partes de um desenvolvimento mais consomem energia. A pesquisa se concentrou em desenvolvedores *desktop* e mobile. O resultado mostra novamente um grande desfoque das pessoas para o consumo de energia de um *software*, onde muitas vezes, segundo a pesquisa, isso acaba sendo mais relacionado ao *hardware* do que o trabalho computacional em si, segundo os entrevistados. O descaso é ainda maior, onde apenas 1% dos participantes da categoria *desktop* acertaram os periféricos que mais consomem energia em um PC. Os criadores de aplicativos para dispositivos móveis, por sua vez, alegaram ser mais uma falta de prioridade, visto que, segundo eles, a parte física acaba gerando mais problemas nesse tema.

Souza et al. (2020) aprofundam-se na avaliação do consumo de energia de *software*. Os autores também propõem uma nova ferramenta, chamada "Containergy" que, como seu nome expõe, usa de contêineres para abstrair e isolar processos durante a medição do consumo. A combinação de grupos de controle (cgroup) e contêineres gera um isolamento de processo (do ponto de vista de SO) e torna possível o uso de registradores específicos para extrair dados crus de performance do *hardware* e em paralelo os dados da aplicação são coletados via log do contêiner. Garantindo esse nível de isolamento, e recolhendo tanto informações de *hardware* e *software*, é possível avaliar com mais precisão cada processo. A ferramenta ainda tem a capacidade de lançar diversas configurações de sistema

e performa a ação várias vezes, baseadas em um limite mínimo de métricas de qualidade para o processo. Esses resultados foram demonstrados em dois experimentos, realizando o *encoding* de um vídeo usando HEVC e classificando imagens utilizando diferentes técnicas de *Machine Learning*. Em ambos os casos, foi possível capturar grandes diferenças no consumo de energia apenas escolhendo melhores configurações, como por exemplo no caso de codificação de vídeo teve-se uma diferença de 300% entre a pior e a melhor configuração; e para a classificação de imagens, trocando algoritmos de ML, obteve-se uma economia de até 55% no consumo de energia.

2.5 Considerações Finais

Assim, foi possível identificar que existe sempre um grande interesse em medir e saber mais sobre o consumo de energia em diversas áreas do desenvolvimento de *software*. Desde as pesquisas realizadas com profissionais da área, até as tentativas de criação de ferramentas para monitorar esses gastos. Assim, este estudo seguiu o mesmo enfoque, se dedicando a medir o consumo de energia durante o desenvolvimento de *softwares*, analisando mais especificamente o consumo referente ao processo de CI de um projeto, tentando assim compreender os perfis energéticos de cada etapa deste processo. No capítulo seguinte, foi abordado a metodologia seguida pelo estudo, mostrando nos caminhos percorridos.

3 METODOLOGIA DO ESTUDO

Neste capítulo foi apresentado em detalhes sobre o passo-a-passo da realização do estudo proposto. Como o objetivo deste trabalho foi estudar o consumo de energia dentro do contexto das etapas de Integração Contínua em projetos Node.js, foram formuladas as seguintes Questões de Pesquisa (QP):

- **QP1 - Como é o consumo de energia nas etapas de CI em projetos Node.js?**: o intuito desta questão é identificar como que acontece a distribuição do consumo de energia nas várias etapas da CI;
- **QP2 - Qual é a relação entre o consumo de energia e o tempo de execução nas etapas de CI?**: a proposta desta questão é identificar se existe uma relação entre o tempo de execução e o consumo real por estas etapas, dado que estudos da literatura apontam o tempo de execução como o fator determinante para o consumo como um todo;
- **QP3 - Como a perspectiva do consumo de energia em CI se relaciona com o consumo no dia-a-dia?**: nesta questão buscou-se fazer um paralelo entre o consumo de energia em CI e o dia-a-dia das pessoas para trazer esses valores para dados mais concretos e palpáveis para a população de maneira geral

Desta forma, o capítulo fica estruturado da seguinte maneira: a Seção 3.1 mostra o processo da escolha da ferramenta de captura de consumo de energia, na Seção 3.2 apresentou-se as configurações da máquina utilizada e as linguagens escolhidas para o desenvolvimento do trabalho, na Seção 3.3 detalhou-se sobre o processo de escolha dos projetos; a Seção 3.4 descreve a criação dos *scripts* utilizados no projeto e nas Seções 3.5 e 3.6 foram descritos, respectivamente, os processos de padronização das etapas da CI e como se deu a análise dos dados durante o estudo. Por fim, na Seção 3.7 descreveu-se possíveis ameaças à validade do trabalho proposto.

3.1 Seleção da ferramenta para capturar consumo de energia

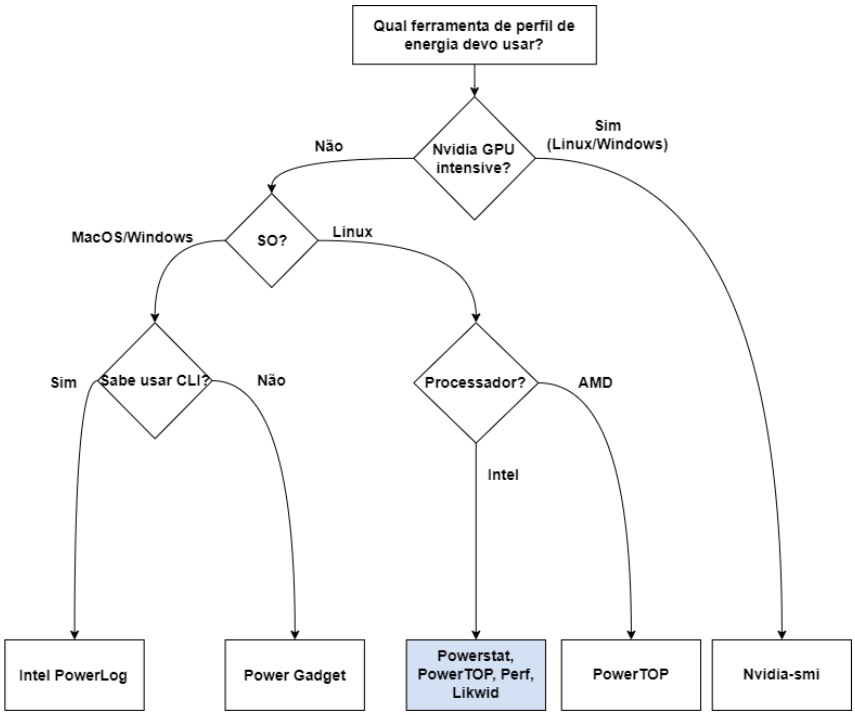
Como um passo inicial do desenvolvimento deste projeto, buscou-se estudar e entender alguns casos de uso das ferramentas citadas por Cruz (2021). O autor discuti diversas aplicações usadas para capturar o consumo de energia de uma máquina, via software, baseando-se em SDKs fornecidas pelos processadores ou placas gráficas. São apresentadas as seguintes ferramentas: *Intel Power Gadget, Intel PowerLog, Powerstat, PowerTOP, Perf, Likwid, Nvidia-smi.*

Cada ferramenta citada tem sua plataforma recomendada de uso. Por exemplo, o PowerLog é mais amplamente usado em máquinas com S.O. MacOS e máquinas sem

placa gráfica com chips AMD conseguem apenas fazer o uso do PowerTOP. O Linux é o S.O. mais versátil, suportando praticamente todas as ferramentas. Logo, este foi o sistema escolhido para o desenvolvimento do projeto.

Cruz (2021) ainda aponta um diagrama (Figura 5) para ajudar com a escolha da ferramenta, de acordo com o S.O. mais indicado para esta. Sendo assim, foram selecionadas as ferramentas: Powerstat (Canonical, 2011), PowerTOP (Ven, 2007), Perf (Kerrisk, 2009) e Likwid (NERSC, 2015). Para todas as ferramentas, procurou-se ler sobre a documentação da mesma e verificar suas aplicações em quesitos de execução de código, argumentos disponíveis para uso e outras funcionalidades mais avançadas.

