UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE LORENA

DANILO BELLINTANI

Criação de um dispositivo IoT para coleta de dados ambientais e o Google Cloud como sistema de computação em nuvem

Lorena

2021

DANILO BELLINTANI

Criação de um dispositivo IoT para coleta de dados ambientais e o Google Cloud como sistema de computação em nuvem

Trabalho apresentado à Escola de Engenharia de Lorena da Universidade de São Paulo como parte dos requisitos para obtenção do título de Engenheiro Físico

Orientador: Prof. Dr. Carlos Yujiro Shigue

Lorena

2021

Ficha Catalográfica

EEL USP

Bellintani, Danilo
Criação de um dispositivo IoT para coleta de dados ambientais e o Google Cloud como sistema de computação em nuvem. / Danilo Bellintani; orientador Carlos Yujiro Shigue. -- Lorena, 2021.
Número de folhas f.: 65 il.
Monografia (Trabalho de graduação de Engenharia Física) - Escola de Engenharia de Lorena - Universidade de São Paulo
1. Computação em nuvem 2. Dados ambientais 3. Aquisição de dados 4. Internet das Coisas I. Título.

AGRADECIMENTOS

Em primeiro lugar, a Deus, que fez com que muitos dos meus objetivos fossem alcançados, durante todo esse tempo da minha vida.

Aos meus pais que foram muito pacientes e sempre me compreenderam e, também, por sempre me incentivar a ser uma pessoa boa com as outras e tentar dar o meu melhor.

Ao meu orientador, Prof. Dr. Carlos Yujiro Shigue, que, em 2016, me deu a oportunidade de começar um projeto de cultura em extensão. Acredito que foi algo extremamente importante para o meu desenvolvimento como pessoa e aluno, me possibilitou conhecer muitas coisas e trabalhar com outros projetos diferentes, como o Show de Ciências, o Planetário. E, é claro, que não poderia esquecer dos projetos feitos nas escolas da cidade de Cunha - SP, Geraldo Costa e Paulo Virginio.

Um gesto simples e um voto de fé, podem mudar muitas coisas.

RESUMO

BELLINTANI, D. **Criação de um dispositivo IoT para coleta de dados ambientais e o Google Cloud como sistema de computação em nuvem.** 2021. Número de folhas 65 f. Monografia (Trabalho de graduação de Engenharia Física) - Escola de Engenharia de Lorena, Universidade de São Paulo, Lorena, 2021.

O presente trabalho tem como objetivo criar e desenvolver um dispositivo IoT (*Internet of Things*) e um sistema de computação em nuvem utilizando *o Google Cloud Platform*, para que se pudesse fazer as análises dos dados em tempo real. O dispositivo IoT coleta dados ambientais em tempo real; o dispositivo foi construído com microcontrolador ESP32 juntamente com sensor BME280, que atua na coleta dos seguintes dados ambientais: pressão atmosférica, temperatura e umidade relativa do ar. A plataforma Google Cloud foi utilizada como sistema de computação em nuvem, e os serviços utilizados foram: o IoT Core como *broker* do protocolo MQTT, validador dos certificados TLS e agente de integração do microcontrolador com a plataforma, o Pub/Sub como *client* para receber as mensagens via protocolo MQTT, Cloud Functions para extrair as mensagens do Pub/Sub e inseri-las no banco de dados BigQuery. Como resultado, utilizando o *Data Studio*, obteve um relatório que mostra os dados ambientais em tempo real de algumas cidades.

Palavras-chave: Computação em nuvem. Dados ambientais. Aquisição de dados. Internet das Coisas.

ABSTRACT

BELLINTANI, D. **Development of an IoT device for environmental data collection and Google Cloud as a cloud computing system.** 2021. Number of sheets 65. Monograph (Undergraduate project in Engineering Physics) - Escola de Engenharia de Lorena, Universidade de São Paulo, Lorena, 2021.

This work aims to create and develop an IoT device (Internet of Things) and a cloud-based system using the Google Cloud Platform, so that real-time data analysis can be performed. The IoT device collects real-time environmental data; the device was built with an ESP32 microcontroller together with a BME280 sensor, which collects the following environmental data: atmospheric pressure, temperature and relative humidity. The Google Cloud platform was used as a cloud cloud system, and the services used were: IoT Core as MQTT protocol broker, TLS certificate validator and microcontroller integration agent with the platform, Pub / Sub as client to receive as messages via MQTT protocol, Cloud Functions to extract messages from Pub / Sub and insert them into BigQuery database. As a result, using Data Studio, you get a report that shows real-time environmental data for some cities.
.

Keywords: Cloud computing. Environmental data. Data Acquisition. Internet of Things.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O avanço da tecnologia está mudando a maneira como interagimos com o mundo ao nosso redor. Para atender às necessidades mais recentes dos consumidores, muitas das vezes essas demandas que os consumidores ainda não conhecem até que o produto seja lançado, as empresas vêm desenvolvendo e criando produtos com interfaces tecnológicas que seriam inimagináveis há algumas décadas atrás. Sistemas automatizados que avisam aos fazendeiros qual é o melhor momento de irrigar a sua plantação, ou que percebem que você está chegando em sua casa e acendem as luzes, até mesmo relógios que mensuram seus batimentos cardíacos e quando percebem uma anormalidade ligam para a emergência. Todos esses são exemplos de manifestações do conceito que vem sendo construído sobre a internet das coisas, mais conhecido pela sua sigla, IoT - *Internet of Things* (MAGRANI, 2018).

A todo momento, diversos dispositivos (“coisas”) se conectam à rede com capacidade para compartilhar, armazenar, processar e analisar um grande volume de dados. Entretanto, a Internet das Coisas pode trazer diversos benefícios às mais diversas estruturas da sociedade, um dos principais benefícios é o econômico. Por outro lado, existem desafios em relação à segurança, proteção e privacidade dos dados que estão relacionados com a grande conectividade. (MAGRANI, 2018).

Apesar da computação em nuvem ter ficado bastante conhecida poucos anos atrás, esse tipo de serviço já existia desde 1950. As máquinas naquela época eram extremamente caras, e uma vez uma empresa comprando esse serviço, ela teria uma máquina que só ela usaria, e toda a configuração seria de responsabilidade da empresa. Já nos anos 2000, a tecnologia de computação e nuvem ganhou forças e passou a ser oferecida comercialmente (ARMBRUST et al., 2009).

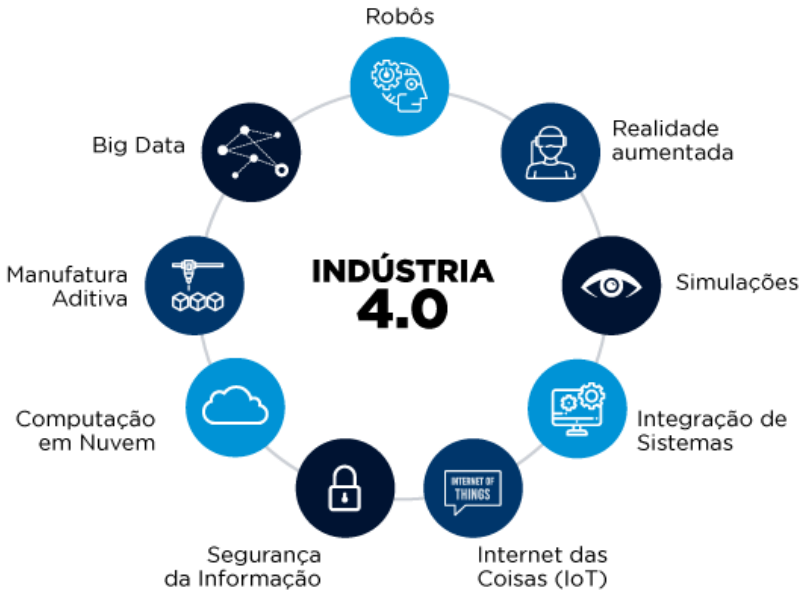
Nos anos 2000, para ser mais exato, em 2006, a AWS (*Amazon Web Services*) começou a oferecer como serviço o aluguel de máquinas virtuais, onde outras empresas ou pessoas poderiam rodar suas próprias aplicações e serviços, nesse momento, foi uma quebra de paradigmas, pois a Amazon fez duas coisas muito diferentes dos demais serviços que eram oferecidos naquela época, cobrança e controle. Em vez de pagar mensalmente para ter uma máquina, pagaria somente pelo uso dos recursos, caso necessitasse um maior poder computacional em um dado período (por exemplo, uma *black friday*), o próprio sistema faria um rearranjo das suas configurações e passaria a cobrar mais nesse tempo, para que seu sistema não tivesse gargalos, esse tipo de cobrança nunca tinha sido feito antes (AKITA, 2019).

Em 2007, o serviço streaming de vídeo foi lançado pela Netflix, usando os serviços de computação em nuvem para transmitir seus conteúdos para seus assinantes ao redor do mundo. Após um ano, em 2008, a Google lançou seus serviços no mercado de nuvem, *o Google Cloud Platform*, com custos baixos e inovações. Já em 2010, para tentar alcançar suas correntes, a Microsoft Azure criou seus sistemas de computação em nuvem (AKITA, 2019).

A partir daí, muitas empresas começaram a adotar a computação em nuvem em seu modelo de negócio e isso ajudou a desenvolver elas, como exemplo, a Uber, esse tipo de serviço ajudou no "boom" das startups. Ou seja, de lá pra cá o investimento na nuvem só vem crescendo e ficando cada vez mais barato, favorecendo diversas empresas (FERREIRA, 2015).

A junção do IoT, juntamente com a computação em nuvem, vem revolucionando a indústria em todos os setores, hoje chamada de indústria 4.0, ilustrado na Figura1 (SCHWAB, 2018).

Figura 1- Indústria 4.0.



Fonte: Tecnicon (2020).

1.2 Justificativa

O ensino de ciências é hoje de vital importância para o desenvolvimento para as pessoas, isso favorece o crescimento econômico, social, cultural do país. Isso tem levado a transformações na educação a fim de formar mentes criativas e transformadoras, onde os conteúdos vistos não sejam meramente informativos, mas sim formativos. As atividades práticas tornaram-se recursos facilitadores para a compreensão no ensino de ciências. Hoje, os dados fazem parte do cotidiano, seja usando um aplicativo de mensagens instantâneas para se comunicar ou só por ter um dispositivo no bolso que capta as informações de onde estamos localizados, ou que mostra quais lugares que mais gostamos de frequentar. Hoje vivemos em um mundo SaaS, *Services as a Services*. CaaS, Car as a Services (Uber, Cabify), HaaS, *House as a Service* (Airbnb), entre outros. Neste trabalho, iremos tratar do WaaS, *Web as a Service*, foi feito o projeto utilizando uma empresa que oferece esse serviço, veremos o quanto ele é importante nos tempos atuais, seja para desenvolver um projeto de iniciação científica ou começar a ter uma ideia sobre empreendedorismo e suas as possibilidades.

1.3 Objetivos

Este trabalho tem como objetivo a criação e desenvolvimento de um projeto científico de aquisição de dados ambientais em tempo real, utilizando a plataforma do Google Cloud como cerne principal do projeto. Será mostrado de forma simples como realizar a integração de um microcontrolador com a plataforma, e de como configurar a plataforma para que ela funcione de acordo com os objetivos do projeto.

2 REVISÃO DA LITERATURA

Neste tópico foi realizado uma revisão da literatura pertinente ao tema da monografia, que permitirá ao leitor que consiga replicar esse projeto por conta própria. Foi abordado o tema da internet das coisas, alguns protocolos de segurança, criptografia, comunicação e, também, sobre o ESP32 e computação em nuvem.

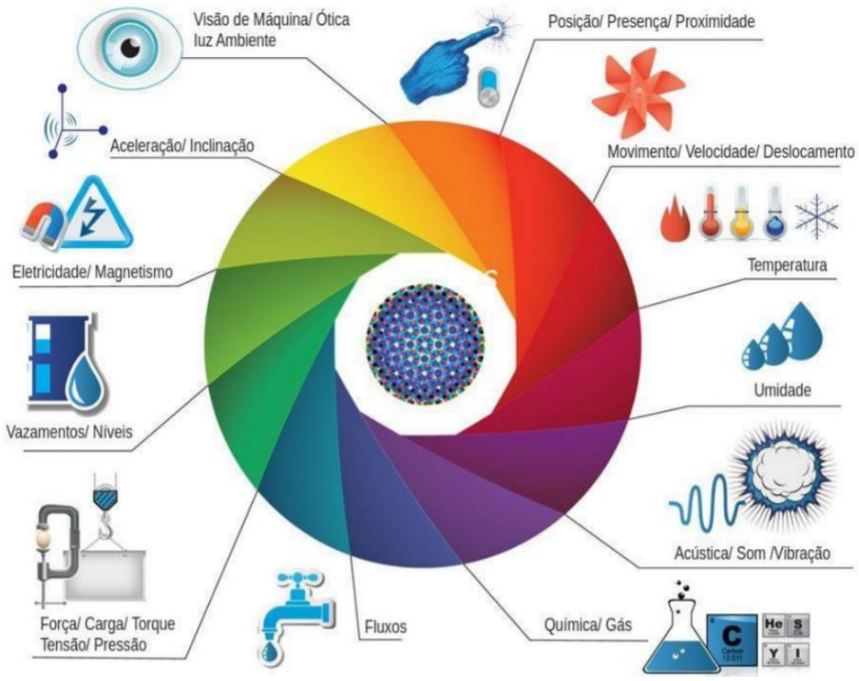
2.1 IoT - Internet das Coisas

A IoT (*Internet of Things*), ou Internet das Coisas, refere-se a qualquer conjunto de dispositivos físicos que recebem e transferem dados através da rede sem fio para sistemas integrados na nuvem, esses sistemas tem o objetivo de entender as “coisas” que estão conectadas (SANTOS, 2018).

Os sensores são uma das principais ferramentas utilizadas no mundo da IoT, com esses dispositivos é possível obter muitas informações, como por exemplo, temperatura, umidade, posição, intensidade da luz, sons, entre outras (SANTOS, 2018).

A Figura 2 mostra alguns tipos de dados que podem ser coletados utilizando sensores.

Figura 2 - Alguns tipos de dados que são coletar utilizando sensores.

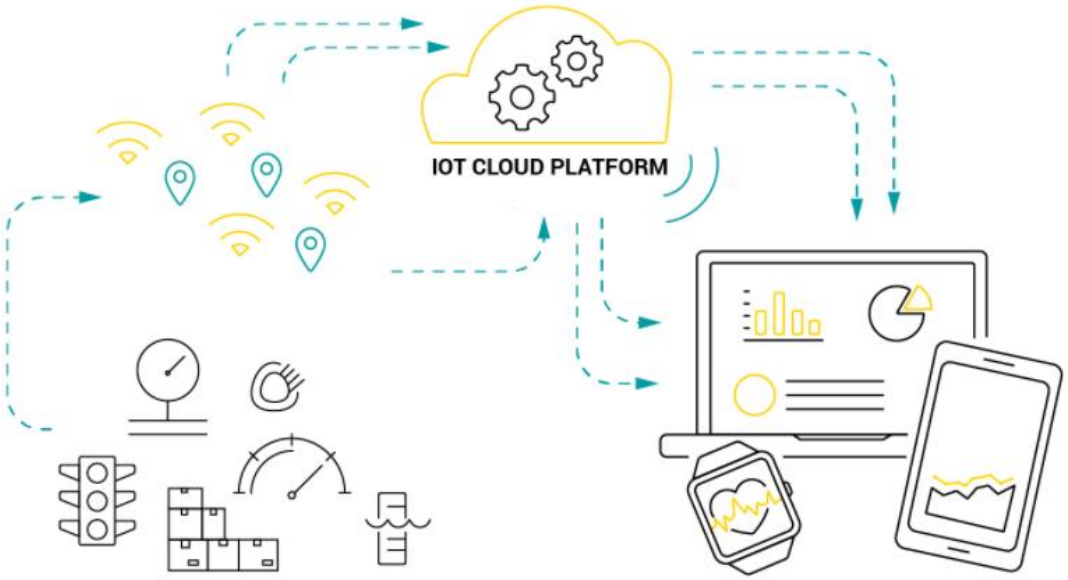


Fonte: Postscapes (2019).

Os dispositivos que possuem esses sensores acoplados costumam enviar dados para servidores de computação em nuvem, onde são agregados e analisados.

Utilizando os serviços de computação em nuvem, os dados coletados por dispositivos IoT são armazenados e processados em grandes data centers, o que garante segurança, confiabilidade e disponibilidade. Com isso, a estrutura física, atualização de *hardware*, processamento e armazenamento de dados fica a cargo das empresas provedoras dos serviços de computação em nuvem. Isso faz com que os custos das empresas que contratam esse tipo de serviço sejam reduzidos e que tenham foco somente em seus produtos e clientes (REDHAT, 2019).

Figura 3 - Imagem de dispositivos IoT se conectando a um cloud platform e outros dispositivos vendo as informações que foram geradas nesse processo.



Fonte: Adaptado de Avsystem (2020).

A ideia da IoT é permitir uma conexão autônoma, segura e a troca de dados entre dispositivos e aplicações do mundo real, onde é incorporado inteligência a esses dispositivos que estão conectados à Internet para agregar valor ao produto final. Com isso, começa a existir maior disponibilidade no mercado de serviços mais precisos (SILVA ,2017).

Hoje a evolução do IoT está bastante avançada, cada vez mais utiliza-se esses tipos de dispositivos, como por exemplo, as pulseiras e relógios inteligentes, que verificam nossos batimentos cardíacos, saturação de oxigênio em nosso sangue e o quanto nos movemos ao longo

do dia. A figura 4 mostra alguns relatórios gerados a partir dos dados captados pela pulseira inteligente.

Figura 4 - Relatório gerado a partir dos dados sensores que estão na pulseira inteligente (Mi Band 6).

Fonte: Próprio autor.

