Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Desenvolvimento de um pipeline de implantação contínua para uma aplicação de Internet das Coisas.

Bruno Henrique Pastor

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
Igor Muzetti Pereira

Outubro, 2022
João Monlevade–MG

Bruno Henrique Pastor

Desenvolvimento de um pipeline de implantação contínua para uma aplicação de Internet das Coisas.

Orientador: Igor Muzetti Pereira

Monografia apresentada ao curso de Engenharia de Computação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto

João Monlevade

Outubro de 2022

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

P293d Pastor, Bruno Henrique.
Desenvolvimento de um pipeline de implantação contínua para uma aplicação de Internet das Coisas. [manuscrito] / Bruno Henrique Pastor. - 2022.
58 f.: il.: color., tab..

Orientador: Prof. Me. Igor Pereira.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Graduação em Engenharia de Computação .

1. Engenharia de software. 2. Internet das coisas. 3. Software de aplicação. 4. Software - Desenvolvimento. I. Pereira, Igor. II. Universidade Federal de Ouro Preto. III. Título.

CDU 004.41

Bibliotecário(a) Responsável: Flavia Reis - CRB6-2431

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTACAO E SISTEMAS

FOLHA DE APROVAÇÃO

Bruno Henrique Pastor

Desenvolvimento de um pipeline de implantação contínua para uma aplicação de Internet das Coisas

Monografia apresentada ao Curso de Engenharia de Computação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação

Aprovada em 27 de Outubro de 2022

Membros da banca

MSc. Igor Muzetti Pereira - Orientador (Universidade Federal de Ouro Preto)
Dr. Filipe Nunes Ribeiro (Universidade Federal de Ouro Preto)
Dr. Diego Zuquim Guimarães Garcia (Universidade Federal de Ouro Preto)

Igor Muzetti Pereira, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 07/11/2022

Documento assinado eletronicamente por **Igor Muzetti Pereira**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 07/11/2022, às 16:47, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0423391** e o código CRC **1EA87307**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0819 - www.ufop.br

Dedico este trabalho primeiramente a Deus, e a jornada de evolução que me é permitido, a minha mãe e toda minha família.

Agradecimentos

Agradeço imensamente as pessoas que me apoiaram em cada momento, a dedicação dos meus professores em me passar conhecimento. Desejo a vocês todas o melhor que esse mundo pode oferecer. Agradeço também aos obstáculos e as pessoas que não ajudaram, pois sem dificuldades não existe superação nem aprimoramento, acredito que todos aspectos da vida fazem parte do processo.

“Notre bonheur sera naturellement proportionnel au bonheur que nous faisons pour les autres.”

— Hippolyte Léon Denizard Rivail - Allan Kardec (1804 – 1869),

Resumo

Um sistema de *software* é geralmente desenvolvido por uma equipe, devido sua complexidade. A cultura colaborativa DevOps, combinação de Desenvolvimento e Operação, visa melhorar o processo de desenvolvimento e implantação do *software*, tornando-o repetível, confiável e automatizado. Um pipeline de integração e implantação contínua, consiste na segmentação dos processos necessários desde o *commit* até a implantação do *software*. Estudos apontam que diversas aplicações, vêm sendo desenvolvidas para sistemas de Internet das Coisas (*Internet of Things* - IoT), e com o avanço da Indústria 4.0, esse número tende a crescer. Aplicar um pipeline que utiliza de ferramentas de *software* de código aberto (*Free/Libre and Open Source Software* - FLOSS) nesse contexto, permite agregar ao projeto vantagens dessas culturas. Este trabalho visa implementar um pipeline de integração e implantação contínua para uma aplicação de Internet das coisas, seguindo a cultura DevOps e utilizando ferramentas FLOSS para seu desenvolvimento. O pipeline construído consiste em um processo automatizado de controle de versionamento, teste de unidade, análise estática, construção, entrega da aplicação em um repositório público, e ao final ocorre a implantação do *software* no *target* (Raspberry Pi). A aplicação de monitoramento do ambiente, tendo passado pelas etapas do pipeline com sucesso, é implantada no Raspberry Pi. Este trabalho teve como resultado uma estrutura de pipeline com padrões replicáveis, que proporcionou disciplina ao processo de desenvolvimento do *software*. O sistema de monitoramento implantado faz o controle automático da umidade do solo, e disponibiliza as informações do ambiente de forma amigável ao usuário do sistema. Trazer confiabilidade, ao processo de desenvolvimento de *software*, proporciona qualidade ao produto e contribui para a motivação da equipe de desenvolvimento. O pipeline construído, permite melhorias no processo de desenvolvimento de aplicações para dispositivos IoT, especificamente para dispositivos Raspberry Pi, proporcionando benefícios como rápidas entregas do *software* ao usuário final. O trabalho desenvolvido disponibiliza instruções detalhadas, documentadas em um arquivo, viabilizando a replicação da instrutura do pipeline por profissionais de outras áreas.

Palavras-chaves: Pipeline de CI/CD. Integração contínua. Entrega contínua. Implantação contínua. IoT.

Abstract

A software system is usually developed by a team, due to its complexity. The collaborative culture of DevOps, a combination of Development and Operation, aims to improve the software development and deployment process, making it repeatable reliable and automated. A continuous integration and deployment pipeline, consists of the segmentation of the necessary processes from the commit to the software deployment. Studies indicate that several applications have been developed for Internet of Things (IoT) systems, and with the advance of Industry 4.0, this number tends to grow. Applying a pipeline that uses open source software tools (Free/Libre and Open Source Software - FLOSS) in this context, allows adding advantages of these cultures to the project. This work aims to implement a continuous integration and deployment pipeline for an Internet of Things application, following the DevOps culture and using FLOSS tools for its development. The build pipeline consists of an automated process of versioning control, unit test, static analysis, construction, delivery of the application in a public repository, and at the end occurs the software deployment on the target (Raspberry Pi). The environment monitoring application, having passed the pipeline steps successfully, is deployed on the Raspberry Pi. This work resulted in a pipeline structure with replicable patterns, which provided discipline to the software development process. The implanted monitoring system makes the automatic control of the soil humidity, and provides the environment information in a friendly way to the system user. The reliability of the software development process provides quality to the product and contributes to the motivation of the development team. The pipeline built, allows improvements in the process of developing applications for IoT devices, specifically for Raspberry Pi device, providing benefits such as fast software delivery to the end user. The developed work provides detailed instructions, documented in a file, enabling the replication of the pipeline structure by professionals from other areas.

Key-words: CI/CD pipeline. Continuous integration. Continuous delivery. Continuous deployment. IoT.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

IoT *Internet of Things*

MQTT *Message Queuing Telemetry Transport*

CI *Continuous Integration*

IC Implantação Contínua

EC Entrega Contínua

CD *Continuous Deploy*

FLOSS *Free/Libre and Open Source Software*

DevOps *"development" (Dev) e "operations" (Ops)*

GeneSIS *Generation and Deployment of Smart IoT Systems*

M2M *Machine to Machine*

SIS *Smart IoT Systems*

Sumário

1 Introdução

Devido à complexidade dos sistemas de *software* atuais, são montadas equipes para realizar a sua criação. Historicamente, equipes na área de Tecnologia da Informação eram divididas em dois departamentos, o de desenvolvimento e de operação (VALENTE et al., 2020). Devido aos objetivos e incentivos opostos, conflitos entre os setores eram frequentes. A cultura de colaboração *"development" (Dev) e "operations" (Ops)* (DevOps) tem como objetivo eliminar os conflitos entre essas equipes. O processo de produção de soluções para *Internet of Things* (IoT) compartilham esses conflitos (PEREIRA; CARNEIRO; FIGUEIREDO, 2021). Dentro da cultura DevOps são propostos princípios como criação de um processo repetível e confiável para entrega de software, automação de processos e controle de versionamento (VALENTE et al., 2020).

Um pipeline consiste na técnica de segmentação de um processo em vários subprocessos que podem ser executados por unidades dedicadas de forma repetida e sequencial (RAMAMOORTHY; LI, 1977). Com o objetivo de tornar os processos de implantação e entrega da aplicação mais prática e confiável, foi proposto o conceito DevOps. Um movimento que visa unificar as culturas de desenvolvimento e operação de forma colaborativa, visando permitir uma implantação mais rápida e ágil de um sistema (VALENTE et al., 2020). DevOps também defende a automação dos processos de desenvolvimento do projeto.

O termo *Free/Libre and Open Source Software* (FLOSS) utilizado para identificar em igual importância as duas comunidades de *software* livre, sendo essas o movimento do *software* livre (do inglês *free software*) e do código aberto (do inglês *open source*) (FOUNDATION, 2021). Soluções FLOSS são utilizadas em diversos contextos. Utilizá-las na construção de um pipeline segundo a cultura DevOps, possibilita agregar ao projeto as vantagens propostas pelos movimentos.

Internet das Coisas (*Internet of Things* - IoT), de modo geral, refere-se à interconexão em rede dos objetos do cotidiano à *Internet*, sendo uma enorme oportunidade para um grande número de aplicações proporcionar a melhora na qualidade de vida (XIA et al., 2012). Aplicar as práticas propostas pelo DevOps nesse contexto de desenvolvimento, apresenta-se como uma ótima estratégia, para automação de todos os passos necessários para colocar o sistema em produção e garantir o seu correto funcionamento (PEREIRA; CARNEIRO; FIGUEIREDO, 2021). Para alcançar o objetivo de estruturar e automatizar o processo de qualidade, há possibilidade da utilização de práticas de Integração Contínua (*Continuous Integration* - CI) e Implantação Contínua (*Continuous Deploy* - CD), empregar esses conceitos em um projeto constitui a prática CI/CD.

Mediante diversos avanços proporcionados pelos sistemas de *software*, são perceptí-

veis os benefícios dos serviços disponibilizados no nosso cotidiano, falhas que consequentemente ocasionam a ausência desses serviços, podem trazer diversas consequências em nossas vidas. Sendo assim, as etapas de desenvolvimento de um sistema de *software*, desde a criação do código-fonte, até o lançamento da aplicação para produção, requer vários cuidados envolvendo o desenvolvimento e a operacionalização dessa solução, bem como, a integração dessas culturas visando uma implantação ágil do sistema. Demandas do mercado como entregas rápidas de qualidade, podem ser atendidas com o uso dos conceitos de CI/CD, mantendo dessa forma a competitividade. Sendo assim, entender e aplicar com eficiência esses conceitos é de fundamental importância para profissionais e empresas na área de tecnologia.

Aplicar a prática de CI significa obter a construção da automação de testes para módulos do *software*, tendo também a automação do processo de entrega de novas versões possibilitando o *deploy* (LÓPEZ-VIANA et al., 2020). Obter um pipeline que aplique as práticas de CI/CD utilizando projetos FLOSS, se torna um grande desafio no contexto IoT. Este trabalho visa implementar as práticas acima citadas em um pipeline de CI/CD de uma aplicação de Internet das Coisas, que consiste no monitoramento de condições do ambiente a realização da irrigação automática.

O pipeline CI/CD é construído com práticas de integração e implantação contínua que permitem agilizar, automatizar e otimizar a entrega de artefatos de software ao cliente com maior qualidade e menos risco (SABAU; HACKS; STEFFENS, 2021). Ao se implantar um pipeline de CI/CD, pretende-se incorporar atividades conhecidas das práticas de CI/CD, implantando atividades como análise de código estático, construção automática e teste de unidade. Todas essas tarefas são executadas em uma ordem definida de estágios. Após cada estágio, os resultados dos testes são avaliados em um padrão de qualidade, que interrompe o processamento se as condições não forem atendidas. Se todos os padrões de qualidade são atendidos, o artefato de software é armazenado e pode ser acessado e usado por clientes externos. Possibilitando também o retorno do código em produção em sua versão anterior de forma rápida, condição que pode ser necessária se algum erro não foi identificado durante os testes.

Este trabalho possui aos seguintes objetivos específicos:

- **Implementar** princípios propostos pela cultura DevOps em uma aplicação para dispositivos IoT.
- **Construir** uma estrutura de possível replicação no monitoramento de um ambiente para cultivo de plantas.
- **Utilizar** ferramentas FLOSS na construção do pipeline de implantação contínua, apoiando o modelo de desenvolvimento colaborativo de *software*.

- **Identificar** problemas relacionados a implementação de um pipeline automatizado de CI/CD para aplicações IoT.

O pipeline construído permitiu a implantação do *software* no dispositivo Raspberry Pi, em que foram observados resultados eficientes do seu funcionamento no protótipo implementado. Os dados de monitoramento do ambiente são apresentados de forma clara no painel de controle construído. O padrão de implementação do *software* possibilitou a obtenção de métricas para possíveis estudos voltados para eficiência na produção da aplicação. Sendo assim gerados artefatos para replicação dessa solução, sendo os mesmos disponibilizados em um repositório público.

O restante deste trabalho é organizado como se segue. O Capítulo 2 apresenta os conceitos principais da solução e os trabalhos correlacionados. Capítulo 3 são apresentadas as ferramentas utilizadas na construção do trabalho e é detalhada a construção do pipeline CI/CD, bem como, o sistema de monitoramento do ambiente. Logo após, no Capítulo 4 são demonstradas as métricas obtidas e o funcionamento da aplicação, e por fim, no Capítulo 5 são discutidos os resultados e alinhadas expectativas de trabalhos futuros.

