UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E ELETRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Tiago Mayer Dörner

Atualização de firmware em campo de forma segura e criptografada

Florianópolis
2022

Tiago Mayer Dörner

Atualização de firmware em campo de forma segura e criptografada

Trabalho de Conclusão do Curso de Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Bacharel em Engenharia Elétrica.
Orientador: Willian Henrique, Me.
Coorientador: Prof. Richard Demo Souza, Dr.

Florianópolis

2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Dörner, Tiago Mayer
Atualização de firmware em campo de forma segura e criptografada / Tiago Mayer Dörner ; orientador, Willian Henrique, coorientador, Richard Demo Souza, 2022.
51 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Elétrica, Florianópolis, 2022.

Inclui referências.

1. Engenharia Elétrica. 2. Firmware. I. Henrique, Willian. II. Souza, Richard Demo. III. Universidade Federal de Santa Catarina. Graduação em Engenharia Elétrica. IV. Título.

Tiago Mayer Dörner

Atualização de firmware em campo de forma segura e criptografada

Este Trabalho Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Engenharia Elétrica" e aceito, em sua forma final, pelo Curso de Graduação em Engenharia Elétrica.

Florianópolis, 21 de março de 2022.

Documento assinado digitalmente
Jean Vianei Leite
Data: 24/03/2022 09:39:11-0300
CPF: 003.474.909-80
Verifique as assinaturas em https://v.ufsc.br

Prof. Jean Vianei Leite, Dr.
Coordenador do Curso de Graduação em Engenharia Elétrica

Banca Examinadora:

Documento assinado digitalmente
WILLIAN HENRIQUE
Data: 23/03/2022 15:48:15-0300
CPF: 074.931.159-23
Verifique as assinaturas em https://v.ufsc.br

Willian Henrique, Me.
Orientador
SENAI

Documento assinado digitalmente
Richard Demo Souza
Data: 22/03/2022 21:03:39-0300
CPF: 004.267.379-89
Verifique as assinaturas em https://v.ufsc.br

Prof. Richard Demo Souza, Dr.
Coorientador
Universidade Federal de Santa Catarina

Documento assinado digitalmente
EDUARDO LUIZ ORTIZ BATISTA
Data: 22/03/2022 22:54:41-0300
CPF: 036.521.889-85
Verifique as assinaturas em https://v.ufsc.br

Prof. Eduardo Batista, Dr.
Universidade Federal de Santa Catarina

Documento assinado digitalmente
GLAUBER GOMES DE OLIVEIRA BRANTE
Data: 23/03/2022 17:24:34-0300
Verifique em https://verificador.iti.br

Prof. Glauber Brante, Dr.
Universidade Tecnológica Federal do Paraná

Dedico este trabalho a minha família.

AGRADECIMENTOS

Eu poderia escrever um clichê para os agradecimentos, e estaria sendo sincero, mas quero lembrar aqui de um acontecimento que levou à existência deste trabalho.

Numa viagem de carro para Joinville, minha mãe me perguntou sobre quanto faltava para eu me formar. Faltava apenas o TCC para mim, mas por uma série de particularidades, eu não conseguia fazê-lo, nem pedir ajuda para isso. Foi um momento desconfortável, mas fui acolhido, por minha mãe e por toda a minha família.

Esse movimento foi necessário para que eu procurasse o Professor Richard Demo Souza, que aceitou me orientar em um trabalho de conclusão de curso. Ele chamou o Willian Henrique, meu colega no Instituto Senai de Inovação, para pegar junto na orientação. Agradeço a eles pelas críticas, sugestões, acompanhamento. Foram essenciais para a conclusão deste TCC.

Passei por momentos difíceis até chegar aqui, mas quando pedi ajuda, vocês -minha família, meus amigos e meus orientadores - estavam todos lá para me ajudar. A vocês, então, meu muito obrigado.

RESUMO

Durante o desenvolvimento de um dispositivo conectado à Internet é possível que erros passem despercebidos pela equipe de desenvolvimento e cheguem ao equipamento final. Numa visão otimista, pode ser apenas uma falha em alguma lógica embarcada, porém, sendo pessimista, também é possível que um bug permita acesso ao dispositivo ou aos servidores. Empresas também podem desejar que os produtos desenvolvidos tenham novas funcionalidades adicionadas com o passar do tempo. Neste trabalho, buscamos desenvolver uma forma onde seja possível que uma atualização de firmware seja enviada para um dispositivo conectado à Internet a fim de resolver as questões citadas, garantindo ainda que todo o processo de comunicação para obter esta atualização seja por um meio seguro. Foi utilizado um microcontrolador STM com um modem LTE como dispositivo alvo deste projeto. Utilizamos a plataforma Amazon Web Services (AWS) para todas as tarefas em nuvem. O trabalho discute todos os serviços do AWS utilizados no processo, além de como eles se relacionam, também como o firmware entra em contato com a nuvem para verificar a existência de uma atualização, todo o processo de download e como lida com eventuais erros durante o processo. Também aborda como que, após baixada, a atualização é aplicada no dispositivo.

Palavras-chave: Atualização de Firmware. Microcontrolador. Amazon Web Services.

ABSTRACT

During the development of a device connected to the Internet it is possible that errors remain unnoticed by the development team and make their way to the final device. In a optimistic view, it may be just a flaw logic in the embedded logic, however, being pessimistic, it is also possible that a bug give access to the device or their servers. Companies may also want the products to have new functionalities added over time. In this work, we aimed to develop a way to deliver a firmware update to a device connected to the Internet in order to solve the above issues, with the assurance that all communication processes were made in a secure channel. We used a STM microcontroller with a LTE modem as a target of this project. We also used the Amazon Web Services (AWS) platform for all cloud tasks. The work discuss all AWS services used in the update process and how they relate to each other. The work also discuss how the firmware and the cloud interact to check if an update is available, how it is downloaded, how errors are handled and how the downloaded update is applied to the device.

Keywords: Firmware update. Microcontroller. Amazon Web Services.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

AWS - Amazon Web Services

CRC - Cyclic Redundancy Check

GPS - Global Positioning System

HTTP - Hypertext Transfer Protocol

IoT - Internet of Things

ISI-SE - Instituto Senai de Inovação em Sistemas Embarcados

ITU - International Telecommunication Union

ITU-T - ITU Telecommunication Standardization Sector

JSON - JavaScript Object Notation

LTE - Long Term Evolution

MQTT - Message Queueing Telemetry Transport

SD - Secure Digital

SDK - Software Development Kit

SPI - Serial Peripheral Interface

TCP - Transmission Control Protocol

TLS - Transport Layer Security

UART - Universal Asynchronous Receiver/Transmitter

SUMÁRIO

1 INTRODUÇÃO

Dispositivos conectados à Internet já são uma realidade na sociedade atual e novos equipamentos são habilitados todos os dias. Essa popularização gera algumas preocupações, principalmente em relação à segurança dos produtos.

Mesmo com o esforço dos desenvolvedores, falhas de segurança podem passar despercebidas em equipamentos que foram colocados em uso. Uma falha, se descoberta por uma pessoa má intencionada, pode gerar roubo de dados confidenciais ou mesmo o “sequestro” do equipamento para uso em grandes esquemas de ataque, como DDoS.

Por outro lado, uma empresa pode lançar um produto no mercado com expectativa de adicionar novas funcionalidades, beneficiando seus clientes.

Tanto para remediar falhas de segurança descobertas, quanto para adicionar novas funcionalidades, é preciso, de alguma forma, atualizar o firmware instalado no dispositivo.

1.1 OBJETIVOS

Este trabalho surgiu como uma demanda de um projeto do Instituto Senai de Inovação em Sistemas Embarcados (ISI-SE) e tem como objetivo implementar uma forma de atualizar o firmware de um dispositivo conectado à Internet. De forma mais específica, esta atualização deve acontecer de forma automática. O dispositivo deve receber a nova versão do software pela Internet e aplicá-la automaticamente.

Além de atualizar o firmware do dispositivo, é essencial garantir que o meio por onde as comunicações e os downloads serão feitos devem ser seguros e criptografados, de forma que a comunicação, se interceptada, não seja decodificada nem alterada.

De forma mais específica, será utilizada uma placa baseada no microcontrolador STM32F446RE que utilizará um modem Quectel BG96 para conexão à Internet. Será utilizada também a plataforma Amazon Web Services (AWS) para fornecer o firmware ao dispositivo. Como o projeto do ISE-SE que deu

origem a este trabalho já utilizava esta plataforma de computação em nuvem, continuamos com o uso da mesma. Outras plataformas, como Google Cloud ou Microsoft Azure também possuem compatibilidade com dispositivos IoT, porém cada uma possui particularidades que impedirão uma replicação deste trabalho sem adaptações.

2 MATERIAIS E FERRAMENTAS

Este trabalho necessitou o desenvolvimento de um aplicação para a plataforma STM32F446RE que foi dividida em dois firmwares independentes: o bootloader e o programa principal. Também foi realizado o desenvolvimento de uma aplicação na plataforma de computação em nuvem Amazon Web Service (AWS), responsável por distribuir a atualização para o dispositivo e gerenciar a comunicação.

Para que se pudesse realizar o desenvolvimento, foram utilizados os seguintes materiais:

- Placa de Desenvolvimento STM32 Nucleo64 F446RE[1];
- Módulo LTE Quectel BG96[2];
- Módulo para Cartão Secure Digital (SD).

