DOI: https://doi.org/10.29367/issn.1809-3957.15.2020.174.54

Volume 16 – Nº 174 – Junho / 2020
XLII International Sodebras Congress
28 a 30 de maio de 2020 – Web Conference.

UMA APLICAÇÃO IOT COM PROTOCOLO MQTT PERSISTINDO EM MONGODB

AN IOT APPLICATION WITH MQTT PROTOCOL PERSISTING IN MONGODB

ALBUQUERQUE, Jesus; CHEN, Daniel; BASTOS, Moisés; SOARES, André; VALENZUELA, Walter
UNIVERSIDADE DO ESTADO DO AMAZONAS
jda.eng17@uea.edu.br;dac.eng17@uea.edu.br;
mpbastos@uea.edu.br;alsoares@uea.edu.br;wvalenzuela@uea.edu.br

Resumo – O protocolo de mensagem MQTT é uma ótima maneira de realizar uma interconexão de dados entre sistemas, utilizando-o junto com um banco de dados e uma maneira de produzir informações, é possível criar facilmente uma aplicação de Internet das Coisas (IoT). Esse trabalho realizou testes e simulações qualitativas para certificar se esse conjunto de protocolos e sistemas são uma boa opção para desenvolvimento de aplicações IoT. Como resultado, foi possível ratificar seu desempenho, concluindo que o uso do protocolo MQTT junto com o servidor de dados MongoDB constitui-se em um modo rápido e simples de desenvolver uma aplicação para emprego em IoT, parte da indústria 4.0. Além disso, concluiu-se que o uso de um banco de dados não estruturado como o MongoDB é mais vantajoso em um cenário de informações incertas, como o IoT e seus microcontroladores, do que o banco de dados estruturado.
Palavras-chave: MQTT. MongoDB. Internet of Things.

Abstract – The MQTT message protocol is a great way to perform data interconnection between systems, using it together with a database and a way to produce information, it is possible to easily create an easy Internet of Things (IoT) application. This article performs qualitative tests and simulations to see if this set of protocols and systems are a good option for IoT application development. As a result, it was possible to confirm its performance, concluding that MQTT together with MongoDB is a quick and simple way to develop an application for use in an IoT environment, part of industry 4.0. Furthermore, it was concluded that the use of an unstructured database as MongoDB is more advantageous in an uncertain information scenario, such as IoT and its micro controllers, than the structured database.
Keywords: MQTT. MongoDB. Internet of Things.

I. INTRODUÇÃO

Desde o surgimento da Internet em 1969, tendo seu uso comercial autorizado em 1987 até meados dos anos 90, o seu crescimento foi dado de maneira estática. Com a evolução tecnológica no final do primeiro milênio, a Internet popularizou-se de forma exponencial (FAROOQ; et. al., 2015). Até então, como explica CARVALHO (2017), existia no mundo uma internet sem "coisas", não estávamos tão conectados, e certamente não éramos.

O conceito de Internet das Coisas (IoT) foi proposto pelo pesquisador do MIT Kevin Ashton, em 1999 (ASHTON, 2009). Como já mencionado, havia no mundo uma internet sem "coisas", ou seja, não havia interruptores de luz, sensores conectados à internet ou câmeras Wi-Fi. Como os seres humanos e o ambiente que os cercam são físicos, toda a sociedade é baseada em coisas. E foram essas coisas, no sentido abstrato propositalmente, segundo CAINÃ, OLIVEIRA e MOTÁ (2018), designando qualquer entidade física que pode interagir pela Internet (MEDEIROS; et. al., 2018), que foram capazes de alimentar os computadores com os mais diversos tipos de informação a serem utilizadas posteriormente.

Para que essas informações sejam transmitidas entres as mais variadas entidades, protocolos foram desenvolvidos para atender a demanda de IoT. Um dos mais utilizados é o MQTT devido às suas características de fácil implementação, conexões entre os clientes assíncronas e por atender a dispositivos de baixa potência como sensores e microcontroladores, estes essenciais dentro da área de IoT, conforme observado por NEPOMUCENO (2020).

Há ainda neste contexto de manuseio de informações por computadores uma terceira preocupação: como são gerados muitos dados para processamento (SOLAGNA; LAZZARETTI, 2016), é necessário um local para armazenar adequadamente todo esse conjunto de registros e, posteriormente, acessá-los para tratá-los, recuperando-os.