2 Revisão bibliográfica

2.1 Conceitos básicos

Nesta seção, serão apresentados conceitos para guiar no entendimento completo da solução criada e entendimento dos mecanismos empregados.

2.1.1 Internet das Coisas

No contexto tecnológico, a Internet das Coisas é um conceito relativamente novo, que vem conquistando novos espaços no mercado, superando desafios tecnológicos ao longo dos últimos anos. Segundo (OLIVEIRA, 2017), a IoT tem como propósito tornar objetos comuns do dia a dia inteligentes e conectados, uma vez que eles sejam capazes de coletar e processar informações do ambiente na qual estão inseridos. Tais objetos, também chamados de *smart objects*, devem possuir capacidade computacional, comunicativa e de processamento de dados junto a sensores (MANCINI, 2017).

2.1.2 FLOSS

Como o termo FLOSS é utilizado para falar das duas comunidades de *software* livre, serão explicados abaixo, os termos utilizados para definir o movimento do *software* livre e do código aberto.

O movimento do software livre (*free software*), apoia a liberdade para os usuários da computação. Um software é definido como "livre", quando são respeitadas uma série de regras (chamadas de liberdades essenciais dos usuários), sendo essas:

- A liberdade de executar o programa como desejado, para qualquer propósito;
- A liberdade de estudar como o programa funciona, e adaptá-lo às necessidades. Para tanto, acesso ao código-fonte é um pré-requisito;
- A liberdade de redistribuir cópias de modo que possa contribuir com a comunidade;
- A liberdade de distribuir cópias de suas versões modificadas a outros.

Desta forma, tem como pré-requisito permitir acesso ao código fonte, e tem como foco a liberdade, sem considerar questões monetárias.

Já ao se identificar uma solução de *software* como código aberto (*open source*), segundo (INITIATIVE, 2020), não se trata apenas do livre acesso ao seu código-fonte. Existem termos de distribuição do *software* de código aberto, sendo estes citados a seguir:

- Redistribuição gratuita do código-fonte e obras derivadas;
- Integridade do código-fonte do autor;
- Nenhuma discriminação contra pessoas ou grupos;
- Não discriminação contra campos de esforço;
- Distribuição de licença;
- A licença não deve ser específica a um grupo;
- A licença não deve restringir outros *softwares*;
- A licença deve ser neutra em termos de tecnologia.

Respeitando assim a liberdade e senso de comunidade dos usuários. Dessa forma os usuários possuem a liberdade de executar, copiar, distribuir, estudar, mudar e melhorar o *software* (FOUNDATION, 2021).

Na prática, algumas licenças de código aberto são mais restritivas, não se qualificando como licenças livres. Dessa forma, existem soluções classificadas e distribuídas como código aberto porém, não são ferramentas de software livre.

2.1.3 Integração, Entrega e Implantação Contínua

Integração contínua CI é uma prática de desenvolvimento de software, na qual membros de um time integram seus trabalhos frequentemente. Geralmente cada pessoa integra seu trabalho pelo menos uma vez no dia, resultando em múltiplas integrações por dia. Cada integração que consiste nas operações de *commit*, *build*, teste de unidade e análise sintática, resultando no *merge* das atualizações ao *branch* principal. Diversos times encontram nesse tipo de abordagem a redução de problemas de integração, como o *merge hell*[1], e permite obter o desenvolvimento coeso de *softwares* mais rapidamente (FOWLER; FOEMMEL, 2006).

Já a prática de Entrega Contínua (EC) consiste na entrega de pequenas atualizações do *software* em produção, resultando em mais de uma entrega de *software* por dia (SAVOR et al., 2016).

No trabalho proposto por (PRENS et al., 2019) são identificados requisitos de CD para dispositivos IoT: Req1: O engenheiro de *software* deve ser capaz de identificar os dispositivos e qual deles será alvo do *deploy*; Req2: O *deploy* do *software* deve ser remoto e semiautomático; Req3: O consumo de energia e o tempo inativo devem ser monitorados

[1] Conhecido como erros, *bugs* e conflitos criados entre as diferentes alterações, integradas ao projeto principal.

para cada atualização instalada; Req4: O engenheiro deve ser capaz de monitorar o dado para cada versão de *software* instalado, por uma interface interativa.

Este trabalho utiliza de ferramentas similares a PRENS et al., para atender a entrega remota e automatizada (Req2). Atendendo de forma mais completa a identificação do *target* (Req1), bem como, o melhor monitoramento de cada passo executado no processo de CI.

No processo de Implantação contínua (CD) todo nova integração de código que chega ao projeto, no repositório principal, entra rapidamente em produção. O processo de fluxo de trabalho quando se usa CD tem as seguintes etapas:

- O desenvolvedor desenvolve e testa na sua máquina local;
- Ele realiza um *commit* e o servidor de CI executa novamente um *build* e os testes de unidade;
- Algumas vezes no dia, o servidor de CI realiza testes mais exaustivos com os novos *commits* que ainda não entraram em produção;
- Se todos os testes passarem, os *commits* entram imediatamente em produção. E os usuários já vão interagir com a nova versão do código.

Esse processo pode trazer vantagens como, reduzir a entrega de novas funcionalidades, receber mais *feedback* dos usuários, não tornar novas implantações um evento doloroso e redução do estresse causado por prazos de implantação, proporcionando assim, a motivação dos desenvolvedores (VALENTE et al., 2020).

Aplicar essas práticas de engenharia de *software* no processo de desenvolvimento de aplicações no contexto dos dispositivos IoT, sendo também utilizadas ferramentas FLOSS na construção da estrutura de desenvolvimento, em que seus processos são automatizados. Agrega ao processo as vantagens resultantes das práticas citadas.

2.2 Trabalhos correlatos

Nesta seção, será apresentada a revisão da literatura realizada. Ademais, serão apresentados trabalhos com objetivo de aplicar soluções no processos de entrega e para garantia da qualidade do *software*.

Em (FERRY et al., 2019) foi apresentada uma solução chamada *Generation and Deployment of Smart IoT Systems* (GeneSIS), um *framework* de CD para sistemas *Smart IoT Systems* (SIS), que permite processamento descentralizado em infraestruturas heterogêneas de dispositivos IoT, borda e nuvem. GeneSIS executa três tipos de *deploy* de artefatos:

1. Sem alteração nos artefatos;
2. Com alteração direcionada dos artefatos permitindo o GeneSIS migrar ou implantar um programa de um hospedeiro para o outro;
3. Executa uma aplicação em um *container* Node-RED[2] dinamicamente adaptável, possibilitando adaptar o aplicativo a sua implantação.

Vemos na abordagem o foco na adaptação da aplicação ao dispositivo *target* durante o processo de *deploy*, dessa forma, visa os processos para garantia da qualidade do *software* desenvolvido para o dispositivo *target*.

No artigo em (YIGITOGLU et al., 2017) é apresentado um *framework*, chamado Foggy, que facilita o provisionamento dinâmico de recursos e automatiza o processo de *deploy* em arquiteturas que trabalham como extensões da arquitetura de nuvem, sendo utilizados dispositivos IoT processando os dados coletados próximos da sua origem. Com o objetivo de aplicar os conceitos, foi implementada uma aplicação de CD utilizando o Raspiberry Pi, tendo como sistema de controle de versão o Git no ambiente Github, na construção do pipeline a ferramenta de CI chamada Concourse[3] e a utilização de *containeres* Docker armazenando as imagens no Docker Hub. Foram identificados requisitos comuns em aplicações IoT como sensibilidade a latência, consumo de largura de banda e cargas de trabalho dinâmicas.

(DORESTE, 2018) implementa um pipeline para apoiar o desenvolvimento contínuo no nicho específico de sistema de *software* para IoT, aplicativos desenvolvidos para Raspberry Pi. Por meio da implementação desse pipeline foi possível observar a prática de desenvolvimento contínuo em cenários IoT, embora problemas intrínsecos a esse nicho, como compatibilidade, limitação das ferramentas e integrações exigidas para tais projetos mereçam atenção da engenharia de software.

No trabalho apresentado em (BELTRÃO; FRANÇA; TRAVASSOS,) foi utilizado *Kubernetes*, para possibilitar o *deploy* em componentes IoT. Com o objetivo de determinar a aplicabilidade da abordagem são utilizados *clusters Kubernetes* em um grupo de dispositivos Raspberry Pi. Os resultados dos testes da abordagem, em cenários que utilizavam protocolos de comunicação síncronos e assíncronos, mostraram que o *cluster* manteve os aplicativos ativos e lidou bem com altas cargas de trabalho no Raspberry Pi. No entanto, foi apoiado o uso de tecnologias com base em *containers*. Sendo uma forma de permitir o a implantação, em ambientes altamente distribuídos como ambientes IoT, similar a forma que foi implementada no trabalho atual.

[2] É uma ferramenta de desenvolvimento baseada em fluxo para programação visual, desenvolvida pela IBM, com o objetivo inicial de conectar dispositivos de *hardware*.<https://nodered.org/>

[3] <https://concourse.ci/>

Para a pesquisa em (ALKHABBAS et al., 2020) foi realizado o levantamento de dados, com o objetivo identificar os principais direcionadores, para escolha de um modelo de *deploy*, praticado em sistemas IoT. Os dados obtidos na pesquisa, foram também utilizados, para analisar o estado prático da engenharia de software na indústria IoT. Para esse propósito foi criado e distribuído um questionário com 35 perguntas respondido por 444 pessoas, sendo esses 66 arquitetos IoT de 18 países distintos. Tendo como primeira descoberta que sistemas de *deploy* utilizam mais recursos de plataformas em nuvem em comparação com recursos de borda da rede. Apresentou que a confiabilidade foi o fator mais influente na escolha do modelo, sendo também identificado fatores como desempenho, segurança e custos. Segundo as respostas obtidas, 73% utilizam o protocolo *Message Queuing Telemetry Transport* (MQTT), sendo o mais utilizado nos sistemas. O protocolo MQTT foi utilizado nesse trabalho, para transmissão das informações entre a aplicação *server* e aplicação *client*.

Em (LÓPEZ-VIANA et al., 2020) foi abordada a aplicação de diferentes estruturas e do fluxo CD, para validação do suporte em aplicações SaaS em agricultura de precisão em áreas isoladas. O protótipo demonstrou ser uma solução de comunicação para dispositivos de borda, flexível o suficiente para diferentes protocolos de comunicação para dispositivos IoT em áreas sem rede de comunicação pública.

Na pesquisa desenvolvida em (PEREIRA; CARNEIRO; FIGUEIREDO, 2021), sendo esta com o objetivo de identificar os trabalhos científicos que dão suporte aos princípios DevOps, no contexto de projetos para IoT, foram apresentadas quatro contribuições, também foram apresentadas duas listas de ferramentas e linguagens de programação para esse contexto. E foi implantado um modelo de pipeline DevOps, para sistemas de *software* IoT. O presente trabalho, tratasse de uma instância do modelo conceitual dessa proposto.

Os trabalhos acima citados têm como objetivo principal a implementação ou pesquisa da utilização de um pipeline de CI/CD, é perceptível na solução de (FERRY et al., 2019) a preocupação da adaptação do processo de *deploy*. Alguns dos desafios ao processo de *deploy*, apresentados em (YIGITOGLU et al., 2017; DORESTE, 2018), servem como pontos a serem observados na implementação de soluções similares. Já os resultados positivos do ponto de vista de comunicação dos dispositivos IoT em (BELTRÃO; FRANÇA; TRAVASSOS, ; LÓPEZ-VIANA et al., 2020), orientado pelas conclusões da pesquisa de (ALKHABBAS et al., 2020) demonstram que a comunicação entre esses dispositivos acontecem de forma satisfatória a atender o processo de *deploy*. Com o aprendizado resultante dos trabalhos citados e selecionando algumas das ferramentas listadas em (PEREIRA; CARNEIRO; FIGUEIREDO, 2021) será proposto no trabalho em questão um pipeline CI/CD para uma aplicação IoT

3 Desenvolvimento

3.1 Ferramentas e tecnologias

Nesta seção serão apresentadas ferramentas e tecnologias utilizadas para construção da solução.

3.1.1 Software

Esta seção apresentará como foco as soluções de *software* utilizadas para construção do presente trabalho.

3.1.1.1 Git

O Git (GIT, 2019) é uma ferramenta para controle de versão de código-fonte muito usada para controlar fluxo de trabalho e gerir atividades em projetos de software, através do controle de versão. Esta ferramenta foi utilizada no controle e versionamento do código da aplicação.

3.1.1.2 GitHub

GitHub (GITHUB, 2022b) é um *website* e um serviço baseado em nuvem que possibilita organizar e maximizar o retorno geral do desenvolvimento de *software*, fornecendo *software* com mais rapidez e eficiência, simultaneamente fortalecendo a segurança e a conformidade. Por proporcionar uma efetiva aplicação dos conceitos principais do trabalho, essa plataforma foi utilizada na construção e automação de alguns dos processos de CI/CD do pipeline.

3.1.1.3 *GitHub Actions*

É uma plataforma que permite automatizar pipelines para construção, teste e entrega de *software*. É possível criar fluxos de trabalho que constroem e testam em todas atividades de *pull request* (processo de integração de alterações nos artefatos da *branch* principal do projeto) ou integração de atualizações do código no ambiente de produção.

