Implementação de uma arquitetura de referência para criação de sistemas de monitoramento e coleta de dados para apoiar a tomada de decisão em desastres utilizando Internet das Coisas

Marcus Vinicius de Oliveira Marques[1], Frâncila Weidt Neiva[1]

[1]Curso de Bacharelado em Sistemas de Informação – Centro de Ensino Superior de Juiz de Fora (CES/JF) – Campus Academia 36016-000 – Juiz de Fora– MG– Brasil

viniciusmarques8@gmail.com, francilaneiva@cesjf.br

Abstract. *This article demonstrates the implementation of an architecture using IoT and cloud computing to assist in obtaining reliable environmental data and to provide better decision making in the event of human or nature disasters. This architecture was created with the purpose of having scalability, extensibility and interoperability characteristics, aiming to provide greater adaptation to the different scenarios found in cases of natural disasters. The designed structure also makes it possible to use it as an incident monitoring system, fully inclined to incident prevention as well as pre-prepared disaster caused by nature.*

Resumo. *Este artigo demonstra a implementação de uma arquitetura utilizando Internet das Coisas e Computação em nuvem para auxiliar na obtenção de dados fidedignos do ambiente e proporcionar uma melhor tomada de decisão em casos de desastres causados pelo homem ou natureza. Esta arquitetura foi criada com o objetivo de possuir características de escalabilidade, extensibilidade e interoperabilidade, com o propósito de proporcionar maior adaptação aos diferentes cenários encontrados em casos de desastres naturais. A estrutura desenhada também possibilita sua utilização como um sistema de monitoramento de incidentes, totalmente inclinada na prevenção de incidentes como também na preparação prévia para desastres causados pela natureza.*

1. INTRODUÇÃO

Desde os primórdios da civilização que temos registros, crises e desastres naturais vêem causando diversos prejuízos para a sociedade, tanto em questões financeiras, como ambientais e sociais. Além dos desastres naturais, cada vez mais incidentes causados pelo homem estão ocorrendo. Somente no Brasil nos últimos 4 anos, pode-se dois casos de grande impacto, como o rompimento da barragem da cidade de Mariana no estado de Minas Gerais em 2015, onde segundo FREITAS *et al*. (2016) "ocorreram 19 óbitos, sendo dois terços de trabalhadores terceirizados.". Outro caso semelhante que ocorreu no mesmo estado, foi no município de Brumadinho em janeiro de 2019.

Na época desses acidentes, popularmente, muito questionou-se às autoridades os motivos de não conseguirem prever estas fatalidades, considerando a evolução tecnológica que temos acesso na atualidade em equipamentos de monitoramento, como citado por GASCÓN(2015) "[...] Basicamente, quase qualquer desastre natural pode ser prevenido, ou, ao menos, ter os danos minimizados". No caso da cidade de Mariana dois parâmetros poderiam

ser utilizados para antecipar os acontecimentos, como exemplos temos sensores de umidade do solo, onde sensores posicionados em diferentes níveis de profundidade no solo poderiam verificar constantemente pontos de infiltrações, em união temos sensores de vibração e pressão que estrategicamente posicionados poderiam indicar micromovimentações suspeitas.

Sensores como os citados acima estão se tornando populares pela crescente demanda por automação e baixo custo, como citado por AGUIRRE(2017).

> [...] a evolução da engenharia de sensores associada a tecnologia de informação e serviços de telecomunicações surge para auxiliar na criação de projetos de baixo custo de implantação e manutenção. A revolução da Internet das Coisas (IoT – Internet of Things) já começou, e sensores que custam menos de U$ 10 são realidade.

Porém, para interligar todos sensores que são responsáveis pelas coletas de dados proveniente do ambiente, ainda é necessário uma outra tecnologia, a Internet das Coisas (IoT), mencionada por AGUIRRE (2017), na citação acima como uma revolução. A IoT segundo ALBERTIN *et al*. (2017) "pode ser entendida como a rede ubíqua e global que ajuda e provê a funcionalidade de integrar o mundo físico e possui um poder incrível para auxiliar as mais variadas tarefas, que vão desde problemas comuns aos mais complexos." Portanto, o uso de Internet das Coisas tem sido internacionalmente pesquisado MARJANI *et al*. (2017).

O presente artigo visa contribuir com o desenvolvimento de um artefato computacional, baseado em Internet das Coisas, de baixo custo, interoperável, escalável e extensível, capaz de prover informações coletadas via sensores distribuídos que permitam monitorar fatores interessantes à prevenção de crises e desastres.

2. TRABALHOS RELACIONADOS

Diferentes pesquisas na literatura investigam como a tecnologia pode auxiliar a humanidade nas questões de contingência e mitigação dos impactos causado pela natureza ou pelo homem em situações de crises e desastres. Alguns estudos são focados em tecnologia social como HALSE *et al*. (2018), onde busca-se ligar os dados gerados por usuários nas redes sociais, como *Twitter* com sensores coletores de dados provenientes de redes de metrô sendo um dos objetivos do trabalho também obter mais dados através das postagens dos usuários para complementar o dados coletados pelos sensores e prover uma resposta mais rica em informações.

A falta de informação acurada após um desastre dificulta a tomada de decisão das pessoas. Os responsáveis por tomar ações após algum incidente ficam desorientados por não conseguirem encontrar informações confiáveis para agir e garantir auxilio para aqueles que estão sofrendo os impactos dos desastres PAULUS *et al*. (2018).

Sempre houve muita dificuldade de se antecipar um desastre, muito se deve por se tratar de um evento repentino como citado por PARIZZI (2014):

> Desastre pode ser associado a um evento imprevisto, que ocorre muitas vezes de forma súbita, e que causa grande dano, destruição e sofrimento humano. Os desastres são a convergência dos perigos com vulnerabilidades. Como tal, um aumento da vulnerabilidade ambiental, social ou econômica pode significar um aumento da frequência das catástrofes.

Essa natureza imprevisível e destrutiva de um desastre acaba por gerar situações de crises e desencontros de informações, em MORDECAI *et al*. (2018), há a preocupação de criar um *dashboard* com informações precisas para o gerenciamento de dados gerados em situações de calamidades.

A IoT entra no cenário em que o objetivo é prever com maior precisão e antecedência o suficiente desastres, como citado por PATRICIO *et al*. (2018) para evitar desastres em infraestruturas, como vazamentos em oleodutos, emergências em rodovias e defeitos na estrutura de pontes e linhas de transmissão.

Com foco no cenário de ajuda à prestação de serviço humanitário, através do fornecimento de dados, este trabalho visa prover uma implementação de referência com características baseadas em alicerces importantes como os apresentados por FURQUIM (2017, *apud* Seal *et al*., 2012):

> Assim, redes de sensores sem fio são uma opção vantajosa para realizar o monitoramento de ambientes naturais, já que apresentam as seguintes características: (i) baixo custo, principalmente em relação à infraestrutura, (ii) acessibilidade em ambientes inóspitos, (iii) simples desenvolvimento e implantação, (iv) possibilidade de utilização de sensores de alta precisão, além da (v) adaptabilidade a mudanças ambientais.

Em união com pontos fortes para garantir a integridade da implementação proposta, além dos alicerces propostos acima, o desenvolvimento apresentado neste trabalho levou em consideração questões de reprodutibilidade, para que assim seja possível que este atual projeto seja implementado em outros centros de pesquisa e seja capaz de apoiar outros programas de monitoramento e gestão de riscos em resposta aos desastres. Como citado por KOBIYAMA *et al*. (2004) "[...]as ações integradas entre comunidades são fundamentais para que os efeitos dos desastres naturais sejam mitigados. As universidades devem contribuir na compreensão de diagnósticos dos desastres naturais e seus mecanismos através de monitoramento e modelagem".

3. OBJETIVO

O objetivo geral deste trabalho é demonstrar um modelo arquitetural funcional utilizando recursos de baixo custo para prover informações coletadas diretamente do ambiente a ser monitorado capaz de alimentar um sistema de *dashboard*.