2.2 Computação em nuvem

A computação em nuvem é o fornecimento de serviços de TI, incluindo servidores, bancos de dados, tráfego de rede, softwares, análises, tudo através da rede de internet. Esse tipo de serviços oferece recursos mais flexíveis, economia e menos preocupação para seus usuários. Normalmente paga apenas pelo tempo e os serviços que é usado, ajudando a reduzir os custos operacionais, a executar sua infraestrutura com mais eficiência e a escalonar conforme as necessidades das empresas ou órgãos que estejam fazendo o uso desse tipo de produto. É uma grande mudança na forma tradicional de pensamento adotada pelas empresas sobre os recursos de TI (RAFAELS, 2015).

Segundo a Microsoft Azure (2021), os principais motivos que as organizações vêm adotando os serviços de computação em nuvem são:

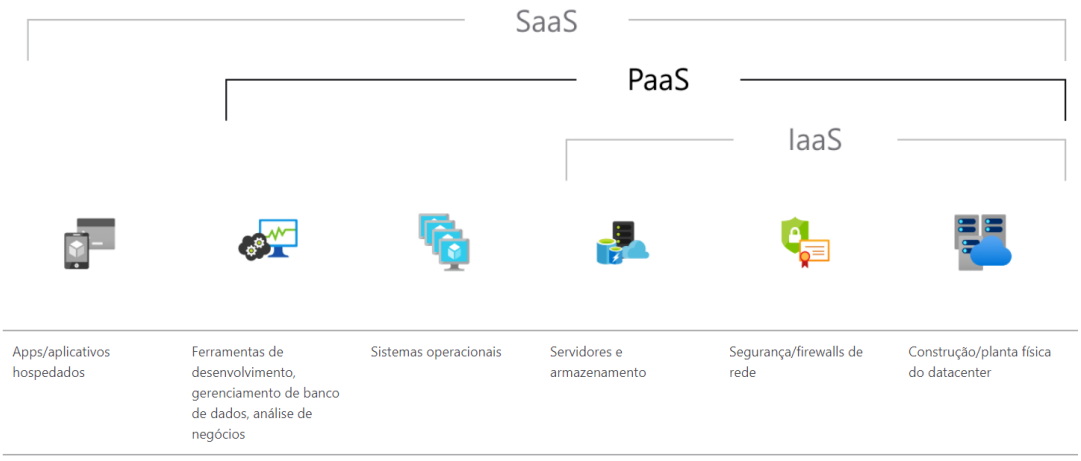
- Custos: Elimina os gastos com *hardwares* (servidores, datacenters locais) e futuros *upgrades*, *softwares* (sistemas operacionais, programas para execução de rotinas, entre outros), eletricidade (energia para os equipamentos, refrigeração) e Recursos Humanos (não precisando mais de mão de obra qualificada, para o serviço de infra estrutura)
- Velocidade: A maior parte dos serviços de computação em nuvem é fornecida por autosserviço e sob demanda, para que até grandes quantidades de recursos de computação possam ser provisionadas em minutos, normalmente com apenas alguns cliques, fornecendo às empresas muita flexibilidade e aliviando a pressão do planejamento de capacidade.
- Desempenho: Os recursos da computação em nuvem são executados em uma rede mundial de datacenters, os hardwares são de última geração e atualizado regularmente, entretanto, como estamos falando de datacenters, é possível gerar configurações diversas de hardwares, muito mais potentes do que encontramos comercialmente.
- Escalabilidade: Um dos maiores benefícios é a capacidade de dimensionamento elástico, que significa fornecer a quantidade certa de recursos necessários sempre que um serviço demanda maior ou menor poder computacional.
- Segurança e confiabilidade: Esse tipo de serviço reduz os custos de backup de dados, recuperação de desastre e continuidade dos negócios, já que os dados são espelhados em diversos servidores para garantir a redundância. Além disso, provedores em nuvem oferecem um amplo conjunto de políticas, tecnologias e controles que fortalecem sua postura geral de segurança, ajudando a proteger os dados, os aplicativos e a infraestrutura contra possíveis ameaças.

Devido à evolução contínua da computação em nuvem, vários modelos passaram a existir para acompanhar a demanda do mercado e a evolução contínua da Internet. (MELL; GRANCE, 2011) definiram três modelos de serviço diferentes:

- ***Cloud Software as a Service (SaaS)*** - o SaaS é um conjunto de aplicativos que são fornecidos na infraestrutura na nuvem. No geral, a responsabilidade e a organização gerencial são administradas pela empresa que fornece os serviços. O usuário é responsável por gerenciar as configurações conforme seu uso.
- ***Platform as a Service (PaaS)*** - PaaS oferece uma estrutura para posicionar aplicativos que já passaram do estágio de desenvolvimento, isso facilita para os desenvolvedores que podem criar rapidamente aplicativos móveis ou da web sem se preocupar com a configuração ou gerenciamento da infraestrutura subjacente de servidores, armazenamento, redes e bancos de dados necessários.
- ***Infrastructure as a Service (IaaS)*** - IaaS fornece os recursos como servidores, máquinas virtuais (VMs), redes e sistemas operacionais. O usuário é responsável por fazer todo o gerenciamento de aplicativos e configurações de rede.

Na figura 5 são mostrados os tipos de serviços que são comportados dentro de cada modelo.

Figura 5 - Sistemas comportados por cada tipo de modelo.



Fonte: Azure (2021).

2.3 Microcontrolador ESP32

O ESP32 é um microcontrolador de circuito integrado compacto, projetado para gerenciar um conjunto de operações específicas em um sistema embarcado. Esse microcontrolador conta com Wi-Fi, Bluetooth, sensor de temperatura e sensor hall integrados, fazendo com que ele seja amplamente usado no mundo IoT (KURNIAWAN, 2019).

Historicamente o Arduino foi um dos grandes precursores da modalidade que hoje é chamado de '*open source-hardware*' e que possibilitou grande parte da criação de novos dispositivos desse tipo no mercado, como por exemplo, ESP32 (MULTILOGICA, 2019).

Na tabela 1, há um comparativo entre os hardwares dos microcontroladores da família ESP e o Arduino.

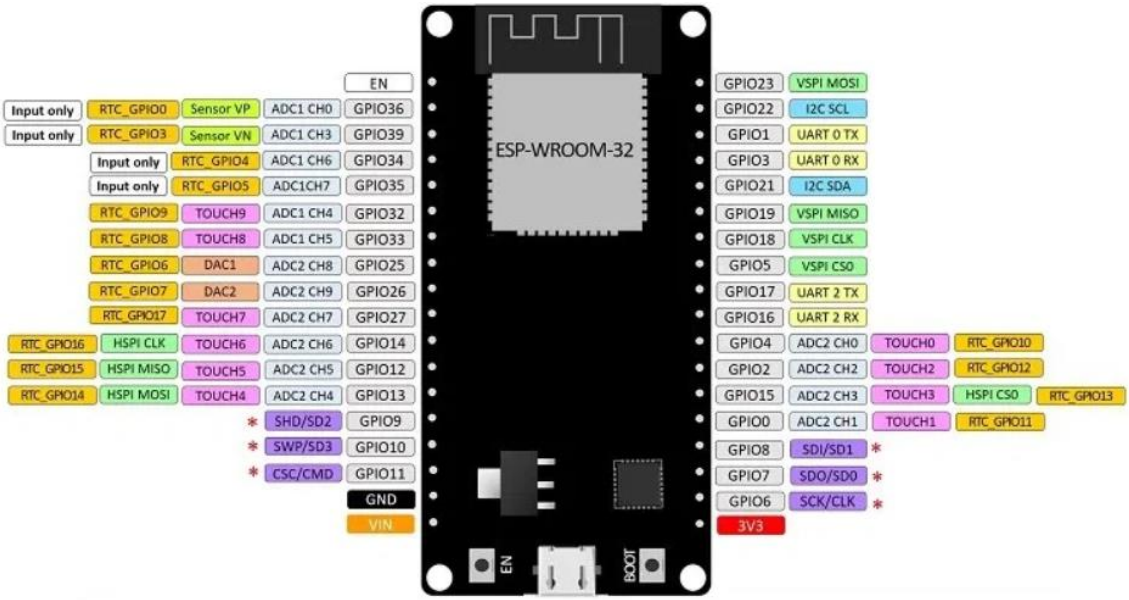
Tabela 1 - Comparativo entre os microcontroladores.

	ESP8266	ESP32	Arduino UNO R3
Corrente	197 mA	220 mA	40 mA
Núcleo	1	2	1
Arquitetura	32 bits	32 bits	8 bits
Clock	80 – 160Mhz	160– 240Mhz	16Mhz
WiFi	SIM	SIM	Não
Bluetooth	Não	SIM Clássico e BLE	Não
RAM	160KB	520KB	2KB
FLASH	16Mb	16Mb	32KB
GPIO	11	22	12
DAC	0	2	0
ADC	1	18	6

Fonte: Fvml (2021).

Na figura 6, é mostrado o ESP32 e todas as suas portas, as GPIO (*General-Purpose Input/Output*).

Figura 6 - Microcontrolador ESP32 e suas portas GPIO

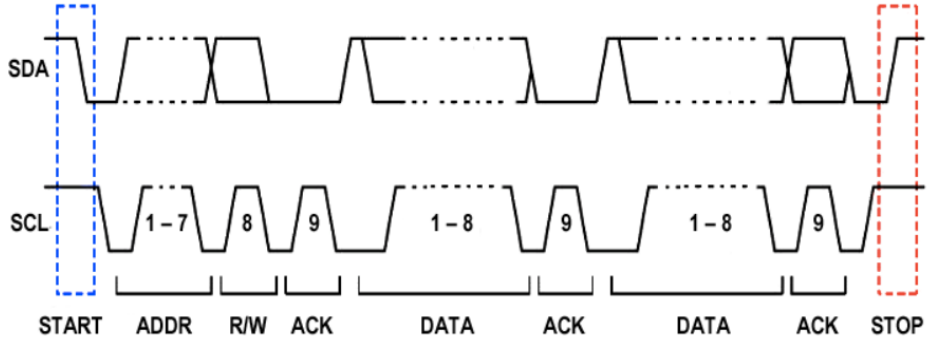


Fonte: RandomNerd Tutorials (2021).

As portas têm diferentes funções, podem receber dados analógicos ou digitais de sensores ou enviar informações para atadores através das portas PWM (*Pulse Width Modulation*). Entretanto, um conjunto de portas pode oferecer diferentes tipos de protocolos de comunicação, como por exemplo, I²C (*Inter-Integrated-Circuit*) e SPI (*Serial Peripheral Interface*) (KURNIAWAN, 2019).

- I²C, é um barramento de comunicação serial *multi-master*, *multi-slave*, síncrono e bidirecional. São necessárias apenas duas linhas de comunicação; uma linha de dados serial (SDA) e uma linha de *clock* serial (SCL). Cada dispositivo conectado ao barramento é endereçado pelo software do microcontrolador, por um endereço exclusivo e com relações *master*/*slave*; os *masters* e os *slaves* operam como transmissores e receptores de dados, entretanto somente os *masters* podem controlar a comunicação. Esse tipo de protocolo tem a principal vantagem de poder conectar até 127 dispositivos (sensores, telas LCD, entre outros) na mesma linha de dados (ROBOT ELECTRONICS, 2021). No caso do ESP32 as duas portas responsáveis por esse protocolo de comunicação são a GPIO21 (SDA) e a GPIO22 (SCL). A figura 7, ilustra como funciona a comunicação I²C.

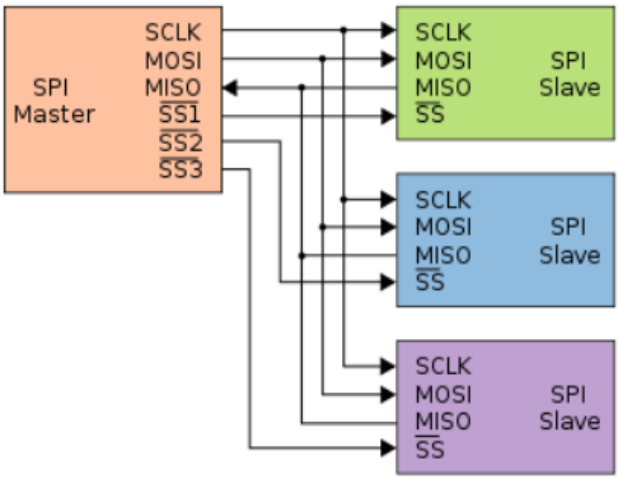
Figura 7 - Diagrama de comunicação I²C.



Fonte: Twilio Inc. (2021).

- SPI, originalmente criado pela Motorola, nos anos 1980, é um barramento de interface serial, síncrono, que possui apenas um único *master* e *multi-slaves*, normalmente usados para comunicação serial entre sistemas de microcomputadores, microcontroladores e outros dispositivos, como por exemplo memórias (Cartões SD, memória EEPROM), também pode ser utilizado para conectar telas LCDs e sensores. Esse protocolo de comunicação é muito similar a um processo de leitura e gravação, por isso ele é muito utilizado em dispositivos que irão fazer os mesmos processos (MAGDY, 2020). No ESP32, as portas responsáveis pela comunicação SPI são a GPIO23 (MOSI), GPIO19 (MISO), GPIO18 (CLK) e GPIO5 (CS). Abaixo, na figura 8, é ilustrada a comunicação SPI entre o dispositivo *master* (geralmente é o microcontrolador) com os dispositivos *slaves* (Sensores, memórias, LCDs, entre outros).

Figura 8 - Comunicação SPI entre *master* e *slaves.*



Fonte: Mendonça (2021)

2.4 Protocolo de segurança TLS

TLS (*Transport Layer Security*), é um protocolo de segurança amplamente adotado, projetado para facilitar a privacidade e a segurança de dados para comunicações pela Internet. Um caso de uso primário de TLS é criptografar a comunicação entre aplicativos da web e servidores, como navegadores que carregam um site. Esse protocolo pode ser usado para criptografar outras comunicações, como e-mail, mensagens, comunicação entre dispositivos e voz sobre IP (VoIP) (INTERNET SOCIETY, 2021).

O TLS evoluiu de um protocolo de criptografia anterior, denominado *Secure Sockets Layer* (SSL), que foi desenvolvido pela Netscape. A versão 1.0 TLS começou a ser desenvolvido a partir da versão 3.1 da SSL, mas o nome do protocolo foi alterado antes da publicação para indicar que não estava mais associado a Netscape. Por causa desse histórico, os termos TLS e SSL às vezes são usados alternadamente (INTERNET SOCIETY, 2021).

Segundo a Cloudflare (2021), o protocolo TLS tem duas finalidades:

- proteger as comunicações de rede entre duas partes
- estabelecer a identidade da parte que serve à outra parte um certificado.

As conexões TLS dependem da existência de um par de chaves, consiste em uma chave privada e uma chave pública. Essas duas chaves estão relacionadas entre si por meio de um algoritmo criptográfico. A chave privada é “privada” para o servidor que recebe a conexão TLS e deve ser mantida em segredo. O servidor se apresenta ao cliente entregando seu certificado.

O certificado é um contêiner assinado (‘*certified*’) que inclui a chave pública do servidor. Para garantir a autenticidade da chave pública contida em um certificado, a chave deve ser assinada por um terceiro confiável, uma Autoridade de Certificação (CA) (CLOUDFLARE, 2021).

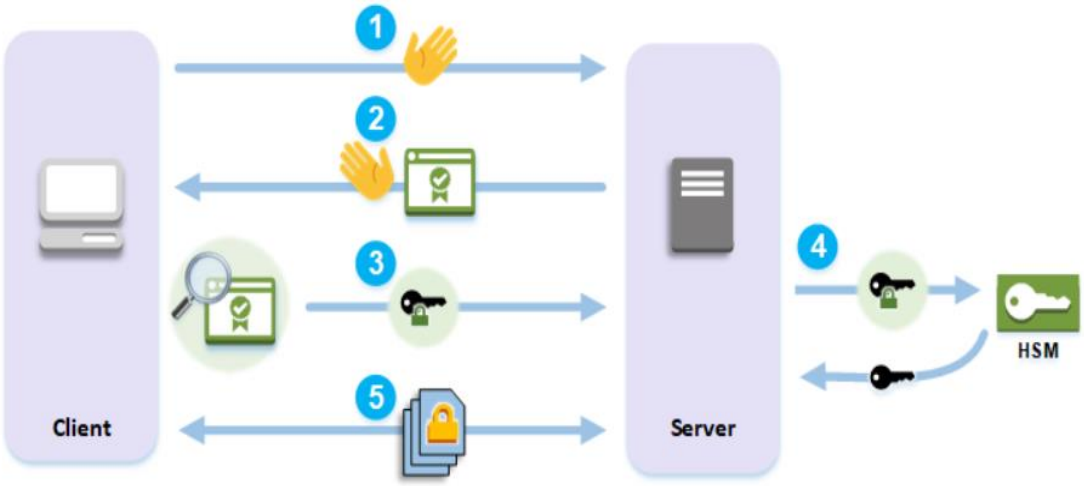
Segundo Amazon (2021), o funcionamento pratico do TLS é:

1 O cliente envia uma mensagem para o servidor.
2. O servidor responde com uma outra mensagem e envia o certificado do servidor.
3. O cliente realiza as seguintes ações:
    - Verifica se o certificado do servidor SSL/TLS está assinado por um certificado raiz em que o cliente confia.
    - Extrai a chave pública do certificado do servidor.
    - Gera um segredo pré-mestre e o criptografa com a chave pública do servidor.
    - Envia o segredo pré-master codificado para o servidor.
4. Para descriptografar o segredo pré-master do cliente, o servidor o envia ao HSM (*Hardware Security Module*). O HSM usa a chave privada no HSM para descriptografar o segredo pré-master e, em seguida, envia o segredo pré-master ao servidor. Independentemente, o cliente e o servidor usam o segredo pré-master e algumas informações das mensagens para calcular um segredo mestre.

5. O processo de *handshake* termina. Para o resto da sessão, todas as mensagens enviadas entre o cliente e o servidor são criptografadas com derivadas do segredo mestre.

Na figura 9, é ilustrado com é o funcionamento pratico do TLS.

Figura 9 - Fluxo do protocolo TLS



Fonte: Amazon (2021).

