Arquitetura de Solução para Indexação de Grandes Volumes de Áudio e Vídeo Usando Reconhecimento de Fala.

Michel Vieira Batista

Março / 2021

Dissertação de Mestrado em Ciência da Computação

Arquitetura de Solução para Indexação de Grandes Volumes de Áudio e Vídeo Usando Reconhecimento de Fala

Esse documento corresponde a Dissertação de Mestrado apresentado à Banca Examinadora para defesa no curso de Mestrado em Ciência da Computação do Centro Universitário Campo Limpo Paulista - UNIFACCAMP.

Campo Limpo Paulista, 11 de Março de 2021.

Michel Vieira Batista

Prof. Dra. Marta Ines Velazco Fontova (Orientadora)
Prof. Dr. Eduardo Javier Huerta Yero (Co-Orientador)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Ficha catalográfica elaborada pela
Biblioteca Central da Unifaccamp

B337a

Batista, Michel Vieira

Arquitetura de solução para indexação de grandes volumes de áudio e vídeo usando reconhecimento de fala / Michel Vieira Batista. Campo Limpo Paulista, SP: Unifaccamp, 2021.

Orientadora: Profª. Drª. Marta Inés Velazco Fontova
Coorientador: Prof°. Dr. Eduardo Javier Huerta Yero

Dissertação (Programa de Mestrado Profissional em Ciência da Computação) – Centro Universitário Campo Limpo Paulista – Unifaccamp.

1. Reconhecimento de fala. 2. Indexação de áudio e vídeo. 3. Busca de palavras faladas em áudio. 4. Busca de palavras faladas em vídeo. 5. *Big Data*. I. Fontova, Marta Inés Velazco. II. Yero, Eduardo Javier Huerta. III. Centro Universitário Campo Limpo Paulista. IV. Título.

CDD- 005.1

DEDICATÓRIA

Dedico este trabalho aos meus queridos pais Selma Maria Vieira da Costa Batista e Oliveira Ramon Batista, e a minha amada futura esposa Tainá Neves Gomes que além do grande incentivo e apoio incondicional, contribuíram diretamente e indiretamente para os resultados deste trabalho.

AGRADECIMENTOS

A DEUS, que sempre esteve comigo em todos os momentos bons e ruins da minha vida.

Aos meus queridos pais Selma Maria Vieira da Costa Batista e Oliveira Ramon Batista que sempre fizeram de tudo para que eu pudesse ter condições de ter uma boa educação e sempre buscar em buscar em ser alguém melhor e sempre fazer o bem.

A minha futura esposa que eu amo Tainá Neves Gomes que sempre me apoiou, me incentivou e teve paciência durante esse árduo processo de pesquisa e desenvolvimento.

Ao Professor Dr. Eduardo Javier Huerta Yero, meu professor e orientador, que me apoiou e orientou para a realização deste projeto. Muito obrigado pela confiança, paciência, dedicação e conhecimento compartilhado durante este período.

A Professora. Dra. Marta Ines Velazco Fontova, professora que assumiu a orientação da dissertação após o processo de qualificação. Muito obrigado pela dedicação, colaboração e paciência na conclusão deste projeto.

Agradeço a todos da UNIFACCAMP, em especial o Professor Dr. Osvaldo Luiz de Oliveira pelo incentivo, pelas aulas e pela estrutura da faculdade e apoio.

A CAPES pelo apoio financeiro realizado durante o desenvolvimento deste trabalho.

Aos meus amigos, familiares, colegas de pesquisa e estudos que estiveram comigo durante esse processo.