Isto é, espera-se que seja possível utilizar esta implementação para abastecer a comunidade de informações precisas sobre situações climáticas de regiões, como também mapear riscos envolvendo movimentações de terra e outros tipos de situações possíveis de serem captados por sensores criados para Internet das Coisas, visando a possibilidade de antecipar com maior assertividade possíveis desastres naturais.

A solução adotada para essa pesquisa visa dispor os dados tanto para a população leiga como para a população altamente técnica como: pesquisadores, professores, profissionais e estudantes com foco no impacto do clima nos desastres nas regiões habitadas.

A utilização de diversas tecnologias se fez necessária, como recursos de infraestrutura de Computação na Nuvem para alta disponibilidade e sensores para coleta de dados que serão amplamente exploradas. As tecnologias utilizadas estão apresentadas na próxima seção. Para atingir o objetivo proposto neste trabalho, algumas atividades de cunho técnico também foram necessárias, das quais destacam-se:

- Analisar modelos de placas de microcontroladores para o caso;
- Criar modelagem da solução do trabalho;
- Implementar o código referente aos microcontroladores;
- Criar a arquitetura modelada como exemplo para replicação em outros casos.

4. TECNOLOGIAS

Esta seção apresenta para o leitor as principais tecnologias utilizadas no desenvolvimento da arquitetura deste trabalho.

4.1. MICROCONTROLADOR

Para a implementação, essa pesquisa irá utilizar a placa denominada ESP8266, como demonstrado na Figura 01.

Figura 01: **ESP8266 NodeMCU modelo LoLin. (Fonte: www.eletronicwings.com, acessado em set/2019)**

Com a crescente popularização da Internet das Coisas as placas de microntroladores em geral possuem um custo baixo para aquisição, esta placa possui a capacidade de se comunicar com protocolo de rede nativamente, como necessário para o desenvolvimento da arquitetura.

Todo código gerado para este microcontrolador é totalmente compatível com outras variantes de placas de microcontroladores existentes no mercado, sendo possível escalar a solução de forma fácil sem a preocupação de acoplamento da solução ao *hardware*.

4.2 . SENSORES

Os sensores são fundamentais para coleta de dados provenientes do ambiente, principalmente aqueles que fazem parte da categoria de sensores sem fio. Estes possibilitam criar de forma mais distribuída uma rede de sensores que podem ser divididos em dois grupos:

- **Sensores digitais:** são sensores que devolvem apenas saídas binárias, ou seja, possuem menos precisão de resposta para soluções que necessitam medir uma escala de valores, porém são ótimos em soluções que demandam o uso de estados tais como: ligado-desligado, cima-baixo, zero-um, etc;
- **Sensores analógicos:** são sensores que devolvem um intervalo de valores como saída, são excelentes para expor uma escala de valores em forma de intensidade, tais como: nível de umidade do ar, nível de umidade do solo, nível de gás metano no ar e quantidade de água em um repositório.

Para a implementação deste trabalho diferentes tipos de sensores serão utilizados, porém, como forma de viabilizar uma prototipação inicial e a fim de demonstrar a capacidade de como é possível escalar a solução a níveis mais elevados de utilização, será realizado também a simulação de sensores a nível de software, via codificação na camada do microcontrolador.

4.3. PROTOCOLO DE COMUNICAÇÃO

O MQTT é um protocolo de comunicação leve criado pela IBM para troca de mensagens comumente utilizado entre sensores-sensores, sensores-microcontrolador ou microcontrolador-microcontrolador. Funciona no padrão de *Publish/Subscribe* onde pode-se

criar um tópico e publicar os dados diretamente em cada tópico criado. Caso alguém se interesse pela obtenção desses dados, basta se inscrever no tópico desejado. De acordo com CONCEIÇÃO *et al*. (2018):

> [...] Nessa arquitetura, o dispositivo é responsável por enviar (publish) as informações ao servidor, que opera como um intermediário (broker). Tendo conhecimento dos clientes que estão interessados nas informações enviadas (subscribers), o broker retransmite as informações recebidas.

Esse protocolo está sobre a camada de aplicação do modelo de TCP, esse protocolo garante integridade da comunicação dos dados trafegados.

O protocolo de MQTT utiliza-se de um ponto centralizador para comunicação dos dados, denominado *Broker* e há diversos tipos de *Brokers* disponíveis para utilização, tanto gratuitos quanto pagos. Como dito por CONCEIÇÃO *et al*. (2018) "[...] o *broker* é responsável por receber todas as mensagens, filtrá-las, determinar quem deve receber cada mensagem e enviar essas mensagens para os destinos".

4.4. COMPUTAÇÃO EM NUVEM

Segundo COUTINHO et al. (*2013 apud* Sá et al. 2011), a Computação em Nuvem propõe a integração de diversos modelos tecnológicos para o provimento de infraestrutura de *hardware*, plataformas de desenvolvimento e aplicações na forma de serviços sob demanda com pagamento baseado em uso.

É um serviço de alta disponibilidade com foco em escalabilidade e elasticidade, isso visa possibilitar a comunicação de dados via web, esta implementação utiliza Computação em Nuvem considerando que há a necessidade de ser possível acessar os dados gerados pelos sensores captados pelos microcontroladores de qualquer parte do mundo como também incrementar a infraestrutura utilizada caso necessário, sem causar indisponibilidade da mesma. Para isso, algumas soluções em nuvem serão adotadas, tais como:

- **CloudMqtt** - Plataforma em nuvem para receber dados gerados a partir do protocolo de comunicação MQTT;
- **Heroku** - Plataforma em nuvem para hospedar o código do servidor que consome os dados que estarão na nuvem do protocolo MQTT;
- **Mlab** - Nuvem onde estará situado o banco de dados da solução e onde o servidor irá armazenar os dados captados;

Vale ressaltar que um requisito importante do projeto é relacionado a segurança dos dados trafegados entre as placas dos microcontroladores e a plataforma responsável por disponibilizar as informações na nuvem. Para que os dados não sejam modificados por outra pessoa invalidando seus resultados a plataforma *Cloud* MQTT dispõe de autenticação para utilização. Com esse fato é possível limitar quem é permitido publicar nos tópicos da plataforma e garantir que os dados irão manter sua integridade do momento da coleta no ambiente até sua exposição na nuvem.

5. METODOLOGIA

A metodologia de pesquisa deste trabalho foi dividida em definição da proposta e refinamento da proposta. A definição da proposta envolveu a realização de uma revisão da literatura (Seção 2) a fim de gerar conhecimento sobre o tema da pesquisa. Com base nesse conhecimento, a proposta foi elaborada. O refinamento da proposta envolveu a realização de estudos

experimentais, em especial relativos a análise de placas de microcontroladores, bem como de tecnologias disponíveis que pudessem impor restrições à proposta. Em seguida, as atividades de modelagem da solução, desenvolvimento da arquitetura, implementação e avaliação da proposta foram realizadas. Tais passos previstos na metodologia de pesquisa são apresentados nas seções seguintes.

5.1. PLANO DE IMPLEMENTAÇÃO

Para seguir a implementação do projeto é importante notar que para todas as integrações realizadas, diferentes camadas de desenvolvimento foram utilizadas para a conclusão do projeto, como pode-se notar na Figura 02.

O predominante para essa escolha foi garantir uma implementação de arquitetura modular para facilitar a manutenção individual de cada camada, como também a possibilidade de substituição das tecnologias utilizadas sem propagar o impacto da alteração para outros componentes do projeto. Com isso é possível trocar os modelos dos microcontroladores sem necessitar modificar a estrutura de *backend*.

A camada mais baixa do projeto é onde se encontram os sensores que se comunicarão com o microcontrolador. Estes sensores serão tanto físicos como simulados para demonstrar a escalabilidade a nível de componentes do projeto. A linguagem adotada para a implementação desta camada foi a linguagem C, comum para implementações de componentes para microcontroladores e sensores. Em seguida, externalizando os dados coletados pelos sensores, o microcontrolador publica os dados na nuvem, utilizando-se do protocolo de comunicação MQTT. Estes dados são livres para consultar a aplicação *backend* e alimenta uma base de dados, além de prover o sistema de dados históricos.