1.1 Protocolo MQTT

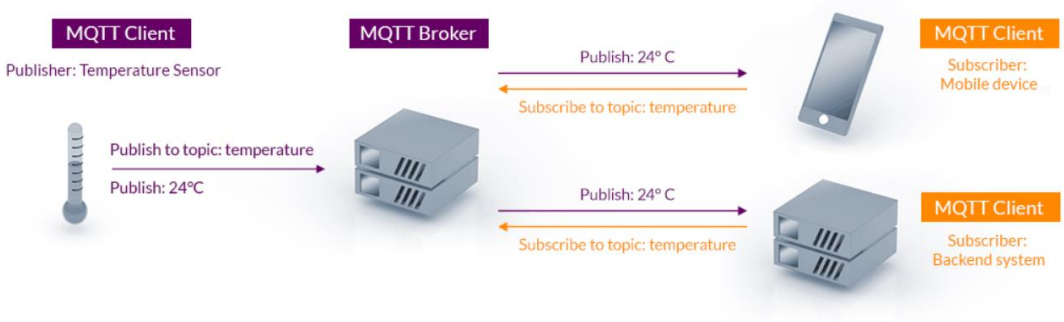
O Protocolo MQTT (*Message Queuing Telemetry Transport*), foi projetado para ser executado em um ambiente integrado, para dispositivos que tenham baixa capacidade de processamento e memória, onde possa fornecer um caminho confiável e eficaz para a comunicação. O protocolo tem aplicações em setores que vão desde automotivo até energia e telecomunicações. Embora tenha começado como um protocolo proprietário usado para se comunicar com sistemas de controle de supervisão e aquisição de dados na indústria de petróleo e gás, ele se tornou popular na área de dispositivos inteligentes e hoje é o protocolo de código aberto líder para conectar à Internet das Coisas (IoT) e dispositivos industriais IoT (IIoT) (BERNSTEIN et al., 2020).

Uma conexão MQTT é dividida em quatro estágios: conexão, autenticação, comunicação e encerramento. É criado uma conexão TCP / IP (*Transmission Control Protocol/Internet Protocol*) com o *broker* usando um porta padrão, a 8883, para comunicação criptografada usando TLS. A autenticação é feita quando as duas partes (cliente e servidor) validam os certificados e chaves privadas e públicas. No momento da conexão, ocorre o envio ou recebimento dos dados. O encerramento da conexão é feito caso uma das partes envie um comando ‘DISCONNECT’ (BERNSTEIN et al., 2020).

O funcionamento do protocolo MQTT é baseado no modelo Pub/Sub, é necessário ter no mínimo dois agentes, um *client* e um *broker*. O *client* pode enviar (Pub) e/ou receber (Sub), entretanto, um *client* pode se conectar somente a um *broker*. Já *broker* tem a função de

autenticar e validar os dispositivos conectados a ele e, também, rotear (Pub) esses dados para outros *clients* (BERNSTEIN et al., 2020). Na figura 10, está ilustrado o funcionamento do protocolo MQTT.

Figura 10 - Arquitetura do protocolo MQTT.



Fonte: Mqtt (2021).

2 MATERIAIS E MÉTODOS

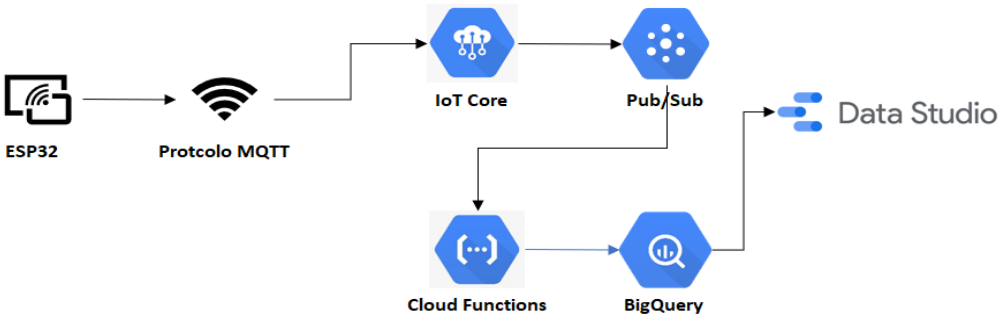
Os materiais utilizados para o projeto foram:

- BME280, sensor responsável por fazer a leitura das variáveis ambientais coletadas (temperatura, pressão e umidade).
- Jumpers sólidos, têm a função de fazer a ligação entre o sensor e o microcontrolador.
- Protoboard, placa de prototipação rápida responsável por fixar o microcontrolador, sensor e os jumpers.
- ESP32, microcontrolador responsável por receber as informações coletadas pelo sensor e enviá-las para o GCP (*Google Cloud Platform*) através do protocolo MQTT.

A metodologia utilizada para o desenvolvimento desse projeto, primeiramente foi construir o hardware, depois configurar os serviços dentro do GCP, que foram responsáveis pelo poder computacional e telemetria do projeto. Com a comunicação estabelecida entre o dispositivo e o GCP, os dados foram inseridos no banco de dados na nuvem, após esta etapa, os dados passaram a ser enviados ao *Data Studio*, plataforma responsável por fazer a análise dos dados que estão dentro do GCP.

A figura 11, mostra o fluxo de dados dentro do sistema criado:

Figura 11 - Fluxo do funcionamento do projeto.

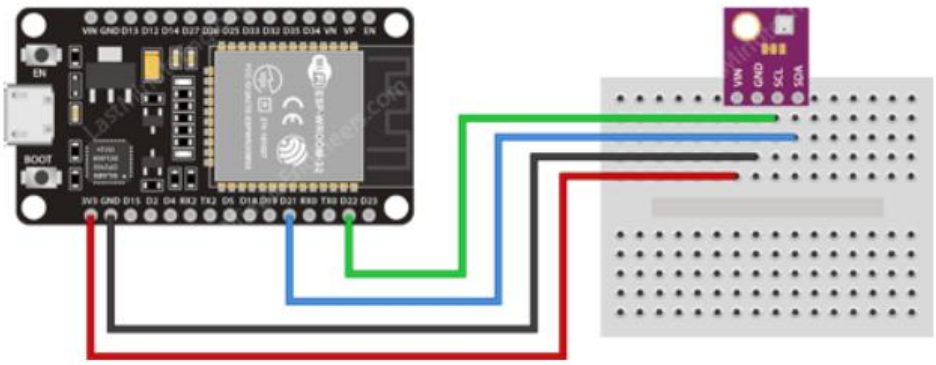


Fonte: Próprio autor.

2.1 Montagem do hardware

Na montagem do hardware, foi utilizado a comunicação I²C, os pinos D21 e D22 do ESP32 foram ligados aos pinos SDA e SCL do sensor respectivamente. Vale ressaltar que tanto o ESP32 quanto o sensor BME280, usam a mesma voltagem de 3.3V. Então, pode-se ligar os pinos do ESP32 3.3V e GND ao Vin e GND do sensor respectivamente, conforme a figura 12.

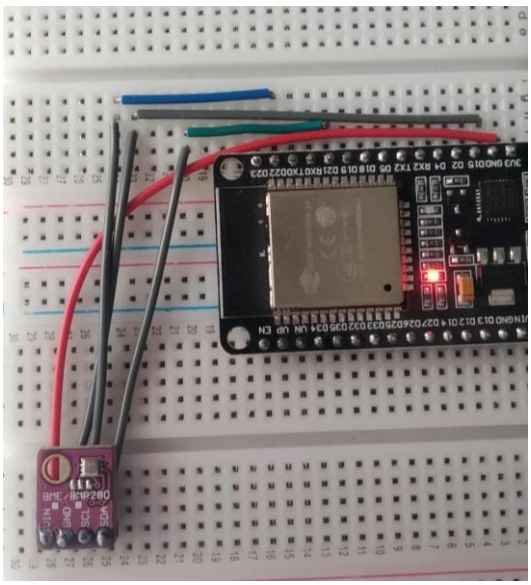
Figura 12 - Esquema de ligação entre os pinos



Fonte: Engineers (2021).

A figura 13 mostra o resultado final da montagem do hardware.

Figura 13 - Protótipo do dispositivo.



Fonte: Próprio autor.

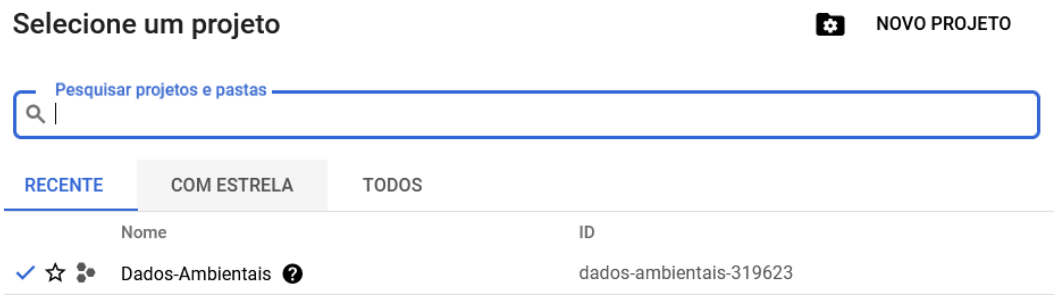
2.2 Introdução ao ambiente prático do *Google Cloud Platform*

O GCP (*Google Cloud Platform*), oferece cerca de $300.00 para testes em sua plataforma, que em uma conversão direta com a cotação do dia 12/07/2021, R$5.17 (B3, 2021), é de aproximadamente R$1.551,00, o período de teste pode ser utilizado durante 90 dias.

Para ter acesso a plataforma, basta criar um *login* no 'GSuite' do Google, que oferece diversos produtos gratuitos para usar, como por exemplo, o GMail.

Após a conta criada, foi feito um cadastro no ambiente do GCP, através do link: https://cloud.google.com. A partir desta etapa, criou-se um projeto, conforme a figura 14.

Figura 14 - tela para criar um novo projeto da plataforma GCP.

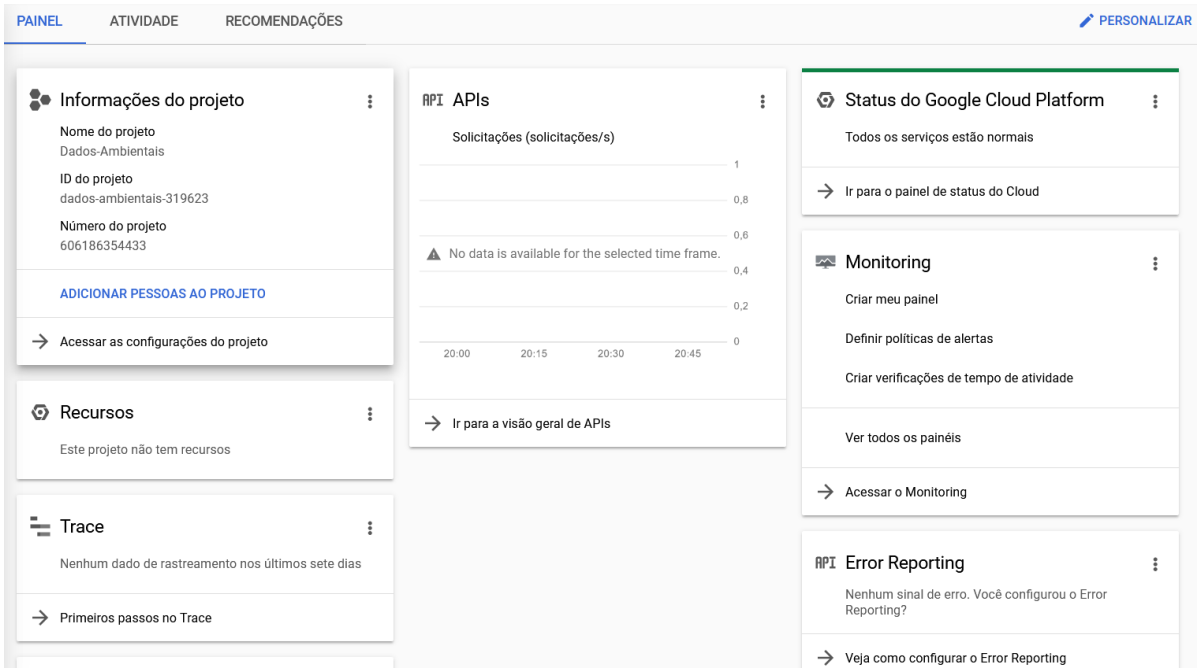


Fonte: Próprio autor.

O nome do projeto escolhido foi: Dados-Ambientais. Uma vez criado o projeto, o nome não poderá ser alterado, caso deseje alterar o nome, um novo projeto deverá ser construído e, todo o legado do projeto anterior será perdido.

Na figura 15 está a tela inicial do projeto, com diversas informações.

Figura 15 - tela com informações do projeto GCP.



Fonte: Próprio autor.

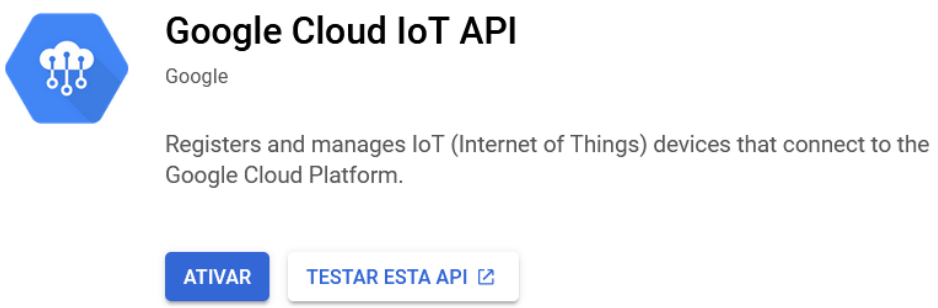
2.3 Estruturando o projeto no *Google Cloud Platform* para receber os dados

A suíte do GCP é vasta, oferece desde produtos para armazenagem de dados até produtos de inteligência artificial. Nesse projeto, foram utilizados os produtos: IoT *Core*, Pub/Sub, *Cloud Functions* e *BigQuery* (GOOGLE, 2021).

- IoT Core, tem a função de gerenciar dispositivos externos que podem se conectar ao projeto no GCP.
- Pub/Sub, tem a função de integrar os dados enviados pelos dispositivos no GCP.
- *Cloud Functions*, oferece a possibilidade de criar códigos em diversas linguagens de programação e tem diversas funções dentro do GCP, no caso desse projeto, a ferramenta foi utilizada para criar um código em python que faz o parser das mensagens recebidas pelo Pub/Sub e insere as mensagens no *BigQuery*.
- *BigQuery*, tem a função de banco de dados, ou seja, ele armazena todos os dados enviados pelo microcontrolador.

Para utilizar os serviços, é necessário ativá-los. Após serem ativados e à medida que o serviço é utilizado, é iniciado a cobrança, seja por tempo de uso, processamento de dados ou tráfego de dados na rede, isso varia de produto a produto. A figura 16, mostra como se ativa um serviço.

Figura 16 - Ativação do IoT *Core* no GCP

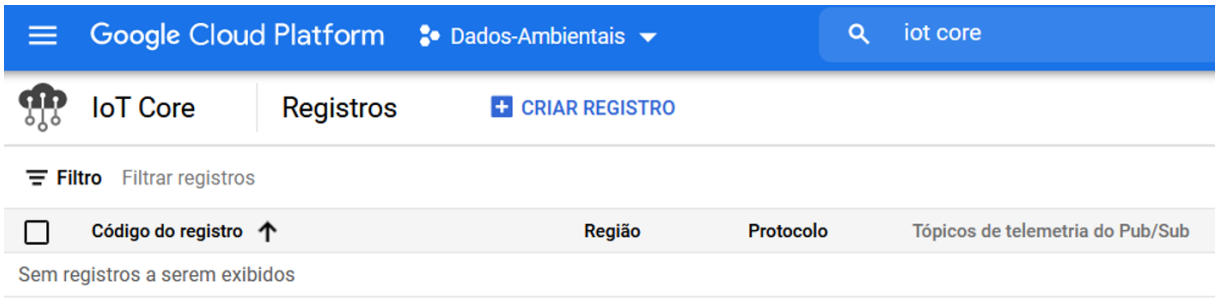


Fonte: Próprio autor.

O primeiro produto ativado foi IoT *Core*, pois durante a configuração desse serviço, o Pub/Sub é ativado automaticamente.

Após essa etapa, aparecerá uma tela para a criação de um registro (figura 17).

Figura 17 - Criar registro IoT Core.



Fonte: Próprio autor.

A criação do registro foi intuitiva, foi necessário criar um ID e escolher a região onde os registros serão armazenados. Nesse momento, também foi configurado um tópico Pub/Sub, mostrado na figura 18.

Figura 18 - Menu de criação de registro e tópicos Pub/Sub.

Fonte: Próprio autor.

Após a criação do registro, a próxima etapa foi criar um dispositivo dentro da plataforma, conforme a figura 19.

Figura 19 - Criação de dispositivo no IoT *Core*.

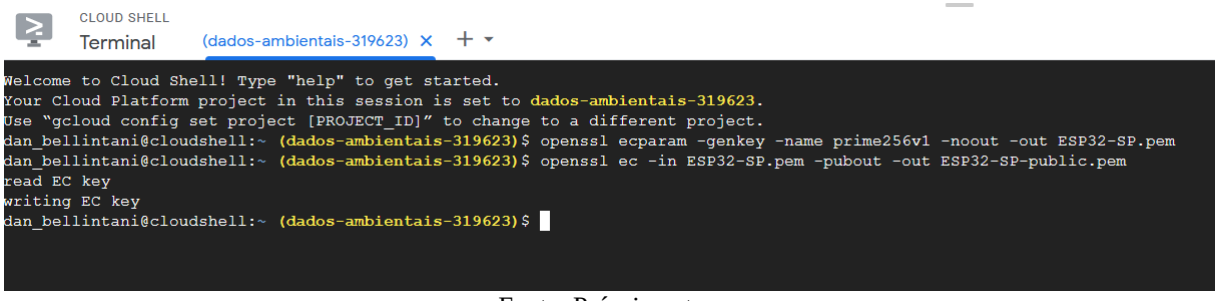
Fonte: Próprio autor.

Em seguida, o próximo passo foi criar um par de chaves públicas e privadas, usando os seguintes códigos abaixo:

```
openssl ecparam -genkey -name prime256v1 -noout -out [Nome Do dispositivo].pem

openssl ec -in [Nome Do dispositivo].pem -pubout -out [Nome Do dispositivo]-public.pem
```

Os códigos foram colocados no *Cloud Shell*, conforme a figura 20.