“Saber não é o bastante, é preciso aplicar. Querer não é o bastante, é preciso fazer. ”

Bruce Lee

Resumo. A quantidade de dados digitais existentes vem crescendo em escala exponencial. Grande parte destes dados está em formatos não estruturados de áudio e vídeo. Eles são gerados por diversas fontes de dados, como smartphones, câmeras digitais, YouTube, Facebook, Instagram ou reportagens radiais e televisivas. Com este volume de dados, a tarefa de processar, indexar e pesquisar estes dados é uma tarefa difícil e desafiadora. Grande parte dos sistemas atualmente utilizam metadados atrelados no arquivo original de áudio/vídeo que descrevem suas principais características. Apenas estes metadados são indexados e pesquisados, enquanto os conteúdos internos dos arquivos são ignorados. Entretanto, existem várias dificuldades para extrair o conteúdo interno dos arquivos, como a heterogeneidade dos arquivos, dificuldade de reconhecimento de fala e indexação desses conteúdos para realização de pesquisas. A dissertação apresenta e descreve uma arquitetura de solução para o problema proposto. Além disso, um protótipo foi construído com o objetivo de testar e provar a arquitetura. Os testes foram realizados em um grande conjunto de dados e os resultados obtidos demonstram a capacidade e a eficácia da arquitetura em processar e indexar os dados.

Palavras-chave: Reconhecimento de Fala; Indexação de Áudio e Vídeo; Busca de Palavras Faladas em Áudio; Busca de Palavras Faladas em Vídeo; Big Data.

Abstract. The amount of existing digital data has been growing on an exponential scale. The most of this data is in unstructured audio and video formats. These data are generated by several data sources such as smartphones, digital cameras, YouTube, Facebook, Instagram, radio and television reports. With this data volume, the task of processing, indexing and searching these data is a difficult and challenging task. Most systems currently use metadata linked to the original audio/video file that describe its main characteristics. Only this metadata is indexed and searched, while the internal contents of the files are ignored. However, there are several difficulties in extracting the internal content of files, such as the files heterogeneity, difficult in speech recognition and indexing these contents to carry out research. The dissertation presents and describes a solution architecture for the proposed problem. In addition a prototype was built in order to test and proof the solution architecture. The tests were performed on a large data set and the results obtained demonstrate the architecture's capacity and effectiveness in processing and indexing the data.

Keywords: Speech Recognition; Audio and Video Indexing; Search for Spoken Words in Audio; Search for Spoken Words in Video; Big Data.

Sumário

Glossário:

CMU Sphinx – Software de reconhecimento de fala desenvolvido pela universidade Carnegie Mellon

IDC – *International Data Corporation*

IDE - Ambiente de Desenvolvimento Integrado

GitHub – Plataforma de hospedagem de código fonte

JAR - É um arquivo compactado usado para distribuir um conjunto de classes Java, um aplicativo Java, ou outros itens como imagens, XMLs, entre outros

JSON - *JavaScript Object Notation*

Librivox – Audiolivros de domínio público gratuitos

MAVIS – Plataforma para processar áudio e vídeo e extrair o conteúdo falado da Microsoft

Metadados - Dados e informações adicionais sobre outros dados

Open Source – Projetos de criação de software de código aberto

Podcast – Nome dado ao arquivo de áudio digital, frequentemente em formato MP3 ou AAC (este último pode conter imagens estáticas e links), publicado através de *podcasting* na internet e atualizado via RSS. Também pode se referir a série de episódios de algum programa quanto à forma em que este é distribuído.

SQL - *Structured Query Language*

Stop Words/Palavras de Parada - Palavras que são consideradas não relevantes para uma consulta de em um texto.

Stories - Fotos e vídeos rápidos que são publicados no Instagram, que ficam disponíveis por 24 horas

Streams - É uma sequência de elementos de dados disponibilizados ao longo do tempo