O *GitHub Actions* possibilita a aplicação das práticas DevOps, bem como, fluxos de trabalho quando eventos específicos acontecem no repositório definido. Por trabalhar integrado ao GitHub, permitindo que recursos sejam compartilhados (GITHUB, 2022b).

Essas funcionalidades fizeram do *GitHub Actions* a melhor escolha para implantação do processo de execução do pipeline CI/CD.

3.1.1.4 YAML

YAML (TWITTER, 2020) é um padrão de serialização de dados. Sendo amigável a qualquer linguagem de programação, utilizada para comunicação entre pessoas e máquinas. Empregada como:

- Arquivos de configuração;
- Arquivos de log;
- Compartilhamento de dados entre linguagens;

Por possibilitar a criação do arquivo de configuração, responsável por definir a execução dos passos de CI/CD no pipeline, este padrão foi utilizado no presente trabalho. Sendo utilizado para definição de processos automatizados na plataforma *GitHub Actions*, por se tratar do formato de dado determinado como formato padrão. Criando assim os arquivos de configuração de cada *action* (etapa de fluxo de trabalho que contém ações coerentes ao processo de CI/CD).

3.1.1.5 QEMU

O QEMU (DEVELOPERS, 2022) é um emulador de máquina, sendo um *software* de código aberto, rápido e multiplataforma que emula um grande número de arquiteturas de hardware. O QEMU permite executar um sistema operacional completo não modificado (*VM Guest*) sobre um sistema existente (*VM Host Server*).

Por permitir a emulação do sistema compatível com a arquitetura do Raspberry Pi durante a execução do processo de CI no *GitHub Actions*, este emulador foi de grande importância na viabilização do pipeline.

3.1.1.6 Python

O Python (FOUNDATION, 2022a) é uma linguagem de programação com estruturas de dados com auto nível de eficiência, se trata também de uma linguagem de programação interpretada, utilizada no desenvolvimento de *scripts* e robusta o suficiente para realizar grandes implementações. A linguagem possui diversas estruturas de dados já implementadas e uma grande coleção de módulos e pacotes desenvolvidos pela comunidade e disponibilizados através de um gerenciador de pacotes, chamado pip.

Pelo seu bem estruturado gerenciamento de pacotes, e execução em diversas plataformas, foi utilizada para codificação da aplicação *server*, que foi implantada na placa Raspberry Pi.

3.1.1.7 Doctest

Doctest (FOUNDATION, 2022b) é um módulo da linguagem Python que procura por trechos de texto interativos especificados, em seguida, executa essas sessões para verificar se funcionam exatamente conforme o que fora definido. Este módulo possibilita a definição de testes de trechos de códigos, bem como funções e funcionalidades, práticas utilizadas para execução de testes de unidade. Por possibilitar a escrita dos testes de unidade da aplicação *target*, essa ferramenta foi utilizada.

3.1.1.8 Pylint

É um analisador de código estático para Python 2 ou 3. Analisa o código sem executá-lo, procura por erros, reforça padrões de codificação, procura por *code smell*[1], e faz sugestões de como o código deve ser corrigido. Pylint é autoconfigurável e permite a escrita de *plugins* com o objetivo de adicionar seus próprios critérios de análise (PYCQA; CONTRIBUTORS, 2022).

O Pylint foi utilizado na fase de teste do pipeline, não permitindo que a execução do pipeline prosseguisse, sem as devidas correções no código.

3.1.1.9 JSON

O *JavaScript Object Notation* (JSON) (SHIN, 2010) é um formato leve de dados que permite troca de informações simples e rápida entre sistemas. Possui uma semântica de fácil entendimento para as máquinas e legível a humanos, já que é no formato de chave e valor. Foi utilizado como formato para o dado transferido via MQTT.

3.1.1.10 MQTT

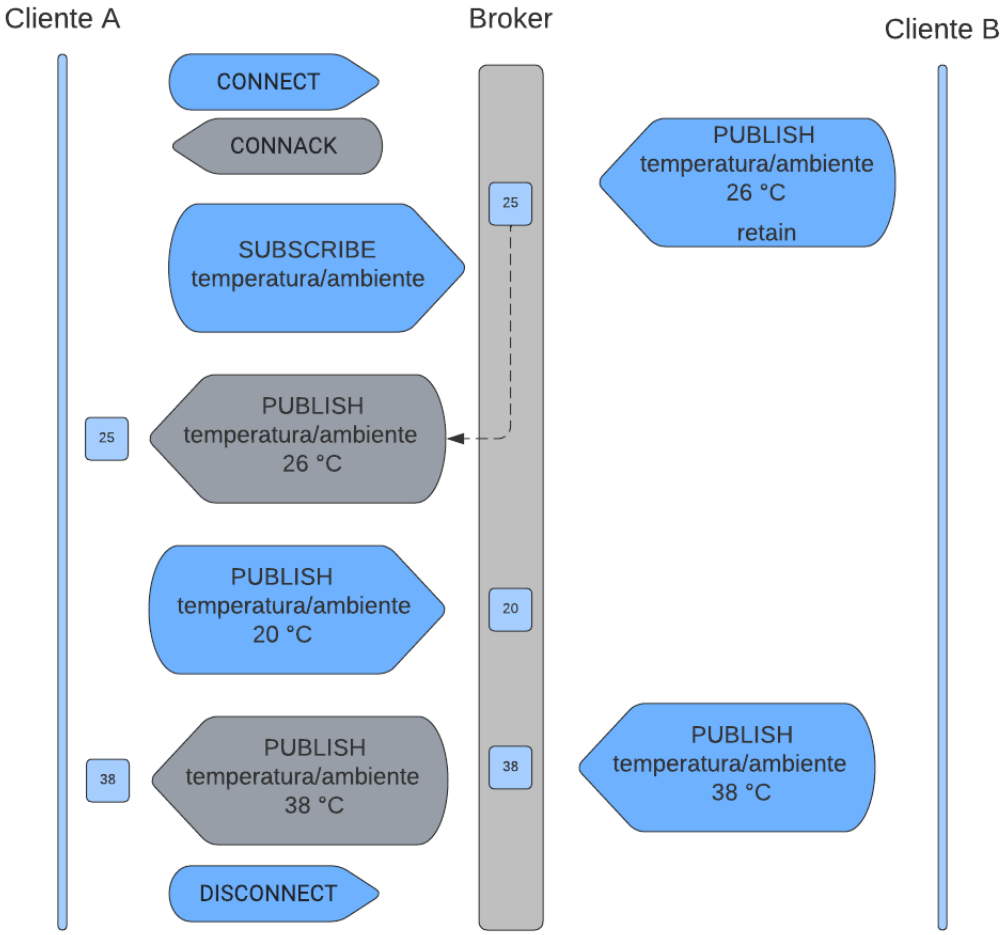
O MQTT (NASTASE, 2017) é um protocolo de transporte de mensagens fundamentado no modelo *publish/subscribe*. É um protocolo leve, aberto e desenvolvido para ser de fácil implementação. Essas características o fazem ser ideal para ser utilizado em diversas situações, tais como comunicação *Machine to Machine* (M2M) e contextos IoT em que a largura de banda é custosa para envio de vários pacotes pequenos.

Um sistema MQTT se baseia na comunicação entre cliente e servidor, em que o primeiro pode realizar tanto "postagens" quanto "captação" de informação e o segundo administra os dados a serem recebidos e enviados. Para isso, é utilizado um paradigma chamado *Publish-Subscribe*. Nesse paradigma possuímos três papéis, o *broker*, o *publisher* e o *subscriber*. Nesse paradigma o *Publisher* enviará mensagens referentes a tópicos, o *Subscriber* irá assinar tópicos para receber mensagens sobre ele e o *Broker* receberá as mensagens dos *publishers* e será responsável em enviar as mensagens para os *subscribers*

[1] <https://martinfowler.com/bliki/CodeSmell.html>

que tenham interesse nesse assunto específico. O protocolo se popularizou pela simplicidade, baixo consumo de dados e pela possibilidade comunicação bilateral (NERI, 2021).

Figura 1



Fluxo de comunicação de dois clientes por meio de um *broker* MQTT. Fonte: Produzido pelo autor.

Ao enviar e receber mensagens, basta instanciar um cliente MQTT, realizar a conexão com o *broker*, e fazer uma subscrição ou publicação em algum tópico determinado. Esse tipo de funcionamento, chamado de *Publish-Subscribe*, é muito utilizado em arquiteturas de microsserviços e em sistemas distribuídos em geral. Também disponibiliza toda a questão de autenticação de acesso, tanto ao *broker* quanto aos tópicos (NERI, 2021). O MQTT foi utilizado como protocolo de comunicação entre o *host* e o *target*.

3.1.1.11 MQTTBox

MQTTBox (UMAPATHY et al.,) é uma aplicação que trabalha com o protocolo MQTT disponível para sistemas Linux, Mac e Windows, utilizada como extensão no

navegador Chrome. É uma das ferramentas IoT usadas para publicação e leitura de dados com tópicos convencionais entre diferentes destinos (UWUMUREMYI, 2021).

O Cliente MQTTBox possibilita:

- Conexão de múltiplos MQTT *brokers* com protocolos TCP ou *Web Sockets*;
- Conectar-se com uma ampla variedade de configurações de conexões do cliente MQTT;
- Publicar e escrever múltiplos tópicos simultaneamente;
- Reconexão do cliente ao *broker*.

Devido essas funcionalidades, o MQTTBox foi utilizado durante período de teste das mensagens enviadas pela aplicação *target*.

3.1.1.12 EMQ X Broker

EMQ X Broker é um *broker* de mensagem MQTT IoT baseado em uma plataforma Erlang/OTP. Erlang/OTP é uma excelente plataforma de desenvolvimento de *software* de tempo real que possui baixa latência e é distribuída. Este *broker* foi projetado para acesso massivo de clientes e realiza roteamento de mensagens rápido e de baixa latência entre dispositivos de rede física em massa (CO, 2022).

O EMQ X Broker foi utilizado no processo de envio das informações entre as aplicações Frontend (*host*) e Backend (*target*) com a implementação do Paradigma *Publish-Subscribe* do MQTT.

3.1.1.13 Docker

Docker (INC, 2021) é um *software* livre para desenvolvimento, envio e execução de aplicações. Permite a separação entre os seus aplicativos e sua infraestrutura para entregas de *software* mais rápidas. Permitindo também o gerenciamento da infraestrutura, as suas metodologias, como a separação de ambientes de execução e rápidas entregas, podem ser aproveitadas para enviar, testar e implantar o código, reduzindo significativamente o atraso entre escrever o código e executá-lo em produção. Docker funciona com uma arquitetura cliente-servidor. O cliente Docker conversa com o *daemon* do Docker, sendo responsável por construir, executar e distribuir seus containeres. O cliente e o *daemon* podem executar no mesmo sistema, ou o *daemon* em um sistema remoto. Os dois se comunicam por uma API REST com outro cliente do Docker e o Docker Compose, permitindo trabalhar com aplicativos que consistem em um conjunto de containeres (INC, 2021). Processo ilustrado na Figura 2.

Figura 2

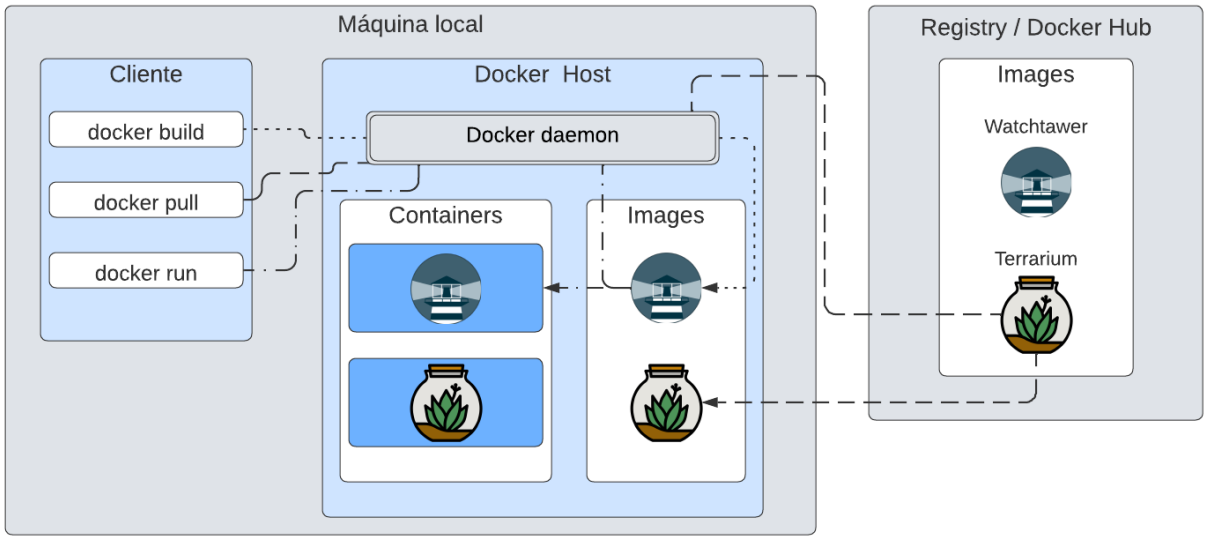


Ilustração da arquitetura Docker. Fonte: Produzido pelo autor.

O Docker cria imagens automaticamente lendo as instruções de um Dockerfile, sendo o Dockerfile um arquivo de texto que contém todos os comandos necessários para construir uma determinada imagem. Possui também módulos que possibilitam desenvolver e executar suas imagens Docker sendo configuradas em um pipeline de CI/CD.

