SSC0952 - Internet das Coisas

Prof. Dr. Julio Cezar Estrella

Grupo 03 - Solução IoT

Utilizando o Eclipse Kura e Eclipse Kapua para salvar os dados do sensor de umidade e temperatura

Johnatas Luiz dos Santos nº USP: 13676388
Aruan Bretas de Oliveira Filho n° USP: 12609731
João Victor de Almeida n° USP: 13695424
Luiz Henrique Benedito n° USP: 12563814

Data de entrega: 5 de dezembro de 2023

Sumário

1 Introdução **2**
- 1.1 Descrição do Estudo de Caso e Problema 2
- 1.2 Objetivos 2
- 1.3 Planejamento do Projeto 2

2 Referencial Teórico **3**
- 2.1 Linguagem de Programação e Tecnologia na Camada de Aplicação 3
- 2.2 Ferramenta do Servidor Broker 4
- 2.3 Ferramenta do Servidor de Armazenamento 4
- 2.4 Tecnologias e Técnicas 4

3 Desenvolvimento **4**
- 3.1 Identificação dos Requisitos 4
- 3.2 Analise e Desenho do Sistema 5

4 Implementação **6**
- 4.1 Conexão Kura e Kapua 6
  - 4.1.1 Kapua 6
  - 4.1.2 Kura 10
- 4.2 Criando o broker no Kura 13
  - 4.2.1 Criação de um instancia de Simple Artemis Broker . . 13
  - 4.2.2 Criando um Subscriber para o Simple Artemis Broker . 14
  - 4.2.3 Criação do Subscriber do Broker do Kura 15
  - 4.2.4 Criando o Kapua Publisher 16
- 4.3 Conexão entro o Subscriber do Kura com Publisher do Kapua 17
- 4.4 Enviando os dados ESP32 20
- 4.5 Manipulação dos Dados 25
  - 4.5.1 Dados no kapua 25
  - 4.5.2 Kapua API 25
  - 4.5.3 Interface visual 26

5 Experimentos no Laboratório **27**

6 Resultados **29**

7 Conclusão **31**

1 Introdução

1.1 Descrição do Estudo de Caso e Problema

Os laboratórios de computação 1006 e 1008 são espaços dinâmicos e de grande atividade, onde a temperatura, a umidade, a presença de pessoas e o funcionamento adequado dos equipamentos desempenham um papel fundamental no desempenho de tarefas e na integridade dos experimentos. No entanto, a falta de um sistema de monitoramento eficaz pode levar a problemas, como:

- **Ineficiência na Gestão do Ambiente:** Sem informações precisas sobre a temperatura e umidade, os gestores dos laboratórios podem ter dificuldades em garantir condições ideais para o funcionamento dos equipamentos e o conforto dos usuários.
- **Risco para Equipamentos:** Variações extremas de temperatura e umidade podem afetar negativamente o desempenho e a vida útil dos equipamentos sensíveis, como computadores, servidores e dispositivos eletrônicos.
- **Desperdício de Energia:** Sem controle adequado da temperatura, o sistema de climatização pode operar de maneira ineficiente, levando a desperdício de energia e custos elevados.
- **Segurança das Pessoas:** O conhecimento sobre a presença de pessoas nos laboratórios é essencial em situações de emergência e para garantir a segurança dos usuários.

1.2 Objetivos

Desenvolver um sistema de dispositivos IoT capaz de medir e coletar dados sobre a temperatura e umidade dos laboratórios de computação 1006 e 1008, fornecer aos usuários (gestores, professores, alunos) uma interface intuitiva para monitorar as condições do laboratório, permitindo a tomada de decisões informadas e ação preventiva.

1.3 Planejamento do Projeto

Será utilizado o aplicativo Trello, em conjunto com a metodologia ágil Scrum, isto é, será feito reuniões semanais para definir os objetivos para a respectiva semana, enquanto todo o histórico de decisões, baseando-se também

na metodologia ágil será gravado no Trello. Além disso, será feitos as reuniões diárias, também conhecidas como daily, na qual, cada um responderá como está o andamento da respectiva parte do projeto.

2 Referencial Teórico

2.1 Linguagem de Programação e Tecnologia na Camada de Aplicação

- **ESP32**

  A ESP32 é uma série de microcontroladores de baixo custo e baixo consumo de energia, integrando Wi-Fi e Bluetooth dual-mode. Desenvolvida pela Espressif Systems, a ESP32 é uma evolução da conhecida ESP8266 e oferece uma solução completa e autônoma para a criação de aplicações de rede, eliminando a necessidade de microcontroladores externos. Além disso, a ESP32 possui um conjunto rico de periféricos, que inclui capacidades de sensoriamento analógico e digital.

  Vantagens:

  1. Conectividade: A ESP32 suporta Wi-Fi e Bluetooth, permitindo uma ampla gama de aplicações de conectividade.
  2. Desempenho: Equipada com um CPU dual-core e capacidade de multitarefa, a ESP32 é capaz de executar múltiplas funções simultaneamente.
  3. Flexibilidade: A ESP32 suporta uma variedade de modos de alimentação, tornando-a ideal para aplicações de baixo consumo de energia.

  Desvantagens:

  1. Consumo de energia: Embora a ESP32 tenha modos de economia de energia, aplicações que utilizam Wi-Fi e Bluetooth constantemente podem drenar a bateria rapidamente.

- **C++ para Programação de Microcontroladores**

  C++ é uma linguagem de programação de propósito geral, conhecida por sua eficiência e controle de baixo nível. No contexto de microcontroladores, como a ESP32, C++ oferece várias vantagens.

  Vantagens:

1. Eficiência: C++ permite a manipulação direta de recursos de hardware, o que pode levar a um código mais eficiente.
2. Orientação a Objetos: A capacidade de C++ de suportar programação orientada a objetos facilita a organização e modularização do código.
3. Portabilidade: Códigos escritos em C++ podem ser facilmente portados para diferentes plataformas com mínimas modificações.

Desvantagens:

1. Gerenciamento de Memória: Em microcontroladores, o gerenciamento de memória é crucial. C++ requer que os desenvolvedores gerenciem a memória manualmente, o que pode levar a erros se não for feito corretamente.

2.2 Ferramenta do Servidor Broker

O projeto usará o Eclipse Kura e Eclipse Kapua como ferramentas de servidor broker para facilitar a comunicação entre dispositivos.

2.3 Ferramenta do Servidor de Armazenamento

Os dados serão armazenados em um banco de dados SQL para permitir o registro de informações passadas.

2.4 Tecnologias e Técnicas

O projeto explorará tecnologias para a medição e o transporte das informações coletadas, utilizando servidores Broker, sensores e o ambiente monitorado, que é o laboratório.

3 Desenvolvimento

3.1 Identificação dos Requisitos

O projeto tem como requisitos funcionais a utilização do Esp32, juntamente com Kura e Kapua,onde ambos são componentes essenciais para um ecossistema de IoT, onde esse funcionará para a coleta de dados que os sensores irão obter. A princípio, o sistema vai agir para coletar dados de temperatura e umidade dentro de salas específicas. Em relação aos requisitos não funcionais, o projeto procura mapear as variações de temperatura e

contribuir para uma melhoria que seja adequado para aqueles que frequentam a sala. O sistema ser simples e prático para que qualquer pessoa possa monitorar esse recurso.

3.2 Analise e Desenho do Sistema

Com os objetivos definidos do projeto, teria que seguir a seguinte linha de raciocínio na parte técnica do projeto.

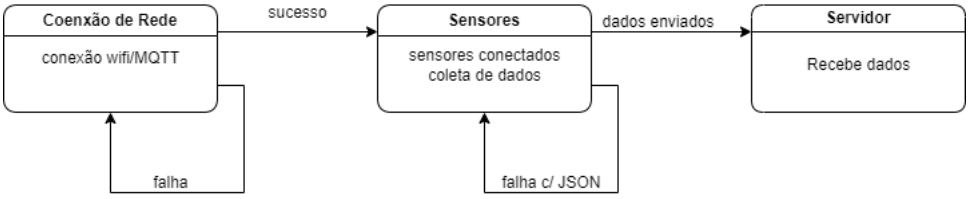

Figura 1: Diagramo de Estado

Teremos um processo de fazer uma conexão wifi, conseguindo acesso com nossos sensores de umidade e temperatura e então coletar os dados necessários para análise.

Pensando no banco de dados e nossas variáveis, montamos o seguinte diagrama:

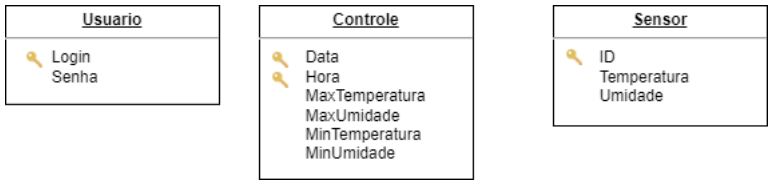

Figura 2: Banco de Dados

Teremos três tabelas, onde uma identificaremos nosso usuário que usará nossa aplicação, uma tabela para controle de informação das temperaturas e umidades dentro da sala e uma responsável pelo sensor.

A arquitetura do sistema por sua vez, vai ser dada desde nossa conexão com a rede wifi, fazendo conexão com nossos controles e sensores e coletando dados para nosso banco de dados e a partir dessas informações planejar melhorias para o ambiente dentro das salas e trazer uma qualidade maior para as pessoas que frequentam o ambiente.