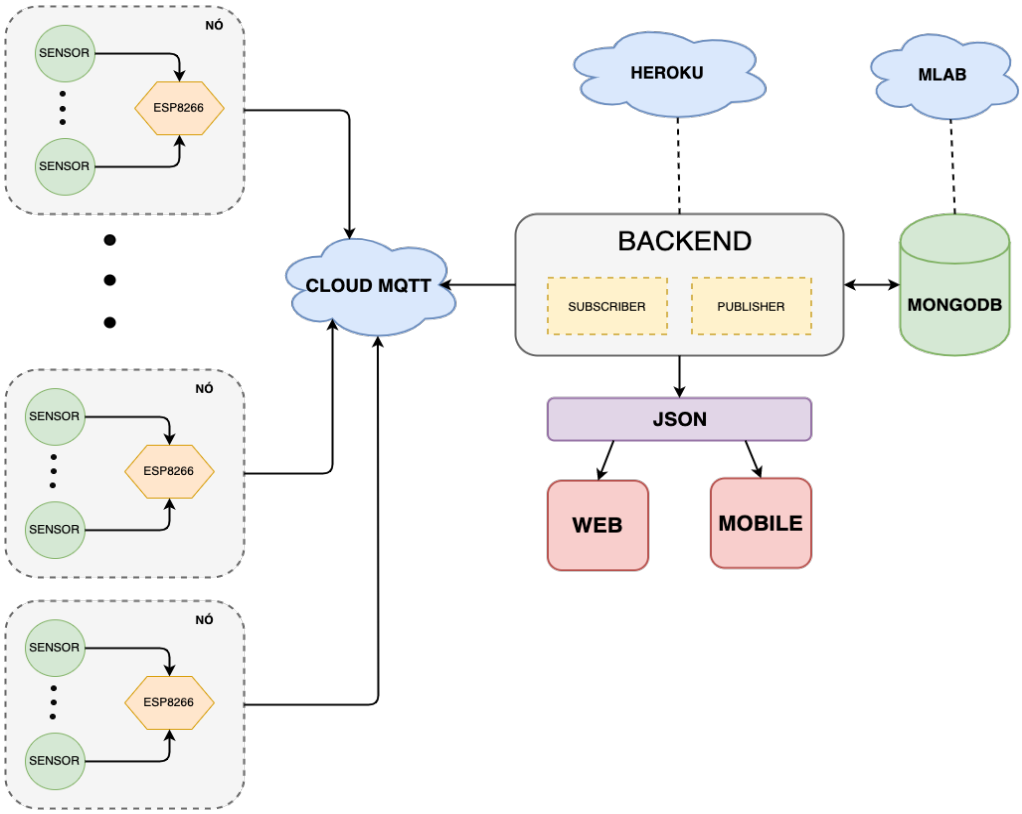

Figura 02: Design da arquitetura da proposta.(Fonte: Autor, 2019)

A implementação dessa camada de *backend* consiste em filtrar os dados recebidos pelos sensores e microcontroladores para abastecer a base de dados.

A base escolhida para suprir essa solução foi o MongoDB, base de dados não relacional que proporciona uma série de facilidades durante a implementação utilizando o conceito de *tableless*, ou seja, não há necessidade de uma modelagem prévia de como os dados devem ser armazenados na base e, com isso, pode-se criar uma solução flexível o bastante para a adaptação aos mais diferentes tipos de inclusão de itens. O MongoDB possui uma excelente velocidade de inserção, fator primordial para o sistema, considerando que os sensores geram informações em alta velocidade.

6. IMPLEMENTAÇÕES

Nesta seção os principais pontos de implementação da solução proposta são apresentados.

6.1. MICROCONTROLADORES E SENSORES

Inicialmente a atividades de implementação da proposta se iniciou pela implementação do código fonte dos microcontroladores e suas respectivas conexões com os sensores, como também está presente nas placas de ESP8266, os trechos de códigos simulando outros tipos de sensores.

O código fonte compatível com as placas é sobre a linguagem C, porém possuem uma padronização específica para esse tipo de *hardware*. Para o funcionamento correto das instruções passadas para as placas microcontroladores é mandatório a implementação de duas funções.

- ***Setup***: ideal para inicializar variáveis e outras funções nas placas, essa função é executada apenas uma vez durante o *bootstrap* do microcontrolador.
- ***Loop***: ideal para executar instruções de forma cíclica e infinita, ou seja, após o *setup*, enquanto houver energia no microcontrolador ele executará essa função indefinidamente.

No desenvolvimento deste trabalho a função de *bootstrap* Setup foi definida da seguinte maneira:

```
1. void setup() {
2.   Serial.begin(115200);
3.   setup_wifi();
4.   client.setServer(mqtt_server, 16418);
5.   client.setCallback(callback);
6. }
```

Figura 03: Inicialização do microcontrolador. (Fonte: Autor, 2019)

Esta função está primeiramente inicializando a porta serial onde o código irá enviar a saída do programa sendo executado, o valor de 115200 é referente a frequência da saída. Importante notar que este valor necessita ser escolhido de acordo com a capacidade da placa de microcontrolador adquirida, para visualizar no console este valor de frequência deverá ser utilizado para obter os dados de saída, logo após é chamada a função *setup_wifi*, ela é responsável por inicializar no microcontrolador as funções referente ao protocolo de comunicação, ou seja, ao executar essa função com sucesso, o ESP8266 pode-se comunicar com o ambiente externo e expor os dados coletados pelos sensores.

```
1. void setup_wifi() {
```

```
2.
3.    delay(10);
4.    Serial.println();
5.    Serial.print("Connecting to ");
6.    Serial.println(ssid);
7.    WiFi.mode(WIFI_STA);
8.    WiFi.begin(ssid, password);
9.
10.   while (WiFi.status() != WL_CONNECTED) {
11.     delay(500);
12.     Serial.print(".");
13.   }
14.
15.   randomSeed(micros());
16.
17.   Serial.println("");
18.   Serial.println("WiFi connected");
19.   Serial.printsln("IP address: ");
20.   Serial.println(WiFi.localIP());
21.}
```

Figura 04: Inicialização do módulo WiFi. (Fonte: Autor, 2019)

Após executar com sucesso a conexão via *wireless* a próxima função a ser requisitada é referente ao protocolo MQTT, *client.setServer(mqtt_server*, 16418). Essa chamada define onde está localizado a nuvem ou servidor MQTT no primeiro parâmetro, já no segundo é definido qual é a porta deste servidor. A função de *loop* é onde o microcontrolador irá publicar os dados coletados dos sensores para o servidor MQTT na nuvem.

```
1. void loop() {
2.
3.    if (!client.connected()) {
4.      reconnect();
5.    }
6.    client.loop();
7.
8.    long now = millis();
9.    if (now - lastMsg > 2000) {
10.     lastMsg = now;
11.     ++value;
12.     snprintf (msg, 50, "Cain ON [N2][#%ld]", value);
13.     Serial.print("Publish message: ");
14.     Serial.println(msg);
15.     client.publish("node1", msg);
16.   }
17.
18.   // INICIO DE SIMULADORES
19.
20.   temperatureSimulator();
21.   humitySimulator();
22.   gasSimulator();
23.   waterSimulator();
24.   gasConcentrationSimulator();
25.   airPressureSimulator();
26.   soilHumitySimulator();
27.
28.   delay(300000);
29.   // FIM
30.}
```

Figura 05: Iteração principal do microcontrolador. (Fonte: Autor, 2019)

Nessa função é definida tarefas importantes que devem ser verificadas sempre. O microcontrolador na implementação atual não pode se manter sem a conexão wireless, com isso, nesse *loop* é verificado sempre se a placa ESP8266 está conectada com a rede WIFI, caso esteja é chamada a função para reconexão.

