Projeto de Computação em Nuvem e Arquitetura Orientadas a Serviços: Weather Data Collection

Lucas A. Roris[1], **Pedro L. C. de Andrade**[1], **Rafael P. Gouveia**[1], **Vinicíus S. C. Paulo**[1]

[1]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo

{lucas.roris,pedroandrade,rafael.p.gouveia,vinicius.cubi}@usp.br

Abstract. *Due to the long distance between USP campi in São Carlos, an inconvenience arises when it doesn't rain at a campus closer to where students live, but rains at a distant one where they have classes. Therefore, a distributed weather monitoring system was developed, using various sensors integrated into ESP32 boards and communicating via MQTT protocol and Kafka messaging system. Additionally, a MongoDB database was used for logging purposes, and a web page was created to display the collected information in graphical form. The project was successful, and the pipeline proved to be suitable for the desired task.*

Resumo. *Devido à grande distância entre campi da USP de São Carlos, um inconveniente ocorre quando não chove em um campus mais próximo de onde os alunos moram, mas chove em um distante no qual eles terão aula. Por isso, criou-se um sistema de monitoramento de clima-tempo distribuído, usando diversos sensores integrados em placas ESP32 e que se comunicam usando protocolo MQTT e sistema de mensageria Kafka. Além disso, para fins de registro usou-se uma base de dados em MongoDB e para exibição dos dados foi criada uma página web que exibe em forma gráfica as informações coletadas. O projeto foi bem sucedido e a pipeline provou ser adequada para a execução da tarefa desejada.*

1. Introdução

Ao analisar-se o cenário tecnológico hordieno, percebe-se que a computação em nuvem torna-se, cada vez mais presente nos diferentes produtos e serviços oferecidos por instituições tanto públicas quanto privadas. Outrossim, esse no uso de serviços de nuvem deve-se à capacidade de armazenamento, processamento e compartilhamento de dados de forma flexível, segura e eficiente conferida às aplicações por meio do uso de tais serviços. Além disso, optar pelo uso de serviços em nuvem acarreta em reduções de custos ao se comparar com o uso e manutenção de servidores locais.

Neste contexto, a disciplina SSC0158 - Computação em Nuvem e Arquitetura Orientadas a Serviços teve como proposta, por meio de suas atividades ao longo do semestre, instruir os discentes a respeito de aspectos teóricos e práticos com relação às aplicações distribuídas a partir do modelo de computação em nuvem, em especial, aspectos de virtualização, arquiteturas orientadas a serviços, escalonamento e interfaces de programação neste setor. [1]

Dentre essas atividades encontra-se o desenvolvimento de fato de uma aplicação em nuvem, usando tecnologias pré-determinadas de ampla utilização no mercado de *software*, como o sistema de mensageria Kafka, bancos de dados não relacionais *etc*. O projeto de desenvolvimento dessa aplicação é o projeto descrito neste artigo.

2. Motivação e Objetivos

Visando oferecer aos ingressantes uma graduação que alinhe conhecimentos teóricos e práticos tanto de hardware quanto de software, em meados de 2003, a USP fundou o curso de Engenharia de Computação no *campus* de São Carlos. Para tal, foi reservado um espaço dentro de sua denominada Área 2, uma região a parte de seu *campus* que, embora mais extensa que a principal, ainda padece de muitos lotes vagos. [2]

Dessarte, a citada Área 2 fica em uma localidade afastada do centro da cidade, onde encontra-se repúblicas estudantis e conjuntos de prédios de locação, bem como o *campus 1* onde a maior parte das disciplinas ministradas por alunos de outros cursos, ou seja, onde reside a esmagadora maioria de estudantes. Considerando que as turmas de Engenharia de Computação possuem, majoritariamente, suas aulas nesta parte do campus, eles são impelidos a, diariamente, usar o transporte coletivo fornecido pela universidade para deslocarem-se entre as duas áreas. [2]

Percebe-se, com isso, que os alunos estão sujeitos a serem fortemente impactados por intempéries que, porventura, possam ser ocasionadas pelo clima. Ademais, este presente trabalho visa arquitetar uma aplicação que, satisfatoriamente, possa auxiliá-los a se planejar a inconvenientes desse tipo antes de irem para a universidade.