As operações fundamentais de armazenamento e recuperação de informações são feitas em um Banco de Dados. Em IoT, por se tratar de ecossistemas digitais com diferentes entidades em que seus dados não necessariamente possuem relacionamentos e havendo a preocupação de que alguns registros são mais importantes que outros, o emprego de Banco de Dados Não Relacionais é mais vantajoso. Um deles é o MongoDB (FERREIRA, 2018), presente nesse trabalho.

O MongoDB é um software de Sistema de Gerenciamento de Banco de Dados (SGBD) criado em 2009 pela empresa de mesmo nome (MONGODB, 2020). O software utiliza conceitos de Banco de Dados Relacionais, como índices e consultas, e Não Relacionais como orientação a documentos livres e hierarquias complexas.

Neste trabalho, será realizada uma aplicação prática de um ambiente de Internet das Coisas, utilizando entidades que se comunicarão entre si pelo protocolo de mensagens MQTT. As informações produzidas por elas serão, por fim, armazenadas no Banco de Dados MongoDB adequadamente.

II. REFERÊNCIA TEÓRICA

2.1 - MQTT

Message Queuing Telemetry Transport é um protocolo de mensagens M2M (Machine-to-Machine) baseado em TCP/IP no qual tópicos são gerenciados por um *broker*,

clientes publicadores enviam informações para eles e clientes inscritos as recebem. O MQTT se preocupa com a integridade de informações trocadas a partir da funcionalidade de Qualidade de Serviço (QoS). Também suporta encriptação fornecida pelo protocolo de segurança TLS nas comunicações sobre redes de computadores (KONESKI, 2018). Toda essa implementação é desenvolvida para atender baixas demandas de energia como dispositivos de pequeno porte tais como sensores e microcontroladores (GARCIA, P.; KLEINSCHMIDT, 2017), sem comprometer o desempenho desejado das aplicações que utilizam o MQTT como protocolo, analisado por TORRES, ROCHA e DE SOUZA (2016).

2.2 - MongoDB

O MongoDB é um Banco de Dados NoSQL, ou Não Relacional, de código aberto, orientado a documentos. Os documentos são baseados em JSON (JavaScript Object Notation) em modo binário. Essa característica possibilita armazenar os dados com hierarquias complexas e serem indexáveis, facilitando as operações de busca. Além disso, os dados dos documentos são desestruturados, o que seria um problema para os Bancos de Dados Relacionais, mas não para o MongoDB. Como explicam POLITOWSKI e MARAN (2014), os documentos podem ser armazenados em coleções (*collections*), onde serão efetuadas operações de busca (*queries*) e indexação (*indexing*). *Queries* são expressas na sintaxe JSON e enviadas ao MongoDB como objetos BSON pelo driver de conexão ao banco (BSON, 2020).

Com isso, o banco foi projetado para suportar a carga de informações de Big Data e, portanto, possui uma relação bem íntima com Internet das Coisas. Aplicações que trabalhem com JSON nativo como Node.js e outras tecnologias baseadas em JavaScript como softwares de celulares e ambientes Web são altamente recomendadas para utilizar MongoDB.

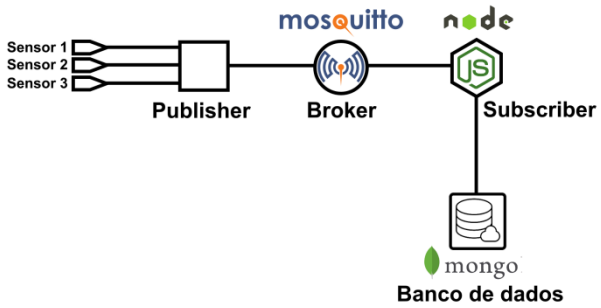
2.3 - IoT (Internet of Things).

Internet das Coisas designa os ecossistemas digitais compostos por máquinas, dispositivos e seres humanos que compartilham informações entre si e atuam para alcançarem objetivos individuais ou que interessem um grupo (MANO; et. al., 2016). Outros termos como Cidades Inteligentes (Smart Cities) e Sistemas Ciber-Físicos trabalham e até compartilham definições com IoT visto que todos estes possuem a filosofia de que há coisas que se comunicam utilizando o protocolo de Internet.