Em termos de software e/ou serviços, os seguintes foram utilizados:

- ARM Mbed[3];
- Biblioteca ArduinoJson[4];
- Biblioteca TinyGSM[5];
- AWS IoT Core;
- AWS S3;
- AWS Lambda.

2.1 HARDWARE

O hardware foi escolhido de forma a facilitar o desenvolvimento do projeto, visto que são componentes já utilizados em projetos no Instituto Senai de Inovação em Sistemas Embarcados (ISI-SE), que tanto o autor quanto o orientador do trabalho já possuíam familiaridade.

---

[1] https://www.st.com/en/evaluation-tools/nucleo-f446re.html
[2] https://www.quectel.com/product/lpwa-bg96-cat-m1-nb1-egprs
[3] https://os.mbed.com/
[4] https://arduinojson.org/
[5] https://github.com/vshymanskyy/TinyGSM

2.1.1 Nucleo64 F446RE

A placa Nucleo64 F446RE é um kit de desenvolvimento baseado no microcontrolador STM32F446RE, que utiliza a arquitetura ARM Cortex-M4, além de 128 kb de memória SRAM e 512 kb de memória para armazenamento do programa. (STMICROELECTRONICS, 2022a; STMICROELECTRONICS, 2022b).

A Figura 1 contém uma imagem da placa Nucleo64 F401RE, porém, visualmente, a placa é idêntica à F446RE, usada neste trabalho.

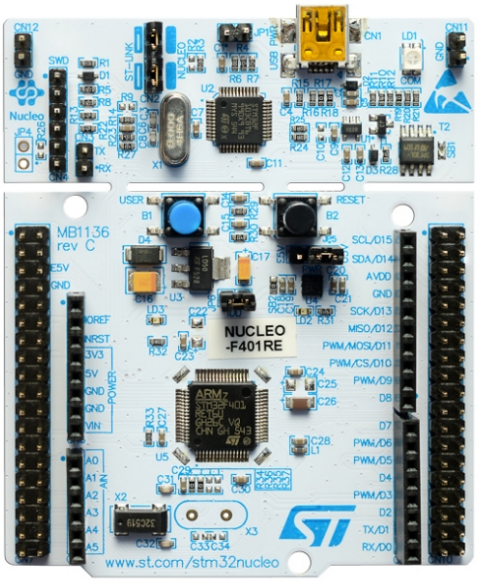

Figura 1 - Placa Nucleo64 F401RE
(STMICROELECTRONICS, 2022a)

O desenvolvimento do projeto iniciou com o kit de desenvolvimento citado, porém foi substituído por uma placa personalizada, desenvolvida no ISI-SE.

A placa desenvolvida também foi baseada no microcontrolador STM32F446RE e se conecta ao Modem Quectel BG96 por meio da Universal Asynchronous Receiver/Transmitter (UART) e ao cartão SD por meio da Serial Peripheral Interface (SPI), conforme exibido na Figura 2. O debugger ST-Link, presente no kit de desenvolvimento Nucleo64 F446RE continuou sendo utilizado para gravações na placa personalizada.

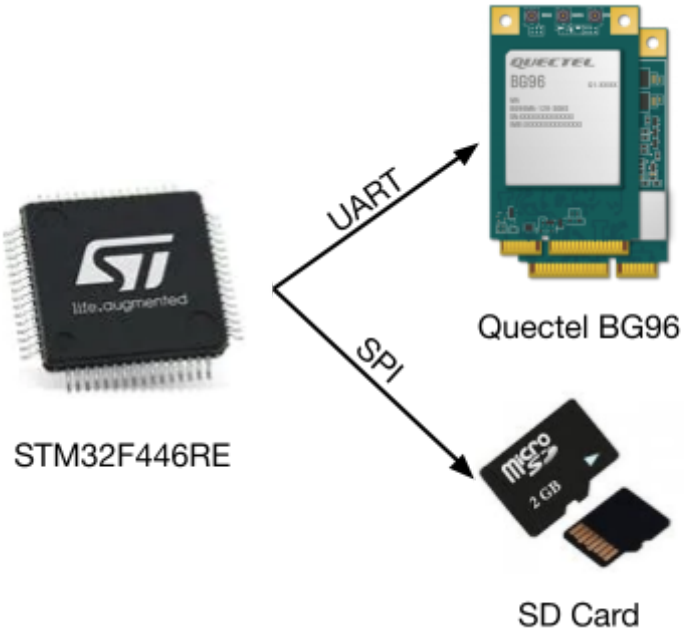

Figura 2 - Conexões da placa personalizada

Fonte: O Autor

2.1.2 Quectel BG96

Para comunicação com a Internet foi escolhido o módulo Quectel BG96. Este módulo possui capacidade de se conectar à rede celular 4G usando os padrões Long Term Evolution (LTE) CAT M1 e LTE CAT NB1, além de poder utilizar redes 2G caso não haja cobertura 4G disponível (QUECTEL, 2020).

2.2 SOFTWARE EMBARCADO

O software embarcado foi desenvolvido usando Mbed, dado que o autor já possuía experiência prévia nesta plataforma de desenvolvimento de firmwares. Também foram utilizadas bibliotecas de terceiros para lidar mais facilmente com o modem LTE e com dados em formato JavaScript Object Notation (JSON).

2.2.1 Mbed

O Mbed é um framework para desenvolvimento de software embarcado em dispositivos Cortex-M criado pela própria ARM. O grande foco do Mbed é ser uma plataforma em que seja fácil desenvolver aplicações para estes microcontroladores,

sendo fácil a migração de um programa para diferentes modelos de microcontroladores (ARM LIMITED, 2022a; ARM LIMITED, 2022b).

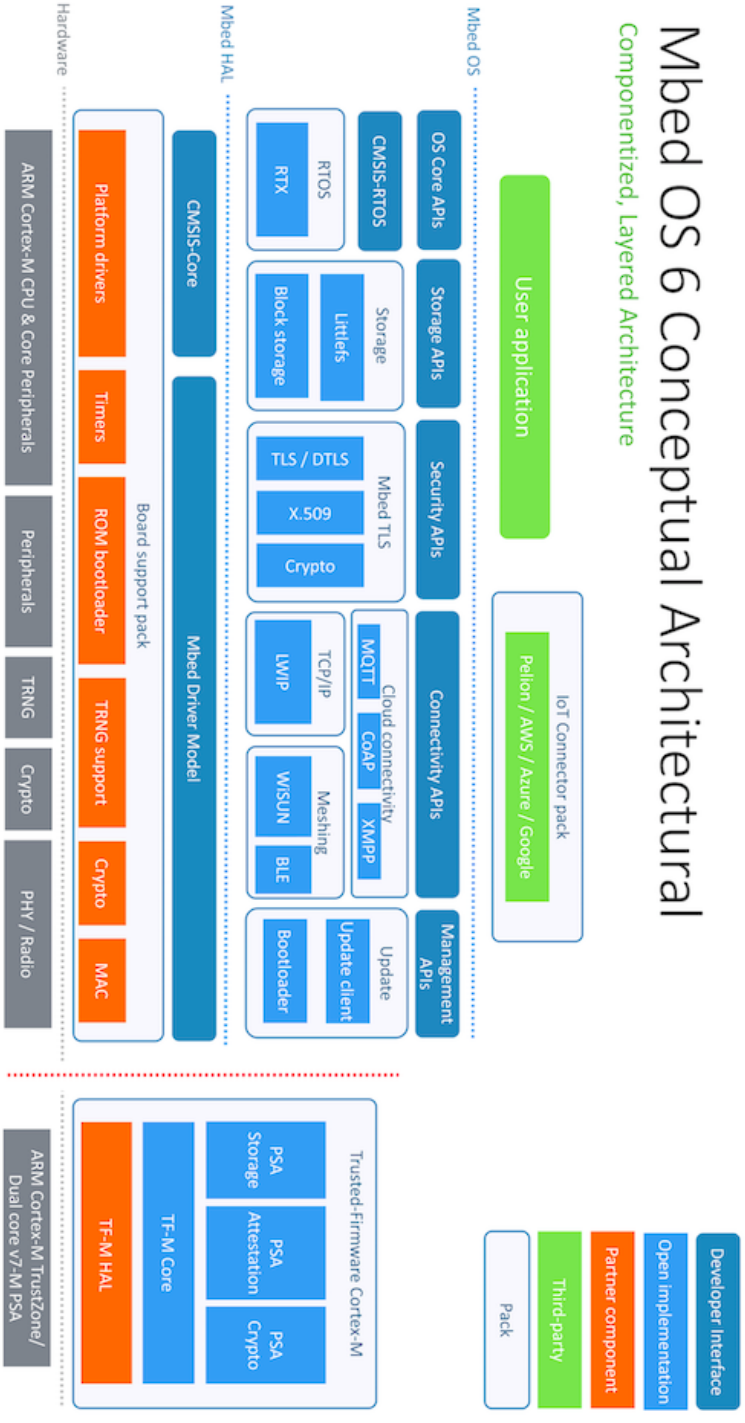

Figura 3 - Arquitetura do Mbed OS
(ARM LIMITED, 2022c)

Como as fabricantes dos microcontroladores também são envolvidas no desenvolvimento do Mbed (ARM LIMITED, 2022c), os drivers e módulos para uso das interfaces dos microcontroladores já estão integrados e são transparentes para o programador, ou seja, não é necessário durante o desenvolvimento que o programador se preocupe com detalhes específicos de cada família de microcontroladores para configuração de uma funcionalidade. A Figura 3 mostra a arquitetura do projeto Mbed OS.

