Ministério da Educação
Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
Programa de Residência em Tecnologia da Informação

# Internet das Coisas: Implementando um sistema de monitoramento do Data Center do TCE/RN

**Davi Ribeiro Cunha**

Natal-RN
Julho de 2019

**Davi Ribeiro Cunha**

# Internet das Coisas: Implementando um sistema de monitoramento do Data Center do TCE/RN

Trabalho de conclusão de curso apresentado ao Programa de Residência em Tecnologia da Informação da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de Especialista em Tecnologia da Informação.

*Núcleo*:
Infraestrutura de TI

Orientador

Prof. Aluizio Ferreira da Rocha Neto

Programa de Residência em Tecnologia da Informação
IMD – Instituto Metrópole Digital
UFRN – Universidade Federal do Rio Grande do Norte

Natal-RN

Julho de 2019

Trabalho de conclusão do curso sob o título *Internet das Coisas: Implementando um sistema de monitoramento do Data Center do TCE/RN* apresentado por Davi Ribeiro Cunha e aceito pelo Programa de Residência em Tecnologia da Informação da Universidade Federal do Rio Grande do Norte, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

---

Aluizio Ferreira da Rocha Neto
Presidente
DIMAp – Departamento de Informática e Matemática Aplicada
UFRN – Universidade Federal do Rio Grande do Norte

---

Marcos Cesar Madruga Alves Pinheiro
Examinador
DIMAp – Departamento de Informática e Matemática Aplicada
UFRN – Universidade Federal do Rio Grande do Norte

---

Vinícius José Miranda Toscano de Brito Filho
Examinador
Diretoria de Informática
Tribunal de Contas do Estado do Rio Grande do Norte - TCE/RN

Natal-RN, 24 de Julho de 2019.

# Agradecimentos

Agradeço ao Tribunal de Contas do Estado do Rio Grande do Norte - TCE/RN, através do seu então presidente, por ter apoiado o programa de residência de TI do IMD.

À Diretoria de Informática, através do seu então Diretor de Informática, pelo apoio aos projetos executados pela equipe de infraestrutura de TI da residência e à equipe de Infraestrutura de TI do TCE/RN, da qual eu faço parte.

Aos residentes, colegas de curso, Daniel Henrique, por ter me ajudado com a parte eletroeletrônica do projeto e Jonas Paiva, por ter me ajudado com a linguagem de programação C++, ambos necessários para a execução desse projeto.

Aos professores do IMD, por terem me guiado e ajudado ao longo do Programa de Residência em Tecnologia da Informação, em especial ao Professor Aluízio, que me ajudou com este trabalho de conclusão do curso.

Por fim, agradeço à minha família, em especial, à minha filha Diva, à minha esposa Pamella e à minha mãe Viviane, que me apoiaram durante toda essa jornada.

*"Quando [a conexão] sem fio for perfeitamente aplicada, a Terra inteira será convertida em um enorme cérebro, o que na verdade é, sendo que todas as coisas são partículas de um conjunto real e rítmico. Seremos capazes de nos comunicar uns com os outros instantaneamente, independentemente da distância."*

Nikola Tesla

# Internet das Coisas: Implementando um sistema de monitoramento do Data Center do TCE/RN

Autor: Davi Ribeiro Cunha
Orientador: Aluizio Ferreira da Rocha Neto

## Resumo

Este trabalho descreve como foi implementado um sistema para monitorar a temperatura, umidade e outras condições da sala do data center do TCE/RN, utilizando Internet das Coisas, com o sistema em um chip ESP32, utilizando a plataforma Arduino, e a ferramenta de monitoramento Zabbix, como uma espécie de DCIM, além dos serviços IFTTT e ThingSpeak, para auxiliar no envio de alertas, e painéis de monitoramento no NOC, utilizando o Grafana para auxiliar a equipe de Infraestrutura de TI da Diretoria de Informática do TCE/RN.

*Palavras-chave*: Monitoramento, Temperatura, Umidade, Data center, Internet das Coisas, Sistema em um chip, ESP32, Arduino, Zabbix, DCIM, IFTTT, ThingSpeak, Alertas, NOC, Grafana, Infraestrutura de TI, TCE/RN.

# Internet das Coisas: Implementando um sistema de monitoramento do Data Center do TCE/RN

Author: Davi Ribeiro Cunha
Supervisor: Aluizio Ferreira da Rocha Neto

## Abstract

*This work describes how a system was implemented to monitor the temperature, humidity and other conditions of the TCE/RN's data center room, using Internet of Things, with ESP32 SoC, using the Arduino platform, and the monitoring tool Zabbix, as a sort of DCIM, in addition to the IFTTT and ThingSpeak services, to assist in the sending of alerts, and monitoring panels in the NOC, using Grafana to assist the IT Infrastructure team of the TCE/RN's* Diretoria de Informática.

*Keywords: Monitoring, Data center, Internet of Things, ESP32, SoC, Arduino, Zabbix, DCIM, IFTTT, ThingSpeak, NOC, Grafana, IT Infrastructure,* TCE/RN.

# Lista de figuras

# Lista de abreviaturas e siglas

TCE/RN - Tribunal de Contas do Estado do Rio Grande do Norte

DIN - Diretoria de Informática

IPMI - Intelligent Platform Management Interface

TI - Tecnologia da Informação

IMD - Instituto Metrópole Digital

IoT - Internet of Things

BNDES - Banco Nacional de Desenvolvimento Econômico e Social

MCTIC - Ministério da Ciência, Tecnologia, Inovações e Comunicações

MCU - Microcontroller

SoC - System-on-a-Chip

NOC - Network Operations Center

DCIM - Data Center Infrastructure Management

IFTTT - If This Then That

API - Application programming interface

JSON - JavaScript Object Notation

LCC – Life Cycle Canvas

# Sumário

# 1 Introdução

O Tribunal de Contas do Estado do Rio Grande do Norte - TCE/RN, de acordo com a Constituição Estadual do Estado do Rio Grande do Norte é responsável pela fiscalização contábil, financeira, orçamentária, operacional e patrimonial das entidades da administração direta e indireta, dotado de autonomia administrativa, orçamentária e financeira, assim como de independência funcional, fiscalizando um total aproximado de 858 (oitocentos e cinquenta e oito) unidades gestoras jurisdicionadas.

Diante da importância do TCE/RN e da imensidão de dados gerados por ele e por essas unidades jurisdicionadas, se fez necessário a criação de um local seguro para guardar e processar esses dados.

Em sua estrutura organizacional, o TCE/RN possui a DIN - Diretoria de Informática, unidade organizacional responsável por manter os aspectos da Tecnologia da Informação e consequentemente responsável pela guarda e processamento desses dados.

Para isso, foi criada uma sala própria na Diretoria de Informática, para acomodação dos equipamentos de rede, sistema de armazenamento e outros equipamentos, com climatização, energia estabilizada e protegida contra falha de energia em uma fonte de alimentação ininterrupta, denominada de sala do data center.

Na ocasião foi implementado o sistema de monitoramento que coletava dados de temperatura dos sensores dos próprios equipamentos, utilizando *Out of band Management*, mais especificamente o padrão IPMI - Intelligent Platform Management Interface.

Assim, apesar do monitoramento já ser feito, existia a dependência do serviço de Internet do TCE/RN e esse monitoramento era feito sem uma solução dedicada (leitura dos sensores nas placas mãe dos equipamentos no datacenter, que estão na parte de dentro desses equipamentos, portanto geram mais calor que o ambiente em si), de forma precária.

Com o passar do tempo, a sala foi crescendo e foram adicionando novos equipamentos, e verificando a necessidade de sensores próprios para coletar dados do ambiente na sala

na qual estes equipamentos estavam inseridos.

Além disso, verificou-se a necessidade de envio de alertas, de forma independente do serviço de Internet existente no TCE/RN.

Diante da ausência de um sistema dedicado para monitoramento do ambiente da sala de equipamentos de TI - Tecnologia da Informação do TCE/RN, e da necessidade de se criar um sistema de monitoramento com sensores, com envio de alertas e de baixo custo, optou-se por desenvolver um projeto de monitoramento do ambiente do data center do TCE/RN utilizando-se tecnologias de Internet das Coisas.

## 1.1 Objetivos

O presente trabalho de conclusão de curso do Programa de Residência em Tecnologia da Informação tem como objetivo relatar o desenvolvimento de um sistema de monitoramento do ambiente do data center do TCE/RN.

Para desenvolvimento dessa solução utilizou-se de tecnologias de Internet das Coisas, com envio de alertas, descrevendo as suas interações com serviços na nuvem e a rede local, através de um dos projetos do Programa de Residência em Tecnologia da Informação do IMD, em parceria com o TCE/RN.

## 1.2 Objetivos Específicos

Para atingir o objetivo geral do trabalho, as seguintes etapas foram seguidas:

- Relatar os requisitos de implantação do sistema de monitoramento de temperatura, umidade e outras condições do ambiente da sala do data center do TCE/RN utilizando Internet das Coisas;
- Apresentar o sistema de alertas;
- Mostrar como foram feitas as interações dos módulos de monitoramento com alguns serviços na nuvem;
- Descrever como foram feitas as interações dos módulos de monitoramento com os serviços existentes na rede local;
- Expor como foram feitos os painéis de monitoramento para serem utilizados na sala de operação e monitoramento;

- Demonstrar os benefícios do novo sistema de monitoramento para o TCE/RN, comparando com soluções existentes no mercado.

## 1.3 Motivação

Este trabalho teve como motivação os recorrentes problemas com o sistema de refrigeração da sala do data center do TCE/RN, ao longo dos últimos anos, que com a falha de algum dos seus componentes ou no fornecimento de energia, ocasionalmente gerava o aumento da temperatura dos equipamentos, presentes na sala do data center, e não eram devidamente relatados, a tempo, em razão da ausência de um sistema dedicado, independente do serviço de Internet do TCE/RN, para o monitoramento dos aspectos ambientais.

# 2 Referencial Teórico

Antes de relatar o desenvolvimento do projeto do sistema de monitoramento do ambiente do do Tribunal de Contas do Estado do Rio Grande do Norte - TCE/RN, é importante explicar alguns conceitos que serão utilizados ao longo deste trabalho.

## 2.1 Internet das Coisas

Internet das Coisas (ou *Internet of Things - IoT,* em inglês) é uma rede de coisas (sensores e atuadores) conectadas à Internet ou rede local, que geram dados de algum objeto/atributo, primordialmente de monitoramento, que podem realizar ações pré-definidas.

A conectividade entre essas coisas pode se dar através de redes WiFi, GSM, LTE, LoRa ou qualquer outra tecnologia de comunicação de fácil operação e de baixo custo.

Nesse sentido, os dados de monitoramento não implicam necessariamente em alguma ação de imediato no ambiente monitorado no momento em que eles são coletados. Os dados podem estar sendo colhidos para que no posteriormente seja feita uma análise e permita tomar algumas decisões com base no histórico de eventos daquele ambiente monitorado.

Para Miguel de Sousa, Internet das Coisas é uma visão em que objetos do cotidiano estão conectados e compartilham dados pela Internet. Acredita-se que terá um enorme impacto em nossas vidas, mudando a maneira como interagimos com as coisas que estão presentes em nossas vidas diárias [6].

O termo *Internet of Things* foi cunhado por Kevin Ashton em 1999, em uma apresentação que ele fez na Procter & Gamble [1].

O sucesso da Internet das Coisas se deu com a difusão da infraestrutura de Internet, uma vez que já existiam os sistemas embarcados e eles podem ser considerados o precursor da Internet das Coisas.

Segundo Mike Kavis, em artigo para a Forbes [12], a Internet das Coisas está dividida

em uma pilha de 4 níveis e 7 camadas, conforme figura 1.

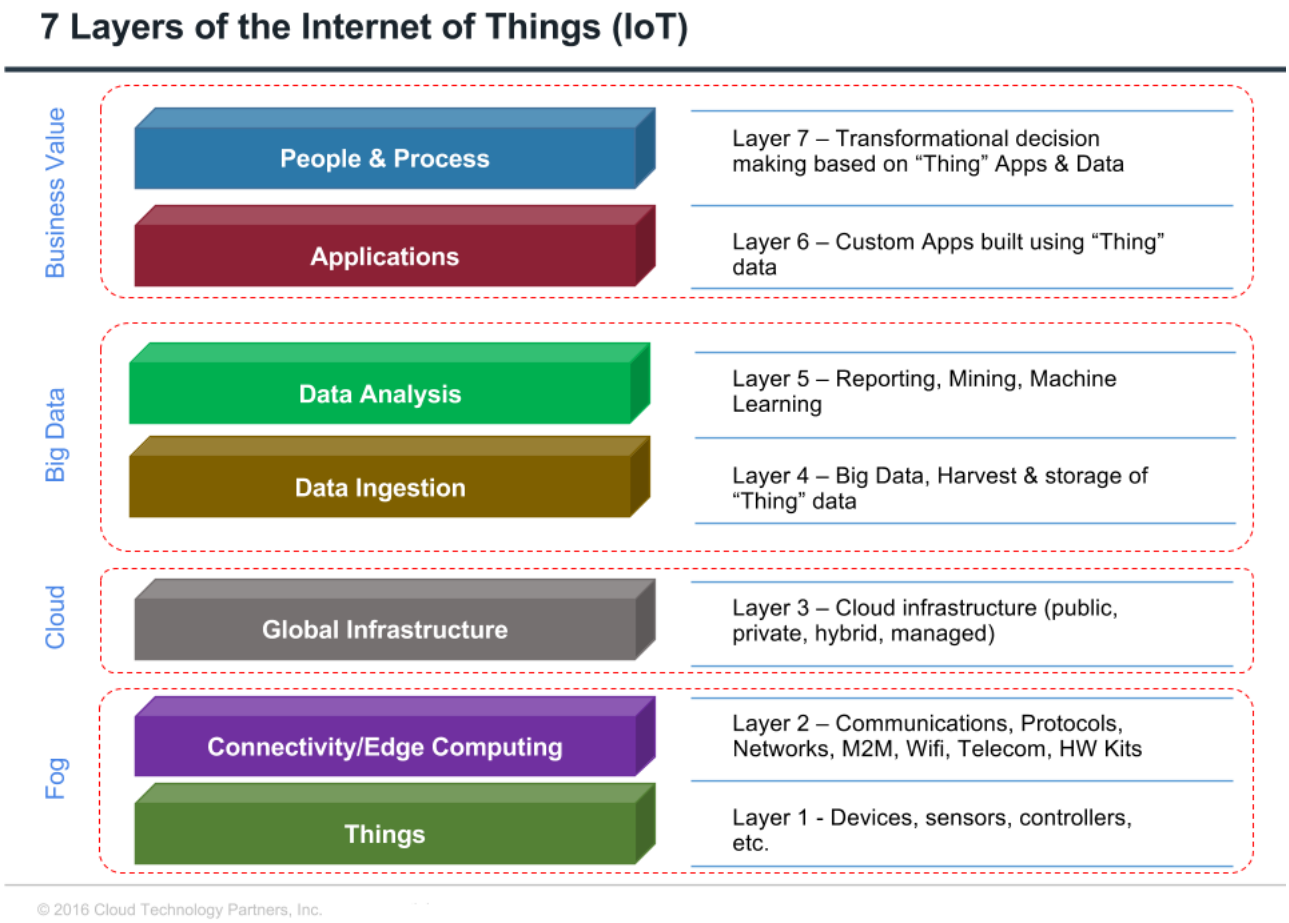



Figura 1: The IoT Ecosystem[12].

Essa divisão em camadas, foi chamada por Mike Kavis [12] de *The IoT Ecosystem* (o ecossistema da Internet das Coisas). Figura traduzida de forma livre:

1. Fog (Névoa):
   (a) Things: Onde estão as coisas (dispositvos, sensores, entre outros).
   (b) Edge Computing/Connectivity: de onde parte a conectividade, protocolos, redes, entre outros.
2. Cloud (Nuvem):
   (a) Global Infrastructure: Para onde os dados são enviados, serviços em nuvem privada, pública, híbrida, entre outras.
3. Big Data:

   (a) Data Ingestion (Ingestão de Dados): Big Data, colher os dados e armazenamento dos dados gerados.

   (b) Data Analysis (Análise de Dados): Relatórios, Mineração dos Dados (BI), aprendizado de máquina, entre outros.

4. Business Value (Valor para o negócio):

   (a) People & Process: Com os dados, as pessoas terão apoio na tomada de decisão.

   (b) Aplications: aplicações customizadas baseadas nos dados.

Internet das Coisas é algo tão importante e atual, que o Banco Nacional de Desenvolvimento Econômico e Social - BNDES , em parceria com o Ministério da Ciência, Tecnologia, Inovações e Comunicações - MCTIC , realizou em 2017, um estudo para o diagnóstico e a proposição de plano de ação estratégico para o Brasil em Internet das Coisas, intitulado "Internet das Coisas: um plano de ação para o Brasil"[3].

Além disso, a Finep - Financiadora de Inovação e Pesquisa possui uma ação de Fomento à Inovação em Internet das Coisas, a Finep IoT [1].

Ademais, o Decreto 9.854 de 25 de junho de 2019, instituiu o Plano Nacional de Internet das Coisas com a finalidade de implementar e desenvolver a Internet das Coisas no País e, com base na livre concorrência e na livre circulação de dados, observadas as diretrizes de segurança da informação e de proteção de dados pessoais[4].

Com a Internet das Coisas, está acontecendo a quarta Revolução Industrial (indústria conectada), a qual vem sendo chamado de Indústria 4.0, que é uma expressão que engloba algumas tecnologias para automação e troca de dados. Dentre os nichos/aplicações da Internet das Coisas temos saúde, agronegócio, cidades, fábricas (indústria de base), veículos e residências.

São exemplos comuns de Internet das Coisas, que já estão presentes nas nossas vidas: cidades inteligentes (*smartcities*: semáforos inteligentes, postes inteligentes, mobilidade urbana), dispositivos vestíveis inteligentes (*wearables*: Apple Watch e a sua integração com o app Saúde, da Apple, *smartbands*, entre outros), termostatos inteligentes (Google Nest), lâmpadas inteligentes (Philips Hue), geladeiras inteligentes (LG e Samsung), entre outros.

[1]Finep IoT: `http://www.finep.gov.br/apoio-e-financiamento-externa/programas-e-linhas/finep-iot`

De forma prática, a Internet das Coisas está presente no presente projeto, no momento em que a temperatura e a umidade da sala de TI são lidos, isto é, capturados através de um microcontrolador com sensores conectado a uma rede WiFi e/ou GSM (telefone/SMS), e enviados para uma base de dados na Internet (nuvem/*cloud*) e/ou rede local (névoa/*fog*).

Assim, com a análise desses dados, é possível tomar algumas decisões, como por exemplo, acrescentar ou trocar o ar condicionado, caso os atuais não estejam atendendo à demanda de refrigeração da sala.

## 2.2 Tecnologias Embarcadas

### 2.2.1 Microcontroladores

Microcontroladores podem ser considerados pequenos computadores que são adicionados a qualquer coisa para dar a ela alguma inteligência. Qualquer aplicação simples de Internet das Coisas precisa de, pelo menos, um microcontrolador (MCU - Microcontroller, em inglês), ou sistema em um chip (SoC - System-on-a-Chip, em inglês), conectado à Internet.

Segundo o Wikipedia, microcontrolador é um pequeno computador em um único circuito integrado, o qual contém um núcleo de processador, memória e periféricos programáveis de entrada e saída. A memória de programação pode ser RAM, NOR flash ou PROM a qual, muitas vezes, é incluída no chip. Os microcontroladores são concebidos para aplicações embarcadas, em contraste com os microprocessadores utilizados em computadores pessoais ou outras aplicações de uso geral [20].

Os microcontroladores contêm um ou mais processadores, juntamente com componentes de entrada e saída programáveis, em um único circuito integrado, por isso são chamados de SoC. Eles são considerados diferentes dos microprocessadores porque são desenhados para funções específicas em que geralmente não demandam alta capacidade de processamento.

### 2.2.2 Arduino

O Arduino é uma plataforma open-source baseada em hardwares e softwares fáceis de usar, na qual você pode dizer à sua placa o que fazer enviando um conjunto de instruções para o microcontrolador na placa, usando a linguagem de programação Arduino (baseada em C++), através da IDE do Arduino, baseado em processamento.

Arduino nasceu como uma ferramenta fácil para prototipagem rápida, destinada a estudantes sem formação em eletrônica e programação, no Instituto de Design de Interação Ivrea, na cidade de Ivrea, na Itália[2].

De acordo com o Wikipedia, o nome Arduino vem de um bar em Ivrea, na Itália, onde alguns dos fundadores do projeto costumavam se encontrar. O bar foi nomeado após Arduin de Ivrea, que foi uma espécie de comandante militar na Marcha de Ivrea e Rei da Itália de 1002 a 1014[18].

O Arduino utiliza o conceito de *Open Source Hardware*, ou hardware de código aberto e compartilha muitos dos princípios e abordagens do software livre e de código aberto.

Os arquivos de design originais do hardware do Arduino são disponibilizados e licenciados sob uma licença Creative Commons Attribution Share-Alike, que permite trabalhos derivados e pessoais, desde que eles credenciem o Arduino e liberem seus designs sob a mesma licença. O software Arduino também é de código aberto. O código-fonte para o ambiente Java é liberado sob a GPL e as bibliotecas do microcontrolador C / C ++ estão sob a LGPL[3].