Finalizando o código do microcontrolador é executada a chamada dos simuladores dos sensores. Para este cenário foram definidos alguns simuladores: temperatura, umidade, gases e água.

```
1.  void temperatureSimulator() {
2.      int randNumber = random(17,36);
3.
4.      char* temp = new char[32];
5.      String(randNumber).toCharArray(temp,32);
6.      Serial.println("Temperature Simulator");
7.      Serial.println(temp);
8.
9.      client.publish("Cain/FTemperature", (char*)  temp);
10. }
11.
12. void humitySimulator() {
13.   int randNumber = random(40,101);
14.
15.     char* humity = new char[32];
16.     String(randNumber).toCharArray(humity,32);
17.     Serial.println("Humity Simulator");
18.     Serial.println(humity);
19.
20.   client.publish("Cain/FHumity", (char*)  humity);
21. }
```

Figura 06: Implementação dos simuladores. (Fonte: Autor, 2019)

Os trechos de código acima são responsáveis por receber os dados coletados e publicar para o *Cloud* MQTT, o servidor localizado na nuvem.

6.2. BACKEND

O código fonte do servidor pode ser dividido em duas partes:

- **MQTT *Publish/Subscribe***: esta camada é responsável por lidar diretamente com o protocolo de comunicação no servidor de *Cloud MQTT* na nuvem.
- **NodeJS**: esta camada trata os dados recebidos, nela é possível filtrar as informações e transformá-las para melhor utilização na camada de *frontend* com o usuário final.

O código a seguir na Figura 07 é onde se define a comunicação com o servidor de MQTT exposto na nuvem.

```
1.  const con = mqtt.connect("mqtt://soldier.cloudmqtt.com",
2.                  {clientId:"========= HEROKU _ NODE ============",
3.                      username: 'bvqnjxaz',
4.                      password: 'mRCgh4BPQqGw',
5.                      port: 16418});
```

Figura 07: Conexão com a plataforma MQTT. (Fonte: Autor, 2019)

Como trata-se de uma conexão MQTT, o *backend* necessita após conectado, se inscrever nos tópicos de interesse para consumir os dados publicados pelo microcontrolador.

```
const topic_list=["Remo/Gas'",
                  "Remo/Temperature",
                  "Remo/Humity",
                  "Remo/SoilHumity",
                  'Remo/Check',
                  'FRemo/Temperature',
                  'FRemo/Vibration',
                  'FRemo/SoilHumity',
                  'FRemo/WaterDetector',
                  'FRemo/AirPresure',
                  'FRemo/GasConcentration',
                  'Cain/FHumity',
                  'Cain/FTemperature',
                  'Cain/FAirPressure',
                  'Cain/FGasConcetration',
                  'Cain/FSoilHumity',
                  'Cain/FWater',
                  'Abel/SoilHumity',
                  'Abel/Vibration',
                  'Abel/Water'
                ];

                [...]

                con.subscribe(topic_list)
                //.setMaxListeners(1)
                con.on('message', function (topic, message) {
                    console.log(topic)
                    console.log(enc.decode(message))

                    MqttDataModel.create({
                        "topic": topic,
                        "data": enc.decode(message)
                    });
                });
```

Figura 08: Declaração dos tópicos e inscrição na plataforma MQTT(Fonte: Autor, 2019)

Após ler as mensagens publicadas (Figura 08) no servidor na nuvem é necessário salvar esses dados em um banco de dados, sendo o MongoDB utilizado no escopo deste trabalho. Uma conexão com o banco de dados fazendo referência à biblioteca Mongoose foi utilizada, como exibido no trecho a seguir na Figura 09:

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://admin:123456a@ds123500.mlab.com:23500/vvtcesjfvmarques', { useUnifiedTopology: true });
mongoose.Promise = global.Promise;
console.log(mongoose.connection.readyState);
module.exports = mongoose;
```

Figura 09: Inicialização da base de dados. (Fonte: Autor, 2019)

7. RESULTADOS E DISCUSSÃO

Esta seção apresenta avaliações da implementação da arquitetura, bem como discussões sobre os resultados obtidos.

7.1. PLANO DE AVALIAÇÃO

O plano de avaliação deve demonstrar como a implementação cumpre os seguintes requisitos não funcionais:

i) A interoperabilidade como refere-se BAPTISTA(2015, *apud* Woodley - tradução livre) é "a capacidade de tipos diferentes de computadores, redes, sistemas operativos e aplicações trabalharem em conjunto com eficácia, sem comunicação prévia, de forma a trocarem informação de uma maneira útil e com significado".

Para avaliar a interoperabilidade, testou-se um cenário capaz de demonstrar como um dado é extraído diretamente do ambiente passando por variadas plataformas, utilizando diferentes linguagens e chegando ao usuário final de forma correta e transparente.

Existem diferentes tipos de definições de escalabilidade nos mais variados meios onde este termo é aplicado. Neste projeto o conceito de escalabilidade objetivado está relacionado à computação em nuvem, como citado por RUSCHEL *et al*. (2008)

> Podemos imaginar a computação em nuvens como uma enorme rede de nós que precisa ser escalável. Para os usuários a escalabilidade deve ser transparente, não necessitando eles saberem onde estão armazenados os dados e de que forma eles serão acessados. A escalabilidade pode ser dividida em horizontal e vertical.

ii) A escalabilidade horizontal da nuvem se define como a adição de uma nova instância para distribuir o processamento a ser realizado na nuvem, em contrapartida, a escalabilidade vertical se define no incremento de poder de processamento de uma mesma instância, não sendo necessário a distribuição do processamento em outros servidores ou nuvens.

iii) A extensibilidade no projeto é de extrema importância, ela pode ser demonstrada com a capacidade da implementação de assimilar novos e diferentes tipos de sensores, sendo eles simulados ou físicos. A extensibilidade com inclusões de sensores reais esta intimamente ligada à capacidade do *hardware* escolhido. Isto é, a capacidade de estender a utilização de novos componentes vai estar relacionada com a capacidade de conexões possíveis que o microcontrolador escolhido é capaz de suportar.

7.2. CENÁRIO DE AVALIAÇÃO

Para demonstração de utilização da implementação da arquitetura serão utilizadas duas placas de microcontroladores de ESP8266, ambas as placas são idênticas, adquiridas do mesmo fornecedor e com o mesmo modelo, LoLin NodeMCU Versão 3.

Com o objetivo de identificação e separação de responsabilidade para avaliar o funcionamento correto da implementação de referência criada, as placas foram denominadas com identificações diferentes.

A primeira placa apelidada de Cain, possui código base de implementado anteriormente com a diferença de possuir somente sensores simulados com dados gerados randomicamente com intuito de comprovar a escalabilidade do modelo proposto.

Já segunda placa, apelidada de Abel, possui o mesmo código base demonstrado anteriormente, porém possui somente sensores reais que podem ser utilizados nas diferentes placas de microcontroladores com o intuito de comprovar o requisito de interoperabilidade proposto, como também a extensibilidade, utilizando mais de um tipo de componente sensor na mesma placa de microcontrolador.

Em Cain foi implementado os seguintes simuladores de coleta de dados do ambiente:

- Umidade;
- Temperatura;
- Pressão barométrica;
- Detector de água;
- Umidade do solo;
- Vibração;
- Concentração de gases;

Estes simuladores possuem a responsabilidade de reproduzir um comportamento comum a sensores reais, gerando dados criados aleatoriamente e publicando na plataforma na nuvem responsável por receber a comunicação via protocolo MQTT.
Em Abel foi implementado os sensores reais para avaliar a coleta de dados diretamente do ambiente, são eles:

- Umidade do solo;
- Temperatura;
- Pressão do ar.

7.3. DEMONSTRAÇÃO DOS REQUISITOS NÃO FUNCIONAIS

Com base no cenário apresentado, a avaliação dos requisitos não funcionais foi realizada.

7.3.1. INTEROPERABILIDADE

Na Figura 10 pode-se verificar como está a esquematização dos componentes em Abel. É possível notar como é utilizado o sensor para coleta de umidade do solo.

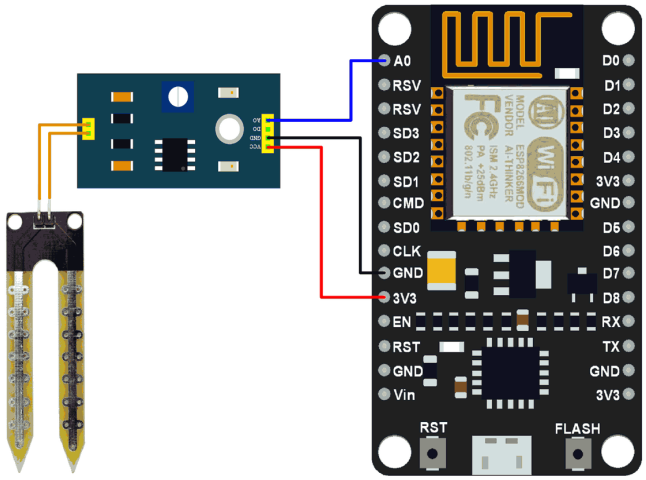

Figura 10: Esquema de utilização do sensor de umidade do solo. (Fonte: www.electronicwings.com, acessado em set/2019)

Após a coleta dos dados, os mesmos são transferidos para a plataforma de *broker*, *Cloud* MQTT, como pode-se verificar abaixo na Figura 11. Pode-se notar quais são os tópicos sendo utilizados e qual *Quality of Service*(QoS) estão configurados, no caso todos os tópicos estão utilizando QoS 0 o que indica que não estão configurados para garantir envio / recebimento de dados, é possível comparar com as características de *Transmission Control Protocol* (TCP) e *User Datagram Protocol*(UDP), esta visão só é possível com a utilização do modo *debugger* no *broker Cloud* MQTT disponível na web.