III. METODOLOGIA

Para o desenvolvimento deste estudo foi definida uma simulação de um sistema IoT de Monitoramento da temperatura ambiente de três cômodos de uma residência. Nesse sistema, os sensores enviam os dados via MQTT que são persistidos no MongoDB que está hospedado em um servidor em nuvem (cloud Server), caracterizando assim uma aplicação IoT. Utiliza-se um cenário simulado pois os sensores não foram implementados fisicamente. Com isso este sistema utiliza dados arbitrários/aleatórios simulando uma medição remota. O esquema do sistema é ilustrado pela figura 1 abaixo.

Figura 1 - *Diagrama do cenário de testes MQTT*



Fonte: Os autores

3.1 - Sistema MQTT

O sistema MQTT foi desenvolvido com base nos três componentes principais do protocolo: o assinante, o publicador e o *broker* (AL-FUQAHA, 2015). A primeira parte do sistema consiste em um cliente MQTT *publisher* denominado "Cômodos" e conectado ao *broker* central. O cliente "Cômodos", por sua vez, está recebendo informações de três sensores de temperatura ambiente de cômodos espalhados por uma residência. O *publisher* foi implementado utilizando o *broker* Mosquitto, desenvolvido pela Oracle. As informações dos cômodos estão organizadas em formato compacto JSON, conforme a seguir:

{
Sensor_Temperatura_1: leitura,
Sensor_Temperatura_2: leitura,
Sensor_Temperatura_3: leitura
}

Os sensores de temperatura servem para simular os três cômodos da residência. No cenário do estudo proposto, também será simulada a ocorrência de algumas falhas na leitura dos sensores, condição essa melhor explicada posteriormente.

Além disso, o cliente "Cômodos" está inscrito no tópico "**residencia1/comodos/temperatura"** do *broker*. A utilização dos tópicos presentes no *broker* é bastante intuitiva. Neste caso, em umas das residências, no caso a residência 1, no tópico "**cômodos"**, as temperaturas estão sendo enviadas para o *broker*. Essa relação do *publisher* e o *broker* está ilustrada na Figura 1.

O *broker*, também implementado utilizando o Mosquito através de um Terminal do Windows, funciona como o intermediador que recebe as informações de diversos clientes *publishers*, as organiza e transmite para todos os clientes *subscribers* que estão utilizando os mesmos tópicos. Outro ponto importante do *broker* é que ele possui a funcionalidade de Qualidade de Serviço (QoS) utilizada para assegurar a integridade dos dados que estão sendo enviados e recebidos. Neste caso de emprego em IoT, como existe um grande fluxo de informações, é comum algum atributo dentro dos pacotes trocados pelas entidades perder-se devido ao tráfego de informações. Por isso o QoS (Quality do Serviço) de todas as entidades do MQTT foram configurados para 0, sendo essa a configuração mais baixa para a segurança da integridade dos dados.

Conforme a figura 1, a última entidade do Sistema MQTT presente no cenário IoT é o outro cliente

ISSN 1809-3957

Intermediário MQTT *subscriber* também conectado no *broker*. Ele está inscrito utilizando tópico do *publisher* "**residencia1/comodos/#"**. O símbolo '#' significa que ele receberá as informações de todos os sensores da casa, ou seja, de todos os subtópicos de "**comodos/"**.

3.2 - Cliente Intermediário

Depois de receber os dados, o cliente Intermediário persistirá no MongoDB. Como ele ao mesmo tempo recebe informações do *broker* e envia para o banco de dados, o mesmo foi desenvolvido utilizando-se o *framework* NodeJs.

3.2.1 - Recebendo as informações

Observa-se o código-fonte em JavaScript responsável pela comunicação do cliente Intermediário com o *broker* do sistema na figura 2. Inicialmente é feito o requerimento dos módulos que implementam as funcionalidades necessárias para se trabalhar com MQTT e MongoDB. Em seguida, a conexão com o banco é realizada através do endereço disponibilizado pela MongoDB no Website. Por fim, a conexão com o broker também é estabelecida a partir do host e porta de acesso. No método 'subscribe' do cliente instanciado, o tópico "**residencia1/comodos/#"** e o QoS do serviço são configurados.