Por essa razão existem muitas variações das placas Arduino, principalmente criadas por empresas chinesas, e até americanas como SparkFun Electronics and Adafruit Industries.

### 2.2.3 ESP32

É uma plataforma bastante popular, de baixo custo, para desenvolvimento de projetos com Internet das Coisas composta por um microcontrolador da empresa Espressif, com processador Tensilica Xtensa LX6, em uma placa de circuito impresso, sendo considerado um sistema-em-um-chip (SoC), com pinos de alimentação (GPIOs), conector micro USB para alimentação/conexão com um computador[4]. É o sucessor do ESP8266.

O ESP32 pode usar a mesma plataforma de desenvolvimento e bibliotecas do Arduino, outra razão pela sua popularidade.

[2] `https://www.arduino.cc/en/Guide/Introduction`
[3] `https://www.arduino.cc/en/Main/FAQ`
[4] `https://www.espressif.com/en/products/hardware/esp32/overview`

### 2.2.4 Sensores, Atuadores e Módulos

Sensores são componentes eletro-eletrônicos que servem para medir um estímulo no ambiente. Para Kimmo Karvinen e Tero Karvinen[13], ao adicionar sensores é possível expandir a capacidade de um circuito. Para eles, os sensores fornecem uma entrada para informações sobre um ambiente e funcionam de modo muito semelhante aos seus próprios sentidos, são como os nossos olhos, nariz e ouvidos.

Kimmo Karvinen e Tero Karvinen[13] explica que o processo de converter a energia detectada em outra forma é chamado de transdução. Assim, sensores são considerados um tipo de transdutor. Alguns exemplos de componentes do tipo sensor usados neste projeto:

- Sensor de umidade e temperatura DHT11 em um único circuito;
- Sensor de gás MQ-7 Monóxido de Carbono;
- Sensor de gás MQ-2 Inflamável e Fumaça;

Outro tipo de transdutor é o atuador, que nada mais é que uma saída de um circuito, que no lugar de reagir com algo do ambiente ele faz algo acontecer. Alguns exemplos de componentes do tipo atuador usados neste projeto:

- Sinalizador de áudio (*Buzzer*) ativo 5V;
- LED RGB de alto brilho difuso 5mm;

Já os módulos são componentes externos e independentes ao microcontrolador que adicionam alguma função a ele. Exemplo de módulo usado neste projeto:

- Módulo GSM GPRS SIM800L.

## 2.3 *Data Center*

Manoel Veras [15] afirma que *data center* é um conjunto integrado de componentes de alta tecnologia que permite fornecer serviços de infraestrutura de valor agregado, tipicamente processamento e armazenamento de dados, em larga escala, para qualquer tipo de organização.

Para Figueiredo[10], os *data centers* estão mais complexos, mais interdependentes e mais críticos do que nunca. Ter uma ferramenta que automatize a gestão da infraestrutura de TI é uma condição indispensável para garantir que esses ambientes operem com eficiência.

Assim, infraestrutura de TI seria todo o hardware, software, redes, instalações, etc. que são necessárias para desenvolver, testar, entregar, monitorar, controlar ou suportar aplicativos e serviços de TI. O termo infraestrutura de TI inclui toda a tecnologia da informação, exceto as pessoas, os processos e a documentação associados [2].

Segundo Manoel Veras [15], uma forma de categorizar o *data center* é pelo porte, que pode ser empresarial, de médio porte, local, sala de servidores e armário de servidores. Nesse sentido, o *data center* do TCE/RN pode ser enquadrado como sala de servidores.

Os *data centers* empresariais são classificados de diversas formas, dentre as mais comuns, temos a classificação em camadas (*tiers*), feita pelo Uptime Institute [5] e pela norma ANSI/TIE 942 [6], que levam em consideração a disponibilidade, confiabilidade e redundância.

A sala do *data center* do TCE/RN não entraria nessa classificação, por ser apenas uma sala de servidores, mas dessa classificação podemos retirar alguns requisitos, dentre os quais manter a Infraestrutura de TI com maior disponibilidade, de forma confiável e com maior redundância possível.

A sala do *data center* é um ambiente crítico que precisa ser monitorado com precisão, então o monitoramento e controle dos níveis de temperatura, umidade e fumaça torna-se imprescindível para uma boa operação de seus ativos, além de evitar que seus equipamentos sejam danificados e tenham uma vida útil menor.

Sobre a temperatura e umidade dos *data centers*, Paulo Sérgio Marin [14] apresentou as recomendações da NBR 14565:2001 e da ANSI/TIA-942, na figura 2.

### 2.3.1 NOC - Network Operations Center

Centro de Operações de Rede, ou NOC - Network Operations Center , em inglês, como é mais conhecido é o espaço/sala, fora do *data center*, na qual a equipe de operações dá o apoio necessário, gerenciando, monitorando, entre outros, à toda as operações e infraestrutura de TI, incluindo o *data center*, em uma organização ou de forma terceirizada.

[5] Uptime Institute: `https://uptimeinstitute.com/tier-certification`
[6] ANSI/TIE 942: `http://www.tia-942.org/`

| Recomendações | NBR 14565:2011 | ANSI/TIA-942 |
|---|---|---|
| Temperatura Ambiente | 18ºC a 27ºC | 20ºC a 25ºC |
| Ponto de condensação | Entre 5,5º e 15ºC | 25ºC |
| Troca máxima de calor | 5ºC/h | 5ºC/h |
| Umidade relativa do ar | Entre 30% e 60% | Entre 40% e 50% |
| Medições | A cada 3m ao longo da linha central dos corredores frios e no ponto de retorno do ar condicionado | Entre 3 e 6m ao longo central dos corredores frios e nos pontos de entrada do ar condicionado |

Fonte: Marin, 2011, p. 72

Figura 2: Recomendações da NBR 14565:2001 e ANSI/TIA-942.[14]

Para Hwaiyu Geng [11], a sala de operações de rede ou o centro de operações de rede (NOC) oferece suporte às operações de TI. O NOC tem técnicos nesta sala que monitoram as operações da rede e do sistema de TI, normalmente em 24/7.

Manoel Veras [16] diz que o Network Operations Center (NOC) é a célula central do gerenciamento. Por definição, o NOC é um local onde se centraliza a gerência da rede e dos componentes do *data center*. A partir desse centro e de aplicações que monitoram a rede os administradores podem saber, em tempo real, a situação de cada "componente" dentro da rede.

Ademais, Manoel Veras[16] explica que o Network Operation Center (NOC) deve ter sistemas de monitoração de energia, clima, temperatura e umidade. Deve operar 24 horas por dia e ter meios redundantes de comunicação.

### 2.3.2 *DCIM - Data Center Infrastructure Management*

O *DCIM - Data Center Infrastructure Management* é uma importante ferramenta para monitoramento e gestão de um data center, que possibilita uma capacidade de ação mais abrangente e eficiente do que as funções convencionais de gestão, objetivando a centralização do monitoramento, controle e o planejamento.

O DCIM fornece dados operacionais, incluindo dados ambientais (temperatura, umidade, entre outros) somado a uma gestão de inventário de ativos de TI, o que proporciona uma visão completa e do desempenho de um , em tempo real, através de painéis, avisos e alertas de acordo com seu nível de criticidade.

Para Fagundes[8], o DCIM centraliza o monitoramento, a gestão e o planejamento de capacidade de forma inteligente para garantir a alta disponibilidade de sistemas de missão

crítica.

As implementações de DCIM envolvem softwares especializados, hardware e sensores e permite o estabelecimento e o monitoramento de métricas de desempenho dos data centers.

## 2.4 Ferramentas e serviços utilizados

### 2.4.1 Zabbix

O Zabbix[7] é uma solução de monitoramento distribuído, gratuita, de código aberto, que monitora vários parâmetros de uma rede e toda sua infraestrutura de TI, com funções de descoberta e inventário, alertas baseados em gatilhos (*triggers*), relatórios, visualização de dados com base nos dados armazenados, permitindo criar gráficos e mapas de aspectos da rede, entre outros[8].

### 2.4.2 IFTTT - If This Then That

IFTTT - If This Then That é uma plataforma gratuita no qual os seus usuários podem criar "receitas"com uma regra básica, que dá nome ao serviço: se acontecer isso, faça aquilo (*if this then that*)[9].

No IFTTT, os usuários podem criar receitas utilizando diferentes serviços na nuvem ou dispositivos, interagindo entre eles, seja enviando *e-mail*, utilizando o Google Gmail, salvando arquivos no Google Drive ou Dropbox, reportando em um canal do Slack, enviando alertas para o Twitter, SMS ou realizando ligações. As possibilidades são inúmeras.

Além disso, o IFTTT permite criar mini aplicativos (*applets*) que funcionam com qualquer dispositivo ou aplicativo que possa fazer ou receber uma solicitação da web (*webhook*)[10].

Para facilitar o entendimento desta regra *IF THIS THAN THAT*, segue um exemplo prático utilizado nesse projeto, no qual integramos um dispositivo IoT com o Twitter, utilizando o serviço IFTTT:

Se (*IF*) a temperatura, lida pelo sensor de temperatura do dispositivo IoT, atingir

[7] `https://www.zabbix.com/`
[8] `https://www.zabbix.com/documentation/3.4/manual/introduction/sobre`
[9] `https://ifttt.com/discover`
[10] `https://ifttt.com/maker_webhooks/`

mais que 28 graus Celsius, ele informará ao serviço IFTTT, através de um webhook, que a temperatura está alta (*THIS*), enviando logo em seguida (*THEN*), a seguinte mensagem, em uma conta do Twitter, "Temperatura muito alta!"(*THAT*).

### 2.4.3 ThingSpeak

O ThingSpeak é uma plataforma aberta com análise de dados para Internet das Coisas, que permite armazenar, visualizar e analisar fluxos de dados em tempo real, na nuvem.

O ThingSpeak é compatível com Arduino, e possibilita a construção de canais abertos ou privados para dispositivos IoT, com painéis para facilitar a visualização dos dados, permitindo ainda, agregar dados sob demanda de fontes de terceiros, usar o poder do MATLAB, conforme figura 3, para entender seus dados coletados por dispositivos IoT e executar análises automaticamente com base em agendamentos ou eventos, permite ainda, integração dos seus dados com serviços de terceiros.[11].

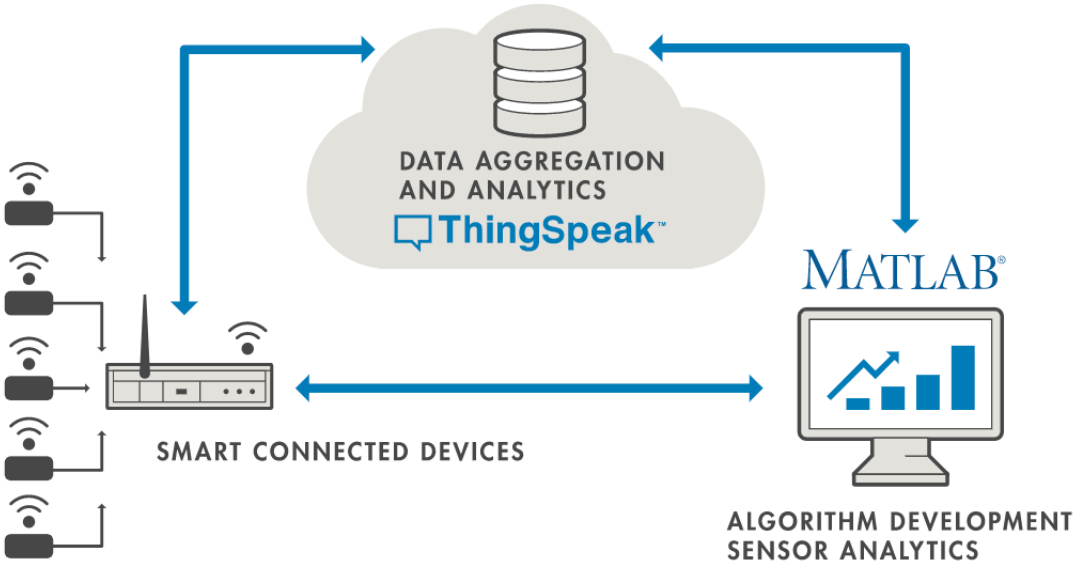



Figura 3: ThingSpeak[12]

## 2.5 Tecnologias de Comunicação utilizadas

### 2.5.1 IPMI e *Out of Band Management*

Segundo o Wikipedia, *IPMI - Intelligent Platform Management* é uma interface padronizada para gerência de hardware utilizada por administradores de sistema para monitorar

[11] `https://thingspeak.com`

sistemas de computadores. Este padrão é promovido pela Intel, Dell, HP e NEC[19].

Com IPMI pode-se monitorar dados dos servidores como temperatura, a tensão, estado da ventilação e das fontes de energia, desligamento e reinício de forma remota. Essa interface é uma espécie de *Out of Band Management*.

*Out of Band Management* permite que o administrador/operador da rede, se conecte de forma segura, a função de gerenciamento do dispositivo. Ele também pode ser usado para garantir a conectividade de gerenciamento independentemente do status de outros componentes desse dispositivo, usando um canal de gerenciamento dedicado para a sua manutenção, permitindo que monitore e gerencie, de forma remota, independentemente do dispositivo estar ligado ou não.

Atualmente, no TCE/RN são utilizados servidores (*hosts*) da empresa Dell, que utiliza o iDRAC, e é baseado no padrão IPMI 2.0, que permite o uso de interfaces fora de banda IPMI[7] e era a forma na qual os dados de temperatura eram colhidos, dos sensores das placas-mães dos servidores (*hosts*).

### 2.5.2 Zabbix Sender e Zabbix Trapper

Para comunicação dos dispositivos IoT com o Zabbix, foi utilizado um protocolo de comunicação baseado em JSON para receber dados do remetente do Zabbix com a ajuda do item de captura, Zabbix Sender e Zabbix Trapper, do próprio Zabbix.

Zabbix Sender e Zabbix Trapper, é a forma que o Zabbix utiliza para receber os dados de outros dispositivos, sem a necessidade de instalar agentes, uma vez que os dados são enviados pelos dispositivos utilizando o Zabbix Sender e recepcionados através do Zabbix Trapper, pelo servidor do Zabbix, com itens de Trap previamente cadastrados.

A utilização do Zabbix Sender e Zabbix Trapper é recomendado quando não for possível instalar o agente do Zabbix no dispositivo ou quando os dados demoram muito para serem processados e ultrapassar o *timeout* do servidor Zabbix ou do agente durante a sua coleta.

### 2.5.3 APIs, Webhooks, POST e JSON

A comunicação dos dispositivos IoT e o serviço IFTTT utiliza Webhook, enquanto a comunicação com o serviço ThingSpeak utiliza API - Application programming interface.

Para André Fernandes[9], API ou Interface de programação de aplicações é um tipo

de “ponte” que conectam aplicações", ele afirma ainda que as APIs "proporcionam a integração entre sistemas que possuem linguagem totalmente distintas de maneira ágil e segura.

Rodrigo Dantas[5], define Webhook como uma maneira prática para um app ou sistema fornecer outras aplicações com informações em tempo real. O webhook fornece dados para outros aplicativos. Eles são muito eficientes tanto para o prestador de serviço, como para o consumidor.

A principal diferença entre o funcionamento dos Webhooks e APIs é que, embora as APIs façam chamadas sem saber se terão alguma resposta, os Webhooks recebem chamadas por meio de POSTs HTTP somente quando eles possuem algumas atualizações de dados.

Segundo o Wikipedia, POST é um dos muitos métodos de requisição suportados pelo protocolo HTTP usado na World Wide Web. O método de requisição POST foi projetado para solicitar que o servidor web aceite os dados anexados no corpo da mensagem de requisição para armazenamento[21].

JSON - JavaScript Object Notation é um formato de troca de dados, de leitura fácil por humanos, e gerável e analisável por máquinas, baseado em um subconjunto da Linguagem de Programação JavaScript, Padrão ECMA-262 3ª Edição - Dezembro de 1999.

JSON é um formato de texto completamente independente do idioma, mas usa convenções que são familiares aos programadores da linguagem C. Essas propriedades tornam o JSON uma linguagem de intercâmbio de dados ideal [13].

[13] `http://www.json.org/`

# 3 Desenvolvimento do Projeto

O projeto foi concebido com intuito de monitorar o ambiente da sala do d*ata center* do TCE/RN, em diferentes pontos da sala, incluindo as saídas dos ares-condicionados, através de quatro (04) módulos IoT, coletando dados de temperatura e umidade em todos eles e no módulo 00/01 também coletando dados de gases e fumaça. Os dados coletados são enviados, de forma independente, para as ferramentas/serviços Zabbix, IFTTT e ThingSpeak, onde são armazenados ou usados para determinadas ações ou alertas, em caso de incidentes ou determinados gatilhos, conforme demonstra a figura 4, detalhado ao longo deste capítulo.

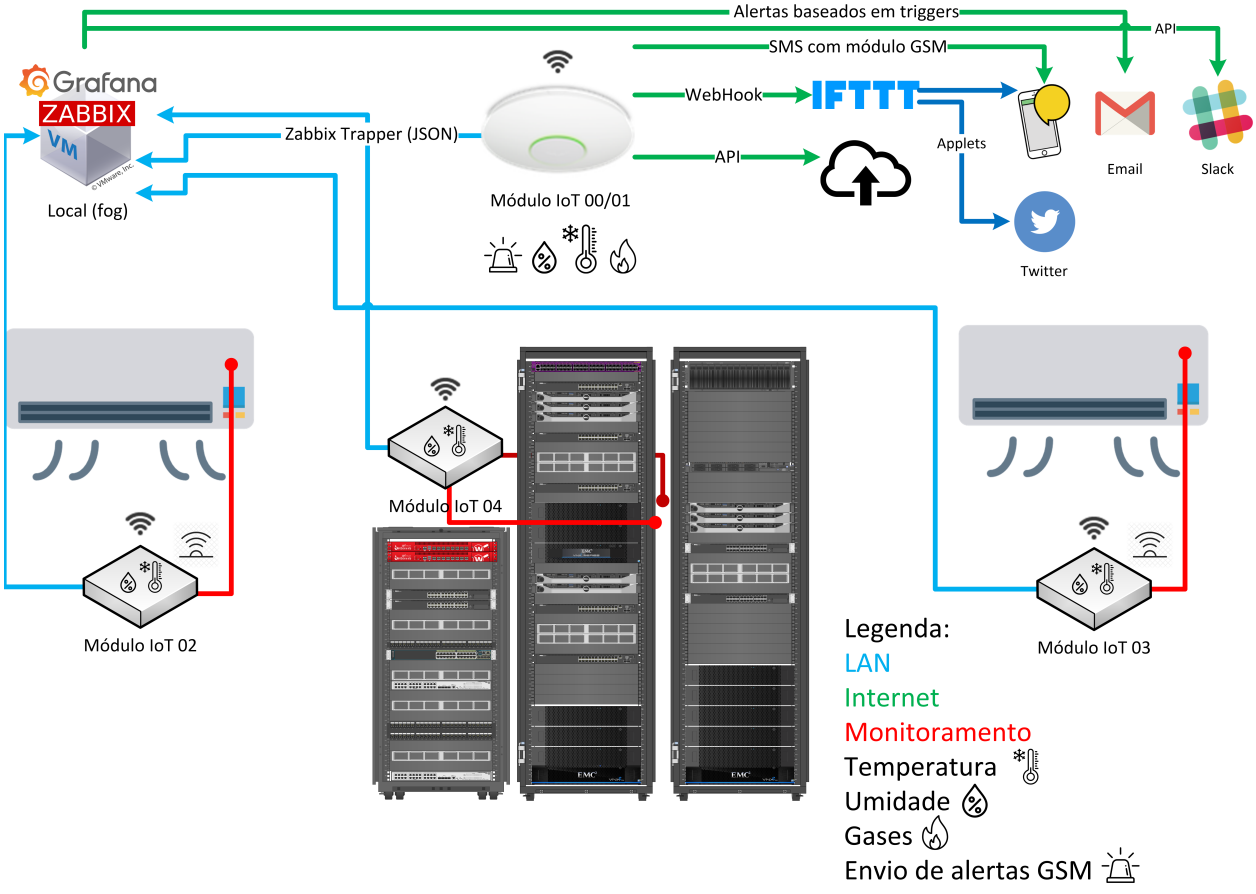



Figura 4: Layout Projeto.

O projeto foi dividido em quatro fases: Iniciação, Planejamento, Execução/Monitoramento, e Encerramento, baseado no Life Cycle Canvas LCC (apêndice C), de Manoel Veras [17], para acompanhamento e documentação do projeto.

## 3.1 Fase de Iniciação

Na fase de iniciação, formalizou-se o projeto, com o Termo de Abertura do Projeto (apêndice C), em reunião com a Diretoria de Informática do TCE/RN e IMD, através dos professores do PBL, partes interessadas do projeto, e a equipe do projeto, Davi Cunha, Analista de Controle Externo do TCE/RN, e Daniel Dantas, Engenheiro Eletricista, ambos, alunos do programa de residência do IMD.