3. Descrição do Problema

Conforme retratado na seção anterior, observa-se o cenário em que estudantes frequentam suas aulas em diferentes locais distantes uns dos outros e das moradias do alunos. Por conseguinte, ter conhecimento de como está o tempo nos *campi* durante o dia é importante para que cada um possa levar roupas e acessórios adequados, no entanto, exitem desafios, como estar chovendo em um *campus* mas não em outro.

Por isso, propõem-se o desenvolvimento de um sistema que possa captar sinais do ambiente - tais como temperatura e umidade - a partir de três regiões diferentes, uma localizada no Instituto de Ciências Matemáticas e de Computação, localizado na área 1 da USP - São Carlos, e duas localizadas na área 2. Essas informações deverão ser, então, disponibilizadas para a fácil visualização dos usuários para os usuários. [2]

O desafio se encontra principalmente na transmissão de dados entre os dispositivos e o usuário. Para tanto deverão ser selecionadas e utilizadas diferentes tecnologias relevantes para a área de computação em nuvem de maneira apropriada para a resolução do problema.

4. Metodologia de Desenvolvimento

O desenvolvimento do projeto foi dividido em diversas etapas, cada um em sua própria tecnologia. Abaixo será explicado um pouco sobre o que foi feito em cada uma detalhadamente.

4.1. ESP32

A placa ESP32 é um módulo de desenvolvimento baseado no chip ESP32, um microcontrolador de baixo consumo de energia com Wi-Fi e Bluetooth integrados. Ela possui um processador dual-core de 32 *bits*, vários periféricos e uma quantidade generosa de memória. Com suporte a *Wi-Fi* e *Bluetooth*, a ESPWROOM32 é amplamente utilizada em projetos de Internet das Coisas (IoT), permitindo a conexão com a internet e a interação com outros dispositivos. Sua versatilidade, suporte a várias linguagens de programação e a disponibilidade de recursos e bibliotecas tornam essa placa acessível e popular entre desenvolvedores, oferecendo a flexibilidade necessária para criar uma ampla gama de aplicativos inteligentes e conectados. [3]

O módulo de desenvolvimento foi utilizado nesse projeto como mediador dos sensores de temperatura e umidade, SHT11 click e DHT22 respectivamente, criando uma interface de comunicação com o sistema de mensageria Kafka através do MQTT, informando os dados de temperatura e umidade em um intervalo de 5 minutos em duas localidades diferentes.

4.2. Kafka

O Apache Kafka é uma plataforma de streaming distribuída que permite a ingestão, armazenamento e processamento de fluxos de dados em tempo real. Ele foi projetado para lidar com grandes volumes de dados de forma eficiente e escalável, tornando-o adequado para casos de uso de alto *throughput* e latência baixa. [4]

No contexto do projeto descrito neste artigo, o Kafka é utilizado como um intermediário de mensagens para capturar os dados disponibilizados pelos sensores de temperatura e umidade e torná-las acessíveis para o *front-end* e para o sistema de *logging* implementado em MongoDB. O ESP32, equipado com os sensores *SHT11 click* e DHT22, envia as leituras desses sensores para o Kafka, que atua como um ”canal de comunicação”entre os dispositivos e os consumidores desses dados.

4.3. MQTT

O MQTT (Message Queuing Telemetry Transport) é um protocolo de mensagens leve e eficiente projetado para comunicações em rede de sensores e dispositivos com recursos limitados. Ele foi desenvolvido para permitir a troca de informações em tempo real de forma confiável e assíncrona, com baixa sobrecarga de rede e consumo de energia. [5]

No contexto do projeto descrito neste artigo, o MQTT é utilizado como o protocolo de comunicação entre os sensores de temperatura e umidade (ESP32) e os componentes do sistema, como o Kafka e o Servidor HTTP. Ele permite que os dispositivos enviem e recebam mensagens em formato de tópicos, seguindo o padrão publicador/assinante.