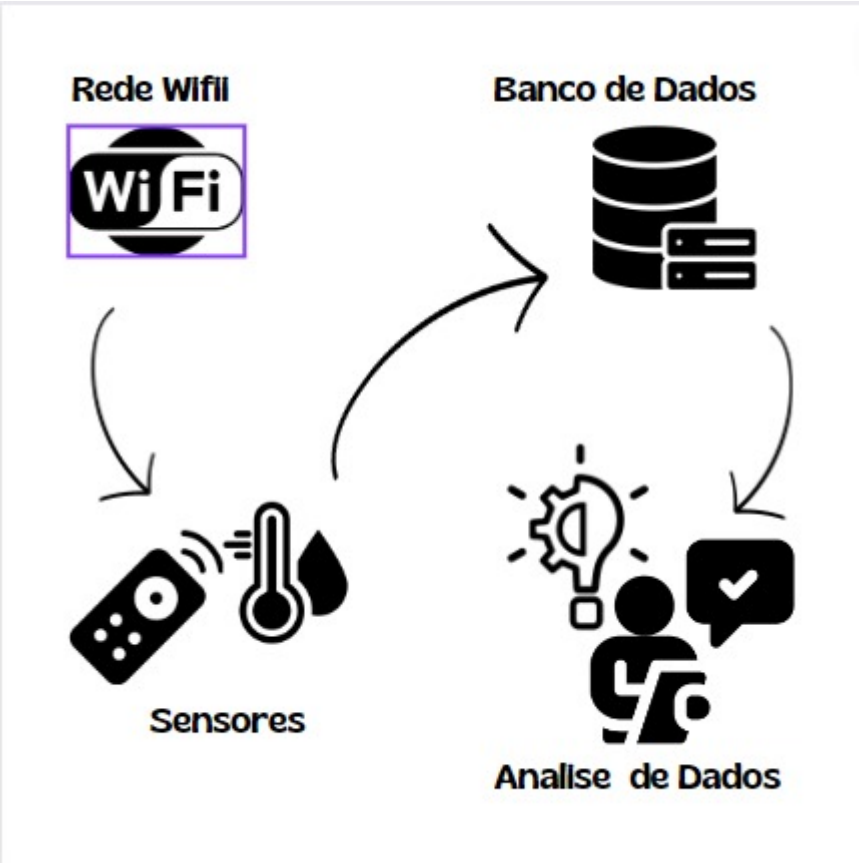

Figura 3: Arquitetura do Sistema

4 Implementação

4.1 Conexão Kura e Kapua

4.1.1 Kapua

Esta primeira parte consiste em várias etapas. Primeiro, você precisa baixar o Java VM, o Docker e o Kapua. Você pode obter o Java aqui e o Docker aqui.

Uma forma de iniciar tanto o Kapua e o Kura, é através do docker compose disponivel no github da disciplina. Outra forma, siga os passos abaixo para baixar e construir o Kapua.[2]

1. Abra o Terminal do seu sistema operacional (OS Shell) e vá para o diretório home.

2. Baixe o projeto Kapua do repositório do Github com o comando

   ```
   git clone https://github.com/eclipse/kapua.git
   ```

3. Após a conclusão da construção, execute o script de implantação Docker em deployment/docker: `./docker-deploy.sh`

   As imagens do Docker necessárias serão baixadas do Docker Hub e todos os contêineres serão iniciados.

   Você pode verificar se cada contêiner está funcionando corretamente digitando o seguinte comando: `docker ps -as`

4. Abra o navegador da web (Firefox ou Chrome) e acesse o console de administração em http://localhost:8080/. Caso esteja em um VM, talvez seja necessário acessar essa porta por meio do da porta SSH. Por exemplo:

   ```
   ssh -L 8080:localhost:8080 usuario@endereco_ip_da_vm
   ```

   Se tudo estiver correto, você deverá ver agora o prompt de login.

   Insira o nome de usuário e senha default.

   Nome de usuário: kapua-sys

   Senha: kapua-password

Agora você pode fazer login no Kapua, mas ainda precisamos criar uma Conta e um Usuário. Esta seção possui quatro etapas. A primeira é criar uma conta filha.

Enquanto estiver logado como kapua-sys, clique em Child Accounts e depois em add.

Nome da Conta:Account123
Nome da Organização:Account123org
Email: Account123@user123.com

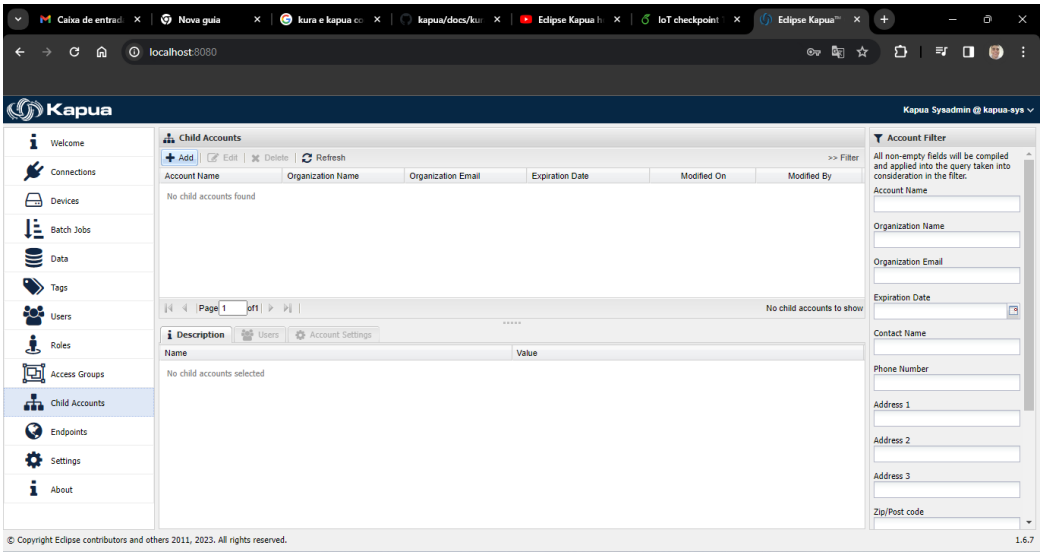

Figura 4: Criando nova conta

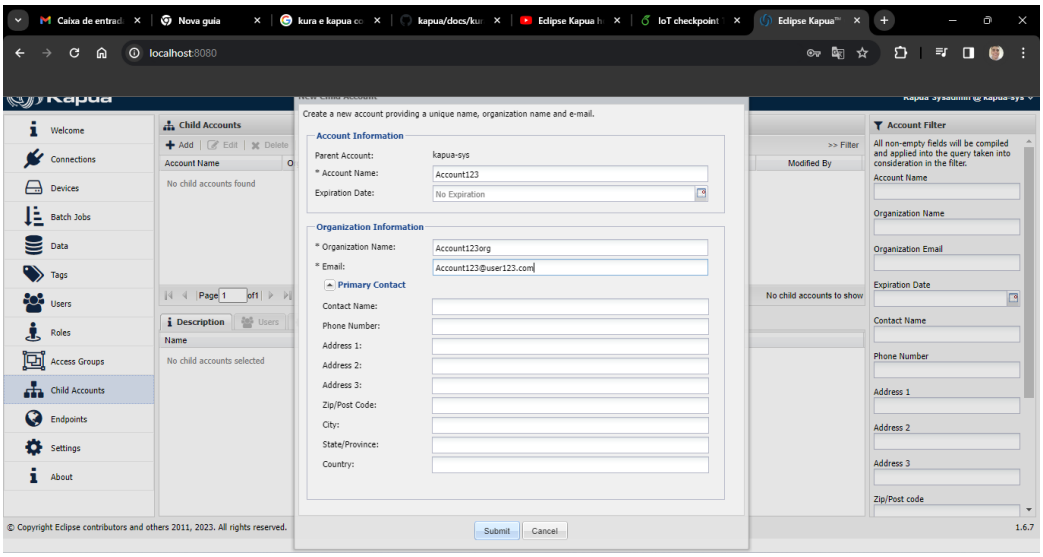

Figura 5: Configurando a Conta

Salve as credenciais ao clicar no botão Submit.

Antes de criarmos um usuário nesta conta, precisamos alterar algumas configurações na guia Account Settings abaixo. Basicamente, cada configuração de Service deve ter sua primeira opção deve ser definida como true, exceto 'TagService e CertificateService'. Nenhuma outra configuração deve ser alterada.