Foi estabelecido cronograma (apêndice D) e preenchido o quadro inicial do LCC, Termo de Abertura do Projeto (apêndice C).

Dentre os requisitos, ficou estabelecido 2 requisitos:

- Monitorar a temperatura da sala do *data center*;
- Que o sistema enviasse alertas por SMS e/ou chamadas telefônicas.

Em relação aos benefícios, foram listados os seguintes:

- Maior segurança da sala do *data center* e anexos;
- Redundância da avaliação da temperatura dos equipamentos;
- Maior agilidade nas medidas reativas em casos de sinistros.

Em contrapartida, o projeto teria pelo menos duas grandes restrições, dentre elas, a falta de conhecimento da solução na tecnologia Arduino, e, em razão de ser um órgão público, não conseguir adquirir as placas e sensores em tempo hábil.

Como principais riscos para o projeto, elencou-se os seguintes:

- Falta de crédito para realizar as ligações/envio dos SMS;
- Extravios;
- Danos aos dispositivos.

Ao final da reunião, foi acordado que seriam feitas entregas quinzenais e que seriam usados meios informais de comunicação para o projeto.

## 3.2 Fase de Planejamento

A fase de planejamento, iniciou-se com a prototipagem dos módulos, chamados, inicialmente, de Alpha e Beta, por ainda serem protótipos, conforme figuras 5 e 6.

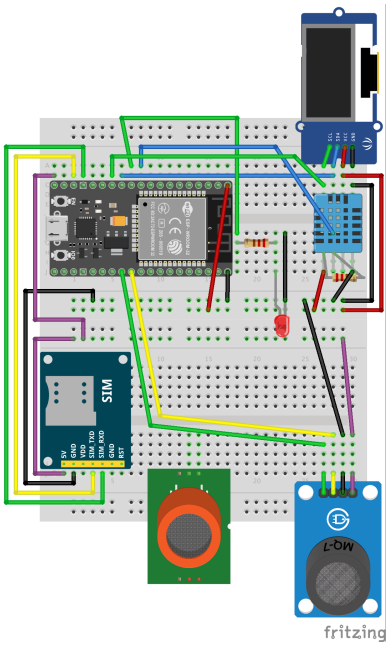



Figura 5: Protótipo Alpha dos módulos 00 e 01.

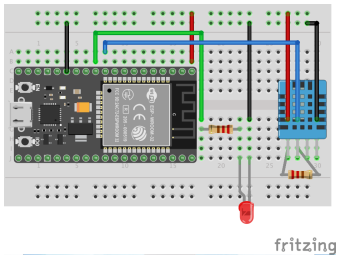



Figura 6: Protótipo Beta dos módulos 02, 03 e 04.

Ademais, foi realizado um comparativo com outras soluções existentes no mercado, pesquisa mercadológica para aquisição, através do TCE/RN, dos componentes para o projeto (microcontroladores, módulos, sensores, entre outros).

Como resultado da pesquisa, optou-se por utilizar a plataforma do Arduino[1], em razão das suas bibliotecas, por ser mais difundida e possuir uma ampla documentação na Internet.

O microcontrolador escolhido foi o ESP32, da Espressif, em razão desse microcontrolador possuir versões genéricas mais baratas no mercado, além de possuir 2 núcleos de

[1] `https://www.arduino.cc/`

processamento, além de ser de baixo consumo energético, possuir interfaces para redes WiFi e Bluetooth embutidas, e ser compatível com a plataforma Arduino.

Foi definido também que os dados gerados pelo sistema de monitoramento seriam salvos na rede local (*fog*), utilizando a ferramenta de monitoramento que o TCE/RN já utilizava, o Zabbix, contudo, para não haver nenhum conflito com o servidor Zabbix de produção, foi definido que seria instalada uma nova instância do Zabbix para homologação dessa solução (o roteiro de instalação pode ser observado no apêndice E), e ao mesmo tempo, enviados para alguma solução gratuita, porém segura, na nuvem (*cloud*), optando-se pelo ThingSpeak por ser uma plataforma para Internet das Coisas bastante consolidada e possuir canais privados para envio dos dados, integrações com outros aplicações e alertas.

Além disso, foi decidido que os dados seriam mostrados na sala de Infraestrutura de TI, através de painéis (*dashboards*) utilizando a ferramenta Grafana, integrado com a ferramenta de monitoramento Zabbix. Para a instalação do Grafana e integração com o Zabbix foi utilizado a documentação oficial do Grafana Labs[2].

Em relação aos alertas, foi definido a necessidade de um módulo de telefonia móvel GSM por ser requisito do projeto o envio de alertas de forma independente, caso o serviço de Internet do TCE/RN estivesse comprometido.

Ademais, estabeleceu-se também a utilização da plataforma IFTTT [3] para envio de alertas para o Slack [4], que é uma plataforma colaborativa de mensagens utilizada pela Diretoria de Informática do TCE/RN, mensagem eletrônica (*e-mail*) para a equipe de Infraestrutura de TI do TCE/RN, e outras interações com serviços na nuvem.

## 3.3 Fase de Execução e Monitoramento

Nesta fase, foram instaladas e configuradas todas as ferramentas necessárias para e execução do projeto nas estações de trabalho da equipe do projeto, dentre elas a IDE do Arduino, *drivers* dos microcontroladores ESP32, e bibliotecas compatíveis com esse microcontrolador.

A fase de Execução e Monitoramento foi dividida em entregas, com prazo de 15 dias, entre elas, conforme cronograma, no apêndice D:

[2]https://grafana.com/docs/installation/debian/
[3]https://ifttt.com/discover
[4]https://slack.com/

- Entrega 1: Sensor de temperatura e umidade (DHT) configurado para leitura via terminal;
- Entrega 2: Sensor de temperatura e umidade (DHT) configurado para leitura via rede;
- Entrega 3: Rede de monitoramento DHT acessada via rede;
- Entrega 4: Detector de fumaça e gases monitorados acessado via rede;
- Entrega 5: Sistema de envio de mensagens com GSM.

Essa entregas foram homologadas e consolidadas em módulos para que fosse possível validá-las em conjunto na fase de monitoramento.

### 3.3.1 Módulos IoT

#### 3.3.1.1 Prototipagem e Esquemas dos Módulos

Para documentação, prototipagem e diagramas esquemáticos dos módulos, utilizou-se a ferramenta Fritzing [5], conforme as figuras dos protótipos do módulo 00/01, figura 7, e módulos 02, 03 e 03, figura 9 e seus respectivos diagrama esquemáticos, figura 8 e figura 10.

#### 3.3.1.2 Módulo 00/01

O módulo 00/01, tem como função a coleta dos dados de ambiente, demonstrando esses valores, através de um display OLED, no próprio módulo, e envio desses dados para a ferramenta de monitoramento Zabbix, interação com o serviço IFTTT, bem como disparar alertas, através do Zabbix ou Internet, caso os limites (*threshold*) de alta temperatura e umidade sejam atingidos, bem como possíveis incidentes como fumaça, vazamentos de gases.

Além de usar o serviço de Internet do TCE/RN, esse módulo usa a rede de telefonia móvel, através de um módulo GSM, para envio de alertas, em caso de incidentes, efetuando uma ligação à cobrar, e enviando mensagem do tipo SMS para números pré-determinados, independente se o serviço de Internet estiver funcionando ou não.

[5]Fritzing: http://fritzing.org/home/

Os componentes do módulo 00/01 foram montados em uma base de prototipagem de equipamentos eletro-eletrônicos, conhecida por *breadboard*, mas comumente chamada de *protoboard*, em uma carcaça de um equipamento de rede sem fio *Access Point* danificado, preso ao teto da sala do *data center* do TCE/RN, em um ponto central da sala, para fácil verificação e melhor coleta dos sensores, conforme figura 13, do apêndice A.

Em relação aos componentes do módulo 00/01, foram utilizados os seguintes módulos, sensores, atuadores e microcontrolador:

- 1 x Microcontrolador ESP32;
- 1 x Sensor de umidade e temperatura DHT11 no próprio módulo;
- 1 x Módulo GSM GPRS SIM800L;
- 1 x Sensor de gás MQ-7 Monóxido de Carbono;
- 1 x Sensor de gas MQ-2 Inflamável e Fumaça;
- 1 x Display OLED 0.96 polegadas I2C Branco;
- 1 x Sinalizador de áudio (*Buzzer*) ativo 5V;
- 1 x Led RGB de alto brilho difuso 5mm;

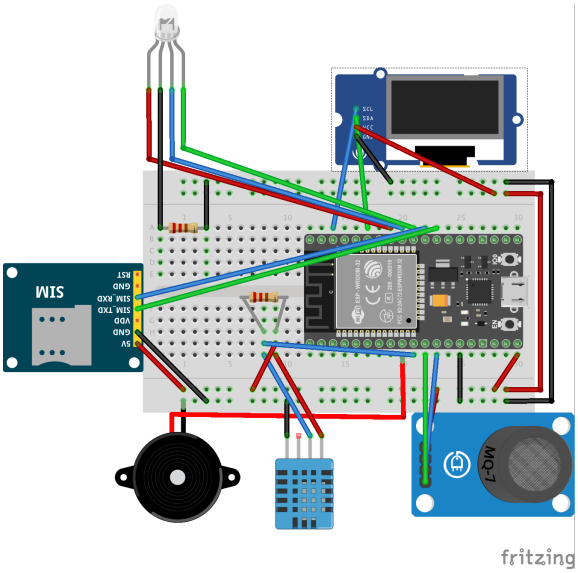



Figura 7: Protótipo: Modulo 00.

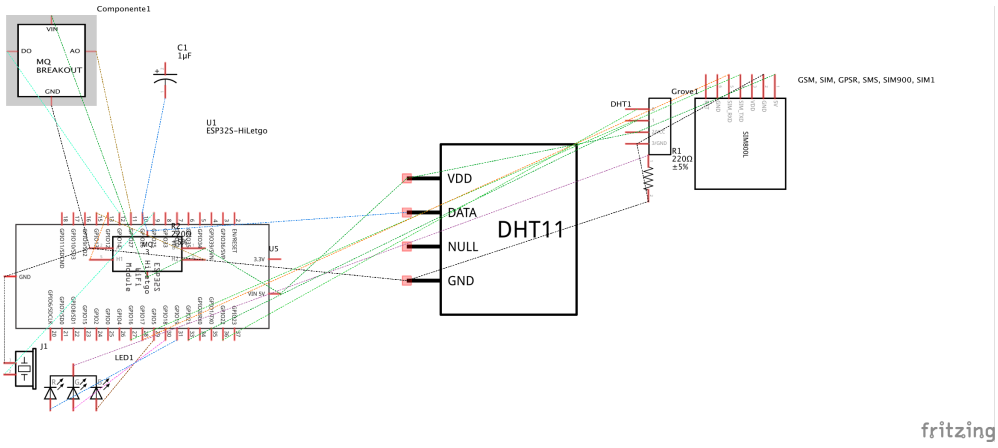



Figura 8: Esquema: Modulos 00 e 01.

#### 3.3.1.3 Módulos Secundários

Os módulos secundários (módulos 02, 03 e 04) tem como objetivo a coleta dos dados de temperatura e umidade em diferentes pontos da sala do *data center* do TCE/RN, por sensores DHT, e envio desses dados para a ferramenta de monitoramento Zabbix, interação com o serviço IFTTT, bem como disparar alertas, através do Zabbix ou Internet, caso os limites (*threshold*) de alta temperatura e umidade sejam atingidos.

Em relação aos componentes dos módulos secundários, foram utilizados os seguintes módulos, sensores, autuadores e microcontrolador:

- 1 x Microcontrolador ESP32;
- 2 x Sensores de umidade e temperatura DHT11 no próprio módulo;
- 1 x Sinalizador de áudio (*Buzzer*) Ativo 5V;
- 1 x LED RGB alto brilho difuso 5mm;
- 1 x LED emissor Infravermelho IR 5mm.

Os componentes dos módulos secundários foram montados em protoboards, em diversos pontos central da sala, com dois sensores DHT11, em cada módulo, sendo um no próprio módulo e outro em um "rabicho", saindo do módulo, para adicionar outro ponto de medição.

Para os módulos secundários, serão confeccionadas/compradas cases, porém, por questões administrativas do TCE/RN, ainda não foram adquiridas e atualmente estão nas protoboards, conforme verifica-se nas figuras 14, 15, 16, 17 e 18.

Além disso, foi utilizado um LED emissor infravermelho para controlar o ar condicionado mais próximo nos módulos 02 e 03.

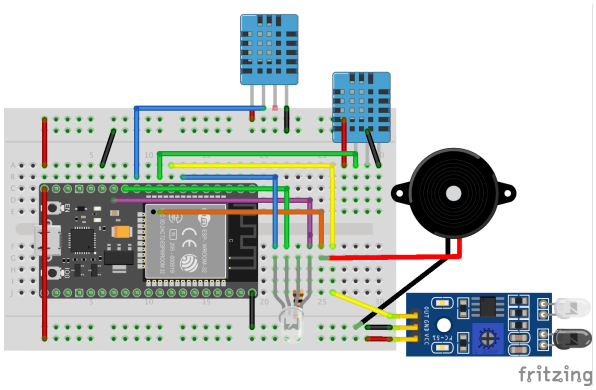



Figura 9: Protótipo: Modulo 02,03 e 04.

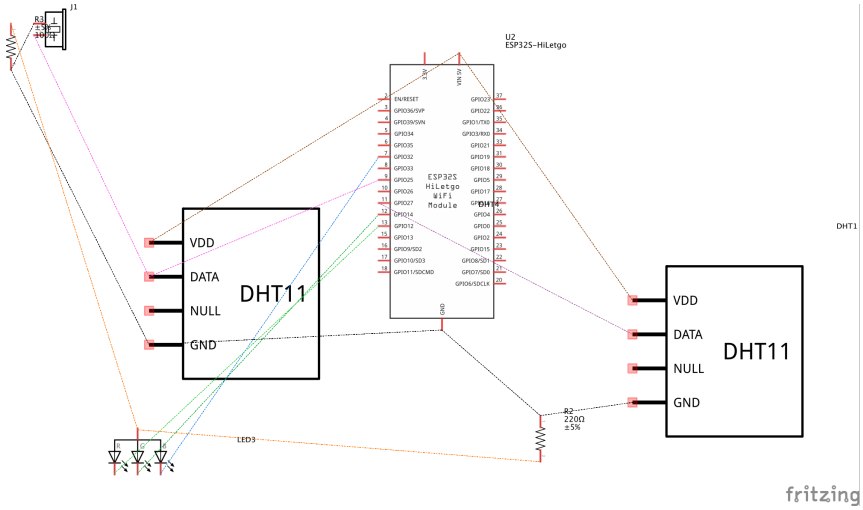



Figura 10: Esquema: Modulos 02, 03 e 04.

#### 3.3.1.4 Código Fonte

Em relação ao código fonte do módulos IoT, foram utilizados os exemplos das próprias bibliotecas do Arduino IDE.

Os códigos dos módulos IoT podem ser verificados no apêndice B do presente trabalho. A estrutura do código desses módulos estão divididas basicamente da seguinte forma:

- Bibliotecas Arduino:

  Wifi, biblioteca para conexão com a rede sem fio;

  WifiClientSecure, biblioteca para conexão através da rede sem fio de forma segura (HTPPS);

NTPClient, biblioteca para sincronizar a hora do microcontrolador com servidor NTP;

Hardware Serial, biblioteca usada para o módulo GSM SIM800L;

SSD1306, biblioteca usada pelo display OLED;

DHTesp, biblioteca usada para o sensor de temperatura e umidade DHT11;

- Definições dos pinos dos módulos, sensores e autuadores;
- Outras definições;
- Setup, inicializa os módulos, definições de saída e entrada dos pinos e chama alguns funções;
- Loop, chama as funções que precisam ser chamadas de forma recorrentes;
- Funções:

  WifiConnect(), função para conectar à rede sem fio;

  OLED(), função para mostrar no display OLED do módulo algumas situações;

  DHT(), função para utilizar o sensor de temperatura e umidade DHT11;

  LED e Buzzer, aciona os LEDs e buzzer em determinadas situações;

  IFTTTReport(), interage com IFTTT para reportar a cada 6 horas os dados dos sensores;

  IFTTTAlert(), interage com IFTTT, reportando, em caso de incidente;

  MQ(), função para utilizar os sensores MQ2 e MQ7;

  SendSMS(), função para enviar SMS em caso de incidente;

  callNumber(), função para ligar em caso de incidente;

  ZabbixFog(), função para enviar os dados para o Zabbix (Zabbix Sender);

  ThingspeakCloud(), função para enviar os dados para ThingSpeak (API).

  DewPoint(), função para calcular o ponto de orvalho, apêndice B.2.

Além disso, os códigos fontes dos módulos foram versionados na ferramenta de versionamento utilizada pelo TCE/RN, GitLab, com intuito de facilitar a manutenção e documentação, em um repositório privado, com intuito de manter o sigilo das senhas e chaves de acesso às *APIs* utilizadas nos códigos.

### 3.3.2 Dados

Os dados coletados através dos módulos IoT são recepcionados pelo Zabbix e serviços ThingSpeak, e IFTTT, já mencionados.

Em relação ao Zabbix, os dados são enviados através Zabbix Sender (JSON), da seguinte forma[6]:

```
{
        "request":"sender data",
        "data":[
                {
                        "host":"nodeiot04",
                        "key":"LeituraDHT01.temperature",
                        "value":"30"
                }
        ]
}
```

Os dados enviados são recebidos pelo Zabbix, na porta 10051, com o Zabbix Trapper[7], através de uma key, previamente configurada, atribuída a um host (módulos IoT) correspondente) no Zabbix. Esses dados são armazenados por 365 dias no Zabbix, por padrão.

No que se refere ao ThingSpeak, os dados são enviados, utilizando a API do ThingSpeak, com uma chave única (*Key*), em uma requisição HTTP, através do método POST (JSON), na seguinte url `https://api.thingspeak.com/update.json`, conforme verificamos:

```
POST /update.json HTTP/1.1
Host: api.thingspeak.com
Content-Type: application/x-www-form-urlencoded
User-Agent: PostmanRuntime/7.15.0
Accept: */*
Cache-Control: no-cache
Postman-Token: bd847149-4f38-4435-a1b2-8e061add54db,ccf428da-43f3-4949-92b2-aa45a07684ca
Host: api.thingspeak.com
cookie: request_method=POST
accept-encoding: gzip, deflate
content-length: 44
```

[6] https://www.zabbix.com/documentation/3.4/manual/appendix/items/trapper
[7] https://www.zabbix.com/documentation/3.4/manual/config/items/itemtypes/trapper

```
Connection: keep-alive
cache-control: no-cache

api_key=KEY_THINGSPEAK&field1=35&field2=35
```

Os dados recepcionados no ThingSpeak são armazenados de forma secundária, já que o armazenamento já está sendo feito no Zabbix.

Quanto ao IFTTT, os dados são enviados utilizando o Webhook, com uma chave única (*Key*), em uma requisição HTTP, através do método POST (JSON), na seguinte url `https://maker.ifttt.com/trigger/EVENT_IFTTT/with/key/KEY_IFTTT`, conforme verificamos:

```
POST /trigger/EVENT_IFTTT/with/key/KEY_IFTTT HTTP/1.1
Host: maker.ifttt.com
Content-Type: application/json
User-Agent: PostmanRuntime/7.15.0
Accept: */*
Cache-Control: no-cache
Postman-Token: 40f179eb-2f9b-4c38-971e-63db8ae01c43,fd038cf1-dba1-4de2-a7a6-166a96ebe37f
Host: maker.ifttt.com
accept-encoding: gzip, deflate
content-length: 50
Connection: keep-alive
cache-control: no-cache

{ "value1" : "3", "value2" : "2", "value3" : "1" }
```

Essas interações foram validadas utilizando a ferramenta Postman [8] e podem ser observadas no layout do projeto, figura 4.

### 3.3.3 Ações e Alertas

Os alertas de temperatura e umidade foram criados baseados na NBR 14565:2001, conforme já vimos na figura 2, do Prof. Dr. Paulo Sérgio Marin [14].

Esses alertas foram configurados baseados em gatilhos (*triggers*), através da ferramenta de monitoramento Zabbix, com envio de mensagens eletrônicas (*e-mail*) em caso de incidentes, conforme figura 11.