O MQTT opera em cima do modelo TCP/IP e utiliza uma arquitetura leve e simples. Ele consiste em três elementos principais: um cliente MQTT, um broker MQTT e um tópico MQTT. [5]

Cliente MQTT: É o dispositivo ou aplicativo que se conecta ao *broker* MQTT para enviar ou receber mensagens. No projeto em questão, o ESP32 atua como um cliente MQTT, sendo capaz de publicar mensagens (leituras de sensores) em tópicos específicos e assinar tópicos para receber mensagens de interesse.

Broker MQTT: É o intermediário de mensagens responsável por receber, encaminhar e entregar as mensagens entre os clientes MQTT. No caso do projeto, o Kafka funciona como um *broker* MQTT, recebendo as mensagens publicadas pelo ESP32 e encaminhando-as para os consumidores interessados, como o consumidor Kafka-MongoDB e o Servidor HTTP.

Tópico MQTT: É um canal virtual para o qual os clientes MQTT podem publicar mensagens ou do qual podem se inscrever para receber mensagens. Os tópicos são organizados em uma hierarquia de níveis, permitindo a filtragem e o roteamento eficiente das mensagens. No exemplo do projeto, os sensores publicam suas leituras nos tópicos de temperatura e umidade, enquanto os consumidores se inscrevem nesses tópicos para receber as mensagens correspondentes. [5]

4.4. MQTT-Kafka bridge

O MQTT-Kafka *bridge* é uma ponte de comunicação entre o MQTT e o Kafka, permitindo a comunicação e transferência de mensagens entre os dois sistemas. Essa ponte é implementada através de um *script* que consome mensagens do MQTT e as publica em um tópico do Kafka. Abaixo, analisaremos as principais partes do código e como a ponte MQTT-Kafka é estabelecida. [6]

4.4.1. Dependências

O código faz uso das seguintes bibliotecas:

- `paho.mqtt.client`: Uma biblioteca MQTT em Python que fornece as funcionalidades necessárias para a comunicação com o broker MQTT.
- `pykafka`: Uma biblioteca Python para interagir com o Kafka.

4.4.2. Função `on_message`

A função `on_message` é um *callback* que é invocado sempre que uma nova mensagem é recebida no MQTT. Ela é responsável por receber a mensagem, adicionar a data/hora atual ao início da string da mensagem e publicar essa mensagem no tópico do Kafka. A função utiliza o objeto `kafka_producer` para publicar a mensagem no tópico do Kafka.

4.4.3. Configuração do MQTT

Nessa seção, o código estabelece a conexão com o *broker* MQTT e configura o cliente MQTT. O *broker* MQTT utilizado é `mqtt.eclipseprojects.io`, e o cliente MQTT é criado com o identificador "MQTTBridge".

4.4.4. Configuração do Kafka

Aqui, o código estabelece a conexão com o *cluster* do Kafka. O endereço do *cluster* é configurado para `localhost:9092`. O código também define o tópico do Kafka chamado "climateInfo"e cria um produtor síncrono para esse tópico utilizando o objeto `kafka_topic`.

4.4.5. Transferência de mensagens

A parte final do código implementa um loop infinito que recebe mensagens do MQTT e as envia para o Kafka. O loop é controlado pelas chamadas `mqtt_client.loop_start()` e `mqtt_client.loop_stop()`, permitindo que as mensagens sejam processadas continuamente. Dentro do loop, o código se inscreve no tópico "climateInfo"do MQTT e define a função `on_message` como o *callback* para o recebimento de mensagens MQTT. As mensagens recebidas são processadas e enviadas para o Kafka através da chamada `kafka_producer.produce()`.