Pelas vantagens proporcionadas no processo de construção e entrega de aplicações, e por sua fácil aplicação no ambiente do pipeline, o Docker foi utilizado neste trabalho.

3.1.1.14 Docker Hub

Docker Hub um registro de serviço em nuvem para compartilhar imagens Docker contendo aplicações. Imagens são distribuídas usando repositórios, os quais permitem o desenvolvimento com o versionamento de imagens e sua manutenção. Através do GitHub é possível realizar a publicação de imagens diretamente no Docker Hub (SHU; GU; ENCK, 2017).

3.1.1.15 Watchtower

O aplicativo Watchtower (DONATH, 2022) monitora os containeres Docker que estão rodando e procura por mudanças nas imagens em seus respectivos registros de imagens, como por exemplo Docker Hub. Se for detectada alguma mudança na imagem, ele irá reiniciar seu container Docker automaticamente utilizando a nova imagem. Com o Watchtower é possível atualizar os containeres das aplicações Docker que estão rodando

apenas publicando uma nova imagem no Docker Hub ou em um próprio registro de imagens.

3.1.1.16 React

React (GACKENHEIMER; PAUL, 2015) é um *framework JavaScript*[2] criado por engenheiros no Facebook para solucionar desafios envolvendo o desenvolvimento complexo de interface de usuários com dados que mudam a todo tempo. Não sendo um trabalho trivial possibilitando não só a manutenibilidade, mas também sendo escalonável. Introduzindo novos paradigmas e mudando o estado atual utilizado para manutenção e criação de aplicações *JavaScript* de interface com o usuário.

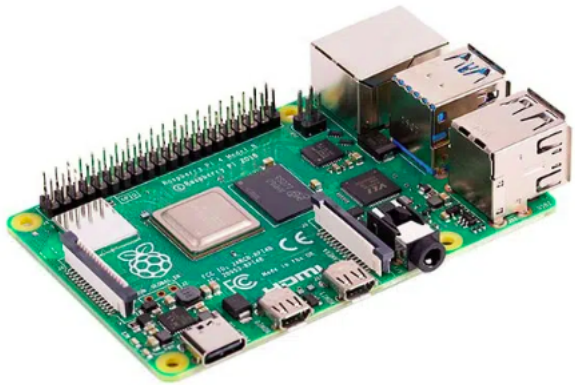
Por possibilitar o desenvolvimento de interface de usuários coerente as necessidades do trabalho, esse *framework* foi escolhido para criação do painel de controle da solução desenvolvida.

3.1.2 Hardware

Nessa seção serão descritos os componentes utilizadas na construção da parte física do projeto.

3.1.2.1 Raspberry Pi

Figura 3



Raspberry Pi 4 Model B. Fonte: (INC, 2021)

O Raspberry (FOUNDATION, 2019) é uma série de microcomputadores desenvolvidos no Reino Unido para facilitar o ensino básico de ciência da computação. Hoje em dia, com sua grande popularização, ele é mundialmente usado em diversas áreas.

[2] Uma linguagem de programação para aplicações web <https://www.javascript.com/>

Por sua portabilidade, conectividade e poder de processamento, foi utilizado a versão Raspberry Pi 4 Model B, visto na Figura 3, para aplicação fonte de envio de informações (*server*) deste trabalho, sendo o dispositivo alvo (*target*) do processo de implantação (*deploy*).

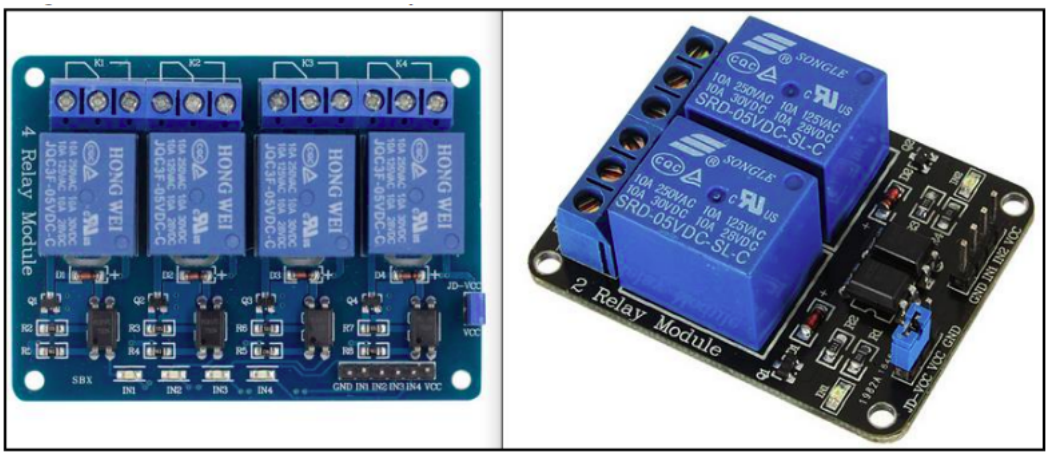
3.1.2.2 Relés

Um relé é um dispositivo eletromecânico que funciona como um interruptor ou chave eletromecânica, ele é acionado quando detecta a passagem de corrente elétrica por meio de uma bobina. Ao aplicar uma tensão sobre a bobina, uma corrente percorre o circuito criando um campo magnético responsável por acionar o sistema de contatos (RIBEIRO, 2005). Este sistema apresentado na Figura 4 pode ser dividido em dois tipos: Normalmente Aberto (NA) e Normalmente Fechado (NF) (FRANCHI, 2018):

- NA: O contato por padrão é aberto, ou seja, até que haja uma ação externa o contato permanece em estado aberto;
- NF: O contato permanece fechado por padrão até que haja uma ação externa que force a troca de estado.

Uma das principais características do relé, é que ele funciona com correntes muito pequenas em relação a corrente que alguns dispositivos exigem. Desta forma, é possível controlar circuitos com correntes altas como motores, 28 lâmpadas e máquinas industriais, diretamente a partir de dispositivos menores como transistores e circuitos integrados (RIBEIRO, 2005).

Figura 4



Módulo relé modelo FL-3FF-S-Z. Fonte: Adaptado de (ELETRôNICOS, 2021)

3.1.2.3 Bomba d'água

A bomba d'água é utilizada na irrigação para levar uma quantidade específica de água de um ponto a outro. Em conjunto a outros dispositivos, ela pode ser programada para funcionar somente em determinados horários ou condições necessárias. O equipamento consiste basicamente em um motor giratório, que possui um número de rotações por minuto. Sua escolha depende do tamanho da área irrigada e alcance de funcionamento, é preciso escolher o equipamento adequado para que o processo de irrigação funcione de maneira correta (SANTOS, 2020). Neste protótipo foi utilizado a bomba d'água submersível modelo Bi0002051.

3.1.2.4 Sensores

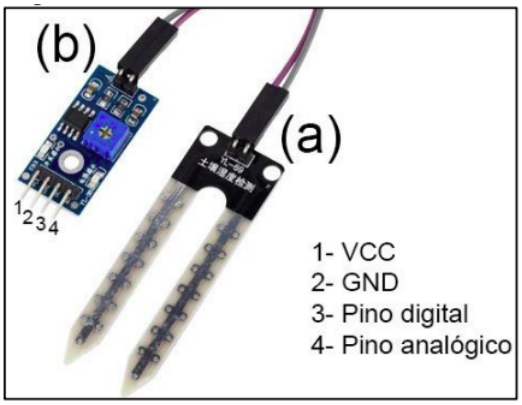
Sensores são equipamentos capazes de captar um determinado sinal dado por um estímulo e responder por meio de um sinal elétrico. Neste contexto, entendesse por estímulo a quantidade, condição ou a forma com que este sinal é detectado e convertido em sinal elétrico. Um sensor tem como saída um sinal na forma de corrente, tensão ou resistência elétrica.

Um sensor de forma isolada não é realmente funcional, já que ele apenas converte uma grandeza física em um sinal elétrico, sendo assim, são normalmente conectados a outros dispositivos que compõem um conjunto de hardware maior. Sua função é extrair informações do ambiente no qual está inserido e convertê-la em um sinal que pode ser interpretado pelo microcontrolador ou sistema da qual faz parte. Após interpretar este sinal, o dispositivo conectado ao sensor tem as informações necessárias para uma tomada de decisão, este processo é realizado pelo microcontrolador e sua ação é executada por algum dos atuadores do sistema (BANZI; SHILOH, 2015).

3.1.2.4.1 Sensores de umidade do solo

O nível de umidade do solo pode ser medido a partir de sensores do tipo capacitivo ou resistivo, que operam respectivamente segundo o princípio da capacitância elétrica e da variação da resistividade do solo (CRUZ et al., 2010; ALVAREZ-BENEDI; MUNOZ-CARPENA, 2004). Sendo empregado nessa solução um sensor resistivo apresentado na Figura 5. Entre as vantagens em utilizar um sensor resistivo, destaca-se o baixo custo de aquisição, manipulação facilitada e a disponibilidade de estudos sobre o mesmo (ALVAREZ-BENEDI; MUNOZ-CARPENA, 2004).

Figura 5



Módulo e sensor de umidade do solo FC-28. Fonte: Adaptado de (ELETRôNICOS, 2021)

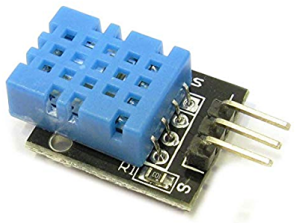
3.1.2.4.2 Sensores de Luminosidade

Para ter controle sobre a luminosidade que incide em um determinado ambiente, normalmente utiliza-se um resistor dependente de luz, do inglês *Light Dependent Resistor (LDR)*. Esse tipo de sensor varia seus valores de resistência conforme a intensidade de radiação eletromagnética que incide sobre ele. Assim, quanto maior for o índice de luz, menor será a resistência lida. Apesar de não operar em variações muito rápidas de luminosidade, como os raios, o sensor LDR é facilmente capaz de detectar se um ambiente está iluminado ou não, além de ter um custo relativamente baixo (LOUREIRO et al., 2018).

3.1.2.4.3 DHT 11

É um módulo complexo com função de monitoramento da temperatura e umidade do ar. Possui uma alta precisão nos valores medidos e garante uma alta confiabilidade na entrega dos dados. O módulo funciona com uma comunicação serial em um único fio. Este módulo envia os dados em pulsos consecutivos em um específico período. Com um pequeno tamanho e baixo consumo de energia, podendo transmitir dados em até 20 metros de distância (SRIVASTAVA; KESARWANI; DUBEY, 2018).

Figura 6



Módulo do sensor de temperatura e umidade DHT11. Fonte: Adaptado de (ELETRôNICOS, 2021)

3.2 Implementação

Nesta seção serão apresentadas as etapas de construção do pipeline CI/CD e aplicação Cliente/Servidor para monitoramento das condições de ambiente.

Para criação de um pipeline com práticas contínuas, foram necessárias definições de aspecto de sistema e infraestrutura.Visando validar a proposta de construção do pipeline, foi desenvolvido pelo autor uma aplicação *server*, que estabelece uma conexão com uma aplicação *client*. Sendo que a aplicação *client* consiste em um painel de controle e monitoramento. As aplicações *server* e *client* estabelecem conexão via *internet*. Dessa forma, torna-se possível ao usuário obter acesso às informações das condições do ambiente geradas pela aplicação *server*, que está em execução na placa Raspberry Pi, e por meio da aplicação *client* ou *Frontend*, lhe é apresentado um painel com os dados de forma gráfica.

A construção da aplicação foi dividida em etapas de forma a aplicar a execução do pipeline CI/CD em cada uma delas, sendo essas etapas:

1. Entrega do monitoramento da umidade do ar e irrigação automática.
2. Adição do monitoramento da luminosidade do ambiente.
3. Adição do monitoramento da temperatura e umidade do ar.

3.2.1 Pipeline CI/CD

Inicialmente serão apresentados os passos realizados para construção do pipeline, que vão de encontro as estratégias de qualidade e controle dos artefatos da aplicação.

Para implementação da solução, são necessários requisitos iniciais para implementar e manipular as plataformas e ferramentas.

Foi utilizada o editor Visual Studio Code[3], na sua versão 1.68, para edição e criação dos arquivos e estrutura de pastas da aplicação. O sistema de controle de versionamento Git, foi instalado no ambiente de desenvolvimento em sua versão 2.3. Para execução da aplicação *server* foi instalado no *target* (placa *Raspberry pi* subseção 3.1.2.1) um Cliente Docker, possibilitando a construção e execução de diferentes containeres. Para utilização das funcionalidades na plataforma GitHub, e dos pacotes do GitHub *Community Edition* foi criado uma conta na plataforma e definido um projeto chamado terrariumtarget, referente a aplicação *server* ou *target* que foi utilizada como foco na implementação do pipeline CI/CD.

3.2.1.1 Arquitetura do pipeline

Seguindo os princípios para entrega de *software*, foi definido uma estrutura para construção do pipeline, representada na Figura 7. Em que o desenvolvedor codifica a aplicação, e através da execução do *push* utilizando o Git, envia os artefatos no repositório principal chamado *main*. Essa ação inicia a execução dos processos automatizados do pipeline no *GitHub Actions*, no qual são executados os seguintes passos:

- Construção da imagem Docker, predefinida no arquivo Dockerfile, podendo usar definições de uma imagem já existente;
- Realização da análise sintática, a execução do container Docker e dos testes de unidade;
- Publicação da imagem Docker, no registro de imagens Docker Hub, e notificação ao desenvolvedor através de um e-mail.