Streaming - É uma forma de distribuição digital é transmitida online

Lista de Tabelas

Lista de Figuras

1. Introdução

A quantidade de dados digitais gerados nos últimos anos vem crescendo em escala exponencial. Um relatório emitido pela IBM em 2013 (IBM 2013) estima que 2.5 exabytes de dados sejam gerados por dia. Outro relatório emitido pela *International Data Corporation* (IDC) em 2012 (International Data Corporation 2012) estima que a quantidade de dados existente em 2012 era de 2.7 zettabytes. Recentemente a IDC (International Data Corporation 2018) reportou que a quantidade de dados existentes subiu de 2.7 zettabytes para 33 zettabytes em 2018, o que representa um crescimento de aproximadamente 12 vezes na quantidade de dados em relação ao ano de 2012. O mesmo relatório classificou os três principais locais onde estes conteúdos são gerados:

- Core, que são *data centers* tradicionais e em servidores em nuvem;
- Edge, que são infraestruturas locais de empresas;
- Endpoints, que são PCs, smartphones e dispositivos de IoT.

O relatório da IDC também prevê que os dados continuem a crescer e que em 2025 cheguem a 175 zettabytes. A Figura 1 apresenta o crescimento anual estimado de dados até o ano de 2025.

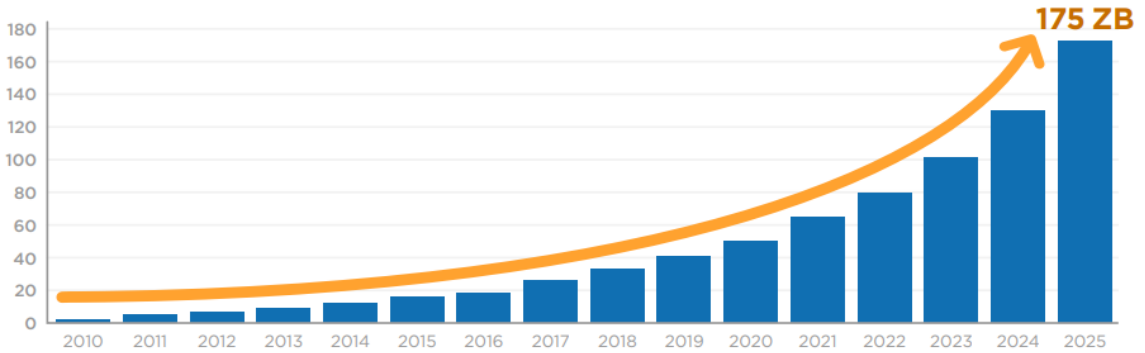

Figura 1. Crescimento anual de dados de 2010 a 2025, extraído do IDC - Date Age 2025 (2018 p.6)

Portanto, com essa grande quantidade de dados uma nova área surge chamada Big Data, termo usado para referir-se a um grande volume de dados com estruturas diferentes e que cresce a ritmo acelerado. Estes dados podem ser de várias fontes e formatos heterogêneos (Oussous, *et al*. 2017). Os dados disponíveis podem ser divididos em três categorias (Gandomi e Haider 2014):

- **Estruturados:** dados tabulares que são encontrados em planilhas ou banco de dados relacionais;
- **Semiestruturados:** uma forma de dados estruturados cuja organização não obedece às estruturas formais dos modelos baseados em tabelas. São usualmente baseados em marcadores utilizados para separar elementos semânticos e formar hierarquias de campos no documento. O exemplo mais conhecido é o *eXtensible Markup Language* (XML);
- **Não estruturados:** textos, imagens, áudio, vídeo e outros tipos de dados que não apresentam uma estrutura definida.

O foco principal desta pesquisa são os tipos dados não estruturados, pois é nesse tipo que os conteúdos no formato de áudio e vídeo estão classificados.

Dentre as fontes de dados disponíveis destacam-se os smartphones, câmeras digitais, YouTube, Facebook, Instagram, *podcasts*, imagens produzidas por satélites, conversas telefônicas, reportagens radiais e televisivas, entre outras formas. A Figura 2 apresenta um estudo feito pela companhia DOMO em 2020 (DOMO 2020) que mostra a variedade e a quantidade de dados que foram gerados ou consumidos por minuto em 2020 por várias empresas.