O *loop* é interrompido por um intervalo de 300 segundos (5 minutos) usando `time.sleep(300)`. Esse intervalo pode ser ajustado de acordo com os requisitos do sistema.

4.5. MongoDB

O MongoDB é um banco de dados *NoSQL*, altamente escalável e orientado a documentos, que se tornou uma escolha popular para aplicações modernas que precisam lidar com grandes volumes de dados não estruturados. Ele difere dos bancos de dados relacionais tradicionais ao adotar uma abordagem flexível de armazenamento de dados usando documentos *JSON-like* em vez de linhas em tabelas. [7]

No contexto do projeto, o MongoDB foi utilizado de forma a armazenar os dados recebidos pelos sensores em forma de *log* para possível análise posterior. Para tanto, bastou usar uma única coluna para armazenar todos os dados (indexados por *timestamp* e identificação do sensor).

4.6. Mongo-Bridge

A ponte entre o MongoDB e o Kafka é implementada por um *script* que consome mensagens do Kafka disponibilizadas no tópico *"climateInfo"* e armazena-as na única coluna *"data"* da base de dados MongoDB nomeada *"SensorData"*, usando as informações de data-hora como *id* para identificação das entradas de dado.

4.6.1. Dependências

O código responsável pela ponte faz apenas uso da seguinte biblioteca não padrão de Python: Pymongo, uma biblioteca de Python contendo ferramentas para trabalhar-se com MongoDB.

4.7. Análise de Dados

Após todo esse percurso dos dados, faz-se necessário uma forma de analisar esses dados que seja objetiva e eficiente. Desse forma, optou-se por uma solução simples e que atende a todos os requisitos: visualização gráfica. Assim, a partir do MongoDB, fez-se funções em código Python que captam os dados e - a partir da principal biblioteca utilizada para construir gráficos na linguagem já citada, a consagrada Matplotlib - são estruturados 4 gráficos, um par responsável por ilustrar as temperaturas obtidas em cada um dos sensores e o outro par representa os valores de umidade captados.

O processo descrito anteriormente é executado em um intervalo de uma hora, de forma a ter-se sempre as variações de temperatura e umidade em um intervalo de tempo que seja, ao mesmo tempo, curto e ideal para um observador externo verificar. Ademais, houve o empecilho de, considerando que a aplicação é hospedada em um $cluster$, não há maneira de se abrir imagem e, ainda mais, era necessário arquitetar uma solução que possibilitasse o acesso por qualquer indivíduo interessado. A saída encontrada será retratada na subseção a seguir.

4.8. Servidor HTTP

Para a visualização de todos esses dados, o grupo optou por hospedar um serviço HTTP responsável por apresentar visualmente os recursos disponibilizados pelos sensores para qualquer pessoa conectada à internet. Tendo em vista que o projeto seja mais focado na computação em nuvem, sensores distribuídos e hardwares heterogêneos, decidimos não priorizar excessivamente a visualização dos dados. No entanto, buscamos demonstrar por meio do código utilizado e do banco de dados um exemplo de uso da ferramenta.

O método escolhido envolve o uso de uma biblioteca Python, na qual hospedamos o servidor na porta 9041 (a porta externa disponibilizada pelo professor). Além disso, utilizamos um arquivo index.html como um exemplo de um possível site. Esse site carrega as imagens geradas por um programa que atualiza um novo gráfico a cada hora e o exibe no site.

4.9. *Scripts* de Inicialização e Finalização

Para garantir a execução correta de todos esses programas e facilitar a inicialização, desenvolvemos dois *scripts*: um para iniciar e outro para encerrar o sistema. O *script* de inicialização foi criado para executar os programas principais primeiro, como o Kafka, MQTT e Mongo, seguidos pelos programas mais simples, como o Consumidor Kafka-Mongo e o Servidor HTTP. Todos os programas são executados em paralelo, sendo criado um processo separado para cada um. Os PIDs (Identificadores de Processo) são armazenados para permitir o encerramento dos programas no futuro.