Figura 5: *Flowchart* guia para escolha da ferramenta



Fonte: Adaptação de http://luiscruz.github.io/2021/07/20/measuring-energy.html

Dentre essas quatro ferramentas testadas, elas se dividem em duas categorias: captura global de performance e captura específica. Com as ferramentas Powerstat e PowerTOP, pertencentes ao grupo de captura global, é possível verificar saídas de performance de forma geral do computador durante períodos de tempo específicos. Então foi possível verificar quantidade de ciclos do processador, frequência média, consumo de energia entre outras informações de todos os processos como um todo, mostrando os dados relativos à máquina no momento de execução dos comandos destas duas ferramentas. Já para o Perf e Likwid, se tem uma captura específica por comando executado, por exemplo, ao se executar uma instalação de dependências, consegue-se verificar informações de performance da máquina apenas para esse comando em específico.

Sendo assim, depois de testes e segundo as documentações lidas, a ferramenta Perf

foi a escolhida para capturar os dados apresentados neste trabalho. Combinando esta ferramenta com arquivos de Script, é possível ter informações do consumo de energia para uma sequência de processos, conseguindo agregar mais de uma tarefa ao mesmo comando.

Uma observação importante quanto a essas ferramentas, e mais assertivamente com relação ao Perf, estas ferramentas não trazem informações com relação ao número de chamadas de placa de rede, acessos à discos ou outros tipos de trocas/consultas realizadas pelo sistema que também geram gastos energéticos. Para conseguir este nível de informação, segundo Schubert et al. (2012) e Souza et al. (2020), é necessário um nível de modificação à nível de Kernel para se conseguir gerenciar estas pequenas interações entre os dispositivos durante várias etapas dos processos. Ferramentas mais robustas e complexas como a citada por Souza et al. (2020) são aplicadas ainda em nível experimental e teórico.

3.2 Ferramentas utilizadas

Após estudo das aplicações disponíveis para captura da níveis de consumo de energia, os meios utilizados de modo geral para a realização deste trabalho foram baseadas nas seguintes ferramentas:

- Notebook: Dell G5-5590, processador Intel(R) Core(TM) i7-8750H CPU @ 2.2 GHZ e 6 núcleos, 16GB memória RAM, armazenamento interno de 1TB HDD e SSD de 512GB.
- Sistema Operacional: Linux PopOS 22.04 LTS - *Desktop image for 64-bit PC (AMD64).*
- Bash Script: criação de simulações de *pipelines* em ambiente de desenvolvimento local, na versão 5.1.16
- Projetos *Open Source* de JavaScript: linguagem de programação escolhida para se avaliar projetos.
- Python: Linguagem escolhida para realizar alguns scripts para facilitar a geração dos resultados deste trabalho, na versão 3.10.4
- Perf: ferramenta para análise de performance em Linux através de linha de comando, na versão 6.1.11

3.3 Seleção de projetos Open Source

Após selecionada a ferramenta a ser usada para capturar informações de consumo de energia, a próxima etapa foi selecionar projetos para realizar essa avaliação. Como o objetivo deste trabalho é medir e avaliar o consumo de energia em *pipelines* de software, buscou-se encontrar projetos já consolidados e mantidos pela comunidade *Open Source*; estes então deveriam ter ao menos mil estrelas no *Github* - sendo este um valor razoável para a definição de "conhecido pela comunidade".

A linguagem alvo para a procura dos projetos foi o JavaScript (JS). Os projetos procurados foram desde simples bibliotecas até *frameworks* já consolidados no mundo do JS. O critério de seleção também contava com alguns outros fatores. Além de utilizar a linguagem JS como base, dado que o objetivo é medir o consumo de testes, foram selecionados projetos com pelo menos dois tipos de testes (Ex.: testes unitários e testes de integração). Os projetos também podiam conter ou não dependências, e nestes casos estas devem ser introduzidas via *Docker* para facilitar a medição de dados associados. É importante salientar que todos estes projetos deviam conter *pipelines* de CI, para ser possível reproduzir estes passos na intenção da obtenção de um resultado mais próximo do real.

Após cada projeto ser selecionado dentro os critérios apresentados, verificou se era realmente possível reproduzir todas as etapas em ambiente local, sendo considerado uma etapa de pré-estudo do projeto em questão para avaliar a viabilidade de utilizá-lo para a realização da tarefa de medição.

3.4 Elaboração de Scripts para coleta de dados

Após a seleção dos projetos, foi desenvolvido um script bash para auxiliar na captura das informações necessárias. Como já abordado, a ferramenta escolhida para captura de informações referente ao consumo de energia foi o Perf, esta aceita algum comando para realizar a medição: daí que surge o interesse em montar em um único script todas as informações que se decidiu obter.

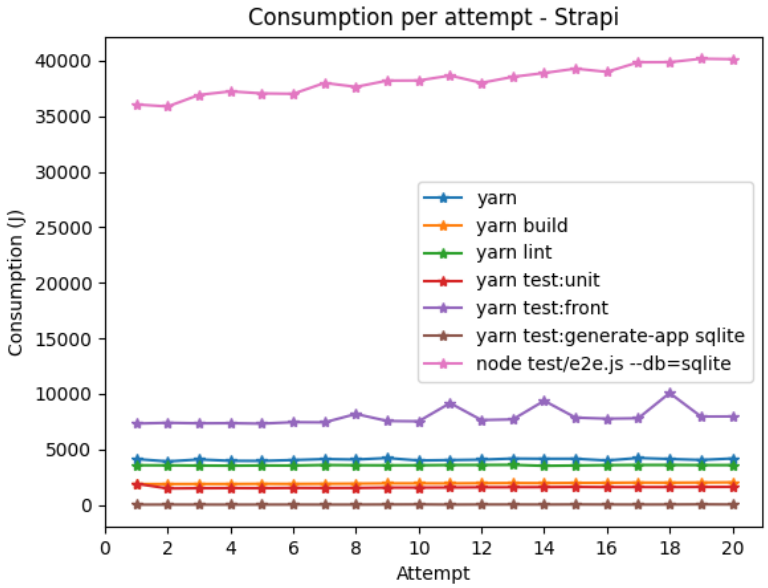
Para o seu desenvolvimento, levou-se em conta medir cada etapa da *pipeline* pegando seu valor individual. Para tal, verificou-se o arquivo responsável pela CI - em muitos casos realizada via GitHub Actions - e a partir da análise destas etapas ali listadas, buscou-se reproduzi-las e apontar o valor consumido em cada uma delas para assim facilitar a amostragem e a formulação dos resultados deste trabalho.

Buscou-se isolar também o valor gasto para construir e montar dependências via *Docker* para os projetos, novamente buscando evidenciar e separar o maior número de informações possíveis, para facilitar as avaliações futuras que eram pretendidas. Para diminuir o desvio padrão do erro ao realizar as medições, múltiplas capturas de uma mesma etapa foram feitas. Cada processo foi executado 20 vezes, onde buscou-se obter a média desses valores e minimizar possíveis variações nos resultados.

Todos estes valores foram adicionados a um arquivo de texto. Assim, criou-se então dois scripts em Python para tratar estes dados. Primeiro, foi necessário um script para ler o arquivo texto, identificar os valores de saída e somá-los e formatá-los preparando-os para serem plotados em um gráfico com todos os 20 pontos colhidos para cada etapa.

Na Figura 6 tem-se um exemplo deste modelo. Onde no eixo X tem-se o número da amostra colhida, e no eixo Y seu valor em Joules. Nesta Figura são colocadas todas as etapas de integração de um projeto, e individualmente foi analisado seu comportamento, buscando não ter dados muito desconexos.

Figura 6: Exemplo de um gráfico gerado após a primeira coleta de dados.



Fonte: Autoria própria

Até aqui, esta etapa do estudo consistiu em verificar se os dados recolhidos eram coesos, para assim ser possível avançar com o tratamento destes valores brutos. Para fins de controle, todos os códigos utilizados para realizar essas medições podem ser consultados em https://github.com/vellone-fabricio/tcc_scripts, bem como os outros dados brutos do projeto.