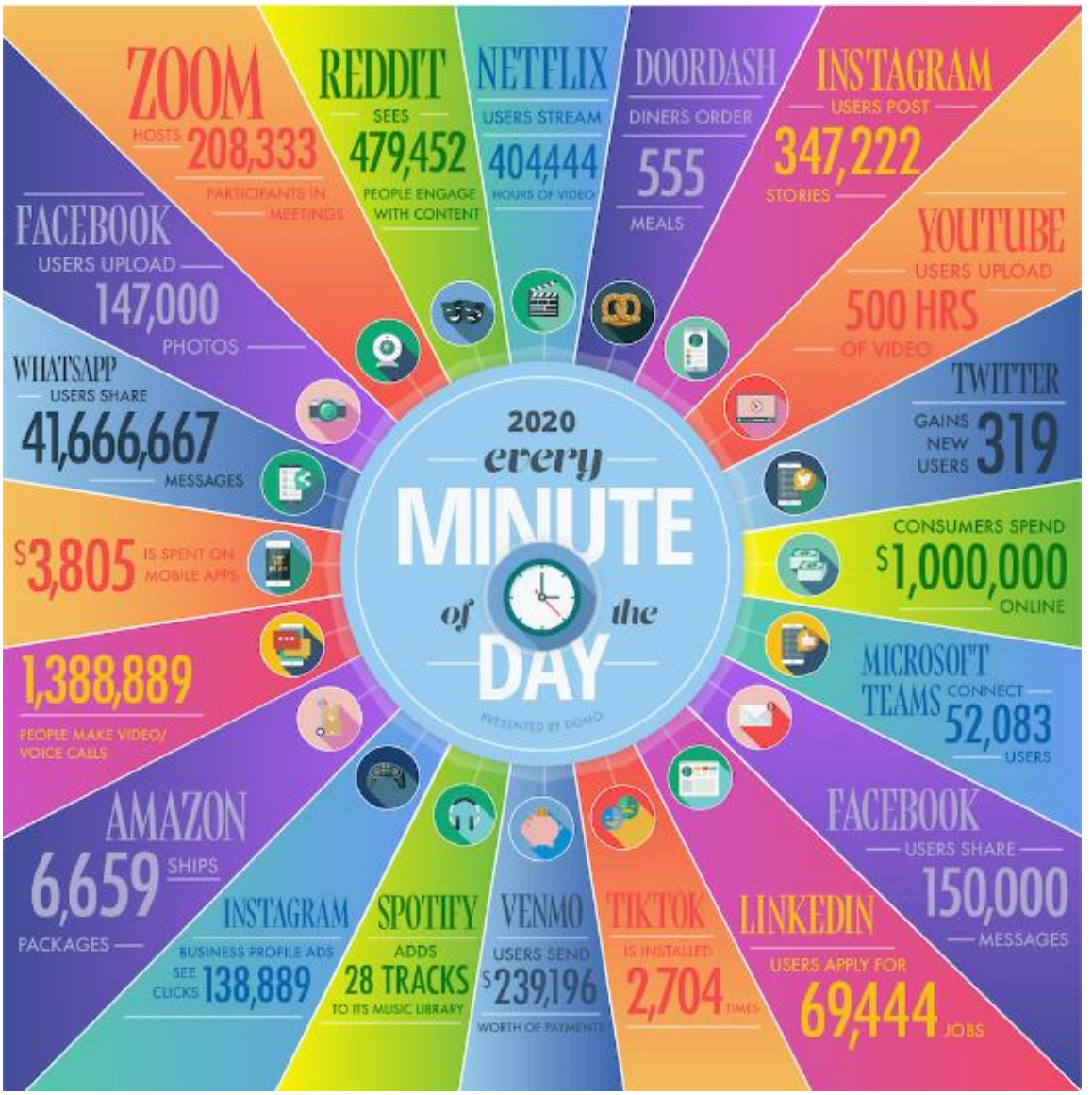

Figura 2. Quantidade de dados que são gerados por minuto em 2020. Cortesia de Domo, Inc. (Data Never Sleeps 8 p 1).

