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

Junho, 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

Junho de 2022

A Ficha Catalográfica é elaborada exclusivamente pela Biblioteca. Substitua esta página pelo documento gerado na versão final da sua monografia.

A Folha de aprovação deverá ser gerada pelo(a) professor(a) orientador(a) após a realização das correções sugeridas pela banca examinadora.

As instruções para elaboração desse documento eletrônico estão disponíveis em <https://www.monografias.ufop.br> (menu “Documentos”, opção “Tutorial Professor Orientador”).

Após gerar a folha de aprovação, o(a) professor orientador(a) enviará o documento ao(à) orientado(a), o qual deverá inseri-lo nesta página.

Dedico este trabalho e todas as minhas conquistas a minha família.

Agradecimentos

Agradeço ...

"Science is more than a body of knowledge; it is a way of thinking."

— Carl Sagan (1934 – 1996),
in: The Demon-Haunted World: Science as a Candle in the Dark.

Resumo

Estudos apontam que diversas aplicações vem sendo desenvolvidas para sistemas IoT, e com o avanço da Industria 4.0, esse número tende a crescer. O processo de desenvolvimento e implantação de *software* para dispositivos IoT não possui muitas abordagens quando se trata da construção de pipelines de entrega continua. A utilização de soluções de código aberto possibilitam que o usuário proponha e implemente melhorias. Porém, essa prática com o objetivo de construir processos automatizados de entrega de *software* para dispositivos IoT, são raramente aplicadas. Sistemas de monitoramento e execução de interações com o ambiente, utilizando dispositivos com baixa uso de recursos, pelo grande número de ambientes de aplicação, é comum existir diversas implementações com esse proposito. Considerando esse contexto, foi construído um pipeline de CI/CD para uma aplicação que utiliza sensores e atuadores para interagir com o meio, sendo esta instalada em uma Placa Raspberry Pi. Foram utilizadas apenas soluções de código aberto na construção do pipeline. O pipeline possibilitou a aceleração no processo de entrega do *software* ao *target*, e a garantia de que o código implantado seguia os padrões preestabelecidos, sendo também possível o levantamento de dados relativos ao processo de entrega do *software*. Também foi possível obter uma aplicação que faz o monitoramento da temperatura do ambiente, umidade do solo e intensidade de luminosidade, possibilitando também a irrigação do solo.

Palavras-chaves: Pipeline de CI/CD. Código aberto. Internet das coisas. Monitoramento de ambiente.

Abstract

Studies indicate that several applications are being developed for IoT systems, and with the advance of Industry 4.0, this number tends to grow. The process of developing and deploying *software* for IoT devices does not have many approaches when it comes to building continuous delivery pipelines. The use of open source solutions makes it possible for the user to propose and implement improvements. However, this practice with the goal of building automated delivery processes of *software* for IoT devices, are rarely applied. Systems for monitoring and execution of interactions with the environment, using devices with low resource usage, due to the large number of application environments, it is common to have several implementations for this purpose. Considering this context, an CI/CD pipeline was built for an application that uses sensors and actuators to interact with the environment, which is installed on a Raspberry Pi board. Only open source solutions were used in the construction of the pipeline. The pipeline made it possible to speed up the process of delivering the *software* to the *target*, and to guarantee that the deployed code followed the pre-established standards. It was also possible to obtain an application that monitors the environment's temperature, soil humidity and luminosity intensity, also allowing the irrigation of the soil.

Key-words: CI/CD pipeline. open-source. internet of things. environment monitoring.

Lista de ilustrações

Lista de abreviaturas e siglas

IoT Internet of Things

MQTT Message Queuing Telemetry Transport

CI *Continuous Integretion*

CD *Continuous Delivery*

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 a 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 os 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 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, automatização 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 automatização dos passos das fases do projeto.

Internet das Coisas (do inglês IoT), de modo geral, refere-se à interconexão em rede dos objetos do cotidiano a *Internet*, sendo uma enorme oportunidade para um grande número de aplicações que propiciem 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 de utilização de práticas de Integração Contínua (do inglês *Continuous Integretion* (CI)) e Entrega Contínua (do inglês *Continuous Delivery* (CD)), a utilização dessas conceitos em um projeto constitui a prática CI/CD.

O pipeline de 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 todas as atividades como análise de código estático, construção automática e teste de unidade. Além disso, executa-se atividades de teste como integração, desempenho e segurança. 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 de qualidade 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 a versão anterior de forma rápida, condição que pode ser necessária se algum erro não foi identificado durante os testes.

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).

1.1 O problema de pesquisa

As etapas de desenvolvimento de uma aplicação 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. Atendendo também às demandas do mercado como rápidas entregas de qualidade para manter a competitividade, podendo ser atendidas com o uso dos conceitos de CI/CD. Sendo assim, entender e aplicar com eficiência esses conceitos é de fundamental importância para profissionais e empresas na área de tecnologia.

Em (ROSENKRANZ et al., 2015) é citado o não conhecimento de trabalhos que utilizam apenas ferramentas *open source*, tendo o objetivo de lidar com problemas relacionados a teste de *softwares* para dispositivos de Internet of Things (IoT). Aplicar

a prática de CI significa obter a construção da automatização de testes para módulos do *software*, tendo também a automatização 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 ferramentas de *software* livre, se torna um grande desafio.

1.2 Objetivos

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. Tendo no processo de CI/CD a utilização de ferramentas *open source*, de forma a apoiar o movimento.

Este trabalho possui aos seguintes objetivos específicos:

- **Implementar** conceitos de qualidade em engenharia de software, propondo um ordenamento e levantamento de informações, em uma aplicação para dispositivos IoT.
- **Utilizar** ferramentas *open-source* na construção do pipeline de entrega continua, com o objetivo de apoiar o modelo de desenvolvimento colaborativo de softwares.
- **Identificar** problemas relacionados a implementação de um pipeline automatizado de CI/CD para aplicações IoT.
- **Construir** uma estrutura de fácil replicação no monitoramento de uma ambiente para cultivo de plantas.

2 Revisão bibliográfica

2.1 Conceitos básicos

Nesta seção, serão apresentados conceitos para guiar no entendimento completo na 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).