3.5 Padronização das Etapas da Integração Contínua

Depois de obter os dados brutos, foi necessário uma etapa de padronização para caracterizar os projetos igualmente. Dado que cada projeto pertence à um usuário ou empresa diferente, é esperado encontrar diferentes nomes de scripts nesses projetos, visto que não existe um "padrão" obrigatório a ser seguido. Então, após a avaliação do que cada *script* de inicialização do projeto, caracterizou-se todas as etapas da CI dos projetos em uma das seguintes categorias:

- **Inst. de dep.**: etapa responsável pelo *download* das bibliotecas e *frameworks* de cada projeto;
- **Build**: processo referente à geração dos arquivos minificados em JS ou CSS;
- **Lint**: processo que verifica a padronização dos estilos de código, tais quais como espaçamento, tabulação e distribuição do código como um todo, facilitando a leitura e detecção de erros;
- **Testes de unidade**: Testes unitários da aplicação, geralmente testa, uma função isolada;
- **Testes de integração**: Testes envolvendo integração entre funções, bibliotecas, bancos de dados e APIs;
- **Testes end-to-end (E2E)**: Testes de ponta à ponta na aplicação, geralmente realizando a total integração do processo. Por exemplo, desde o clique em uma interface gráfica até o seu retorno;
- **Limpar cache**: remoção de cache antes do download ou criação de arquivos;
- **Type check**: etapa responsável por validar os tipos para algumas funções com arquivos TypeScript que aparecem no projeto;
- **Check security**: comando encarregado de validar as versões das bibliotecas presente no projeto, identificando possíveis vulnerabilidades nas versões em uso reportadas à comunidade;
- **Docker-up**: inicialização dos contêineres *Docker* para gerenciar dependências;
- **Docker-down**: remoção dos contêineres *Docker* para gerenciar dependências;
- **Preparação e2e**: etapa preparatória em alguns projetos, referente à uma possível criação de dependências na máquina do usuário (como um mini cliente, por exemplo) para execução dos testes E2E do programa.

Desta maneira, todos projetos podem ser comparados pois estão usando as mesmas nomenclaturas de referência. Na Tabela 1, pegou-se o projeto Tsed como exemplo, para enunciar como foi feita esta transformação. Na coluna da esquerda, tem-se os nomes originais (em Inglês) e na coluna da direita sua padronização.

Tabela 1: Exemplo de padronização de etapas de um projeto

Nome original	**Nome após padronização**
yarn	Inst. de dep.
yarn clean	Limpar cache
yarn test:lint	Lint
yarn test	testes de unidade
yarn test:integration	testes de integração
yarn build	Build

Fonte: Autoria própria

3.6 Análise dos dados brutos

Após a padronização, foi possível calcular as médias dos valores obtidos. Esta média foi o valor trabalhado durante todo o escopo deste estudo, julgando-se este valor com menor erro acumulado após a medição dos dados como um todo.

Por fim, muitas agregações de dados foram feitas, tabelas foram geradas com o auxílio da ferramenta *Google Sheets*, tentando ao máximo trabalhar os valores visando responder às QPs propostas. Algumas transformações de unidades foram necessárias ao decorrer deste tratamento de dados dado que a ferramenta usada para o recolhimento dos valores (Perf) retornava os valores brutos em Joule (J). Por exemplo no cálculo da Potência de cada etapa, para este caso, usou-se da seguinte fórmula:

$$P(W) = \frac{E}{\Delta T} \tag{1}$$

Além desta transformação de Energia para Potência, transformou-se também o valor da Energia de Joules para Quilowatt-hora (kWh) buscando a comparação com aparelhos elétricos residenciais, para tal conversão de unidades foi utilizado:

$$E(kWh) = \frac{E(J)}{3,6 \times 10^6} \tag{2}$$

3.7 Ameaças à validade

Nesta seção, são discutidas ameaçadas com relação à validade deste projeto. Daqui, poderão ser tiradas possíveis melhorias de escopo ou também, melhorias para possíveis trabalhos futuros. Um primeiro ponto que pode ser visto como uma ameaça foi o número de projetos avaliados. Neste, avaliou-se 8 projetos dado o tempo e a finalidade que este estudo tem como objetivo, sendo talvez um número limitado de projetos. Logo, não é possível dizer que os resultados aqui apresentados sejam válidos para todas as situações.

Assim, mais projetos deveriam ser levados em conta no estudo, com uma gama maior de variação entre seus tipos.

Outro fator determinante foi a ferramenta utilizada. Todos os cálculos e valores gerados por este estudo ficam limitados ao que a ferramenta Perf (Kerrisk, 2009) retorna para o seu usuário. Além de que, ela tem módulos apenas para medir as energias referentes ao processador, RAM, GPU e processos do SO. Para um estudo mais abrangente, outras alterações deverão ser feitas para abranger todos os componentes do computador: placa de rede, *drivers*, discos utilizados, etc.

A ferramenta Perf também não fornece uma forma direta de medir alguns processos como o caso de contêineres *Docker*. Assim, as etapas do processo que contavam com essa tecnologia podem ter sofrido alguma variação de valores reais com os colhidos. Fica destacado então que todos os valores do projeto fazem jus ao que o *software* retorna, podendo haver algum erro de precisão neste.

Vale a pena citar que, as eventuais padronizações de etapas das *pipelines* dos projetos estudados podem ter gerado algum certo nível de erro, principalmente com relação à agregação de conjuntos de testes, separados nas três categorias abordadas: Teste de Unidade, Teste de Integração e Testes e2e (End-to-End). Muitos projetos contavam com nomenclaturas diferentes, então coube ao autor deste projeto avaliar e separar algumas destas etapas e agrupá-las em conjuntos. E por mais que o script possa ser padronizado, podem existir interferências de processos externos pelo uso dos recursos da máquina durante a determinação dos valores. Apesar de serem feitas 20 coletas para cada *pipeline* não se pode garantir que essas interferências não ocorreram durante as medições.

3.8 Considerações Finais

Portanto, após seguir todas as etapas descritas neste capítulo, foram obtidos todos os dados já refinados com relação ao estudo. Como as análises ocorreram de acordo com as questões de pesquisa, todos os gráficos ou tabelas gerados e coletados durante as etapas citadas, foram agrupados e discutidos no próximo capítulo bem como a discussão dos resultados.

4 ANÁLISE DE RESULTADOS

Neste capítulo serão apresentados os resultados obtidos durante a execução do estudo citado no capítulo anterior. Serão realizadas também as discussões e análises destes. O capítulo ficou estruturado da seguinte maneira: a Seção 4.1 analisou em detalhes os projetos selecionados, nas Seções 4.2 a 4.4 discorreu-se mais em detalhes sobre os dados colhidos visando responder as questões de pesquisas propostas por este trabalho.

4.1 Caracterização dos projetos selecionados

Dado os critérios de seleção apresentados no Capítulo 3 (Seção 3.3), selecionou-se os seguintes projetos do GitHub: *Tsed, Svgo, NuxtJS, Bitcoinjs-lib, Nest, Moleculer, Marble, Strapi*. Todos os escolhidos foram selecionados através da pesquisa padrão do GitHub.

A Tabela 2 mostra os projetos selecionados: a primeira coluna faz referência ao nome do projeto, a segunda a versão em que foi feita a aferição das medidas, a terceira contém o número de estrelas no GitHub e na última coluna têm-se o número de linhas de códigos por projeto. É possível observar que o maior projeto escolhido foi o Marble com 830k linhas de código, e o menor foi a biblioteca Bitcoinjs-lib, com 19k de linhas.