2.2.2 Bibliotecas de Terceiros

Para a comunicação com o modem, foi utilizada a biblioteca TinyGSM, que é uma biblioteca desenvolvida originalmente para uso com o Arduino (e compatíveis). A biblioteca permite o uso fácil de protocolos como Hypertext Transfer Protocol (HTTP), Transmission Control Protocol (TCP), Global Positioning System (GPS), entre outros (SHYMANSKYY, 2022). Foi portada para o Mbed pelo Orientador deste trabalho.

Como a comunicação entre o dispositivo e a AWS é feita utilizando documentos no padrão JSON, a biblioteca ArduinoJson facilitou bastante para lidar com este formato de dados. Apesar do nome, a biblioteca é compatível com qualquer programa em C++ (BLANCHON, 2022).

2.3 AMAZON WEB SERVICES

O Amazon Web Services (AWS) é uma das principais plataformas de computação em nuvem disponíveis hoje. Possui mais de uma centena de serviços disponíveis para os clientes, além de servidores espalhados por todos os continentes (AMAZON WEB SERVICES, 2022a).

Dentre a grande quantidade de serviços disponíveis, foram utilizados no projeto o IoT Core, Lambda e S3.

2.3.1 IoT Core

O IoT Core pode ser visto como uma porta de entrada de dispositivos IoT para os demais serviços do AWS. As mensagens enviadas pelo dispositivo podem

ser facilmente encaminhadas para os demais serviços (AMAZON WEB SERVICES, 2022b; AMAZON WEB SERVICES, 2022c).

A comunicação entre o dispositivo e o AWS pode ser feita por meio de Message Queueing Telemetry Transport (MQTT), HTTP ou LoraWAN (AMAZON WEB SERVICES, 2022c). Para a execução deste projeto, foi utilizada comunicação via MQTT.

2.3.2 Lambda

O AWS Lambda é um serviço que permite que o usuário desenvolva programas que podem ser executados na nuvem sem a preocupação em alocar servidores (AMAZON WEB SERVICES, 2022d).

Outros serviços do AWS podem gerar gatilhos que chamam a execução de uma função Lambda (AMAZON WEB SERVICES, 2022d). No caso deste trabalho, uma mensagem enviada pelo dispositivo, via MQTT, para o IoT Core gera um gatilho que executa uma função no AWS Lambda.

O AWS Lambda suporta desenvolvimento de programas em linguagens como Python, Ruby, Go, C#, entre outras (AMAZON WEB SERVICES, 2022k). No desenvolvimento deste projeto, utilizamos um programa desenvolvido em Python.

2.3.3 S3

O Amazon S3 é um serviço de armazenamento de arquivos que pode ser integrado aos demais serviços do AWS. Os arquivos são organizados em buckets (ou containers) e acessados por meio de keys (em geral o nome do arquivo). Cada bucket pode ter uma política de segurança específica (AMAZON WEB SERVICES, 2022e), por exemplo: um bucket pode ter acesso liberado para qualquer pessoa pela Internet, mas outro pode ser restrito apenas ao usuário que o criou.

Para o projeto foi criado um bucket que contém três arquivos: firmware, versão e códigos de verificação cíclica de redundância (CRC).

2.4 PROTOCOLOS

Diversos protocolos são utilizados na comunicação e na forma de lidar com dados, de forma que acabam sendo praticamente transparentes para o usuário.

Esta seção lista os principais protocolos utilizados no desenvolvimento deste projeto.

2.4.1 SEGURANÇA

Segurança é um ponto essencial do projeto. É preciso garantir que apenas dispositivos conhecidos conectem-se aos servidores do AWS e que, caso a comunicação seja interceptada, os dados não possam ser decodificados pelo interceptador, visto que a comunicação entre um dispositivo e a nuvem pode conter informações sensíveis e/ou segredos industriais.

Os dois principais pontos para a segurança do projeto são o uso de certificados X.509 para a autenticação de identidade dos dispositivos e o uso do Transport Layer Security (TLS) nas comunicações via Internet.

2.4.1.1 X.509

A X.509 é uma norma da ITU que trata de certificados de chave pública. Um certificado X.509 é emitido por uma Autoridade Certificadora (CA) e contém, além da chave, uma data de validade anterior e posterior, versão e algoritmo utilizado.

O certificado gerado pode ser distribuído livremente, porém a chave privada que foi usada para gerá-lo deve ser guardada e ser conhecida apenas pelo dispositivo que será representado pelo certificado. (ITU-T, 2019)

2.4.1.2 TLS

O TLS (Transport Layer Security) é um protocolo que visa garantir segurança nas comunicações via Internet. Deve garantir a autenticação ao menos do servidor, a confidencialidade e integridade dos dados.

É composto, em essência, por um handshake - que é o momento em que as informações sobre certificados, capacidades de criptografia, entre outras informações são trocadas, exemplificado na Figura 4 - e o record, que usa os parâmetros estabelecidos no handshake para proteger a comunicação (RESCORLA *et al.*, 2018).

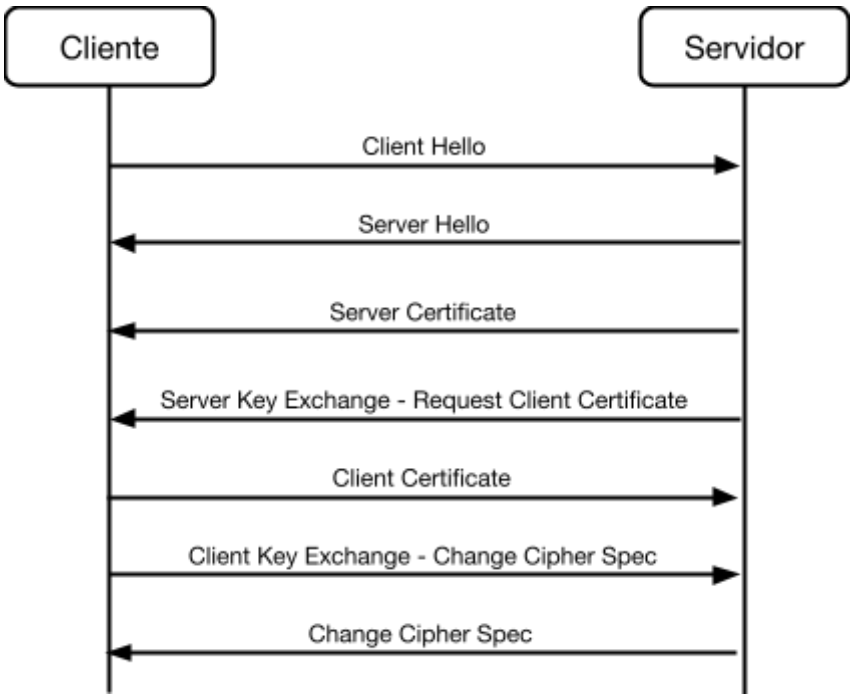

Figura 4 - Handshake TLS

Fonte: O Autor

As etapas do handshake permitem o estabelecimento de uma comunicação segura entre as pontas. Uma falha neste processo a inviabiliza.

Em “Client Hello”, o dispositivo inicia o processo enviando, entre outros dados, os modos de criptografia e autenticação suportados (cipher suite), sendo respondido com um “Server Hello”, onde o servidor seleciona um dos métodos enviados pelo cliente. Num cliente em Python, desenvolvido para testes, é escolhida a Cipher Suite TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, que também é suportada pelo modem Quectel BG96.

Logo na sequência, o servidor enviará o seu certificado, uma mensagem contendo informações sobre criptografia (Server Key Exchange) e pedirá o certificado do cliente (Request Client Certificate).

O cliente então responde com o seu certificado e envia as suas informações de criptografia (Client Key Exchange).

A mensagem “Change Cipher Spec” indica que a negociação do TLS já ocorreu e que a partir deste momento as mensagens entre as pontas estarão criptografadas.

2.4.2 COMUNICAÇÃO

Toda a comunicação entre o dispositivo e o AWS é feita por meio de mensagens MQTT. Todo serviço fornecido pela Amazon, ou funcionalidades criadas para um projeto são feitas por cima deste protocolo.

2.4.3 MQTT

É um protocolo desenvolvido inicialmente nos anos 1990 pela IBM, porém hoje é um padrão aberto (YUAN, 2021).

Ainda de acordo com Yuan (2021), a rede é composta por duas entidades: um cliente e um broker. O cliente se conecta ao broker e pode assinar tópicos para receber mensagens deles, além de enviar mensagens para tópicos. O broker recebe as mensagens enviadas pelos clientes e as encaminha para todos os clientes que assinaram aquele tópico.

O MQTT é uma das formas de comunicação usadas pelo AWS IoT Core (AMAZON WEB SERVICES, 2022c) e foi a escolhida para utilização neste projeto.

2.4.4 BASE64

O padrão Base64 permite que se transmita informação binária por meio de caracteres ASCII. Dos 65 caracteres disponíveis, 64 são utilizados para representação de dados, sendo que cada um representa seis bits de informação. O 65º caractere geralmente representa o fim de uma sequência (JOSEFSSON, 2006).