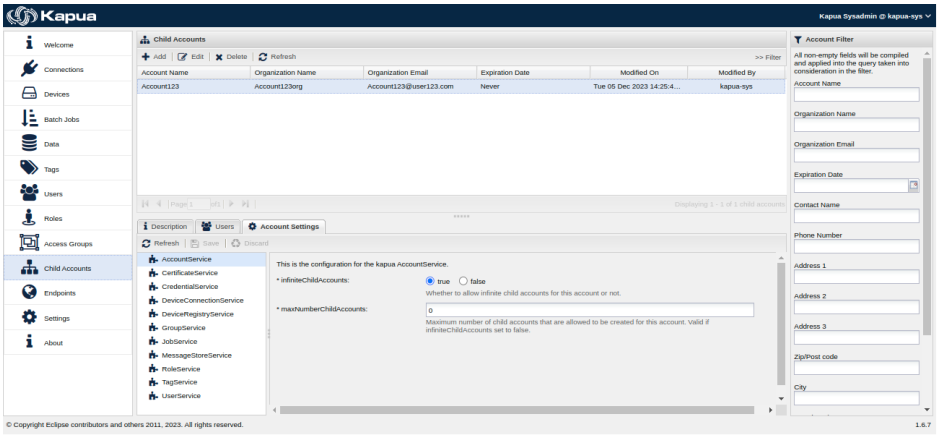

Figura 6: Configurando

Após isso, no topo direito selecione "Switch to account..."e selecione a nova conta Account123

Agora, clique na guia Users e depois em Add e crie um novo usuário com as credenciais:

Nome de Usuário: User123

Senha: Kapu@12345678

Confirmar Senha: Kapu@12345678

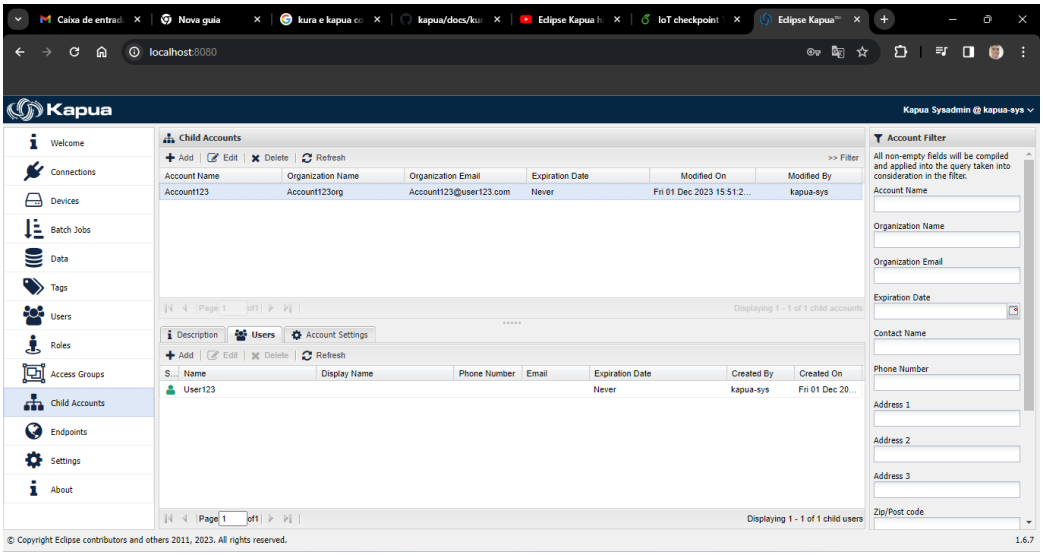

Figura 7: Child Account Criada

Vá para Users e selecione 'User123'. Na parte inferior, você encontrará

as guias Roles e Permissions. Clique em Roles, depois em Add, selecione Admin e confirme as alterações.

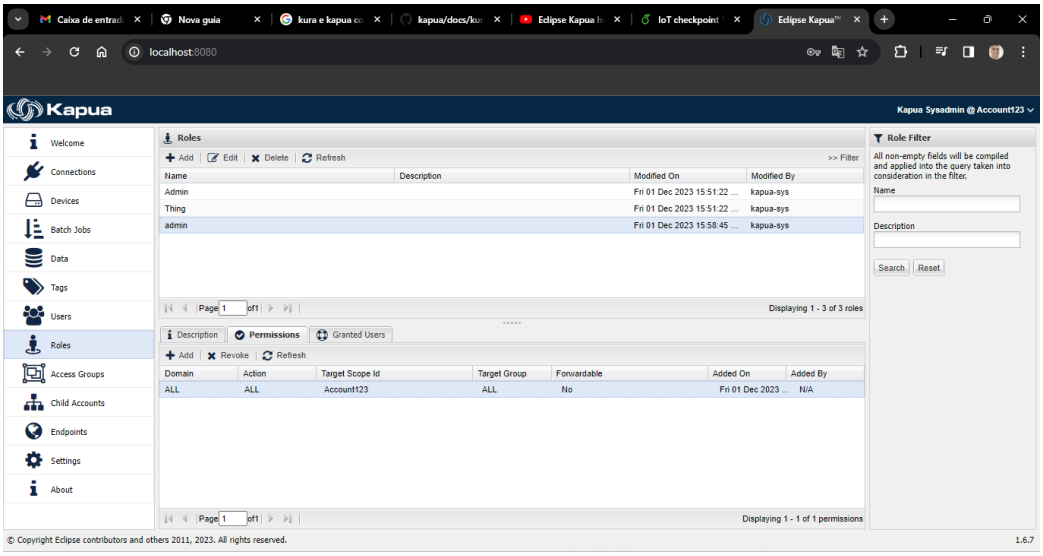

Figura 8: Roles

Após isso, vá para a guia Permissions, clique em add, selecione ALL em cada caixa de seleção e confirme as alterações.

Agora temos tudo o que precisamos para conectar o Kura ao Kapua.

4.1.2 Kura

Nessa parte agora é necessário iniciar o serviço do Kura, iniciamos pelo comando `docker run -d -p 443:443 -p 7043:7043 -t eclipse/kura`

Note que foi exposto duas portas a 443, que será usada para a configuração web do Kura e a porta 7043 que será usada posteriormente para receber tópicos do Broker. Outra forma é através do docker compose, que inicia tanto o kura e o kapua. Esse foi o escolhido pelo projeto, e está no GitHub da disciplina. [1]

A primeira etapa será configurar o serviço na nuvem para que possamos conectar ao nosso Kapua. Precisaremos preencher os dados da conta e do usuário que criamos anteriormente no Kapua. Para isso, clique em Cloud Connections e depois em New Connection, depois configure como abaixo

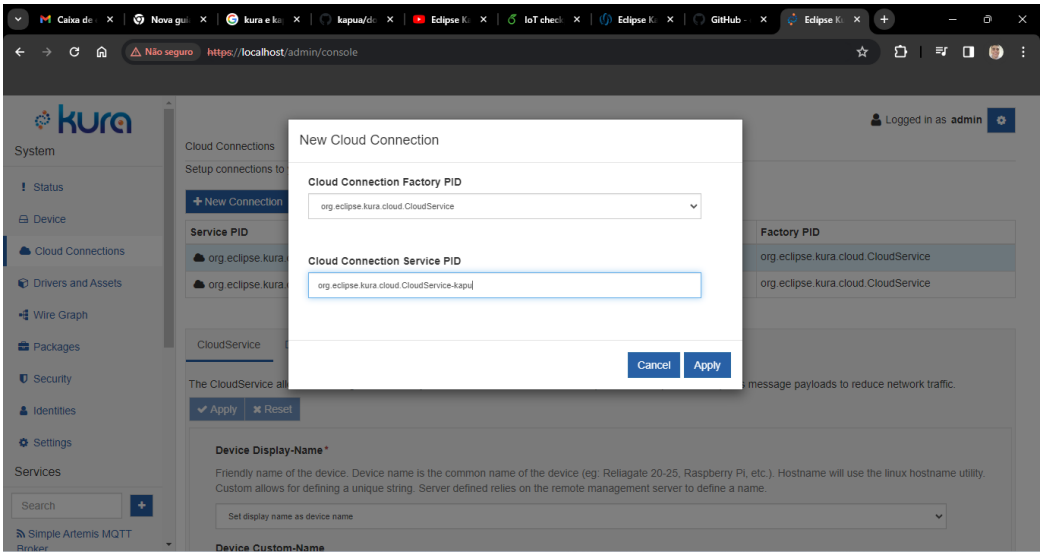

Figura 9: Criando Nova Conexão

Acesse o Cloud Service recém criado, e em 'MqttDataTransport', preencha os seguintes campos:

URL do Broker: mqqt://[endereço IP do PC local]:1883/, por exemplo, mqtt://192.168.1.27:1883/, ou mqtt://broker:1883

Tópico.context.account-name: Account123

Nome de Usuário: User123

Senha: Kapu@12345678

Client-id: kuraBroker (esse será o nome que a aparecerá em connections do kura)

Vá para 'DataService' e em 'connect.auto-on-startup' selecione true.