Tabela 2: Projetos Node.js

Projeto	Versão utilizada	Nº de estrelas	linhas de código
Tsed	v6.126.1	2.3k	346k
Svgo	v2.8.0	18.5k	57k
NuxtJS	v2.15.8	41.6k	800k
Bitcoinjs-lib	v6.0.2	4.8k	19k
Nest	v9.0.0	52.3k	47k
Moleculer	v0.14.22	5.4k	71k
Marble	v4.0.0	2.1k	830k
Strapi	v4.4.1	49.9k	430k

Fonte: Dados retirados da página de cada projeto no GitHub, linhas de códigos obtidas através da ferramenta *cloc*

Além do mais, para ajudar na identificação de padrões dos consumos e no auxílio das análises feitas, foi identificado o total de testes para cada projeto. Assim, ficou-se dividido na Tabela 3 na primeira coluna o projeto em questão, e, respectivamente, o número de testes de unidade, de integração e E2E e, por último, o número Total de testes na última coluna à direita. Logo, é possível dizer que o projeto com mais testes de unidade, integração, e2e e quantidade total, são, respectivamente: Bitcoinjs-lib, Nest, Strapi e Bitcoinjs-lib.

Tabela 3: Quantidade de testes por projeto, por tipo.

Projeto	Unidade	Integração	E2E	Total
Tsed	515	8	0	523
Svgo	432	526	0	958
NuxtJS	567	339	87	993
Bitcoinjs-lib	2523	55	0	2578
Nest	1481	885	0	2366
Moleculer	2259	166	9	2434
Marble	399	30	0	429
Strapi	1129	0	1197	2326

Fonte: Autoria própria - análise dos projetos

4.2 QP 1 - Como é o consumo de energia nas etapas de CI em projetos Node.js?

Quando analisados os 8 projetos, foi possível observar que existe uma característica bem diversa com relação aos gastos energéticos; estes estão enunciados na Tabela 4. Nesta Tabela com 14 colunas, a 1ª identifica os 8 projetos analisados neste trabalho, e da 2ª até a 13ª coluna, tem-se as etapas de CI. Assim, cada linha mostra o consumo do projeto na etapa, sendo que foram deixados em branco os processos não contidos no projeto. Por exemplo, o projeto Tsed não contém as etapas de testes e2e, type check, check-security, docker-up, docker-down e preparação e2e. A última coluna apresenta o valor total de consumo no projeto, sendo este o somatório de cada etapa individual. Logo, pode-se dizer que o projeto com maior consumo de energia foi o **Strapi**, tendo 128,3 KJ de energia consumidos no processo.

Ainda assim, as últimas 3 linhas da tabela buscaram medir os valores mínimos, máximos e médios de cada etapa. Tal que é possível ver que a etapa com maior valor médio foi a de testes de integração, devido ao alto consumo da mesma no projeto Strapi. Ademais, foi evidenciado em vermelho a etapa com maior consumo dentro de cada projeto.

Tabela 4: Energia (em Joules) consumida por todos os projetos analisados

	Inst. de dep.	Build	Lint	Testes de unidade	Testes de integração	Testes e2e	Limpar cache	Type check	Check security	Docker-up	Docker-down	Preparação e2e	Total
tsed	2.380,76	14.006,97	1.133,07	11.777,88	3.950,93		215,62						33.465,21
svgo	608,84		189,47	633,48	3.723,77			260,30					5.415,85
nuxt	519,83	115,83	1.696,72	11.797,19	3.757,83	463,20		130,99	70,37				18.551,94
bitcoinjs-lib	359,96	2.650,88	604,68	2.557,29	562,59								6.735,39
nest	649,21	1.933,09	948,10	1.484,40	6.591,50					1.168,01	84,95		12.859,24
moleculer	555,94			1.968,30	618,06	1.919,70							5.061,99
marble	359,96	2.650,88	604,68	2.557,29	4.801,21								10.974,01
strapi	4.089,73	1.942,30	3.571,19	9.246,35		109.485,81						34,26	128.369,63
min	359,96	115,83	189,47	633,48	562,59	463,20	215,62	130,99	70,37	1.168,01	84,95	34,26	
max	4.089,73	14.006,97	3.571,19	11.797,19	6.591,50	109.485,81	215,62	260,30	70,37	1.168,01	84,95	34,26	
média	1.190,53	3.883,32	1.249,70	5.252,77	3.429,41	37.289,57	215,62	195,65	70,37	1.168,01	84,95	34,26	

Fonte: Autoria própria

Considerando que os dados brutos às vezes não refletem quanto cada etapa realmente está consumindo, transformou-se os dados em porcentagem. Então, considerando que o consumo total da aplicação é 100% calculou-se quantos por cento cada etapa consumiu. Dado este cálculo, a Tabela 5 se apresenta de maneira similar à Tabela 4.

Tabela 5: Energia (em Porcentagem) consumida por todos os projetos analisados

	Inst. de dep.	Build	Lint	Testes de unidade	Testes de integração	Testes e2e	Limpar cache	Type check	Check security	Docker-up	Docker-down	Preparação e2e	Total
tsed	7,11%	41,86%	3,39%	35,19%	11,81%		0,64%						100,00%
svgo	11,24%		3,50%	11,70%	68,76%			4,81%					100,00%
nuxt	2,80%	0,62%	9,15%	63,59%	20,26%	2,50%		0,71%	0,38%				100,00%
bitcoinjs-lib	5,34%	39,36%	8,98%	37,97%	8,35%								100,00%
nest	5,05%	15,03%	7,37%	11,54%	51,26%					9,08%	0,66%		100,00%
moleculer	10,98%			38,88%	12,21%	37,92%							100,00%
marble	3,28%	24,16%	5,51%	23,30%	43,75%								100,00%
strapi	3,19%	1,51%	2,78%	7,20%		85,29%						0,03%	100,00%
min	2,80%	0,62%	2,78%	7,20%	8,35%	2,50%	0,64%	0,71%	0,38%	9,08%	0,66%	0,03%	
max	11,24%	41,86%	9,15%	63,59%	68,76%	85,29%	0,64%	4,81%	0,38%	9,08%	0,66%	0,03%	
média	6,12%	20,42%	5,81%	28,67%	30,91%	41,90%	0,64%	2,76%	0,38%	9,08%	0,66%	0,03%	
SD	3,39%	17,95%	2,71%	18,99%	23,62%	41,54%		2,90%					

Fonte: Autoria própria

É possível traçar algumas observações a respeito dos perfis encontrados. Nota-se que, em 75% dos projetos analisados, a etapa de maior consumo energético foi uma etapa de testes. Nos 25% restantes, o gasto na etapa de maior consumo (novamente nota-se que a

etapa de destaque foi a de *Build*) foi similar à outra etapa de testes (nos dois projetos, vê-se que foi o processo de testes de unidade).

Para estes projetos (tsed e bitcoinjs-lib), a etapa de *Build* do projeto acaba sendo mais intensa, o que nos leva a questionar o porquê disto ocorrer, destoando das outras análises. Alguns pontos para se considerar são: a complexidade dos testes e da biblioteca em si, o tamanho do software escrito (linhas de código existentes) e a quantidade de bibliotecas externas que são utilizadas nestes dois projetos. Estes fatores acabam transformando a etapa de criação de projeto mais custosa devido ao montante de código necessário para minificar e importar em comparação com a complexidade dos testes executados.

Outra análise é com relação ao gasto nas etapas de teste. Realizar testes em um software acaba sendo tão custoso que, em todos os projetos, ao analisar as porcentagens referentes às etapas de teste, é possível ver que esse valor de gasto energético foi sempre muito alto, sendo que o mínimo encontrado para a somatória destas etapas foi um consumo de 46% de todo o gasto energético da *pipeline* (Bitcoinjs-lib) e chegando até 92% do total (Strapi), levando a conclusão de que testes são uma parte importante do software mas que também são uma das partes mais custosas do ponto de vista energético.

Ao analisar as Tabelas 3 e 4, nota-se que quanto maior a complexidade de um teste (unidade $<$ integração $<$ *end-to-end*), maior o seu custo energético. Por exemplo, o projeto com maior custo foi o Strapi, chegando a consumir em média 128 kJ por execução, onde 85% desse valor foi gasto somente na etapa de testes *end-to-end* e este foi o projeto com maior número de testes *end-to-end* dentre os analisados, tendo 1197 testes desta categoria.