Como cada caractere (que possui um byte) é equivalente a apenas seis bits, há um aumento na banda necessária para transmitir uma mensagem em comparação ao dado binário bruto. Por exemplo, um dado binário de 2 kbytes precisará de 2.731 caracteres para ser transmitido, aproximadamente 2,67 kbytes.

3 IMPLEMENTAÇÕES

Esta seção do trabalho detalha como as aplicações foram construídas e a forma de funcionamento.

3.1 AWS

Em todo o processo de atualização, os serviços do AWS são utilizados de forma combinada para fornecer o serviço desejado.

3.1.1 Autenticação de Dispositivos

A autenticação dos dispositivos é realizada durante o handshake do TLS, sendo que, para cada dispositivo conectado, é esperado um certificado do tipo X.509 único. Estes certificados podem ser emitidos diretamente pela Amazon, ou por qualquer outra autoridade certificadora. Pode ser vinculada a cada certificado de dispositivo uma política única de permissão/restrição de acesso à serviços no AWS (AMAZON WEB SERVICES, 2022j).

A identidade do dispositivo é verificada pelo AWS durante o handshake do TLS, no início da conexão. Toda a comunicação pós-autenticação se dá por meio de um canal protegido pelo protocolo TLS 1.2.

3.1.2 Bucket

Embora sempre acessado de forma indireta, o S3 é uma ferramenta chave para o processo. A Figura 5 representa o bucket “tmd-file-storage-test”, que foi utilizado para o desenvolvimento do projeto.

No bucket, o arquivo “firmware.bin” é o próprio firmware disponível para download, que será baixado em blocos de 2 kbytes. O arquivo “version.txt” é um arquivo de texto que contém apenas o número da versão do firmware. Já “split.json” é um arquivo que contém o código CRC para todas as partes de 2 kbytes do firmware, além de campos com o número total de partes e o tamanho total do firmware.

Figura 5 - Bucket tmd-file-storage-test

Fonte: O Autor

3.1.3 Solicitação de Atualização

A Figura 6 mostra como acontece o processo de solicitação de uma atualização.

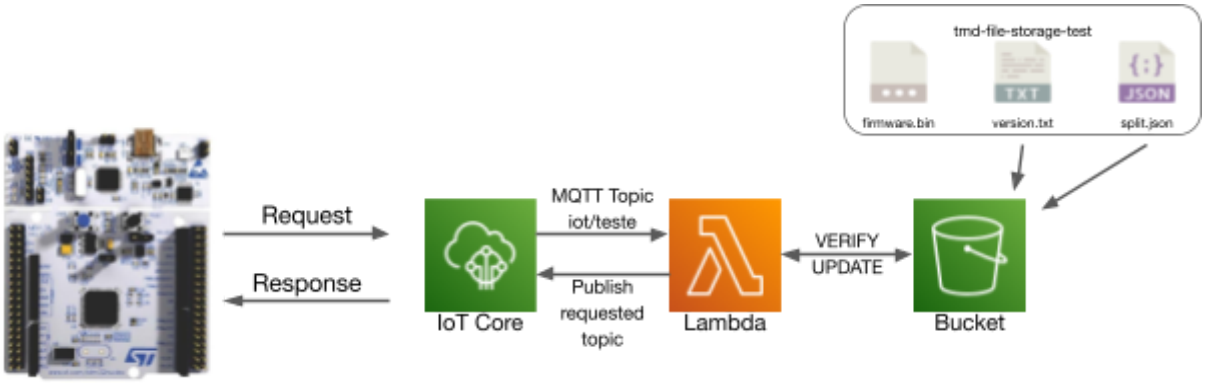

Figura 6 - Solicitação de atualização

Fonte: O Autor

O dispositivo envia ao AWS uma mensagem de acordo com o padrão comentado na seção **“Protocolo de Comunicação com AWS”**. Quando a mensagem chega no broker do IoT Core ela é filtrada. Com uma mensagem enviada para o tópico “iot/teste” é ativado um gatilho que encaminha a mensagem para uma função Lambda.

A função, verificando que o código de ação é correspondente à ação “VERIFY_UPDATE”, lê os arquivos “version.txt” e “split.json” do Bucket “tmd-file-storage-test” e monta uma resposta, que será enviada de volta para o dispositivo pelo tópico especificado na primeira mensagem. Um diagrama mais detalhado do processo é mostrado na Figura 7.

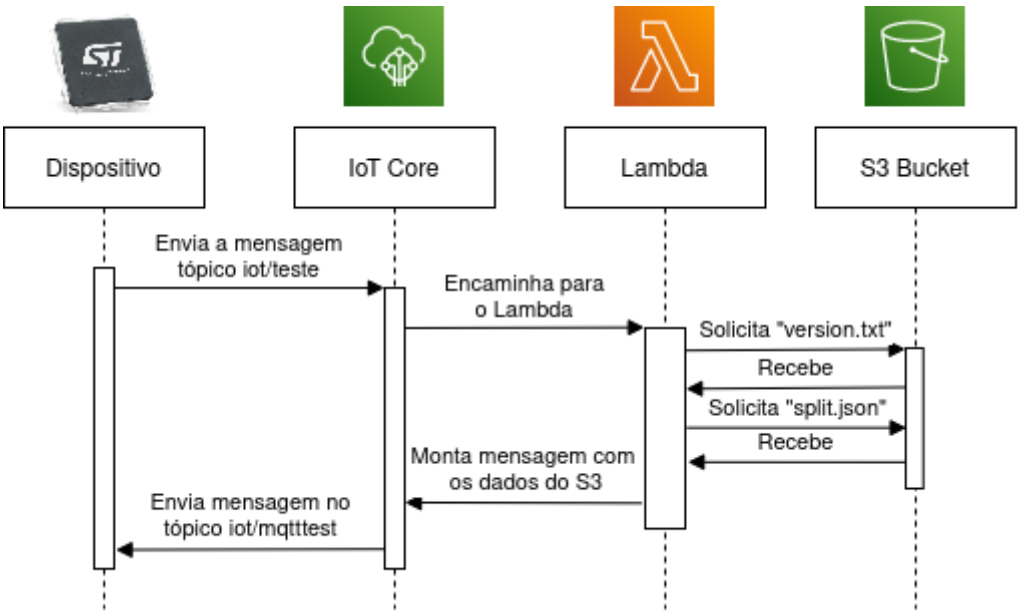

Figura 7 - Diagrama de sequência da solicitação de atualização

Fonte: O Autor

A verificação se a versão disponível é mais recente que a instalada é feita localmente pelo dispositivo.

3.1.4 Transporte de Arquivos por MQTT

O AWS Iot Core prevê uma funcionalidade para transferência de arquivos via MQTT, com a possibilidade de transferências codificadas em Base64 ou binária (AMAZON WEB SERVICES, 2022f). Os arquivos que serão transmitidos para o dispositivo devem estar em um bucket do S3 (AMAZON WEB SERVICES, 2022i), no caso será utilizado o bucket “tmd-file-test-storage”, o mesmo acessado pelo Lambda. Por questão de simplicidade, neste projeto foi adotada a transmissão codificada em Base64, mesmo consumindo mais recursos de rede para transmissão.

Dado que, ao contrário dos demais serviços do AWS utilizados, a transmissão de arquivos via MQTT não pode ser configurada pelo dashboard do AWS, apenas via linha de comando ou pelo Software Development Kit (SDK) (AMAZON WEB SERVICES, 2022g), a configuração da transmissão aconteceu de forma bastante manual, algo que deve ser revisto para uma aplicação final. Um breve guia das configurações está disponível no Apêndice B.

A Figura 8 mostra um esquema de como funciona o transporte de arquivos via MQTT para um dispositivo conectado ao IoT Core.

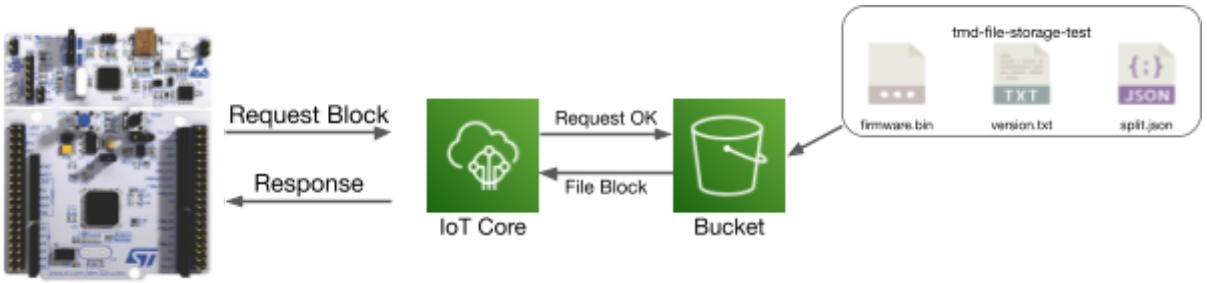

Figura 8 - Requisição de arquivo via MQTT

Fonte: O Autor

Como esta funcionalidade já é prevista dentro do IoT Core, há uma forma bem definida de como devem ser feitas as requisições para download de um bloco do arquivo. De acordo com a documentação do Amazon Web Services (2022h), as mensagens enviadas como uma requisição de um bloco tem alguns parâmetros obrigatórios e outros opcionais. Para o projeto, os seguintes parâmetros foram utilizados:

- “f”: Indica qual o arquivo que será baixado. No projeto tem sempre valor 1.
- “l”: Indica o tamanho do bloco que será baixado. Tem sempre o valor 2048.
- “n”: Número de blocos que será recebido de uma vez. Sempre possui valor 1.
- “o”: Indica qual bloco de tamanho l será baixado. Apenas este parâmetro é alterado em cada mensagem.