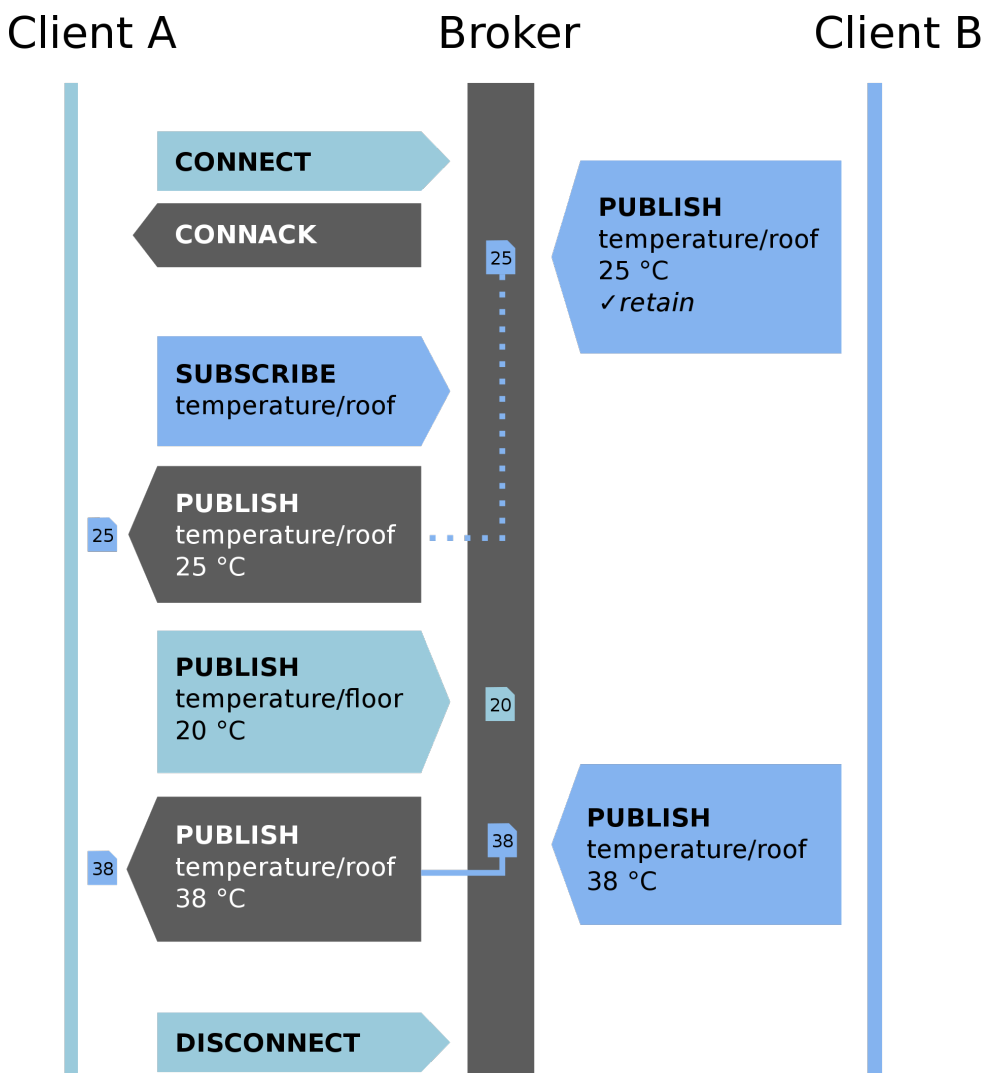
2.2 MQTT

O Message Queuing Telemetry Transport (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 onde 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 *subscribe* 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* que tenham interesse nesse assunto específico. O protocolo se popularizou pela simplicidade, baixo consumo de dados e pela possibilidade comunicação bilateral.

É possível realizar a comunicação com o *broker* tanto por meio de *WebSockets* quanto puramente com o MQTT. Os navegadores ainda não possuem compatibilidade para realizar puramente com o MQTT. Tanto para receber quanto para enviar mensagens,

Figura 1



Fluxo de comunicação de dois clientes por meio de um *broker*. Fonte: (NERI, 2021)

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. O MQTT foi utilizado como protocolo de comunicação entre o *host* e o *target*.

2.3 Trabalhos correlatos

Nesta seção, será apresentada a revisão da literatura realizada. Ademais, serão apresentadas trabalhos que com objetivo de aplicar soluções de processos de entrega e para garantia de qualidade de *software*.

Em (FERRY et al., 2019) e 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 contêiner Node-RED 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*, já este trabalho tem como foco os processos para garantia da qualidade do *software* desenvolvido para o *target*.

No trabalho proposto por (PRENS et al., 2019) são identificados requisitos de entrega continua 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 semi-automático; Req3: O consumo de energia e o tempo inativo devem ser monitorados para cada atualização instalada; Req4: O engenheiro deve ser capaz de monitorar o dado para cada versão de *software* instalado, por uma interfase interativa. Como o modelo apresentado na abordagem o presente trabalho utiliza de ferramentas similares para atender a entrega remota e automatizada (Req2), porém o atual trabalho atende de forma mais completa a identificação do *target*(Req1), bem como a melhor monitoramento de cada passo executado no processo de CI.

Em (LÓPEZ-VIANA et al., 2020) é abordado 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.

(DORESTE, 2018) implementa um pipeline para apoiar o desenvolvimento continuo de no nicho específico de sistema de *software* para IoT, aplicativos desenvolvidos para Raspiberry Pi. Por meio da implementação desse pipeline foi passível observar a prática de desenvolvimento continuo 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.

Na obra (SABAU; HACKS; STEFFENS, 2021) é apresentada uma implementação baseada na técnica de CD, porém disponibilizando dados superficiais do pipeline, sendo

este, empregado no processo de manutenção do modelo de arquitetura corporativa do inglês *enterprise architecture* (EA). Foi constatado, que mesmo com a grande valia dos conceitos de CI no processo de manutenção automatizada, ainda se faz necessário adaptações para gerenciamento dos artefatos do modelo.

Na pesquisa desenvolvida em (PEREIRA; CARNEIRO; FIGUEIREDO, 2021), onde o objetivo era 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. Porém, é citado apenas uma configuração de estados de um pipeline de implantação de DevOps para sistemas de *software* IoT.

Os trabalhos acima citados tem como objetivo principal a implementação ou pesquisa da utilização de um pipeline de CI/CD, sendo os trabalhos (LÓPEZ-VIANA et al., 2020; DORESTE, 2018; PEREIRA; CARNEIRO; FIGUEIREDO, 2021) focados em dispositivos IoT. Porém é possível observar na literatura, poucas pesquisas e implementações com o objetivo de utilizar ferramentas para construção de pipelines de CI/CD para dispositivos IoT, esse ponto se torna ainda mais evidente quando se fala da utilização de ferramentas *open-source*.

3 Desenvolvimento

3.1 Ferramentas e tecnologias

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

3.1.1 Software

Nessa seção serão descritas as ferramentas e tecnologias que foram utilizadas diretamente na construção da aplicação, sendo servidores 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 GitLab

GitLab (B.V., 2022a) é uma plataforma DevOps que possibilita organizações e maximizar o retorno geral do desenvolvimento de *software*, fornecendo *software* com mais rapidez e eficiência, ao mesmo tempo em que fortalece a segurança e a conformidade. Por proporcionar uma efetiva aplicação dos conceitos principais do trabalho, essa plataforma foi utilizada na construção e automatização de alguns dos processos de CI/CD da pipeline. O GitLab *Community Edition* é um pacote de software auto-hospedado que fornece hospedagem de repositório subseção 3.1.1.1, acompanhamento de projetos, serviços de CI/CD, e um registro de imagem Docker.

3.1.1.3 Doctest

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

3.1.1.4 YAML

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

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

E similar ao JSON e o XML. Por possibilitar a criação do arquivo de configuração responsável por definir os passos de execução de CI/CD do pipeline, este padrão foi utilizado no presente trabalho.

3.1.1.5 GitLab CI/CD e GitLab Runner

GitLab CI/CD (AMIRAULT, 2022) é uma ferramenta para desenvolvimento de *software* que utiliza as metodologias de integração e entrega contínua. Com o GitLab CI/CD é possível detectar *bugs* e erros no início do ciclo de desenvolvimento. Certificando-se de que todo o código implantado na produção esta em conformidade com os padrões de código que você estabeleceu para sua aplicação.

Com o GitLab CI/CD é possível criar, testar, implantar e monitorar automaticamente seus aplicativos usando uma coleção de pre-configurações que definem funcionalidades em conformidade a cultura DevOps, no processo de entrega de *software*.

O GitLab Runner (AMIRAULT, 2022) é um aplicativo que funciona com o GitLab CI/CD para executar determinados comandos chamados *jobs*, na estrutura de uma pipeline.

O GitLab Runner é de código aberto e escrito em Go . Ele pode ser executado como um único binário; não são necessários requisitos específicos de idioma.

É possível utilizar GitLab CI/CD e o GitLab Runner no próprio GitLab.com[1], permitindo executar trabalhos de CI/CD em *runners* hospedados pelo GitLab. Esses *runners* são gerenciados pelo GitLab e totalmente integrados ao GitLab.com. Através do arquivo 3.1.1.4 nomeado por .gitlab-ci.yml junto ao repositório, é possível especificar ao *runner* os comandos e passos a serem executados em um pipeline. Sendo esta abordagem escolhida para a implementação da solução no presente trabalho.