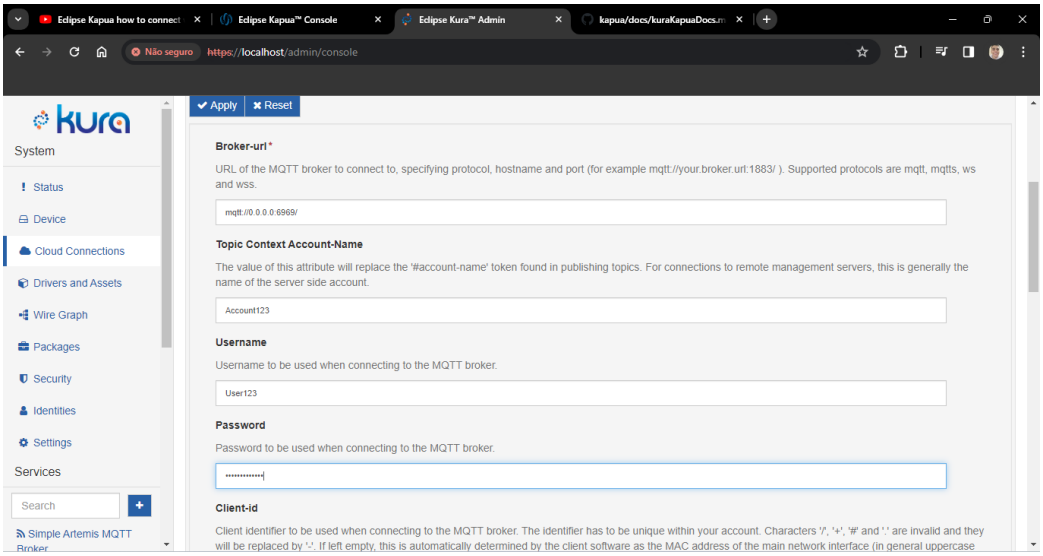

Figura 10: Conectando Kura com Kapua

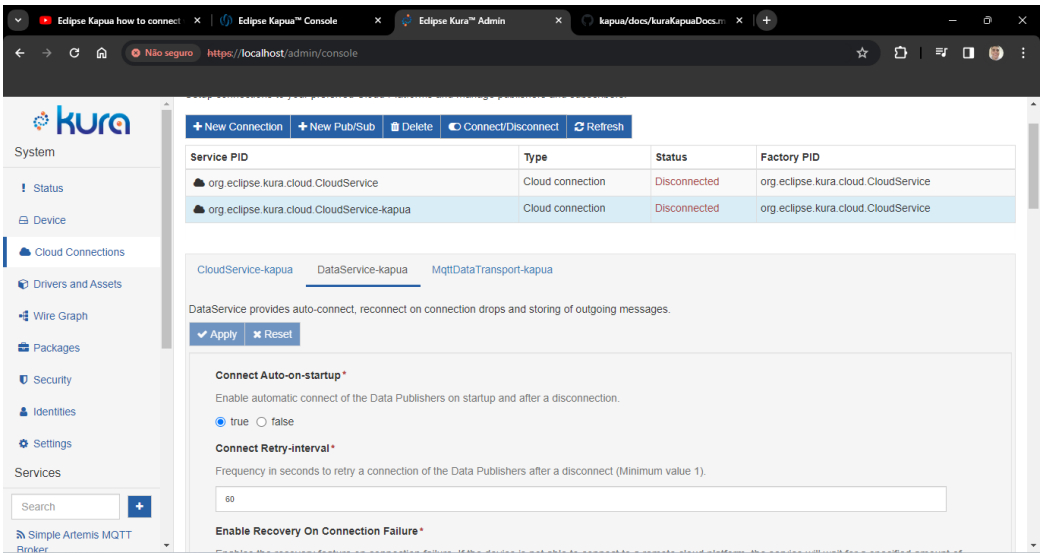

Figura 11: Selecionando Item como True

Note que o Topic Context Account-Name, Username e Password, são respectivamente a conta, usuario e senha de usuario que acabamos de criar no kapua. O Broker-url é o IP e a porta do broker do Kapua. É necessário que o porta esteja aberta para conexões de fora. Outra forma de realizar isso, é atraves do uso de mesma network entre os containers. Um exemplo está no repositório da disciplina. Após isso, clique em Connect/Disconnect.

Caso apresente o status Connected, houve a correta conexão entre o Kapua e Kura [6] [4]

4.2 Criando o broker no Kura

Após iniciado o Kura, iremos criar um broker para receber os dados da ESP-32. Para isso, iremos usar o Simple Artemis Broker.[5]

4.2.1 Criação de um instancia de Simple Artemis Broker

Vá para Services, escolha o Simple Artemis Broker, e faça a configuração dá seguinte maneira

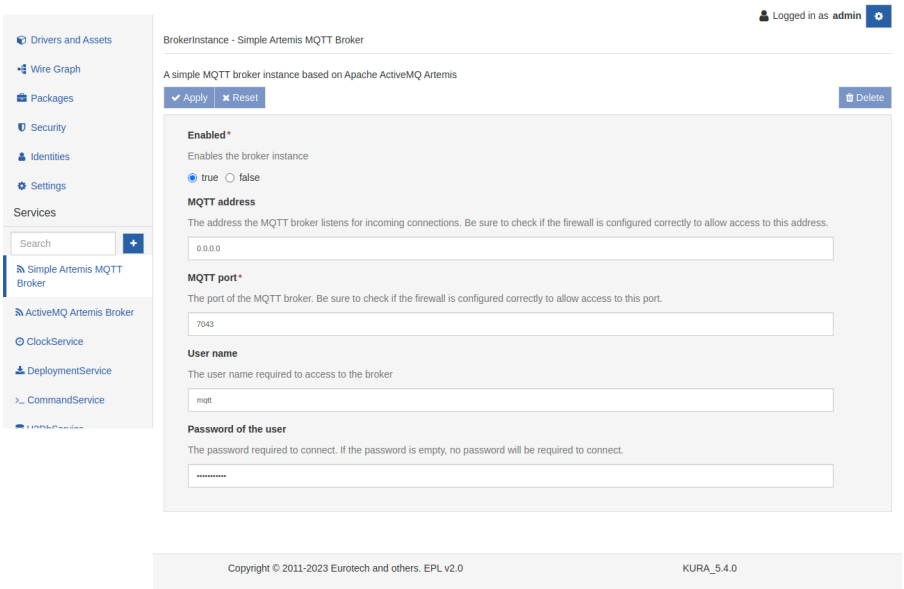

Figura 12: Configuração do Simple Artemis Broker

Sendo enable igual a true, indicando a criação da instância do broker, MQTT adress indicando que o serviço é local, MQTT port, a porta que a ESP32 irá enviar os dados (é necessário que ela esteja aberta para acesso via web, sendo a mesma porta extra aberta quando iniciado o Kura). Username e password, sendo o usuario e a senha do broker, aqui você tem uma escolha arbitrária, entretando usaremos esse usuário e senha posteriormente no Cloud Connections.

4.2.2 Criando um Subscriber para o Simple Artemis Broker

Em Cloud Connections, selecione New Connection. Após isso, selecione o Cloud Connection Factory PID, como org.eclipse.kura.cloud.CloudService, depois forneça um Service PID, isto é um nome para a conexão, na qual o prefixo inicia-se com "org.eclipse.kura.cloud.CloudService-". Por exemplo, org.eclipse.kura.cloud.CloudService-kuraBroker.

Após isso configure dá seguinte forma

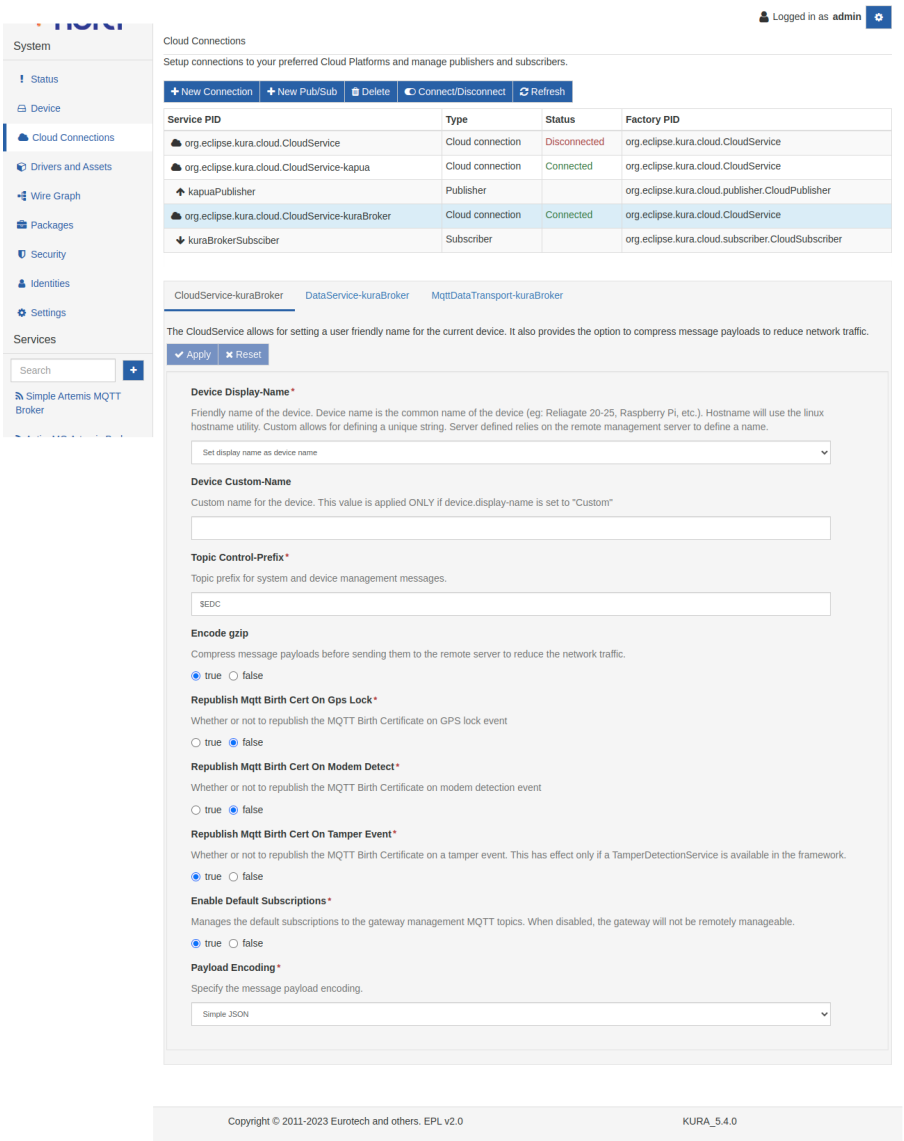

Figura 13: Selecionando o Payload Encoding

Selecionne para o tipo de Payload Enconding como Simple JSON. Além disso em DataService-kuraBroker, selecione true para Connect Auto-on-startup