Por fim, o container Watchtower em execução no Raspberry Pi (*target*), por meio do monitoramento de atualização das imagens no *target*, monitoramento que ocorre periodicamente de hora em hora, efetuando a atualização do container da aplicação, mediante mudanças da imagem em registro no Docker Hub. Todos esses passos se encontram representados na Figura 7. O arquivo YAML responsável pelo processo de automação no *GitHub Actions*, se encontra detalhadamente explicado em Materiais elaborados pelo autor.

[3] <https://code.visualstudio.com/>

Figura 7

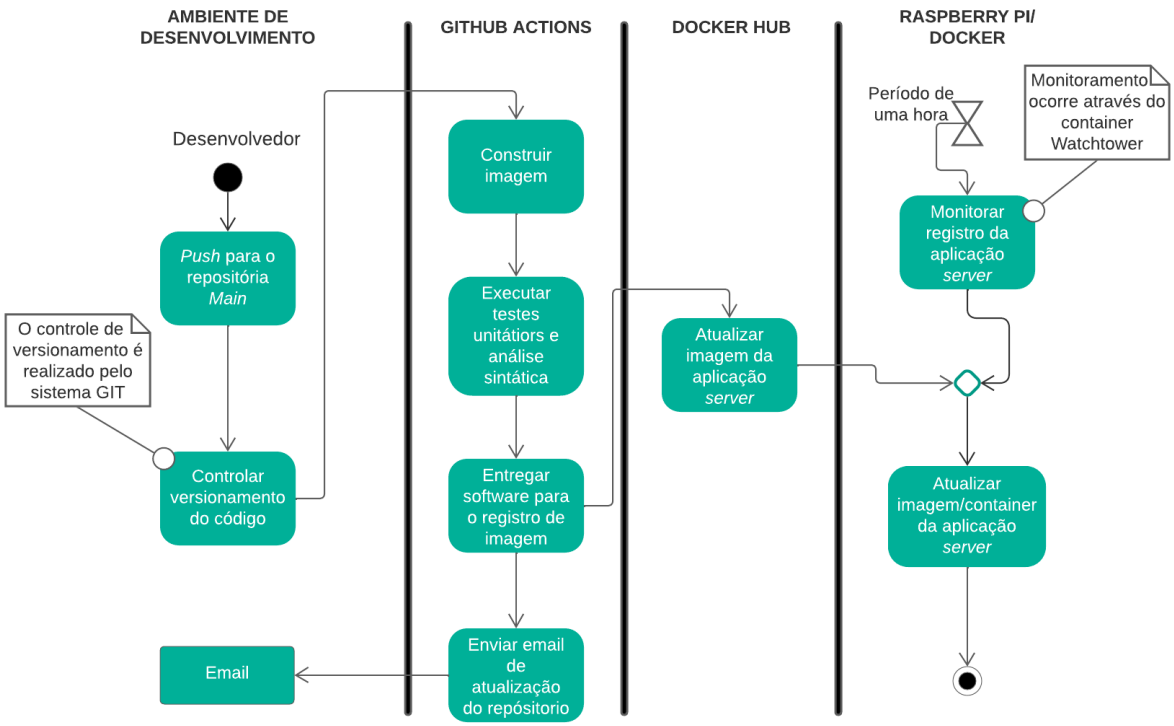


Diagrama de atividade UML. Fonte: Produzido pelo autor

3.2.1.2 Construção do pipeline

Para construção da imagem definida no arquivo Dockerfile, foi realizada pesquisa para garantir a execução da aplicação na arquitetura correspondente ao *target* (Raspberry Pi). Também por esse motivo foi escolhido a linguagem de programação Python, se tratando de uma linguagem interpretada apresenta maior adaptabilidade ao ambiente de execução.

Para definição dos passos do pipeline executadas na plataforma *GitHub Actions*, foi utilizado um arquivo YAML, que especifica desde a construção da imagem até seu processo de publicação.

Foi realizado o processo de escrita dos testes de unidade, mediante estabelecimento de algumas das unidades de código necessárias para o correto funcionamento da aplicação, sendo essas implementadas pelo autor, bem como, a definição de seus comportamentos esperados. Para o processo de escrita dos testes de unidade foi utilizado o módulo Doctest, em que os comportamentos definidos são observados e conferidos. Para teste da escrita do código conforme boas práticas, foi executado também a análise sintática do código com a ferramenta Pylint.

Sendo que ao final da execução do pipeline no *GitHub Actions* e executado o processo de entrega (*delivery*) da imagem da aplicação na plataforma Docker Hub, tendo essa imagem passada pelos processos de construção (*build*) e teste (*test*) sem apresentar erros. Cada parte do arquivo YAML utilizado para definir o pipeline se encontra detalhado no capítulo A. É importante destacar, que comportamentos fora do especificado nos testes e erros detectáveis ocorridos durante a passagem código pelo pipeline, interrompem a sua execução. E com a interrupção do pipeline que apresentou anomalias, a imagem com possíveis problemas é impedida de ser implantada no dispositivo *target*. Por fim, o Watchtower realiza o processo de atualização do container devida alteração da imagem no registro Docker Hub. Executando assim o processo de Implantação Contínua (IC) ou CD.

3.2.2 Desenvolvimento do protótipo para monitoramento do ambiente de irrigação automática

Durante o planejamento do protótipo foi criado um diagrama de implantação, mostrado na Figura 8, para definir em uma visão geral, os principais componentes para construção do protótipo. Na Figura 8 o atuador de irrigação representa o relé, no que lhe concerne atua a bomba, para irrigação do solo. Para estruturar o protótipo objeto deste trabalho foram definidos os requisitos do sistema, os quais possuem duas classificações: requisitos funcionais, que definem como o sistema deverá atuar a partir de diferentes entradas de dados (funcionamento); e requisitos não-funcionais, que especificam como o sistema deve ser, explicitando suas propriedades e restrições.

3.2.2.0.1 Requisitos Funcionais

1. O sistema deve ser capaz de perceber a umidade do solo;
2. O sistema deve ser capaz de irrigar o solo;
3. O sistema deve perceber luminosidade;
4. O sistema deve aferir a temperatura do ambiente;
5. O sistema deve possibilitar acesso remoto às informações monitoradas.

3.2.2.0.2 Requisitos Não-Funcionais

1. As informações sobre as condições do ambiente devem ser atualizadas em tempo real, e com baixa latência;
2. Os valores aferidos devem ser fáceis de visualizar;

Figura 8

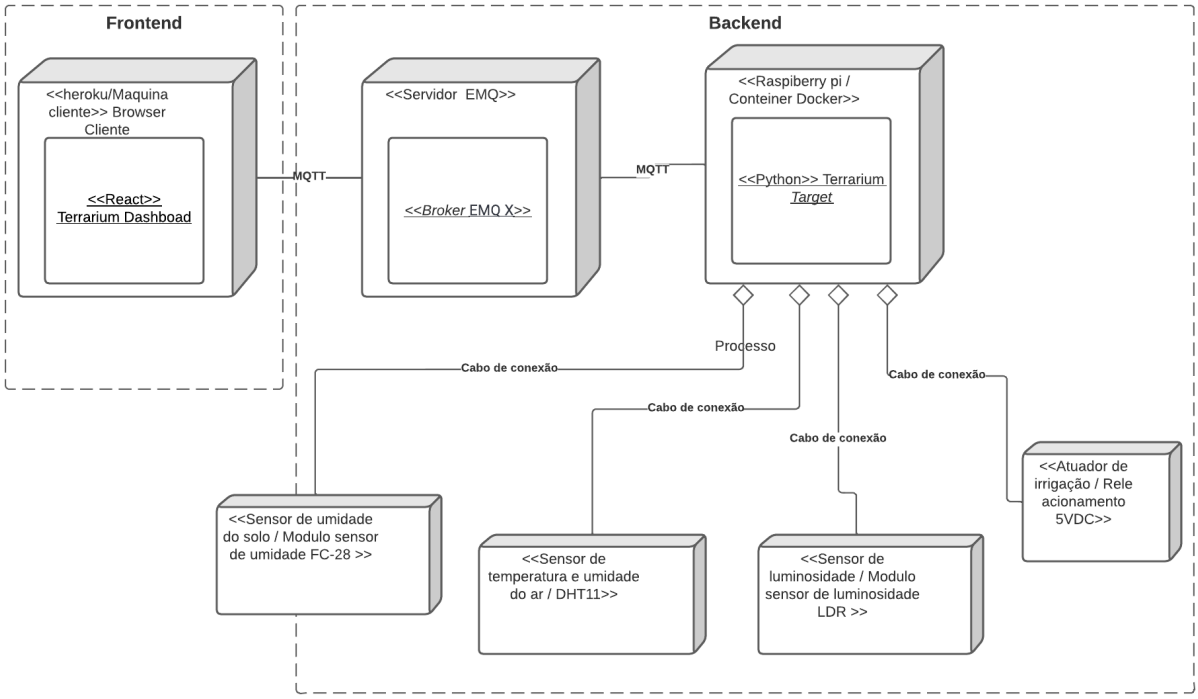


Diagrama de implantação UML. Fonte: Produzido pelo autor

3.2.2.1 *Hardware*

De forma a atender alguns dos requisitos estabelecidos, foi realizada a simulação do circuito proposto, utilizando a plataforma Fritzing[4], sendo possível a análise e reavaliação de componentes a serem utilizados. Imagem da simulação pode ser visualizada na Figura 9.

Com a definição dos componentes necessários, foi realizada uma pesquisa nas lojas virtuais em busca dos componentes especificados na lista da Tabela 1.

Componente	Quantidade
1	Raspberry Pi
1	Sensores de umidade do solo FC-28
1	Bomba d'água 5 VCC
1	Relés 5 VCC
1	DHT 11
1	Sensores de Luminosidade KY-018
1	Fonte de alimentação 5 VCC

Tabela 1

Lista dos componentes utilizados para aplicação. Fonte: Produzido pelo autor

[4] <https://fritzing.org/>

Figura 9

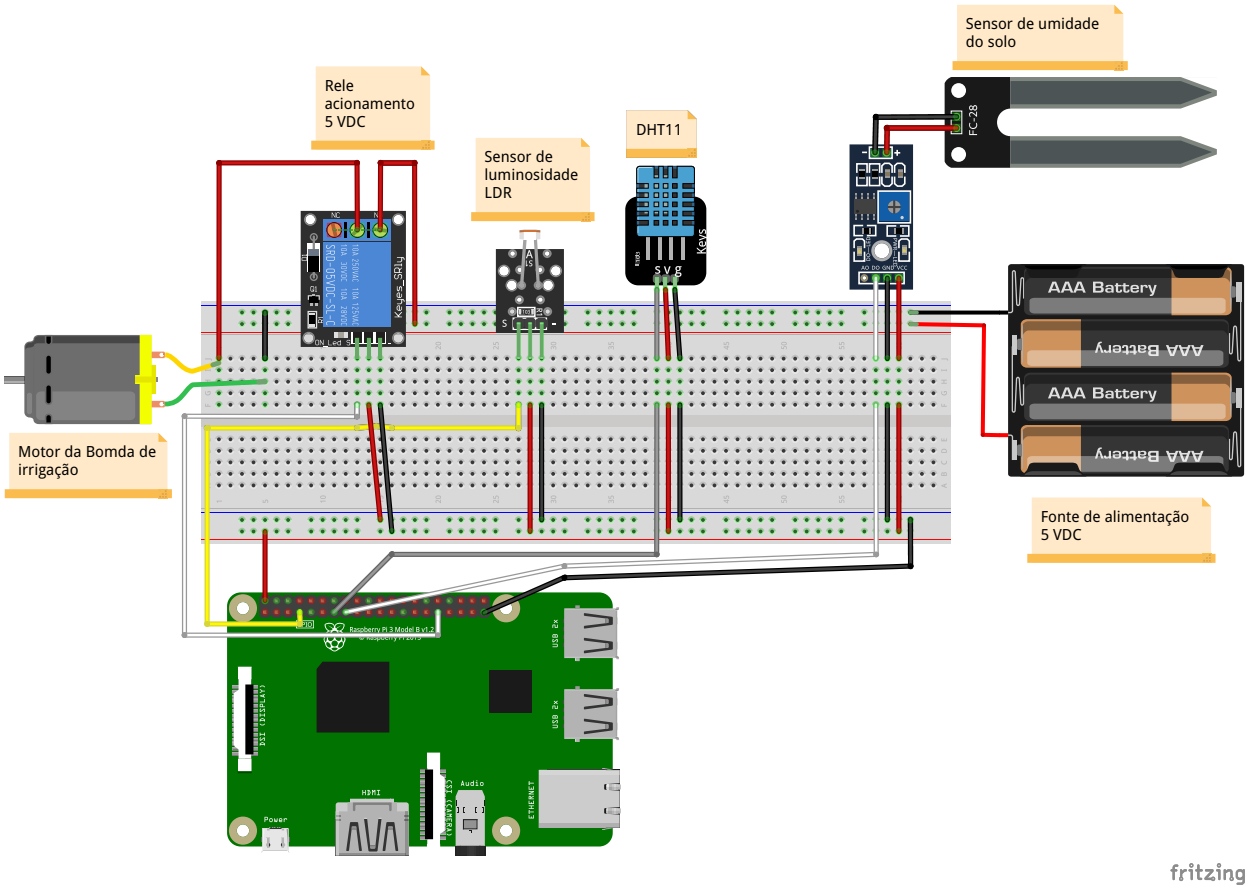


Imagem digital dos componentes e ligações. Fonte: Produzido pelo autor

De posse dos componentes foram montadas as ligações sobre uma *protoboard*. A conexão entre a placa Raspberry Pi e os sensores foi realizada por meio de cabos de cobre, e a conexão do Raspberry Pi com a *Internet* foi estabelecida via *wifi*.