Por sua vez, o script de encerramento lê os PIDs armazenados em um arquivo e finaliza os programas um por um. Além disso, ele também executa alguns comandos para desligar corretamente o Mongo e o Kafka.

4.10. Esquemático Geral

De maneira geral, a aplicação desenvolvida neste presente trabalho pode ser resumido no seguinte esquemático:

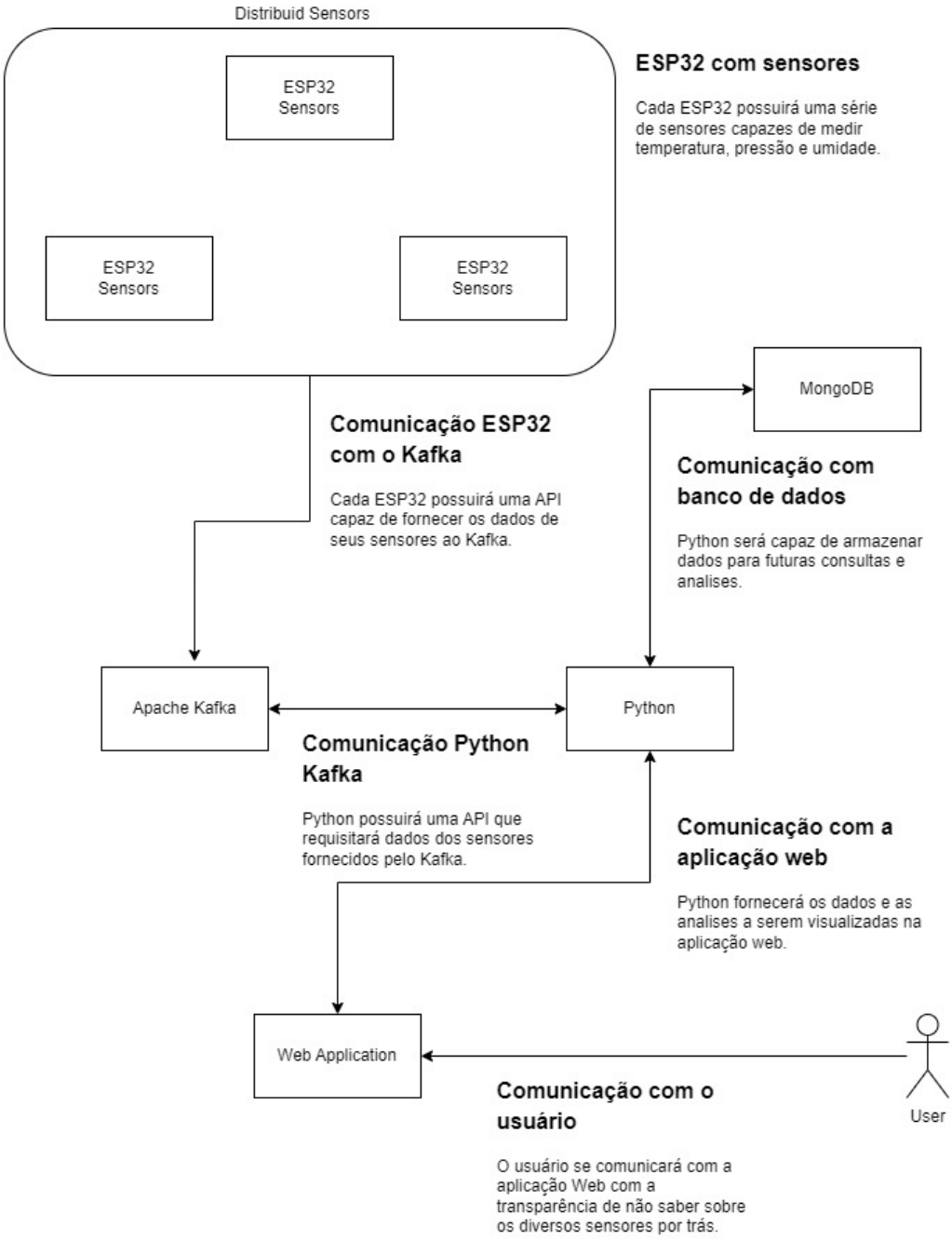

Figura 1. Diagrama da Aplicação.