[8] https://www.getpostman.com/

Para os alertas foram estabelecidos cinco (05) níveis de criticidade, padrão do Zabbix:

1. Information: temperatura superior a 27 graus celsius e/ou umidade abaixo de 30% ou acima de 65% por um período maior de tempo.
2. Warning: temperatura superior a 28 graus celsius ou umidade acima de 70% por um certo período de tempo.
3. Average: Temperatura acima de 30 graus celsius ou umidade acima de 75% por um curto período de tempo.
4. High: Temperatura acima de 35 graus celsius ou umidade acima de 80%.
5. Disaster: Temperatura acima de 40 graus ou umidade acima de 90%.

| Severity | Name ▲ | Expression | Status |
|---|---|---|---|
| Information | Temperatura acima de 27 graus por 30 minutos | {Template IoT - DHT:LeituraDHT01.temperature.**min**(30m)}>27 | Enabled |
| Warning | Temperatura acima de 28 graus por 20 minutos | {Template IoT - DHT:LeituraDHT01.temperature.**min**(20m)}>28 | Enabled |
| Average | Temperatura acima de 30 graus por 10 minutos | {Template IoT - DHT:LeituraDHT01.temperature.**min**(10m)}>30 | Enabled |
| High | Temperatura acima de 35 graus | {Template IoT - DHT:LeituraDHT01.temperature.**last**()}>35 | Enabled |
| Disaster | Temperatura acima de 40 graus | {Template IoT - DHT:LeituraDHT01.temperature.**last**()}>40 | Enabled |
| Information | Umidade abaixo de 30 % por 60 minutos | {Template IoT - DHT:LeituraDHT01.humidity.**min**(30m)}<30 | Enabled |
| Information | Umidade acima de 65 % por 60 minutos | {Template IoT - DHT:LeituraDHT01.humidity.**min**(60m)}>65 | Enabled |
| Warning | Umidade acima de 70 % por 50 minutos | {Template IoT - DHT:LeituraDHT01.humidity.**min**(50m)}>70 | Enabled |
| Average | Umidade acima de 75 % por 40 minutos | {Template IoT - DHT:LeituraDHT01.humidity.**min**(40m)}>75 | Enabled |
| High | Umidade acima de 80 % por 30 minutos | {Template IoT - DHT:LeituraDHT01.humidity.**min**(30m)}>80 | Enabled |
| Disaster | Umidade acima de 90 % por 20 minutos | {Template IoT - DHT:LeituraDHT01.humidity.**min**(20m)}>90 | Enabled |

Displaying 11 of 11 found

Figura 11: Triggers configuradas na ferramenta Zabbix.

No serviço IFTTT, foram configuradas receitas com determinadas ações em caso de incidentes, utilizando os mesmos níveis estabelecidos no Zabbix, através do Webhook, com envio de mensagem eletrônica (*e-mail*), mensagem do tipo SMS, em um serviço gratuito do próprio IFTTT (com limite mensal de SMS) e publicação de mensagem no canal de alertas no Slack e no Twitter em conta privada, com notificação nos telefones da equipe de infraestrutura de TI.

Por último, envio de mensagem do tipo SMS e ligação à cobrar para números pré-estabelecidos, utilizando o módulo GSM, no módulo 00/01, quando os níveis de criticidade mais altos (high e disaster) forem atingidos.

Em relação aos alertas de gases e fumaça, foram baseados nas informações técnicas dos próprios sensores e consiste na simples presença ou não do gás/fumaça no ambiente, e sempre serão considerados de criticidade mais alta (Disaster), alertando imediatamente através do sistema de monitoramento Zabbix, serviço IFTTT e módulo GSM.

Por fim, foi configurado painel (*dashboard*) na sala da equipe de infraestrutura de TI do TCE/RN (centro de operações de rede - NOC), utilizando a ferramenta Grafana, figura 12, com alertas visuais dos módulos IoT para que a equipe possa visualizá-los, enquanto estiverem na sala, em um televisor de 55".

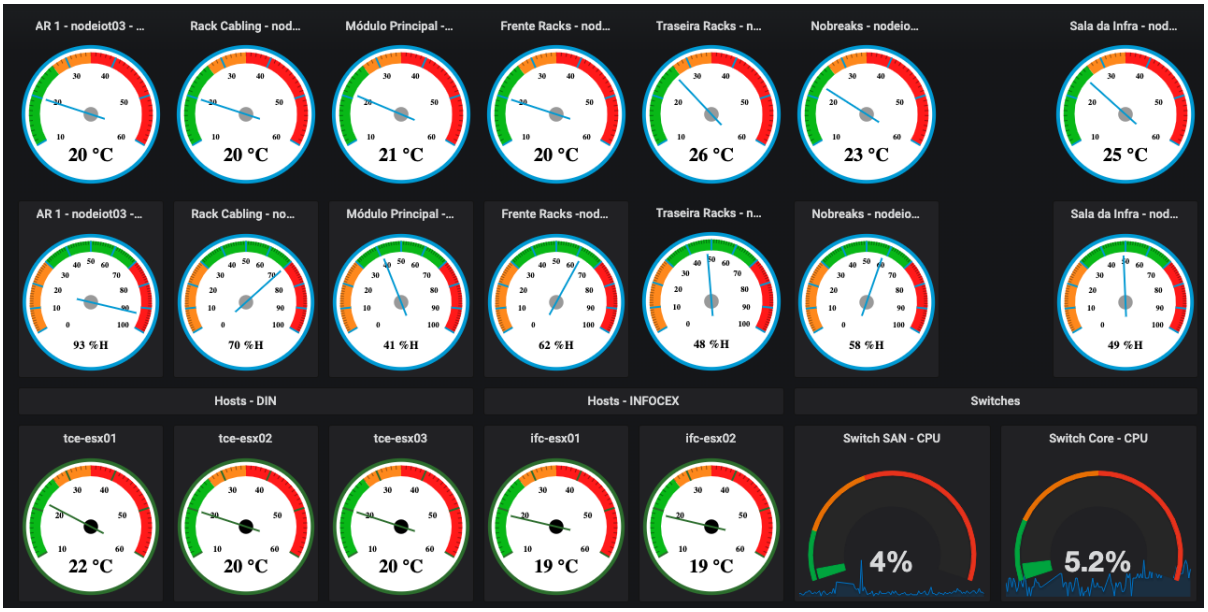



Figura 12: Painel de monitoramento do NOC.

## 3.4 Fase de Encerramento

Na última fase, a fase de encerramento, a solução foi colocada em produção, validada e sua entrega formalizada, conforme cronograma, apêndice D, e apresentada no Workshop realizado entre o Instituto Metrópole Digital - IMD e o TCE/RN.

# 4 Conclusões

Este trabalho teve como intuito relatar a implantação do sistema de monitoramento do ambiente do data center do TCE/RN utilizando Internet das Coisas, com sistema de alertas, através da ferramenta de monitoramento Zabbix, serviços na nuvem, e módulo de telefonia móvel GSM, com painéis de monitoramento para a sala de operação.

Ao comparar o sistema implantado (apêndice F) com outras soluções existentes no mercado (apêndice G), apesar de não ter incluído a mão de obra dos integrantes do projeto, uma vez que as outras soluções também teria custo com a implantação, aprendizagem, entre outros, que não foram considerados também, verifica-se que a solução com Internet das Coisas custou bem menos para o TCE/RN e resultou em um aprendizado em diversos benefícios além do custo.

Dentre os principais benefícios, do novo sistema de monitoramento utilizando IoT, para o TCE/RN temos:

- Solução de baixo custo;
- Monitoramento dedicado em vários pontos da sala do data center;
- Sistema de alertas independente do serviço de Internet do TCE/RN;
- Sistema integrado as ferramentas existentes;
- Menor tempo de resposta a incidentes.

Conclui-se, que a implementação de um sistema de monitoramento da sala do data center do TCE/RN, com a utilização de Internet das Coisas, combinados com a instalação de ferramentas e integração com serviços, oferece maior confiança, disponibilidade e integridade dos dados coletados.

## 4.1 Principais contribuições

Dentre as principais contribuições que esse sistema trouxe para o equipe de Infraestrutura de TI do TCE/RN, com pouco tempo após a sua implementação, foi poder demonstrar o histórico de temperatura para os superiores e solicitar a troca do sistema de ar condicionado atual, que se mostrou ineficaz para refrigerar a sala do data center.

## 4.2 Trabalhos Futuros

As aplicações da Internet das Coisas são inúmeras, dentre alguns possíveis trabalhos futuros a ser realizados no TCE/RN, utilizando IoT, temos:

- O monitoramento energético dos equipamentos críticos do data center, geradores, entre outros, utilizando IoT.
- Controle de acesso à sala do data center com IoT, e biometria ou outra forma de controle mais eficiente.
- Monitorado o fluxo de pessoas nos pontos de entrada do TCE/RN, com IoT e visão computacional, para calcular o número de visitantes e auxiliar a assessoria de segurança do TCE/RN.
- Controle de patrimônio (utilização de beacons ou etiquetas RFID/NFC e IoT) para auxiliar o setor de patrimônio do TCE/RN.
- Controle de frota veicular do TCE/RN, com utilização de dispositivos IoT, com módulos GPS, tags RFID/NFC, nos veículos, para otimização e evitar desperdícios.
- Por fim, os módulos IoT existentes poderão se integrados com sistemas para análises de dados, de forma preditivas (Watson Internet of Things[1], por exemplo), com base no histórico da temperatura e umidade, que já vem sendo coletado, e uma vez que a incidência de calor é diferente ao longo do ano, principalmente em uma das paredes da sala do data center TCE/RN, onde há incidência solar no período da tarde. Este projeto faria com que tenha um melhor ajuste da temperatura ambiente e melhor controle da umidade da sala, trazendo ainda, melhor eficiência energética dos equipamentos.

[1] `https://www.ibm.com/internet-of-things`

# Referências


[1] Kevin Ashton. That 'internet of things' thing., Junho 2009. Disponível em: <https://www.rfidjournal.com/articles/view?4986>. Acesso em junho 24, 2019.

[2] Axelos. Glossário itil® de português do brasil, v1.0, 29 de julho de 2011, baseado no glossário em inglês v1.0, Julho 2011. Disponível em: <https://www.axelos.com/Corporate/media/Files/Glossaries/ITIL_2011_Glossary_BR-PT-v1-0.pdf>. Acesso em junho 24, 2019.

[3] BNDES. Estudo "internet das coisas: um plano de ação para o brasil"., Novembro 2017. Disponível em: <https://www.bndes.gov.br/wps/portal/site/home/conhecimento/pesquisaedados/estudos/estudo-internet-das-coisas-iot/estudo-internet-das-coisas-um-plano-de-acao-para-o-brasil>. Acesso em junho 24, 2019.

[4] Presidência da República. Plano nacional de internet das coisas., Junho 2019. Disponível em: <http://www.in.gov.br/en/web/dou/-/decreto-n-9.854-de-25-de-junho-de-2019-173021041>. Acesso em junho 25, 2019.

[5] Rodrigo Dantas. O que são webhooks?, Dezembro 2015. Disponível em: <https://blog.vindi.com.br/o-que-sao-webhooks/>. Acesso em junho 28, 2019.

[6] Miguel de Sousa. *Internet of Things with Intel Galileo: Employ the Intel Galileo board to design a world of smarter technology for your home.* Packt Publishing, Birmingham, UK, 1st edition, 2015.

[7] Dell. Integrated dell remote access controller 8 (idrac8) - version 2.05.05.05 user's guide., Dezembro 2014. Disponível em: <https://topics-cdn.dell.com/pdf/idrac8-with-lc-v2.05.05.05_users-guide_en-us.pdf>. Acesso em junho 25, 2019.

[8] Eduardo Fagundes. Dcim – data center infrastructure management., Julho 2014. Disponível em: <https://efagundes.com/artigos/dcim-data-center-infrastructure-management/>. Acesso em junho 25, 2019.

[9] André Fernandes. O que é api? entenda de uma maneira simples., Março 2018. Disponível em: <https://vertigo.com.br/o-que-e-api-entenda-de-uma-maneira-simples/>. Acesso em junho 28, 2019.

[10] Ênio Figueiredo. Dcim (data center infrastructure management)., Julho 2017. Disponível em: <http://www.redesecia.com.br/data-center/dcim-data-center-infraestruture-management/>. Acesso em junho 24, 2019.

[11] Hwaiyu Geng. *Data Center Handbook.* John Wiley & Sons, Hoboken, New Jersey, 1st edition, 2015.

[12] Mike Kavis. Investors guide to iot part 1 - understanding the ecosystem., Fevereiro 2016. Disponível em: <https://www.forbes.com/sites/mikekavis/2016/02/24/investors-guide-to-iot-part-1-understanding-the-ecosystem/#6cdf663311a1>. Acesso em junho 24, 2019.

[13] Tero Karvinen Kimmo Karvinen. *Primeiros Passos com Sensores: Perceba o mundo usando eletrônica, Arduino e Raspberry Pi.* Novatec, São Paulo, 1st edition, 2014.

[14] Paulo Sérgio Marin. *Data centers: desvendando cada passo: projeto, infraestrutura física e eficiência energética.* Érica, São Paulo, 1st edition, 2011.

[15] Manoel Veras. *Datacenter: componente central da infraestrutura da TI.* Brasport, Rio de Janeiro, 1st edition, 2009.

[16] Manoel Veras. *Cloud Computing. Nova Arquitetura da TI.* Brasport, Rio de Janeiro, 1st edition, 2012.

[17] Manoel Veras. *Gestão Dinâmica de Projetos: LifeCycleCanvas.* Brasport, Rio de Janeiro, 1st edition, 2016.

[18] Wikimedia. Arduino., 2019.

[19] Wikimedia. Intelligent platform management interface ., 2019. Disponível em: <https://pt.wikipedia.org/wiki/Intelligent_Platform_Management_Interface>. Acesso em junho 24, 2019.

[20] Wikimedia. Microcontrolador., 2019. Disponível em: <https://pt.wikipedia.org/wiki/Microcontrolador>. Acesso em junho 24, 2019.

[21] Wikimedia. Post (http)., 2019. Disponível em: <https://pt.wikipedia.org/wiki/POST_(HTTP)>. Acesso em junho 25, 2019.

# APÊNDICE A – Fotos dos Módulos IoT

Figura 13: Foto do Módulo principal pendurado no teto do Data Center.


Figura 14: Foto do Módulo 02.

Figura 15: Foto do Sensor DHT na entrada de ar de um dos ar condicionados.

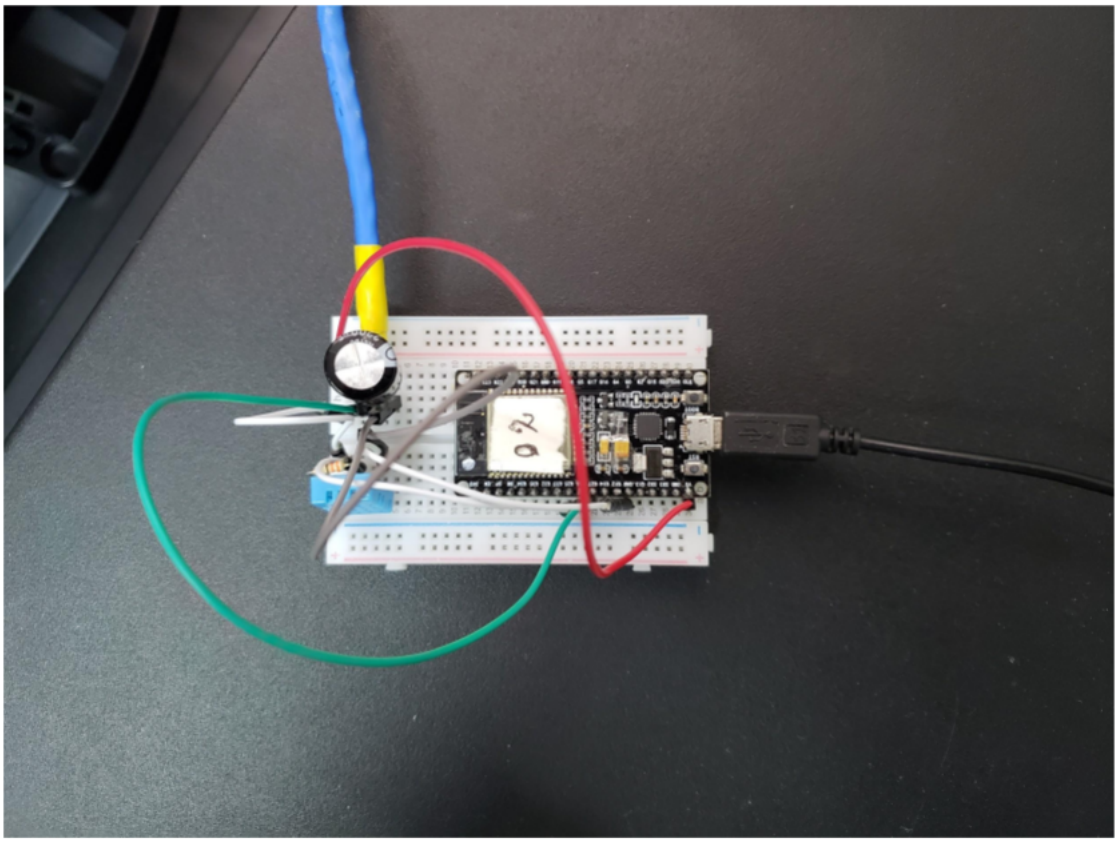


Figura 16: Foto do Módulo 03.

Figura 17: Foto do sensor do Módulo 04 na traseira dos racks - Corredor Quente.


Figura 18: Foto do sensor do Módulo 04 na frente dos racks - Corredor Frio.