3.2.2.2 *Software*

A aplicação *server* foi desenvolvida na linguagem Python, citada na subseção 3.1.1.6. Na implementação da comunicação entre o módulo luminosidade LDR (3.1.2.4.2) e do módulo do sensor de umidade do solo (3.1.2.4.1) com a Raspberry Pi foi utilizada a biblioteca gpiozero[5]. E no processo de aferição da temperatura do ambiente com o sensor DHT 11, foi utilizada a biblioteca Adafruit-DHT[6], sendo também possível se obter dados relativos à umidade relativa do ar no ambiente monitorado.

Durante o desenvolvimento da aplicação *server*, que roda na placa Raspberry Pi (*target*) foi utilizada a ferramenta MQTTBox, o que possibilitou o teste no envio das

[5] <https://gpiozero.readthedocs.io/en/stable/>
[6] <https://pypi.org/project/Adafruit-DHT/>

mensagens via MQTT. Para estabelecer a comunicação entre os dois dispositivos via MQTT foi utilizado o EMQ X Broker, sendo a comunicação implementada utilizando a biblioteca Paho MQTT[7].

Foram também implementadas regras de negócios, com o objetivo de acionar a irrigação de forma automática, sendo elas:

- Solo seco e presença de luz;
- Solo seco e horário entre os intervalos das 6:00:00 às 6:59:59 ou intervalo 18:00:00 às 18:59:59.

O processo de criação da aplicação (*server*) foi realizado paralelo ao processo de construção do pipeline. Por meio da estratégia de desenvolvimento por etapas, sendo essas definidas no início da seção 3.2, através dessas etapas foi possível aplicar e aprimorar o funcionamento e execução do pipeline CI/CD.

Para o desenvolvimento do *Dashboard* (aplicação *client*) foi utilizado o *framework* React, no qual foi utilizado o *broker* EMQ X Broker para envio e recebimento dos dados via MQTT. Possibilitando a construção de gráficos em tempo real dos dados aferidos pela aplicação *server*.

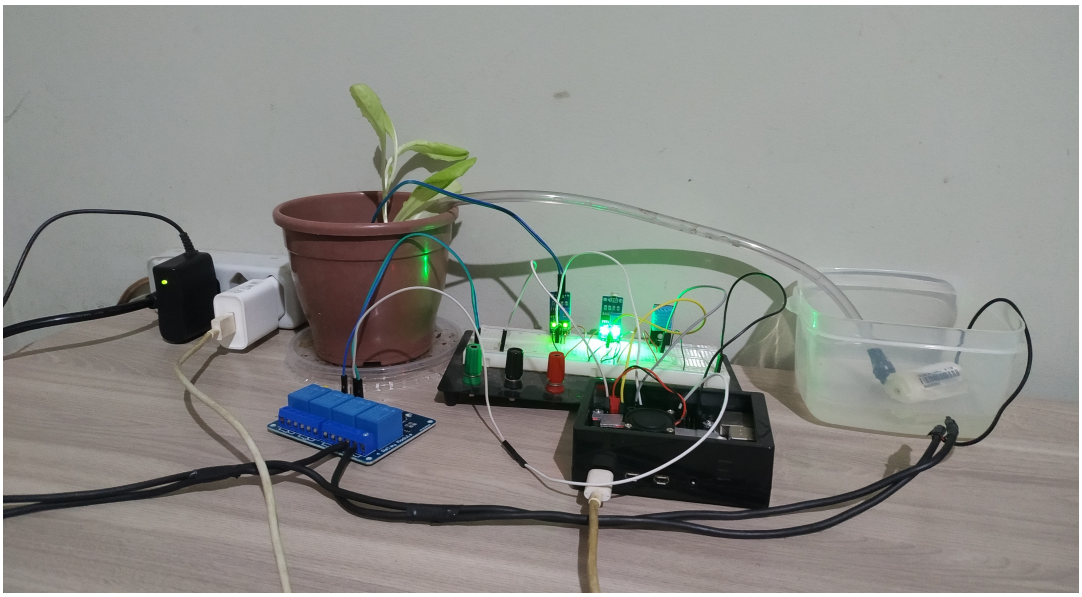
[7] <https://pypi.org/project/paho-mqtt/>

4 Resultados

O *software* para monitoramento do ambiente e irrigação do solo se mostrou funcional e prático. Sua inicialização é rápida, e ele inicia a leitura das condições de ambiente e as envia pela rede assim que é ligado, possibilita em poucos segundos a visualização dos dados ao cliente.

A irrigação ocorre de forma eficiente se tratando do protótipo proposto, sendo possível manter um nível de umidade no solo suficiente a possibilitar o desenvolvimento da planta. Sendo que fatores importantes, como a luminosidade e a temperatura, quais a planta está sendo exposta, podem ser adequado pelo cliente por meio da disponibilidade do monitoramento.

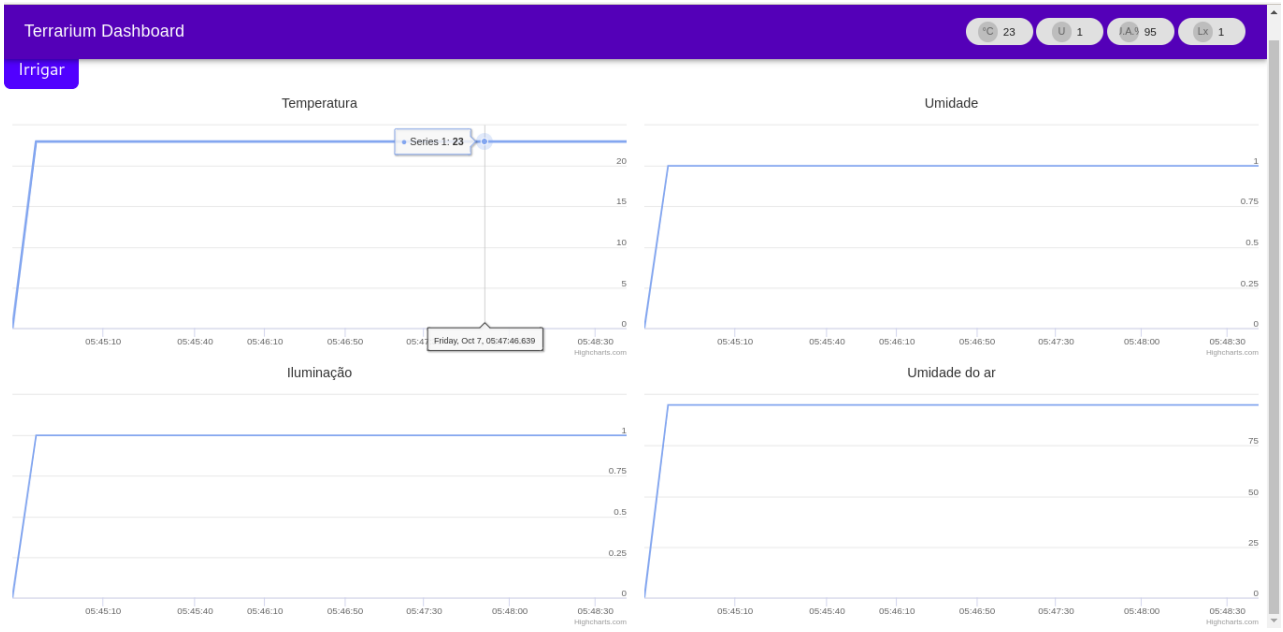
Figura 10



Protótipo de irrigação e monitoramento. Fonte: Produzido pelo autor

O *dashboard* construído possibilita a fácil visualização dos dados gerados em tempo real, por meio de gráficos de fácil entendimento disponibilizando a hora e data de cada valor aferido. Com um *display* inteligente é possível verificar o valor no momento desejado ao deslizar o mouse sobre a linha do gráfico.

Figura 11

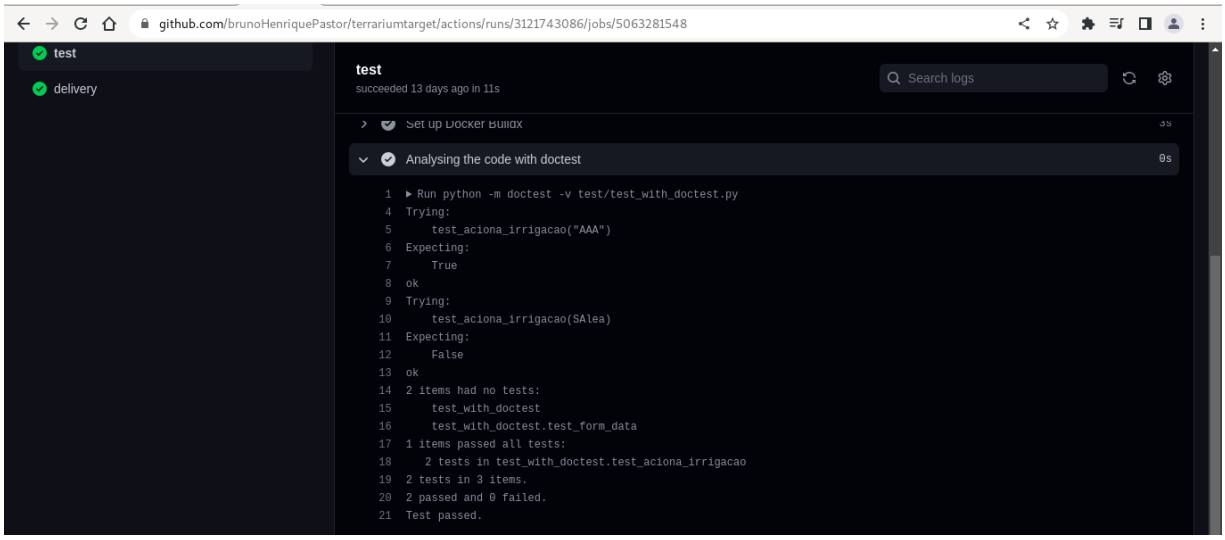


Painel de monitoramento web. Fonte: Produzido pelo autor

A aplicação do pipeline CI/CD possibilitou estruturar um processo sistemático que favorece a disciplina e permite a obtenção de métricas que ajudam a quantificar as abordagens no processo de desenvolvimento, operação e manutenção do *software*. Sendo essas definições, para engenharia de *software*, elaboradas pelo Instituto de engenheiros eletricistas e eletrônicos (IEEE) (PRESSMAN; MAXIM, 2021).

No painel do *GitHub Actions* é possível acompanhar o status de sucesso de cada processo estabelecido. Um exemplo desse painel pode ser visto na Figura 12, informações estas geradas durante a execução de cada processo na plataforma. O acompanhamento de cada estágio de execução é de fácil visualização. A interrupção no processo do pipeline quando são apresentados erros, principalmente quando falamos dos testes unitários implementados na estrutura, possibilita um maior controle e correção destes por uma equipe, o que efetivamente reduz a possibilidade de entrega da aplicação com erros.

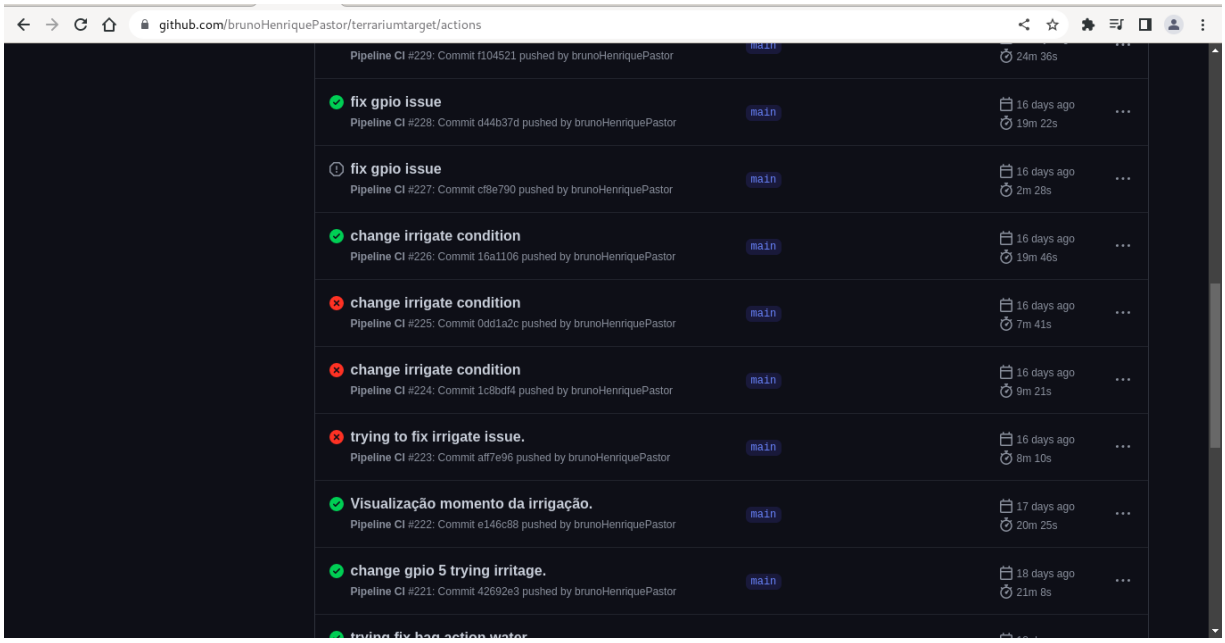
Figura 12



Painel com detalhamento do teste de unidade. Fonte:(GITHUB, 2022a).