Para MqttDataTransport, é necessario a seguinte configuração

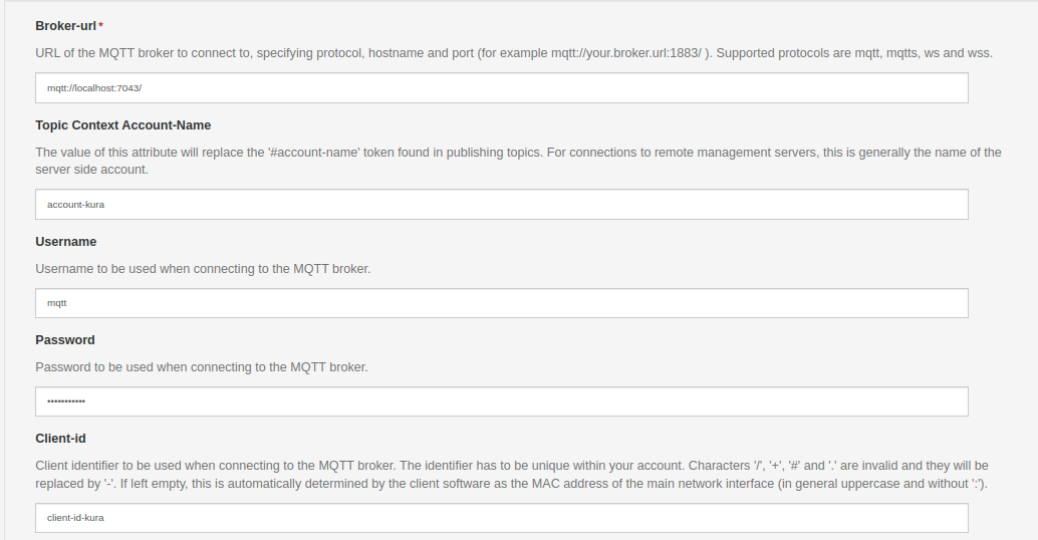

Figura 14: Configuração MqttDataTransport KuraBroker

O broker-url mqtt://localhost:7043, indica pelo localhost que é um serviço local e 7043 a porta extra aberta no Simple Artemis Broker. Configure username e password, como sendo os mesmo do Simple Artemis Broker. A seção Topic Context Account-Name e Client-id, podem ser escolhidos da forma que quiser. Entretanto, eles serão necessários na hora de enviar os tópicos para o Kura.

Depois disso, se configurado corretamente, você pode clicar em Connect/-Disconnect e não apresentar nenhum erro.

4.2.3 Criação do Subscriber do Broker do Kura

Após isso, em Cloud Connections, selecione o kuraBroker, clique em New Pub/Sub, selecione o org.eclipse.kura.cloud.subscriber.CloudSubscriber e forneça um nome arbitrário, como kuraBrokerSubscriber

Diante disso, forneça o application id e o application Topic , sendo esses necessarios para a configuracao dos topicos da ESP32

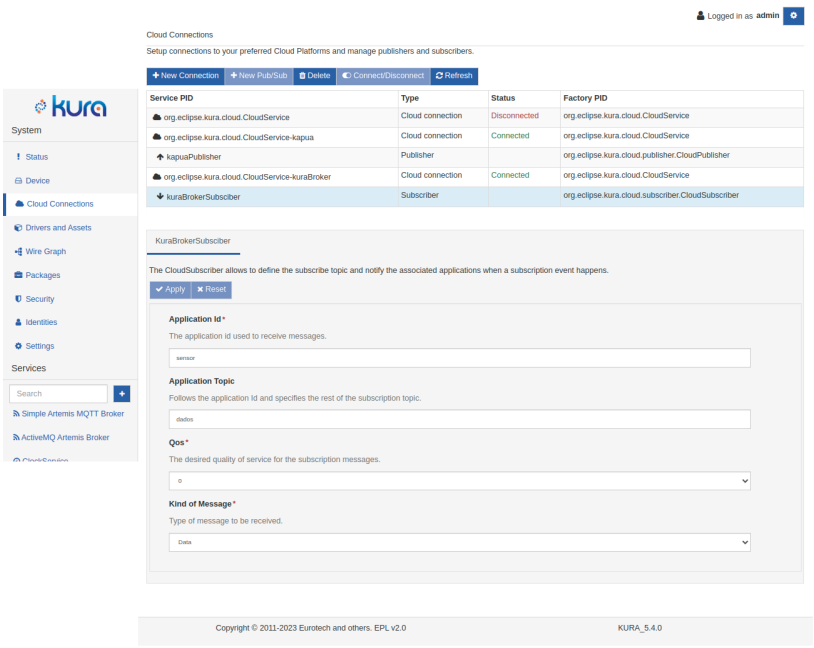

Figura 15: Configurando os tópicos do subscriber

Após aplicar as mudanças, você já deve ser capaz de enviar os tópicos no seguinte formato "account-name/client-id/application-id/application-topic". No nosso exemplo, isso seria "account-kura/client-id-kura/sensor/dados". Sendo o account-kura e client-id-kura, escolhidos no Cloud Connection do kura com o Simple Artemis Broker e o sensor e dados, escolhidos no nosso Cloud Subscriber

4.2.4 Criando o Kapua Publisher

Após isso, em Cloud Connections, selecione a Cloud Connection do Kapua, clique em New Pub/Sub selecione o org.eclipse.kura.cloud.publisher.CloudPublisher e forneça um nome arbitrario, como kapuaPublisher.

Diante disso, forneça o application id e o application Topic, de forma arbitraria. Entretanto esse valores seram usados na hora de salvar os dados no Kapua.

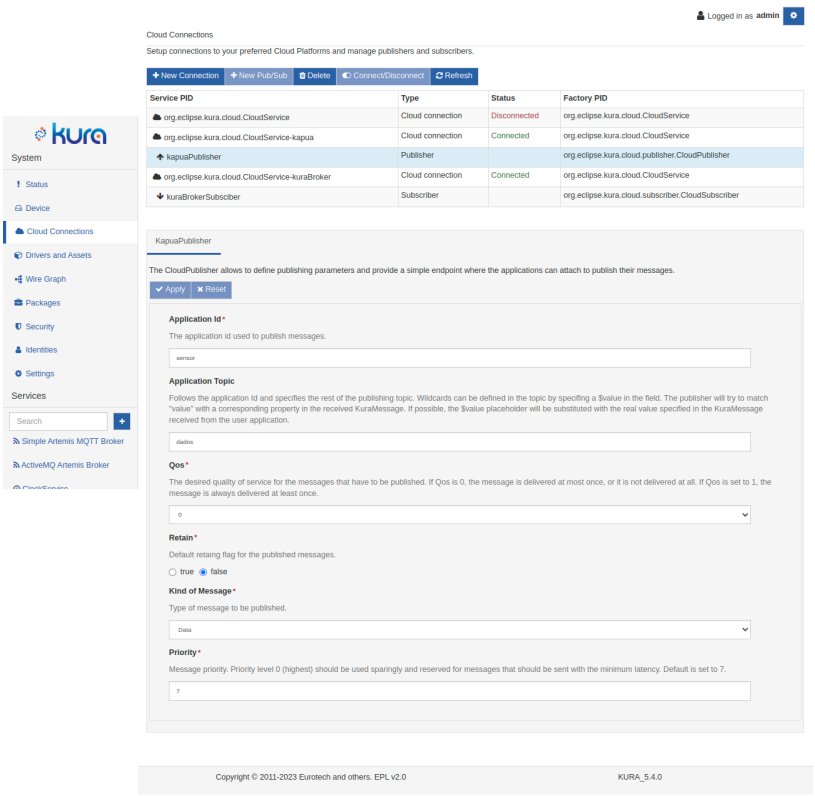

Figura 16: Configurando os tópicos do publisher

Você poderá enviar agora os dados agora para o kapua.

4.3 Conexão entro o Subscriber do Kura com Publisher do Kapua

Nessa etapa basta acessar Wire Graph para conectar e configurar o subscriber e o publisher.

