Uma API para armazenamento distribuído de *databooks*.

**Guilherme S. de Oliveira, Renan Tarouco da Fonseca,
Eder M. N. Gonçalves, Eduardo N. Borges**

[1]Centro de Ciências Computacionais, Universidade Federal do Rio Grande
Av. Itália, km 8, 96203-900, Rio Grande - RS

guilhermesi@furg.br, renantarouco@gmail.com

Abstract. *This paper presents a tool that implements a service to insert, search and list databooks in a distributed storage environment, making it easier to access the files and their assigned metadata. We developed an API, packaged as a container, with routes capable of providing these features in a document management system for the extractive industry.*

Resumo. *Este artigo apresenta uma ferramenta que implementa um serviço de inserção, busca e listagem de databooks em um ambiente de armazenamento distribuído, facilitando o acesso aos arquivos e seus metadados atribuídos. Foi desenvolvida uma API, empacotada como um contêiner, com rotas capazes de prover essas funcionalidades num sistema de gerenciamento de documentos da indústria extrativista.*

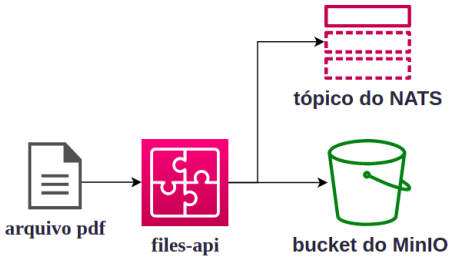
1. Introdução

Na indústria de produção e suprimento de óleo e gás, o Controle de Qualidade trata dos mecanismos e padronizações de procedimentos, processos e recursos destes ambientes. A inspeção é uma etapa crítica desta indústria. Dados relacionados à inspeção de equipamentos e estruturas estão contidos em documentos denominados *databooks*. Eles reúnem toda informação gerada e registrada, incluindo documentos de engenharia e suas revisões, ordens de compra, ensaios, certificados de qualidade, relatórios de inspeção, entre outros, e podem conter milhares de páginas.

A análise destes *databooks* é feita por inspetores especializados que devem buscar anormalidades tais como a falta de relatórios, documento sem assinatura, ou mesmo atos de má fé. Essa análise consome muito tempo e recursos humanos. Com o objetivo principal de desenvolver um sistema para gerenciar e analisar *databooks* digitais ou digitalizados, surgiu a necessidade de armazená-los em um ambiente seguro, com alta disponibilidade e escalável. Neste contexto, este artigo apresenta o desenvolvimento de uma *Application Programming Interface* (API) RESTful [Jacobson et al. 2012], que implementa um serviço Web de *upload*, *download* e listagem de arquivos.

2. Metodologia

A Figura 1 (à esquerda) apresenta a arquitetura da API desenvolvida, denominada *files-api*. A interface do usuário ou um componente interno do sistema de gerenciamento de *databooks* requisita o armazenamento de um documento PDF. A *files-api* persiste o documento em *bucket* lógico de um sistema de armazenamento de objetos distribuído.




```
type Metadata struct {
    Size            int64
    FileName        string
    Target          string
    ProjectName     string
    CreatedAt       time.Time
    UserID          string
    CustomMetadata  string
}
```

Figura 1. Arquitetura da API desenvolvida (à esquerda) e estrutura de metadados para *upload* de arquivos (à direita).

Um serviço de mensageria é utilizado para notificar outros componentes do sistema que existe um novo documento para pré-processamento, reconhecimento ótico de caracteres, extração de informação, carga em bancos de dados, entre outros.

MinIO [1] é um sistema de armazenamento de objetos de alto desempenho, desenvolvido especialmente para a construção de aplicativos e serviços nativos da nuvem. Esse sistema implementa um modelo de dados NoSQL baseado em chave e valor [Pokorny 2013]. Cada objeto digital é identificado de forma única pela sua chave, gerada automaticamente pela *files-api*, e seu valor é o próprio binário que representa o objeto. A escolha do MinIO se justifica pelas seguintes características: *open-source*; praticidade na operação, visto que possui API compatível com o Amazon S3; vasta documentação; e capacidade de associar metadados descritivos aos objetos.