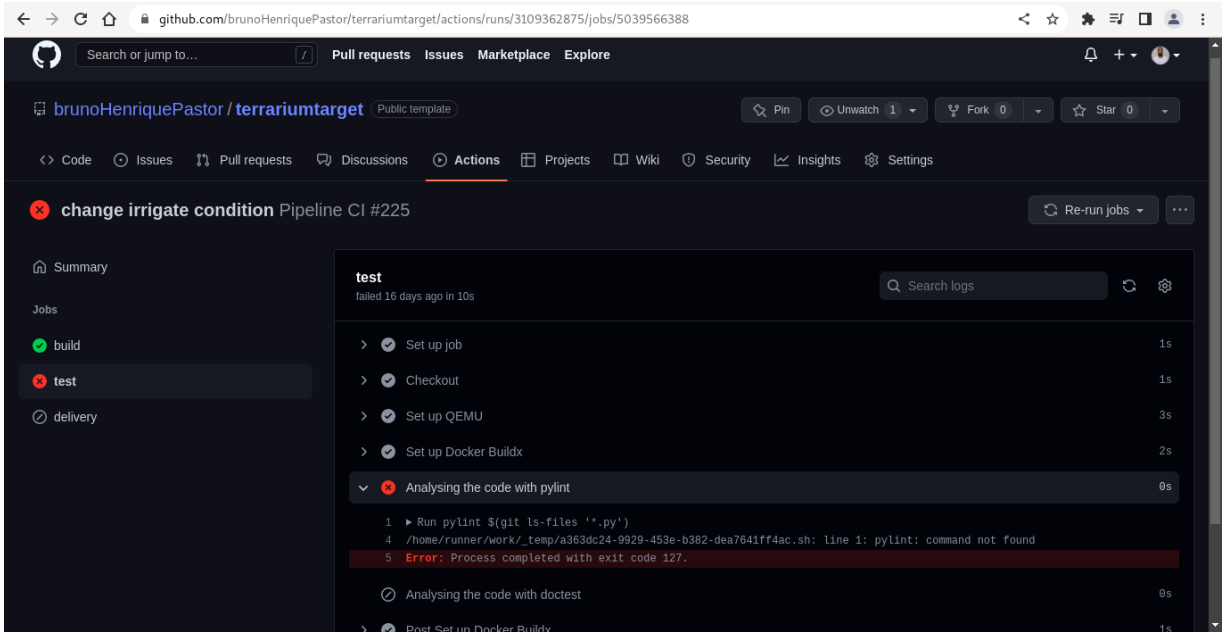
Os processos do pipeline são replicáveis e são estabelecidos como um padrão para cada *commit* ao projeto da aplicação, sendo que durante esses *commits*, em que ocorre o desenvolvimento do *software*, bem como sua manutenção. A submissão de atualizações que acarretavam erros foram facilmente identificadas, e seus erros podem ser visualizados possibilitando sua correção, podendo ser vistas nas Figuras 13 e 14. O arquivo YAML teve todas suas linhas de código explicadas, criando o arquivo disponibilizado em Materiais elaborados pelo autor, permitindo seu fácil entendimento e replicação.

Figura 13



Painel de status de cada pipeline. Fonte:(GITHUB, 2022a).

Figura 14



Painel de detalhamento da execução de cada processo. Fonte:(GITHUB, 2022a).

O processo de CI/CD possibilita escalabilidade, podendo alcançar um grande número de dispositivos, sendo uma vez instaladas as dependências necessárias nos dispositivos de destino, o acesso à imagem no Docker Hub pode ser feito por várias máquinas, a

implantação automatizada pode acontecer nesses ambientes simultaneamente.

Na construção do pipeline CI/CD e do sistema de monitoramento do ambiente foram empregadas algumas soluções FLOSS, suas licenças se encontram descritas na Tabela 2, apoiando a cultura FLOSS e incentivando sua utilização em casos de replicação do presente trabalho.

Nome	Site	Licença
Git	<https://git-scm.com/>	GNU General Public License v2 (GPLv2)
GitHub	<https://github.com/>	Creative Commons Zero v1.0 Universal (CC0 1.0)
YAML	<https://yaml.org/>	Apache License 2.0
QEMU	<https://www.qemu.org/>	GNU General Public License v2 (GPLv2)
Python	<https://www.python.org/>	Python 3.4.0 (PSL 3.4)
Pylint	<https://pypi.org/project/pylint/>	GNU General Public License v2 (GPLv2)
EMQ X Broker	<https://docs.emqx.com/>	Apache License 2.0
Paho MQTT	<https://pypi.org/project/paho-mqtt/>	Eclipse Distribution License 1.0 (BSD)
Adafruit IO	<https://io.adafruit.com/>	MIT
Docker	<https://www.docker.com/>	Apache License 2.0
Watchtower	<https://containrrr.dev/watchtower/>	Apache License 2.0
React	<https://pt-br.reactjs.org/>	MIT

Tabela 2

Ferramentas, sites oficiais e licenças. Fonte: Produzido pelo autor.

Métricas fornecidas pela plataforma podem ser utilizadas como base de estudos para melhorias no processo de desenvolvimento do *software*, é apresentado na Figura 15, um gráfico que representa a quantidade de código deletado (dados em vermelho) e inserido (dados em verde) dentro de um período. Ao analisarmos as datas 08/21 e 09/11 (mês/dia), em que foram liberadas as funcionalidade de monitoramento de luminosidade, bem como, temperatura e umidade do ar, respectivamente. É possível observar o comportamento compatível com a prática de desenvolvimento proposta, na qual frequentes integrações são executadas, o que pode ser visto na Figura 16. Ademais, ao se comparar as mesmas datas na Figura 15, é perceptível que a quantidade de código removido é similar a quantidade inserida. Fato esse, que pode indicar um comportamento de refatoração de código, uma prática que não favorece a produtividade no processo de desenvolvimento de *software*.

Figura 15

Gráfico de linhas de código incluídos e excluídos por data. Fonte:(GITHUB, 2022a).

Figura 16

Gráfico de número de *commits* por data. Fonte:(GITHUB, 2022a).

Na construção do pipeline, foram enfrentadas dificuldades, sendo uma delas, o processo automatizado de integração da aplicação. Pois, devido a arquitetura do dispositivo IoT utilizado, a placa Raspberry Pi, não ser compatível com a utilizado no ambiente de execução da plataforma *GitHub Actions*, não permitiu a execução da imagem Docker com a aplicação *server*. Foi necessário a realização de pesquisar mais específicas de forma a encontrar uma solução, o que resultou na escolha da utilização do emulador QEMU. Tornando possível, a execução da imagem que contém a aplicação, no ambiente de teste do GitHub.

5 Considerações Finais

5.1 Contribuições

A utilização de novas tecnologias tem como o objetivo trazer confiabilidade e reduzir a ocorrência de possíveis erros já conhecidos. A construção do Pipeline estabelece um padrão para produção da aplicação de monitoramento, automatizando processos que antes precisavam ser executados manualmente por um desenvolvedor. As preocupações com a execução da aplicação em um ambiente de teste e entregas contínuas do *software*, se encontram segundo a cultura DevOps. Problemas encontrados foram solucionados na construção de um ambiente de teste compatível com o de produção, contribuem para replicação do modelo proposto.

Trazer qualidade de vida para as pessoas é muito importante quando se é proposto a construção de novas soluções. Apoiar a produção de alimentos de forma inteligente contribui para avanços no mercado. Este trabalho gerou um sistema de monitoramento do ambiente e irrigação automática, podendo muito bem ser empregado no cultivo de alimentos. Os artefatos e ferramentas disponibilizados no repositório <https://github.com/brunoHenriquePastor/terrariumtarget> possibilitam sua replicação.

Em equipes de trabalho, manter a motivação é um critério importante para geração de bons resultados. Propor e implantar processos que permitam facilitar novas entregas de uma equipe, pode influenciar diretamente na motivação de um grupo.

Desta forma, os objetivos gerais do trabalho foram atendidos, sendo que o pipeline construído possibilita a implantação de uma aplicação IoT em um ambiente de produção garantindo seu correto funcionamento e execução das atividades esperadas.

5.2 Trabalhos Futuros

Com o pipeline implementado foi possível realizar testes importantes para o funcionamento da aplicação porém, ao se adicionar algum tipo de teste de integração, como por exemplo o teste de interfase. Sendo uma categoria de teste que permite maior cobertura da solução, é possível favorecer a reusabilidade e permitir que maiores problemas sejam identificados. No terceiro requisito citado no trabalho de (PRENS et al., 2019), é levantada a necessidade do monitoramento do consumo de energia e o tempo de inatividade durante o processo de atualização de um novo *software*, em uma abordagem para dispositivos IoT a energia consumida é sempre um critério que pode inviabilizar uma solução. Sugere-se que sejam armazenados os dados referentes ao consumo de energia do dispositivo *target* e

tempo de inanição da aplicação, durante a realização do *deploy*.

Com o protótipo inicial, a confecção de uma placa para os componentes permite a redução de falhas por mal contato. E a implantação em um ambiente real de produção de alimentos permitirá a validação da solução de irrigação criada. Os dados de umidade do solo e da luminosidade obtidas pela solução, são medidas digitais, devido limitações dos sensores utilizados. Alterar esse modelo de dado para medidas analógicas, como taxa de luminosidade, permitirá obter informações para base de estudos na melhoria do cultivo de plantas.

Referências

ALKHABBAS, F. et al. On the deployment of iot systems: An industrial survey. In: IEEE. *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*. [S.l.], 2020. p. 17–24. Citado na página 22.

ALVAREZ-BENEDI, J.; MUNOZ-CARPENA, R. *Soil-water-solute process characterization: an integrated approach*. [S.l.]: CRC Press, 2004. Citado na página 31.

BANZI, M.; SHILOH, M. *Primeiros Passos com o Arduino–2ª Edição: A plataforma de prototipagem eletrônica open source*. [S.l.]: Novatec Editora, 2015. Citado na página 31.

BELTRÃO, A. C.; FRANÇA, B. B. N. de; TRAVASSOS, G. H. Performance evaluation of kubernetes as deployment platform for iot devices. Citado 2 vezes nas páginas 21 e 22.

CO, E. T. *EMQ X Broker*. 2022. <https://www.emqx.io/>. Accessed: 2022-03-10. Citado na página 27.

CRUZ, T. M. et al. Avaliação de sensor capacitivo para o monitoramento do teor de água do solo. *Engenharia Agrícola*, SciELO Brasil, v. 30, p. 33–45, 2010. Citado na página 31.

DEVELOPERS, T. Q. P. *About QEMU*. 2022. <https://www.qemu.org/>. Accessed: 2022-02-01. Citado na página 24.

DONATH, M. *Introduction*. 2022. <https://containrrr.dev/watchtower/introduction/>. Accessed: 2022-03-04. Citado na página 28.

DORESTE, A. C. de S. *PIPELINE DE IMPLANTAÇÃO CONTÍNUA NO CONTEXTO DE INTERNET DAS COISAS PARA RASPBERRY PI*. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2018. Citado 2 vezes nas páginas 21 e 22.

ELETRôNICOS, F. C. *FILIPEFLOP*. 2021. <https://www.filipeflop.com/>. Accessed: 2022-05-08. Citado 3 vezes nas páginas 30, 32 e 33.

FERRY, N. et al. Genesis: Continuous orchestration and deployment of smart iot systems. In: IEEE. *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. [S.l.], 2019. v. 1, p. 870–875. Citado 2 vezes nas páginas 20 e 22.

FOUNDATION, I. F. S. *O Sistema Operacional GNU*. 2021. <https://www.gnu.org/>. Accessed: 2022-03-02. Citado 2 vezes nas páginas 15 e 19.

FOUNDATION, P. S. *O tutorial de Python*. 2022. <https://docs.python.org/pt-br/3/tutorial/index.html>. Accessed: 2022-02-04. Citado na página 24.

FOUNDATION, P. S. *doctest — Test interactive Python examples*. 2022. <https://docs.python.org/3/library/doctest.html>. Accessed: 2022-02-04. Citado na página 25.

FOUNDATION, T. R. P. *Teach, Learn, and Make with Raspberry Pi*. 2019. <https://www.raspberrypi.org/>. Accessed: 2021-08-08. Citado na página 29.

FOWLER, M.; FOEMMEL, M. *Continuous integration*. 2006. Citado na página 19.

FRANCHI, C. M. *Acionamentos elétricos*. [S.l.]: Saraiva Educação SA, 2018. Citado na página 30.

GACKENHEIMER, C.; PAUL, A. *Introduction to React*. [S.l.]: Springer, 2015. v. 52. Citado na página 29.