Figura 20 - Exemplo de como inserir o código no *cloud shell*.

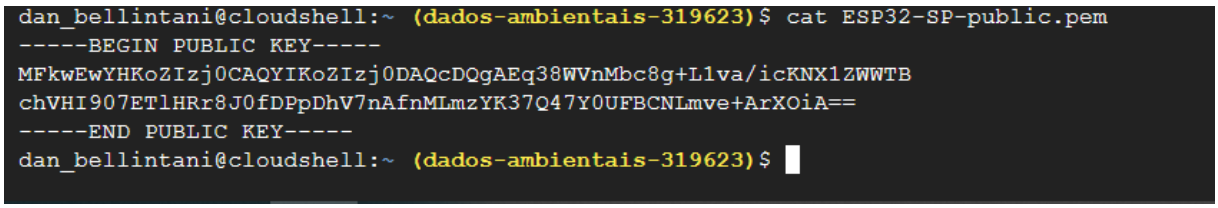


Fonte: Próprio autor.

Uma vez que os códigos foram inseridos, as chaves foram geradas dentro do servidor do GCP. Então, usou-se o código abaixo para capturar a chave pública, a figura 21 exemplifica essa parte.

```
cat [Nome Do dispositivo]-public.pem
```

Figura 21 - Como capturar a chave pública do servidor.



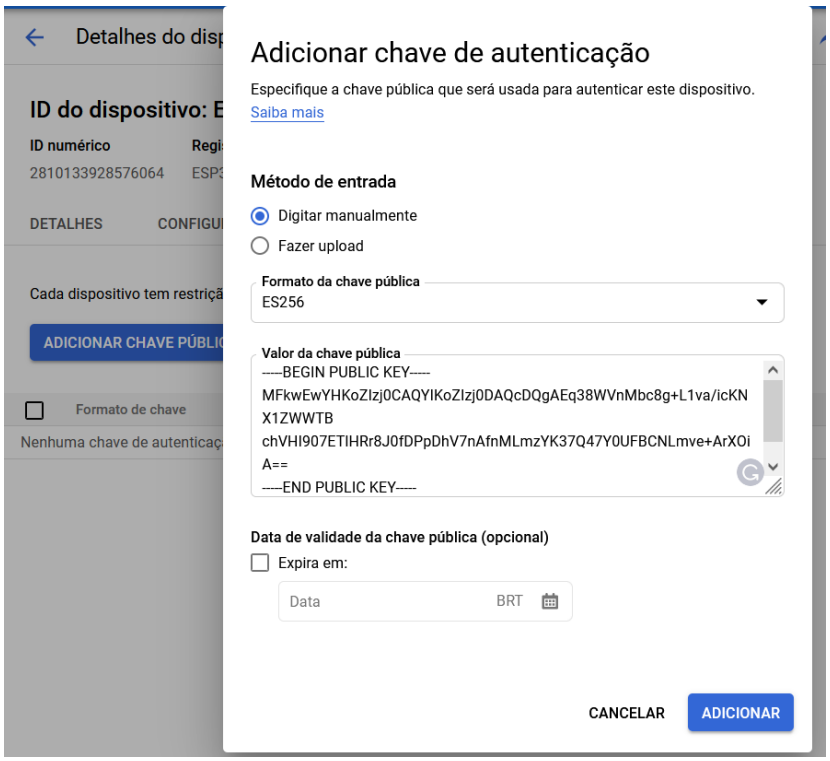
Fonte: Próprio autor.

A *public key* gerada foi copiada e guardada para uso posterior. Observe que para gerar uma chave é necessário ter o nome do dispositivo.

```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEq38WVnMbc8g+L1va/icKNX1ZWWTB
chVHI907ETlHRr8J0fDPpDhV7nAfnMLmzYK37Q47Y0UFBCNLmve+ArXOiA==
-----END PUBLIC KEY-----
```

A chave foi adicionada ao dispositivo, conforme a figura 22.

Figura 22 - Exemplo de como inserir o a chave pública.

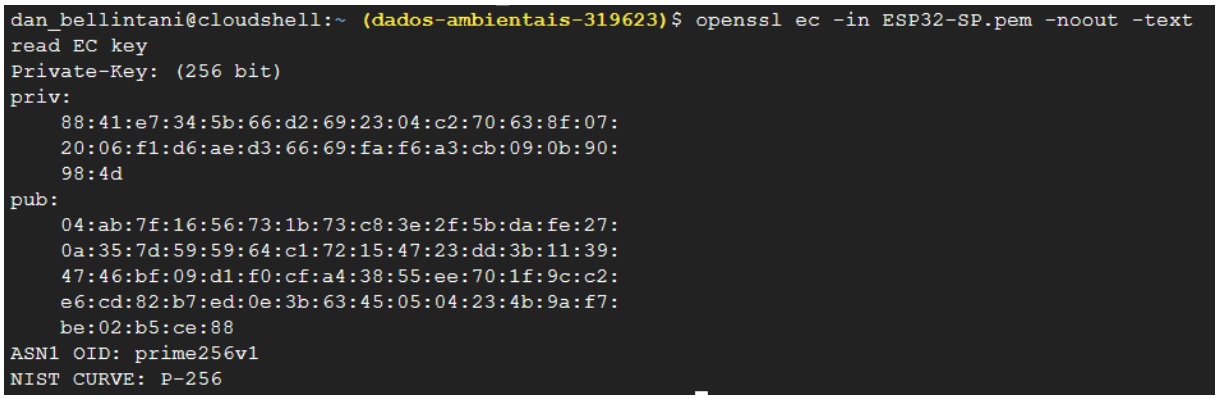


Fonte: Próprio autor.

Após a etapa anterior, foi gerado a chave privada e o certificado CA. No *Cloud Shell*, foi inserido o seguinte código para obter a *private key*, a imagem 23 mostra o que deverá ser retornado.

```
openssl ec -in [Nome do Dispositivo].pem -noout -text
```

Figura 23 - Exemplo de como inserir o código para obter a *private key*.



Fonte: Próprio autor.

Abaixo está a chave pública e privada que o código retornou.

```
priv:
    88:41:e7:34:5b:66:d2:69:23:04:c2:70:63:8f:07:
    20:06:f1:d6:ae:d3:66:69:fa:f6:a3:cb:09:0b:90:
    98:4d
pub:
    04:ab:7f:16:56:73:1b:73:c8:3e:2f:5b:da:fe:27:
    0a:35:7d:59:59:64:c1:72:15:47:23:dd:3b:11:39:
    47:46:bf:09:d1:f0:cf:a4:38:55:ee:70:1f:9c:c2:
    e6:cd:82:b7:ed:0e:3b:63:45:05:04:23:4b:9a:f7:
    be:02:b5:ce:88
```

Já para o certificado CA, usou-se o seguinte código abaixo:

```
openssl s_client -showcerts -connect mqtt.2030.ltsapis.goog:8883
```

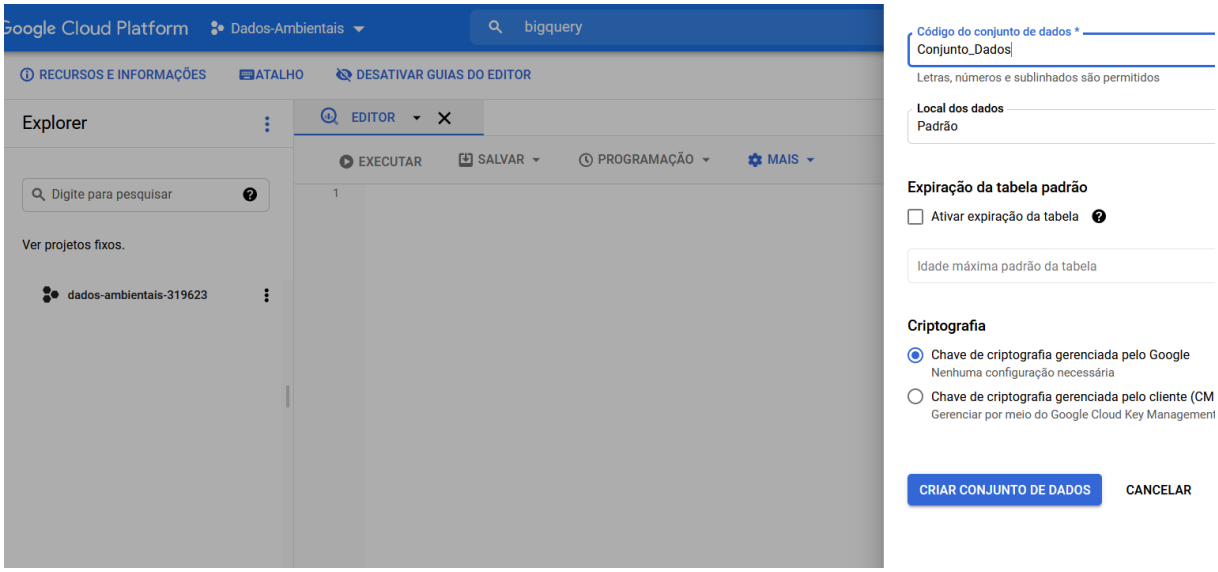
O código acima retornou dois certificados CA:

```
-----BEGIN CERTIFICATE-----
MIIDDDCCArKgAwIBAgIUXIRd61ARosjr5tpYAQK1udlptnswCgYIKoZIzj0EAwIw
RDELMAkGA1UEBhMCVVMxIjAgBgNVBAoTGUdvb2dsZSBUcnVzdCBTZXJ2aWNlcyBM
TEMxETAPBgNVBAMTCEdUUyBMVFNYMB4XDTIxMDUyNTAwMDAwMFoXDTIyMDUyNDAw
MDAwMFowbTELMAkGA1UEBhMCVVMxEzARBgNVBAgMCkNhbGlmb3JuaWExFjAUBgNV
BAcMDU1vdW50YWluIFZpZXcxEzARBgNVBAoMCkdvb2dsZSBMTEMxHDAaBgNVBAMM
EyouMjAzMC5sdHNhcGlzLmdvb2cwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAARR
VX2FBT6/ZCFlCwom7Pr7jtlh99RHfH0cxO51PZ0gifi8mo2UasKfsw0ikuZvaEkG
busnKgGwa6TrBElBabLNo4IBVzCCAVMwEwYDVR0lBAwwCgYIKwYBBQUHAwEwDgYD
VR0PAQH/BAQDAgeAMB4GA1UdEQQXMBWCEyouMjAzMC5sdHNhcGlzLmdvb2cwDAYD
VR0TAQH/BAIwADAfBgNVHSMEGDAWgBSzK6ugSBx+E4rJCMRAQiKiNlHiCjBpBggr
BgEFBQcBAQRdMFswLwYIKwYBBQUHMAKGI2h0dHA6Ly9wa2kuZ29vZy9ndHNsdHNy
L2d0c2x0c3guY3J0MCgGCCsGAQUFBzABhhxodHRwOi8vb2NzcC5wa2kuZ29vZy9H
VFNMVFNYMCEGA1UdIAQaMBgwDAYKKwYBBAHWeQIFAzAIBgZngQwBAgIwMAYDVR0f
BCkwJzAloCOgIYYfaHR0cDovL2NybC5wa2kuZ29vZy9HVFNMVFNYLmNybDAdBgNV
HQ4EFgQUxp0CLjzIieJCqFTXjDc9okXUP80wCgYIKoZIzj0EAwIDSAAwRQIgAIuJ
1QvJqFZwy6sZCP1+dXOX4YTWAbum6FtqyJwOKIACIQDENBALkXPS9jo0g8X5+eT9
MlOQcPMMtbXGtK/ENpE2rw==
-----END CERTIFICATE-----


-----BEGIN CERTIFICATE-----
MIIC0TCCAnagAwIBAgINAfQKmcm3qFVwT0+3nTAKBggqhkjOPQQDAjBEMQswCQYD
VQQGEwJVUzEiMCAGA1UEChMZR29vZ2xlIFRydXN0IFNlcnZpY2VzIExMQzERMA8G
A1UEAxMIR1RTIExUU1IwHhcNMTkwMTIzMDAwMDQyWhcNMjkwNDAxMDAwMDQyWjBE
MQswCQYDVQQGEwJVUzEiMCAGA1UEChMZR29vZ2xlIFRydXN0IFNlcnZpY2VzIExM
QzERMA8GA1UEAxMIR1RTIExUU1gwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAARr
6/PTsGoOg9fXhJkj3CAk6C6DxHPnZ1I+ER40vEe290xgTp0gVplokojbN3pFx07f
zYGYAX5EK7gDQYuhpQGIo4IBSzCCAUcwDgYDVR0PAQH/BAQDAgGGMB0GA1UdJQQW
MBQGCCsGAQUFBwMBBggrBgEFBQcDAjASBgNVHRMBAf8ECDAGAQH/AgEAMB0GA1Ud
DgQWBBSzK6ugSBx+E4rJCMRAQiKiNlHiCjAfBgNVHSMEGDAWgBQ+/v/MUuu/ND49
80DQ5CWxX7i7UjBpBggrBgEFBQcBAQRdMFswKAYIKwYBBQUHMAGGHGh0dHA6Ly9v
Y3NwLnBraS5nb29nL2d0c2x0c3IwLwYIKwYBBQUHMAKGI2h0dHA6Ly9wa2kuZ29v
Zy9ndHNsdHNyL2d0c2x0c3IuY3J0MDgGA1UdHwQxMC8wLaAroCmGJ2h0dHA6Ly9j
cmwucGtpLmdvb2cvZ3RzbHRzci9ndHNsdHNyLmNybDAdBgNVHSAEFjAUMAgGBmeB
DAECATAIBgZngQwBAgIwCgYIKoZIzj0EAwIDSQAwRgIhAPWeg2v4yeimG+lzmZAC
DJOlalpsiwJR0VOeapY8/7aQAiEAiwRsSQXUmfVUW+N643GgvuMH70o2Agz8w67f
SX+k+Lc=
-----END CERTIFICATE-----
```

Os dois certificados e a chave privada devem ser salvos, pois são constantes no código do microcontrolador. Vale salientar que, para cada dispositivo, existe uma *public key*, *private key* e certificados CA diferentes.

A próxima etapa foi configurar o banco de dados no *BigQuery*. Foi necessário criar um novo conjunto de dados, os quais contêm as tabelas. Foi feito conforme a figura 24.

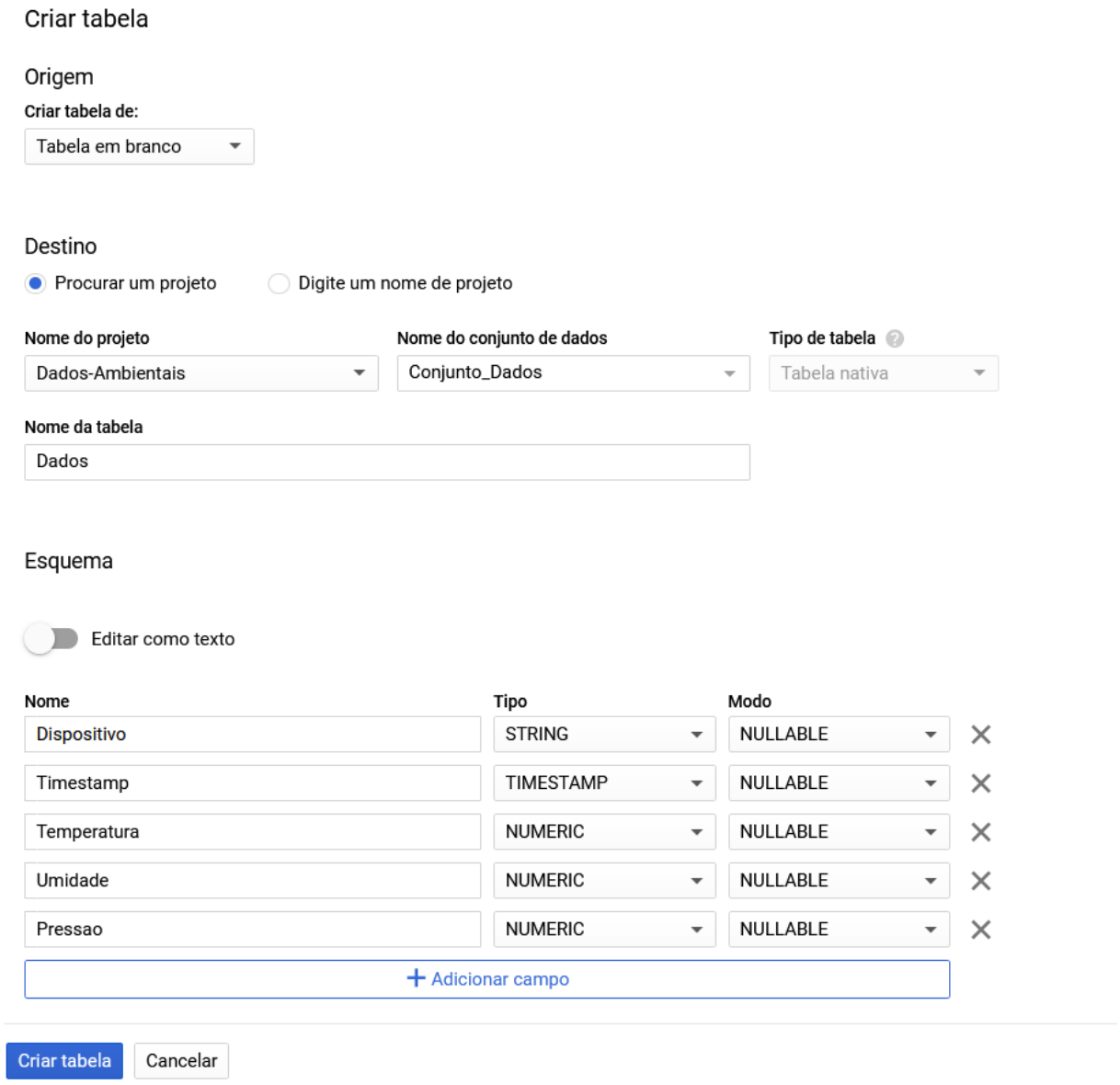
Figura 24 - Criação de um conjunto de dados no *Google Cloud*.