```
2019-11-10 13:46:27:    Cain/FHumity (QoS 0)
2019-11-10 13:46:27:    Cain/FTemperature (QoS 0)
2019-11-10 13:46:27:    Cain/FAirPressure (QoS 0)
2019-11-10 13:46:27:    Cain/FGasConcetration (QoS 0)
2019-11-10 13:46:27:    Cain/FSoilHumity (QoS 0)
2019-11-10 13:46:27:    Cain/FWater (QoS 0)
2019-11-10 13:46:27: Sending SUBACK to ========= HEROKU _ NODE ============
```

Figura 11: Saída do log do CloudMQTT (Fonte: Autor, 2019)

Com a chegada dos dados no *broker* do *Cloud* MQTT, torna-se possível a coleta desses dados pelo *backend,* construído em NodeJS e armazenamento no banco de dados utilizando Mongo, disponível na nuvem pela plataforma MLab. Com isso, esses dados podem ser consultados posteriormente para estudos dos históricos das coletas.

O MLab é uma plataforma gratuita até determinada quantidade de transações e armazenamento. Para demonstração e provas de conceito como no caso deste artigo, o pacote oferecido é adequado para a maioria dos testes, caso haja necessidade de escalar a solução, a plataforma é capaz de executar escalabilidade de forma vertical e fornecer um servidor dedicado unicamente ao processo, porém, como esperado para este caso, está capacidade já não se enquadra no pacote gratuito e isso pode desqualificar a solução como uma implementação de baixo custo.

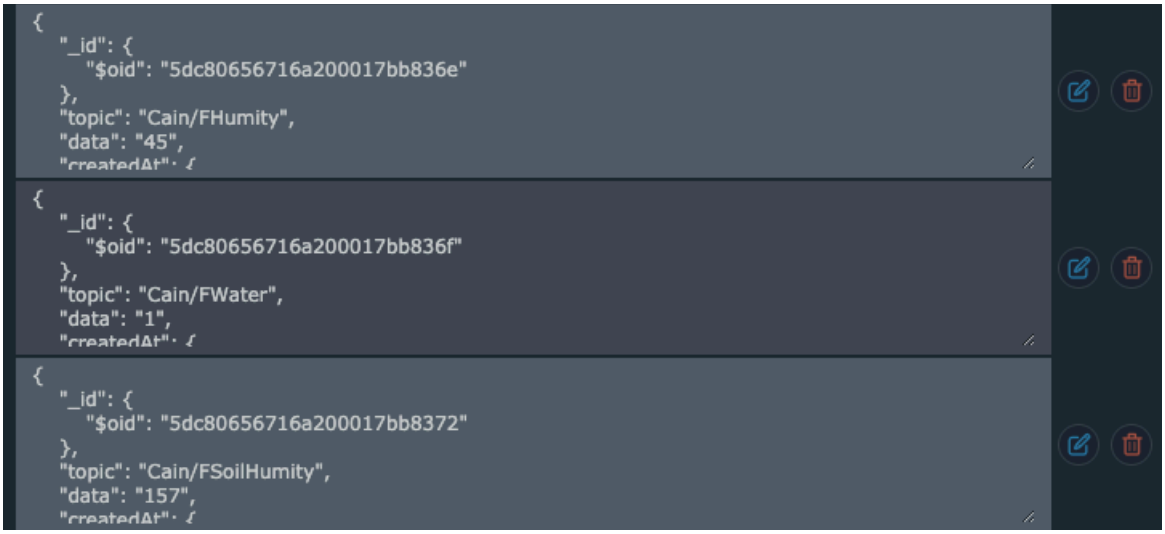

Figura 12: Dados dos sensores registrados no MongoDB(Fonte: Autor, 2019)

Com estes dados armazenados em uma plataforma comum, é possível retorná-los de varias formas para o usuário final.

Outro ponto característico da interoperabilidade da implementação apresentada é a utilização do formato *JavaScript Object Notation*(JSON) para expor o dados coletados pelos microcontroladores na rede, este formato permite que os dados sejam serializados de maneira simplificada facilitando a utilização de qualquer pessoa interessada apenas utilizando protocolo HTTP como citado por FONSECA (2007) "[...] o JSON foi desenhado com o objetivo de ser simples, portável, textual[...]"

É possível consultar dados neste formato em qualquer navegador disponível, na Figura 13 segue um exemplo de uma consulta dos valores gerados por Abel, neste caso, com o sensor de umidade no solo. Consegue-se notar a estrutura gerada munida por:

- _id: identificação do objeto registrado na base criado automaticamente;
- *topic*: tópico onde os dados estão sendo transmitidos pelos sensores;
- *data*: valor captado pelos sensores;
- *createdAt*: data e hora da criação dos dados;
- _v: *flag* de modificação dos dados.

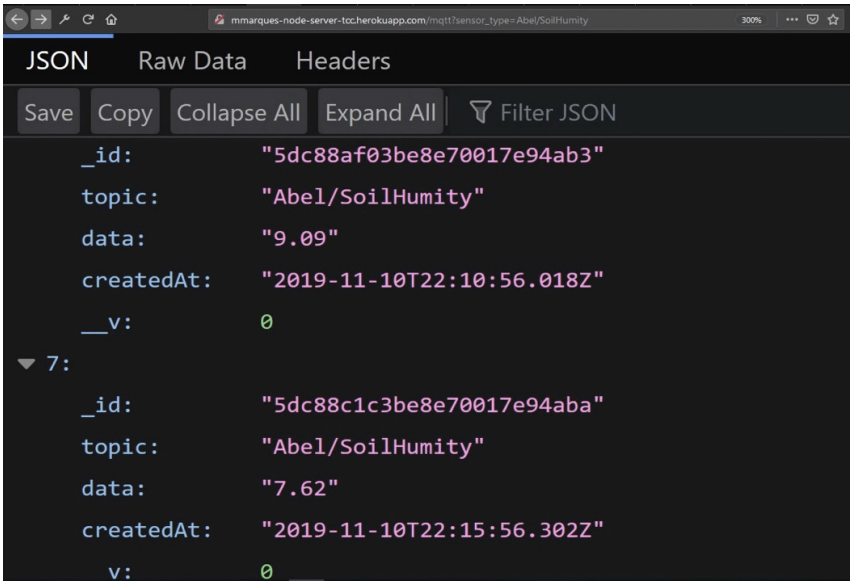

Figura 13 Representação de arquivo JSON - retorno de Abel.(Fonte: Autor, 2019)

Todos os microcontroladores estão ligados a um endereço base disponível na *web*1, isso significa que para consultar sobre diferentes sensores é necessário trocar apenas o caminho do atributo *sensor_type* no endereço. Na tabela 1 é demonstrado os sensores providos por Cain e Abel:

Tabela 1. Nomenclatura de acessos aos dados dos sensores nos microcontroladores.

Tipo	Microcontrolador	Denominação
Temperatura	Cain	/FTemperature
Umidade	Cain	/FHumity
Concentração de gás	Cain	/FGasConcetration
Umidade do solo	Cain	/FSoilHumity
Detector de água	Cain	/FWater
Umidade do solo	Abel	/SoilHumity
Temperatura	Abel	/Temperature
Umidade do Ar	Abel	/AirHumity

Ainda no cenário de avaliação da interoperabilidade, foi desenvolvido uma aplicação utilizando a linguagem Javascript com complemento da biblioteca React Native onde é possível desenvolver aplicações para as plataformas mais populares do mercado atual. Esta aplicação tem a utilidade simples de demonstrar como é possível utilizar os dados provenientes de diferentes fontes.

Como definido anteriormente dois microcontroladores foram selecionados para utilização neste artigo, Cain sensor ESP8266 com finalidade de prover dados apenas de sensores simulados e Abel sensor de mesmo modelo com objetivo de fornecer dados de

1 https://mmarques-node-server-tcc.herokuapp.com/mqtt?sensor_type=

sensores reais. Na Figura 14 é possível verificar a exibição dos dados provenientes de Cain do simulador de temperatura.

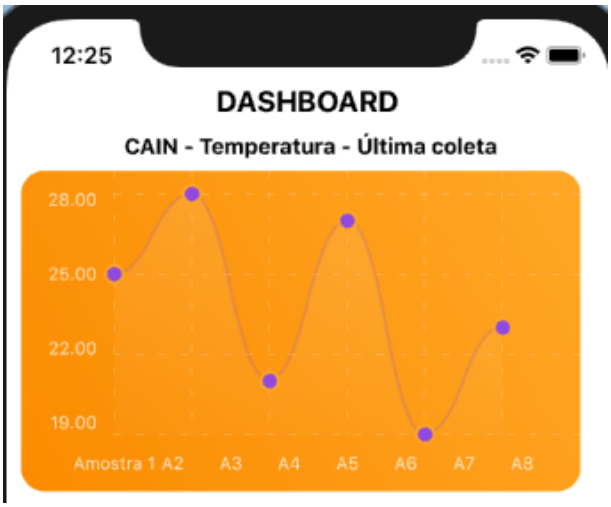

Figura 14: Tela do Dashboard mobile criado em React Native(Fonte: Autor, 2019)

O sensor físico de umidade do solo em Abel foi escolhido para a demonstração de obtenção dos dados de uma fonte diferente. Um técnico qualificado pode induzir os dados rapidamente quando os mesmos são exibidos em gráficos, na representação abaixo os dados indicam um solo que estava completamente seco até a coleta da amostra 4 (A4), onde foi detectado que o solo estava úmido e logo em seguida o mesmo iniciou o processo de secagem com o passar do tempo nas amostras A5, A6, A7 e A8.