Exemplificando este último parâmetro: no caso de um firmware que possua tamanho de 101 kbytes (103424 bytes), haverão 51 blocos para download. Quando o valor de “o” for 0, serão recebidos os bytes 0 até 2047. Quando “o” for um, serão recebidos os bytes 2048 até 4095. Seguindo assim até o último bloco (“o” = 50), que neste exemplo será menor que 2 kbytes, recebendo do byte 102400 até 103424.

A resposta do AWS tem um padrão parecido com o da requisição e contém os seguintes parâmetros:

- “f”: Indica o arquivo.
- “i”: ID do bloco. crescente, começando em 0.
- “l”: Tamanho do bloco.
- “p”: O bloco do firmware baixado.

O firmware é entregue em forma de texto, codificado no formato Base64. O dispositivo deve decodificá-lo.

3.2 BOOTLOADER

No contexto do Mbed OS, o bootloader é criado como uma aplicação regular, porém com tamanho restrito. A principal tarefa do bootloader é carregar a aplicação principal, porém é possível que o mesmo seja programado para aplicar atualizações de software no dispositivo (ARM LIMITED, 2022d).

A Figura 9 mostra como fica a organização da memória do microcontrolador com um bootloader (fora de escala).

Figura 9 - Organização da memória do microcontrolador
Fonte: O Autor

No projeto, foi utilizado um dispositivo baseado no microcontrolador STM32F446RE, que possui uma memória interna de 512 kbytes, dividida em sete setores de diferentes tamanhos (STMICROELECTRONICS, 2021). Foram reservados os três primeiros setores para o bootloader, totalizando 48 kbytes. O espaço restante (até 464 kbytes) está disponível para a aplicação principal.

3.2.1 Memória Externa

O firmware baixado pelo programa principal será armazenado num cartão SD. A Figura 10 (fora de escala) mostra como deve estar organizada a memória.

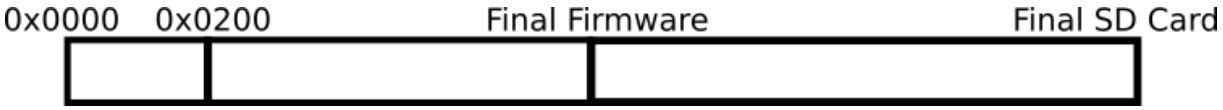

Figura 10 - Organização do Cartão SD.

Fonte: O Autor

Devido à características dos cartões SD, as leituras e escritas são feitas a partir de um comprimento múltiplo do valor de um setor, que no caso do cartão utilizado é de 512 bytes.

O primeiro setor do cartão é reservado para indicar se existe alguma atualização a ser aplicada. Esta indicação é feita a partir da escrita do comprimento da atualização nos primeiros quatro bytes, de forma que o programa possa ler este valor como um valor do tipo uint32. O firmware é armazenado a partir do endereço 0x0200.

Como o firmware é limitado em até 464 kbytes, o restante do cartão SD poderia ser utilizado para outras necessidades da aplicação.

3.2.2 Processo de Atualização

A Figura 11 contém um diagrama do processo de atualização da placa.

Inicialmente a placa fará a leitura dos primeiros quatro bytes do cartão SD para verificar se existe uma atualização. Caso o valor lido seja 0xFFFFFFFF, o bootloader interpretará o valor como se não houvesse uma atualização e iniciará o programa que já está escrito na memória interna. Caso exista um valor diferente, o bootloader entende que existe uma atualização pronta para ser aplicada e que o valor lido é o tamanho do firmware armazenado no cartão SD.

Este método para verificar a existência de uma atualização tem alguns problemas, por exemplo, pode haver algum “lixo” neste endereço, o que não é verificado, além da possibilidade de acesso ao cartão SD. Hoje estamos considerando que não há acesso físico ao hardware e que o cartão SD é sempre

inserido num estado adequado, sendo completamente preenchido com valor 0xFF. Esta verificação deve sofrer modificações em futuras versões deste projeto.

Considerando a existência de uma atualização, toda a memória do microcontrolador após o bootloader será apagada, de forma que novos dados possam ser escritos.

O processo de escrita é feito em blocos de 1 kbyte. O bootloader lê o primeiro bloco a partir do endereço 0x0200 no cartão de memória e o escreve a partir do endereço 0x0800C000 na memória interna.

Após a escrita, é verificado se o dado lido do cartão SD é igual ao escrito na memória interna. Para isso, é feito uma operação XOR entre todos os bytes do bloco lido do cartão SD e da memória interna. Os valores devem ser iguais para os dois casos.

Caso o valor obtido na operação seja diferente, o bloco é reescrito na memória interna. O processo pode ser repetido até que se obtenha a igualdade.

Assegurado que os dados são iguais, o bootloader acrescenta o tamanho do bloco nas variáveis que indicam os endereços de leitura no SD, escrita na memória interna e em uma variável que indica o total de bytes escritos.

Enquanto o total de bytes escritos for menor que o tamanho do novo firmware, o bootloader seguirá escrevendo novos blocos na memória interna.

Terminada a escrita, o bootloader inicia o novo programa.

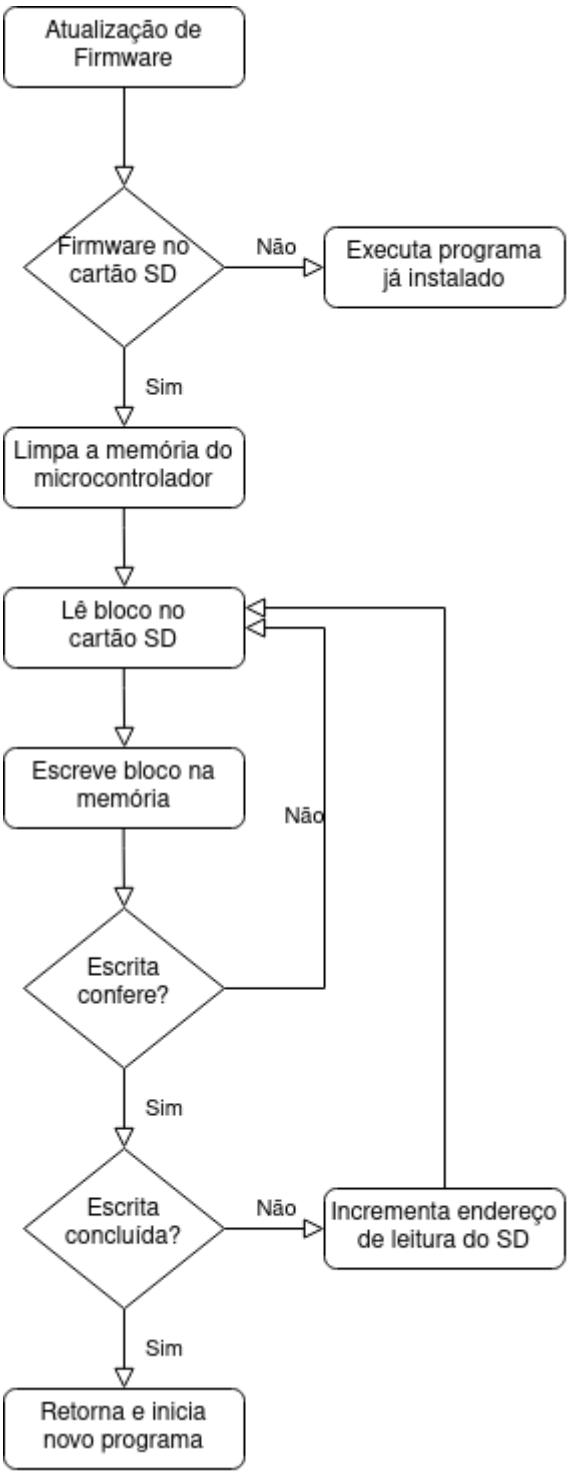

Figura 11 - Diagrama do processo de atualização

Fonte: O Autor

3.3 PROGRAMA PRINCIPAL

Apesar da importância, o bootloader é executado rapidamente. Na maior parte do tempo o dispositivo estará executando outro programa, que aqui chamamos de “Programa Principal”. É neste programa que o dispositivo realizará todas as tarefas para as quais foi projetado. Neste projeto, a única função do programa principal é justamente verificar a existência de atualizações de firmware.

3.3.1 Estrutura do Programa

O código original do programa, isto é, sem as bibliotecas de terceiros que foram utilizadas, foi organizado da seguinte forma:

- main;
- MainSystem;
- Updater;
- CommonData.

Todo programa C/C++ precisa de uma função main. Neste caso ela é utilizada apenas para configurações iniciais do programa, como a serial que é utilizada para debug, passando logo em sequência para MainSystem.

MainSystem executa o loop principal do programa. Na aplicação deste projeto a única tarefa executada é a verificação de atualizações. Em programas mais complexos aqui seria introduzida toda a lógica desejada.

CommonData e Updater serão abordadas nas seções seguintes.