Fonte: Próprio autor.

No conjunto de dados, criou-se uma tabela que tinha as mesmas informações que seriam enviadas pelo dispositivo (temperatura, pressão, umidade e o nome do dispositivo), conforme a figura 25.

Figura 25 - Criando uma tabela no *BigQuery*.

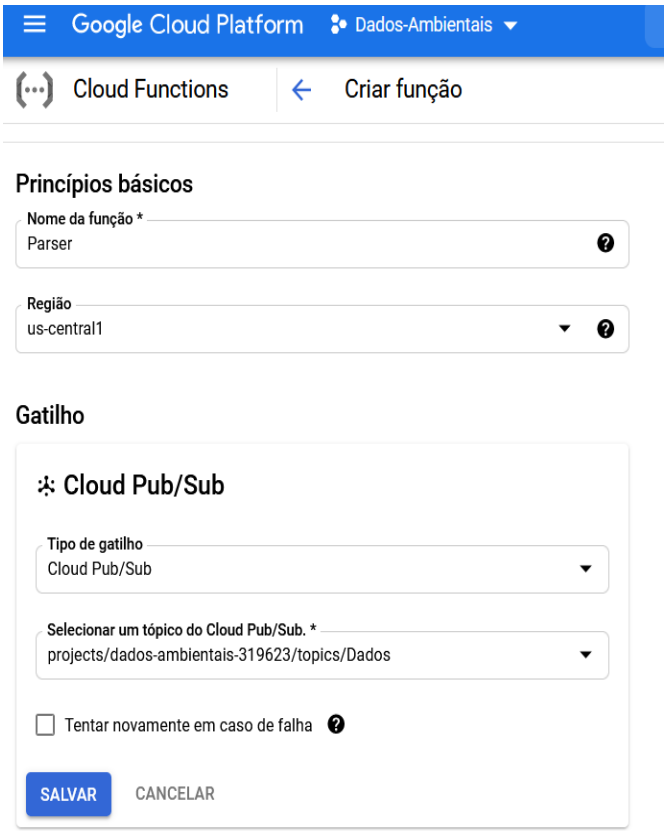


Fonte: Próprio autor.

Ao estruturar o IoT *Core*, Pub/Sub e o *BigQuery*, criou-se uma função no *Cloud Functions* que tinha como finalidade capturar os dados recebidos Pub/Sub e inserir na tabela do *BigQuery*.

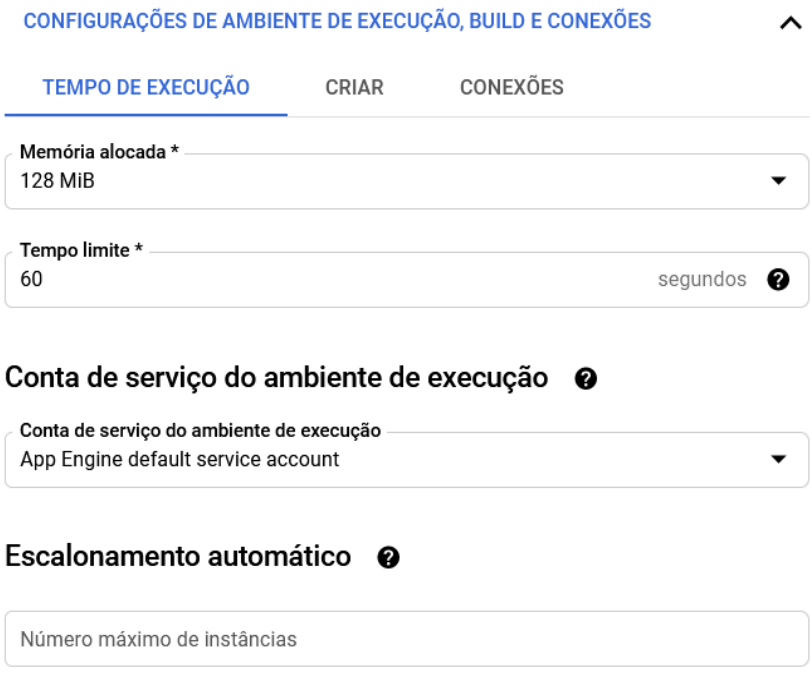
A função teve a seguinte estrutura, conforme as figuras 26, 27 e 28.

Figura 26 - Estrutura inicial da função.



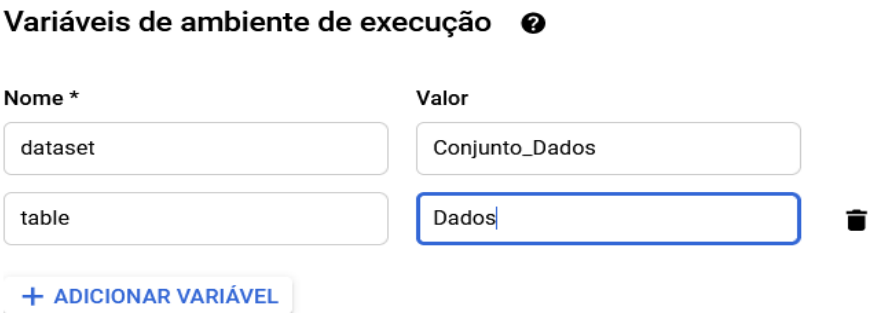
Fonte: Próprio autor.

Figura 27 - Configuração do ambiente de execução.



Fonte: Próprio autor.

Figura 28 - Criação de variáveis de execução.



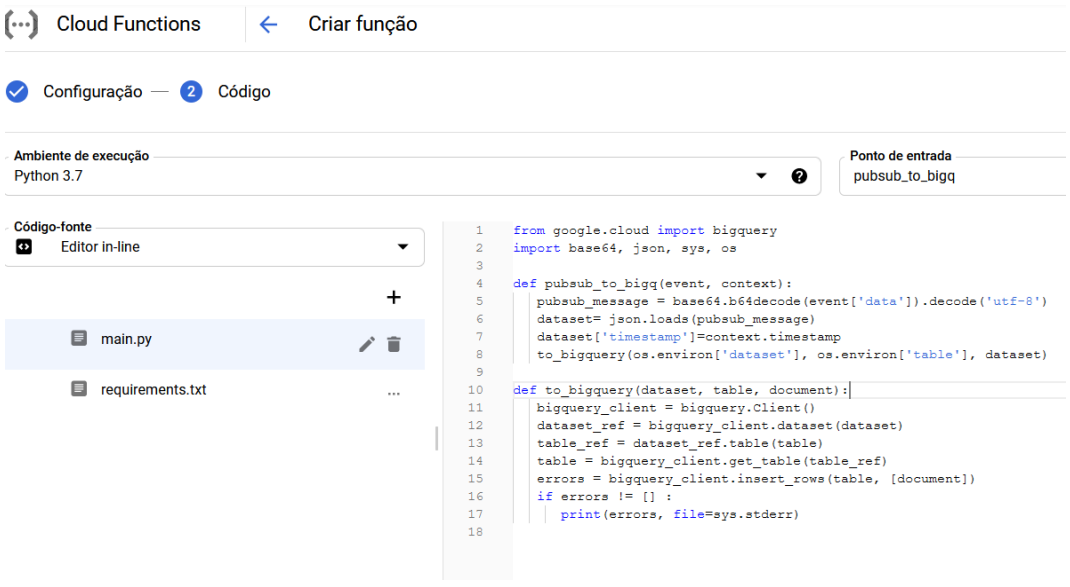
Fonte: Próprio autor.

O nome da função criada foi ‘Parser’ e o gatilho para que ela seja executada é o momento em que uma mensagem é recebida pelo Pub/Sub. Foram criadas as variáveis de ambiente de execução, que são os mesmos nomes atribuídos ao conjunto de dados (*dataset*) e a tabela do BigQuery.

Após ser salvo, foi ativado o *Cloud Build* (API responsável por fornecer as bibliotecas do GCP). Foi selecionado o ambiente de execução ‘Python 3.7’ e criado o ponto de entrada (`pubsub_to_bigq`). Uma vez selecionado o ambiente de execução, foram criados dois arquivos dentro do código-fonte, um arquivo main.py e outro requirements.txt. O arquivo com a extensão .py contém a função escrita em Python, responsável por inserir as informações no banco de dados e o arquivo .txt, contém as bibliotecas necessárias para que o *script* Python funcione.

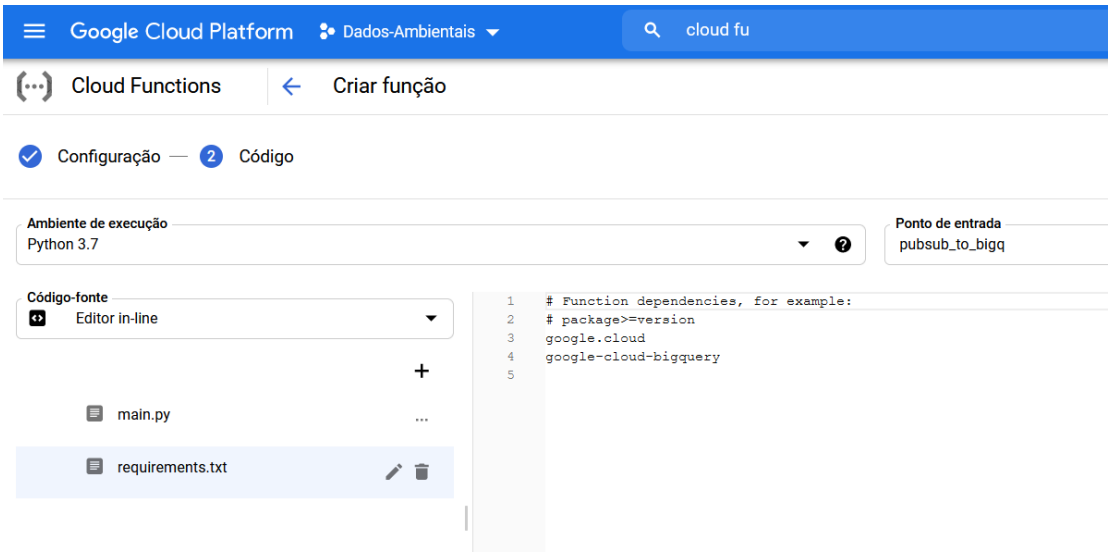
Nas figuras 29 e 30, estão mostrados como ficaram os arquivos.

Figura 29 - Estrutura main.py.



Fonte: Próprio autor.

Figura 30 - Estrutura requirements.txt.



Fonte: Próprio autor.

Abaixo, estão os códigos utilizados para melhor entendimento.

- main.py:

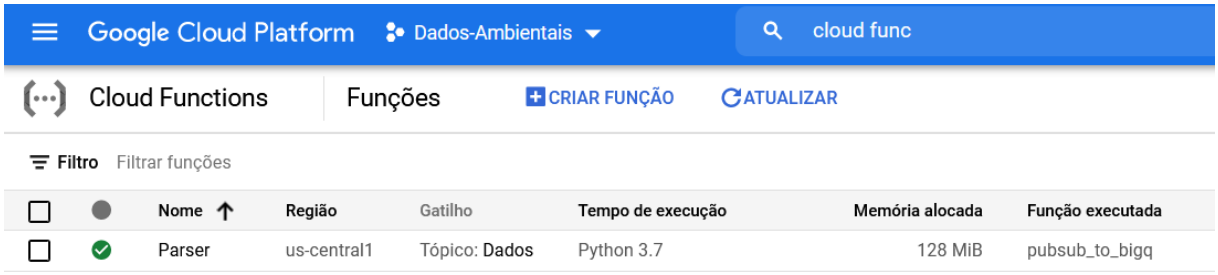
```
from google.cloud import bigquery
import base64, json, sys, os

def pubsub_to_bigq(event, context):
   pubsub_message = base64.b64decode(event['data']).decode('utf-8')
   dataset= json.loads(pubsub_message)
   dataset['timestamp']=context.timestamp
   to_bigquery(os.environ['dataset'], os.environ['table'], dataset)

def to_bigquery(dataset, table, document):
   bigquery_client = bigquery.Client()
   dataset_ref = bigquery_client.dataset(dataset)
   table_ref = dataset_ref.table(table)
   table = bigquery_client.get_table(table_ref)
   errors = bigquery_client.insert_rows(table, [document])
   if errors != [] :
      print(errors, file=sys.stderr)
```

- requeriments.txt:

```
google.cloud
google-cloud-bigquery
```

A última etapa, foi clicar no botão implementar, nesse momento criou-se uma máquina virtual para rodar o script. Na figura 31, é possível ver o resultado final. Nesse ponto, o ambiente no GCP já está pronto para receber os dados do dispositivo.

Figura 31 - Função em Python sendo executada dentro do Cloud Functions.



Fonte: Próprio autor.

2.4 Software do microcontrolador ESP32

Para realizar o desenvolvimento do *software*, foi usado IDE do Arduino e o Microsoft Visual Studio, lembrando que a IDE do Arduino deve estar configurada para que possa aceitar o microcontrolador ESP32.

A construção do software do ESP32 foi composta pelo código principal e as bibliotecas auxiliares, que têm o objetivo de facilitar a vida dos programadores, oferecendo funções já prontas ou, no caso analisado a seguir, “quase prontas”, pois foi necessário fazer modificações.

As linguagens de programação que foram utilizadas no projeto são: C/C + + (com extensões .h e .cpp) e Wiring (.ino).

Bibliotecas que foram necessárias no projeto são:

- Arduino.h, Wire.h, Client.h, WiFi.h, são bibliotecas padrão, responsáveis por oferecer as funções mais básicas e para criar a conexão com a internet.
- WiFiClientSecure.h, responsável pelo envio e validação dos certificados CA.
- MQTT.h, biblioteca que oferece as funções de comunicação via protocolo MQTT.
- CloudIoTCore.h, CloudIoTCoreMqtt.h, o GCP, criou as bibliotecas para que terceiros conseguissem se conectar à plataforma.
- Adafruit_Sensor.h, Adafruit_BME280.h, responsáveis por fornecerem as funções que retornam a leitura do sensor.
- ArduinoJson.h, tem funções para construir e estruturar dados do tipo JSON (JavaScript Object Notation), utilizados no projeto.

No Github do GCP, foi feito download da biblioteca ‘google-cloud-iot-arduino’, através do link: https://github.com/GoogleCloudPlatform/google-cloud-iot-arduino.

Após o download ser concluído e a biblioteca ser devidamente instalada, foi necessário fazer alterações em alguns arquivos para que essa biblioteca funcionasse corretamente. As alterações foram feitas com muito cuidado, pois qualquer alteração sem nenhum conhecimento poderia fazer com que o código não funcione adequadamente.

Com o programa Visual Studio Community, oferecido gratuitamente pela Microsoft, foram feitas as alterações nos códigos das bibliotecas: ‘CloudIoTCoreMqtt.h’ e ‘CloudIoTCoreMqtt.h’.

No arquivo ‘CloudIoTCoreMqtt.h’, foi necessário adicionar a biblioteca ‘WiFiClientSecure.h’ e substituir uma variável de uma classe, a ‘class CloudIoTCoreMqtt’, de ‘Client*_netClient’, para ‘WiFiClientSecure *_netClient’, essa alteração ficou conforme a figura 31.

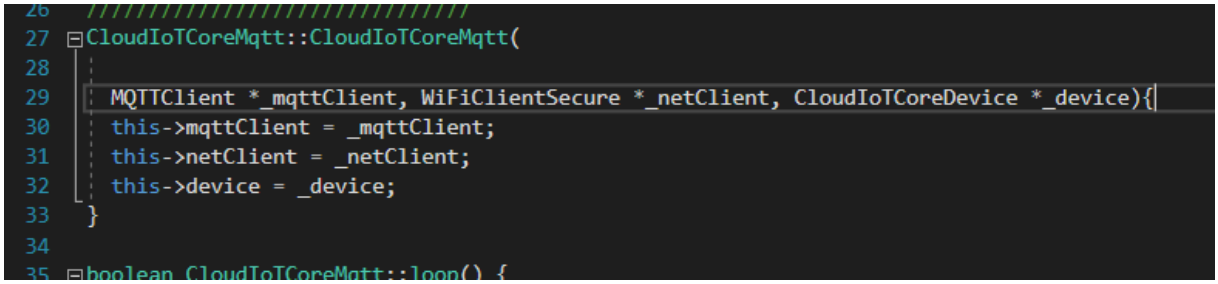
Figura 31 - CloudIoTCoreMqtt.h, resultado final.

```
15  #ifndef __CLOUDIOTCORE_MQTT_H__
16  #define __CLOUDIOTCORE_MQTT_H__
17  #include <Arduino.h>
18  #include "CloudIoTCore.h"
19  #include "CloudIoTCoreDevice.h"
20  #include <Client.h>
21  #include <MQTTClient.h>
22  #include <WiFiClientSecure.h>
23
24  class CloudIoTCoreMqtt {
25    private:
26      int __backoff__ = 1000; // current backoff, milliseconds
27      static const int __factor__ = 2.5f;
28      static const int __minbackoff__ = 1000; // minimum backoff, ms
29      static const int __max_backoff__ = 60000; // maximum backoff, ms
30      static const int __jitter__ = 500; // max random jitter, ms
31      boolean logConnect = true;
32      boolean useLts = false;
33
34      MQTTClient *mqttClient;
35      WiFiClientSecure *netClient;
36      CloudIoTCoreDevice *device;
37
```

Fonte: Próprio autor.

Já para o arquivo ‘CloudIoTCoreMqtt.h’ foi alterado as variáveis ‘Client*_netClient’, para ‘WiFiClientSecure *_netClient’**,** a função ficou conforme a figura 32.

Figura 32 - Função CloudIoTCoreMqtt



Fonte: Próprio autor.

Após essas alterações serem feitas, foi construído o software para o dispositivo, que foi composto por três códigos, sendo 2 construídos em C + + e outro na linguagem Wiring. Usou-se a IDE do Arduino para construí-los.

