UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO

PROPOSTA DE PROJETO DE GRADUAÇÃO

Aluno: Matheus Hoffmann Fernandes Santos
hoffmann@poli.ufrj.br

Orientador: Miguel Elias Mitre Campista, D.Sc

1. TÍTULO

Implementação de uma plataforma de dados para sistemas de monitoramento remoto utilizando mensagens assíncronas.

2. ÊNFASE

Computação/Telecomunicações

3. TEMA

O estudo de tecnologias aplicadas a Internet das Coisas (*Internet of Things* - IoT) tem se tornado uma tendência mundial dada a gama de possibilidades que tais plataformas permitem: cidades inteligentes, residências inteligentes, telemedicina, cadeias logísticas inteligentes etc. Com tais aplicações do ecossistema IoT, utilizando transmissão de dados sem fio, existe uma grande demanda de estudo sobre o monitoramento dos dispositivos de sensoriamento, armazenamento e visualização dos dados bem como a sua disponibilização para terceiros. Portanto, o tema deste trabalho é a implementação de uma plataforma de dados, cobrindo todas as etapas de sistemas de sensoriamento remoto, sendo elas a aquisição, transmissão, persistência, entrega e segurança dos dados. Tal implementação terá como base o uso de tecnologias de código aberto, para que o estudo tenha a maior extensibilidade e reprodutibilidade possível.

4. DELIMITAÇÃO

Um conjunto de microsserviços será implementado, resultando em uma plataforma, que compreende o protocolo de transmissão, aquisição dos dados, o monitoramento e alarmes das condições dos dispositivos do sistema de sensoriamento,

persistência dos registros em bases de dados temporais, visualização em tempo real dos dados ingeridos em *dashboards* e disponibilização dos dados pela Internet de períodos posteriores ou em tempo real. Esses microsserviços serão disponibilizados, separadamente e virtualizados a nível do sistema operacional Unix, através de imagens docker [2], podendo então ser replicados em ambientes na nuvem. Por fim, esta plataforma será capaz de gerar eventos, por uma interface de controle, que levarão a reconfiguração dos dispositivos remotos. O caso de uso para plataforma de dados será o laboratório do Grupo de Teleinformática e Automação. No laboratório já existem diversos tipos de sensores conectados a rede sem fio. Tais sensores irão enviar dados para a plataforma, a partir da qual será possível monitorar e obter dados dos sensores. Ao final, uma comparação breve com outras plataformas com funcionalidades semelhantes será realizada.

5. JUSTIFICATIVA

A indústria de IoT vem tendo um crescimento vertiginoso com o passar dos anos, gerando até o termo IoE (*Internet of Everything*). A estimativa é que, até o final de 2020, mais de 20 bilhões de dispositivos conectados ao redor do mundo co-existam [1]. Dado esse cenário e a popularização das comunicações utilizando a tecnologia 5G, a estimativa é que esse número irá continuar a aumentar.

A arquitetura de aplicações IoT se baseia em dispositivos embarcados, contendo na maioria das vezes uma camada de percepção denominada sensor. Esses sensores podem aferir diversas variáveis de ambiente como luz, umidade, temperatura, calor, imagem, pressão, som, nível do mar etc. A partir disso, uma camada de controle, chamada controlador, é configurada para de tempos em tempos obter os dados dos sensores, resultando em um processo de amostragem. Dada a amostragem dos dados, o dispositivo embarcado pode enviar os dados e/ou armazená-los em memória. O caso ideal é que o sistema embarcado seja capaz de enviar os dados a um servidor, conforme forem amostrados, através de protocolos que compõem a camada de comunicação da arquitetura IoT. Esse requisito, porém, nem sempre pode ser atendido, dependendo dos requisitos da aplicação, da disponibilidade dos nós sensores e da presença de redes de comunicação. Estas últimas são usadas para transferência de dados entre os nós sensores e servidores centrais. A transferência

de dados, em todo caso, pode ser feita através de redes com fio ou sem fio e se dar de forma síncrona ou assíncrona. Enfim, os registros dos sensores chegam à camada de aplicação, que é composta por serviços que fazem uso dos dados para exibição, transformação e distribuição.