Começando pelo Subscriber, dentro de Emitters terá o item subscriber

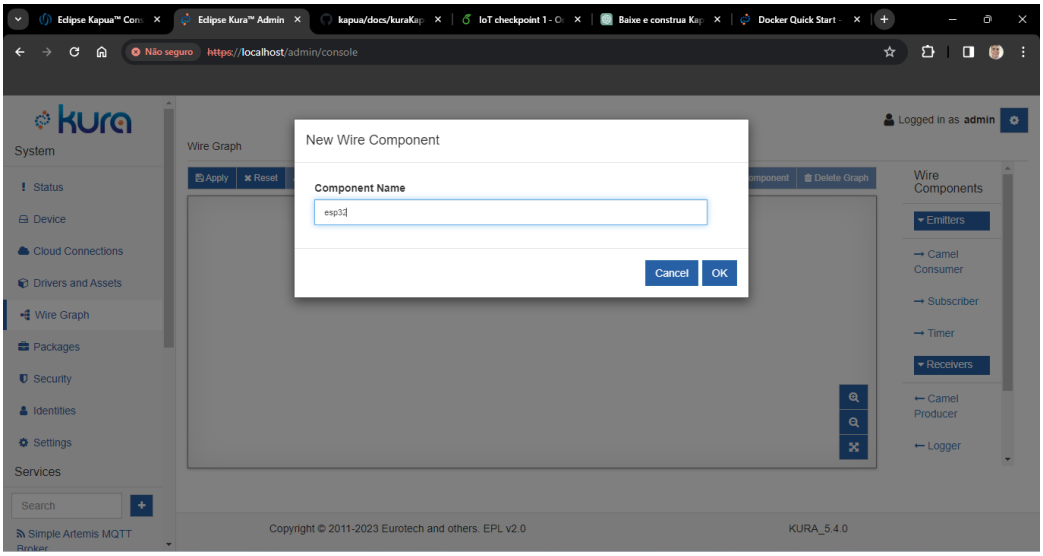

Figura 17: Criando Subscriber

Colocamos o nome como esp32, pois ela é a representação do nosso subscriber que estrá coletando os dados. Após isso configuramos ela como feito abaixo:.

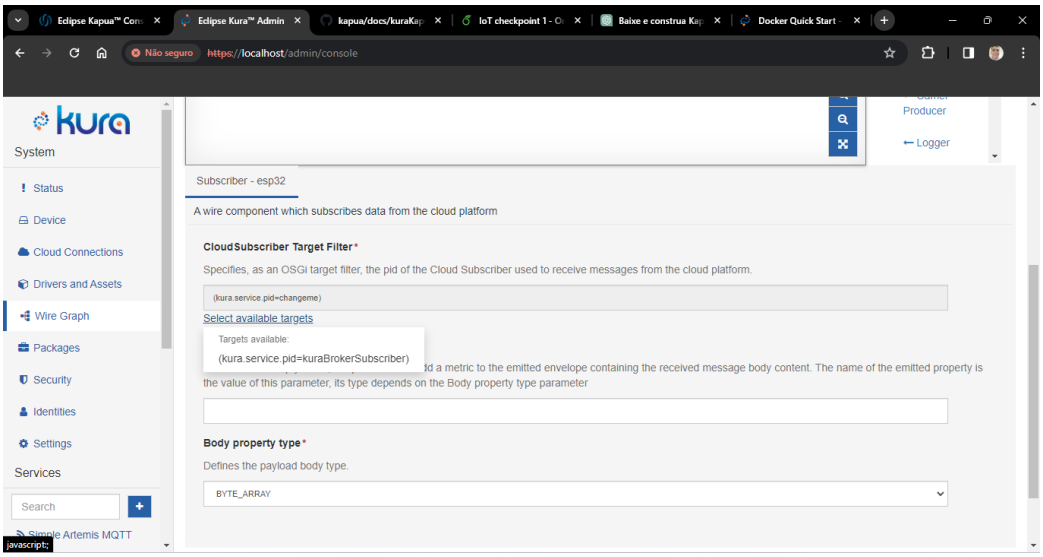

Figura 18: Configurando Subscriber

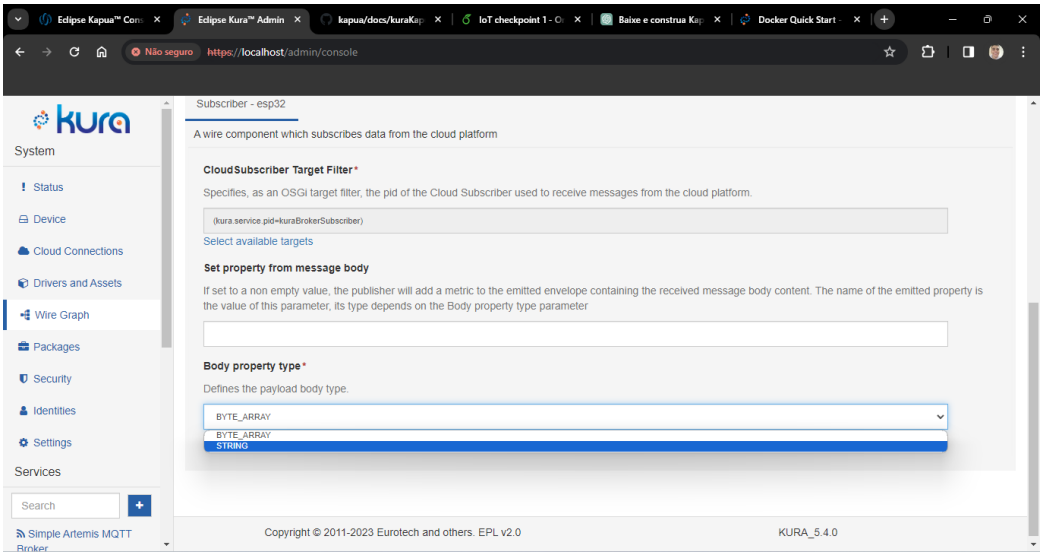

Figura 19: Mudando Tipagem para String

Após a criação e configuração do subscriber criaremos o publisher em Wire Graph também com passos similares ao anterior, porém dessa vez dentro de receivers será encontrado o publisher,sendo mostrado nos passos abaixo:

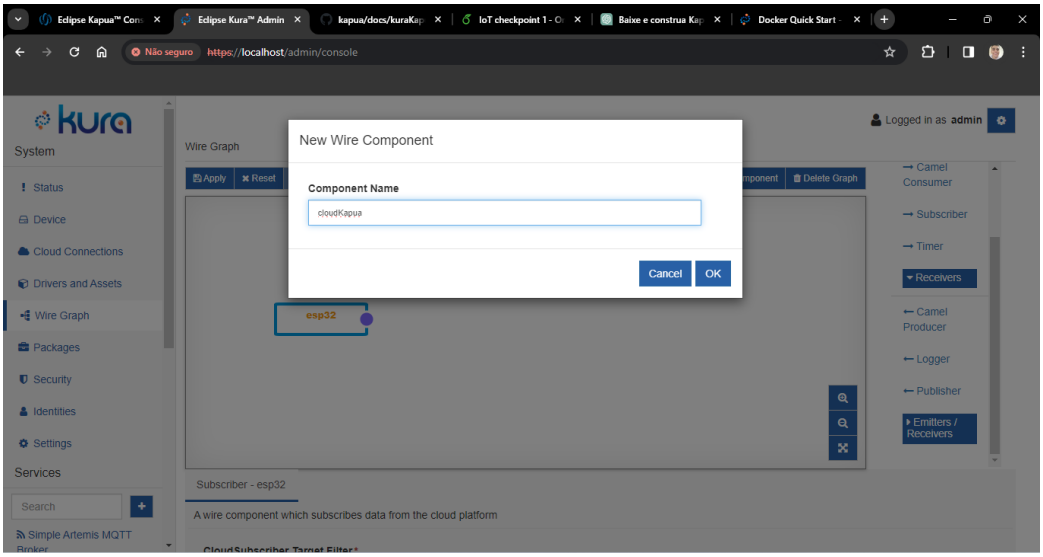

Figura 20: Criando Publisher

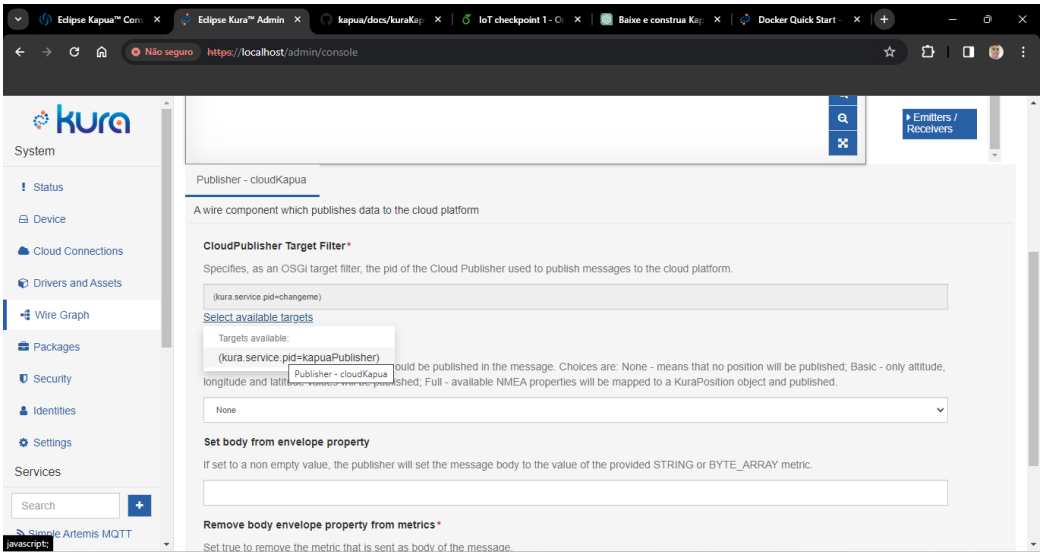

Figura 21: Configurando Publisher

Finalmente basta apenas conectá-los como na imagem a seguir:

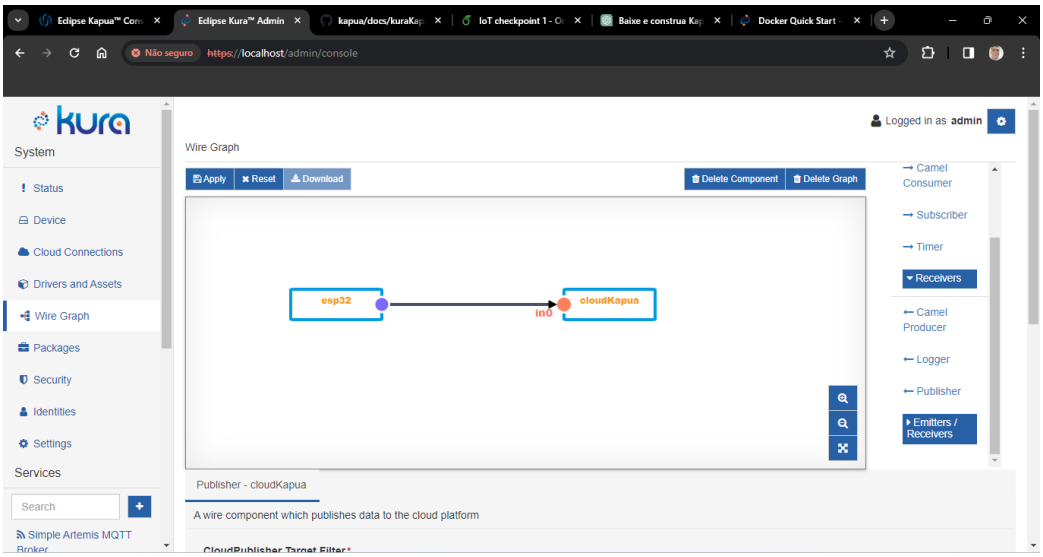

Figura 22: Conectando Subscriber e Publisher

4.4 Enviando os dados ESP32

Se tudo deu certo por aqui, você deve ser capaz de enviar através de um cliente MQTT, mensagem para o topico "account-kura/client-id-kura/sensor/dados".

Agora, configura-se a ESP32 para enviar os dados do sensor de humidade e temperatura para o broker do KURA.

Entretanto, a mensagem deve ser enviada por um JSON, no seguinte formato[3]:

```
{
  "metrics": {
    "temperatura" : 27.5,
    "umidade" : 57.5,
    "temperatura" : 30.2,
  }
}
```

Note que, o JSON apresenta uma chave metrics, e então um dupla de chaves com nome da metrica e seu respectivo valor. Aqui está o código da ESP32 para isso.

Listing 1: Exemplo de código para a ESP32

```
#include "DHT.h"
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>

#define DHTPIN 22     // Digital pin connected to the DHT sensor
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321

const char* ssid = "ROTA-69-2G";  // Configura  es do WIFI
const char* password = "SADDAN69";

// Configura  es do broker MQTT do KURA
const char* mqttServer = "143.107.232.252"; // IP da VM com o KURA
const int mqttPort = 7043; // Porta aberta do kura
const char* mqttUser = "mqtt"; // Usuario do broker
const char* mqttPassword = "mqtt@123"; // Senha do broker

DHT dht(DHTPIN, DHTTYPE);

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
```

```
24    Serial.begin(115200);
25    Serial.println(F("DHTxx test!"));
26    dht.begin();
27
28
29    WiFi.begin(ssid, password);
30    while (WiFi.status() != WL_CONNECTED) {
31      delay(1000);
32      Serial.println("Conectando ao WiFi...");
33    }
34
35    Serial.println("Conectado ao WiFi");
36    client.setServer(mqttServer, mqttPort);
37    client.setCallback(callback);
38
39    while (!client.connected()) {
40      if (client.connect("client-id-kura", mqttUser, mqttPassword)) {
41        Serial.println("Conectado ao broker MQTT");
42      } else {
43        Serial.print("Falha na conexao ao broker MQTT, rc=");
44        Serial.println(client.state());
45        // Serial.println(client.)
46        delay(2000);
47      }
48    }
49  }
50
51  void loop() {
52    // Aguardo entre leituras.
53    delay(6000);
54    if (!client.connected()){
55      reconnect();
56    }
57
58    // Lendo os dados do sensor
59    float h = dht.readHumidity();
60    // Read temperature as Celsius (the default)
61    float t = dht.readTemperature();
62    // Read temperature as Fahrenheit (isFahrenheit = true)
63    float f = dht.readTemperature(true);
64
```

```
65    // Verifica os dados.
66    if (isnan(h) || isnan(t) || isnan(f)) {
67      Serial.println(F("Failed to read from DHT sensor!"));
68      return;
69    }
70
71    // Compute heat index in Fahrenheit (the default)
72    float hif = dht.computeHeatIndex(f, h);
73    // Compute heat index in Celsius (isFahreheit = false)
74    float hic = dht.computeHeatIndex(t, h, false);
75
76    Serial.print(F("Humidity: "));
77    Serial.print(h);
78    Serial.print(F("%  Temperature: "));
79    Serial.print(t);
80    Serial.print(F(" C "));
81    Serial.print(f);
82    Serial.print(F(" F  Heat index: "));
83    Serial.print(hic);
84    Serial.print(F(" C "));
85    Serial.print(hif);
86    Serial.println(F(" F "));
87
88    StaticJsonDocument<200> doc;
89    JsonObject metrics = doc.createNestedObject("metrics");
90    metrics["temperatura"] = t;
91    metrics["umidade"] = h;
92    metrics["sensacao_termica"] = hic;
93
94    char jsonBuffer[256];
95    char* topico = "account-kura/client-id-kura/sensor/dados";
96
97    serializeJson(doc, jsonBuffer);
98    bool envio = client.publish(topico, jsonBuffer);
99
100   if (envio) {
101     Serial.println("JSON enviados com sucesso");
102   } else {
103     Serial.println("Falha ao enviar os dados");
104   }
105 }
```

```
106
107 void callback(char* topic, byte* payload, unsigned int length) {
108   // Funcao de callback chamada quando uma mensagem e recebida no topi
109   Serial.print("Mensagem recebida no topico: ");
110   Serial.println(topic);
111
112   Serial.print("Payload: ");
113   for (int i = 0; i < length; i++) {
114     Serial.print((char)payload[i]);
115   }
116   Serial.println();
117 }
118
119 void reconnect() {
120    while (!client.connected()) {
121       Serial.println("Tentando reconectar ao broker MQTT...");
122       if (client.connect("ESP32", mqttUser, mqttPassword)) {
123          Serial.println("Conectado ao broker MQTT");
124       } else {
125          Serial.print("Falha na conexao ao broker MQTT, rc=");
126          Serial.println(client.state());
127          delay(5000);  // Espere 5 segundos antes de tentar novamente
128       }
129    }
130 }
```

Nesse código, destaca-se as seguintes funcionalidades:

1. **DHT.h:** Essa biblioteca permite a leitura dos dados do sensor DHT22. Isto é possivel, por meio dos metodos de readHumidity, readTemperature e computeHeatIndex, que retornam a umidade, temperatura e sensação térmica.

2. **WiFi.h:** Essa biblioteca permite a conexao com WiFi e assim, enviar os dados para o broker

3. **PubSubClient.h:** Essa biblioteca permite a conexao com o broker, passando o client-id, usuario MQTT e senha MQTT;

4. **ArduinoJson.h:** Essa biblioteca permite a manipulação e criação do JSON necessário para o Kura;

Note que na linha 88-92, definimos os dados necessarios dentro de um json com metrics. Além disso, na linha 95, definimos o tópico, para qual, enviaremos a mensagem.

Depois disso, faremos o upload desse código para a ESP32. Você deve ver as linhas "Conectado ao WiFi"e "Conectado ao broker MQTT". Após isso, é necessário reiniciar a conexão Kura com o Simple Artemis Broker para a primeira conexão da ESP32. No nosso caso, ela é a conexão org.eclipse.kura.cloud.CloudService-kuraBroker, dentro de Cloud Connections do Kura, sendo preciso apenas clicar em Connect/Disconnect novamente.

4.5 Manipulação dos Dados

4.5.1 Dados no kapua

Se você chegou até aqui, deve conseguir ver uma tela, dentro de Data, na Conta Account123

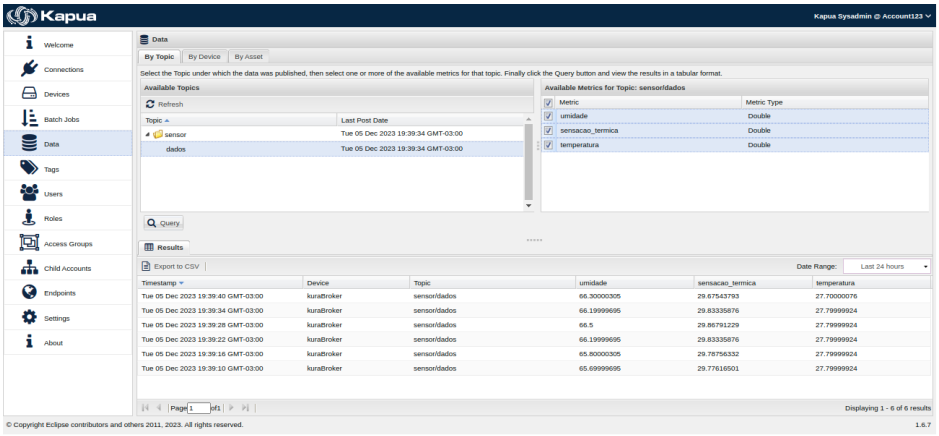

Figura 23: Dados no kapua

Seus dados, estão agora armazenados no kapua.