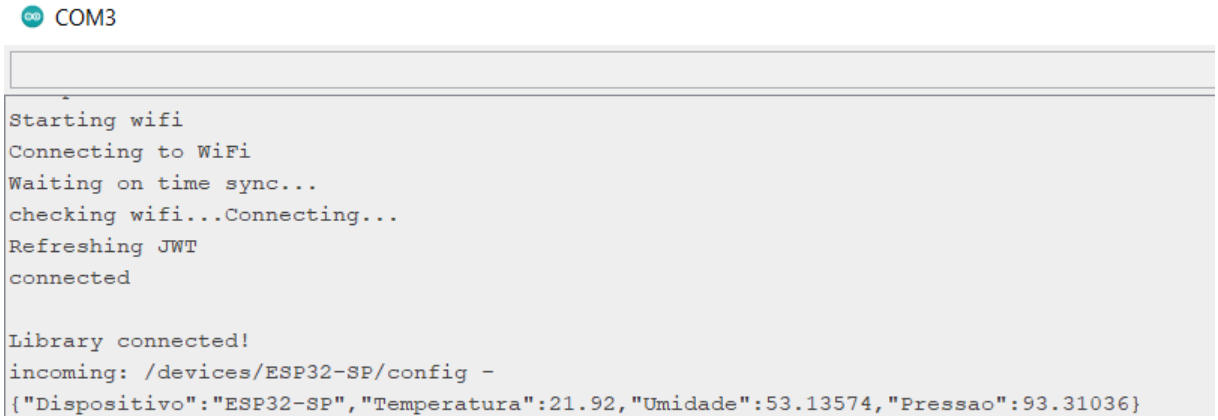
Primeiramente foram feitos os códigos em C ++ e posteriormente o código em Wiring com extensão ‘. ino’.

- Config.h, código responsável por armazenar todas as variáveis constantes do projeto, como por exemplo: o nome da rede e a senha que o dispositivo se conecta, o certificado CA, a chave privada, a localização, o registro do dispositivo, o nome do dispositivo e o id do projeto que foi obtido no tópico 3.3; e configurações do NTP.
- ESP32-MQTT.h, é a parte mais complexa do software, contém funções para conexão com a internet, funções de envio dos certificados TLS (CA e chave privada), funções para envio dos dados coletados pelo sensor via estrutura JSON.
- ESP32.ino, contém a função de leitura do sensor e invoca as funções e variáveis que estão nos arquivos ‘ESP32-MQTT.h’ e ‘Config.h’.

Dado a quantidade de linhas que contém os códigos, estão disponíveis no apêndice, que está anexo ao trabalho.

O código feito foi carregado no ESP32, a mensagem abaixo, conforme a figura 33, no monitor serial da IDE do Arduino, mostra que tudo ocorreu conforme o esperado. Isso significa que os dados foram aceitos pelo GCP e que a plataforma está gravando no banco de dados.

Figura 33 - Resultado no monitor serial.

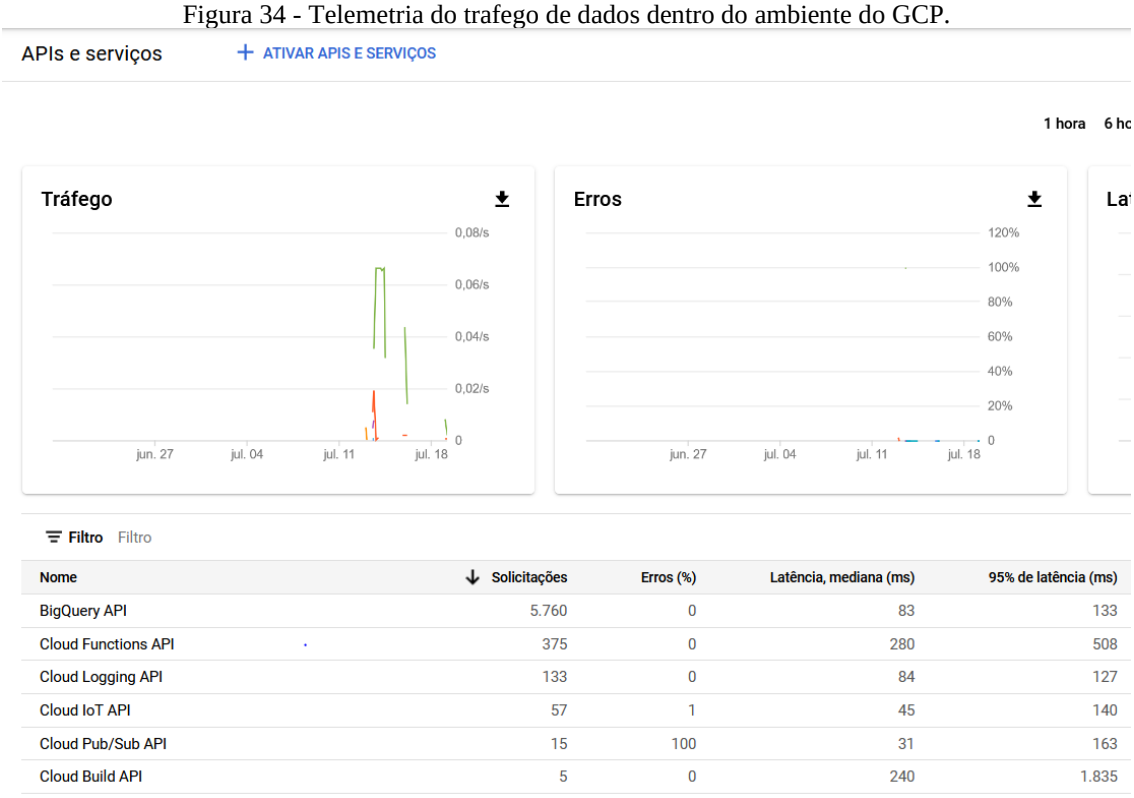


Fonte: Próprio autor.

2.5 Extração de dados do *BigQuery*

Uma vez que uma existe mensagem conforme a figura 33 isso significa que os dados estarão sendo recebidos pelo GCP, uma vez que isso acontece, o sistema de telemetria começa a funcionar e fornece diversas informações, conforme a figura 34 abaixo.

Figura 34 - Telemetria do trafego de dados dentro do ambiente do GCP.



Nome	Solicitações	Erros (%)	Latência, mediana (ms)	95% de latência (ms)
BigQuery API	5.760	0	83	133
Cloud Functions API	375	0	280	508
Cloud Logging API	133	0	84	127
Cloud IoT API	57	1	45	140
Cloud Pub/Sub API	15	100	31	163
Cloud Build API	5	0	240	1.835

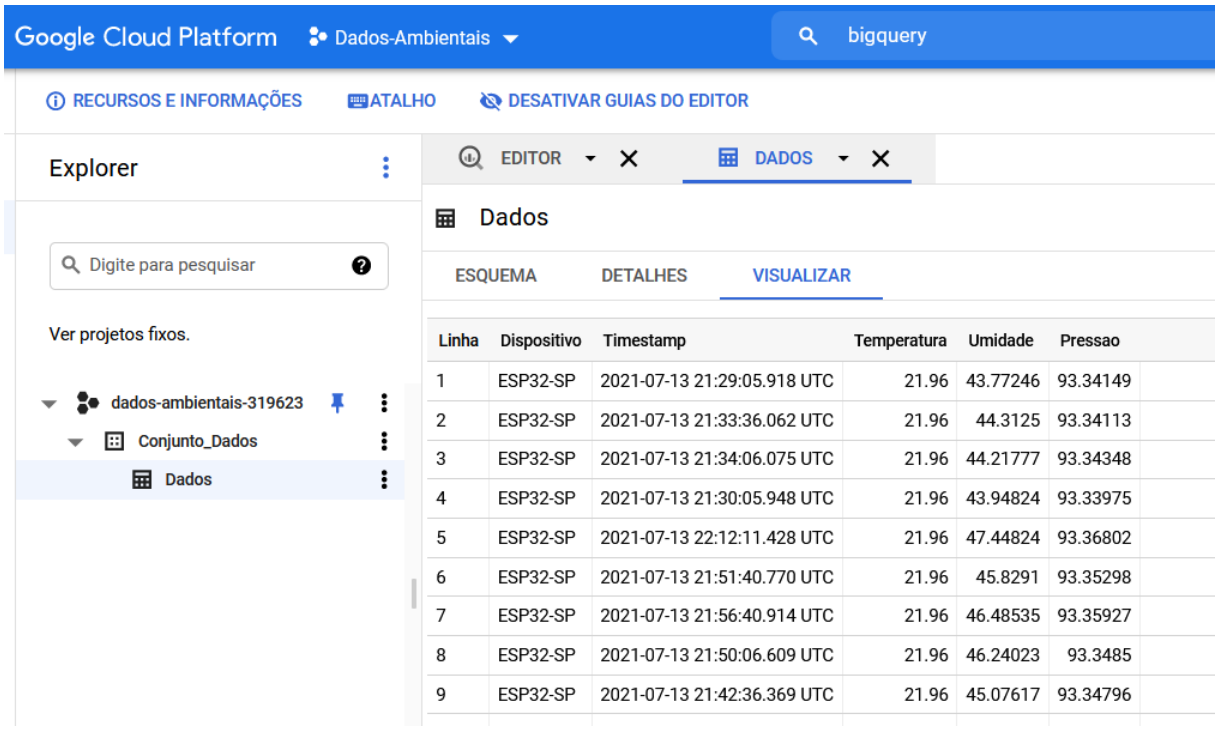
Fonte: Próprio autor.

Os dados ficam no *BigQuery*, esse banco de dados possui uma estrutura relacional, conhecida como '*Structured Query Language'*. Esse tipo de banco de dados possui uma linguagem de consulta, que é comumente chamada de linguagem SQL.

Os comandos em SQL são usados para extrair os dados de acordo com as necessidades de visualização do relatório no *Data Studio*.

Abaixo, na figura 35, é mostrado os dados que estão sendo armazenados no banco de dados.

Figura 35 - Dados armazenados no *BigQuery*.

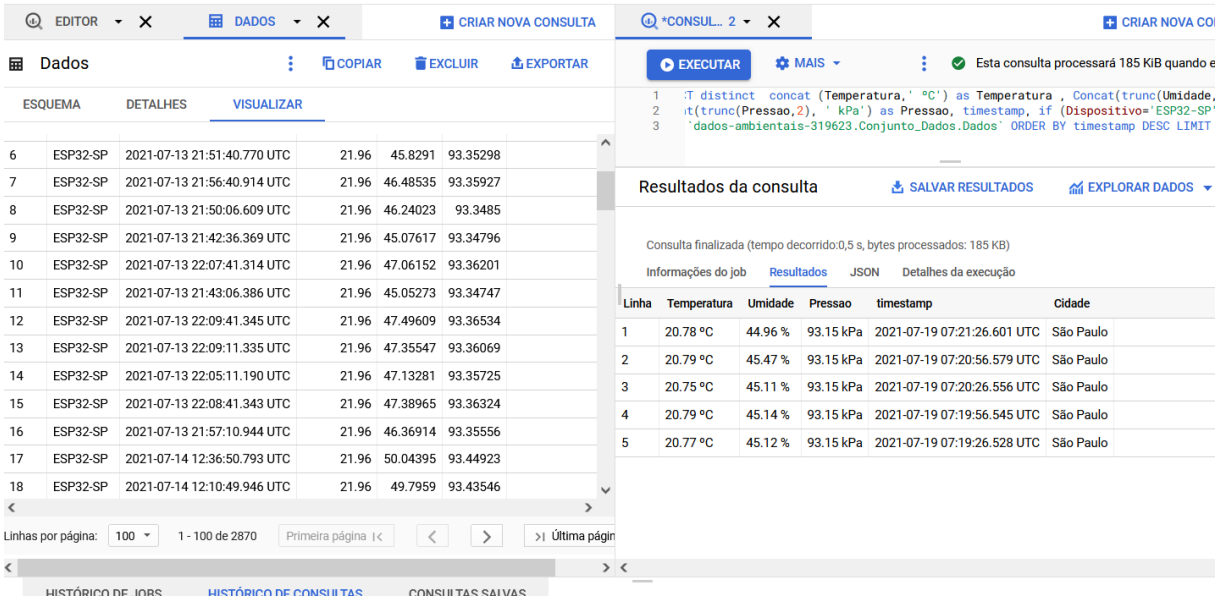


Linha	Dispositivo	Timestamp	Temperatura	Umidade	Pressao
1	ESP32-SP	2021-07-13 21:29:05.918 UTC	21.96	43.77246	93.34149
2	ESP32-SP	2021-07-13 21:33:36.062 UTC	21.96	44.3125	93.34113
3	ESP32-SP	2021-07-13 21:34:06.075 UTC	21.96	44.21777	93.34348
4	ESP32-SP	2021-07-13 21:30:05.948 UTC	21.96	43.94824	93.33975
5	ESP32-SP	2021-07-13 22:12:11.428 UTC	21.96	47.44824	93.36802
6	ESP32-SP	2021-07-13 21:51:40.770 UTC	21.96	45.8291	93.35298
7	ESP32-SP	2021-07-13 21:56:40.914 UTC	21.96	46.48535	93.35927
8	ESP32-SP	2021-07-13 21:50:06.609 UTC	21.96	46.24023	93.3485
9	ESP32-SP	2021-07-13 21:42:36.369 UTC	21.96	45.07617	93.34796

Fonte: Próprio autor.

Entretanto, se se for utilizado a linguagem SQL, os dados podem ser transformados, abaixo, na figura 36, está o exemplificando. Nesse caso, o nome do dispositivo ESP32-SP foi alterado para o nome de uma cidade, São Paulo e, para as variáveis, foi adicionado unidades de medidas, ºC para temperatura, % a umidade relativa do ar e, kPa, para pressão atmosférica.

Figura 36 - Utilização da linguagem SQL para transformar variáveis.



Fonte: Próprio autor.

O código SQL utilizado para consulta, está abaixo:

```
SELECT concat (Temperatura,' °C') as Temperatura , Concat(trunc(Umidade,2),
" %") as Umidade ,
Concat(trunc(Pressao,2), ' kPa') as Pressao, timestamp, if
(Dispositivo='ESP32-SP', 'São Paulo', '')as Cidade
FROM `dados-ambientais-319623.Conjunto_Dados.Dados` ORDER BY timestamp DESC
LIMIT 5
```

Portanto, para visualizar os dados da maneira desejada, é necessário utilizar esse tipo de linguagem. Ela será utilizada juntamente com o Data Studio, criando um relatório para visualização dos dados.

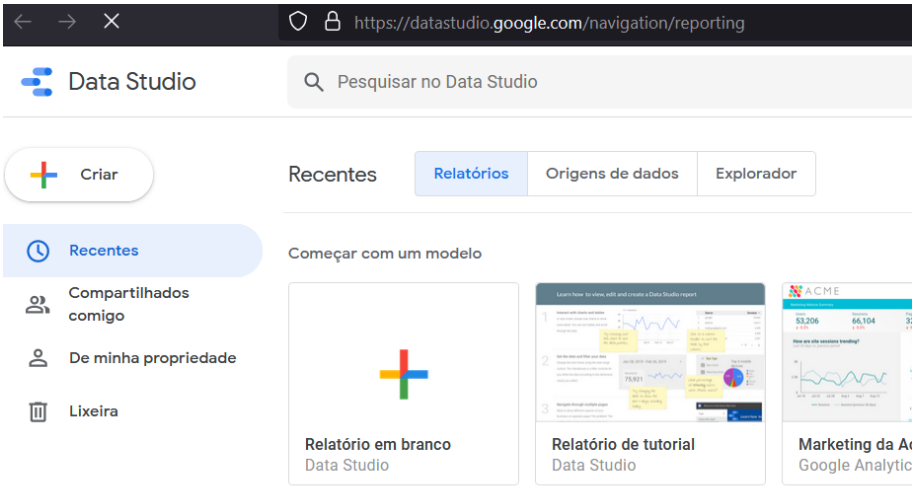
2.6 Utilizando o *Data Studio*

O *Data Studio* é uma ferramenta de BI (*Business Inteligence*), oferecida gratuitamente pela Google, esse produto faz parte do GSuite da empresa, ou seja, possui integração total com todos os outros serviços que a empresa oferece.

Utilizando essa ferramenta, usou-se consultas SQL para extrair os dados do *BigQuery* e exibi-los no relatório.

No site do Data Studio, foi criado um relatório em branco, conforme a figura 37.

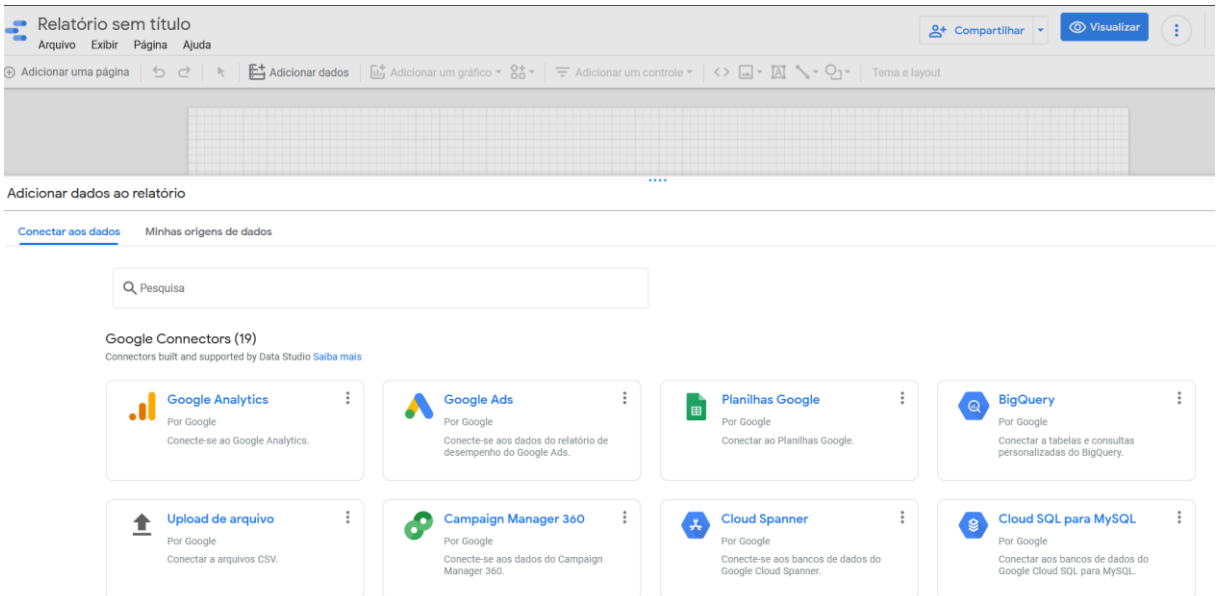
Figura 37 - Tela inicial do Data Studio.