Uma parte significativa dos dados gerados diariamente está no formato de áudio e vídeo. A Figura 2 apresenta alguns exemplos: 500 horas de vídeos por minuto sendo enviado para o YouTube, 28 músicas por minuto adicionadas no Spotify, 404.444 horas de vídeos sendo assistidos no Netflix ou 347.222 *stories* sendo postados no Instagram, entre outros.

Apesar da ampla disponibilidade de dados em formato de áudio e vídeo, a tarefa de processar, indexar e pesquisar estes dados ainda se mostra desafiadora. A maior parte dos sistemas existentes hoje utilizam metadados atrelados ao arquivo original de áudio/vídeo em que se descrevem suas principais características. Apenas estes

metadados podem ser indexados e pesquisados, enquanto o conteúdo do arquivo é ignorado.

Em particular, uma solução que use reconhecimento de fala para detectar as palavras utilizadas no áudio do arquivo e indexe o arquivo baseado nas palavras detectadas seria de especial interesse. Há, no entanto, vários desafios para uma solução deste tipo:

- **Heterogeneidade:** Os arquivos estão disponíveis em vários formatos de áudio e vídeo como, por exemplo, WMA, MP4, WAV, AVI, 3GP, entre outros;
- **Reconhecimento de fala:** Neste processo, existem desafios como o ruído ambiente onde o som foi gravado, distorção do canal de gravação, sotaque do interlocutor, pronúncias diferentes ou até erradas de determinadas palavras por parte do interlocutor, uso de gírias, neologismos e expressões regionais (Saon e Chien 2012);
- **Indexação:** O mecanismo de indexação de resultados deve permitir acessar o trecho do arquivo de áudio e/ou vídeo onde aparecem as palavras pesquisadas, ou, pelo menos, armazenar estas informações e exibi-las ao usuário para facilitar a localização. Além disso, a indexação deve otimizar a busca pelo conteúdo textual.

Como explicado anteriormente, a maioria dos sistemas atuais utilizam metadados para realizar uma análise ou pesquisa em áudio e vídeo. Nestes sistemas, o usuário fica responsável por associar esses metadados aos arquivos. A título de exemplo, um arquivo de vídeo representando um filme teria: nome do filme, diretor, ano de publicação e descrição, entre outros. Desta forma, é possível pesquisar por filmes de determinados autores ou ano, mas não pesquisar por uma palavra ou frase falada dentro do filme.

As técnicas de reconhecimento e análise de fala permitem extrair o conteúdo falado em arquivos de áudio e vídeo e indexá-lo de forma a permitir pesquisas textuais sobre um determinado assunto, tal como é feito na pesquisa de documentos na Web.

A solução apresentada consiste em uma arquitetura de propósito geral que descreve os principais componentes e funcionalidades necessárias. Como prova de conceito, este trabalho descreve também a implementação de um protótipo, baseado na arquitetura proposta, que serve para demonstrar a viabilidade da solução.

Esta dissertação está estruturada da seguinte forma. **O Capítulo 2** apresenta a revisão bibliográfica e detalha projetos que são similares ao proposto nesta dissertação. O **Capítulo 3** descreve a arquitetura criada para solucionar o problema apresentado. **O Capítulo 4** fornece detalhes da implementação do protótipo. **O Capítulo 5** apresenta e expõe os resultados coletados durante os testes. **O Capítulo 6** descreve as conclusões sobre o projeto. **O Capítulo 7** contém detalhes futuros sobre o projeto. **O Capítulo 8** contém as referências bibliográficas utilizadas na dissertação. **O Apêndice I** apresenta o script para realizar a configuração do Apache Kafka. **O Apêndice II** apresenta o script para realizar a configuração do Elasticsearch. **O Apêndice III** contém detalhes técnicos de como configurar a base de dados no Kibana. **O Apêndice IV** mostra como criar alguns gráficos no Kibana para apresentar os dados processados.