Figura 2 - *Código em NodeJS da aplicação cliente MQTT*

```
const mqtt = require('mqtt');  //MQTT
var Comodo = require('../models/ComodosModel_Artigo.js')
var Mongoose = require('mongoose')

Mongoose.connect(
'mongodb+srv://usuario1:asd123456@coleta-4xqe0.mongodb.net/sistic?retryW
);

Mongoose.connection.on('connected', () => console.log('Connected'));

var config = {host: 'localhost', port: 1884}

var cliente = mqtt.connect(config);
cliente.subscribe({"residencia1/comodos/#": {qos: 0}});
```

Fonte: Os autores

3.2.2 - Armazenando as informações

Nessa etapa, apresentada na figura 3, o cliente inscrito aguardará as informações vindas do *broker*. Ao serem recebidas, as mensagens serão convertidas primeiramente para formato *String* e depois para JSON. Após isso, a lógica para armazenamento no MongoDB foi implementada juntamente com algumas mensagens no console. Como o QoS foi configurado para 0, pode ser que algum sensor não leia corretamente a temperatura ou no pior dos cenários não leia nada. Neste caso, como as informações do MongoDB são desestruturadas, não há problema em armazená-las nos documentos.

Figura 3 - *Código em NodeJS da aplicação cliente MQTT*

```
cliente.on('message', function (topic, message){
    sensores = JSON.parse(message.toString())
    console.log("Informações recebidas: ")
    console.log(sensores)
    Comodo.create(sensores).then((listasensores)=>{
        if(listasensores.n == 0){
            console.log("Ocorreu um erro em encontrar sensores.")
        }else{
            console.log("Informações enviadas ao mongo.")
        }
    }).catch((erro) =>{
        console.log("Ocorreu um erro")
    })
})
```

Fonte: Os autores

Para a implementação do banco de dados, foi utilizada uma instância do MongoDB através uma conta criada gratuitamente na sua plataforma Web Mongo Atlas.

3.4 - Inicialização das entidades

Primeiramente, foi criada uma simulação que reproduziu o diagrama da Figura 1. Portanto, dentro do Prompt de Comando, o Mosquitto Broker foi iniciado utilizando o comando **mosquitto -v -p 1884** (-v inicia o *broker* com as configurações padrões; -p escolhe a porta do *broker*) conforme figura 4.

Figura 4 - Inicialização do *Broker* MQTT

```
C:\Users\Sword>mosquitto -p 1884 -v
1582908664: mosquitto version 1.6.8 starting
1582908664: Using default config.
1582908664: Opening ipv6 listen socket on port 1884.
1582908664: Opening ipv4 listen socket on port 1884.
```

Fonte: Os autores

Em seguida, na figura 5, a aplicação NodeJS criada serve de *subscriber* para o ambiente de simulação. Para se inicializar, foi utilizado nodemon, uma biblioteca **npm do NodeJS**. Não entraremos em muitos detalhes, mas essa biblioteca facilita a simulação, pois permite iniciar o sistema sempre que este é salvo.

Figura 5 - Inicialização da aplicação NodeJS

```
C:\Users\Sword\Documents\ProgramasApp\sistic_backe
nd\app\controllers>nodemon ClienteMQTT_Artigo.js
[nodemon] 2.0.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ClienteMQTT_Artigo.js`
Connected
```

Fonte: Os autores

Assim que esses dois elementos supracitados foram inicializados, a simulação pode prosseguir com o envio de mensagens. Para enviar as mensagens, o MQTT junto com uma formatação em JSON foi utilizado: **mosquitto_pub -h localhost -p 1884 -t residencia1/comodos/temperaturas -q 0 -m "{\"Sensor_Temperatura_X\": Y}"**

Onde:

- **Mosquitto_pub** é o comando publicador
- **-h** indica o host a ser mandado, localhost no caso
- **-p** indica a porta de entrada
- **-t** indica o tópico e subtópicos
- **-q** indica o QoS usado
- **-m** indica a mensagem em si, ela foi propositalmente escrita para ser reconhecida como JSON.

A propósito de testes, o conteúdo das três mensagens foi diferente. Isto está representado na figura 6. No primeiro envio de leitura dos sensores, as três informações foram enviadas: 25, 28 e 30 do sensor 1, 2 e 3, respectivamente. Já no segundo, os dois primeiros enviaram informação de temperatura: 24 e 27. No terceiro envio, somente o sensor 1 enviou a informação de 23.