O protocolo MQTT (*Message Queuing Telemetry Transport*) [3] tem sido adotado como uma forma de comunicação padrão entre os dispositivos remotos e os servidores de armazenamento de dados em casos de aplicações IoT com mensagens assíncronas. Esse modo de comunicação, também conhecido como Pub/Sub – *Publicador-Subscritor* – se baseia na troca de mensagens, através de tópicos, no qual as mensagens são publicadas pelo cliente publicador e entregue a todos os clientes subscritores do mesmo tópico. Os tópicos são gerenciados em um nó intermediador entre as partes, chamado de *broker*. O MQTT possui vantagens no contexto de IoT em comparação a outros protocolos mais consolidados para comunicação remota, como o HTTP, por exemplo [4]. Isso se deve ao fato do protocolo priorizar o baixo uso de recursos de processamento e de rede das aplicações clientes – dispositivos embarcados – bem como oferecer garantia de entrega dos registros, mesmo considerando nós com limitações de energia sujeitos a elevadas perdas de transmissão, além de restrições de banda passante.

Uma desvantagem do protocolo MQTT é que ele atua somente na camada de comunicação entre os publicadores e os subscritos – cliente e servidor, respectivamente. O protocolo não específica padrões de armazenamentos de dados por parte do nó *broker*, ainda que as muitas implementações de *broker* MQTT implementem essa funcionalidade em versões com licença comercial [5]. Em sistemas distribuídos com alto volume de dados bem como alta disponibilidade de serviço, isto é, sistemas reativos [7], é usual que subscritores – aplicações que recebem os dados dos sensores remotos em tempo real – possam se desconectar temporariamente do *broker* por instabilidade de rede ou por atualizações de novas versões. Dado esse cenário, é importante que durante intervalos de desconexão a camada de recepção de dados possa armazenar os dados temporariamente para que não haja perda de informação quando os subscritores voltarem a receber dados. Uma solução viável para este tipo de problema é conectar o broker MQTT a alguma aplicação de *streaming* assistida de algum mecanismo de persistência de baixa durabilidade [8], como é o caso do

Apache Kafka.

O Apache Kafka [9] é, como uma plataforma de streaming distribuído, uma das principais escolhas de projetos de engenharia de dados que tenham como requisitos alta escalabilidade e tolerância a falhas. Ele possui um sistema de armazenamento configurado, para cada tópico, permitindo que mesmo quando consumidores percam a conexão, os mesmos possam consumir mensagens previamente enviadas por produtores, garantindo a entrega dos dados em semântica pelo menos uma vez. O Kafka também, quando configurado na forma de cluster, possui um sistema de replicação dos dados contidos na plataforma para, dessa maneira, proporcionar tolerância a falhas. Logo, mesmo que um nó da aplicação do Apache Kafka esteja indisponível, ainda assim a plataforma continua absorvendo e transmitindo dados entre aplicações.

Este projeto explora a combinação do protocolo de comunicação MQTT com o protocolo de streaming Kafka para que uma rede de dispositivos IoT possa conter milhares de dispositivos e ter dezenas de aplicações conectadas consumindo e produzindo dados dependentes dos mesmo dispositivos. Muitos projetos usam as duas tecnologias combinadas e essa demanda atualmente já é atendida pelo mercado. O maior distribuidor de *brokers* MQTT, EMQ X [5], possui um plugin de conexão direta com o Apache Kafka [10] e a maior empresa de consultoria da plataforma Kafka, a Confluent, já possui uma ferramenta chamada MQTT *Proxy* [11] em que é possível enviar e receber mensagens diretamente ao Kafka utilizando o protocolo MQTT. A limitação é que ambas as soluções só estão disponíveis em versões pagas das aplicações.