3.1.1.6 Docker

Docker (INC., 2021) é uma plataforma aberta para desenvolvimento, envio e execução de aplicações. Permite a separação entre o seus aplicativos e sua infraestrutura

[1] <https://gitlab.com>

para entregas de *software* mais rapidamente. Permitindo também o gerenciamento da infraestrutura, as metodologias do Docker podem ser aproveitadas para enviar, testar e implantar o código reduzindo significativamente o atraso entre escrever o código e executá-lo em produção. O Docker funciona com uma arquitetura cliente-servidor. O cliente Docker conversa com o *daemon* do Docker, que é responsavel por construir, executar e distribuir seus contêineres. 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 é o Docker Compose, permitindo trabalhar com aplicativos que consistem em um conjunto de contêineres. Processo ilustrado na Figura Figura 2.

Figura 2

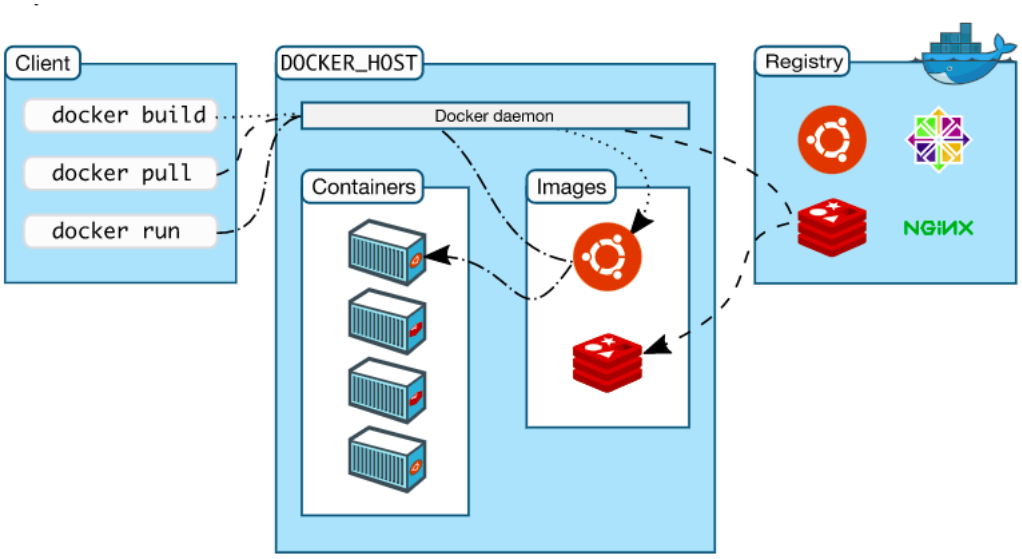


Ilustração da arquitetura Docker. Fonte: (INC., 2021)

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

3.1.1.7 Watchtawer

O aplicativo Watchtower (DONATH., 2022) monitora os contêineres Docker que estão rodando e procura por mudanças nas imagens em suas respectivas origens. Se for detectada alguma mudança na imagem, ele irá reiniciar seu contêiner Docker automaticamente utilizando a nova imagem. Com o Watchtower é possível atualizar os contêineres das aplicações Docker que estão rodando apenas publicando uma nova imagem no Docker Hub ou em um próprio registro de imagens.

3.1.1.8 JSON

O *JavaScript Object Notation* (JSON) () 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 em todo processo de transferência de dados pela *Internet*.

3.1.1.9 Python

O Python (FOUNDATION., 2022a) é uma linguagem de programação com estruturas de dados com auto nível de eficiência.é 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 aplicativo *server*, que foi executado na placa Raspberry.

3.1.1.10 MQTTBox

MQTTBox(UMAPATHY et al.,) é uma aplicação que trabalha com o protocolo seção 2.2 disponível para sistemas Linux, Mac e Windows, é 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 TCP ou protocolos *Web Sockets*;
- Conectar-se com uma ampla variedade de configurações de conexões do cliente mqtt;
- Publicar e escrever múltiplos tópicos ao mesmo tempo;
- Reconexão do cliente ao *broker*.

Devido essas funcionalidades o MQTTBox foi utilizado para teste das mensagns enviadas pela aplicação *server*.

3.1.1.11 Mosquitto

Mosquitto(FOUNDATION, 2022)é um *Broker* de mensagem MQTT de *software* livre. Sendo um servidor intermediário da informação. É ele quem recebe os dados enviados pelos sensores e é nele onde esses dados são tratados e passados adiante. Permite ser utilizado em aplicações que utilizam Python, O Mosquito foi utilizado para desenvolvimento desse *Broker* da aplicação.

3.1.1.12 PyFirmata

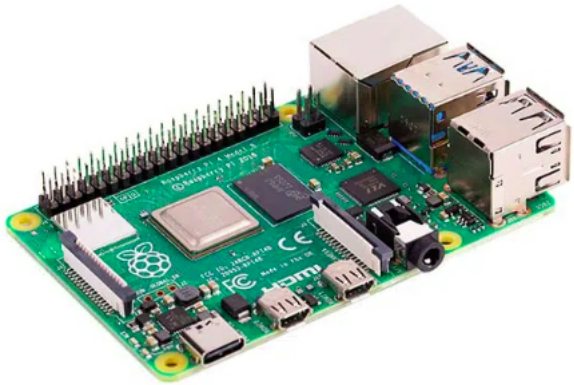
PyFirmata (BRUIJN., 2010) é uma interfase Python para o protocolo de comunicação entre microcontroladores e *software* denominado Firmata. Permitindo a manipulação de placas com microcontroladores como o Arduino, utilizando apenas a linguagem Python. Por possibilitara a utilização da linguagem Python, ao manipular a placa Arduino. Sendo o Arduino responsável por coletar as informações dos sensores analógicos, foi utilizado esse modulo na implementação da solução.

3.1.2 Hardware

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

3.1.2.1 Raspberry

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. Por sua portabilidade, conectividade e poder de processamento, neste trabalho foi utilizado a versão Raspberry Pi 4 Model B, visto na Figura Figura 3, para montar um *server* para aplicação.

3.1.2.2 Arduino

O Arduino é uma plataforma eletrônica de código aberto baseada em hardware e software livre. Desenvolvido na Itália, no Instituto de Design e Interação Ivera, a plataforma conquistou adeptos rapidamente, tendo que se adaptar as novas necessidades e desafios que

surgiram ao longo do tempo. Deste modo a fabricante passou a desenvolver não somente placas simples, mas também produtos com foco em IoT, tecnologias vestíveis, impressão 3D e ambientes incorporados (GINGL et al., 2019).

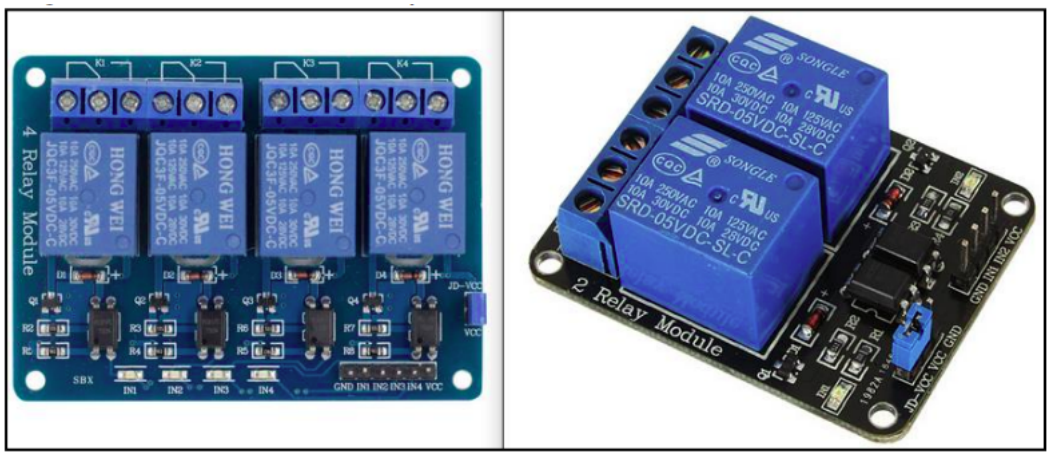
O Arduino consiste em um equipamento semelhante a um pequeno computador, que pode ser programado de forma a possibilitar o processamento de entradas e saídas com outros dispositivos e componentes externos conectados a ele (MCROBERTS, 2015). Na solução aqui apresentada foi utilizado o Arduino Uno.