ISSN 1809-3957

Figura 6 - Linhas de código utilizado para fazer uma publicação teste MQTT

```
C:\Users\Sword\Documents\ProgramasApp\sistic_backend
\app\controllers>mosquitto_pub -h localhost -p 1884
-t residencia1/comodos/temperaturas -q 0 -m "{\"Sens
or_Temperatura_1\": 25, \"Sensor_Temperatura_2\": 28
, \"Sensor_temperatura_3\": 30}"

C:\Users\Sword\Documents\ProgramasApp\sistic_backend
\app\controllers>mosquitto_pub -h localhost -p 1884
-t residencia1/comodos/temperaturas -q 0 -m "{\"Sens
or_Temperatura_1\": 24, \"Sensor_Temperatura_2\": 27
}"

C:\Users\Sword\Documents\ProgramasApp\sistic_backend
\app\controllers>mosquitto_pub -h localhost -p 1884
-t residencia1/comodos/temperaturas -q 0 -m "{\"Sens
or_Temperatura_1\": 23}"
```

Fonte: Os autores

IV. RESULTADOS

Após a inicialização do *broker* e da aplicação (figuras 4 e 5, respectivamente) é possível perceber por meio da figura 7 que houve uma alteração no *broker*. Tal alteração é apresentada pelo fato de que a aplicação se inscreveu no *broker*, causando o comportamento mostrado na figura 7, confirmada pelo comando "*New connection from 127.0.01 on port 1884.*".

Figura 7 - Reação do *broker* ao ser inicializado e receber a inscrição

```
C:\Users\Sword>mosquitto -p 1884 -v
1582914543: mosquitto version 1.6.8 starting
1582914543: Using default config.
1582914543: Opening ipv6 listen socket on port 1884.
1582914543: Opening ipv4 listen socket on port 1884.
1582914543: New connection from 127.0.0.1 on port 1884
.
1582914543: New client connected from 127.0.0.1 as mqt
tjs_a072410c (p2, c1, k60).
1582914543: No will message specified.
1582914543: Sending CONNACK to mqttjs_a072410c (0, 0)
1582914543: Received SUBSCRIBE from mqttjs_a072410c
1582914543:     residencia1/comodos/# (QoS 0)
1582914543: mqttjs_a072410c 0 residencia1/comodos/#
1582914543: Sending SUBACK to mqttjs_a072410c
```

Fonte: Os autores

Em seguida, as mensagens da figura 6 foram enviadas, causando certas mudanças no Prompt de comando da aplicação. As mudanças, basicamente, são sinais que as mensagens foram bem-sucedidas. Tudo isso foi ilustrado na figura 8, que apresenta a mensagem "*Informações recebidas*". Em seguida, o sistema informa os dados recebidos e confirma o envio ao MongoDB.

Figura 8 - Resposta da aplicação ao receber a mensagem MQTT

```
C:\Users\Sword\Documents\ProgramasApp\sistic_backend\ap
p\controllers>nodemon ClienteMQTT_Artigo.js
[nodemon] 2.0.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ClienteMQTT_Artigo.js`
Connected
Informações recebidas:
{ Sensor_Temperatura_1: 25,
  Sensor_Temperatura_2: 28,
  Sensor_temperatura_3: 30 }
Informações enviadas ao mongo.
Informações recebidas:
{ Sensor_Temperatura_1: 24, Sensor_Temperatura_2: 27 }
Informações enviadas ao mongo.
Informações recebidas:
{ Sensor_Temperatura_1: 23 }
Informações enviadas ao mongo.
```

Fonte: Os autores

Por fim, entrou-se no site web do Mongo Atlas, onde podemos perceber que as mensagens e aplicações foram bem-sucedidas nos testes: recebendo informações pelo broker, enviando-as até a aplicação (*subscriber*) e guardando-os no banco de dados. É possível visualizar este cenário por meio do banco de dados mongo na figura 9.

Figura 9 - Imagem do banco MONGODB

Fonte: Os autores

V. CONCLUSÃO

Pelos resultados obtidos, observamos que é possível criar aplicações para a indústria 4.0, utilizando poucos recursos. Usando um protocolo de informações como o MQTT e um banco de dados (MongoDB, no caso), neste trabalho conseguimos criar um simples processo de IoT, guardando informações externas para que elas possam ser utilizadas mais tarde.

ISSN 1809-3957

Além disso, através dos resultados, conclui-se que um banco de dados não estruturado é muito útil para IoT, pois se tivesse sido utilizado um banco de dados estruturado como um banco de dados relacional SQL, o sistema não reconheceria as várias nuances de possíveis mensagens que podem ocorrer. Por exemplo, durante a metodologia, foram enviadas três mensagens de conteúdos diferentes e elas foram todas aceitas. Se fosse utilizada linguagem SQL, as tratativas seriam mais complexas. Como outros sistemas irão tratar essas mensagens não está no escopo deste trabalho.