3.3.2 Protocolo de Comunicação com AWS

Para a comunicação entre o dispositivo e o AWS foi criado um pequeno protocolo. A comunicação entre o dispositivo e o AWS é iniciada sempre pelo dispositivo, que envia uma mensagem com os seguintes parâmetros: “require_response”: que indica que a mensagem precisa ou não de uma resposta, “response_topic”: que indica o tópico MQTT onde a resposta deve ser enviada, caso exista, “action”: um código que indica a ação que será feita e “data”: que é algum dado que pode ser enviado junto.

Em caso de resposta é esperada a seguinte estrutura: “action”: que é o mesmo código da ação enviado na requisição, “lengh_or_version”: que indica o tamanho do dado enviado ou a versão do firmware disponível e “data”: que contém dados enviados.

No momento, este protocolo é utilizado apenas para o firmware perguntar ao AWS se existe alguma atualização disponível, porém foi desenvolvido pensando na possibilidade de ser utilizado em outros locais do programa, como envio de dados de sensores, por exemplo.

Como o AWS trabalha com dados em JSON, a conversão deste formato para uma Struct acontece dentro de CommonData.

3.3.3 Download da Atualização

O módulo Updater do firmware é o ponto central deste projeto. É o responsável por fazer o download da atualização disponível. Este processo pode retornar três valores:

- NO_UPDATE: quando não existe uma atualização disponível;
- FAILED: quando o processo, por algum motivo, falha e deve ser reiniciado;
- UPDATE_OK: quando o download foi bem sucedido.

A Figura 12 mostra a primeira parte do processo de download de uma atualização de firmware.

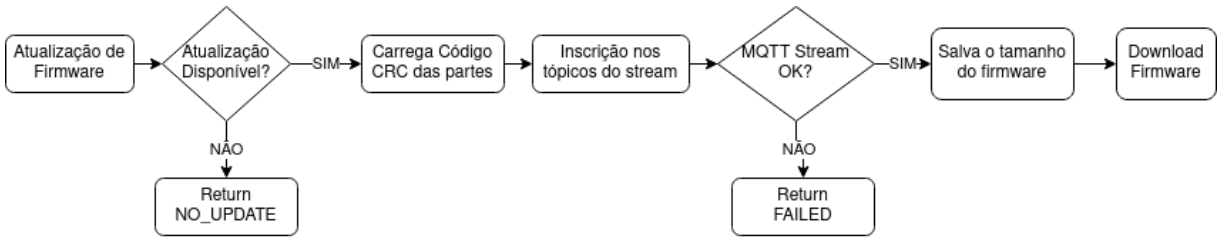

Figura 12 - Diagrama do processo de download. Parte 1

Fonte: O Autor

Inicialmente o firmware envia uma requisição, no formato discutido na seção anterior, para verificar a existência de uma atualização. O AWS retornará com um

JSON contendo o número da nova versão e o CRC de cada bloco de 2 kbytes. Caso a versão disponibilizada pelo AWS seja igual ou menor a do dispositivo, a função retornará com “NO_UPDATE” e o programa principal segue com as tarefas previstas. Caso a versão seja mais recente, o programa irá carregar cada CRC em um array para acesso mais fácil durante a execução do download.

Seguindo o processo, existem quatro tópicos MQTT que o dispositivo deve se inscrever para a transferência de arquivos (AMAZON WEB SERVICES, 2022h), sendo que estes tópicos são exclusivos para cada dispositivo conectado ao IoT Core. Após inscrito nos tópicos, o dispositivo enviará uma mensagem para verificar a existência do stream MQTT. Caso tudo esteja correto, será salvo o tamanho do novo firmware em uma variável da classe e o programa segue conforme a Figura 13. Caso haja algum problema, o atualizador retornará ao programa principal com valor FAILED.

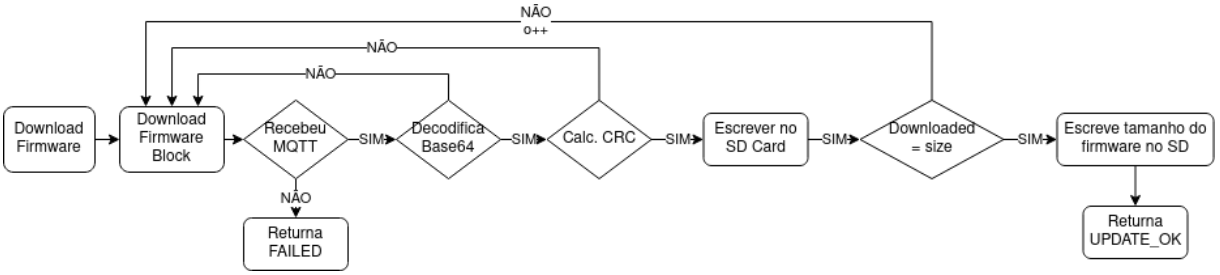

Figura 13 - Diagrama do processo de download. Parte 2

Fonte: O Autor

Com todas as verificações feitas, o programa pode iniciar o download da nova versão do firmware.

Será enviada uma mensagem ao IoT Core, conforme citado em “Transporte de Arquivos por MQTT”, com o valor de “o” sendo 0. Após enviar a mensagem, o dispositivo espera que uma mensagem chegue no tópico MQTT correto. Existe, neste momento, a possibilidade da conexão com os servidores falhar, assim o modem envia ao dispositivo uma mensagem de erro na conexão MQTT, fazendo com que o dispositivo encerre toda a conexão com os servidores do AWS e reinicie o processo, ressaltando que o progresso até o momento é mantido. Por exemplo,

se o atualizador já havia baixado oito blocos e a conexão falhou no nono bloco, com a reconexão o download será reiniciado a partir do nono bloco.

Recebendo a mensagem corretamente, ela será entregue em formato JSON, com o conteúdo codificado em Base64, sendo necessário decodificar este conteúdo. Caso a decodificação falhe, é solicitado um novo download do bloco.

Com os dados brutos após a decodificação é possível fazer a verificação do CRC do bloco baixado, sendo então comparado com o valor armazenado correspondente. Caso os valores não sejam iguais, é requisitado um novo download do bloco.

Com os dados conferidos e corretos, é possível escrever o bloco na memória externa (cartão SD), a partir do endereço 0x200. Após escrito, é efetuada uma leitura no cartão SD. O buffer que contém os dados que foram escritos é comparado com o buffer dos dados lidos. Caso haja diferença, os dados serão reescritos na memória. A execução segue caso sejam iguais.

Neste momento é verificado se o download de todas as partes foi concluído. Caso não, o valor da variável “o”, utilizada na requisição para o AWS é acrescido de 1. Caso já tenha sido concluído, será escrito nos primeiros quatro bytes do cartão SD o tamanho do novo firmware (em bytes) e retorna com o código “UPDATE_OK” para o loop principal. Neste momento fica a cargo do programador decidir, mas recomenda-se que o dispositivo seja imediatamente reiniciado (o que é possível fazer por software) para que o bootloader aplique a atualização.

3.4 TESTES E RESULTADOS

Para testar o projeto o firmware foi gravado na placa pelo método convencional (debugger ST-Link) e um firmware com número de versão diferente foi enviado ao Bucket do AWS S3, junto com os arquivos “version.txt” e “split.json”. E, na sequência, os arquivos foram substituídos no bucket e o stream foi atualizado, simulando a inclusão de novas versões do firmware.

O firmware desenvolvido em conjunto com a aplicação no AWS foi capaz de atualizar uma placa baseada no microcontrolador STM32F446RE sem interferência

de uma pessoa. Além disso, com algumas alterações, é possível que o projeto seja adaptado para outros modelos de microcontroladores.

Embora o processo de atualização, como um todo, não tenha apresentado falhas durante o desenvolvimento, é importante ressaltar que é comum que ocorram falhas na recepção das mensagens MQTT ou falhas na rede, de forma que o dispositivo aparenta estar em um estado de travamento, porém em cerca de quatro ou cinco minutos é relatado que a conexão com o broker MQTT falhou, permitindo que todo o processo de conexão seja refeito. Desta forma, mesmo com uma demora no download, o firmware sempre foi obtido de forma correta e íntegra.

No Apêndice A está o log que o firmware do dispositivo envia para a UART. Por meio deste log é possível ver o processo de inicialização do dispositivo, que no momento em que é ligado não há uma atualização armazenada.

É possível observar no log a única utilização, até o momento, do protocolo de comunicação definido (quando é verificado se há uma atualização nos servidores) e também todo o processo de download do novo firmware pelo stream. Neste caso, o processo de download aconteceu sem nenhuma falha.

4 CONCLUSÃO

Como apresentado anteriormente, o desenvolvimento deste trabalho atingiu os objetivos propostos: o firmware de uma placa foi atualizado de forma remota e todo o meio de comunicação entre a placa e os servidores do AWS é seguro e criptografado.

Ao longo do desenvolvimento, alguns componentes utilizados foram alterados. Inicialmente havia a ideia de utilizar um modem mais simples, o SIMCom SIM800, que se conectava apenas por GPRS, mas acabou sendo substituído visto que se encontrava em uma “zona cinza” em relação ao suporte ao TLS na versão exigida pelo AWS Alguns documentos citavam o suporte ao padrão TLS 1.2, requerido pelo AWS, porém outros citavam suporte apenas ao padrão TLS 1.1, além da incompatibilidade com o uso de certificados para autenticação. O uso do modem Quectel BG96 resolveu todas estas questões.