Para os testes de integração quando comparados aos de unidade, pode-se cruzar novamente os dados das Tabelas 3 e 4 para confirmar que, na maioria dos casos, a quantidade de testes de integração é inferior a quantidade de testes de unidade, sendo a diferença entre estes números bem maior do que as porcentagens dos mesmos. Há casos como o do Marble, em que a quantidade de testes de integração é 1/10 a de unidade e, mesmo assim, o seu consumo (em porcentagem) é quase o dobro.

4.3 QP 2 - Qual é a relação entre o consumo de energia e o tempo de execução nas etapas de CI?

Para ajudar nesta análise, foi levado em consideração o tempo médio de execução para os projetos, distribuídos em etapas, como consta na Tabela 6. Esta tabela segue as mesmas especificações da Tabela 4, porém agora apresentando as quantidades relacionadas ao tempo de execução, em segundos, para cada processo da CI.

Tabela 6: Tempo (em Segundos) consumido por todos os projetos analisados

	Inst. de dep.	Build	Lint	Testes de unidade	Testes de integração	Testes e2e	Limpar cache	Type check	Check security	Docker-up	Docker-down	Preparação e2e	Total
tsed	35,80	104,86	15,35	114,27	47,84		3,30						321,42
svgo	8,35		2,78	6,83	76,75			3,45					98,15
nuxt	9,55	2,23	30,33	173,74	98,97	9,58		1,96	2,41				328,77
bitcoinjs-lib	5,32	29,25	8,90	50,63	17,34								111,44
nest	12,13	27,11	7,72	26,60	54,97					32,13	6,83		167,50
moleculer	7,45			13,26	19,19	53,52							93,43
marble	5,32	29,25	8,90	50,63	17,34								111,44
strapi	85,48	25,80	65,58	62,63		728,39						0,62	968,51
min	5,32	2,23	2,78	6,83	17,34	9,58	3,30	1,96	2,41	32,13	6,83	0,62	
max	85,48	104,86	65,58	173,74	98,97	728,39	3,30	3,45	2,41	32,13	6,83	0,62	
média	21,18	36,42	19,94	62,32	47,49	263,83	3,30	2,70	2,41	32,13	6,83	0,62	

Fonte: Autoria própria

É possível observar que o tempo das etapas com maior duração são as etapas relacionadas aos três tipos de testes considerados, confirmando as evidências da Seção 4.2 de que, do ponto de vista energético, estes passos são os mais custosos para a *pipeline* também em tempo de execução. E, novamente, seguindo-se as especificações da Tabela 5, apresentou-se os dados em forma de porcentagem.

Tabela 7: Tempo (em Porcentagem) consumido por todos os projetos analisados

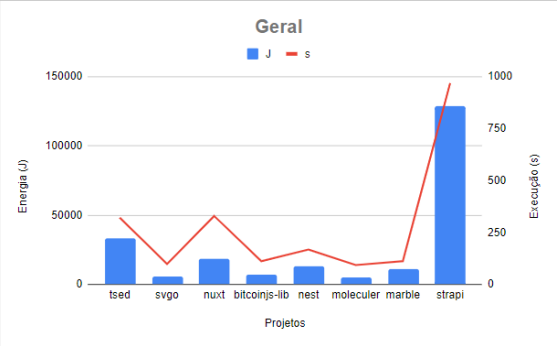
	Inst. de dep.	Build	Lint	Testes de unidade	Testes de integração	Testes e2e	Limpar cache	Type check	Check security	Docker-up	Docker-down	Preparação e2e	Total
tsed	11,14%	32,62%	4,77%	35,55%	14,88%		1,03%						100,00%
svgo	8,51%		2,83%	6,95%	78,20%			3,51%					100,00%
nuxt	2,90%	0,68%	9,22%	52,85%	30,10%	2,91%		0,60%	0,73%				100,00%
bitcoinjs-lib	4,77%	26,25%	7,99%	45,43%	15,56%								100,00%
nest	7,24%	16,19%	4,61%	15,88%	32,82%					19,18%	4,08%		100,00%
moleculer	7,97%			14,19%	20,55%	57,29%							100,00%
marble	4,77%	26,25%	7,99%	45,43%	15,56%								100,00%
strapi	8,83%	2,66%	6,77%	6,47%		75,21%						0,06%	100,00%
min	2,90%	0,68%	2,83%	6,47%	14,88%	2,91%	1,03%	0,60%	0,73%	19,18%	4,08%	0,06%	
max	11,14%	32,62%	9,22%	52,85%	78,20%	75,21%	1,03%	3,51%	0,73%	19,18%	4,08%	0,06%	
média	7,02%	17,44%	6,31%	27,84%	29,67%	45,14%	1,03%	2,05%	0,73%	19,18%	4,08%	0,06%	
SD	2,68%	13,32%	2,30%	19,00%	22,60%	37,65%		2,06%					

Fonte: Autoria própria

Portanto, usaram essas informações para realizar análises mais detalhadas. Na Figura 7 tem-se em um gráfico misto de barras e linhas com o perfil do consumo médio e

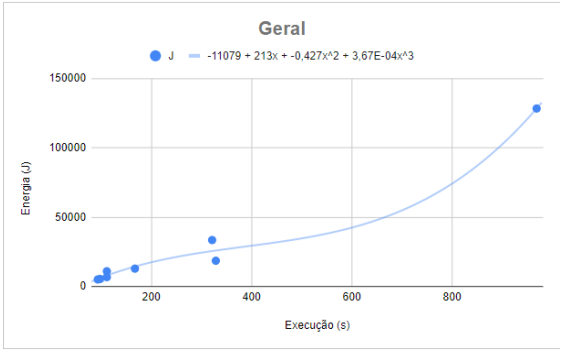
do tempo de execução médio das etapas de integração contínua para cada um dos oito projetos analisados. As barras em azul evidenciam o consumo em Joules, com sua legenda ao lado esquerdo do gráfico e a linha em vermelho evidencia o tempo em segundos para a execução média total de um ciclo da CI, com sua legenda ao lado direito. Já para a Figura 8, no eixo Y foi colocado os valores do consumo médio por ciclo completo da CI e no eixo X seu valor correspondente de tempo, respectivamente em Joules e Segundos, sem enunciar o projeto em questão. É possível verificar a curva de tendência para esta, sendo um polinômio de ordem três a melhor escolha para esta distribuição de dados. Ambas as Figuras 7 e 8 apresentam os mesmos dados de formas diferentes para ajudar na interpretação das questões:

Figura 7: Gráfico do consumo total em Joules e do tempo total para cada projeto



Fonte: Autoria própria

Figura 8: Gráfico do custo total (J) por tempo total (s)

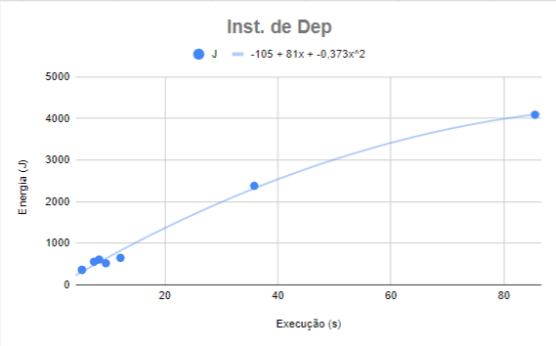


Fonte: Autoria própria

Foi observado que o consumo de energia aumenta com o passar do tempo para o perfil geral de consumo dos projetos em questão. E ainda, quando analisada a tendência desse consumo (curva de tendência), foi obtido um perfil crescente se assemelhando à uma curva exponencial. Para melhor responder à questão proposta, foi necessário também olhar cada etapa da CI individualmente e considerando apenas as etapas mais comuns dentre os projetos. Estas etapas foram: Instalação de Dependências, *Build*, *Lint*, Testes de Unidade e Testes de Integração.