Um caso diferente foi abordado na demonstração do consumo de dados através de um sistema web. Para tal, foi construído uma página simples para expor os dados obtidos pela *Application Programming Interface* (*API*), criada em gráficos de fácil leitura (Figura 15), ela também obtém informações de ambos os sensores, mas nesse caso, ambos fazendo referência aos sensores de umidade do solo e um gráfico complementar demonstrado os níveis de detecção de água.

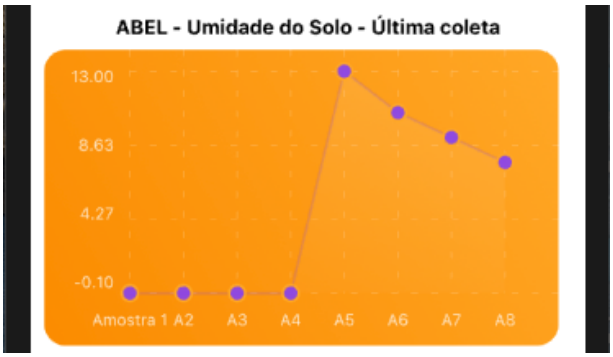

Figura 15: Tela do Dashboard mobile criado em React Native (Fonte: Autor, 2019)

Essas informações visuais objetivam demonstrar como a *API* criada pela implementação é capaz de ser personalizada em diferentes estilos, se adequando a diversos casos, de acordo com as necessidades do usuário final. Na plataforma móvel se apresenta gráficos variáveis, já na plataforma *web*, dados correlacionados e, ainda assim, ambas não utilizam todos os dados disponíveis da base.

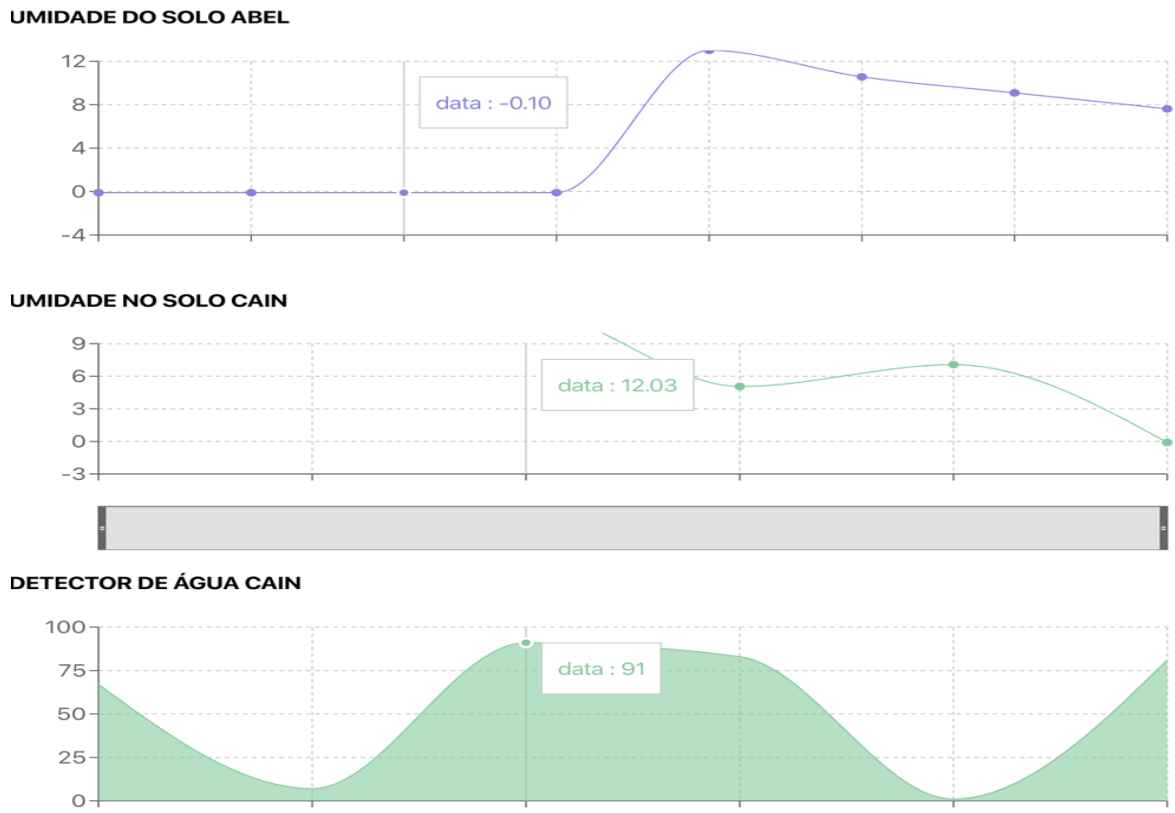

Figura 16: Tela do Dashboard Web criado em React. (Fonte: Autor, 2019)

7.3.2. ESCALABILIDADE

A escalabilidade da implementação da arquitetura está parcialmente ligada a infraestrutura na nuvem na qual está acoplada, no caso atual temos as ligações como demonstrado abaixo na Figura 17.

A infraestrutura disponível na plataforma de MQTT escolhida é gratuita, porém, possui uma limitação de uso para esse pacote, em que é possível criar até 5 conexões simultâneas. Cada nó demonstrado na Figura 17 é considerado uma conexão e é importante notar que o *backend* está diretamente conectado com a mesma plataforma de MQTT, consumindo uma conexão em período integral.

Com isso, para a avaliação da escalabilidade da arquitetura, temos disponíveis 4 conexões para utilizar com os nós de microcontrolador. Este *broker* privado, porém, oferece pacotes de até dez mil conexões simultâneas, fato que evidencia que podemos escalar com este modelo de arquitetura proposto para até 9.999 nós de sensores. Há ainda a possibilidade de migrar para outros *brokers* gratuitos ou privados.

Além dos fatos citado acima vale ressaltar que, o desacoplamento da implementação responsável por cada nó foi desenvolvida pensando justamente na questão da escalabilidade da arquitetura. Ao contrario do que comumente correlacionado, uma implementação desenvolvida para Internet das Coisas não se define naturalmente como escalável, este aspecto deve ser criado e integrado a arquitetura ou solução desenvolvida.

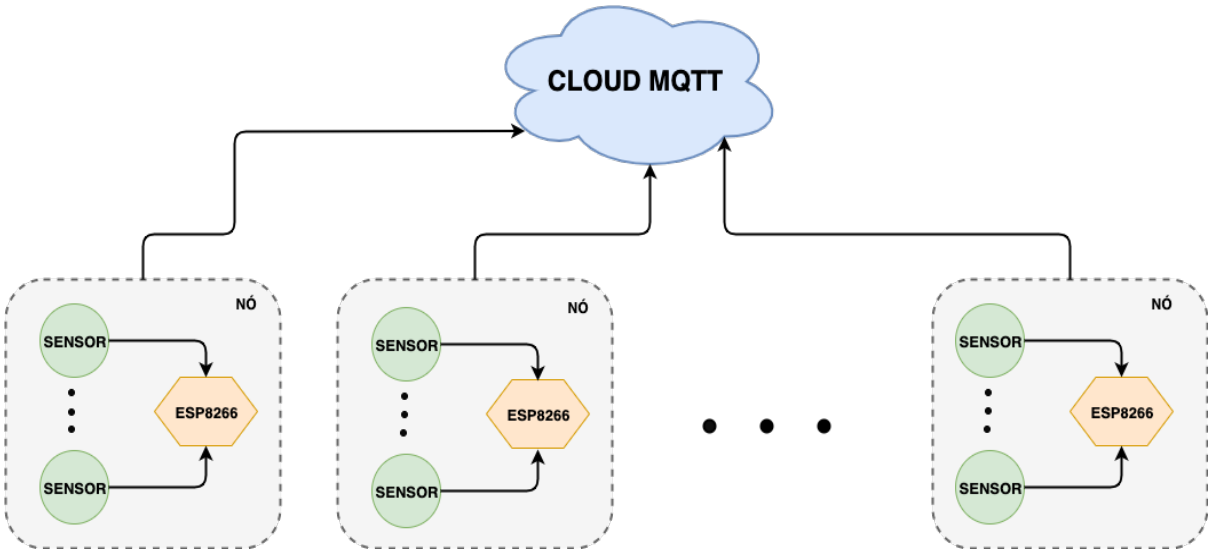

Figura 17: Modelo dos sensores com microcontroladores (Fonte: Autor, 2019)

Vale ressaltar que os *brokers* gratuitos conhecidos não oferecem a opção de criação de tópicos para publicação totalmente privados, isso impacta diretamente na segurança e fidedignidade dos dados transitados, considerando que qualquer pessoa que tenha a nomenclatura de um tópico que não exija autenticação pode publicar nos mesmos. Com esse fato podemos concluir que a modelagem da arquitetura pode reduzir ainda mais o custo e aumentar consideravelmente sua capacidade de escalar suas conexões com novos nós de microcontroladores, porém com o custo de sacrificar a segurança da transação dos dados entre um microcontrolador e um *broker* gratuito exposto na rede.

7.3.3. EXTENSILIDADE

Para comprovar a extensibilidade foi criado um outro sensor em Abel. Ambos sensores trabalharam gerando dados simultaneamente para as estruturas disponíveis na nuvem. Os sensores acoplados para a métrica foram o de temperatura e umidade do ar. Durante a avaliação, o funcionamento correto prosseguiu sem interrupções e ambos sensores funcionaram provendo dados sem interferência mútua, o microcontrolador ESP8266 possui uma limitação de portas para serem utilizadas, apenas uma porta analógica e nove portas digitais estão disponíveis para utilização, isso implica que, se desconsiderarmos a parte de fornecimento e aterramento de energia temos a possibilidade de utilizar dez sensores simultaneamente no mesmo modelo, caso seja considerado que toda responsabilidade energética será provida pelo microcontrolador, a possibilidade de utilização é de três sensores, devido ao fato da placa prover três portas de energia 3.3v e três portas para aterramento.