3.1.2.3 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 Figura 4 pode ser dividido em dois tipos: Normalmente Aberto (NA) e Normalmente Fechado (NF) (FRANCHI, 2018): a) NA: O contato por padrão é aberto, ou seja, até que haja uma ação externa o contato permanece em estado aberto; b) 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



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

3.1.2.4 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.

3.1.2.5 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).

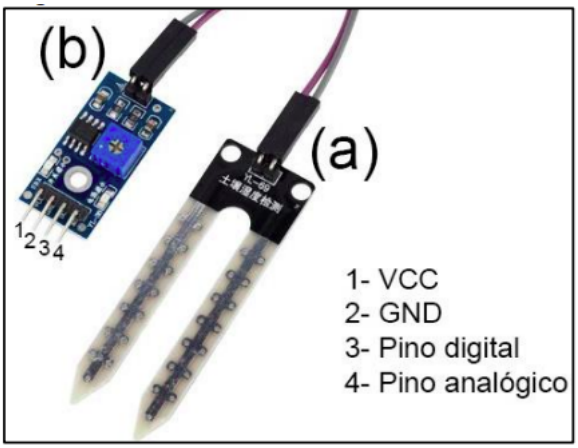
3.1.2.5.1 Sensores de umidade do solo

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).

3.1.2.5.2 Sensores de Temperatura

Os sensores de temperatura são utilizados com intuito de identificar o nível de energia térmica presente nos sistemas ou equipamentos. A medida de temperatura pode ser

Figura 5



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

obtida por meio de alguns princípios físicos como o termoelétrico e o resistivo (SANTOS, 2020). No mercado existe uma ampla variedade de sensores de temperatura, com diferentes características e aplicações. Amplamente utilizado em projetos acadêmicos que visam a implementação de uma solução de baixo custo, encontra-se o uso dos sensores NTC, sendo este modelo utilizado no protótipo desenvolvido.

3.1.2.5.3 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 de acordo com a intensidade de radiação eletromagnética que índice 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).

3.2 Implementação

Esta seção serão apresentadas as etapas de construção do Pipeline CI/CD e do servidor da aplicação de monitoramento.

Para criação de um pipeline com práticas continuas, é necessários a definição de aspecto de sistema e infraestrutura, também com o objetivo de validar a proposta na construção do Pipeline foi desenvolvido pelo autor uma aplicação *server* conectado à *internet*, onde o usuário tem acesso às informações de condições do ambiente, sendo

inicialmente aplicado o monitoramento da umidade do solo e irrigação remotamente. Também com o objetivo de observar os benefícios na prática de implantação do Pipeline CI/CD, foram implementadas duas novas funcionalidades sendo essas, o monitoramento de temperatura e de intensidade de luminosidade no ambiente.

3.2.1 Pipeline CI/CD

Inicialmente será apresentado os passos realizados para construção da Pipeline, que vão de encontro as estratégias de qualidade e controle dos artefatos da aplicação.

Para implementação da solução, são necessários requisitos iniciais para implementação e manipulação das plataformas e ferramentas.

Foi utilizada o editor Visual Studio Code[2] na sua versão 1.68, para edição e criação dos arquivos e sua estrutura de pastas. O sistema de controle de versionamento Git, citado na subseção 3.1.1.1, foi instalado em um computador em sua versão 2.3. Para execução da aplicação no *target* (placa *Raspberry* subseção 3.1.2.1) foi instalado o instalado o Cliente Docker no *target*, possibilitando a instalação e execução de diferentes contêineres. Para utilização das funcionalidades na plataforma GitLab, citada na subseção 3.1.1.2, e dos pacotes de *software* do GitLab *Community Edition* foi criado uma conta na plataforma e definido um projeto chamado terrariumtarget, referente a aplicação *server* que serviu de base na implementação da Pipeline.

3.2.1.1 Arquitetura da Pipeline

Seguindo os princípios para entrega de *software* foi definido uma estrutura para construção da Pipeline representado na Figura 6. Onde o desenvolvedor codifica a aplicação, e através da execução do *push* utilizando o Git, envia os artefatos no repositório principal chamado *main*. Essa ação da inicio a execução dos processos automatizados da Pipeline, onde são executados os seguintes passos:

- Construção da imagem Docker, predefinida no arquivo Dockerfile, podendo usar definições de uma imagem já existente;
- Execução do contêiner Docker e dos testes de unidade;
- Publicação da imagem Docker no registro de imagens e notificação do desenvolvedor através e-mail.

Por fim o contêiner do Watchtawer, citado na subseção 3.1.1.7, monitora a atualização das imagens dos contêineres instalados no *target*, efetuando a atualização do contêiner da aplicação mediante alterações na sua imagem.

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

Figura 6

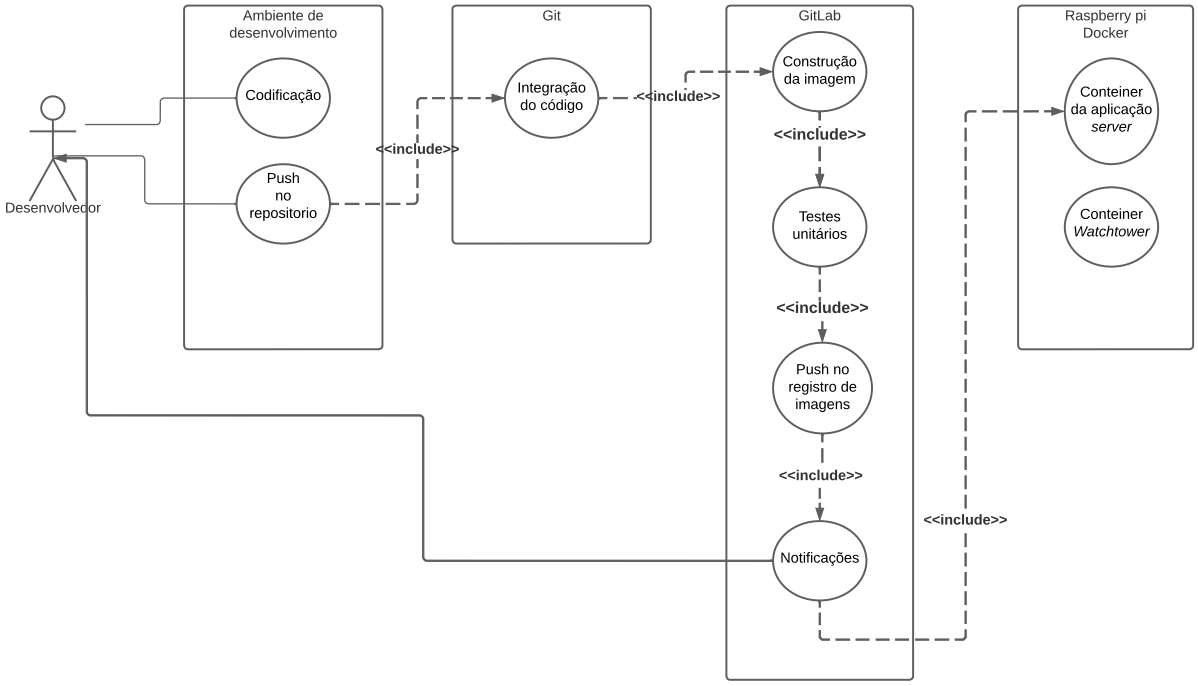


Diagrama de implementação. Fonte: Produzido pelo autor

3.2.1.2 Construção da Pipeline

Para definição dos passo da Pipeline executados na plataforma do GitLab, foi utilizado um arquivo YAML especificando desde a construção da imagem até o processo de publicação da imagem. Esses comandos são executados pelo GitLab CI/CD e o GitLab Runner, ambos citados na subseção 3.1.1.5.