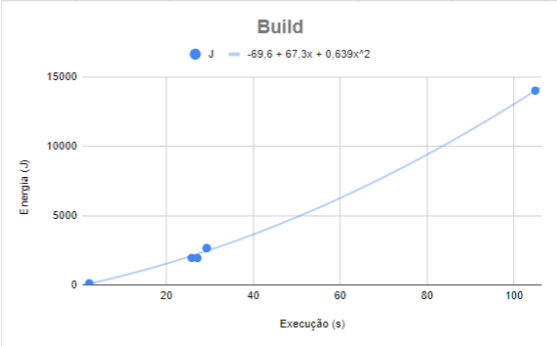
As Figuras de 9 à 13, assim como na Figura 8, mostram no eixo Y o custo do consumo em Joules e no eixo X o tempo gasto para aquele projeto. A diferença agora é que, ao invés do perfil geral de consumo, foi analisada cada etapa individualmente da CI. Na Figura 14 foi usado novamente o gráfico misto de consumo e tempo, para ajudar na análise da etapa de Testes de Integração.

Figura 9: Gráfico do custo da etapa de Instalação de Dependências (J) pelo tempo (s)



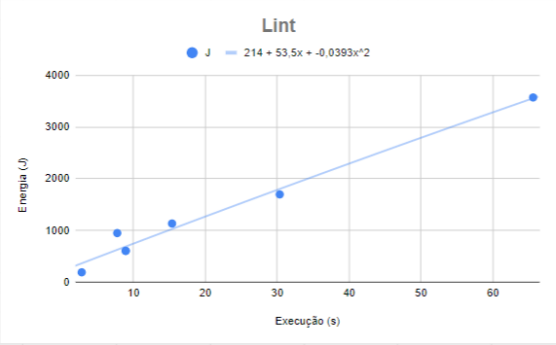
Fonte: Autoria própria

Figura 10: Gráfico do custo da etapa de Build (J) pelo tempo (s)



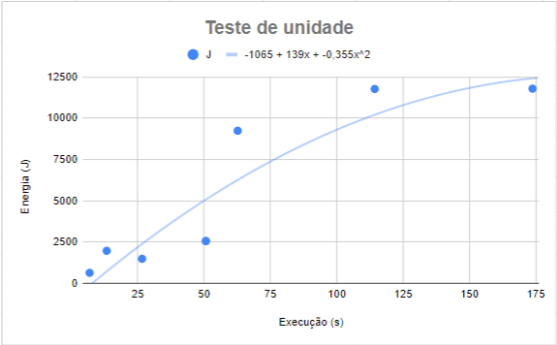
Fonte: Autoria própria

Figura 11: Gráfico do custo da etapa de Lint (J) pelo tempo (s)



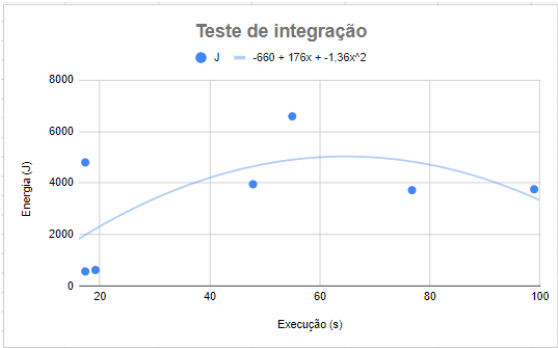
Fonte: Autoria própria

Figura 12: Gráfico do custo da etapa de Testes de unidade (J) pelo tempo (s)



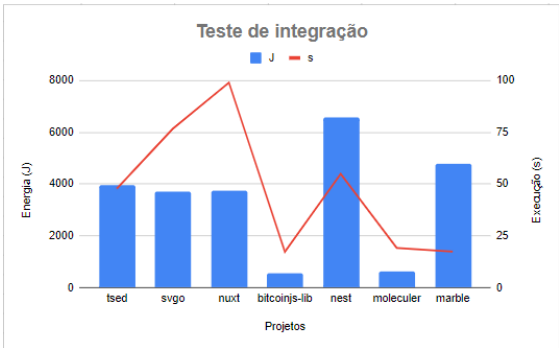
Fonte: Autoria própria

Figura 13: Gráfico do custo da etapa de Testes de integração (J) pelo tempo (s)



Fonte: Autoria própria

Figura 14: Gráfico do consumo em Joules e do tempo total para os Testes de integração



Fonte: Autoria própria

Apesar do perfil global mostrado nas Figuras 7 e 8 seguir uma linha crescente de consumo com relação ao tempo de execução, quando detalhado em etapas menores outras coisas se tornam explícitas. As Figuras 10 e 11 continuam seguindo o panorama geral

do consumo, onde com o passar do tempo de execução o valor da energia acumulada continua aumentando. Este perfil mais previsível pode ser explicado pelo simples fato de que são as duas etapas de acabam dependendo exclusivamente da máquina do usuário: são procedimentos que exigem apenas do processador e das memórias do computador onde estão executando, onde são realizadas rotinas de verificação de código para padronização - dentro da etapa de *linting* - e compactação/minificação dos arquivos - para a etapa de *build* -, sendo este um trabalho executado apenas localmente, sem interferências externas.

Para as Figuras 9 e 12, obteve-se uma curva um pouco mais achatada, mostrando um possível perfil de diminuição ou talvez estabilização do consumo em questão. Aqui, já foi obtido um padrão diferente do cenário global da CI. A possível explicação aqui se dá pelo fato de que ambas etapas são processos que dependem mais de fatores "externos" ao que foi executado. Para a etapa de instalação de dependências, por exemplo, é necessário fazer o *download* das bibliotecas/*frameworks* utilizados, o que acaba gerando tempos ociosos durante estas requisições, transferindo o consumo de energia para a placa de rede, por exemplo. O mesmo pode ser dito da etapa de testes de unidade. Apesar de geralmente esta etapa não contar com interferências externas, ainda fica sujeita ao método de como as ferramentas de testes trabalham, pausas de execução ou qualquer outro evento adverso que estes códigos trazem consigo.

Já para a etapa de Testes de Integração, o cenário é mais atípico se comparado com o perfil geral visto anteriormente. Na Figura 13 é possível notar que esta fase conta com o decrescimento do consumo ao aumentar-se o tempo, e na Figura 14 se tentou mapear como esses perfis foram distribuídos por projetos. Uma das possíveis explicações para esse acontecimento se dá por fortes interferências externas. Geralmente, no desenvolvimento de um *software*, esta etapa é a que contém ligações com o mundo externo, é onde chamada à outros serviços, conexões e muitos outros fatores são testados no projeto. Sendo essa uma camada não isolada (diferente dos testes de unidade, nos quais se tenta ao máximo não interligar componentes/métodos), inúmeros podem ser os fatores que ocasionam essa diferença, alguns destes projetos até criavam mini-instâncias dos seus serviços para poderem realizar esta etapa. Outro fator, é que certos projetos (Nest, Moleculer e Marble) contém contêineres ativos para complementarem e tornarem possível a realização neste nível de testes. Projetos como o Nest e o Marble necessitavam de Bancos de Dados executando na máquina para testar operações, custos esses que foram somados e levados em consideração na análise dos dados. Outras informações relevantes que possivelmente interferiram na medição destes valores foram detalhados no Seção 3.7.

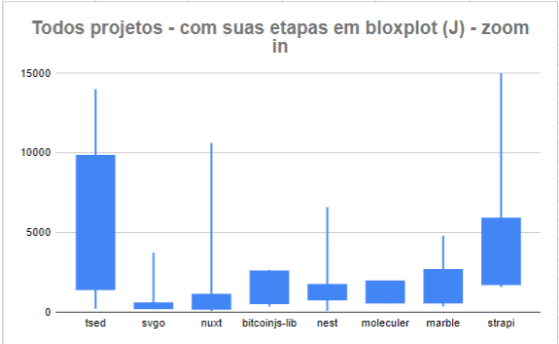
4.4 QP 3 - Como a perspectiva do consumo de energia em CI se relaciona com o consumo no dia-a-dia?