# APÊNDICE B – Código Fonte dos módulos

## B.1 Código do Módulo 00/01

```
1  //Inclusao das bibliotecas principais
2  #include <DHTesp.h> //Carrega a biblioteca do sensor de temperatura e umidade DHT.
3  #include <Wire.h>  //Carrega a biblioteca para a comunicacao I2C.
4  #include "SSD1306Wire.h" //Carrega a biblioteca para o display OLED.
5  #include <WiFi.h> //Carrega a biblioteca da rede sem fio.
6  #include <HardwareSerial.h> //Carrega a biblioteca usada pelo SIM800L.
7  #include <String.h> //Carrega a biblioteca usada pelo SIM800L.
8  #include <NTPClient.h> //Carrega a biblioteaca usada para atualizar o relógio.
9  #include "Adafruit_MQTT.h"
10 #include "Adafruit_MQTT_Client.h"
11 #include "BluetoothSerial.h"
12 //
13 //Definicoes dos pinos no ESP32
14 #define SIM_RX  17  //Cria um alias para o RX do SIM800L no pino 17 do ESP32 TX.
15 #define SIM_TX  16  //Cria um alias para o TX do SIM800L no pino 16 do ESP32 RX.
16 #define LEDR 12 //Cria um alias para o LED Vermelho no pino 12 do ESP32.
17 #define LEDG 14 //Cria um alias para o LED Verde no pino 14 do ESP32.
18 #define LEDB 32 //Cria um alias para o LED Azul no pino 32 do ESP32.
19 #define BUZZER 33 //  //Cria um alias para o Buzzer no pino 33 do ESP32.
20 #define DHT01data 26  //Cria um alias para o modulo DHT1 no pino 26 do ESP32.
21 #define OLED_SCL 22 //Cria um alias para o pino SCL (verde) do OLED com o pino 32 do ESP32.
22 #define OLED_SDA 21 //Cria um alias para o pino SDA (azul) do OLED com o pino 33 do ESP32.
23 #define MQ2_analog 34 //Cria um alias para DO do sensor fumaca MQ7 no pino 34 do ESP32.
24 #define MQ2_digital 33 //Cria um alias para DO do sensor fumaca MQ7 no pino 33 do ESP32.
25 #define MQ7_analog 35 //Cria um alias para DA do sensor gas MQ2 no pino 35 do ESP32.
26 #define MQ7_digital 13  //Cria um alias para DA do sensor gas MQ2 no pino 13 do ESP32.
27 //
28 DHTesp dht01; //Cria um objeto do tipo DTHesp para o modulo DHT1.
29 //
30 //Definições do display OLED.
31 SSD1306Wire display(0x3c, OLED_SDA, OLED_SCL);
32 int OLEDState = 1;
33 //
```

```
34 //Definicoes do SIM800L (Enviar SMS e efetuar ligacao).
35 HardwareSerial MSIM800L(2); //Para o SIM800L
36 #define SIM800Lbauds 115200
37 #define SIM800Lserial SERIAL_8N1
38 int _timeout;
39 String _buffer;
40 String SMSNumber1 = "+55849XXXX"; //Tel. do coordenador Infraestrutura de TI.
41 String SMSNumber2 = "+55849XXXX"; //Tel. do técnico de sobreaviso.
42 String CallNumber = "90909XXXX"; //Liga a cobrar p/ tel. do técnico de sobreaviso.
43 //
44 //Define as variaveis da conexao sem fio (WiFi)
45 WiFiClient client; //Inicializa a biblioteca client
46 #define WIFI_SSID  "XXXX" //Nome da rede (SSID)
47 #define WIFI_PASS "XXXX" //Senha de rede sem fio
48 String IP;
49 //
50 // Definições do Adafruit.IO
51 #define AIO_SERVER      "io.adafruit.com"
52 #define AIO_SERVERPORT  1883
53 #define AIO_USERNAME    "XXXX"
54 #define AIO_KEY         "XXXX"
55 Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);
56 #define DHT1T_FEED "XXXX/feeds/dht1Temp" //XXXX/feeds/nodeiot01.dht1temp
57 #define DHT1U_FEED "XXXX/feeds/dht1Umid"
58 #define MQ2_FEED "XXXX/feeds/mq2"
59 #define MQ7_FEED "XXXX/feeds/mq7"
60 Adafruit_MQTT_Subscribe mqttled = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME "/feeds/led");
61 Adafruit_MQTT_Publish mqttdht1t = Adafruit_MQTT_Publish(&mqtt, DHT1T_FEED);
62 Adafruit_MQTT_Publish mqttdht1u = Adafruit_MQTT_Publish(&mqtt, DHT1U_FEED);
63 Adafruit_MQTT_Publish mqttmq2 = Adafruit_MQTT_Publish(&mqtt, MQ2_FEED);
64 Adafruit_MQTT_Publish mqttmq7 = Adafruit_MQTT_Publish(&mqtt, MQ7_FEED);
65 //
66 //Bluetooth Serial
67 #if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
68 #error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
69 #endif
70 BluetoothSerial SerialBT;
71 //
72 //Define as variaveis e APIs do IFTTT.
73 String IFTTT_API_KEY = "XXXX"; //Define a API do IFTTT.
74 String IFTTT_EVENT_NAME =  "ReportESP32"; // Define o evento REPORT do IFTTT.
75 String IFTTT_EVENT_NAME2 =  "AlertDHTESP32"; // Define o evento ALERT do IFTTT.
76 String IFTTT_EVENT_NAME3 =  "AlertSmokeESP32"; // Define o evento ALERT do IFTTT.
77 const char* IFTTT_host = "maker.ifttt.com"; //URL para usar o WebHook.
78 //
79 //Integracao com ThingSpeak (API)
80 //String thingspeakAPI = "XXXX"; //API do ThingSpeak para o Modulo IoT 01.
81 String thingspeakAPI = "XXXX"; //API do ThingSpeak para o Modulo IoT 00.
```

```
82  //String thingspeakAPI = "XXXX"; //API do ThingSpeak para o Modulo IoT 05.
83  const char* thingspeakHost = "api.thingspeak.com"; //URL da API do ThingSpeak.
84  //
85  // Configurações do Servidor NTP
86  const char* servidorNTP = "a.st1.ntp.br"; // Servidor NTP para pesquisar a hora
87  const int fusoHorario = -10800; // Fuso horário em segundos (-03h = -10800 seg)
88  const int taxaDeAtualizacao = 1800000; // Taxa de atualização do servidor NTP em milisegundos
89  WiFiUDP ntpUDP; // Declaração do Protocolo UDP
90  NTPClient timeClient(ntpUDP, servidorNTP, fusoHorario, 60000);
91  //
92  //Define as variaveis e dados do Zabbix (trapper)
93  float temperature;
94  float humidity;
95  const char* zbxserver = "0.0.0.0"; //IP do Zabbix.
96  const char* zbxhostname = "nodeiot00"; //Host no Zabbix.
97  int zbxporta = 10051;
98  //
99  //Outros
100 // Contador de tempo para a funcao do IFTTT, SMS e Call...
101 int IFTTT_Tempo = 0;
102 int IFTTT_Intervalo = 0;
103 int IFTTT_Alerta = 0;
104 int SMS_Intervalo = 0;
105 int SMS_Alerta = 0;
106 int Call_Intervalo = 0;
107 int Call_Alerta = 0;
108 int MQ_Intervalo = 0;
109 int MQ_Alerta = 0;
110 int Zbx_Tempo = 0;
111 int TS_Tempo = 0;
112 int Loop = 30;
113 int Loop2 = 60;
114 unsigned long previousMillis = 0;
115 unsigned long previousMillis2 = 0;
116 unsigned long previousMillis3 = 0;
117 const long interval10 = 10000;
118 const long interval30 = 30000;
119 const long interval60 = 60000;
120 //
121 Sensor MQ2 e MQ7
122 int MQ2_digital_value;//variavel usada para receber o sinal digital do sensor.
123 int MQ7_digital_value;//variavel usada para receber o sinal digital do sensor.
124 int MQ2_analog_value;//variavel usada para receber o sinal analogico do sensor.
125 int MQ7_analog_value;//variavel usada para receber o sinal analogico do sensor.
126 //
127 void setup() {
128   MSIM800L.begin(SIM800Lbauds, SIM800Lserial, SIM_TX, SIM_RX); // Inicia o SIM800L.
129   Serial.begin(115200); //Inicia o terminal para a leitura via console.
```

```
130   while (! Serial);
131   Serial.println("Inicializando o ESP32...");
132   SerialBT.begin(zbxhostname); //Bluetooth nome visível.
133   Serial.println("Hostname: " + String(zbxhostname));
134   timeClient.begin(); //Inicia o cliente NTP
135   pinMode(BUZZER, OUTPUT);  //Define que o Buzzer sera um pino de saída.
136   pinMode(LEDR, OUTPUT);  //Define que o pino do LEDR sera um pino de saída.
137   pinMode(LEDG, OUTPUT);  //Define que o pino do LEDG sera um pino de saída.
138   pinMode(LEDB, OUTPUT);  //Define que o pino do LEDB sera um pino de saída.
139   digitalWrite(BUZZER, LOW); //Define que o pino do Buzzer estará desligado.
140   digitalWrite(LEDR, LOW); //Define que o pino do LEDR estará desligado.
141   digitalWrite(LEDG, LOW); //Define que o pino do LEDG estará desligado.
142   digitalWrite(LEDB, LOW); //Define que o pino do LEDB estará desligado.
143   pinMode(MQ2_analog, INPUT);  //Define que o pino do sensor MQ2 sera um pino de entrada.
144   pinMode(MQ7_analog, INPUT);  //Define que o pino do sensor MQ7 sera um pino de entrada.
145   pinMode(MQ2_digital, INPUT);  //Define que o pino do sensor MQ2 sera um pino de entrada.
146   pinMode(MQ7_digital, INPUT);  //Define que o pino do sensor MQ7 sera um pino de entrada.
147   pinMode(DHT01data, INPUT);  //Define que o pino do sensor DHT sera um pino de entrada.
148   dht01.setup(DHT01data, DHTesp::DHT11); //Inicializacao do sensor DHT01.
149   display.init(); //Inicia o display
150   display.flipScreenVertically(); //Gira a exibicao do dislay.
151   SetupOLED();
152   testeLEDs();
153   BeepLongo();
154   BeepMedio();
155   BeepCurto();
156   wifiConnect();  //Conecta ao WiFi
157 }
158 void loop() {
159   unsigned long currentMillis = millis();
160   // Chama as seguintes funcoes:
161   if (currentMillis - previousMillis >= interval10) {
162     //  Serial.println("Entrou no 10s loop... " + String(currentMillis) + " - "
163     + String(previousMillis2));
164     //  Serial.println("Hora1: " + String(timeClient.getFormattedTime()) );
165     previousMillis = currentMillis;
166     if (OLEDState == 0) {
167     OLEDState = 1;
168     ambienteOLED(); //OLED1
169     } else {
170       previousMillis = currentMillis;
171       OLEDState = 0;
172       outrosOLED(); //OLED2
173     }
174   }
175   if (currentMillis - previousMillis2 >= interval30) {
176     //  Serial.println("Entrou no 30s loop... " + String(currentMillis) + " - "
177     + String(previousMillis2));
```

```
178       //  Serial.println("Hora2: " + String(timeClient.getFormattedTime()) );
179       previousMillis2 = currentMillis;
180       wifiConnect();  //Conecta a rede sem fio
181       MQTT_connect(); //Adafruit MQTT
182       MQTT_publish(); //Adafruit publica os feeds
183       MQTT_Subscribe();
184       blueToothSerial(); //Funcao para publicar no serial do BLE
185       DHT1(); //Chama a funcao do sensor DHT para mostrar no serial
186       sensoresMQ(); //Chama a funcao dos sensores MQ2 e MQ7 para mostrar no serial
187       nodeESP32(); //Chama a funcao para mostrar informações sobre o ESP32
188       notificacoes(); //Chama a funcao para notificar em dois niveis de notificacao
189       IFTTTAlertDHT(); //IFTTT alerta em caso de incidente com temperatura/umidade
190       IFTTTAlertSmoke(); //IFTTT alerta em caso de incidente com gás/fumaça
191       IFFTTTReport(); //IFTTT report a cada 6 horas
192       sendSMS();  //Chama aFuncao para enviar SMS em caso de incidente
193       callNumber(); //Chama aFuncao para ligar em caso de incidente
194     }
195     if (currentMillis - previousMillis3 >= interval60) {
196       //  Serial.println("Entrou no 60s loop... " + String(currentMillis) + " - "
197       + String(previousMillis3));
198       //  Serial.println("Hora3: " + String(timeClient.getFormattedTime()) );
199       previousMillis3 = currentMillis;
200       zabbixFog(); // Envia os dados para a fog (Zabbix)
201       thingspeakCloud(); // Envia os dados para a cloud (ThingSpeak)
202     }
203   }
204   void testeLEDs() {
205     digitalWrite(LEDB, HIGH); //Liga o LED azul
206     delay(2000);
207     digitalWrite(LEDB, LOW); //Desliga o LED azul
208     delay(100);
209     digitalWrite(LEDG, HIGH); //Liga o LED verde
210     delay(2000);
211     digitalWrite(LEDG, LOW); //Desliga o LED verde
212     delay(100);
213     digitalWrite(LEDR, HIGH); //Liga o LED vermelho
214     delay(2000);
215     digitalWrite(LEDR, LOW); //Desliga o LED vermelho
216   }
217   //Funcao para conectar a rede Wifi...
218   void wifiConnect() {
219     int cont = 0;
220     if (WiFi.status() != WL_CONNECTED) {
221       Serial.print("Tentando conectar na rede sem fio ");
222       Serial.println(WIFI_SSID);
223       do {
224         WiFi.begin(WIFI_SSID, WIFI_PASS);
225         cont = cont + 1;
```

```
226         delay(2000);
227       } while (WiFi.status() != WL_CONNECTED && cont != 2);
228     }
229     else {
230       Serial.print("SSID: ");
231       Serial.println(WIFI_SSID);
232       Serial.print("IP: ");
233       Serial.println(WiFi.localIP());
234       Serial.print("MAC: ");
235       Serial.println(WiFi.macAddress()); //retorna o endereço MAC do node.
236       Serial.println("Hostname: " + String(zbxhostname));
237       IP = WiFi.localIP().toString();
238     }
239   }
240   //Bluetooth serial
241   void blueToothSerial() {
242     SerialBT.println("-------------------------------------");
243     SerialBT.println("Hora: " + String(timeClient.getFormattedTime()) );
244     SerialBT.print("SSID: ");
245     SerialBT.println(WIFI_SSID);
246     SerialBT.print("IP: ");
247     SerialBT.println(WiFi.localIP());
248     SerialBT.print("MAC: ");
249     SerialBT.println(WiFi.macAddress());
250     SerialBT.println("Hostname: " + String(zbxhostname));
251     TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
252     MQ2_analog_value = analogRead(MQ2_analog);
253     MQ7_analog_value = analogRead(MQ7_analog);
254     SerialBT.println("Temperatura: " + String(LeituraDHT01.temperature, 0) + String(" ºC")
255                      + " | Umidade: " + String(LeituraDHT01.humidity, 0) + String(" %"));
256     SerialBT.println("MQ2: " + String(MQ2_analog_value)
257                      + " | MQ7: " + String (MQ7_analog_value));
258   }
259   //Funcao para ler a temperatura e a umidade...
260   void DHT1() {
261     TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
262     Serial.println("Temperatura: " + String(LeituraDHT01.temperature, 0) + String(" ºC")
263                    + " | Umidade: " + String(LeituraDHT01.humidity, 0) + String(" %"));
264   }
265   //Funcao responsavel por disparar alerta caso o sensor detectar gas/fumaca...
266   void sensoresMQ() {
267     MQ2_digital_value = digitalRead(MQ2_digital);
268     MQ7_digital_value = digitalRead(MQ7_digital);
269     MQ2_analog_value = analogRead(MQ2_analog);
270     MQ7_analog_value = analogRead(MQ7_analog);
271     Serial.println("MQ2: " + String(MQ2_analog_value)
272                    + " | MQ7: " + String (MQ7_analog_value));
273   }
```

```
274 //Funcao para mostrar informacoes do ESP32...
275 void nodeESP32()  {
276   Serial.print("CPU Temp.: ");
277   Serial.print(temperatureRead()); //returns CPU Temperature in ºC
278   Serial.print(" ºC");
279   Serial.print(" | Hall sensor: ");
280   Serial.println(hallRead()); // reads Hall sensor
281   //Serial.print(" Touch: ");
282   //Serial.println(touchRead(4));// reads touch sensor on pin 4
283   //
284 }
285 //Funcoes para definir os status entre OK, alerta e problema...
286 void BeepLongo() {
287   digitalWrite(BUZZER, HIGH);
288   delay(2000);
289   digitalWrite(BUZZER, LOW);
290 }
291 void BeepMedio() {
292   digitalWrite(BUZZER, HIGH);
293   delay(1000);
294   digitalWrite(BUZZER, LOW);
295 }
296 void BeepCurto() {
297   digitalWrite(BUZZER, HIGH);
298   delay(500);
299   digitalWrite(BUZZER, LOW);
300 }
301 void StatusOK() {
302   digitalWrite(LEDB, LOW); //Desliga o LED azul caso esteja ligado
303   digitalWrite(LEDR, LOW); //Desliga o LED vermelho caso esteja ligado
304   digitalWrite(LEDG, HIGH); //Liga o LED verde (OK)
305   digitalWrite(BUZZER, LOW); //Desliga o Buzzer caso esteja ligado
306   Serial.println("Status: OK!");
307 }
308 void StatusAlert(int Beeps) {
309   digitalWrite(LEDG, LOW); //Desliga o LED verde caso esteja ligado
310   digitalWrite(LEDR, LOW); //Desliga o LED vermelho caso esteja ligado
311   digitalWrite(LEDB, HIGH); //Liga o LED azul (Alert)
312   switch (Beeps) {
313     case 3:
314       BeepMedio();
315       BeepCurto();
316       break;
317     case 4:
318       BeepMedio();
319       break;
320   }
321   Serial.println("Status: Alerta...");
```

```
322 }
323 void StatusProblem(int Beeps) {
324   digitalWrite(LEDB, LOW); //Desliga o LED azul caso esteja ligado
325   digitalWrite(LEDG, LOW); //Desliga o LED verde caso esteja ligado
326   digitalWrite(LEDR, HIGH); //Liga o LED vermelho (Problem)
327   switch (Beeps) {
328     case 1:
329       BeepLongo();
330       BeepMedio();
331       BeepCurto();
332       break;
333     case 2:
334       BeepLongo();
335       BeepCurto();
336       break;
337   }
338   Serial.println("Status: Problema...");
339 }
340 void notificacoes() {
341   MQ2_analog_value = analogRead(MQ2_analog);
342   MQ7_analog_value = analogRead(MQ7_analog);
343   TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
344   if (LeituraDHT01.temperature >= 35 || LeituraDHT01.humidity >= 90
345       || MQ2_analog_value > 1000 || MQ7_analog_value > 1000) {
346     StatusProblem(1);
347   }
348   else if (LeituraDHT01.temperature >= 30 || LeituraDHT01.humidity >= 80
349            || MQ2_analog_value > 800 || MQ7_analog_value > 800) {
350     StatusProblem(2);
351   }
352   else if (LeituraDHT01.temperature >= 29 || LeituraDHT01.humidity >= 70
353            || MQ2_analog_value > 600 || MQ7_analog_value > 600) {
354     StatusAlert(3);
355   }
356   else if (LeituraDHT01.temperature >= 28 || LeituraDHT01.humidity >= 60
357            || MQ2_analog_value > 500 || MQ7_analog_value > 500) {
358     StatusAlert(4);
359   }
360   else {
361     StatusOK();
362   }
363 }
364 //Funcao para mostrar uma mensagem inicial no OLED na inicializacao do ESP32...
365 void SetupOLED() {
366   //Limpa o display
367   display.clear();
368   //Centraliza o texto no display
369   display.setTextAlignment(TEXT_ALIGN_CENTER);
```

```
370   //Seleciona a fonte
371   display.setFont(ArialMT_Plain_10);
372   //Mostra informacao no "boot"
373   display.drawString(63, 10, "TCERN");
374   display.drawString(63, 20, "Modulo Principal");
375   display.drawString(63, 30, "Projeto IMD");
376   display.drawString(63, 40, "IoT DCIM");
377   display.drawString(63, 50, "v.1.0");
378   display.display();
379 }
380 //Funcao para exibir leitura do ambiente no OLED...
381 void ambienteOLED() {
382   TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
383   display.clear();
384   display.setTextAlignment(TEXT_ALIGN_CENTER);
385   display.setFont(ArialMT_Plain_16);
386   display.drawString(63, 10, IP);
387   display.drawString(63, 26, zbxhostname);
388   display.drawString(63, 42, String(LeituraDHT01.temperature, 0) + String(" ºC")
389                       + String(" | ") + String(LeituraDHT01.humidity, 0) + String(" %") );
390   display.display();//Escreve as informações acima no display.
391 }
392 //Funcao para exibir outras informacoes no OLED...
393 void outrosOLED() {
394   //MQ2_analog_value = analogRead(MQ2_analog);
395   //MQ7_analog_value = analogRead(MQ7_analog);
396   TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
397   timeClient.update();
398   String horario = timeClient.getFormattedTime();
399   display.clear();
400   display.setTextAlignment(TEXT_ALIGN_CENTER);
401   display.setFont(ArialMT_Plain_16);
402   display.drawString(63, 10, String(LeituraDHT01.temperature, 0) + String(" ºC")
403                       + String(" | ") + String(LeituraDHT01.humidity, 0) + String(" %") );
404   display.drawString(63, 26, "MQ7: " + String(MQ7_analog_value));
405   display.drawString(63, 42, "MQ2: " + String(MQ2_analog_value));
406   display.display();
407 }
408 //Funcao para enviar os dados para a nuvem ThingSpeak (cloud)...
409 void thingspeakCloud() {
410   if (client.connect(thingspeakHost, 80)) {
411     //MQ2_analog_value = analogRead(MQ2_analog);
412     //MQ7_analog_value = analogRead(MQ7_analog);
413     TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
414     TS_Tempo += Loop2;
415     //Condicao para enviar as informações ao ThingSpeak a cada 2 minutos...
416     if (TS_Tempo >= 120) {
417       String thingspeak = thingspeakAPI;
```

```
418         thingspeak += "&amp;field1=";
419         thingspeak += String(LeituraDHT01.temperature);
420         thingspeak += "&amp;field2=";
421         thingspeak += String(LeituraDHT01.humidity);
422         //thingspeak += "&amp;field3=";
423         //thingspeak += String(MQ2_analog_value);
424         //thingspeak += "&amp;field4=";
425         //thingspeak += String(MQ7_analog_value);
426         client.print("POST /update HTTP/1.1\n");
427         client.print("Host: api.thingspeak.com\n");
428         client.print("Connection: close\n");
429         client.print("X-THINGSPEAKAPIKEY: " + thingspeakAPI + "\n");
430         client.print("Content-Type: application/x-www-form-urlencoded\n");
431         client.print("Content-Length: ");
432         client.print(thingspeak.length());
433         client.print("\n\n");
434         client.print(thingspeak);
435         Serial.println("ThingSpeak:");
436         Serial.println(thingspeak);
437         Serial.print("\n");
438         TS_Tempo = 0;
439       }
440     }
441     //Serial.println("Debug contador ThingSpeak:");
442     //Serial.println(TS_Tempo);
443   }
444   //Funcao para enviar os dados p/ o Zabbix na LAN (fog)...
445   void zabbixFog (void) {
446     Zbx_Tempo += Loop2;
447     //Condicao para enviar as informações ao Zabbix a cada 2 minutos...
448     if (Zbx_Tempo >= 120) {
449       TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
450       String zbxhost = zbxhostname;
451       String itens[] = {"LeituraDHT01.temperature", "LeituraDHT01.humidity"};
452       float valores[] = {LeituraDHT01.temperature, LeituraDHT01.humidity};
453       for (int i = 0; i < (sizeof(valores) / sizeof(int)); i++) {
454         if (client.connect(zbxserver, zbxporta)) {
455           String zabbix = "";
456           zabbix += String("{\"request\":\"sender data\",\"data\":");
457           zabbix += String("[{");
458           zabbix += String("\"host\":") + String("\"") + String (zbxhost)
459                     + String("\"") + String(",");
460           zabbix += String("\"key\":") + String("\"") + String (itens[i])
461                     + String("\"") + String(",");
462           zabbix += String("\"value\":") + String("\"") + String (valores[i])
463                     + String("\"");
464           zabbix += String("}]}");
465           Serial.println("Zabbix:");
```

```
466           Serial.println(zabbix);
467           client.println(zabbix);
468           zabbix = "";
469           String respostazbx = client.readStringUntil('\r');
470           Serial.println(respostazbx);
471           Zbx_Tempo = 0;
472         }
473       }
474     }
475     //Serial.println("Debug contador Zbx:");
476     //Serial.println(Zbx_Tempo);
477   }
478   //Funcao para enviar os dados para o IFTTT (WebHook)...
479   void IFFTTTReport() {
480     TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
481     if (client.connect(IFTTT_host, 80)) {
482       IFTTT_Tempo += Loop;
483       //Condicao para enviar um relatorio a cada 6 horas...
484       if (IFTTT_Tempo >= 21600) {
485         //POST json.
486         String IFTTT_url = "/trigger/";
487         IFTTT_url += IFTTT_EVENT_NAME;
488         IFTTT_url += "/with/key/";
489         IFTTT_url += IFTTT_API_KEY;
490         String IFTTT_jsonObject = String("{\"value1\":\"")
491                                   + String(LeituraDHT01.temperature, 0)
492                                   + "\",\"value2\":\""
493                                   + String(LeituraDHT01.humidity, 0)
494                                   + "\",\"value3\":\"" + String(IP) + "\"}";
495         client.println(String("POST ") + IFTTT_url + " HTTP/1.1");
496         client.println(String("Host: ") + IFTTT_host);
497         client.println("Connection: close\r\nContent-Type: application/json");
498         client.print("Content-Length: ");
499         client.println(IFTTT_jsonObject.length());
500         client.println();
501         client.println(IFTTT_jsonObject);
502         Serial.println("IFTTT:");
503         Serial.println(IFTTT_jsonObject);
504         IFTTT_Tempo = 0;
505       }
506     }
507     //Serial.println("Debug contador IFTTT:");
508     //Serial.println(IFTTT_Tempo);
509   }
510   void IFTTTAlertDHT() {
511     TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
512     if (client.connect(IFTTT_host, 80)) {
513       //Se temperatura estiver 30 graus e/ou umidade 90 por cento, alerta a cada 1 hora...
```

```
514       if (IFTTT_Alerta == 0 && (LeituraDHT01.temperature >= 30
515                                 || LeituraDHT01.humidity >= 90)) {
516         //POST json.
517         String IFTTT_url = "/trigger/";
518         IFTTT_url += IFTTT_EVENT_NAME2;
519         IFTTT_url += "/with/key/";
520         IFTTT_url += IFTTT_API_KEY;
521         String IFTTT_jsonObject = String("{\"value1\":\"")
522                                   + String(LeituraDHT01.temperature, 0)
523                                   + "\",\"value2\":\""
524                                   + String(LeituraDHT01.humidity, 0)
525                                   + "\",\"value3\":\""
526                                   + String(IP) + "\"}";
527         client.println(String("POST ") + IFTTT_url + " HTTP/1.1");
528         client.println(String("Host: ") + IFTTT_host);
529         client.println("Connection: close\r\nContent-Type: application/json");
530         client.print("Content-Length: ");
531         client.println(IFTTT_jsonObject.length());
532         client.println();
533         client.println(IFTTT_jsonObject);
534         Serial.println("IFTTT:");
535         Serial.println(IFTTT_jsonObject);
536         Serial.println("Alerta para o IFTTT enviado...");
537         IFTTT_Alerta = 1;
538       }
539     }
540     if (IFTTT_Alerta == 1) {
541       IFTTT_Intervalo += Loop;
542     }
543     if (IFTTT_Intervalo >= 3600) { // 1 hora
544       IFTTT_Intervalo = 0;
545       IFTTT_Alerta = 0;
546     }
547     //Serial.println("Debug contador IFTTT:");
548     //Serial.println(IFTTT_Alerta);
549     //Serial.println(IFTTT_Intervalo);
550   }
551   void IFTTTAlertSmoke() {
552     if (MQ_Alerta == 0 && MQ2_analog_value > 600 || MQ7_analog_value > 600) {
553       Serial.println("Status: Gas e/ou fumaca detectado...");
554       if (client.connect(IFTTT_host, 80)) {
555         //POST json.
556         String IFTTT_url = "/trigger/";
557         IFTTT_url += IFTTT_EVENT_NAME3;
558         IFTTT_url += "/with/key/";
559         IFTTT_url += IFTTT_API_KEY;
560         String IFTTT_jsonObject = String("{\"value1\":\"") + String(MQ2_analog_value)
561                                   + "\",\"value2\":\"" + String(MQ7_analog_value) + "\"}";
```

```
562         client.println(String("POST ") + IFTTT_url + " HTTP/1.1");
563         client.println(String("Host: ") + IFTTT_host);
564         client.println("Connection: close\r\nContent-Type: application/json");
565         client.print("Content-Length: ");
566         client.println(IFTTT_jsonObject.length());
567         client.println();
568         client.println(IFTTT_jsonObject);
569         Serial.println("IFTTT:");
570         Serial.println(IFTTT_jsonObject);
571       }
572       MQ_Alerta = 1;
573     }
574     if (MQ_Alerta == 1) {
575       MQ_Intervalo += Loop;
576     }
577     if (MQ_Intervalo >= 900) { //15 minutos
578       MQ_Intervalo = 0;
579       MQ_Alerta = 0;
580     }
581     //Serial.println("Debug contador MQ:");
582     //Serial.println(MQ_Alerta);
583     //Serial.println(MQ_Intervalo);
584   }
585   //Funcao para enviar SMS em caso de incidente, precisa de credito.
586   void sendSMS() {
587     TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
588     if (SMS_Alerta == 0 && WiFi.status() != WL_CONNECTED && (LeituraDHT01.temperature >= 30
589         || LeituraDHT01.humidity >= 90)) {
590       Serial.println ("Sending Message");
591       MSIM800L.println("AT+CMGF=1");    //Coloca o modulo GSM em modo texto
592       delay(1000);
593       MSIM800L.println("AT+CMGS=\"" + SMSNumber1 + "\"\r"); //Numero para enviar o SMS
594       delay(1000);
595       String MensagemSMS = (String("Alerta - temperatura ")
596                             + String(LeituraDHT01.temperature, 0)
597                             + String(" C") + String(" e umidade ")
598                             + String(LeituraDHT01.humidity, 0)
599                             + String("% no node ")
600                             + String(zbxhostname) + String(".") );
601       Serial.println(MensagemSMS);
602       MSIM800L.println(MensagemSMS);
603       delay(100);
604       MSIM800L.println((char)26);// ASCII para CTRL+Z
605       delay(1000);
606       Serial.println("SMS enviado com sucesso...");
607       _buffer = _readSerial();
608       SMS_Alerta = 1;
609     }
```

```
610   if (SMS_Alerta == 1) {
611     SMS_Intervalo += Loop;
612   }
613   if (SMS_Intervalo >= 1800) { //30 minutos
614     SMS_Intervalo = 0;
615     SMS_Alerta = 0;
616   }
617   //Serial.println("Debug contador SMS:");
618   //Serial.println(SMS_Alerta);
619   //Serial.println(SMS_Intervalo);
620 }
621 //Funcao para ligar a cobrar para os telefones em caso de incidente.
622 void callNumber() {
623   TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
624   if (Call_Alerta == 0 && LeituraDHT01.temperature >= 25
625       || LeituraDHT01.humidity >= 90) {
626     delay(1000);
627     Serial.println("Ligando...");
628     MSIM800L.print (F("ATD"));
629     MSIM800L.print (CallNumber);
630     MSIM800L.print (F(";\r\n"));
631     MSIM800L.print (F("WAIT=4"));
632     MSIM800L.print (F("ATH")); //Comando para desligar
633     MSIM800L.print (F(";\r\n"));
634     Serial.println("Desligando...");
635     _buffer = _readSerial();
636     Serial.println(_buffer);
637     delay(1000);
638     Call_Alerta = 1;
639   }
640   if (Call_Alerta == 1) {
641     Call_Intervalo += Loop;
642   }
643   if (Call_Intervalo >= 900) { //15 minutos
644     Call_Intervalo = 0;
645     Call_Alerta = 0;
646   }
647   //Serial.println("Debug contador Call:");
648   //Serial.println(Call_Alerta);
649   //Serial.println(Call_Intervalo);
650 }
651 //String para ler o serial do SIM800L, usado pelas funcoes do SMS e Call...
652 String _readSerial() {
653   _timeout = 0;
654   while  (!MSIM800L.available() && _timeout < 12000  )
655   {
656     delay(13);
657     _timeout++;
```

```
658     }
659     if (MSIM800L.available()) {
660       return MSIM800L.readString();
661     }
662   }
663   * /
664   void MQTT_connect() {
665     int8_t ret;
666
667     // Stop if already connected.
668     if (mqtt.connected()) {
669       return;
670     }
671
672     Serial.print(F("Connecting to MQTT... "));
673
674     uint8_t retries = 3;
675     while ((ret = mqtt.connect()) != 0) { // connect will return 0 for connected
676       Serial.println(mqtt.connectErrorString(ret));
677       Serial.println(F("Retrying MQTT connection in 5 seconds..."));
678       mqtt.disconnect();
679       delay(5000);  // wait 5 seconds
680       retries--;
681       if (retries == 0) {
682         // basically die and wait for WDT to reset me
683         while (1);
684       }
685     }
686     Serial.println(F("MQTT Connected!"));
687   }
688   void MQTT_publish() {
689     TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
690     //MQ2_analog_value = analogRead(MQ2_analog);
691     //MQ7_analog_value = analogRead(MQ7_analog);
692     //mqttmq2.publish(MQ2_analog_value);
693     //mqttmq7.publish(MQ7_analog_value);
694     mqttdht1t.publish(LeituraDHT01.temperature);
695     mqttdht1u.publish(LeituraDHT01.humidity);
696     /*if (! mqttcpu.publish(temperatureRead()) && mqttmq2.publish(MQ2_analog_value)
697        && mqttmq7.publish(MQ7_analog_value) ) {
698           Serial.println(F("MQTT Adafruit.IO: Publish Failed."));
699       } else {
700           Serial.println(F("MQTT Adafruit.IO: Publish Success!"));
701           delay(500);
702       }
703         delay(10000);*/
704   }
705   void MQTT_Subscribe () {
```

```
706   mqtt.subscribe(&mqttled);
707   Adafruit_MQTT_Subscribe *subscription;
708   while ((subscription = mqtt.readSubscription(5000))) {
709     // Check if its the onoff button feed
710     if (subscription == &mqttled) {
711       Serial.print(F("On-Off button: "));
712       Serial.println((char *)mqttled.lastread);
713
714       if (strcmp((char *)mqttled.lastread, "ON") == 0) {
715         digitalWrite(LEDR, LOW);
716       }
717       if (strcmp((char *)mqttled.lastread, "OFF") == 0) {
718         digitalWrite(LEDR, HIGH);
719       }
720     }
721   }
722 }
```