V. REFERÊNCIAS BIBLIOGRÁFICAS

Ala AL-FUQAHA, Senior Member, IEEE, et al. **Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications**. IEEE COMMUNICATION SURVEYS & TUTORIALS, VOL. 17, NO. 4, FOURTH QUARTER 2015

Ashton, Kevin. **That 'Internet of Things' thing**. RFID Journal, 2009. Disponível em <http://www.rfidjournal.com/article/view/4986>. Acesso em 28 jun. 2020.

BSON Webpage. Disponível em <http://bsonspec.org/> Acesso em 20 jan. 2020.

CAINÃ, L.; OLIVEIRA, L.; MOTÁ, S. **Internet das coisas (IOT): um estudo exploratório em agronegócios**. VI Simpósio da Ciência do Agronegócio, 2018.

CARVALHO, M. **ELFA MONITORING SYSTEMS: UM SISTEMA DE MONITORAMENTO DE PESSOAS IDOSAS POR MEIO DE SENSORES, UTILIZANDO INTERNET DAS COISAS**. Universidade do Estado do Rio Grande do Norte. Programa de Pós-Graduação em Ciência da Computação, 2019.

FERREIRA, L. **As diferenças entre SQL e NoSQL: MySQL x MongoDB**. Medium, 2018. Disponível em <https://medium.com/devtranslate/diferencas-entre-sql-e-nosql-51311f9069bd>. Acesso em 20 jan. 2020.

GARCIA, P.; KLEINSCHMIDT, J. **Tecnologias Emergentes de Conectividade na IoT: Estudo de Redes LPWAN**. XXXV SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES E PROCESSAMENTO DE SINAIS, 2017.

KONESKI, E. **Ambiente de comunicação segura de Internet das Coisas com a utilização do MQTT e TLS**. Universidade Federal de Santa Catarina, 2018.

MANO, L.; VOLPANO, T.; FUNES, M.; NETO, J. **Explorando tecnologias de IoT no contexto de Health Smart Home: uma abordagem para detecção de quedas em pessoas idosas**. JADI – Marília – v. 2, 2016.

MEDEIROS, F.; COLPO, I.; SCHNEIDER, V.; CARVALHO, P. **INTERNET OF THINGS: UMA INVESTIGAÇÃO DO CONHECIMENTO CIENTÍFICO EM ARTIGOS ACADÊMICOS NA ÚLTIMA DÉCADA**. Revista Eletrônica de Administração e Turismo (ReAT), 2018.

MONGODB Official Webpage. Disponível em <https://www.mongodb.com/>. Acesso em 20 jan. 2020.

MQTT Official Webpage. Disponível em <http://mqtt.org/faq>. Acessado em 11/09/2019

NEPOMUCENO, C., **UMA ABORDAGEM TEÓRICA E PRÁTICA EM UM PROTOCOLO PARA IOT.** XLI International Sodebras Congress, N° 169, 2020.

POLITOWSKI, C.; MARAN, V. **Comparação de Performance entre PostgreSQL e MongoDB**. Departamento de Ciências Exatas e Engenharias - Universidade Regional ˆ do Noroeste do Estado do Rio Grande do Sul, 2014.

TORRES, A. B. B.; ROCHA, A. R.; DE SOUZA, J. N.

ANÁLISE DE DESEMPENHO DE BROKERS MQTT

EM SISTEMA DE BAIXO CUSTO. In Anais do XXXVI

congresso da sociedade brasileira de computação, 2016.

SOLAGNA, E.; LAZZARETTI A. **UM ESTUDO COMPARATIVO ENTRE O MONGODB E O POSTGRESQL**. Trabalho de Conclusão de Curso (TCC) - Instituto Federal Sul-Rio-Grandense, 2016.

U.Farooq, M., Waseem, M., Mazhar, S., Khairi, A., & Kamal, T. (2015). **A Review on Internet of Things (IoT)**. International Journal of Computer Applications, 113(1), 1–7. https://doi.org/10.5120/19787-1571

VI. AGRADECIMENTOS

Este trabalho conta com o apoio da Samsung Eletrônica da Amazônia Ltda. através da Lei de Informática, AGIN - Agência de Inovação da UEA e FUEA - Fundação Universitas de Estudos Amazônicos.

VII. COPYRIGHT