Partindo para a última análise, apresentou-se os perfis energéticos de cada projeto em um gráfico de quartis, para se considerar como se deram as variações de medidas dentro do projeto por etapas. Assim, nas Figuras 15 e 16 têm-se estas amostragens. A Figura 15 contém a visão geral dos perfis em Joule, e na Figura 16 foi usado um limitador no eixo Y, dado que a variação da distribuição para o projeto Strapi foi muito alta, para ser possível visualizar melhor todos os dados.

Figura 15: Apresentação gráfica em quartis para as médias de cada projeto (em Joules)

Fonte: Autoria própria

Figura 16: Apresentação gráfica em quartis para as médias de cada projeto (em Joules) - com zoom



Fonte: Autoria própria

É possível concluir aqui que as variações dos valores máximos são bem grandes, dado que as etapas que obtiveram maior consumo de energia (geralmente de testes, como já apresentado na Seção 4.2) na maioria dos casos tinham diferenças bem significativas com os valores das outras fases em um mesmo projeto, sendo bem maiores que os valores para etapas menos custosas dos mesmos (por exemplo, o processo de *linting*). Levando em consideração todos os dados apresentados até agora, pegou-se os valores de médias gerais de consumo energético e de tempo de execução de cada projeto das Tabelas 4 e 6 e, utilizando da Equação 1, foi obtida a Tabela 8 com os valores de Potência Total gasta para cada projeto durante sua execução.

Tabela 8: Potência Total (em W) dos valores médios totais de cada projeto

Projeto	Potência Total (W)
Tsed	104,12
Svgo	55,18
NuxtJS	56,43
Bitcoinjs-lib	60,44
Nest	76,77
Moleculer	54,18
Marble	98,47
Strapi	132,54

Fonte: Autoria própria

Para ser possível aproximar esses dados de medidas conhecidas do dia-a-dia da população, juntou-se na Tabela 9 os valores de Potência média de alguns produtos de uso diário de uma residência, além de conter o valor estimado de consumo médio de energia mensal para os mesmos. O cálculo do consumo médio mensal é baseado no uso estimado do produto (em horas) vezes o número estimado de dias que esse aparelho é utilizado pelo consumidor, multiplicados pela sua potência nominal média.

Tabela 9: Consumo equipamentos elétricos

Aparelhos Elétricos	Potência (W)	Consumo médio Mensal (kWh)
Rádio relógio	5	3,6
Roteador	6	1,44
Modem de internet	8	1,92
Aparelho de DVD	15	0,24
Lâmpada fluorescente compacta	15	2,25
Prancha (chapinha)	33	0,33
Monitor	55	13,2
Ventilador de mesa	72	17,28
Ventilador de teto	73	17,52
TV em cores - 32" (LCD)	95	14,25
Lâmpada incandescente	100	15
Máquina de costura	100	3
Aparelho de som	110	6,6
Secador de cabelo	1000	5,21

Fonte: http://www.procelinfo.com.br/main.asp?View=%7BE6BC2A5F-E787-48AF-B485-439862B17000%7D

Logo, é possível dizer que, da análise das Tabelas 8 e 9 que, em termos de potência, por *pipelines* tem-se o equivalente à potência de um Monitor ou de um ventilador de mesa/teto. Pode-se dizer também que, para projetos mais custosos como o Strapi, por exemplo, a potência exigida foi maior do que uma máquina de costura ou um aparelho de som.

Para maior efeito de comparação, foi também estimado o valor do consumo médio mensal para estas *pipelines*. Buscou-se estimar, assim como foi feito para os aparelhos elétricos, o uso mensal destes processos para uma equipe pequena, trabalhando o mês todo em um desses projetos. Portanto, foram utilizados os seguintes números:

1. Dias de uso estimado no mês: 22 dias úteis de trabalho
2. Quantidade de pessoas na equipe: 5 pessoas
3. Média de requisições de códigos feitas por dia/pessoa: 1,5

Utilizando estes números com os valores obtidos para tempo total na Tabela 6 e a transformação da energia consumida na Tabela 4 ao aplicar a Fórmula 2, foi gerada a Tabela 10 onde tem-se na primeira coluna os projetos e na segunda o valor calculado do seu consumo médio de energia em kWh.

Tabela 10: Média de consumo das pipelines (em kWh/mês)

Projeto	Consumo médio Mensal (kWh)
Tsed	1,53
Svgo	0,25
NuxtJS	0,85
Bitcoinjs-lib	0,31
Nest	0,59
Moleculer	0,23
Marble	0,50
Strapi	5,88

Fonte: Autoria própria

Desse modo, novamente foi possível observar na Tabela 9 que as *pipelines* bateram um consumo médio mensal de uma chapinha, por exemplo, e em alguns casos (novamente citando o Strapi - projeto de maior consumo) notou-se que o projeto poderia consumir mais que um secador de cabelos na fatura final de uma residência. Por isso, mesmo os projetos avaliados terem apresentado potências relativamente altas para um ciclo completo da *pipeline*, dado o tempo de execução baixo quando comparados ao uso de eletrodomésticos em uma residência, estes consumos acabam não impactando significativamente a conta final.

4.5 Considerações Finais

Após as análises apresentadas fornecem uma visão de como é a distribuição do uso de energia nas etapas de CI. Com relação ao consumo geral de energia dessas *pipelines*, acabou por se considerar apenas uma equipe pequena cuidando de um único projeto.

Porém, sabe-se que no mundo empresarial os times e a quantidade de aplicações acaba sendo bem maior.

Considerando que cada projeto aqui analisado fosse uma amostra de um serviço dentro de uma arquitetura de microserviços, e levando em consideração o montante de projetos grandes empresas do mundo da tecnologia, este número de serviços podem passar facilmente de centenas, com centenas de times trabalhando nas mesmas. Assim, se levar em consideração o valor consumido pelo Strapi segundo à Tabela 10 - um valor de 5,88 kWh/mês -, ao se extrapolar este valor por no mínimo 100 vezes, seria obtido um consumo mensal médio de 588 kWh. Segundo Fedrigo, Gonçalves e Lucas (2009), um consumo residencial mensal é de, em média, 150 kWh, o que mostra que este valor encontrado acaba sendo aproximadamente 4 vezes maior que uma residência comum no Brasil.

Todas as conclusões observadas e retiradas das análises deste capítulo serão apresentadas no capítulo seguinte.

5 CONSIDERAÇÕES FINAIS

O objetivo do trabalho foi estudar projetos JavaScript, buscando medir o consumo de energia nas diversas etapas da CI e conseguindo assim responder as QPs propostas. Verificou-se que as etapas que envolvem os testes são as mais custosas. Dos dados coletados, viu-se que quanto maior a complexidade do teste, maior o seu gasto. O projeto com maior quantidade de testes e2e (Strapi) consumiu, aproximadamente, 4 vezes mais energia que o segundo da lista por consumo de energia. Além disso, metade dos projetos analisados precisaram de ferramentas auxiliares para conseguir executar os testes mais complexos, como bancos de dados ou outras aplicações, aumentando ainda mais o consumo para esta categoria em específico. Observou-se também que o processo de *build* de um projeto pode consumir mais energia que os testes em si para projetos com grandes quantidades de códigos em comparação com a quantidade de testes totais escritas para a aplicação em questão - como foi o caso do Tsed. Logo, pode-se dizer que seguir a pirâmide de testes de *software* é uma prática sustentável, visto que quanto mais complexo, além de tornar a CI mais lenta, o consumo de energia foi constatado maior para as amostras.

Quanto a relação entre o tempo de execução das etapas e o consumo energético, observou-se que quanto mais o processo for dependente da máquina exclusivamente (por exemplo, processos de *build* e *linting*), o consumo aumenta linearmente com o passar do tempo pois o processamento se concentra na CPU e no acesso as memórias do sistema sendo processos mais diretos. Para os outros casos, onde a dependência de outras partes do *hardware* existe, termina-se mais difícil computar todos os consumos fora do SO, e foi observado que existe uma tendência logarítmica na relação do tempo e o consumo de energia para os demais processos (ou até uma inflexão para os testes de integração). O fato ocorre devido à distribuição deste processamento, por exemplo a utilização da placa de rede para *downloads* de bibliotecas como pode ser visto na etapa de Instalação de dependências, ou até pausas e interações com sistemas externos onde a aplicação fica em espera e não necessariamente mantém o consumo de energia em um valor alto.