5. Resultados

Com a correta produção de todas as etapas do *pipeline*, conforme elencadas na sessão anterior, obteve-se uma aplicação que, de maneira objetiva e eficiente, consegue captar os valores obtidos pelos sensores e deixá-los de fácil visualização ao usuário final.

Em síntese, a metodologia e o desenvolvimento da aplicação estruturada neste trabalho culmina em uma página web onde é possível verificar as variações de temperatura e umidade de cada um dos sensores.

Figura 2. Site do Projeto.

Destaca-se o fato dos dados serem atualizados a cada 1 (uma) hora. Dessa forma, conforme as preocupações levantadas nas sessões 2 e 3 sobre, respectivamente, motivação e descrição do problema, os gráficos gerados fornecem aos usuários informações suficientes para que possam se preparar adequadamente para o dia de aula no Área 2 da USP, $campus$ São Carlos

6. Conclusão

De forma concisa, percebe-se que o presente projeto apresentou uma problemática relevante no contexto dos estudantes da USP $campus$ São Carlos que majoritariamente possuem aulas na denominada Área 2. Dessarte, por meio dos conhecimentos de computação em nuvem adquiridos e aprofundados ao longo das aulas e atividades da disciplina em que este trabalho está inserido, estruturou-se e desenvolveu-se uma aplicação que seja capaz se auxiliar os citados alunos. Com isso, identifica-se a importância e o poderoso alcance que produtos e serviços que usufruem de computação em nuvem possuem.

O repositório contendo os códigos deste projeto pode ser acessado por meio da seguinte url: `https://github.com/ICMC-SSC0158-2023/gcloud03`. No momento, disponível apenas para usuários selecionados, para acessar os mesmos códigos em um repositório público, é possível acessar por meio da seguinte url: `https://github.com/rafael-p-gouveia/Sensoriamento_ESP32_Kafka`.

O *website* em que a aplicação é hospedada pode ser acessada pela url: `http://andromeda.lasdpc.icmc.usp.br:9041/`.

7. Forma de execução dos códigos

Temos duas opções para rodar o código, uma rodando diretamente os arquivos em uma VM já configurada no *cluster* e outra usando o Dockerfile. Explicaremos as duas formas

abaixo.

7.1. Diretamente na VM

Acessando a maquina localizada na porta 2041, toda a configuração e arquivos já estão lá, então apenas executar a comando abaixo para inicializar os programas.

```
$ bash start_script.sh
```

Uma vez os programas rodando sem nenhum erro, deve-se ligar os sensores para o envio de dados. Para interromper os programas rodando em *background*, executar o *script* abaixo.

```
$ bash kill_processes.sh
```

7.2. Dockerfile

Dentro dos arquivos disponibilizados, existe um arquivo chamado "Dockerfile", onde com ele é possível utilizar o programa *Docker* para gerar uma imagem de Docker e rodar em qualquer maquina com o comando *"docker run"*, sendo que automaticamente inicializa os programas intermediários e hospeda o servidor http.

Com o arquivo *Dockerfile* no diretório, executar o comando abaixo para gerar a imagem

```
$ docker build . --tag image_cluster_temp_sensor
```

Com a imagem gerada, executamos o comando abaixo para inicializar em *background* o Docker.

```
$ docker run -d -p 9041:9041 image_cluster_temp_sensor
```

Lembrando que a porta de acesso externo do servidor http funciona no *cluster* Andromeda, com a maquina na porta 2041 (maquina liberada para o acesso externo da porta 9041), caso executado em outra maquina, deverá ser trocado o valor das portas para o correto funcionamento. Esse comando com a *flag -p* redireciona a porta 9041 do *host* para a porta do 9041 Docker, esse valor pode ser alterado para as necessidades de quem estiver executando.