4.5.2 Kapua API

É possivel, utilizar o kapua-api para recuperar os dados necessários. Para isso, é necessario acessar a porta 8081 por meio de `http://localhost:8081/doc`

Aqui você poderá fazer testes, e conseguir diversos dados como os dados conexão do kapua, as metricas, mensagens e muito mais.

Para isso, configure a api para receber os dados da seguinte forma.

Figura 24: Swagger Kapua

Acesse o método de Authentication e o metodo /authentication/user, clique em "Try it out!", depois defina o username como "User123", e password "Kapu@12345678". Esses dados, referem-se a usuario criada no começo da configuração do kapua, sendo as mesmas da conexao kapua e kura dentro do Cloud Connections. Depois de executar, você deve recer um json com o tokenId. Copie esse tokenId e copie dentro de Autorize, no lado inferior esquerdo da imagem acima.

A partir de agora, você poderá usar outras funcionalidades do swagger. Como o GET "/scopeId/data/messages", dentro de Data Message. O scopeId deve permancer "_", ou outros métodos.

4.5.3 Interface visual

Utilizando o Kapua Api, é possivel criar uma aplicação web. Para isso é necessário, que tanto o container do kapua-api e da aplicação web, estejam na mesma network do Docker. Depois disso, todos os urls do kapua api permanceram os mesmos, trocando "localhost:8081"por "kapua-api:8080". Isso ocorre porque, o docker tem um DNS próprio capaz de fazer a ligacao entre os containers. Um exemplo de URL é

```
http://kapua-api:8080/v1/_/data/messages?clientId=kuraBroker&
                    sortDir=DESC&limit=50&offset=0
```

Com isso, foi possivel criar a aplicação a seguir

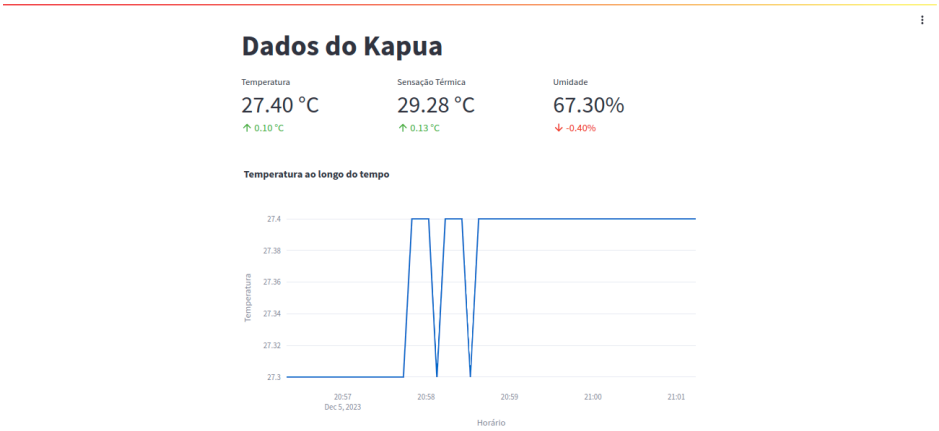

Figura 25: Interface Web

5 Experimentos no Laboratório

Após toda a implementação, foi feitos testes no laboratório, para verificar o sistema de monitoramento no laboratório em questão e no Data Center ao lado. A seguir, apresentamos fotos de todos os integrantes do grupo no laboratório e do sistema implementado no Data Center.

Figura 26: Foto em Conjunto do Grupo

A seguir a foto do sensor em uma região da sala do Data-Center.

Figura 27: Sensor no Data-Center ICMC

Em suma, os dados coletados no laboratório e no Data Center apresentaram diferenças devido à variação de temperatura e umidade em ambos os locais. O Data Center possui um ar-condicionado ajustado para temperaturas consideravelmente baixas, o que afeta a umidade do ar. Enquanto isso, o laboratório conta com um ar-condicionado que mantém a temperatura em um nível moderado.

6 Resultados

Dado a implementação do projeto, nós conseguimos o seguinte resultado de teste.

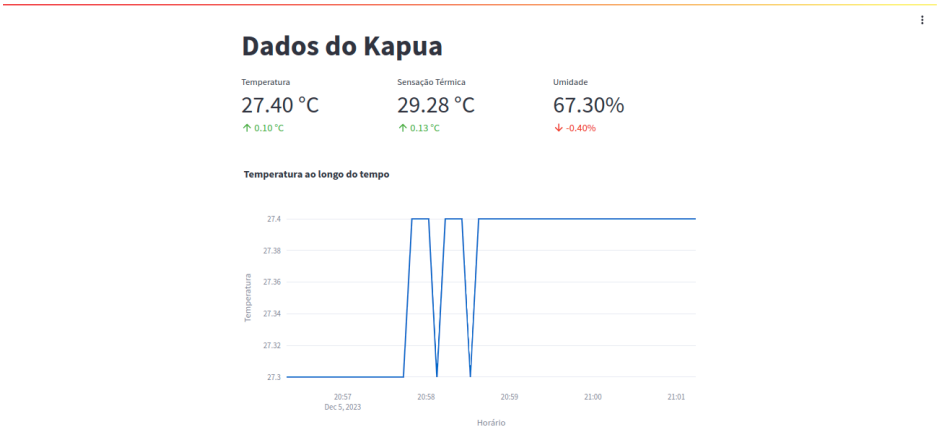

Figura 28: Interface Web

A conexão do kura e kapua, e a criação do broker do Kura.

Service PID	Type	Status	Factory PID
org.eclipse.kura.cloud.CloudService-kapua	Cloud connection	Connected	org.eclipse.kura.cloud.CloudService
kapuaPublisher	Publisher		org.eclipse.kura.cloud.publisher.CloudPublisher
org.eclipse.kura.cloud.CloudService-kuraBroker	Cloud connection	Connected	org.eclipse.kura.cloud.CloudService
kuraBrokerSubsciber	Subscriber		org.eclipse.kura.cloud.subscriber.CloudSubscriber

Figura 29: Conexões dentro do Kura

Além de guardados os dados no Kapua.

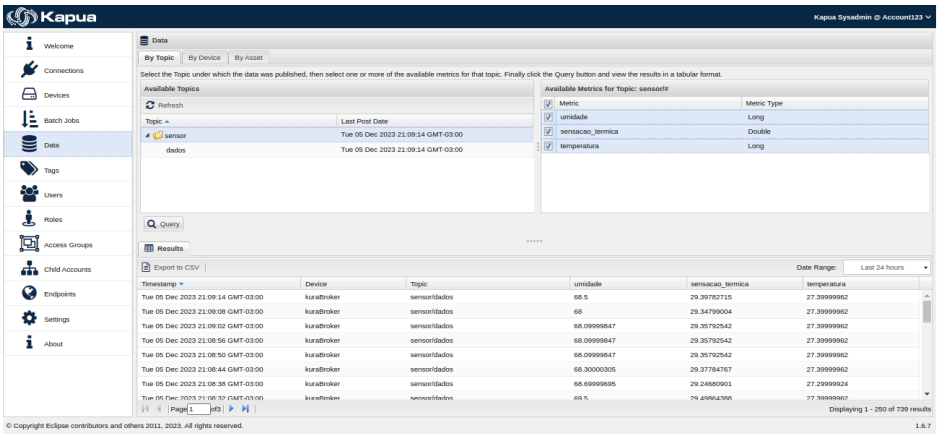

Figura 30: Dados no Kapua

7 Conclusão

Com o projeto, podemos analiasar como ficam as questões climáticas dentro da sala e poder monitorar o ambiente de forma a criar um local melhor para aqueles que frequetam o local. Além disso, foi possivel entender qual a necessidade de 'Internet das Coisas' utilizando sensores e frameworks para gerir dados importantes e que impactam no nosso dia a dia, além de compreender melhor como funciona o Kura e o Kapua nesse contexto. Por fim segue o link para acessar o APP: `http://andromeda.lasdpc.icmc.usp.br:8043/`

Referências

[1] Eclipse Kura Documentation. *Docker Quick Start*. Acessado em 10 de novembro de 2023. 2023. URL: `https://eclipse.github.io/kura/docs-release-5.3/getting-started/docker-quick-start/`.

[2] Eclipse Kapua. *Eclipse Kapua GitHub Repository*. Acessado em 15 de novembro de 2023. 2023. URL: `https://github.com/eclipse/kapua`.

[3] Eclipse Kapua. *K Payload JSON Format*. Acessado em 15 de novembro de 2023. 2023. URL: `https://github.com/eclipse/kapua/wiki/K-Payload-JSON-Format`.

[4] Eclipse Kapua. *Kura and Kapua Documentation*. Acessado em 15 de novembro de 2023. 2023. URL: `https://github.com/eclipse/kapua/blob/develop/docs/kuraKapuaDocs.md`.

[5] Eclipse Kura. *Subscribing data to in-built MQTT Server and sending data to Cloud Platform (Kapua)*. Acessado em 15 de novembro de 2023. 2023. URL: `https://github.com/eclipse/kura/wiki/Subscribing-data-to-in-built-MQTT-Server-and-sending-data-to-Cloud-Platform-(Kapua)`.

[6] YouTube. *Eclipse Kapua how to connect with Eclipse Kura*. 2023. URL: `https://www.youtube.com/watch?v=SoOhhgKO4As`.