Devido a limitações de tempo, o software desenvolvido acabou muito vinculado ao modelo da placa e modem. Certamente é um detalhe que pode ser resolvido futuramente. Como já comentado, as falhas de conexão com o MQTT também poderiam ser melhor tratadas, porém o tempo gasto para realizar a atualização não é uma variável crítica no projeto.

Outra limitação do projeto até o momento é a falta de automatização na nuvem do AWS. Ainda é necessário que sejam enviados manualmente o firmware e todos os códigos CRC das partes.

Como opções para seguir o desenvolvimento deste projeto, destacamos o uso de memórias mais confiáveis, como as FRAM, em substituição ao SD card. No bootloader convém uma revisão do método de verificação da existência de uma atualização. Hoje tal verificação é feita através dos quatro primeiros bytes do cartão SD, o que poderia gerar um alerta falso de que existe atualização. Também pode ser conveniente uma verificação melhor se o firmware salvo no cartão SD está correto.

Embora ainda não esteja pronto para ser utilizado em projetos da vida real, fora do ambiente de desenvolvimento, certamente este trabalho está muito próximo disso. Sabemos que estes ajustes podem levar algum tempo, mas temos confiança

de que ainda será possível integrar este método de atualização como parte do desenvolvimento de um projeto maior.

REFERÊNCIAS

STMICROELECTRONICS. **NUCLEO-F446RE**. 2022a. Disponível em: https://www.st.com/en/evaluation-tools/nucleo-f446re.html. Acesso em: 25 fev. 2022.

STMICROELECTRONICS. **STM32F446**. 2022b. Disponível em: https://www.st.com/en/microcontrollers-microprocessors/stm32f446.html. Acesso em: 25 fev. 2022.

STMICROELECTRONICS. **RM0390**. 2021. Disponível em: https://www.st.com/resource/en/reference_manual/rm0390-stm32f446xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf. Acesso em: 26 fev. 2022.

QUECTEL. **Quectel BG96**. 2020. Disponível em: https://www.quectel.com/wp-content/uploads/2021/03/Quectel_BG96_LPWA_Specification_V1.8.pdf. Acesso em: 25 fev. 2022.

SHYMANSKYY, Volodymyr. **TinyGSM**. Disponível em: https://github.com/vshymanskyy/TinyGSM. Acesso em: 25 fev. 2022.

BLANCHON, Benoît. **ArduinoJson**. Disponível em: https://arduinojson.org/. Acesso em: 25 fev. 2022.

ARM LIMITED. **Mbed OS**. 2022a. Disponível em: https://os.mbed.com/mbed-os/. Acesso em: 25 fev. 2022.

ARM LIMITED. **Introduction**. 2022b. Disponível em: https://os.mbed.com/docs/mbed-os/v6.15/introduction/index.html. Acesso em: 25 fev. 2022.

ARM LIMITED. **Architecture**. 2022c. Disponível em: https://os.mbed.com/docs/mbed-os/v6.15/introduction/architecture.html. Acesso em: 25 fev. 2022.

ARM LIMITED. **Creating and using a bootloader**. 2022d. Disponível em: https://os.mbed.com/docs/mbed-os/v6.15/program-setup/creating-and-using-a-bootloader.html. Acesso em: 26 fev. 2022.

AMAZON WEB SERVICES. **O que é AWS?** 2022a. Disponível em: https://aws.amazon.com/pt/what-is-aws/?nc1=f_cc. Acesso em: 26 fev. 2022.

AMAZON WEB SERVICES. **AWS IoT Core**. 2022b. Disponível em: https://aws.amazon.com/pt/iot-core/?c=i&sec=srv. Acesso em: 26 fev. 2022.

AMAZON WEB SERVICES. **O que é o AWS IoT?** 2022c. Disponível em: https://docs.aws.amazon.com/pt_br/iot/latest/developerguide/what-is-aws-iot.html. Acesso em: 26 fev. 2022.

AMAZON WEB SERVICES. **O que é o AWS Lambda?** 2022d. Disponível em: https://docs.aws.amazon.com/pt_br/lambda/latest/dg/welcome.html. Acesso em: 26 fev. 2022.

AMAZON WEB SERVICES. **O que é Amazon S3**. 2022e. Disponível em: https://docs.aws.amazon.com/pt_br/AmazonS3/latest/userguide/Welcome.html. Acesso em: 26 fev. 2022.

AMAZON WEB SERVICES. **MQTT-based file delivery**. 2022f. Disponível em: https://docs.aws.amazon.com/iot/latest/developerguide/mqtt-based-file-delivery.html. Acesso em: 27 fev. 2022.

AMAZON WEB SERVICES. **Managing a stream in the AWS Cloud**. 2022g. Disponível em: https://docs.aws.amazon.com/iot/latest/developerguide/mqtt-based-file-delivery-managing.html. Acesso em: 27 fev. 2022.

AMAZON WEB SERVICES. **Using AWS IoT MQTT-based file delivery in devices**. 2022h. Disponível em: https://docs.aws.amazon.com/iot/latest/developerguide/mqtt-based-file-delivery-in-devices.html. Acesso em: 27 fev. 2022.

AMAZON WEB SERVICES. **What is a stream?** 2022i. Disponível em: https://docs.aws.amazon.com/iot/latest/developerguide/mqtt-based-file-delivery-what-is.html. Acesso em: 27 fev. 2022.

AMAZON WEB SERVICES. **X.509 client certificates**. 2022j. Disponível em: https://docs.aws.amazon.com/iot/latest/developerguide/x509-client-certs.html. Acesso em: 27 fev. 2022.

AMAZON WEB SERVICES. **Perguntas frequentes sobre o AWS Lambda**. 2022k. Disponível em: https://aws.amazon.com/pt/lambda/faqs/. Acesso em: 21 mar. 2022.

JOSEFSSON, Simon. **RFC 4648 - The Base16, Base32, and Base64 Data Encodings**. 2006. Disponível em: https://datatracker.ietf.org/doc/html/rfc4648. Acesso em: 27 fev. 2022.

YUAN, Michael. **Getting to know MQTT**. 2021. Disponível em: https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot. Acesso em: 27 fev. 2022.

RESCORLA, Eric *et al*. **RFC 8446 - The Transport Layer Security (TLS) Protocol Version 1.3**. 2018. Disponível em: https://datatracker.ietf.org/doc/html/rfc8446. Acesso em: 27 fev. 2022.

ITU-T. **X.509 : Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks**. 2019. Disponível em: https://www.itu.int/rec/T-REC-X.509-201910-I/en. Acesso em: 06 mar. 2022.

APÊNDICE A - SAÍDA NA UART DURANTE O PROCESSO DE ATUALIZAÇÃO

```
Hello World!
No update stored in SD card
Version 2022021210

Init
BG96 Iniciado com sucesso
Modem info: Quectel BG96 Revision:
CCID:
IMEI:
IMSI:
linksolutions.br
GPRS/LTE Connected
Network Connected
Operadora: VIVO VIVO
IP: 10.232.183.59
Signal Quality: 29
connection: 0, result: 0

SD Init
sd size: 7927234560
sd read size: 512
sd program size: 512
sd erase size: 512

Operadora: VIVO VIVO
IP: 10.232.183.59
Signal Quality: 29
AWS topic subscribed: 0
buff json (with library):
{"require_response":true,"action":0,"response_topic":"iot/mqtttest","data":""
}
AWS topic iot/teste publish result.........: 0
data rec:
+QMTRECV: 0,0,"iot/mqtttest","{"action": 0, "lenght_or_version": 2022021215,
"data": {"crc": {"0": 3067313637, "1": 2957856364, "2": 2597385937, "3":
1704859145, "4": 3506744434, "5": 3736105166, "6": 3583040540, "7":
```

1057356574, "8": 4000016840, "9": 3656349762, "10": 2026648505, "11": 1374177095, "12": 2066749866, "13": 444424459, "14": 419199985, "15": 2795493462, "16": 3214828758}, "total_size": 34712, "number_of_parts": 17}}"
json substring:
{"action": 0, "lenght_or_version": 2022021215, "data": {"crc": {"0": 3067313637, "1": 2957856364, "2": 2597385937, "3": 1704859145, "4": 3506744434, "5": 3736105166, "6": 3583040540, "7": 1057356574, "8": 4000016840, "9": 3656349762, "10": 2026648505, "11": 1374177095, "12": 2066749866, "13": 444424459, "14": 419199985, "15": 2795493462, "16": 3214828758}, "total_size": 34712, "number_of_parts": 17}}"

update available
$aws/things/deviceTest/streams/testestream/description/json
$aws/things/deviceTest/streams/testestream/rejected/json
$aws/things/deviceTest/streams/testestream/data/json
$aws/things/deviceTest/streams/testestream/describe/json
$aws/things/deviceTest/streams/testestream/description/json

AWS topic $aws/things/deviceTest/streams/testestream/describe/json publish result.....: 0

data: +QMTRECV:
0,0,"$aws/things/deviceTest/streams/testestream/description/json","{"s":6,"d":"This stream is used for testing.","r":[{"f":1,"z":34712}]}"
contains correct topic
quote: {"s":6,"d":"This stream is used for testing.","r":[{"f":1,"z":34712}]}"

firmware size: 34712 bytes
stream ok
download firmware
$aws/things/deviceTest/streams/testestream/get/json
o: 0