## B.2 Código do ponto de orvalho

```
1  /*****************************************************************/
2  /* Example how to read DHT sensors from an ESP32 using multi- */
3  /* tasking.                                                      */
4  /* This example depends on the ESP32Ticker library to wake up */
5  /* the task every 20 seconds                                     */
6  /* Please install Ticker-esp32 library first                     */
7  /* bertmelis/Ticker-esp32                                        */
8  /* https://github.com/bertmelis/Ticker-esp32                     */
9  /*****************************************************************/
10
11 DHTesp dht01;
12
13 float heatIndex;
14 float dewPoint;
15
16 void tempTask(void *pvParameters);
17 bool getTemperature();
18 void triggerGetTemp();
19
20 /** Task handle for the light value read task */
21 TaskHandle_t tempTaskHandle = NULL;
22 /** Ticker for temperature reading */
23 Ticker tempTicker;
24 /** Comfort profile */
25 ComfortState cf;
```

```
26  /** Flag if task should run */
27  bool tasksEnabled = false;
28  /** Pin number for DHT11 data pin */
29  int dhtPin = 26;
30
31  /**
32   * initTemp
33   * Setup DHT library
34   * Setup task and timer for repeated measurement
35   * @return bool
36   *    true if task and timer are started
37   *    false if task or timer couldn't be started
38   */
39  bool initTemp() {
40    byte resultValue = 0;
41    // Initialize temperature sensor
42          dht01.setup(dhtPin, DHTesp::DHT11);
43          Serial.println("DHT initiated");
44
45    // Start task to get temperature
46          xTaskCreatePinnedToCore(
47                          tempTask,                       /* Function to implement the task */
48                          "tempTask ",                    /* Name of the task */
49                          4000,                           /* Stack size in words */
50                          NULL,                           /* Task input parameter */
51                          5,                              /* Priority of the task */
52                          &tempTaskHandle,                /* Task handle. */
53                          1);                             /* Core where the task should run */
54
55    if (tempTaskHandle == NULL) {
56      Serial.println("Failed to start task for temperature update");
57      return false;
58    } else {
59      // Start update of environment data every 20 seconds
60      tempTicker.attach(20, triggerGetTemp);
61    }
62    return true;
63  }
64
65  /**
66   * triggerGetTemp
67   * Sets flag dhtUpdated to true for handling in loop()
68   * called by Ticker getTempTimer
69   */
70  void triggerGetTemp() {
71    if (tempTaskHandle != NULL) {
72            xTaskResumeFromISR(tempTaskHandle);
73    }
```

```
74  }
75
76  /**
77   * Task to reads temperature from DHT11 sensor
78   * @param pvParameters
79   *    pointer to task parameters
80   */
81  void tempTask(void *pvParameters) {
82          Serial.println("tempTask loop started");
83          while (1) // tempTask loop
84    {
85      if (tasksEnabled) {
86        // Get temperature values
87                          getTemperature();
88                  }
89      // Got sleep again
90                  vTaskSuspend(NULL);
91          }
92  }
93
94  /**
95   * getTemperature
96   * Reads temperature from DHT11 sensor
97   * @return bool
98   *    true if temperature could be aquired
99   *    false if aquisition failed
100  */
101 bool getTemperature() {
102         // Reading temperature for humidity takes about 250 milliseconds!
103         // Sensor readings may also be up to 2 seconds 'old' (it's a very slow sensor)
104   TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
105         // Check if any reads failed and exit early (to try again).
106         if (dht01.getStatus() != 0) {
107                 Serial.println("DHT11 error status: " + String(dht01.getStatusString()));
108                 return false;
109         }
110
111         heatIndex = dht01.computeHeatIndex(LeituraDHT01.temperature, LeituraDHT01.humidity);
112   dewPoint = dht01.computeDewPoint(LeituraDHT01.temperature, LeituraDHT01.humidity);
113   float cr = dht01.getComfortRatio(cf, LeituraDHT01.temperature, LeituraDHT01.humidity);
114
115   String comfortStatus;
116   switch(cf) {
117     case Comfort_OK:
118       comfortStatus = "OK";
119       break;
120     case Comfort_TooHot:
121       comfortStatus = "TooHot";
```

```
122         break;
123       case Comfort_TooCold:
124         comfortStatus = "TooCold";
125         break;
126       case Comfort_TooDry:
127         comfortStatus = "TooDry";
128         break;
129       case Comfort_TooHumid:
130         comfortStatus = "TooHumid";
131         break;
132       case Comfort_HotAndHumid:
133         comfortStatus = "HotAndHumid";
134         break;
135       case Comfort_HotAndDry:
136         comfortStatus = "HotAndDry";
137         break;
138       case Comfort_ColdAndHumid:
139         comfortStatus = "ColdAndHumid";
140         break;
141       case Comfort_ColdAndDry:
142         comfortStatus = "ColdAndDry";
143         break;
144       default:
145         comfortStatus = "Unknown:";
146         break;
147     };
148 
149     Serial.println(" T:" + String(LeituraDHT01.temperature) + " H:" + String(LeituraDHT01.humidity)
150           console.println(" T:" + String(LeituraDHT01.temperature) + " H:" + String(LeituraDHT01.hum
151           return true;
152 }
```