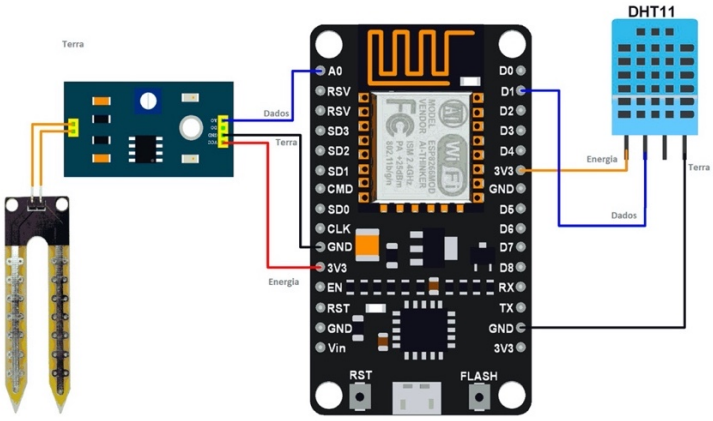

Figura 18: Modelagem de utilização sensor de umidade do solo(Fonte: www.electronicwings.com, acessado em set/2019)

Porém, vale salientar que, as limitações encontradas no modelo de desenvolvimento da arquitetura são de exclusividade do modelo da placa do microcontrolador, há disponível no mercado dezenas de variantes que podem suprir tanto o fator enérgico quanto o de transporte de dados para os sensores.

É importância citar que os modelos de microcontroladores disponíveis são em quase sua totalidade composto de placas para prototipação, não sendo indicados para utilização em produção pelos mais variados motivos, como:

- formato não se adequar ao uso requerido;
- necessidade de atualizações;
- possíveis falhas de utilização;
- fragilidade dos componentes;
- falta de preparação para exposição aos variados tipos de ambiente;

Com isso podemos concluir que a extensibilidade proposta pela arquitetura, se desconsiderarmos os gastos com as estruturas disponíveis na nuvem, é limitada apenas pelo hardware dos microcontroladores e essa limitação se restringe na estrutura física de disponibilidade de portas necessárias para ligação de comunicação dos sensores. Essas questões podem ser resolvidas com o desenvolvimento de hardwares personalizados para o uso fim. A customização tanto da placa microcontroladora, quanto dos sensores a serem utilizados na mesma seria a melhor forma de garantir a melhor performance, a adequação aos requisitos e a durabilidade dos componentes.

Eliminando as limitações observadas pela própria natureza de protótipo da implementação, isto é, dos componentes de prototipagem, os resultados apontam que a arquitetura desenvolvida é capaz de suportar requisitos de interoperabilidade, escalabilidade e extensibilidade. Além disso, a arquitetura proposta se mostrou viável de ser implementada e portanto, possível de ser reproduzida.

8. CONCLUSÃO

Este trabalho foi desenvolvido com o objetivo de prover à comunidade uma arquitetura para apoiar a tomada de decisões no domínio de desastres, incluindo sua prevenção. A arquitetura desenvolvida objetivou atingir requisitos de interoperabilidade, escalabilidade, extensibilidade e potencial baixo custo. Na avaliação da arquitetura, as evidências indicaram que há a possibilidade de fornecer aos interessados dados precisos e coletados diretamente do ambiente, criando-se assim uma maneira de prover mais agilidade e assertividade em tomadas

de decisões das pessoas competentes. Através do *dashboard* criado em React Native pôde-se ler através de gráficos, os dados do ambiente, com isso potencialmente desastres causados por falta de ação antecipada podem ser evitados.

Com a utilização de computação em nuvem, este projeto visa manter o sistema com alta disponibilidade, como também fornecer para o público geral os dados expostos através de uma *API Representational State Transfer* (*REST*) possibilitando que diversas pessoas com diferentes tipos de interesse utilizem os dados para executar ações específicas ao replicar este modelo para coletar dados do ambiente, abrindo a possibilidade de criar modelos personalizados de *dashboard*.

Foi identificado no decorrer da avaliação da arquitetura que a utilização do microcontrolador ESP8266 não é a ideal para o acoplamento de múltiplos sensores da categoria analógica, pois este modelo de placa possui apenas uma porta de mesma tipagem, esta que é ideal para coleta de dados de forma mais precisa, esta mesma placa possui oito portas de uso digital. Porém esta questão não prejudica de qualquer forma a arquitetura desenvolvida, pois como citado anteriormente no decorrer do trabalho, o código implementado pode ser utilizado em toda família de microcontroladores por utilizar nesta camada apenas a linguagem C.

Mostrou-se assim, que é possível utilizar de maneira simples e com baixo custo uma arquitetura que é interoperável, extensível e escalável, que possui fácil replicação e pode ser utilizada nos mais diversos tipos de incidentes, como os desastres causados tanto pela natureza como pelo homem. Há possibilidade também de utilizar este mesmo modelo para realizar monitoramento de ambientes de forma fácil e prática.

Finalizando, é importante salientar que todo código exposto acima tem o destaque explícito neste presente artigo pois, são exatamente os pontos onde seria necessário mudança ou complementação por parte daqueles que estejam interessados em replicar esta arquitetura. Para acessar o restante da implementação da *API* o código foi disponibilizado na plataforma de versionamento GitHub[2] de forma aberta.