Foi realizado o processo de escrita dos testes de unidade, mediante levantamento das menores unidades de código necessárias para o correto funcionamento da aplicação, bem como, a definição dos comportamentos esperados para cada uma delas. No processo de escrita dos testes de unidade foi utilizado modulo Doctest, citado na subseção 3.1.1.3, onde os comportamentos definidos são observados e conferidos.

Para construção da imagem definida pelo arquivo Dockerfile, foi realizada pesquisa com o objetivo de garantir a execução da aplicação na arquitetura correspondente ao *target* (Raspberry Pi). Também por esse motivo foi escolhido a linguagem de programação Python, citada na subseção 3.1.1.9, como se trata de uma linguagem interpretada a geração do executável é adaptável ao seu ambiente de execução.

YAML

Parte do código onde são definidos os estágios da pipeline, sendo estes o *buid*, *test* e *delivery*:

```yaml

stages:
  - build
  - test
  - deploy
```

YAML

É criada uma variável IMAGE_NAME, que defini o nome da imagem que será disponibilizada no registro de imagens do GitLab. as variáveis CI_REGISTRY_IMAGE e CI_BUILD_REF_NAME são preenchidas automaticamente pelo GitLab Runner. O bloco de comandos definido pelo *script*, especifica cada linha de comando executada, aonde os parâmetros para a construção da imagem *docker* são definidos no arquivo DockerFile.

```yaml
variables:
  IMAGE_NAME: $CI_REGISTRY_IMAGE:$CI_BUILD_REF_NAME

build:
  image: docker:latest
  services:
    - docker:dind
  stage: build
  script:
      - docker login -u "gitlab-ci-token" -p "$CI_JOB_TOKEN"
        $CI_REGISTRY
      - docker build -t $IMAGE_NAME .
      - docker push $IMAGE_NAME
  tags:
    - docker
```

YAML

Estagio onde é executado os testes de unidade, definidos no arquivo test_with_doctest.py:

```yaml
test:
  image: $IMAGE_NAME
  stage: test
```

```
4           script:
                - python -m doctest -v test/test_with_doctest.py
6           tags:
              - docker
```

YAML

Neste estagio e executado o processo de entrega da imagem, ao registro de possuindo a aplicação :

```
2         delivery:
            image: docker:latest
4           stage: delivery
            services:
6             - docker:dind

8           before_script:
              - echo $CI_BUILD_TOKEN | docker login -u $CI_REGISTRY_USER --
                  password-stdin $CI_REGISTRY
10
            script:
12            - docker build --pull -t $CI_REGISTRY_IMAGE .
              - docker push $CI_REGISTRY_IMAGE
```

É importante destacar que comportamentos fora do especificado, e erros detectáveis que ocorram durante a passagem código pela Pipeline interrompe sua execução. Impossibilitando que o atual código com problema seja implantado no sistema.

3.2.2 Desenvolvimento do protótipo para monitoramento e controle de uma estufa

Para planejamento do protótipo foi criado um diagrama de implantação, mostrado na Figura 7, para definir em uma visão geral, os principais componentes para construção do protótipo.

Na Figura 7 o atuador de irrigação representa o rele, que por sua vez atua a bomba, para irrigação do solo.

Figura 7

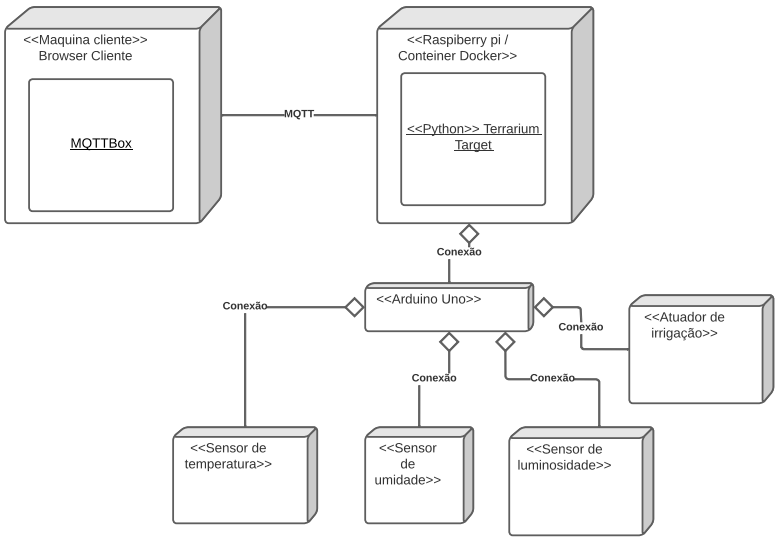


Diagrama de implantação. Fonte: Produzido pelo autor

3.2.2.1 *Hardware*

Também foi possível através do Tinkcard gerar o diagrama do circuito, como pode ser visto na Figura 8.

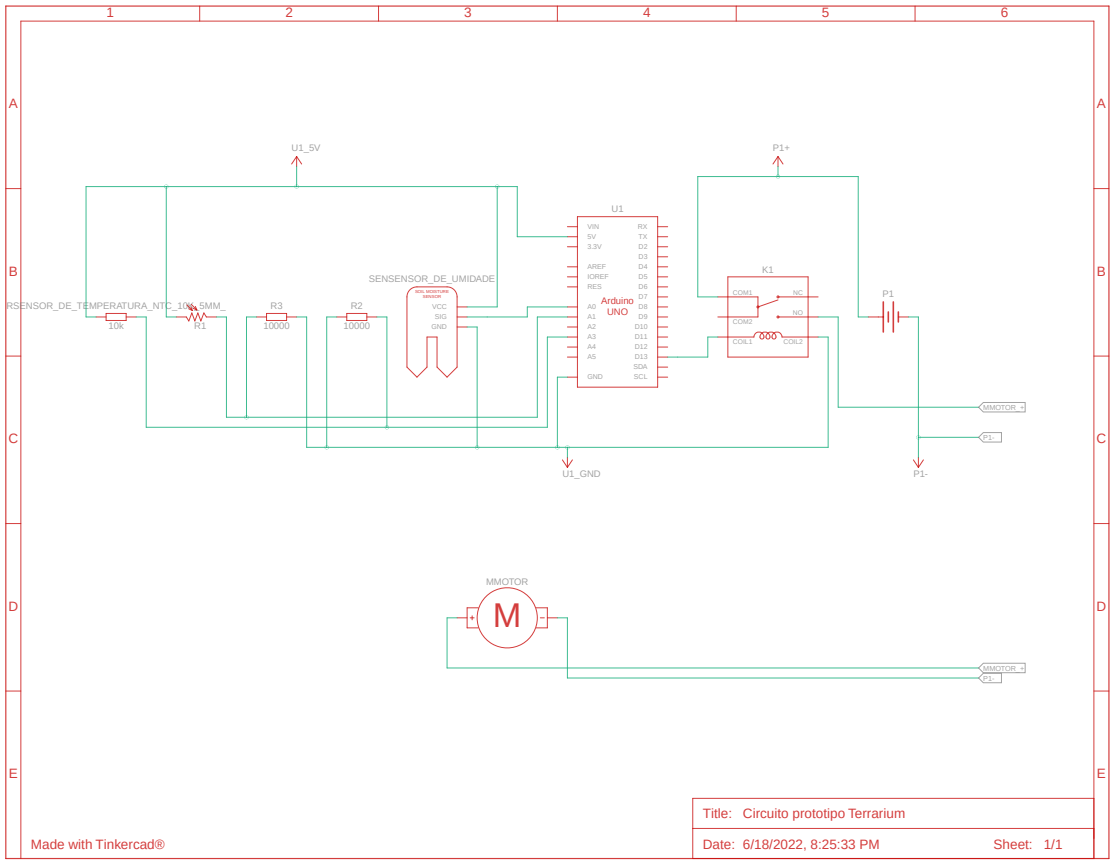
Antes da construção do protótipo foi realizada a simulação do circuito, utilizando a plataforma Tinkcard[3], sendo possível a analise e reavaliação de componentes a serem utilizados. Imagem da simulação pode ser visualizada na Figura 9.