## B.3 Código fonte do Módulo 02, 03, 04

```
1  //Inclusão das bibliotecas
2  #include <DHTesp.h> //Carrega a biblioteca usada pelo sensor DHT.
3  #include <WiFi.h> //Carrega a biblioteca da rede sem fio.
4  //
5  //Definições dos pinos no ESP32
6  #define IR_TX_LED 25 //Cria um alias para o LED IR no pino 25 do ESP32.
7  #define LEDR 12 //Cria um alias para o LED Vermelho no pino 12 do ESP32.
8  #define LEDG 14 //Cria um alias para o LED Verde no pino 14 do ESP32.
9  #define LEDB 32 //Cria um alias para o LED Azul no pino 32 do ESP32.
10 #define BUZZER 33 //Cria um alias para o Buzzer no pino 33 do ESP32.
11 #define DHT01data 26 //Cria um alias para o módulo DHT1 no pino 26 do ESP32.
```

```
12 #define DHT02data 27 //Cria um alias para o módulo DHT2 no pino 27 do ESP32.
13 //
14 DHTesp dht01; //Cria um objeto do tipo DTHesp para o DHT1 no IoT ou frente do rack.
15 DHTesp dht02; //Cria um objeto do tipo DTHesp para o DHT2 na boca do ar ou traseira do rack.
16 //
17 //Define as variáveis e APIs do FTTT
18 String IFTTT_API_KEY = "XXXX"; //Define a API do IFTTT
19 String IFTTT_EVENT_NAME =  "ReportESP32"; // Define o evento do IFTTT
20 const char* IFTTT_host = "maker.ifttt.com";
21 //
22 //Define as variáveis e APIs do ThingSpeak
23 //String thingspeakAPI = "XXXX"; //API do Módulo IoT 02.
24 //String thingspeakAPI = "XXXX"; //API do Módulo IoT 03.
25 //String thingspeakAPI = "XXXX"; //API do Módulo IoT 04.
26 String thingspeakAPI = "XXXX"; //API do Módulo IoT 05.
27 const char* thingspeakHost = "api.thingspeak.com"; //URL da API do ThingSpeak.
28 //
29 //Define as variáveis da conexão sem fio (wifi)
30 #define WIFI_SSID "XXXX" //Nome da rede (SSID)
31 #define WIFI_PASS "XXXX" // senha de rede sem fio
32 String IP;
33 WiFiClient client; //Inicializa o Wifi...
34 //
35 //Define as variaveis e dados do Zabbix (trapper)
36 float temperature;
37 float humidity;
38 const char* zbxserver = "0.0.0.0"; //IP do Zabbix.
39 const char* zbxhostname = "nodeiot01"; //Host no Zabbix.
40 int zbxporta = 10051;
41 //
42 //Outros
43 // Contador de tempo para a funcao do IFTTT...
44 int IFTTT_Tempo = 0;
45 int IFTTT_Intervalo = 0;
46 int IFTTT_Alerta = 0;
47 int TS_Tempo = 0;
48 int Zbx_Tempo = 0;
49 int IR_Tempo = 0;
50 int Loop = 30;
51 int Loop2 = 60;
52 unsigned long previousMillis = 0;
53 unsigned long previousMillis2 = 0;
54 unsigned long previousMillis3 = 0;
55 const long interval10 = 10000;
56 const long interval30 = 30000;
57 const long interval60 = 60000;
58 //
59 void setup() {
```

```
60    // put your setup code here, to run once:
61    Serial.begin(115200); //Inicia o terminal para a leitura via console.
62    Serial.println("Inicializando o ESP32...");
63    pinMode(BUZZER, OUTPUT);  //Define que o Buzzer sera um pino de saída.
64    pinMode(LEDR, OUTPUT);  //Define que o pino do LEDR sera um pino de saída.
65    pinMode(LEDG, OUTPUT);  //Define que o pino do LEDG sera um pino de saída.
66    pinMode(LEDB, OUTPUT);  //Define que o pino do LEDB sera um pino de saída.
67    digitalWrite(IR_TX_LED, LOW); //Define que o pino está desligado.
68    digitalWrite(BUZZER, LOW);//Define que o pino está desligado.
69    digitalWrite(LEDR, LOW);//Define que o pino está desligado.
70    digitalWrite(LEDG, LOW);//Define que o pino está desligado.
71    digitalWrite(LEDB, LOW);//Define que o pino está desligado.
72    pinMode(DHT01data, INPUT);  //Define que o pindo do sensor DHT será um pino de entrada.
73    pinMode(DHT02data, INPUT);  //Define que o pindo do sensor DHT será um pino de entrada.
74    dht01.setup(DHT01data, DHTesp::DHT11);  //Inicialização do sensor DHT01.
75    dht02.setup(DHT02data, DHTesp::DHT11);  //Inicialização do sensor DHT02.
76    testeLEDs();
77    BeepLongo();
78    BeepMedio();
79    BeepCurto();
80    wifiConnect();  //Conecta ao WiFi
81  }
82  void loop() {
83    unsigned long currentMillis = millis();
84    // Chama as seguintes funcoes:
85    if (currentMillis - previousMillis >= interval10) {
86      previousMillis = currentMillis;
87      notificacoes();
88      //Serial.println("Entrou no loop de 10s");
89    }
90    if (currentMillis - previousMillis2 >= interval30) {
91      previousMillis2 = currentMillis;
92      IFFTTTReport(); //IFTTT report a cada 6 horas
93      IFTTTAlert(); //IFTTT alerta em caso de incidente
94      //Serial.println("Entrou no loop de 30s");
95    }
96    if (currentMillis - previousMillis3 >= interval60) {
97      previousMillis3 = currentMillis;
98      ThingSpeakCloud();  //ThingSpeak (cloud)
99      ZabbixFog();  //Zabbix (Fog)
100     wifiConnect();  //Conecta ao WiFi
101     //Serial.println("Entrou no loop de 60s");
102   }
103 }
104 void testeLEDs() {
105   digitalWrite(LEDB, HIGH); //Liga o LED azul.
106   delay(2000);
107   digitalWrite(LEDB, LOW); //Desliga o LED azul.
```

```
108     delay(100);
109     digitalWrite(LEDG, HIGH); //Liga o LED verde.
110     delay(2000);
111     digitalWrite(LEDG, LOW); //Desliga o LED verde.
112     delay(100);
113     digitalWrite(LEDR, HIGH); //Liga o LED vermelho.
114     delay(2000);
115     digitalWrite(LEDR, LOW); //Desliga o LED vermelho.
116   }
117   //Função para conectar a rede Wifi...
118   void wifiConnect() {
119     int cont = 0;
120     if (WiFi.status() != WL_CONNECTED) {
121       Serial.print("Tentando conectar na rede sem fio ");
122       Serial.println(WIFI_SSID);
123       do {
124         WiFi.begin(WIFI_SSID, WIFI_PASS);
125         cont = cont + 1;
126         delay(2000);
127       } while (WiFi.status() != WL_CONNECTED && cont != 2);
128     }
129     else {
130       Serial.print("SSID: ");
131       Serial.println(WIFI_SSID);
132       Serial.print("IP: ");
133       Serial.println(WiFi.localIP());
134       Serial.print("MAC: ");
135       Serial.println(WiFi.macAddress()); //retorna o endereço MAC do node.
136       Serial.println("Hostname: " + String(zbxhostname));
137       IP = WiFi.localIP().toString();
138     }
139   }
140   void BeepLongo() {
141     digitalWrite(BUZZER, HIGH);
142     delay(2000);
143     digitalWrite(BUZZER, LOW);
144   }
145   void BeepMedio() {
146     digitalWrite(BUZZER, HIGH);
147     delay(1000);
148     digitalWrite(BUZZER, LOW);
149   }
150   void BeepCurto() {
151     digitalWrite(BUZZER, HIGH);
152     delay(500);
153     digitalWrite(BUZZER, LOW);
154   }
155   void StatusOK() {
```

```
156   digitalWrite(LEDB, LOW); //Desliga o LED azul caso esteja ligado.
157   digitalWrite(LEDR, LOW); //Desliga o LED vermelho caso esteja ligado.
158   digitalWrite(LEDG, HIGH); //Liga o LED verde (OK).
159   digitalWrite(BUZZER, LOW); //Desliga o Buzzer caso esteja ligado.
160   Serial.println("Status: OK!");
161 }
162 void StatusAlert(int Beeps) {
163   digitalWrite(LEDG, LOW); //Desliga o LED verde caso esteja ligado.
164   digitalWrite(LEDR, LOW); //Desliga o LED vermelho caso esteja ligado.
165   digitalWrite(LEDB, HIGH); //Liga o LED azul (Alert).
166   switch (Beeps) {
167     case 3:
168       BeepMedio();
169       BeepCurto();
170       break;
171     case 4:
172       BeepMedio();
173       break;
174   }
175   Serial.println("Status: Alerta...");
176 }
177 void StatusProblem(int Beeps) {
178   digitalWrite(LEDB, LOW); //Desliga o LED azul caso esteja ligado.
179   digitalWrite(LEDG, LOW); //Desliga o LED verde caso esteja ligado.
180   digitalWrite(LEDR, HIGH); //Liga o LED vermelho (Problem).
181   switch (Beeps) {
182     case 1:
183       BeepLongo();
184       BeepMedio();
185       BeepCurto();
186       break;
187     case 2:
188       BeepLongo();
189       BeepCurto();
190       break;
191   }
192   Serial.println("Status: Problema...");
193 }
194 void notificacoes() {
195   TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
196   TempAndHumidity LeituraDHT02 = dht02.getTempAndHumidity();
197   Serial.println("DHT1 - Temperatura: " + String(LeituraDHT01.temperature, 0) + String(" ºC")
198                  + " | Umidade: " + String(LeituraDHT01.humidity, 0) + String(" %"));
199   Serial.println("DHT2 - Temperatura: " + String(LeituraDHT02.temperature, 0) + String(" ºC")
200                  + " | Umidade: " + String(LeituraDHT02.humidity, 0) + String(" %"));
201   if (LeituraDHT01.temperature >= 35 || LeituraDHT01.humidity >= 90) {
202     StatusProblem(1);
203   }
```

```
204     else if (LeituraDHT01.temperature >= 32 || LeituraDHT01.humidity >= 80) {
205       StatusProblem(2);
206     }
207     else if (LeituraDHT01.temperature >= 31 || LeituraDHT01.humidity >= 70) {
208       StatusAlert(3);
209     }
210     else if (LeituraDHT01.temperature >= 30 || LeituraDHT01.humidity >= 60) {
211       StatusAlert(4);
212     }
213     else {
214       StatusOK();
215     }
216   }
217   //Funcao para enviar os dados para a nuvem ThingSpeak (cloud)...
218   void ThingSpeakCloud() {
219     TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
220     TempAndHumidity LeituraDHT02 = dht02.getTempAndHumidity();
221     if (client.connect(thingspeakHost, 80)) {
222       TS_Tempo += Loop2;
223       //Condicao para enviar as informações ao ThingSpeak a cada 4 minutos...
224       if (TS_Tempo >= 240) {
225         String thingspeak = thingspeakAPI;
226         thingspeak += "&amp;field1=";
227         thingspeak += String(LeituraDHT01.temperature);
228         thingspeak += "&amp;field2=";
229         thingspeak += String(LeituraDHT01.humidity);
230         thingspeak += "&amp;field3=";
231         thingspeak += String(LeituraDHT02.temperature);
232         thingspeak += "&amp;field4=";
233         thingspeak += String(LeituraDHT02.humidity);
234         thingspeak += "\r\n\r\n";
235         client.print("POST /update HTTP/1.1\n");
236         client.print("Host: api.thingspeak.com\n");
237         client.print("Connection: close\n");
238         client.print("X-THINGSPEAKAPIKEY: " + thingspeakAPI + "\n");
239         client.print("Content-Type: application/x-www-form-urlencoded\n");
240         client.print("Content-Length: ");
241         client.print(thingspeak.length());
242         client.print("\n\n");
243         client.print(thingspeak);
244         Serial.println("ThingSpeak:");
245         Serial.println(thingspeak);
246         Serial.print("\n");
247         TS_Tempo = 0;
248       }
249     }
250     else {
251       Serial.println("Dados não enviados para a cloud (ThingSpeak).");
```

```
252     }
253     Serial.println("Debug contador ThingSpeak:");
254     Serial.println(TS_Tempo);
255   }
256   //Funcao para enviar os dados p/ o Zabbix na LAN (fog)...
257   void ZabbixFog (void) {
258     Zbx_Tempo += Loop2;
259     //Condicao para enviar as informações ao Zabbix a cada 3 minutos...
260     if (Zbx_Tempo >= 180) {
261       TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
262       TempAndHumidity LeituraDHT02 = dht02.getTempAndHumidity();
263       String zbxhost = zbxhostname;
264       String itens[] = {"LeituraDHT01.temperature", "LeituraDHT01.humidity",
265                         "LeituraDHT02.temperature", "LeituraDHT02.humidity"
266                        };
267       float valores[] = {LeituraDHT01.temperature, LeituraDHT01.humidity,
268                          LeituraDHT02.temperature, LeituraDHT02.humidity
269                         };
270       for (int i = 0; i < (sizeof(valores) / sizeof(int)); i++) {
271         if (client.connect(zbxserver, zbxporta)) {
272           String zabbix = "";
273           zabbix += String("{\"request\":\"sender data\",\"data\":");
274           zabbix += String("[{");
275           zabbix += String("\"host\":") + String("\"") + String (zbxhost)
276           + String("\"") + String(",");
277           zabbix += String("\"key\":") + String("\"") + String (itens[i])
278           + String("\"") + String(",");
279           zabbix += String("\"value\":") + String("\"") + String (valores[i])
280           + String("\"");
281           zabbix += String("}]}");
282           Serial.println("Zabbix:");
283           Serial.println(zabbix);
284           client.println(zabbix);
285           zabbix = "";
286           String respostazbx = client.readStringUntil('\r');
287           Serial.println(respostazbx);
288           Zbx_Tempo = 0;
289         }
290       }
291     }
292     Serial.println("Debug contador Zbx:");
293     Serial.println(Zbx_Tempo);
294   }
295   //Envia os dados para o IFTTT (WebHook) para enviar p/ email, Twitter e Slack ou outro...
296   void IFFTTTReport() {
297     TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
298     TempAndHumidity LeituraDHT02 = dht02.getTempAndHumidity();
299     if (client.connect(IFTTT_host, 80)) {
```

```
300       IFTTT_Tempo += Loop;
301       //Condicao para enviar um relatorio a cada 6 horas...
302       if (IFTTT_Tempo >= 21600) {
303         //POST json.
304         String IFTTT_url = "/trigger/";
305         IFTTT_url += IFTTT_EVENT_NAME;
306         IFTTT_url += "/with/key/";
307         IFTTT_url += IFTTT_API_KEY;
308         String IFTTT_jsonObject = String("{\"value1\":\"") + String(LeituraDHT01.temperature, 0)
309                                   + "\",\"value2\":\"" + String(LeituraDHT01.humidity, 0)
310                                   + "\",\"value3\":\"" + String(IP) + "\"}";
311         client.println(String("POST ") + IFTTT_url + " HTTP/1.1");
312         client.println(String("Host: ") + IFTTT_host);
313         client.println("Connection: close\r\nContent-Type: application/json");
314         client.print("Content-Length: ");
315         client.println(IFTTT_jsonObject.length());
316         client.println();
317         client.println(IFTTT_jsonObject);
318         Serial.println("IFTTT:");
319         Serial.println(IFTTT_jsonObject);
320         IFTTT_Tempo = 0;
321       }
322     }
323     Serial.println("Debug contador IFTTT:");
324     Serial.println(IFTTT_Tempo);
325   }
326   void IFTTTAlert() {
327     TempAndHumidity LeituraDHT01 = dht01.getTempAndHumidity();
328     if (client.connect(IFTTT_host, 80)) {
329       if (IFTTT_Alerta == 0 && (LeituraDHT01.temperature >= 28 || LeituraDHT01.humidity >= 90)) {
330         //POST json.
331         String IFTTT_url = "/trigger/";
332         IFTTT_url += IFTTT_EVENT_NAME;
333         IFTTT_url += "/with/key/";
334         IFTTT_url += IFTTT_API_KEY;
335         String IFTTT_jsonObject = String("{\"value1\":\"") + String(LeituraDHT01.temperature, 0)
336                                   + "\",\"value2\":\"" + String(LeituraDHT01.humidity, 0)
337                                   + "\",\"value3\":\"" + String(IP) + "\"}";
338         client.println(String("POST ") + IFTTT_url + " HTTP/1.1");
339         client.println(String("Host: ") + IFTTT_host);
340         client.println("Connection: close\r\nContent-Type: application/json");
341         client.print("Content-Length: ");
342         client.println(IFTTT_jsonObject.length());
343         client.println();
344         client.println(IFTTT_jsonObject);
345         Serial.println("IFTTT:");
346         Serial.println(IFTTT_jsonObject);
347         IFTTT_Alerta = 1;
```

```
348       }
349     }
350     if (IFTTT_Alerta == 1) {
351       IFTTT_Intervalo += Loop;
352     }
353     if (IFTTT_Intervalo >= 1800) { //30 minutos
354       IFTTT_Intervalo = 0;
355       IFTTT_Alerta = 0;
356     }
357     Serial.println("Debug contador IFTTT:");
358     Serial.println(IFTTT_Alerta);
359     Serial.println(IFTTT_Intervalo);
360   }
```

# APÊNDICE C – Life Cycle Canvas | TAP - Termo de Abertura do Projeto

| Projeto: | | Ciclo de vida: | | Data: |
|---|---|---|---|---|
| Projeto IdC: Monitoramento do ambiente do datacenter do TCE/RN | | Iniciação (IN) | | 25/06/2018 |
| | | Artefato: | | Local: |
| | | Termo de Abertura do Projeto (TAP) | | TCE/RN |
| **Justificativas** | **Produto** | **Partes interessadas** | **Premissas** | **Riscos** |
| Ausência de um sistema dedicado de monitoramento do ambiente da sala do Data Center do TCE/RN | Sistema de monitoramento da sala do datacenter, com detecção de fumaça e monitoramento da temperatura e umidade com envio de aletas por telefonia móvel, baseado em IdC (IoT) | PI1: Setor de Infraestrutura<br>PI2: Alexandre (Diretor)<br>PI3: IMD | PR1: Placas e dispostivos<br>PR2: Ferramentas<br>PR3: Computador<br>PR4: Plataforma de desenvolvimento<br>PR5: Conexão a Internet | RI1: Falta de crédito no SIM Card<br>RI2: Extravio das encomendas<br>RI3: Danos dos dispotivos |
| **Objetivos** | **Requisitos** | **Equipe** | **Entregas** | **Tempo** |
| Criar um sistema de monitoramento da temperatura e umidade da sala do Data Center, no 9o. andar, com envio de alertas, utilizando a rede celular e link principal de Internet do TCE/RN | R1: Monitorar a temperatura<br>R2: Enviar alertas por SMS e/ou chamadas telefônicas | E1: Davi<br>E2: Daniel | EN1: Sensor de temperatura e umidade (DHT) configurado para leitura via terminal<br>EN2: Sensor de temperatura e umidade (DHT) configurado para leitura via rede<br>EN3: Rede de monitoramento DHT acessada via rede<br>EN4: Detector de fumaça e gases monitorados acessado via rede<br>EN5: Sistema de envio de mensagens com GSM | EN1: 2 Semanas<br>EN2: 2 Semanas<br>EN3.1: 2 Semanas (implatação)<br>EN3.2: 1 Semanas (validação)<br>EN4: 2 Semanas<br>EN5: 2 Semanas |
| **Benefícios** | **Restrições** | **Comunicações** | **Aquisições** | **Custo** |
| B1: Maior segurança do Data Center e anexos<br>B2: Redundância da avaliação da temperatura dos equipamentos<br>B3: Maior agilidade nas medidas reativas em casos de sinistros | RT1: Falta de conhecimento da solução na tecnologia backend<br>RT2: Não conseguir adquirir as placas e sensores | C1: E-mail<br>C2: Informal (Whatsapp)<br>C3: Trello | AQ1: Placas IoT (ESP)<br>AQ2: Sensor DHT22 ou DHT11<br>AQ3: Sensor MQ2<br>AQ4: CI MM74HC151<br>AQ5: Placa GSM (A6 ou A7)<br>AQ6: Placa de ckt universal<br>AQ7: Ferro de Solda 60W/220V<br>AQ8: Solda 0.5<br>AQ9: Miniretífica | R$800,00 |
| **Lições aprendidas:** | **Versão:** | | **Patrocinador:** | **Cliente:** |
| | **1.0** | | **Alexandre** | **TCE/RN** |

# APÊNDICE D – Cronograma do Projeto

| ID | Task Mode | Task Name | Duration | Start | Finish | Predec |
|---|---|---|---|---|---|---|
| 1 | | **Projeto IoT Datacenter** | **119,67 days** | **21/06/201** | **23/11/2018** | |
| 2 | | **Workshop** | **2,33 days** | **23/07/201** | **24/07/2018** | |
| 3 | | Apresentação PBL3 (início do projeto) | 1 day | 23/07/201 | 23/07/2018 | |
| 4 | | Ajuste no cronograma | 1 day | 24/07/201 | 24/07/2018 | |
| 5 | | **Iniciação** | **5,67 days** | **21/06/201** | **28/06/2018** | |
| 6 | | Reunião de definições | 1 day | 21/06/201 | 21/06/2018 | |
| 7 | | Criar LCC | 2 days | 25/06/201 | 26/06/2018 | |
| 8 | | Estabelecer cronograma | 1 day | 26/06/201 | 28/06/2018 | 7 |
| 9 | | **Planejamento** | **22 days** | **28/06/201** | **26/07/2018** | **5** |
| 10 | | Pesquisa | 1 day | 28/06/201 | 28/06/2018 | 8 |
| 11 | | Cotação inicial da solução | 1 day | 29/06/201 | 29/06/2018 | 10 |
| 12 | | Adquirir o módulo principal e acessórios | 20 days | 29/06/201 | 26/07/2018 | 11 |
| 13 | | **Execução e Monitoramento** | **75 days** | **26/07/201** | **01/11/2018** | **9** |
| 14 | | EN1: Sensor de temperatura e umidade (DHT) configurado para leitura via terminal | 15 days | 26/07/201 | 14/08/2018 | 12 |
| 15 | | EN2: Sensor de temperatura e umidade (DHT) configurado para leitura via rede | 15 days | 16/08/201 | 04/09/2018 | 14 |
| 16 | | EN3: Rede de monitoramento DHT acessada via rede | 15 days | 04/09/201 | 24/09/2018 | 15 |
| 17 | | EN4: Detector de fumaça e gases monitorados acessado via rede | 15 days | 24/09/201 | 12/10/2018 | 16 |
| 18 | | EN5: Sistema de envio de mensagens com GSM | 15 days | 12/10/201 | 01/11/2018 | 17 |
| 19 | | **Encerramento** | **17 days** | **02/11/201** | **23/11/2018** | **13** |
| 20 | | Colocar a solução em produção | 1 day | 02/11/201 | 02/11/2018 | 18 |
| 21 | | Validação da solução em produção | 15 days | 02/11/201 | 22/11/2018 | 20 |
| 22 | | Formalização da entrega da solução em produção | 1 day | 23/11/201 | 23/11/2018 | 21 |

Project: Projeto IoT Datacenter
Date: 06/09/2018

| | | | |
|---|---|---|---|
| Task | | Inactive Summary | |
| Split | | Manual Task | |
| Milestone | | Duration-only | |
| Summary | | Manual Summary Rollup | |
| Project Summary | | Manual Summary | |
| External Tasks | | Start-only | |
| External Milestone | | Finish-only | |
| Inactive Task | | Deadline | |
| Inactive Milestone | | Progress | |

# APÊNDICE E – Roteiro de Instalação do Zabbix

**Procedimento Operacional Padrão**

**para instalação do Zabbix Server 3.4 no Debian 9.5 para o monitoramento dos node IoT:**

**Hostname**: prd-zabbix03
**Endereço IP(**IP_SERVIDOR):xxxxx
**Usuário**: sutemplate
**Senha**: xxxxx
**CNAMES**: zbx

xxxxx

**Roteiro de instalação:**

Adição do repositório, atualização do sistema e instalação do Zabbix e dependências.

su
cd /tmp
wget
http://repo.zabbix.com/zabbix/3.4/debian/pool/main/z/zabbix-release/zabbix-release_3.4-1+stretch_all.deb
dpkg -i  zabbix-release_3.4-1+stretch_all.deb
apt-get update
apt-get upgrade -y
apt-get install zabbix-server-mysql zabbix-frontend-php zabbix-agent zabbix-get -y

Teste da instalação

Reiniciar o servidor web
#systemctl reload apache2

Verificação via browser

Em um host da mesma rede digitar no navegador:

IP_SERVIDOR/zabbix

Deve aparecer:


**Observação**: Não é para fazer a configuração via web browser ainda, pois precisa finalizar a configuração em um arquivo do apache e criar o banco para o servidor.

Ajustes no Servidor Web e Criação do Banco para o Zabbix

Determinação do fuso horário no zabbix.conf

Editar o “time zone” para America/Fortaleza na linha “ php_value date.timezone ” no bloco mod_php7.c do arquivo:

#vi /etc/apache2/conf-available/zabbix.conf

Reiniciar o apache

#systemctl restart apache2

Configuração inicial do SGBD (MariaDB)

A configuração inicial do banco foi feita usando o script mysql_secure_installation

Senha do root do banco atribuída para o mysql (SENHA_ROOT_DB): xxxxxx

No script foram desabilitados o login anonymous, o acesso remoto do root e recarregados os privilégios.

Criação da tabela, do usuário e atribuição dos privilégios.