NATS *Messaging* [2] [Sharvari and Sowmya 2019] é um serviço de mensageria *Open Source* que implementa um modelo *pub/sub*, que é uma forma de comunicação assíncrona muito utilizada em sistemas distribuídos baseados em arquiteturas de microserviços. Editores (*pub*) publicam mensagens em tópicos e os ouvintes inscritos nestes tópicos (*sub*) recebem essas mensagens. A escolha do NATS foi baseada na popularidade e facilidade de implantação.

A *files-api* foi desenvolvida na linguagem de programação Go[3]. Foi utilizada a ferramenta Swagger[4] para documentação da API, aproveitando da sua capacidade de gerar um arquivo baseado em JSON/YAML documentando as rotas, suas entradas e saídas.

De modo a compilar a aplicação e disponibilizar para uso no sistema, utilizou-se o Docker[5]. Dessa forma, foi criada uma imagem da aplicação e alocada em um contêiner através de um arquivo YAML denominado Dockerfile, contendo os comandos para a criação do mesmo. Segundo [Cito et al. 2017], Docker é um serviço de sistemas de contêineres cujo objetivo é melhorar a reprodutibilidade de aplicativos. O Docker possibilita criar, testar e implementar aplicações em um ambiente de contêiner, e também o empacotamento de uma aplicação ou ambiente em um contêiner, se tornando portátil para qualquer outro *host* que contenha o Docker instalado.

[1] https://min.io
[2] https://nats.io
[3] https://golang.org
[4] https://swagger.io
[5] https://www.docker.com

3. Resultados

A *files-api* foi desenvolvida com a criação de algumas rotas principais: *upload*, *download* e listagem de arquivos. Visando a utilização restrita do sistema principal. Para solicitar o envio de um documento PDF para análise, a rota de *upload* **./files/v1/analysis** deve ser acionada. O arquivo é carregado para o *bucket* de destino no MinIO. Além do próprio objeto, alguns metadados são adicionados ao serviço de armazenamento. A Figura 1 (à direita) apresenta um exemplo de esquema de metadados contendo informações como tamanho e nome do arquivo, análise alvo do documento baseada no tipo de inspeção (*Target*), nome do projeto de engenharia relacionado, a data em que foi inserido e que usuário do sistema fez essa operação. Ao realizar a operação de *upload*, uma pequena parte desses metadados são definidos pelo usuário, tais como *Target* e o nome do projeto. Ainda é possível especificar novos metadados sob demanda utilizando o *CustomMetadata*. O restante dos metadados ficam sob responsabilidade da API.

Após o armazenamento do PDF no MinIO, uma mensagem é publicada no serviço de mensageria NATS, diretamente no tópico referente a inserção de um novo *databook* no sistema, informando aos demais microserviços que há um novo documento para análise. Neste momento, funções *serverless* disparam outras funcionalidades dos *pipelines* de processamento.

A rota de *download* dos arquivos **./files/v1/download** é utilizada para recuperar do MinIO *databooks* já analisados. Assim, apenas utilizando o modelo de busca chave-valor, o arquivo é encontrado em um dos nós da nuvem e é disponibilizado para que o usuário faça o *download* em seu computador. Essa rota possui alguns parâmetros *Query*, como o ID do arquivo e o nome do *bucket* lógico em que está armazenado.

Ademais, a listagem de arquivos é utilizada para relacionar todos os documentos de um determinado *bucket*, podendo filtrá-los pelo *Target* e/ou pelo usuário que fez a inserção do arquivo no sistema. Existem duas rotas específicas, uma para usuários administrativos, os quais têm acesso a todos os documentos do sistema, e outra para os demais usuários com permissões restritas aos próprios documentos. Dessa maneira, a informação de qual usuário está solicitando a listagem é recebida pelo *Header*. Também é informado o *bucket* desejado para listagem de arquivos via parâmetros *Query*. Opcionalmente, também é possível filtrar os arquivos por *Target*. Na Figura 2 pode ser observado um exemplo de listagem de arquivos do *bucket* “pdf-analisados” caracterizado pelo tipo de inspeção (*Target*) “norma-325”.