Com a definição dos componentes necessários foi realizada uma pesquisa nas lojas virtuais em busca dos componentes especificados na lista da Figura 10.

De posse dos componentes foram montadas as ligações sobre uma *protoboard*, seguindo o esquemático elétrico definido na Figura 8. A conexão entre a placa Arduino Uno e o Raspberry Pi foi realizada por meio do cabo USB, e a conexão do Raspberry Pi com a *Internet* foi estabelecida via *wifi*.

[3] <https://www.tinkercad.com/>

Figura 8



Esquemático elétrico. Fonte: Gerado pelo autor em (AUTODESK, 2022)

3.2.2.2 *Software*

A aplicação *server* foi desenvolvida na linguagem Python, citada na subseção 3.1.1.9, e para implementar a interface de comunicação com a placa arduino, foi utilizado a biblioteca PyFirmata. Para utilizar os sensores analógicos foi utilizado a placa Arduino Uno, que roda a biblioteca Firmata, um protocolo de comunicação do arduino com computadores.

O processo de criação da aplicação (*server*) que roda na placa Raspberry (*target*) foi realizado paralelo ao processo de construção da Pipeline. Para envio e recebimento de mensagens do *server* foi utilizado o *Broker*. No processo de teste do envio e recebimento das mensagens foi utilizado a aplocação MQTTBox, citada na subseção 3.1.1.10.

Figura 9

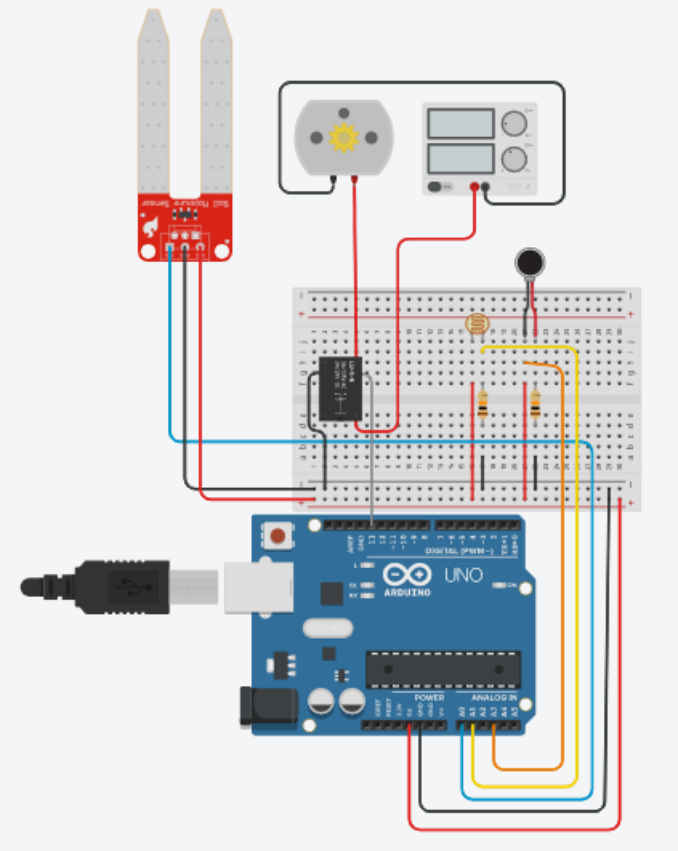


Imagem digital dos componentes e ligações. Fonte: Produzido pelo autor

Figura 10

Nome	Quantidade	Componente
U1	1	Arduino Uno R3
R2, R3	2	10000 Ω Resistor
Sensor de umidade	1	Sensor de humidade de solo
R1	1	Fotorresistor
Motor	1	Motor CC
K1	1	Relé SPDT
P1	1	5 , 2 Fonte de energia
Sensor de Temperatura NTC 10K 5mm	1	10 kΩ Resistor

Lista de componentes. Fonte: Produzido pelo autor

4 Resultados

Segundo (SOMMERVILLE, 2011), a fase de testes em um *software* possui dois objetivos distintos. O primeiro deles é o demonstrar ao desenvolvedor e ao cliente que o software atende aos requisitos propostos, e estão relacionados aos chamados testes de validação, onde são utilizados conjuntos de casos de teste que representam o uso esperado do sistema. O segundo propósito é descobrir possíveis situações em que o software se comporta de maneira inesperada, indesejável ou fora do que foi especificado.

Abstraindo a definição proposta, para nossa implementação, foram executados processos de testes dos padrões especificados na pipeline. Alguns resultados estão apresentados nesta seção.

Foi possível o acompanhamento de cada fase de execução especificado no Pipeline, fato que pode ser visto através da Figura 11, informações estas geradas em tempo real de execução. O acompanhamento de cada estagio de execução é de fácil visualização na plataforma, como demonstrado na Figura 12.

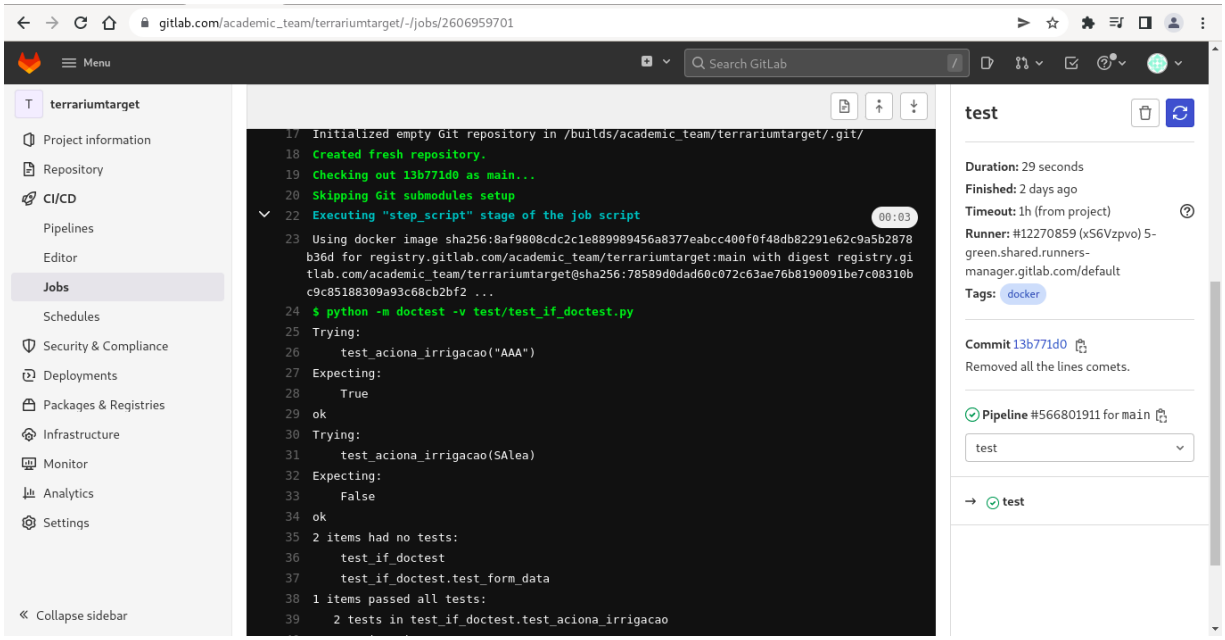
Com a utilização da plataforma GitLab na construção da Pipeline, foi possível a geração de informações úteis na análise do comportamento do processo de desenvolvimento de *software*. Diferentes métricas e dados ficaram acessíveis para futuras análises dos processos, em busca de apontar melhorias contínuas no processo de desenvolvimento de *software*. Alguns exemplos de dados que podem ser coletados na plataforma podem ser visualizados nas Figuras 13, 14

No painel apresentado na Figura 12, a visualização dos estágios de execução do pipeline entre cada versão do código se torna fácil, sendo também possível, visualizar detalhes de cada comando executado no pipeline,bem como passíveis erros, que pode ser visto na Figura 11.