Se não for relatado nenhum erro, o servidor http está ligado esperando dados chegarem dos sensores para geração do gráfico.

OBS: Atualmente, durante a redação deste artigo, o servidor do *mongo-org* apresenta inconsistências no momento de instalar as versões especificas do mongo, o que pode apresentar falhas na instalação por conta de *Unable to locate package mongodb-org*, oriundo do servidor não enviar os *packages* corretos, necessitando a execução de diversas tentativas, e caso ainda sim o erro persistir, tentar a primeira opção de execução do código.

7.3. Códigos no ESP32

Para a compilação do código dos sensores com ESP32, foi utilizado a IDE do Arduino, com uma biblioteca para compilação para diversos tipos de ESP32 (*https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json*). O código para compilação deverá ser escolhido

de acordo com o sensor disponível, na pasta há códigos para os sensores SHT11 e DHT22. Com o *setup* da IDE pronto, trocar o valor da variável *ssid* e *password* para a rede *Wi-Fi* onde o sensor ficará coletando dados. Após compilar e fazer *upload* do código, caso o LED do ESP esteja piscando, significa que ele está enviando os dados.

Para o correto funcionamento da aquisição de dados os sensores devem estar nos pinos indicados pelo código, por exemplo, no caso de SHT11 os pinos SDA e SCL são ligados respectivamente nos pinos 17 e 16. Caso seja necessário uma mudança, deve-se consultar o *datasheet* do microcontrolador para garantir que a nova porta suporta esse tipo de conexão, e atualizar no código com os novos valores de pinos.

8. Referências

[1] UNIVERSIDADE DE SÃO PAULO (SP). SSC0158: Computação em Nuvem e Arquitetura Orientadas a Serviços. In: UNIVERSIDADE DE SÃO PAULO (SP). Júpiter : Sistema de Gestão Acadêmica da Pró-Reitoria de Graduação. [S. l.], 15 jul. 2020. Disponível em: https://uspdigital.usp.br/jupiterweb/obterDisciplina?sgldis=SSC0158codcur=97001codhab=0. Acesso em: 25 jun. 2023.

[2] UNIVERSIDADE DE SÃO PAULO (SP). Curso de Engenharia de Computação. In: UNIVERSIDADE DE SÃO PAULO (SP). EESC-USP. [S. l.], 15 jul. 2020. Disponível em: https://eesc.usp.br/graduacao/curso.php?id=97001. Acesso em: 25 jun. 2023.

[3] ESPRESSIF SYSTEMS (China). ESP32. In: ESPRESSIF SYSTEMS (China). ESPRESSIF. [S. l.], 15 jul. 2020. Disponível em: https://www.espressif.com/en/products/socs/esp32. Acesso em: 25 jun. 2023.

[4] APACHE SOFTWARE FOUNDATION (EUA). Apache Kafka. In: APACHE SOFTWARE FOUNDATION (EUA). Apache Kafka. [S. l.], 15 jul. 2020. Disponível em: https://kafka.apache.org/. Acesso em: 25 jun. 2023.

[5] OASIS (EUA). MQTT: The Standard for IoT Messaging. In: OASIS (EUA). MQTT. [S. l.], 15 jul. 2020. Disponível em: https://mqtt.org/. Acesso em: 25 jun. 2023.

[6] BANKS, Andrew; BRIGGS, Ed; BORGENDALE, Ken; GUPTA, Rahul. OASIS Standard: MQTT Version 5.0. [S. l.]: OASIS, 7 mar. 2019. Disponível em: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. Acesso em: 25 jun. 2023.

[7] MONGODB. MongoDB. In: MONGODB. MongoDB. [S. l.], 11 fev. 2009. Disponível em: https://www.mongodb.com/. Acesso em: 25 jun. 2023.