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish result.....: 0
CRC Matches

```
CRC = 3067313637
result: 0
downloaded: 2048, size 34712, addr: 2560
o: 1

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
CRC = 2957856364
result: 0
downloaded: 4096, size 34712, addr: 4608
o: 2

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
CRC = 2597385937
result: 0
downloaded: 6144, size 34712, addr: 6656
o: 3

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
CRC = 1704859145
result: 0
downloaded: 8192, size 34712, addr: 8704
o: 4

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
CRC = 3506744434
result: 0
downloaded: 10240, size 34712, addr: 10752
o: 5
```

```
AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
CRC = 3736105166
result: 0
downloaded: 12288, size 34712, addr: 12800
o: 6

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
CRC = 3583040540
result: 0
downloaded: 14336, size 34712, addr: 14848
o: 7

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
CRC = 1057356574
result: 0
downloaded: 16384, size 34712, addr: 16896
o: 8

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
CRC = 4000016840
result: 0
downloaded: 18432, size 34712, addr: 18944
o: 9

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
CRC = 3656349762
result: 0
```

```
downloaded: 20480, size 34712, addr: 20992
o: 10

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
CRC = 2026648505
result: 0
downloaded: 22528, size 34712, addr: 23040
o: 11

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
CRC = 1374177095
result: 0
downloaded: 24576, size 34712, addr: 25088
o: 12

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
CRC = 2066749866
result: 0
downloaded: 26624, size 34712, addr: 27136
o: 13

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
CRC = 444424459
result: 0
downloaded: 28672, size 34712, addr: 29184
o: 14

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
```

```
CRC = 419199985
result: 0
downloaded: 30720, size 34712, addr: 31232
o: 15

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
CRC = 2795493462
result: 0
downloaded: 32768, size 34712, addr: 33280
Heap size: 6597 / 115976 bytes
o: 16

AWS topic $aws/things/deviceTest/streams/testestream/get/json publish
result.........: 0
CRC Matches
CRC = 3214828758
result: 0
downloaded: 34712, size 34712, addr: 35224
sd sync: 0

Done

returned. ret: 0
Update downloaded. Resetting system to apply
Hello World!
Size = 34712
Update Available to Install
Size of firmware: 34712
Erased sector with address 800C000
Erased sector with address 8010000
Erased sector with address 8020000
Erased sector with address 8040000
Erased sector with address 8060000
44 44 true 134267904 1024 1536
b7 b7 true 134268928 2048 2560
```

```
5d 5d true 134269952 3072 3584
ef ef true 134270976 4096 4608
fa fa true 134272000 5120 5632
65 65 true 134273024 6144 6656
9b 9b true 134274048 7168 7680
82 82 true 134275072 8192 8704
c1 c1 true 134276096 9216 9728
91 91 true 134277120 10240 10752
1e 1e true 134278144 11264 11776
54 54 true 134279168 12288 12800
c c true 134280192 13312 13824
31 31 true 134281216 14336 14848
f0 f0 true 134282240 15360 15872
aa aa true 134283264 16384 16896
9f 9f true 134284288 17408 17920
43 43 true 134285312 18432 18944
73 73 true 134286336 19456 19968
9c 9c true 134287360 20480 20992
fd fd true 134288384 21504 22016
b5 b5 true 134289408 22528 23040
82 82 true 134290432 23552 24064
1c 1c true 134291456 24576 25088
df df true 134292480 25600 26112
18 18 true 134293504 26624 27136
a0 a0 true 134294528 27648 28160
2f 2f true 134295552 28672 29184
2c 2c true 134296576 29696 30208
82 82 true 134297600 30720 31232
39 39 true 134298624 31744 32256
a8 a8 true 134299648 32768 33280
ca ca true 134300672 33792 34304
45 45 true 134301592 34712 35224
Update Installed
Hello, Mbed!
Hello, Mbed!
```

APÊNDICE B - CONFIGURAÇÃO MQTT STREAM AWS

O stream de arquivos por MQTT não é tão intuitivo de configurar como as outras funções do AWS utilizadas.

Após criar uma Função dentro de IAM Manager para uma Entidade: Serviço do AWS, Caso de Uso: IoT, conforme a Figura 14.

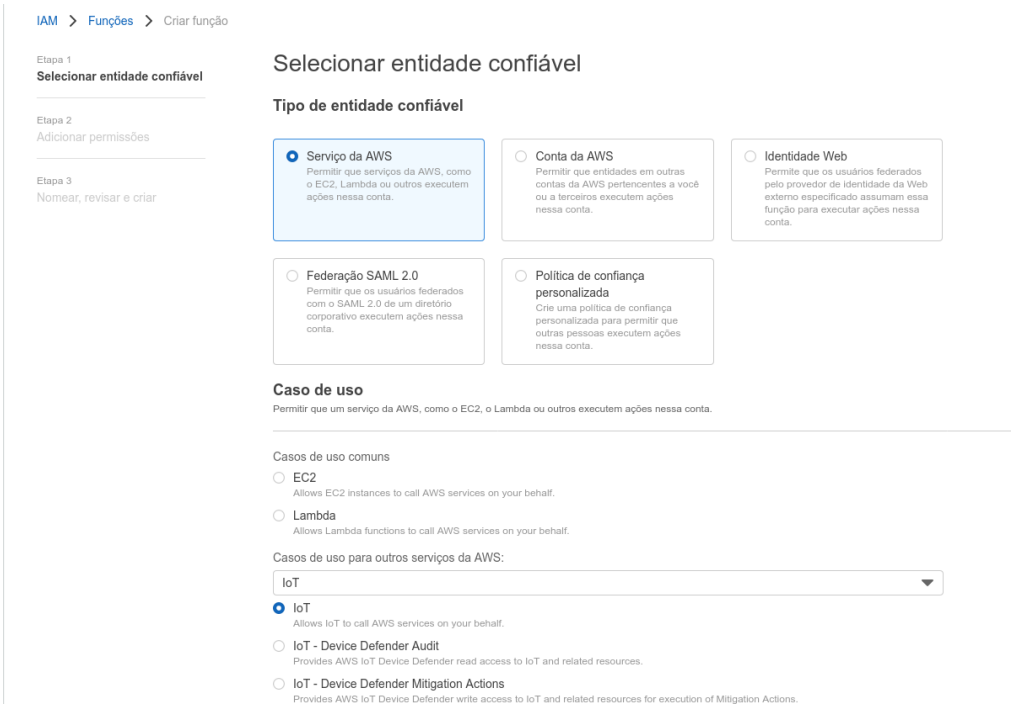

Figura 14 - Criando a Função

Fonte: O Autor

Após criar a Função é preciso editá-la, indo em “Adicionar Permissões” e “Criar Política em Linha”, conforme a Figura 15.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "iot:CreateStream",
                "iot:UpdateStream",
                "iot:DeleteStream",
                "s3:*",
                "iot:ListStreams",
                "iot:DescribeStream"
            ],
            "Resource": "*"
        }
    ]
}
```

Figura 15 - Política em Linha

Fonte: O Autor

Na Linha de Comando do AWS, crie um arquivo JSON conforme a Figura 16

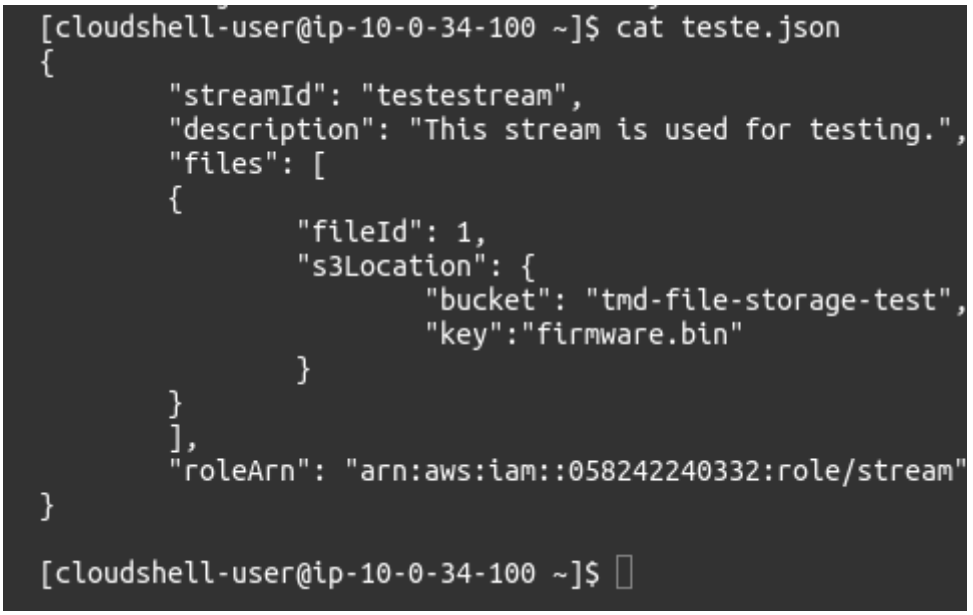

Figura 16 - Arquivo de configuração do stream

Fonte: O Autor

Com o comando “aws iot create-stream --cli-input-json file://teste.json” será criado o stream MQTT, permitindo que dispositivos façam download dos arquivos listados. O valor em “roleArn” é o código da Função criada anteriormente.

Toda vez que o arquivo é modificado dentro do Bucket do S3, é preciso atualizar o stream com o comando “aws iot update-stream --cli-input-json file://teste.json”