GIT. *Git*. 2019. <https://git-scm.com/book/>. Accessed: 2022-02-04. Citado na página 23.

GITHUB, I. *GitHub*. 2022. <https://github.com/>. Accessed: 2022-06-20. Citado 3 vezes nas páginas 42, 43 e 45.

GITHUB, I. *Understanding GitHub Actions*. 2022. <https://docs.github.com/>. Accessed: 2022-02-04. Citado na página 23.

INC, D. *Docker overview*. 2021. <https://docs.docker.com/get-started/overview/>. Accessed: 2022-03-08. Citado 2 vezes nas páginas 27 e 29.

INITIATIVE, O. S. *Open Source Initiative*. 2020. <https://opensource.org/history>. Accessed: 2022-03-04. Citado na página 18.

LÓPEZ-VIANA, R. et al. Continuous delivery of customized saas edge applications in highly distributed iot systems. *IEEE Internet of Things Journal*, IEEE, v. 7, n. 10, p. 10189–10199, 2020. Citado 2 vezes nas páginas 16 e 22.

LOUREIRO, J. F. et al. Automação de estufa agrícola integrando hardware livre e controle remoto pela internet. *Revista de Computação Aplicada ao Agronegócio*, v. 1, n. 1, p. 38–55, 2018. Citado na página 32.

MANCINI, M. Internet das coisas: História, conceitos, aplicações e desafios. *Project Management Institute–PMI*, 2017. Citado na página 18.

NASTASE, L. Security in the internet of things: A survey on application layer protocols. In: IEEE. *2017 21st international conference on control systems and computer science (CSCS)*. [S.l.], 2017. p. 659–666. Citado na página 25.

NERI, M. L. e. G. B. R. *MQTT*. 2021. <https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/mqtt/>. Accessed: 2022-02-08. Citado na página 26.

OLIVEIRA, S. de. *Internet das coisas com ESP8266, Arduino e Raspberry PI*. [S.l.]: Novatec Editora, 2017. Citado na página 18.

PEREIRA, I. M.; CARNEIRO, T. G. de S.; FIGUEIREDO, E. Understanding the context of iot software systems in devops. In: IEEE. *2021 IEEE/ACM 3rd International Workshop on Software Engineering Research and Practices for the IoT (SERP4IoT)*. [S.l.], 2021. p. 13–20. Citado 2 vezes nas páginas 15 e 22.

PRENS, D. et al. Continuous delivery of software on iot devices. In: IEEE. *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. [S.l.], 2019. p. 734–735. Citado 3 vezes nas páginas 19, 20 e 46.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software-9*. [S.l.]: McGraw Hill Brasil, 2021. Citado na página 41.

PYCQA; CONTRIBUTORS. *What is Pylint?* 2022. <https://pylint.pycqa.org/>. Accessed: 2022-02-10. Citado na página 25.

RAMAMOORTHY, C. V.; LI, H. F. Pipeline architecture. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 9, n. 1, p. 61–102, 1977. Citado na página 15.

RIBEIRO, M. A. Controle de processo. *Oitava edição± Tek Treinamento & consultoria± 2005*, 2005. Citado na página 30.

SABAU, A. R.; HACKS, S.; STEFFENS, A. Implementation of a continuous delivery pipeline for enterprise architecture model evolution. *Software and Systems Modeling*, Springer, v. 20, n. 1, p. 117–145, 2021. Citado na página 16.

SANTOS, B. S. d. Estudo de um protótipo para controle e monitoramento em uma estufa de hortaliças baseado em internet das coisas e o microcontrolador esp8266. 2020. Citado na página 31.

SAVOR, T. et al. Continuous deployment at facebook and oanda. In: IEEE. *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. [S.l.], 2016. p. 21–30. Citado na página 19.

SHIN, S. Introduction to json (javascript object notation). *Presentation www. javapassion. com*, 2010. Citado na página 25.

SHU, R.; GU, X.; ENCK, W. A study of security vulnerabilities on docker hub. In: *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. [S.l.: s.n.], 2017. p. 269–280. Citado na página 28.

SRIVASTAVA, D.; KESARWANI, A.; DUBEY, S. Measurement of temperature and humidity by using arduino tool and dht11. *International Research Journal of Engineering and Technology (IRJET)*, v. 5, n. 12, p. 876–878, 2018. Citado na página 32.

TWITTER, I. *Welcome to the YAML Data Project*. 2020. <https://yaml.com/>. Accessed: 2022-03-08. Citado na página 24.

UMAPATHY, K. et al. Smart wireless sensor and automatic function dependent fire suppression robot with iot. *Renewable Energy with IoT and Biomedical Applications*, p. 113. Citado na página 26.

UWUMUREMYI, G. N. *Design and implementation of an IoT-Based diabetes remote monitoring system*. Tese (Doutorado), 2021. Citado na página 27.

VALENTE, M. T. D. O. et al. Engenharia de software moderna: princípios e práticas para desenvolvimento de software com produtividade. Universidade Federal de Minas Gerais, 2020. Citado 2 vezes nas páginas 15 e 20.

XIA, F. et al. Internet of things. *International journal of communication systems*, v. 25, n. 9, p. 1101, 2012. Citado na página 15.

YIGITOGLU, E. et al. Foggy: a framework for continuous automated iot application deployment in fog computing. In: IEEE. *2017 IEEE international conference on AI & Mobile Services (AIMS)*. [S.l.], 2017. p. 38–45. Citado 2 vezes nas páginas 21 e 22.

Apêndices

APÊNDICE A – Materiais elaborados pelo autor

Será apresentado a seguir o arquivo de configuração yml, que executa a automatização do processo de integração, teste e entrega da imagem Docker.

A.1 Construção do arquivo yml

É inicialmente descrito o nome do fluxo de trabalho. Logo após são especificados os eventos que darão início a execução da pipeline, sendo esses o *push* no *branch "main"* e o *pull request* no mesmo *branch*.

```
name: Pipeline CI

on:
    push:
      branches: [ "main" ]
    pull_request:
      branches: [ "main" ]
```

Logo em seguida são definidos os *jobs* a serem executados, criando cada um deles uma *action* com seus respectivos passos.

Para execução dos processos se faz necessário definir um ambiente de execução, para este, foi utilizado o linux Ubuntu na versão 20.04, sendo esta configuração utilizada em todos os jobs neste pipeline. Internamente a *action "build"* foi declarada uma variável ambiente DOCKER_IMAGE utilizada para armazenar o nome da imagem Docker que será criada.

```
jobs:
  build:

    runs-on: ubuntu-20.04
    env:
      DOCKER_IMAGE: bhpdocker/terrarium_target
```

Nos primeiros passos de execução é obtido o código do repositório e configurado o ambiente e instaladas dependências necessárias para execução.

```
    steps:
2
    # Get the repository's code
4   - name: Checkout
      uses: actions/checkout@v2
```

Como estamos executando uma aplicação que irá rodar em uma arquitetura diferente da padrão utilizada pelo *runner* do Github Actions, foi necessário utilizar o QEMU, um emulador de código aberta genérico que possibilita emular sistemas como arm64.

```
1   # https://github.com/docker/setup-qemu-action
    - name: Set up QEMU
3     uses: docker/setup-qemu-action@v1
```

A.1.0.1 Build

Para execução de comandos e construção de contêineres Docker em compatibilidade com o ambiente determinado, foi utilizado o *plug-in* do CLI do Docker build.

```
1     # https://github.com/docker/setup-buildx-action
    - name: Set up Docker Buildx
3     id: buildx
      uses: docker/setup-buildx-action@v1
```

Os passos acima citados são utilizados nas 3 *actions* construídas, padronizando o estabelecimento do ambiente que foi utilizado no processo de CI.

Dentro da *action "build"* temos como último passo a construção da imagem Docker, sendo definida através do Dockerfile, sendo este previamente configurado para criação dos parâmetro do sistema especificado tendo como plataforma linux arm64, bem como as dependências da aplicação e outras definições na imagem Docker base utilizada.

```
    - name: build the image
2     run: |
        docker buildx build \
4         --tag ${{ env.DOCKER_IMAGE }} \
          --platform linux/arm64   .
```

A.1.1 Test

Para que seja executado a *action* de teste (test), é estabelecido como critério a execução com sucesso da *action "build"*.

```
test:

  runs-on: ubuntu-20.04
  needs: [build]
```

Para execução de testes de lint na linguagem Python através do gerenciador de pacotes pip é instalado a ferramenta pylint sendo utilizada para teste de lint em todos os códigos da aplicação.

```
  - name: Install dependencies
    run: |
      python -m pip install --upgrade pip
      pip install pylint
   - name: Analyze code with pylint
    run: |
      pylint $(git ls-files '*.py')
```

Para execução dos testes de unidade é utilizado o doctest, sendo executados códigos de teste de unidade definidos.

```
  - name: Analyze code with doctest
    run: |
      python -m doctest -v test/test_with_doctest.py
```

A.1.2 Delivery

Para que seja executado a *action* de Delivery, é estabelecido como critério a execução com sucesso da *action* Test.

```
delivery:

  runs-on: ubuntu-20.04
  needs: [test]
```

Com o objetivo de entrega da imagem que já passou com sucesso pelo processo de construção e testes, foi estabelecido o uso do repositório de registro Docker Hub, sendo

assim necessário ao acesso o login na plataforma, utilizando de credenciais predefinidas no Github.

```
    - name: login to docker hub
      run: echo "${{ secrets.DOCKER_TOKEN }}" | docker login -u "${{
          secrets.DOCKER_USERNAME }}" --password-stdin
```

Após o login com sucesso é executado o comando de *push* da imagem anteriormente construída e testada ao Docker Hub, finalizando assim o processo de entrega da imagem contendo a aplicação.

```
    - name: Push to Docker Hub
      uses: docker/build-push-action@v2
      with:
        context: .
        platforms: linux/arm64
        push: true
        tags: ${{ secrets.DOCKER_HUB_USERNAME }}/terrarium_target:latest
        cache-from: type=registry,ref=${{ secrets.DOCKER_HUB_USERNAME }}/
            terrarium_target:buildcache
        cache-to: type=registry,ref=${{ secrets.DOCKER_HUB_USERNAME }}/
            terrarium_target:buildcache,mode=max
```

Arquivo completo:

```
name: Pipeline CI

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]


jobs:
  build:

    runs-on: ubuntu-20.04
    env:
      DOCKER_IMAGE: bhpdocker/terrarium_target

    steps:

    # Get the repository's code
    - name: Checkout
```

```
21      uses: actions/checkout@v2
        # https://github.com/docker/setup-qemu-action
23
    - name: Set up QEMU
25      uses: docker/setup-qemu-action@v1
        # https://github.com/docker/setup-buildx-action
27    - name: Set up Docker Buildx
        id: buildx
29      uses: docker/setup-buildx-action@v1

31    - name: build the image
        run: |
33        docker buildx build \
            --tag ${{ env.DOCKER_IMAGE }} \
35          --platform linux/arm64  .


37


39  test:

41    runs-on: ubuntu-20.04
      needs: [build]
43
      steps:
45    # Get the repository's code
      - name: Checkout
47      uses: actions/checkout@v2
        # https://github.com/docker/setup-qemu-action
49    - name: Set up QEMU
        uses: docker/setup-qemu-action@v1
51      # https://github.com/docker/setup-buildx-action
      - name: Set up Docker Buildx
53      id: buildx
        uses: docker/setup-buildx-action@v1
55
      - name: Install dependencies
57      run: |
          python -m pip install --upgrade pip
59        pip install pylint
       - name: Analyze code with pylint
61      run: |
          pylint $(git ls-files '*.py')
63    - name: Analyze code with doctest
        run: |
65        python -m doctest -v test/test_with_doctest.py


67
```

```
  delivery:
69
    runs-on: ubuntu-20.04
71  needs: [test]

73  env:
      DOCKER_IMAGE: env.DOCKER_IMAGE
75
    steps:
77  # Get the repository's code
    - name: Checkout
79    uses: actions/checkout@v2
      # https://github.com/docker/setup-qemu-action
81  - name: Set up QEMU
      uses: docker/setup-qemu-action@v1
83    # https://github.com/docker/setup-buildx-action
    - name: Set up Docker Buildx
85    id: buildx
      uses: docker/setup-buildx-action@v1
87
    - name: login to docker hub
89    run: echo "${{ secrets.DOCKER_TOKEN }}" | docker login -u "${{ secrets
          .DOCKER_USERNAME }}" --password-stdin

91  - name: Push to Docker Hub
      uses: docker/build-push-action@v2
93    with:
        context: .
95      platforms: linux/arm64
        push: true
97      tags: ${{ secrets.DOCKER_HUB_USERNAME }}/terrarium_target:latest
        cache-from: type=registry,ref=${{ secrets.DOCKER_HUB_USERNAME }}/
            terrarium_target:buildcache
99      cache-to: type=registry,ref=${{ secrets.DOCKER_HUB_USERNAME }}/
            terrarium_target:buildcache,mode=max
```

A.2 Características técnicas do *Deploy*

Para êxito no processo de implantação e execução do contêiner Docker, é necessário ao efetuar o *pull* da imagem a mesma deve ser feita com a tag: –privileged, devido acesso ao módulo para controle da GPIO ser permitido apenas como usuário *root*.

SHELL

```
1 docker run --name=terrarium --privileged --restart=always bhpdocker/
    terrarium_target:latest
```