Outro aspecto das atuais plataformas de dados, é que apesar do Apache Kafka ter a capacidade de coordenar a troca de mensagens assíncronas entre aplicações e serviços, ele não é uma ferramenta própria para persistência dos dados. Portanto, conectores são utilizados para portar os dados para outros mecanismos de persistência, como por exemplo bases de dados temporais, para que seja possível visualizar os dados a partir de *dashboards*. A persistência em arquivos também tem sua utilidade, uma vez que é possível de maneira simples fornecer os dados pré-processados ou anonimizados para terceiros.

A Figura 1 representa o ecossistema de microsserviços que compõem a pla-

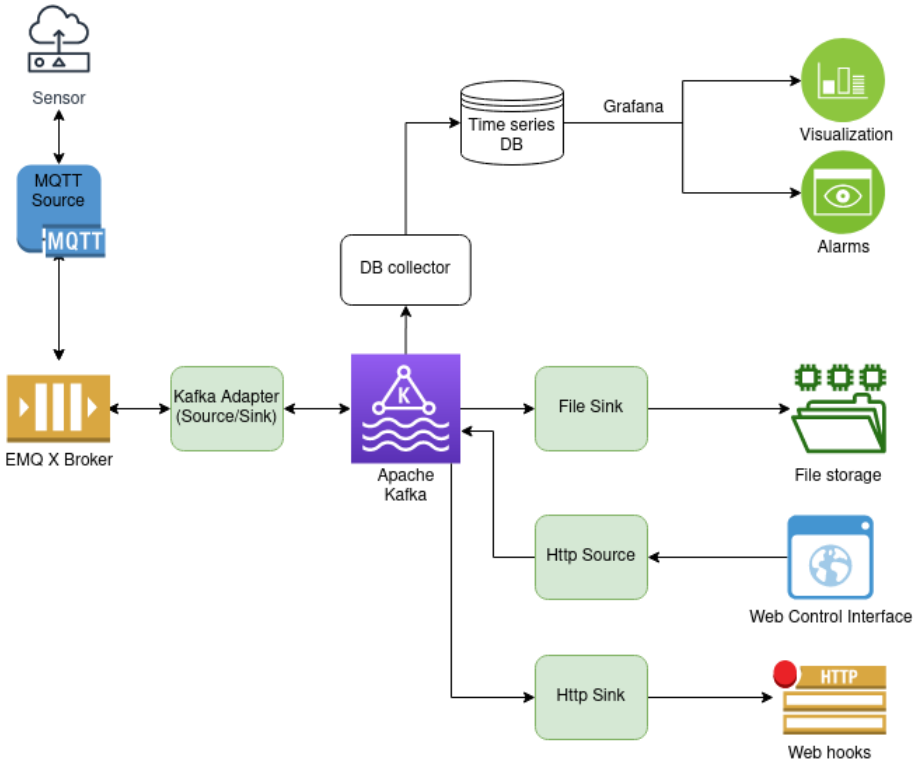

Figura 1: Arquitetura da plataforma de dados.

taforma de dados. Todas as aplicações são projetadas em formato de *streams*. Todo componente dentro de um streaming de dados é classificado como *source*, *flow* ou *sink*. *Source* são as etapas de ingestão de informação, como eventos provindos de uma interface ou uma fila – como um nó subscritor. *Flows* são etapas de manipulação ou enriquecimento de dados. *Sink* é a entrega final dos dados, que podem ser endereçados a mecanismos de persistência, servidores HTTP ou filas – atuando como um nó publicador. A ideia é que o *Kafka Broker* realize todo o pipeline de dados entre os serviços.

6. OBJETIVO

O objetivo geral do projeto é, então, desenvolver uma plataforma de dados para sistemas de monitoramento remoto, utilizando tecnologias de código aberto, detalhando o porquê de tais escolhas a fim de prover uma plataforma confiável no que diz respeito a entrega e armazenamento de dados, mesmo sob alto volume. Ao final, tem-se uma plataforma, modularizada através de containers docker [2], onde é possível obter dados do sistema de sensoriamento remoto pela Internet de diversas formas: visualização gráfica, download de arquivos, recebimento de alarmes e

obtenção em tempo real via protocolo *Restful* HTTP. Também será provido a infraestrutura necessária para atuação direta nos dispositivos remotos, podendo envolver cálculos complexos, como algoritmos de Aprendizado de Máquina.