Fonte: Próprio autor.

Após criar o relatório em branco, conectou-se ao *BigQuey*, selecionando o conector que possui o mesmo nome, conforme a figura 38.

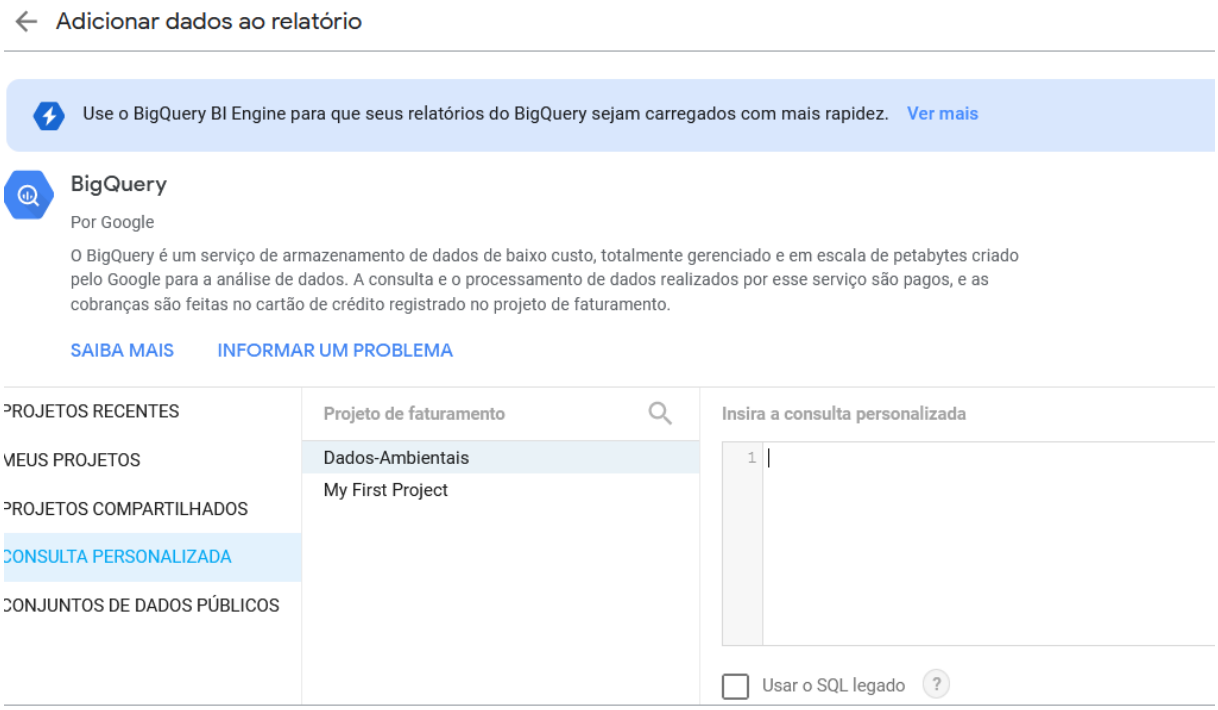
Figura 38 - Pagina de conector de dados do Bigquery.



Fonte: Próprio autor.

Após a etapa anterior, foi selecionada a consulta personalizada e o projeto ‘Dados-Ambientais’, nesse momento, foram inseridos os códigos SQL, conforme a figura 39, está ilustrando. Criou-se consultas personalizadas diferentes, para montar o relatório com os dados nas disposições necessárias.

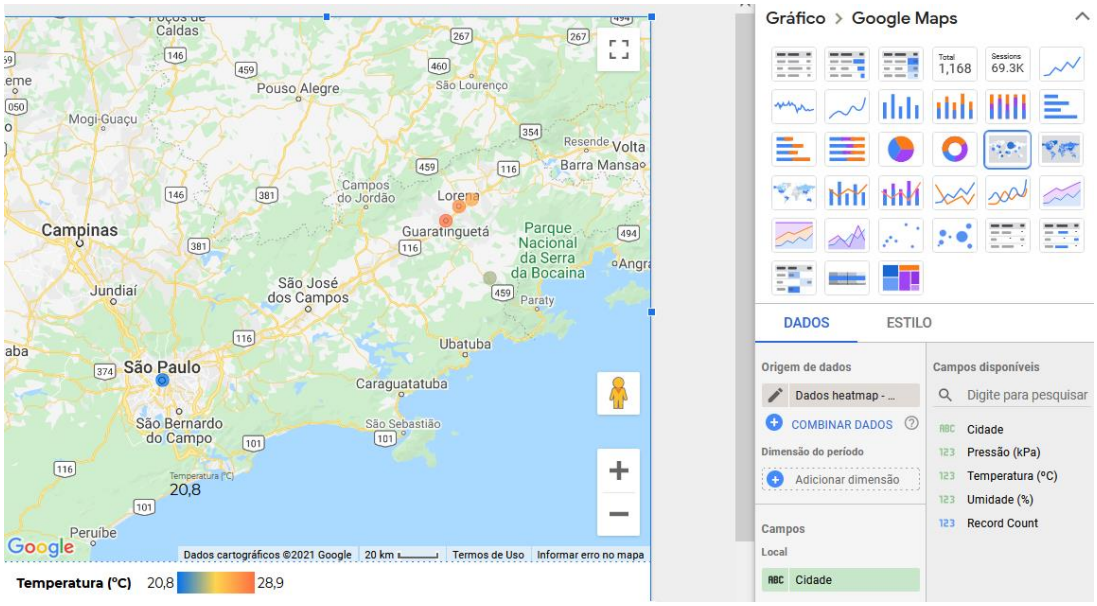
Figura 39 - Consulta personalizada *BigQuery*.



Fonte: Próprio autor.

Uma vez criando uma consulta, conforme a imagem acima, terá um painel com diversas opções de gráficos para serem utilizados, abaixo, na figura 40, está sendo mostrado, no canto superior direito.

Figura 40 - Opções de gráficos no *BigQuery*.



Fonte: Próprio autor.

Para o relatório ficar mais informativo e dinâmico, foram gerados outros dados no *BigQuery*, utilizando o mesmo dispositivo, com outros nomes. Abaixo, na tabela 2, da relação do nome do dispositivo com a cidade.

Tabela 2 - Relação do nome dos sensores, com as cidades.

Sensor	Cidade
ESP32-SP	São Paulo - SP
ESP32-LR	Lorena - SP
ESP32-GU	Guaratingueta - SP
ESP32-CU	Cunha - SP

Fonte: Próprio autor.

Foram feitas três consultas em SQL, para criar o relatório.

- Consulta 1:

```
SELECT distinct Temperatura , Umidade , Pressao, timestamp,
if (Dispositivo='ESP32-SP', 'São Paulo',
 if (Dispositivo='ESP32-LR', 'Lorena',
   if (Dispositivo='ESP32-CU', 'Cunha',
       if (Dispositivo='ESP32-CA', 'Canas',
          if (Dispositivo='ESP32-GU', 'Guaratingueta','')as Cidade
FROM `dados-ambientais-319623.Conjunto_Dados.Dados` ORDER BY timestamp DESC
LIMIT 5
```

- Consulta 2:

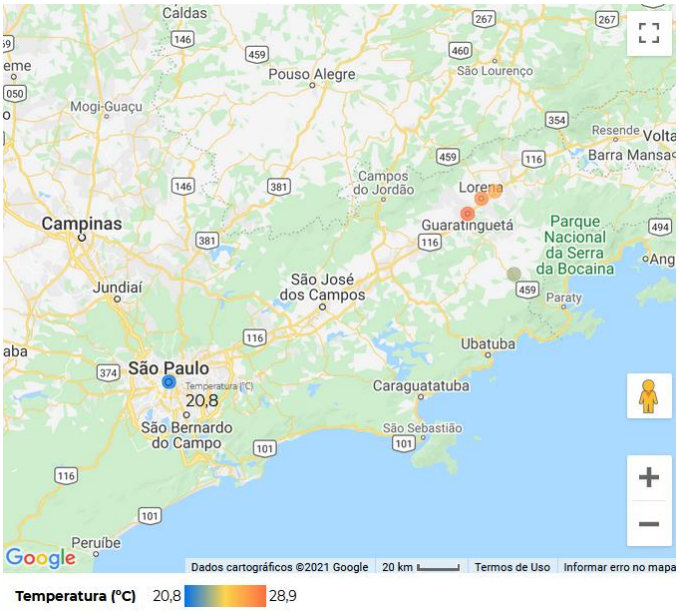
```
SELECT distinct Temperatura , Umidade , Pressao, timestamp,
if (Dispositivo='ESP32-SP', 'São Paulo',
 if (Dispositivo='ESP32-LR', 'Lorena',
   if (Dispositivo='ESP32-CU', 'Cunha',
       if (Dispositivo='ESP32-CA', 'Canas',
          if (Dispositivo='ESP32-GU', 'Guaratingueta','')as Cidade
FROM `dados-ambientais-319623.Conjunto_Dados.Dados` ORDER BY timestamp DESC
LIMIT 5
```

- Consulta 3:

```
SELECT distinct Temperatura , Umidade , Pressao,
timestamp (DATE 'dd-MM-yyyy') as Date,
if (Dispositivo='ESP32-SP', 'São Paulo','')as Cidade
FROM `dados-ambientais-319623.Conjunto_Dados.Dados` ORDER BY Date DESC LIMIT 5
```

Utilizando a Consulta 1, juntamente com o mapa de balões, foi feito um gráfico do tipo mapa, a Figura 41 que contém a localização dos dispositivos, assim como a escala de cor de temperatura, do lugar mais quente para o mais frio.

Figura 41 - Mapa de balões, com os dados coletados pelo dispositivo.



Fonte: Próprio autor.

Utilizando a Consulta 2, juntamente com o gráfico de tabela, foi feita a tabela que mostra a temperatura, umidade relativa do ar e pressão atmosférica, conforme a figura 42, nesse tipo de gráfico é possível ver rapidamente o valor de todas variáveis obtidas.

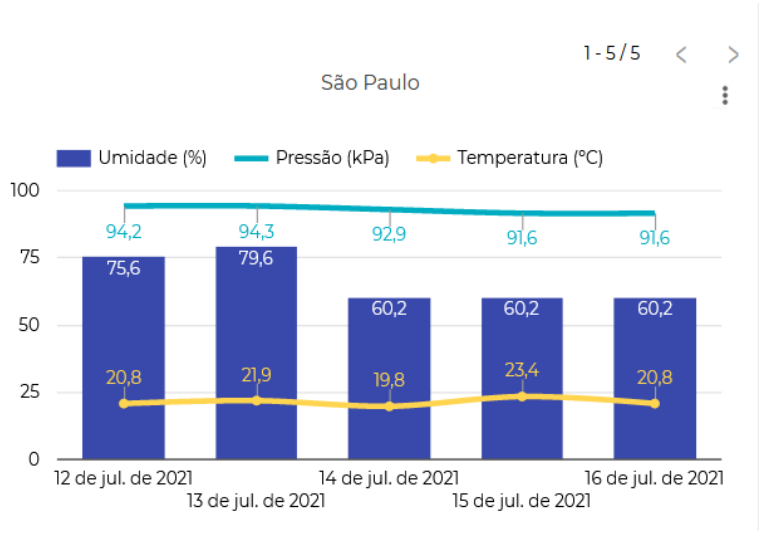
tabela 3 - Gráfico de tabelas com os dados do dispositivo.

	Cidade	Temperatura (°C)	Umidade (%)	Pressão (kPa)
1.	Guaratingueta - SP	28,9	79,55	94,26
2.	Lorena - SP	27,8	75,55	94,23
3.	São Paulo - SP	20,8	60,23	91,56
4.	Cunha - SP	23,43	60,23	91,56
5.	Canas - SP	26,77	60,23	92,92

Fonte: Próprio autor.

Já para a Consulta 3, foi utilizado um gráfico de série temporal, para analisar o comportamento da temperatura, umidade e pressão de uma cidade ao longo de 5 dias, na figura 42, está exemplificado.

Figura 42 - Gráfico de serie temporal dos dados coletados.



Fonte: Próprio autor.

Observe que para cada consulta feita, ao final, existe o comando:

```
ORDER BY timestamp DESC LIMIT 5
```

Isso significa que os dados são ordenados pela data e horário que eles foram inseridos no banco de dados, ou seja, mostra os 5 últimos dados inseridos.

3 RESULTADOS E DISCUSSÃO

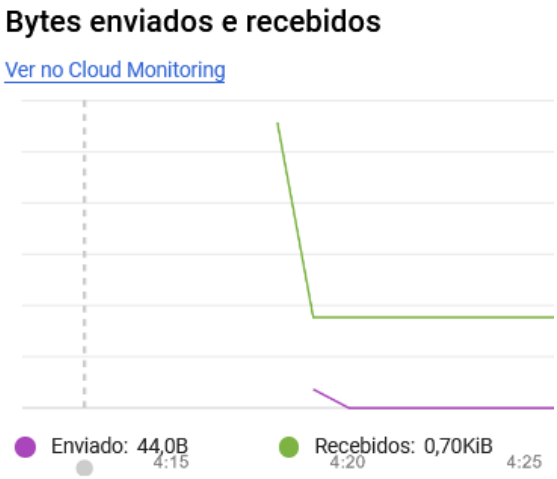
Nesse tópico, serão analisados os custos do GCP e do dispositivo e, também, uma breve discussão sobre o relatório gerado pelo *Data Studio*.

3.1 Custo do *Google Cloud Platform*

As empresas que fornecem o serviço de computação em nuvem, na maioria das vezes, cobram pelo uso do poder computacional, da rede de dados e armazenamento. No caso do GCP, para os produtos utilizados neste projeto, os custos serão todos variáveis, ou seja, eles dependem diretamente da escalabilidade do projeto. Como se trata de uma prototipação, de um único dispositivo conectado à plataforma, os custos relacionados ao projeto são ínfimos.

Através da telemetria fornecida pela plataforma, figura 43, nota-se que os dados enviados ao GCP, tem um peso unitário de dados (por envio) de aproximadamente 0.7 kB.

Figura 43 - Peso de uma mensagem por envia.



Fonte: Próprio autor.

O IoT Core tem custos relativamente baixos, a tabela 4, mostra o valor por megabytes. Com isso, é possível estimar os custos para um dispositivo.

Tabela 4 - Preços para o IoT Core.

Nível	Preço	Volume de dados (por mês)
Gratuito	US$ 0,00	Primeiros 250 MB
Padrão	US$ 0,0045 por MB	250 MB a 250 GB
	US$ 0,0020 por MB	250 GB a 5 TB
	US$ 0,00045 por MB	5 TB ou mais

Fonte: Google (2021).

A tabela 5, abaixo, está mostrando o custo, em dólares, de cada envio o GCP:

Tabela 5 - Relação tamanho, requisição e custo.

Tamanho (kB)	Requisições	Custos US$
0,7	1	0,000003

Fonte: Próprio autor.

O *Cloud Functions* tem um preço que depende da máquina criada para executar as funções, a máquina utilizada do projeto foi de 128 MB de memória ram e CPU de 200 MHz.

A tabela 6, abaixo, mostra o valor mensal para essa máquina, que totaliza US$7,2 dólares por mês.

Tabela 6- Preços praticados para o *Cloud Function*.

Métrica	Valor bruto	Nível gratuito	Valor líquido	Preço unitário	Preço total
Invocações	10.000.000	2.000.000	8.000.000	US$ 0,0000004	US$ 3,20
GB-segundos	375.000	400.000	< 0	US$ 0,0000025	US$ 0,00
GHz-segundos	600.000	200.000	400.000	US$ 0,0000100	US$ 4,00
Rede	0	5	0	US$ 0,12	US$ 0,00
Total/mês					US$ 7,20

Fonte: Google (2021).

Para *BigQuery*, os custos são diferenciados, envolve custos de armazenamento, custos de consulta de dados, entre outros (GOOGLE, 2021). Para esse serviço utilizado o custo foi zero, pois todos os meses o cliente ganha 10 GB iniciais. Nota-se que, 10 GB é muito espaço de armazenamento, como o arquivo enviado pesa cerca de 0.77 kB, pode-se enviar aproximadamente 14.285.714 de mensagens para os serviços do *BigQuery*.

Abaixo, na tabela 7, mostra os valores e os detalhes.

Tabela 7 - Preços praticados para o *BigQuery*.

Operação	Preços	Detalhes
Armazenamento ativo	$0.020 per GB	Os primeiros 10 GB são gratuitos todos os meses.
Armazenamento de longo prazo	$0.010 per GB	Os primeiros 10 GB são gratuitos todos os meses.

Fonte: Google (2021).

3.2 Custos com dispositivo IoT

Os custos para construir o dispositivo IoT, são mostrados na tabela 8.

Tabela 8 – Preços dos produtos comprados para criar o dispositivo.