Por fim, com relação ao consumo de energia da CI e itens do dia-a-dia, constatou-se que a grande maioria dos projetos teve um consumo baixo de energia. Para fins de avaliação, foi usado como exemplo o projeto mais custoso (Strapi). Mesmo utilizando-o, quando analisado uma única execução de *pipeline*, o consumo não é muito pertinente, visto que para este projeto a execução leva, aproximadamente, 15 minutos. Porém, uma *pipeline* não é executada somente uma vez ao dia e não se tem apenas um desenvolvedor trabalhando em um projeto. Ao se considerar uma equipe maior e outras métricas, observou-se que o consumo de energia da *pipeline* para este projeto assemelhou-se ao de um secador de cabelos e consumindo muito mais que outros aparelhos eletrônicos como, por exemplo, aparelhos de DVD, modens de Internet ou roteadores.

Ao levar em conta equipes ainda maiores, onde este projeto poderia ser considerado

apenas uma parte de um projeto maior da empresa (levando-se em conta, por exemplo, uma arquitetura de microsserviços), o cenário muda ainda mais. Grandes empresas da área de tecnologia podem ter milhares de funcionários, o que consistiria em centenas de times. Considerando 100 desenvolvedores no setor de tecnologia de uma determinada empresa e o mesmo consumo médio do projeto, poderia chegar à um consumo de energia 4 vezes maior que a média do consumo de uma residência brasileira, tornando-se um valor significativo. Portanto, deve-se sim levar em consideração o consumo de energia nesse segmento e buscar maneiras de minimizar a quantidade de tempo e vezes que a CI é executada ao longo do dia para um desenvolvimento mais consciente e sustentável.

Na condução deste estudo, vislumbrou-se alguns trabalhos futuros. Por exemplo, seria possível aumentar o número de projetos analisados, buscando ter uma maior cobertura e diversidade. Outra possibilidade projeto seria a escolha de outras linguagens de programação, para investigar como a distribuição de energia nas *pipelines* acontecem para outras linguagens, podendo assim possivelmente associar o tipo da mesma e seu consumo. Além disso, poderia-se comparar como se dá esse perfil de consumo energético em ambientes de CI como, por exemplo, *GitHub Actions*, *Jenkis* ou *Circle CI*. Além disso, seria possível verificar o impacto do SO nesse consumo: se existe diferença entre a utilização do mesmo quando o assunto é a eficiência energética desses processos. Outras melhorias podem ser propostas a partir da Seção 3.7, possibilitando outros pesquisadores de buscar soluções para contornar esses problemas.

Referências

1 REIS, D. *ONU: Relatório aponta que intervalo entre 2015 e 2022 pode ser o mais quente da história*. 2022. <https://www.cnnbrasil.com.br/internacional/onu-relatorio-aponta-que-intervalo-entre-2015-e-2022-pode-ser-o-mais-quente-da-historia/>. Matéria publicada pela CNN sobre problemas climáticos.

2 DEWAN, A.; CASSIDY, A. *COP26 termina com acordo climático; veja o que deu certo e as falhas nas negociações*. 2021. <https://www.cnnbrasil.com.br/internacional/cop26-termina-com-acordo-climatico-veja-o-que-deu-certo-e-as-falhas-nas-negociacoes/>. Matéria publicada pela CNN sobre acordos climáticos.

3 CORAM, A.; KATZNER, D. W. Reducing fossil-fuel emissions: Dynamic paths for alternative energy-producing technologies. *Energy Economics*, v. 70, p. 179 – 189, 2018. ISSN 0140-9883.

4 TERRA, N. *Cresce busca das empresas por transformação digital*. 2022. <https://www.terra.com.br/noticias/cresce-busca-das-empresas-por-transformacao-digital,daafafa560173198120af3f9eee011d5mj6zu2fa.html>. Noticia recente no contexto da pandemia, acessado em 08/09/2022.

5 KURP, P. Green computing. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 51, n. 10, p. 11–13, oct 2008. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/1400181.1400186>.

6 RIEGER, F.; BOCKISCH, C. Survey of approaches for assessing software energy consumption. In: *Proceedings of the 2nd ACM SIGPLAN International Workshop on Comprehension of Complex Systems*. New York, NY, USA: Association for Computing Machinery, 2017. (CoCoS 2017), p. 19–24. ISBN 9781450355216. Disponível em: <https://doi.org/10.1145/3141842.3141846>.

7 MEYER, M. Continuous integration and its tools. *IEEE Software*, v. 31, n. 3, p. 14–16, 2014.

8 Deutshe Welle. *Data centers keep energy use steady despite big growth*. 2022. <https://www.dw.com/en/data-centers-energy-consumption-steady-despite-big-growth-because-of-increasing-efficiency/a-60444548>. Acessado em 18/11/22.

9 MANOTAS, I. et al. An empirical study of practitioners' perspectives on green software engineering. In: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. [S.l.: s.n.], 2016. p. 237–248.

10 LI, Q.; ZHOU, M. The survey and future evolution of green computing. In: *2011 IEEE/ACM International Conference on Green Computing and Communications*. [S.l.: s.n.], 2011. p. 230–233.

11 AHMED, K. M. U.; BOLLEN, M. H. J.; ALVAREZ, M. A review of data centers energy consumption and reliability modeling. *IEEE Access*, v. 9, p. 152536–152563, 2021.

12 CRUZ, L. *Tools to Measure Software Energy Consumption from your Computer*. 2021. <http://luiscruz.github.io/2021/07/20/measuring-energy.html>. Blog post.

13 SHAHIN, M.; BABAR, M. A.; ZHU, L. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, v. 5, p. 3909–3943, 2017.

14 TREINAWEB. *No final das contas: o que é o Docker e como ele funciona?* <https://www.treinaweb.com.br/blog/no-final-das-contas-o-que-e-o-docker-e-como-ele-funciona>. Acessado em 18/11/2022.

15 AMSEL, N.; TOMLINSON, B. Green tracker: A tool for estimating the energy consumption of software. In: *CHI '10 Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2010. (CHI EA '10), p. 3337–3342. ISBN 9781605589305. Disponível em: <https://doi.org/10.1145/1753846.1753981>.

16 SCHUBERT, S. et al. Profiling software for energy consumption. In: *2012 IEEE International Conference on Green Computing and Communications*. [S.l.: s.n.], 2012. p. 515–522.

17 PANG, C. et al. What do programmers know about software energy consumption? *IEEE Software*, v. 33, n. 3, p. 83–89, 2016.

18 SOUZA, W. Silva-de et al. Containergy—a container-based energy and performance profiling tool for next generation workloads. *Energies*, v. 13, n. 9, 2020. ISSN 1996-1073. Disponível em: <https://www.mdpi.com/1996-1073/13/9/2162>.

19 CANONICAL. *Powerstat canonical Documentation*. 2011. <https://manpages.ubuntu.com/manpages/xenial/man8/powerstat.8.html>. Web documentation.

20 VEN, A. van de. *Powertop readme page on Github*. 2007. <https://github.com/fenrus75/powertop>. Web documentation.

21 KERRISK, M. *Perf tool documentation*. 2009. <https://www.man7.org/linux/man-pages/man1/perf.1.html>. Web doc.

22 NERSC. *Likwid Documentation*. 2015. <https://docs.nersc.gov/accounts/policy/#acknowledge-use-of-nersc-resources>. Web documentation.

23 FEDRIGO, N. S.; GONÇALVES, G.; LUCAS, P. F. *Usos Finais de Energia Elétrica no Setor Residencial Brasileiro*. 2009. <https://labeee.ufsc.br/pt-br/node/480#:~:text=Observou\%2Dse\%20que\%20o\%20consumo,3\%20kWh\%2Fm\%C3\%AAs\%20no\%20inverno>. Artigo UFSC sobre consumo residencial energético no Brasil.