7. METODOLOGIA

Cada serviço da plataforma seguirá o ciclo padrão de desenvolvimento de software, compreendendo a construção dos requisitos, projeto da arquitetura, implementação, testes e entrega. A entrega final dos serviços utilizará o método de virtualização de ambientes, containerização – Docker, para que torne a plataforma facilmente reprodutível em diversos ambientes, inclusive na nuvem.

Alguns componentes dos módulos já estão disponibilizados como ferramentas de código aberto, prontas para uso. Logo, não será necessário testes unitários dessas ferramentas, como por exemplo, do protocolo MQTT, plataforma de mensagens Apache Kafka e a base de dados temporal não relacional para métricas. Os testes relativos a essas ferramentas serão apenas de integração. Quanto ao dashboard de visualização, usando o Grafana [12], será validado em acordo com os membros do laboratório do Grupo de Teleinformática e Automação (GTA).

Os componentes restantes, que serão desenvolvidos de forma integral, incluindo a integração entre os componentes já disponibilizados e os serviços de persistência em arquivos, streaming e controle, serão desenvolvidos utilizando a linguagem Scala, versão 2.13. Todos os códigos, incluindo os de automação de infraestrutura, serão disponibilizados em um repositório público de versionamento na plataforma GitHub [13]. Os resultados do protótipo serão divulgados, em forma de artigo.

8. CRONOGRAMA

Etapa	**Prazo**
Revisão bibliográfica e definição prévia dos capítulos	28/09/2020
Implementação do módulo de aquisição de dados	19/10/2020
Implementação do módulo de visualização em tempo real e alarmes	23/11/2020
Implementação do módulo de arquivos e streaming	21/12/2020
Implementação do módulo de controle	18/01/2021
Revisão da arquitetura e Início dos testes no laboratório	01/02/2021
Revisão ortográfica e Entrega Final	01/03/2021
Defesa	A partir de 15/03/2021

Tabela 1: Cronograma do Projeto de Graduação.

Referências Bibliográficas

[1] NGUYEN, Bang; SIMKIN, Lyndon. The Internet of Things (IoT) and marketing: the state of play, future trends and the implications for marketing. 2017.

[2] MERKEL, Dirk. Docker: lightweight linux containers for consistent development and deployment. Linux journal, v. 2014, n. 239, p. 2, 2014.

[3] HUNKELER, Urs; TRUONG, Hong Linh; STANFORD-CLARK, Andy. MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks. In: 2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08). IEEE, 2008. p. 791-798.

[4] NAIK, Nitin. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In: 2017 IEEE international systems engineering symposium (ISSE). IEEE, 2017. p. 1-7.

[5] https://www.emqx.io/products/pricing

[6] KOZIOLEK, Heiko; GRÜNER, Sten; RÜCKERT, Julius. A Comparison of MQTT Brokers for Distributed IoT Edge Computing. In: European Conference on Software Architecture. Springer, Cham, 2020. p. 352-368.

[7] https://www.reactivemanifesto.org/pt-BR

[8] WEN, Y. A. N. G. et al. Industrial big data platform based on open source software. In: International Conference on Computer Networks and Communication Technology (CNCT 2016). Atlantis Press, 2016.

[9] WANG, Guozhang et al. Building a replicated logging system with Apache Kafka. Proceedings of the VLDB Endowment, v. 8, n. 12, p. 1654-1655, 2015.

[10] https://docs.emqx.io/tutorial/v3/en/bridge/bridge_to_kafka.html

[11] https://docs.confluent.io/current/kafka-mqtt/index.html

[12] https://grafana.com/

[13] https://github.com/h0ffmann/bigiot

Rio de Janeiro, 9 de outubro de 2020.

Matheus Hoffmann Fernandes Santos - Aluno

Miguel Elias Mitre Campista - Orientador