9. TRABALHOS FUTUROS

A partir deste trabalho há diversas possibilidades de trabalhos futuros para desenvolvimento, como relatado, toda a implementação utilizou sistemas de computação em nuvem, isso acarreta que para esta arquitetura funcionar de acordo, há necessidade de alta disponibilidade de rede para comunicação entre os nós de microcontroladores com os sensores, como também para a comunicação com os sistemas disponíveis nas nuvens, com isso abre-se a possibilidade de implementar uma forma mais otimizada de comunicação.

Outra vertente interessante se encontra em transformar esta arquitetura em uma rede intercomunicativa de microcontroladores através do mundo, onde através de uma estrutura de dados comum e *templates* de código prontos para os variados tipos de microcontroladores e sensores, qualquer pessoa especialista ou leiga, poderia adquirir um kit de prototipação das placas controladoras com sensores e prover dados das mais diferentes partes do mundo e alimentar um big data de dados livres do ambiente. Isto permite a exploração de uma rede global e colaborativa, em que pesquisadores de todo o mundo possam acompanhar e criar aplicações de impacto para a sociedade.

10. REFERÊNCIA

ALBERTIN, Alberto Luiz; DE MOURA ALBERTIN, Rosa Maria. A internet das coisas irá muito além as coisas. **GV EXECUTIVO**, v. 16, n. 2, p. 12-17, 2017.

[2] https://github.com/mViniciusMarques/TCC_CES_MVMARQUES/

BAPTISTA, Ana Alice. A falar nos entendemos: **a interoperabilidade entre repositórios digitais**. 2010.

COUTINHO, E. et al. Elasticidade em computaçao na nuvem: Uma abordagem sistemática. **XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos (SBRC 2013)-Minicursos**, p. 1-44, 2013.

DA CONCEIÇÃO, Wellington Nogueira Elizeu; DE RESENDE COSTA, Romualdo Monteiro. Análise do Protocolo MQTT para Comunicação IoT através de um Cenário de Comunicação. **Caderno de Estudos em Sistemas de Informação**, v. 5, n. 2, 2019.

FONSECA, Rúben; SIMOES, Alberto. **Alternativas ao xml: Yaml e json**. 2007.

FREITAS, Carlos Machado de; SILVA, Mariano Andrade da; MENEZES, Fernanda Carvalho de. O desastre na barragem de mineração da Samarco: fratura exposta dos limites do Brasil na redução de risco de desastres. **Ciência e Cultura**, v. 68, n. 3, p. 25-30, 2016.

FREITAS, Carlos Machado de et al. Da Samarco em Mariana à Vale em Brumadinho: desastres em barragens de mineração e Saúde Coletiva. **Cadernos de Saúde Pública**, v. 35, p. e00052519, 2019.

GASCÓN, David. **Tecnologia de sensores pode evitar tragédias como Mariana.** O Globo, 20 nov. 2015. Disponível em: < https://oglobo.globo.com/economia/tecnologia-de-sensores-pode-evitar-tragedias-como-mariana-18097054 >. Acesso em: 02 nov. 2019.

HALSE, Shane Errol et al. Bad Weather Coming: Linking social media and weather sensor data. In: **ISCRAM**. 2018.

KOBIYAMA, MASATO et al. Papel da comunidade e da universidade no gerenciamento de desastres naturais. **Simpósio Brasileiro de Desastres Naturais**, v. 1, p. 834-846, 2004.

MINI, Raquel AF; NATH, Badri; LOUREIRO, Antonio AF. Prediction-based approaches to construct the energy map for wireless sensor networks. In: **21o Simpósio Brasileiro de Redes de Computadores**. 2003.

MORDECAI, Yaniv; KANTSEPOLSKY, Boris. Intelligent Utilization of Dashboards in Emergency Management. In: **ISCRAM**. 2018.

PARIZZI, Maria Geovana. Desastres naturais e induzidos e o risco urbano. **Revista Geonomos**, v. 22, n. 1, 2014.

PAULUS, David; MEESTERS, Kenny; VAN DE WALLE, Bartel. Turning data into action: supporting humanitarian field workers with open data. In: **ISCRAM**. 2018.

RUSCHEL, Henrique; ZANOTTO, Mariana Susan; MOTA, WC da. **Computação em nuvem**. **Curitiba, abr**, p. 1-3, 2010.

SÁ, Thiago Teixeira; SOARES, José Marques; GOMES, Danielo G. Cloudreports: Uma ferramenta gráfica para a simulação de ambientes computacionais em nuvem baseada no framework cloudsim. In: **IX Workshop em Clouds e Aplicações-WCGA**. sn, 2011.

SEAL, Victor et al. A simple flood forecasting scheme using wireless sensor networks. **arXiv preprint arXiv:1203.2511**, 2012.

SETI PATRICIO, Thiago et al. INTERNET DAS COISAS (IOT): AS CONSEQUÊNCIAS DA COMPUTAÇÃO UBÍQUA NA SOCIEDADE. In: **Colloquium Humanarum**. 2018.

NEVES, Aline Faustino Veiga. Metadados: **análise da produção científica brasileira à luz das funcionalidades de busca e recuperação.** 2016.