2. Revisão Bibliográfica

Este capítulo apresenta o estado da arte em temas relacionados a esta dissertação. A pesquisa abrangeu publicações nas seguintes bases de dados: Google Scholar, ERIC, ResearchGate, IEEE, ACM e também pesquisas feitas diretamente no Google para procurar softwares proprietários cujos objetivos sejam similares ao proposto neste trabalho. Algumas das palavras chaves utilizadas foram "*Speech Recognition*", "*Search in Audio*", "*Audio indexing and search*", "*Audio and video processing*" e "*Audio transcriber*". Dentre os resultados obtidos demos especial atenção a pesquisas e softwares criados a partir de 2010.

Os resultados obtidos na pesquisa foram avaliados de acordo com os critérios listados a seguir.

1. **Formato dos dados:** Tipos de formato de áudio e vídeo suportados, por exemplo, WAV, MP3, MP4, AVI entre outros;
2. **Fontes de dados:** Plataforma onde os dados são originalmente armazenados antes de serem processados;
3. **Suporte a grande quantidade de dados:** Capacidade do sistema de processar uma grande quantidade de arquivos de áudio ou vídeo;
4. **Escalabilidade:** Capacidade do sistema de aumentar de tamanho para suportar cargas maiores sem que o tempo de resposta seja afetado;
5. **Uso de software de reconhecimento de fala:** O trabalho obrigatoriamente deve fazer o uso de alguma ferramenta de reconhecimento de fala para realizar a extração do conteúdo falado nos arquivos de áudio ou vídeo;
6. **Indexação e pesquisa de dados:** O trabalho deve indexar o conteúdo extraído dos arquivos, e também permitir realizar uma busca no conteúdo;
7. **Tipo de licenciamento:** Tipo de licenciamento que garante ao usuário o direito de acessar, modificar e executar o software.

As seções 2.1 e 2.2 descrevem, respectivamente, os trabalhos correlatos e softwares encontrados na literatura, assim como, algumas de suas características e funcionalidades.

2.1. Trabalhos Correlatos

A equipe da divisão de pesquisa da IBM criou em 2010 um sistema para tornar reuniões gravadas on-line compartilháveis e pesquisáveis (Topkara, *et al.* 2010). Este sistema focou exclusivamente em arquivos de reuniões da empresa. A pesquisa não informa nenhum suporte a escalabilidade em seu processamento. Não é *open source* e também não fornece muitos detalhes sobre a implementação da solução. Este projeto tem um diferencial que permitia indexar os slides utilizados durante as reuniões identificando o conteúdo nele presente.

Lawto *et al*. apresenta em (Lawto, *et al.* 2011) uma plataforma capaz de indexar transmissões de notícias e reportagens vindas de diversas fontes e em várias linguagens. Este trabalho precisa lidar com uma grande quantidade de dados e conta com um gerenciador de tarefas robusto. O sistema é distribuído por algumas máquinas e permite adicionar ou remover componentes sem afetar as tarefas que estão sendo executadas. O sistema é capaz de processar mais de 150 arquivos de áudio e/ou vídeos por dia, 3.5 GB ou 15 horas de conteúdo por dia. Os dados apresentados indicam que o sistema tem a capacidade de processar 100 horas de vídeos por dia. No entanto, não é possível inferir se é possível escalar o sistema para atender cargas de trabalhos maiores. Além disso, está é uma solução proprietária e não fornece muitos detalhes de sua implementação sendo apenas uma parte do estudo de um sistema comercial chamado Voxalead que na época continha 70 mil *podcasts* para serem processados.