4. Conclusão

Nesse artigo foi apresentado o desenvolvimento de uma API para inserção, busca e listagem de documentos em um ambiente de armazenamento distribuído que compõe um sistema de gerenciamento de *databooks* para a indústria extrativista. Foram criadas rotas específicas que implementam essas funcionalidades. Foram utilizadas ferramentas e serviços escaláveis com alto desempenho, como o MinIO para armazenamento distribuído, e o NATS para comunicação entre todos os microserviços que integram o sistema.

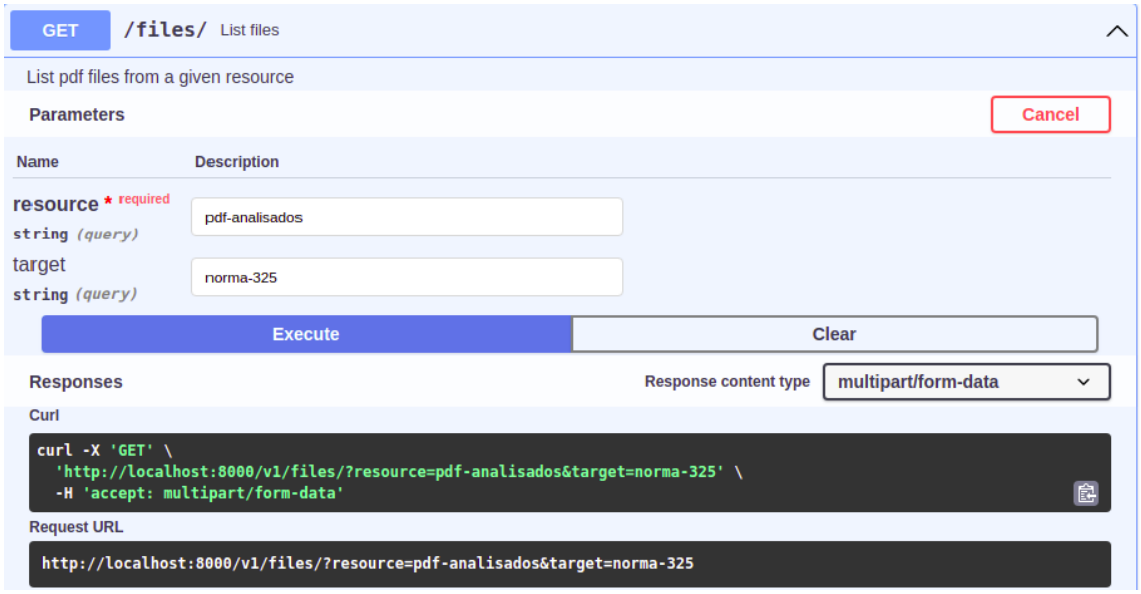


Figura 2. Estrutura da rota de listagem de arquivos.

Dessa forma, a *files-api* possibilitou controlar e gerenciar todos os documentos carregados no sistema, não apenas submetidos pela interface gráfica, mas também por qualquer módulo interno do sistema. Os testes funcionais foram validados por funcionários da empresa de extração. Assim, todos os subprodutos de processos e funcionalidades podem facilmente ser persistidos e recuperados. Outro diferencial é a possibilidade de cada cliente da API estabelecer um conjunto de metadados específico, que atenda os seus requisitos internos, através do *CustomMetadata*.

Entre os trabalhos futuros, destaca-se a criação de um rota que proporciona agrupar determinadas páginas de um *databook* completo, agregando partes de documentos em um novo arquivo PDF e disponibilizá-lo para *download*. Esta funcionalidade permitirá segmentar documentos de acordo com os requisitos de determinadas aplicações.

Referências

Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S., and Gall, H. C. (2017). An empirical analysis of the docker container ecosystem on github. In *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 323–333.

Jacobson, D., Brail, G., and Woods, D. (2012). *APIs: A Strategy Guide*. Oreilly and Associate Series. O'Reilly Media.

Pokorny, J. (2013). Nosql databases: a step to database scalability in web environment. *International Journal of Web Information Systems*, 9(1):69–82.

Sharvari, T. and Sowmya, N. K. (2019). A study on modern messaging systems-kafka, rabbitmq and nats streaming. arXiv:1912.03715 [cs.DC].