A identificação do tempo gasto para cada execução da pipeline, pode indicar melhorias necessário na sua estrutura, o que tem consequente impacto no processo de qualidade do *software*. Na Figura 13, pode ser observado a identificação singular para cada execução do pipeline e do respectivo tempo de execução.

O dados disponibilizados na plataforma, referentes a quantidade de *commits* em um determinado período de tempo, identificando entre eles os que deram certo. Informações apresentadas na Figura 14, podem servir como parâmetros para iniciar campanhas de engajamento nas equipes de desenvolvimento, comportamento apoiado em (VALENTE et al., 2020).

Figura 11

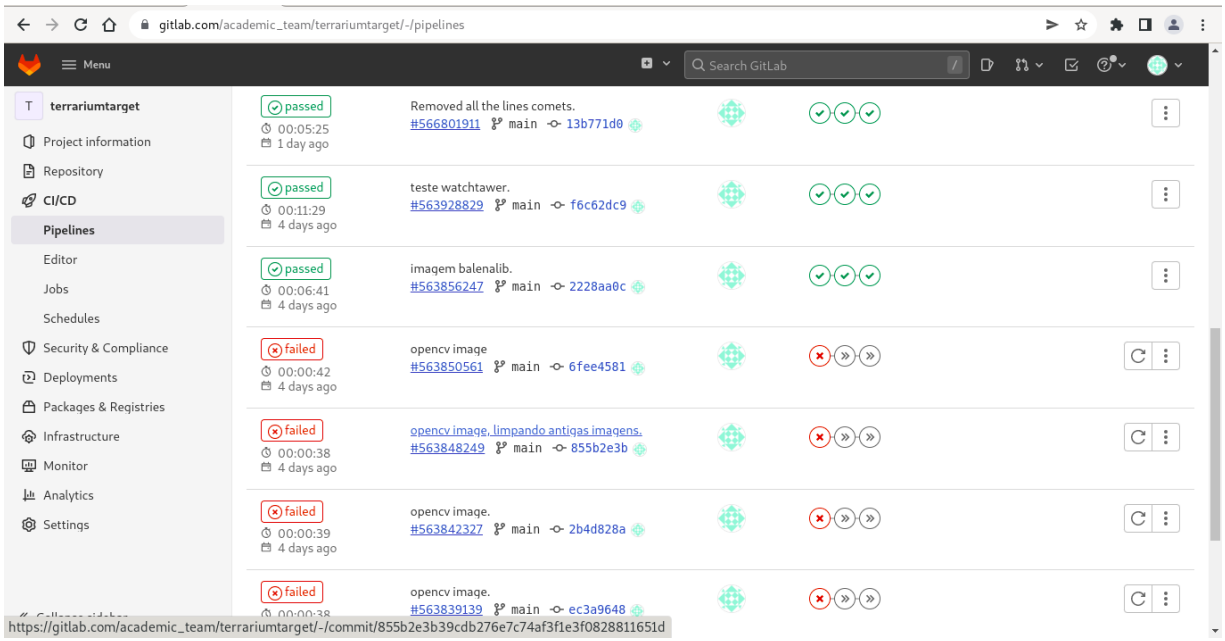


Painel com registro de execução. Fonte:(B.V., 2022b).

Com a criação dos passos automatizados da Pipeline, foi realizado a padronização de requisitos mínimos para implantação de novas versões do servidor (*server*), dentro desse, a execução de testes unidade coerentes.

O servidor implementado com protocolo de comunicação MQTT seção 2.2 possibilita fácil integração com uma aplicação cliente, que podem ser desenvolvidas com distintas abordagens. A comunicação bidirecional utilizada na troca de informações com o servidor possibilita o monitoramento e execução de ações remotamente. Este servidor permite acesso remoto e simultâneo de dispositivos as informações por ele gerada.

Figura 12



Painel de status de cada pipeline. Fonte:(B.V., 2022b).

Figura 13

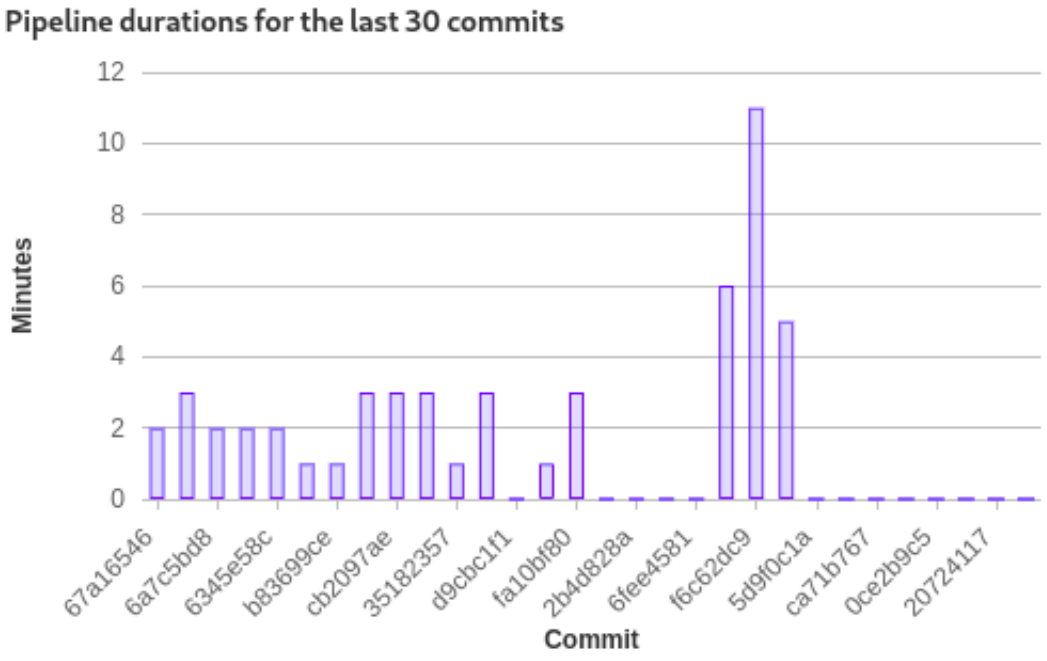


Gráfico *commits* por tempo de execução. Fonte:(B.V., 2022b).