Produto	Preço (R$)
ESP32	56,8
BME280	34,8
2 uni. mini Protoboard	21,7
Jumpers	16,8
Total	130,1

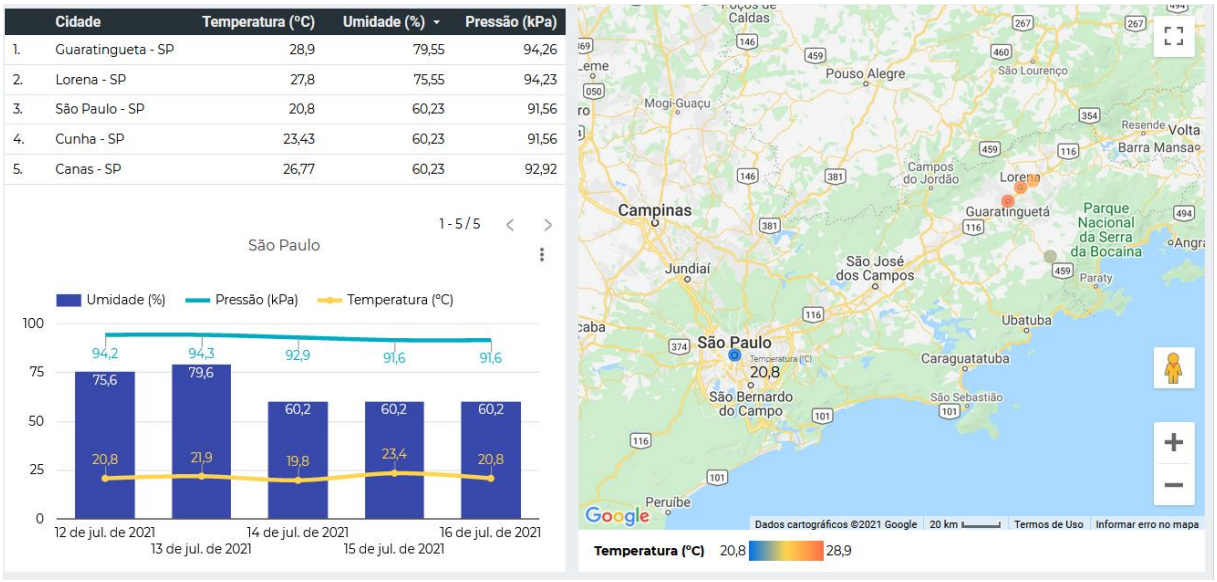
Fonte: Próprio autor.

Vale ressaltar que, atualmente, o dólar se encontra em um patamar alto e os itens são todos importados.

3.3 Relatório com o Data Studio

Como resultado final do tópico 3.6, na figura 44, está o relatório. Vale ressaltar que é possível publicar em sites ou enviar para acesso através de links. Esse relatório é atualizado em tempo real, ou seja, à medida que os dados vão sendo inseridos na base do *BigQuery*, esse relatório irá se atualizando. Além disso, a utilização dessa ferramenta é gratuita e não possuem custos associados.

Figura 44 - Relatório final com o *Data Studio*.



Fonte: Próprio autor.

4 CONCLUSÃO

A construção do projeto IoT para coleta de dados ambientais, utilizando a plataforma do GCP, teve um resultado interessante que pode ser visto no relatório com o *Data Studio*. A obtenção de dados ambientais em tempo real tem várias aplicações de caráter científico e econômico. Sistemas como esse, são amplamente utilizados em *SmartFarms*, Sistemas AgroTecs. Vale ressaltar que a plataforma do GCP é muito boa para o desenvolvimento de sistemas IoT, já que o ambiente oferece diversos tipos de recursos e suporte.

Com esse trabalho, podemos notar que utilizar o GCP para projetos IoT é bastante vantajoso, já que a plataforma oferece escalabilidade para o projeto, caso tome outras proporções. Também, existe o fato de que há uma gama de produtos dentro da plataforma, como por exemplo, inteligência artificial, isso pode tornar os projetos mais robustos e precisos.

Durante o desenvolvimento do projeto, tiveram alguns desafios, como por exemplo, a questão de segurança TLS entre o GCP e o microcontrolador, a obtenção dos certificados. Entretanto, para os ajustes feitos nas bibliotecas, por menor que eles foram, foi necessário te um entendimento total do funcionamento dos códigos e do fluxo de transmissão dos dados, para que pudesse encontrar uma solução viável.

Esse projeto não seria possível sem o conhecimento adquirido na graduação, juntamente com os programas cultura e extensão, que foram imprescindíveis para o desenvolvimento desse projeto, já que nesses programas de cultura e extensão acabei conhecendo o mundo IoT, passando por vários projetos tanto com o Arduino e ESP8266.

REFERÊNCIAS

AFAELS, Ray J. **Cloud Computing: From Beginning**. South Carolina: Createspace, 2015. 154 p.

AZURE, Microsoft. **O que é PaaS?** 2021. Disponível em: https://azure.microsoft.com/pt-br/overview/what-is-paas/. Acesso em: 20 jul. 2021.

AKITA, Fábio. **Entendendo "Devops" para Iniciantes em Programação**. Disponível em: https://www.akitaonrails.com/2019/04/17/akitando-48-entendendo-devops-para-iniciantes-em-programacao-parte-2-serie-comecando-aos-40. Acesso em: 20 jul. 2021.

ARMBRUST, M. et al. (2009). **Above the Clouds: A Berkeley View of Cloud Computing**. Technical Report No. UCB/ EECS-2009-28, University of California, Berkley.

AVSYSTEM. **Benefits of a cloud platform in the IoT**. 2020. Disponível em: https://www.avsystem.com/blog/iot-cloud-platform/. Acesso em: 21 jul. 2021.

BERNSTEIN, Corinne; BRUSH, Kate; GILLIS, Alexander S. **MQTT (MQ Telemetry Transport)**. 2020. Disponível em: https://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport. Acesso em: 21 jul. 2021.

CLOUDFLARE. **Transport Layer Security TLS**. 2021. Disponível em: https://www.cloudflare.com/pt-br/learning/ssl/transport-layer-security-tls/. Acesso em: 21 jul. 2021.

FVML. **Conhecendo ESP8266, Especificações e Comparações com ESP32 e Arduino**. 2021.Disponívelem:http://www.fvml.com.br/2018/12/conhecendo-esp8266-especificacoes-e.html. Acesso em: 21 jul. 2021.

GOOGLE. **Google Docs**. 2021. Disponível em: https://cloud.google.com/docs. Acesso em: 21 jul. 2021.

INTERNET SOCIETY. **TLS Basics**. 2021. Disponível em: https://www.internetsociety.org/deploy360/tls/basics/. Acesso em: 21 jul. 2021.

KURNIAWAN, Agus. **Internet of Things Projects with ESP32: Build exciting and powerful IoT projects**. London: Packt Publishing, 2019. 252 p.

RANDOMNERDTUTORIALS. **ESP32 Pinout Reference: Which GPIO pins should you use?** 2021. Disponível em: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/. Acesso em: 21 jul. 2021.

ROMANO, Matheus. **Entenda o que é IoT na indústria 4.0 e porque isso é uma aposta que vai revolucionar o mercado industrial**. 2018. Disponível em: https://www.logiquesistemas.com.br/blog/iot-na-industria-4-0/. Acesso em: 21 jul. 2021.

ROBOT ELECTRONICS. **Using the I2C Bus**. 2021. Disponível em: https://www.robot-electronics.co.uk/i2c-tutorial. Acesso em: 20 jul. 2021

MICROSOFT AZURE. **O que é computação em nuvem?** 2021. Disponível em: https://azure.microsoft.com/pt-br/overview/what-is-cloud-computing/#cloud-computing-models. Acesso em: 21 jul. 2021.

MAGRANI, Eduardo. **A internet das coisas**. Rio de Janeiro: Fgv Editora, 2018. 190 p.

MAGDY, Khaled. **I2C Communication Protocol Tutorial**. 2020. Disponível em: https://deepbluembedded.com/i2c-communication-protocol-tutorial-pic/. Acesso em: 21 jul. 2021.

MENDONÇA, Hélio Sousa. **SPI e I2C**. 2021. Disponível em: https://paginas.fe.up.pt/~hsm/docencia/comp/spi-e-i2c/. Acesso em: 21 jul. 2021.

MQTT (org.). **MQTT Publish / Subscribe Architecture**. 2021. Disponível em: https://mqtt.org. Acesso em: 21 jul. 2021.

MULTILOGICA. **Arduinos originais**. 2021. Disponível em: https://multilogica-shop.com/arduinos-originais-e-compatíveis. Acesso em: 21 jul. 2021.

POSTSCAPES. **IoT Overview Handbook**. 2019. Disponível em: https://www.postscapes.com/iot. Acesso em: 21 jul. 2021.

P. Mell and T. Grance, **The NIST Definition of Cloud**, US National Institute of Standards and Technology, Gaithersburg, MD, 2011.

REDHAT (org.).**O que é IoT?** 2019. Disponível em: https://www.redhat.com/pt-br/topics/internet-of-things/what-is-iot. Acesso em: 20 jul. 2021.

SANTOS, Sandro. **Introdução à IoT - Desvendando a Internet das Coisas**. São Paulo: Createspace Independent Publishing Platform, 2018. 170 p.

TECNICON. **4 exemplos práticos da adoção da Indústria 4.0 nas fábricas**. 2020. Disponível em:https://www.tecnicon.com.br/blog/4764_exemplos_praticos_da_adocao_da_Industria_4_0 nas_fabricas. Acesso em: 21 jul. 2021.

APÊNDICE A

Códigos do microcontrolador ESP32:

ESP32-MQTT.h

```
#include <Client.h>
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <MQTT.h>
#include <CloudIoTCore.h>
#include <CloudIoTCoreMqtt.h>
#include "config.h"

void messageReceived(String &topic, String &payload){
  Serial.println("incoming: " + topic + " - " + payload);
}

WiFiClientSecure *netClient;
CloudIoTCoreDevice *device;
CloudIoTCoreMqtt *mqtt;
MQTTClient *mqttClient;
unsigned long iat = 0;
String jwt;


String getDefaultSensor(){
  return "Wifi: " + String(WiFi.RSSI()) + "db";
}

String getJwt(){
  iat = time(nullptr);
  Serial.println("Refreshing JWT");
  jwt = device->createJWT(iat, jwt_exp_secs);
  return jwt;
}

void setupWifi(){
  Serial.println("Starting wifi");

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.println("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED){
    delay(100);
  }

  configTime(0, 0, ntp_primary, ntp_secondary);
  Serial.println("Waiting on time sync...");
  while (time(nullptr) < 1510644967){
    delay(10);
  }
}

void connectWifi(){
  Serial.print("checking wifi...");
  while (WiFi.status() != WL_CONNECTED){
    Serial.print(".");
```

```
    delay(1000);
  }
}

bool publishTelemetry(String data){
  return mqtt->publishTelemetry(data);
}

bool publishTelemetry(const char *data, int length){
  return mqtt->publishTelemetry(data, length);
}

bool publishTelemetry(String subfolder, String data){
  return mqtt->publishTelemetry(subfolder, data);
}

bool publishTelemetry(String subfolder, const char *data, int length){
  return mqtt->publishTelemetry(subfolder, data, length);
}

void connect(){
  connectWifi();
  mqtt->mqttConnect();
}

void setupCloudIoT(){
 device = new CloudIoTCoreDevice(
      project_id, location, registry_id, device_id,
      private_key_str);

  setupWifi();

  netClient = new WiFiClientSecure();
  netClient->setCACert(root_cert);
  mqttClient = new MQTTClient(512);
  mqttClient->setOptions(180, true, 1000);
  mqtt = new CloudIoTCoreMqtt(mqttClient, netClient, device);
  mqtt->setUseLts(true);
  mqtt->startMQTT();
}
```

Config.h

```
// Credenciais da rede.
const char *ssid = "Nome da rede WiFi";
const char *password = "Senha da rede WiFi";
// Configuração NTP.
const char* ntp_primary = "pool.ntp.org";
const char* ntp_secondary = "time.nist.gov";
// Configuração JWT.
const int jwt_exp_secs = 3600; // Maximo 24H (3600*24)
// Variaveis do projeto que serão recebidas pelo GCP.
const char *project_id = "esp32-iot-315202";
const char *location = "us-central1";
const char *registry_id = "ESP32_Registry";
const char *device_id = "ESP32";
const char *private_key_str =
    "84:00:37:29:cd:bb:18:44:c4:d3:12:65:c1:17:b6:"
```

```
    "78:a3:37:86:97:85:b1:d7:ff:3f:c3:77:f9:5b:b4:"
    "ae:0e";

const char *root_cert =
"-----BEGIN CERTIFICATE-----\n"
"MIIDDDCCArKgAwIBAgIURjzskJE39lD3SX0IkTs5jt4noWYwCgYIKoZIzj0EAwIw\n"
"RDELMAkGA1UEBhMCVVMxIjAgBgNVBAoTGUdvb2dsZSBUcnVzdCBTZXJ2aWNlcyBM\n"
"TEMxETAPBgNVBAMTCEdUUyBMVFNYMB4XDTIwMDcxMzAwMDAwMFoXDTIxMDcxMjAw\n"
"MDAwMFowbTELMAkGA1UEBhMCVVMxEzARBgNVBAgMCkNhbGlmb3JuaWExFjAUBgNV\n"
"BAcMDU1vdW50YWluIFZpZXcxEzARBgNVBAoMCkdvb2dsZSBMTEMxHDAaBgNVBAMM\n"
"EyouMjAzMC5sdHNhcGlzLmdvb2cwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAARh\n"
"fb8NcIBEcwbwSJKKlnjJ7cDwKMHiHeQd1xgE4PWzqEFxauPG30WniyoFZQDtgqft\n"
"Tq6iImzs1iP4cR5xN9Hao4IBVzCCAVMwEwYDVR0lBAwwCgYIKwYBBQUHAwEwDgYD\n"
"VR0PAQH/BAQDAgeAMB4GA1UdEQQXMBWCEyouMjAzMC5sdHNhcGlzLmdvb2cwDAYD\n"
"VR0TAQH/BAIwADAfBgNVHSMEGDAWgBSzK6ugSBx+E4rJCMRAQiKiNlHiCjBpBggr\n"
"BgEFBQcBAQRdMFswLwYIKwYBBQUHMAKGI2h0dHA6Ly9wa2kuZ29vZy9ndHNsdHNy\n"
"L2d0c2x0c3guY3J0MCgGCCsGAQUFBzABhhxodHRwOi8vb2NzcC5wa2kuZ29vZy9H\n"
"VFNMVFNYMCEGA1UdIAQaMBgwDAYKKwYBBAHWeQIFAzAIBgZngQwBAgIwMAYDVR0f\n"
"BCkwJzAloCOgIYYfaHR0cDovL2NybC5wa2kuZ29vZy9HVFNMVFNYLmNybDAdBgNV\n"
"HQ4EFgQU3VXinuGqY4eo6dBUviU9iLPkWTcwCgYIKoZIzj0EAwIDSAAwRQIhAO4t\n"
"PT9rr/pZxYx3KSzXRK0l0bxpVELE2WS43FIYvWYzAiBwEHAunJZ5oBXfPXH6sJkr\n"
"Xt+ea9ZwVMYLXWQnGvKQ+Q==\n"
"-----END CERTIFICATE-----\n"
"-----BEGIN CERTIFICATE-----\n"
"MIIC0TCCAnagAwIBAgINAfQKmcm3qFVwT0+3nTAKBggqhkjOPQQDAjBEMQswCQYD\n"
"VQQGEwJVUzEiMCAGA1UEChMZR29vZ2xlIFRydXN0IFNlcnZpY2VzIExMQzERMA8G\n"
"A1UEAxMIR1RTIExUU1IwHhcNMTkwMTIzMDAwMDQyWhcNMjkwNDAxMDAwMDQyWjBE\n"
"MQswCQYDVQQGEwJVUzEiMCAGA1UEChMZR29vZ2xlIFRydXN0IFNlcnZpY2VzIExM\n"
"QzERMA8GA1UEAxMIR1RTIExUU1gwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAARr\n"
"6/PTsGoOg9fXhJkj3CAk6C6DxHPnZ1I+ER40vEe290xgTp0gVplokojbN3pFx07f\n"
"zYGYAX5EK7gDQYuhpQGIo4IBSzCCAUcwDgYDVR0PAQH/BAQDAgGGMB0GA1UdJQQW\n"
"MBQGCCsGAQUFBwMBBggrBgEFBQcDAjASBgNVHRMBAf8ECDAGAQH/AgEAMB0GA1Ud\n"
"DgQWBBSzK6ugSBx+E4rJCMRAQiKiNlHiCjAfBgNVHSMEGDAWgBQ+/v/MUuu/ND49\n"
"80DQ5CWxX7i7UjBpBggrBgEFBQcBAQRdMFswKAYIKwYBBQUHMAGGHGh0dHA6Ly9v\n"
"Y3NwLnBraS5nb29nL2d0c2x0c3IwLwYIKwYBBQUHMAKGI2h0dHA6Ly9wa2kuZ29v\n"
"Zy9ndHNsdHNyL2d0c2x0c3IuY3J0MDgGA1UdHwQxMC8wLaAroCmGJ2h0dHA6Ly9j\n"
"cmwucGtpLmdvb2cvZ3RzbHRzci9ndHNsdHNyLmNybDAdBgNVHSAEFjAUMAgGBmeB\n"
"DAECATAIBgZngQwBAgIwCgYIKoZIzj0EAwIDSQAwRgIhAPWeg2v4yeimG+lzmZAC\n"
"DJOlalpsiwJR0VOeapY8/7aQAiEAiwRsSQXUmfVUW+N643GgvuMH70o2Agz8w67f\n"
"SX+k+Lc=\n"
"-----END CERTIFICATE-----\n";


// Topicos extras

const int ex_num_topics = 0;
const char* ex_topics[ex_num_topics];


ESP32.ino

#include <Arduino.h>
#include <WiFiClientSecure.h>
#include "esp32-mqtt.h"
#include <Adafruit_Sensor.h>
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_BME280.h>
#include <ArduinoJson.h>
```

```
Adafruit_BME280 bme;
char buffer[100];
void setup() {
  Serial.begin(115200);
  Serial.println("Setup.....");
    unsigned status;
  status=bme.begin(0x76);
  setupCloudIoT();

}

unsigned long lastMillis = 0;

void loop() {
  mqtt->loop();
  delay(10);  // <- corrige alguns problemas com a estabilidade do WiFi

  if (!mqttClient->connected()) {
    connect();
  }

  if (millis() - lastMillis > 3600) {

    lastMillis = millis();
    float Temp = bme.readTemperature();
    float Umi = bme.readHumidity();
    float Press = bme.readPressure()/1000.0F;
    String NomedoDispositivo = "ESP32-SP";
    StaticJsonDocument<300> doc;
    doc["Dispositivo"] = NomedoDispositivo;
    doc["Temperatura"] = Temp;
    doc["Umidade"] = Umi;
    doc["Pressao"] = Press;

    serializeJson(doc, buffer);
    //publishTelemetry(mqttClient, "/sensors", getDefaultSensor());
    publishTelemetry( buffer);
    Serial.println(buffer);
    delay( 30000);
  }
}
```