Entrar no mysql:

mysql -u root -p

Dentro do mysql:

create database zabbix_db character set utf8 collate utf8_bin;

create user 'zabbix_usr'@'localhost' identified by 'SENHA_ROOT_DB';

grant all privileges on zabbix_db.* to zabbix_usr@localhost identified by 'SENHA_ROOT_DB';

quit;

Popular o banco do zabbix

Fora do mysql (MariaDB) digitar como root:

#zcat /usr/share/doc/zabbix-server-mysql/create.sql.gz | mysql -u zabbix_usr -pSENHA_ROOT_DB zabbix_db

**Observação**: após o parâmetro '-p' não tem espaços e caso haja espaço na senha ela deve ser colocada entre aspas simples;

Configuração do servidor Zabbix

Edição do arquivo Zabbix_server.conf

#vi /etc/zabbix/zabbix_server.conf

Dentro do arquivo acima deve configurar nos itens: DBName; DBUser e DBPassword com as informações do banco criado anteriormente, isto é:

DBName=zabbix_db
DBUser=zabbix_usr
DBPassword=SENHA_ROOT_DB

Reinicialização do Zabbix e Carregamento no Boot

#systemctl restart zabbix-server
#systemctl enable zabbix-server

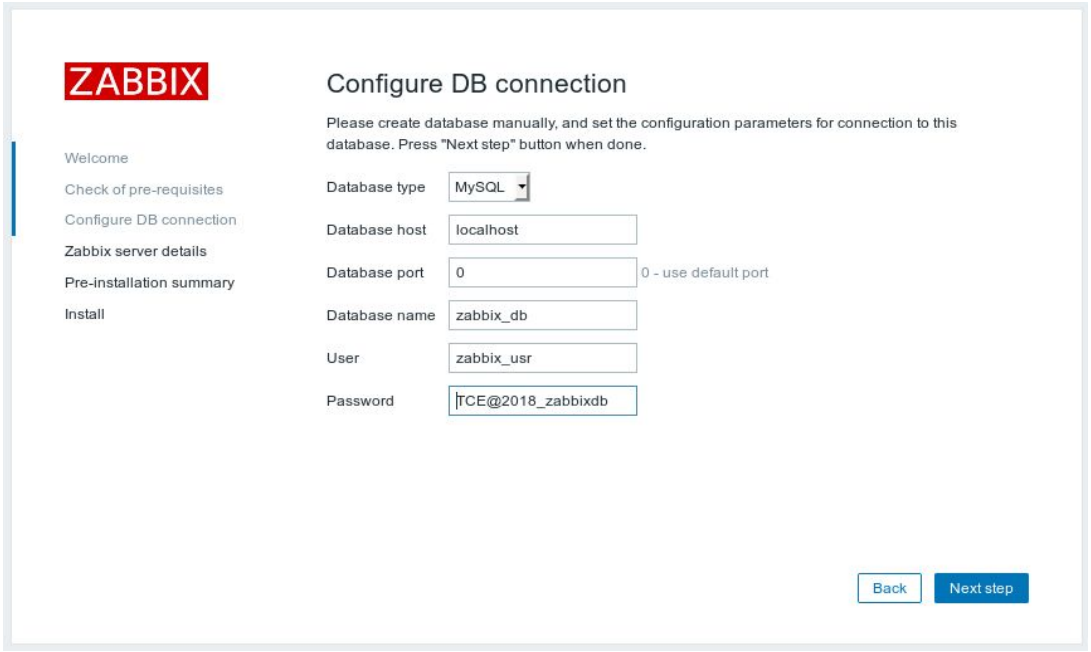
Finalizando a configuração via browser

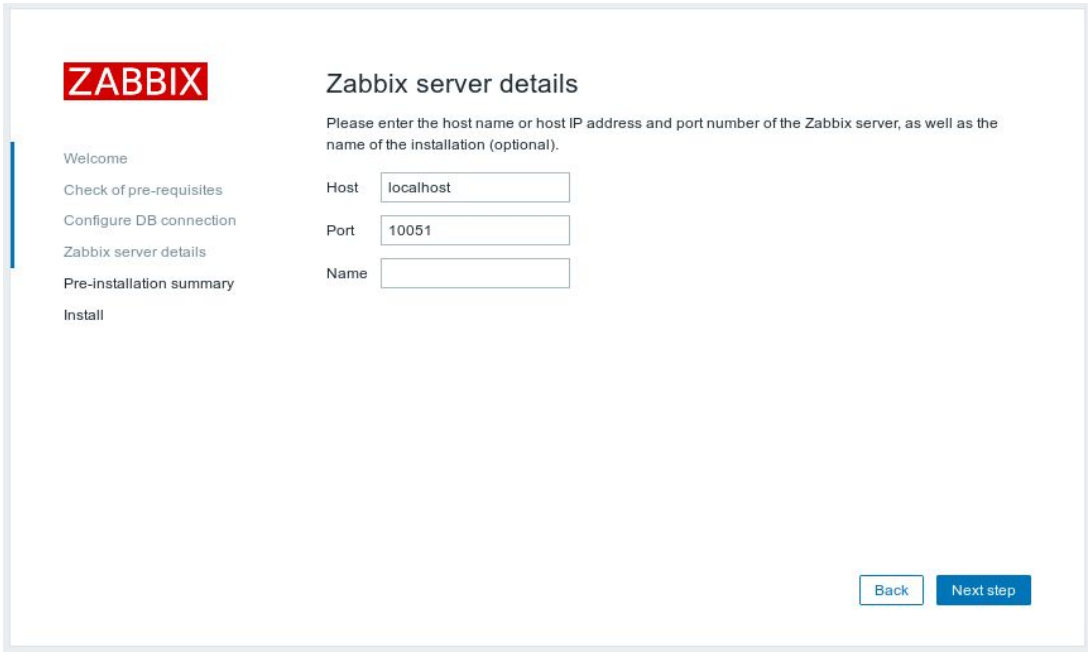
Acessar o endereço seguido de '/zabbix', neste caso:

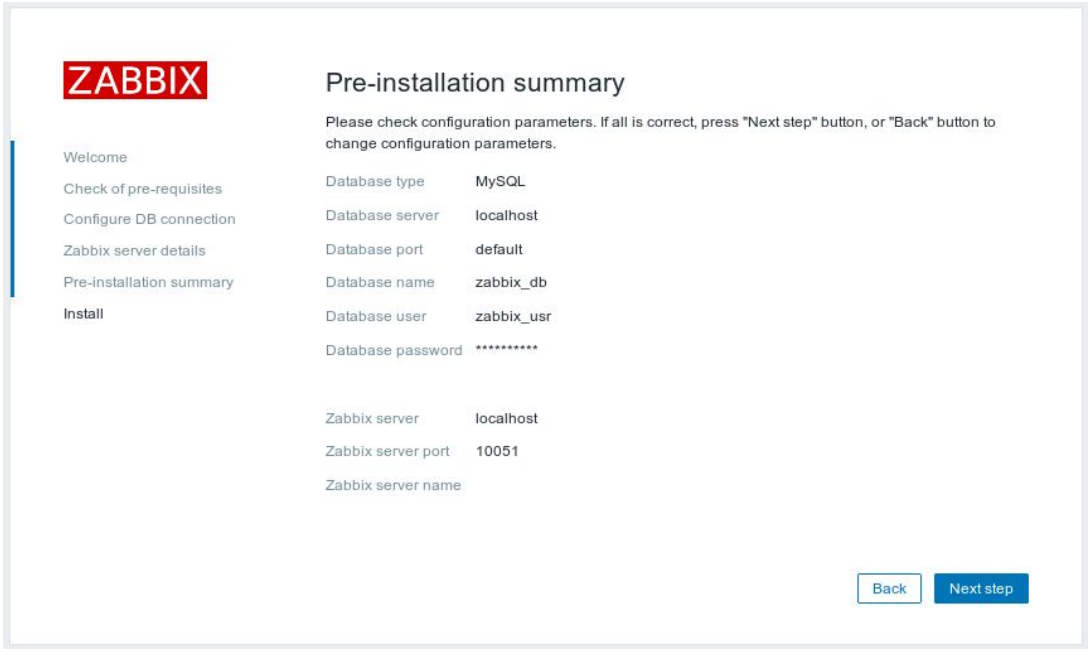
IP_SERVIDOR/zabbix

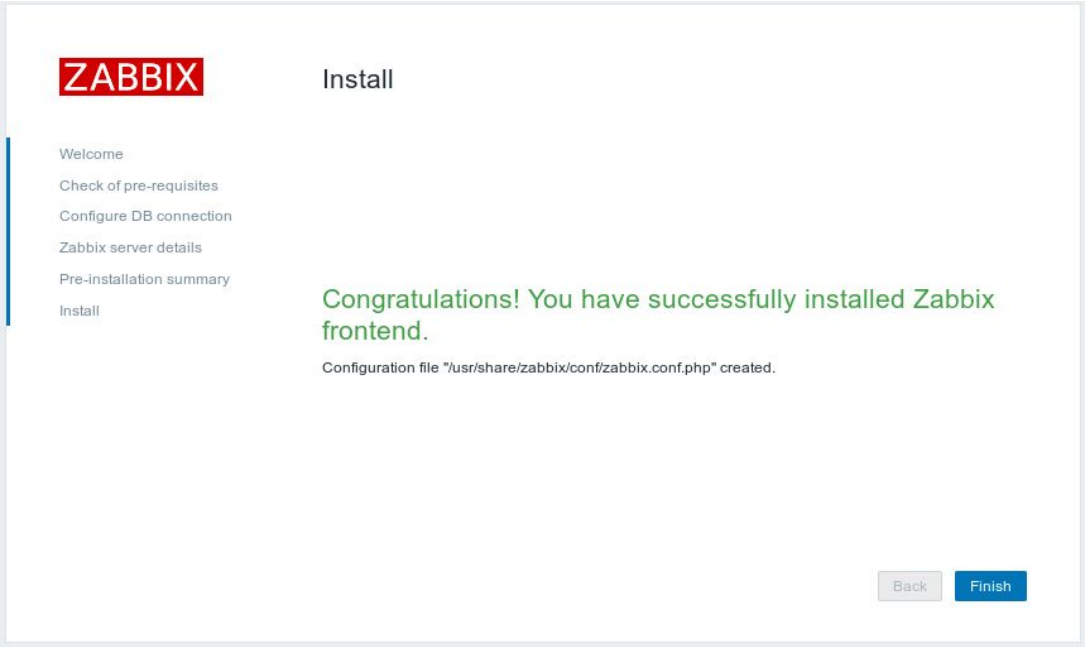
e seguir as telas abaixo:


ZABBIX
Welcome
Check of pre-requisites
Configure DB connection
Zabbix server details
Pre-installation summary
Install
Welcome to
Zabbix 3.4
Back
Next step
Licensed under GPL v2


ZABBIX
Welcome
Check of pre-requisites
Configure DB connection
Zabbix server details
Pre-installation summary
Install
Configure DB connection
Please create database manually, and set the configuration parameters for connection to this database. Press "Next step" button when done.
Database type
MySQL
Database host
localhost
Database port
0
0 - use default port
Database name
zabbix_db
User
zabbix_usr
Password
TCE@2018_zabbixdb
Back
Next step


ZABBIX
Welcome
Check of pre-requisites
Configure DB connection
Zabbix server details
Pre-installation summary
Install
Zabbix server details
Please enter the host name or host IP address and port number of the Zabbix server, as well as the name of the installation (optional).
Host
localhost
Port
10051
Name
Back
Next step


ZABBIX
Welcome
Check of pre-requisites
Configure DB connection
Zabbix server details
Pre-installation summary
Install
Pre-installation summary
Please check configuration parameters. If all is correct, press "Next step" button, or "Back" button to change configuration parameters.
Database type
MySQL
Database server
localhost
Database port
default
Database name
zabbix_db
Database user
zabbix_usr
Database password
**********
Zabbix server
localhost
Zabbix server port
10051
Zabbix server name
Back
Next step




Primeiro acesso

O login e a senha do primeiro acesso são:

Admin
zabbix

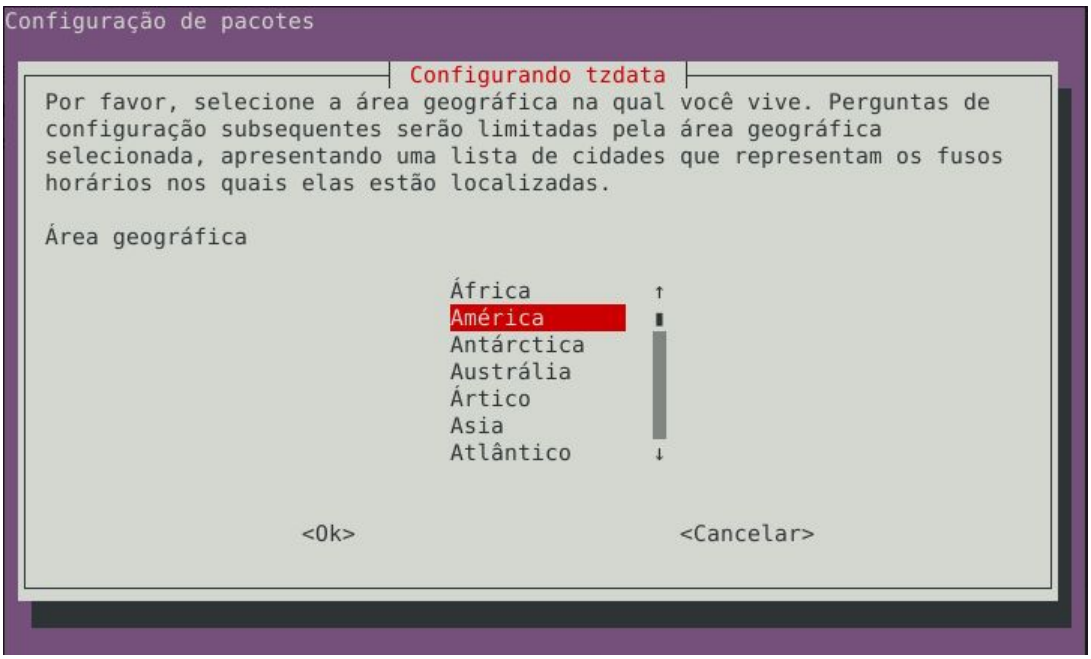
A senha deve ser mudada no primeiro acesso.

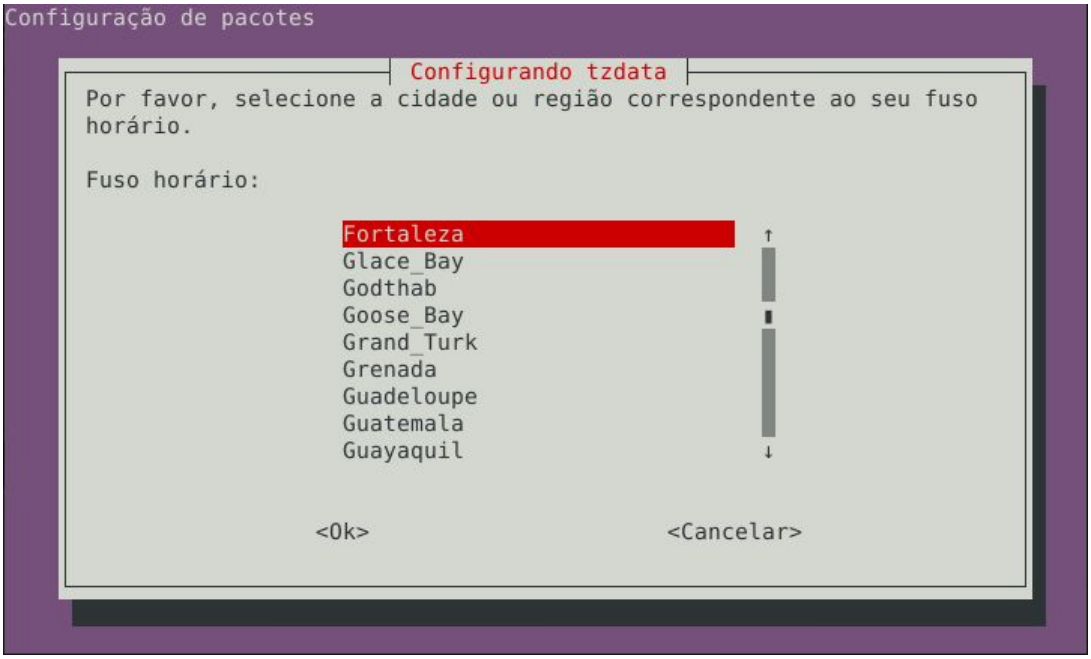
Ajuste da Time Zone e da Hora

Por se tratar de uma máquina sensível aos registros de horários, isto é, precisa ter uma maior acuracidade nos registros dos eventos, instalou-se também o ntpdate e ntp.

Definição da Time Zone adequada

#dpkg-reconfigure tzdata




Instalação dos pacotes para o NTP

#apt-get install ntp ntpdate -y

Edição do arquivo de configuração do ntp (ntp.conf)

#vi /etc/ntp.conf

Inserir os servidores NTP do NIC.br

Inserir abaixo da linha que traz o comentário “Specify one or more NTP servers.” as linhas:

server a.ntp.br
server b.ntp.br
server c.ntp.br
server gps.ntp.br

Reiniciado o serviço ntp e Carregando na Inicialização

systemctl restart ntp
systemctl enable ntp
Sincronizando o Relógio

#ntpd -g -n -q

Atualizando a hora da BIOS

#hwclock --systohc

---

# cd /tmp
# wget https://grafanarel.s3.amazonaws.com/b...
# dpkg -i grafana_4.1.1-1484211277_amd64.deb
# apt-get install -y adduser libfontconfig
# /bin/systemctl daemon-reload
# /bin/systemctl enable grafana-server
# /bin/systemctl start grafana-server
# grafana-cli plugins install alexanderzobnin-zabbix-app
# service grafana-server restart

# APÊNDICE F – Tabela gastos com a solução

| Atuador/Sensor/Módulo | Quant. | Valor | Total | |
|---|---|---|---|---|
| Buzzer | 6 | 1,99 | **R$ 11,94** | |
| LED | 6 | 1,49 | **R$ 8,94** | |
| OLED | 2 | 45,9 | **R$ 91,80** | |
| DHT1 | 6 | 14,9 | **R$ 89,40** | |
| DHT2 | 6 | 17,45 | **R$ 104,70** | |
| MQ7 | 2 | 21,9 | **R$ 43,80** | |
| MQ2 | 2 | 19,9 | **R$ 39,80** | |
| SIM800L | 2 | 75,9 | **R$ 151,80** | |
| ESP32 | 6 | 99,9 | **R$ 599,40** | |
| Capacitor 2200 | 7 | 1,98 | **R$ 13,86** | |
| Capacitor 1000 | 7 | 0,98 | **R$ 6,86** | |
| Resistor | 40 | 0,99 | **R$ 39,60** | |
| | | | **Total** | Total por módulo |
| | | | R$ 1.201,90 | R$ 200,32 |

# APÊNDICE G – Tabela preço de outras soluções

| # | Soluções de monitoramento encontradas no MercadoLivre: | | Valor | | |
|---|---|---|---|---|---|
| 1 | SE-10 - Monitor de Temperatura e Umidade para datacenter por Ethernet* | | R$ 1.650,00 | | |
| 2 | Medidor De Co2 Monitor De Temperatura E Umidade Detector Gás** | | R$ 880,00 | | |
| 3 | Monitor De Temperatura, Umidade E Fumaça Com Painel E Snmp | | R$ 3.450,00 | | |
| 4 | Sw-10 Monitor De Temperatura E Umidade Wifi - Snmp V2c | | R$ 1.425,00 | | |
| 5 | Sw-32 Monitor De Temperatura E Umidade Wifi & 915mhz | | R$ 1.849,85 | | |
| | * Não tem painel. | | | | |
| | **Não tem alertas. | | | | |
| 6 | Solução de monitoramento modular da APC NetBotz (em USD)*** | | 2.913,83 | | |
| | ** Em dólar, sem impostos de importação e outros impostos. | | | | |
| | Sub-item da solução NetBotz | Part Number | Valor (USD) | Qtd. | Sub-total |
| | APC NetBotz 250 | NBRK0250 | 549,00 | 1 | 549 |
| | APC Alarm Beacon | AP9324 | 179,99 | 1 | 179,99 |
| | APC Temperature Sensor | AP9335T | 99,99 | | |
| | APC Temperature & Humidity Sensor | AP9335TH | 149,99 | | |
| | APC Temperature & Humidity Sensor with Display | AP9520TH | 259,99 | | |
| | APC NetBotz Particle Sensor PS100 | NBES0201 | 329,99 | | |
| | NetBotz Spot Fluid Sensor - 15 ft. | NBES0301 | 119,99 | | |
| | NetBotz Door Switch Sensor for Rooms or 3rd Party I | NBES0302 | 59,99 | 1 | 59,99 |
| | NetBotz Door Switch Sensors (2) for an APC Rack - 1 | NBES0303 | 109,99 | 1 | 109,99 |
| | NetBotz Dry Contact Cable - 15 ft. | NBES0304 | 39,99 | | |
| | NetBotz 0-5V Cable - 15 ft. | NBES0305 | 69,99 | | |
| | NetBotz Vibration Sensor - 12 ft. | NBES0306 | 109,99 | 1 | 109,99 |
| | NetBotz Smoke Sensor - 10 ft. | NBES0307 | 199,99 | 1 | 199,99 |
| | NetBotz Leak Rope Sensor - 20 ft. | NBES0308 | 309,99 | | |
| | NetBotz Leak Rope Extension - 20 ft. | NBES0309 | 239,99 | | |
| | NetBotz Temperature Sensor - 32 in. (used with NetB | NBES0311 | 89,99 | 3 | 269,97 |
| | NetBotz Door Switch Sensor for 3rd Party Racks - 62 | NBES0312 | 44,99 | 1 | 44,99 |
| | NetBotz Door Switch Sensor for an APC Rack - 62 in | NBES0313 | 59,99 | 1 | 59,99 |
| | APC NetBotz 4-20mA Sensor Pod | NBPD0129 | 389,99 | | |
| | NetBotz Rack Sensor Pod 150 | NBPD0150 | 289,99 | | |
| | NetBotz Room Sensor Pod 155 | NBPD0155 | 339,99 | | |
| | NetBotz Wireless Sensor Pod 180 | NBPD0180 | 259,99 | 3 | 779,97 |
| | NetBotz Wireless USB Coordinator & Router | NBWC100U | 99,99 | 1 | 99,99 |
| | NetBotz Wireless Temperature & Humidity Sensor | NBWS100H | 149,99 | 3 | 449,97 |
| | NetBotz Wireless Temperature Sensor | NBWS100T | 99,99 | | |
| | Total (US$) = | | | | 2913,83 |
| | Convertando para Real (1 USD to BRL (R$ 4,0596) em 08/09/2018) e não incluindo os imposto | | | | R$ 11.828,98 |