Figura 14

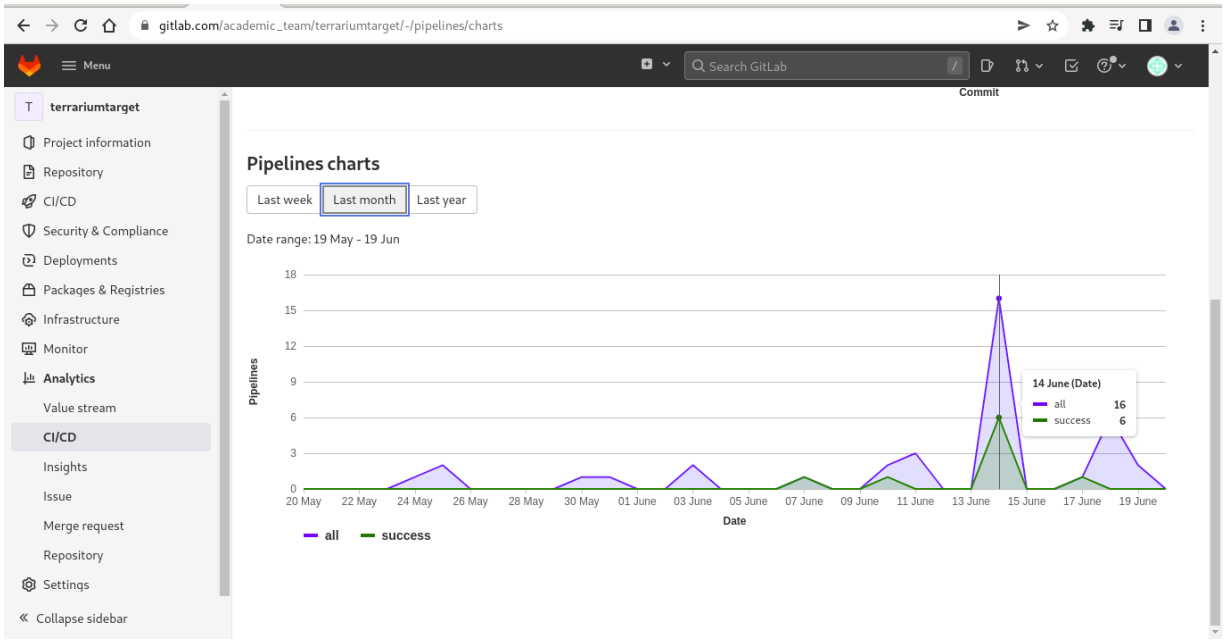


Gráfico quantidade de *commits* por dia. Fonte:(B.V., 2022b).

5 Conclusão

Com a construção do Pipeline CI/CD foi possível implementar testes de unidade, aplicar estrategias de controle de versionamento e pre-execução da imagem utilizada no ambiente do servidor, sendo essa aplicação, implantada de forma automatizada remotamente na placa Raspberry Pi. Disponibilizando serviços da aplicação via *Internet*.

Na construção da Pipilene CI/CD foram utilizada apenas ferramentas de código aberto, apoiando a cultura e possibilitando ao desenvolvedor apontar melhorias e contribuir na construção das soluções utilizadas. Sendo as licenças essas: GNU GPLv2, Creative Commons CC BY-SA 4.0, Python license e Apache 2.0.

O sistema de monitoramento do ambiente que roda pela aplicação servidor, utiliza componentes de relativo baixo custo e e de fácil implementação, a disponibilização dos artefatos criados no presente trabalho, por meio do repositório (PASTOR., 2022), permite a fácil replicação do sistema de monitoramento remoto.

5.1 Trabalhos Futuros

O terceiro requisito citado no trabalho de (PRENS et al., 2019), que estabelece a necessidade do monitoramento do consumo de energia e o tempo de inatividade durante o processo de atualização de um novo *software*, não é implementado no presente trabalho. Aplicar essa forma de levantamento de dados pode possibilitar analises que possam reduzir o consumos energéticos, uma questão relevante quando tratamos de dispositivos com limitação de recursos. Permitindo também propor melhorias na implantação de códigos para dispositivos IoT.

Referências

ALVAREZ-BENEDI, J.; MUNOZ-CARPENA, R. *Soil-water-solute process characterization: an integrated approach*. [S.l.]: CRC Press, 2004. Citado na página 26.

AMIRAULT, M. *GitLab Runner*. 2022. <https://docs.gitlab.com/>. Accessed: 2022-02-29. Citado na página 21.

AUTODESK, I. *AUTODESK Tinkercard*. 2022. <https://www.tinkercad.com/>. Accessed: 2022-05-08. Citado na página 33.

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 26.

BRUIJN., T. de. *Welcome to pyFirmata's documentation*. 2010. <https:https://pyfirmata.readthedocs.io/>. Accessed: 2021-09-08. Citado na página 24.

B.V., G. *GitLab*. 2022. <https://about.gitlab.com/company/>. Accessed: 2022-02-29. Citado na página 20.

B.V., G. *GitLab*. 2022. <https://gitlab.com/company/>. Accessed: 2022-06-20. Citado 3 vezes nas páginas 36, 37 e 38.

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 26.

DONATH., M. *Introduction*. 2022. <https://containrrr.dev/watchtower/introduction/>. Accessed: 2022-03-04. Citado na página 22.

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 18 e 19.

ELETRôNICOS., F. C. *FILIPEFLOP*. 2021. <https://www.filipeflop.com/>. Accessed: 2022-05-08. Citado 2 vezes nas páginas 25 e 27.

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 na página 18.

FOUNDATION, I. E. *Eclipse Mosquitto*. 2022. <https://mosquitto.org/>. Accessed: 2022-02-08. Citado na página 23.

FOUNDATION, I. F. S. *O Sistema Operacional GNU*. 2021. <https://www.gnu.org/philosophy/free-sw.pt-br.html>. Accessed: 2022-03-02. Citado na página 14.

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 23.

FOUNDATION., P. S. *doctest — Test interactive Python examples*. 2022. <https://docs.python.org/3/library/doctest.html>. Accessed: 2022-02-04. Citado na página 20.

FOUNDATION., T. R. P. *Teach, Learn, and Make with Raspberry Pi.* 2019. <https://docs.docker.com/get-started/overview/>. Accessed: 2021-08-08. Citado na página 24.

FRANCHI, C. M. *Acionamentos elétricos.* [S.l.]: Saraiva Educação SA, 2018. Citado na página 25.

GINGL, Z. et al. Universal arduino-based experimenting system to support teaching of natural sciences. In: IOP PUBLISHING. *Journal of Physics: Conference Series.* [S.l.], 2019. v. 1287, n. 1, p. 012052. Citado na página 25.

GIT. *Git.* 2019. <https://git-scm.com/book/>. Accessed: 2022-02-04. Citado na página 20.

INC., D. *Docker overview.* 2021. <https://https://www.raspberrypi.org/>. Accessed: 2022-03-08. Citado 3 vezes nas páginas 21, 22 e 24.

INITIATIVE., O. S. *Open Source Initiative.* 2020. <https://opensource.org/history>. Accessed: 2022-03-04. Citado na página 14.

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 3 vezes nas páginas 15, 18 e 19.

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 27.

MANCINI, M. Internet das coisas: História, conceitos, aplicações e desafios. *Project Management Institute–PMI*, 2017. Citado na página 16.

MCROBERTS, M. Arduino. *Básico-2ª Ed. São Paulo: Novatec*, 2015. Citado na página 25.

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 16.

NERI, M. L. e. G. B. R. *MQTT.* 2021. <https://www.gta.ufrj.br/>. Accessed: 2022-02-08. Citado na página 17.

OLIVEIRA, S. de. *Internet das coisas com ESP8266, Arduino e Raspberry PI.* [S.l.]: Novatec Editora, 2017. Citado na página 16.

PASTOR., B. H. *terrariumtarget.* 2022. <https://gitlab.com/academic_team/terrariumtarget.git>. Accessed: 2022-02-08. Citado na página 39.

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 13 e 19.

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 2 vezes nas páginas 18 e 39.

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 13.

RIBEIRO, M. A. Controle de processo. *Oitava edição± Tek Treinamento & consultoria± 2005*, 2005. Citado na página 25.

ROSENKRANZ, P. et al. A distributed test system architecture for open-source iot software. In: *Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems*. [S.l.: s.n.], 2015. p. 43–48. Citado na página 14.

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 2 vezes nas páginas 13 e 18.

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 2 vezes nas páginas 26 e 27.

SOMMERVILLE, I. *Engenharia de Software-9a Edição, Tradução*. [S.l.]: São Paulo, SP: Person Education, 2011. Citado na página 35.

TWITTER, I. *Welcome to the YAML Data Project*. 2020. <https://yaml.com/>. Accessed: 2022-03-08. Citado na página 21.

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 23.

UWUMUREMYI, G. N. *Design and implementation of an IoT-Based diabetes remote monitoring system*. Tese (Doutorado), 2021. Citado na página 23.

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 13 e 35.

XIA, F. et al. Internet of things. *International journal of communication systems*, v. 25, n. 9, p. 1101, 2012. Citado na página 13.