Nouza, Blavka, *et al.* em (Nouza, Blavka, *et al.* 2012) e Nouza, Cerva, *et al.* em (Nouza, Cerva, *et al.* 2014) propõem a criação de um software para o processamento, indexação e busca nos arquivos de áudio e vídeos do patrimônio histórico e cultural da República Tcheca. Este software processou mais de 100 mil horas de arquivos, porém não deixa claro se é possível escalar o sistema. Ele utiliza filtros para tentar minimizar os problemas com ruídos nos arquivos. Este software foi patrocinado pelo governo da República Tcheca e não menciona se é de livre uso.

Chaudhary *et al*. em (Chaudhary, *et al.* 2017) descreve a construção de um reprodutor de mídia capaz de buscar palavras chaves enquanto o vídeo está sendo reproduzido. O sistema permitiria também buscar por tópicos específicos dentro do arquivo a fim de fazer o vídeo começar a rodar a partir do ponto escolhido. Esta solução

utiliza o CMU Sphinx para fazer o reconhecimento de fala. O sistema foi testado em alguns vídeos de palestras disponíveis *online*, apresentadas por diferentes locutores e com duração de 2 a 10 minutos. O sistema foi capaz de encontrar uma média de 60-65% das palavras chaves pesquisadas. Entretanto o sistema não menciona nenhum suporte a grande quantidade de dados e o processamento é realizado somente no arquivo em reprodução.

Eljazzar, Hassan e Al-Sharkawy *et al*. em (Eljazzar, Hassan e Al-Sharkawy 2017) apresenta um projeto aplicado ao estudo do Alcorão, o livro sagrado dos muçulmanos. Segundo estudos do autor, a maioria das pessoas ao pesquisar sobre o significado de algumas palavras específicas do Alcorão prefere assistir vídeos curtos em vez de ler um texto para entender o significado. Com isso a proposta do trabalho é dividir vídeos sobre o Alcorão em vídeos menores, extrair o conteúdo, disponibilizar nas redes sociais e permitir que o usuário comente sobre um trecho específico do arquivo. Este projeto usa o serviço em nuvem da Heroku que oferece a escalabilidade necessária para hospedar a aplicação. O resultado de cada pesquisa realizada é salvo para futuras análises. O projeto não menciona nenhum suporte a grande quantidade de dados e é focado em uma fonte de dados muito específica e restrita.

Sigcha *et al*. em (Sigcha, *et al.* 2017) apresenta um projeto similar ao de (Nouza, Cerva, *et al.* 2014), mas aplicado a transmissões de uma rádio equatoriana e com foco no espanhol. O diferencial desse projeto é que ele tenta identificar trechos do arquivo de áudio ou vídeo para identificar trechos de música, fala ou música com fala. O projeto tenta usar ferramentas disponíveis de livre uso. Entretanto, este projeto também não apresenta preocupação com processamento de um grande volume de dados e em indexação do conteúdo extraído.

2.2. Softwares

No geral, todas as ferramentas encontradas que se assemelham à solução proposta nesta pesquisa são softwares proprietários que exigem custos para o seu uso e utilizam tecnologias específicas, que não permitem muitas mudanças em sua arquitetura.

Sonix (Sonix 2020) é um software proprietário que transcreve os arquivos de áudio e vídeo para que seja possível pesquisar, editar e compartilhar em várias

linguagens. Este software promete converter o arquivo em um tempo muito pequeno: normalmente 30 minutos de áudio são processados em apenas três ou quatro minutos. Este software também permite que o usuário edite o texto extraído para poder corrigir partes que foram mal transcritas, como também permite destacar ou riscar partes do texto extraído para consultas futuras. Os arquivos a serem processados são entregues ao sistema através de uma página web ou o usuário pode disponibilizar os arquivos no Google Drive, ou Dropbox. O diferencial deste software é a utilização de técnicas de aprendizado de máquinas para melhorar o índice de acerto dos resultados conforme mais arquivos são processados. O sistema apresenta um bom investimento na aplicação web para poder gerenciar e utilizar o texto extraído. A título de exemplo, é possível pesquisar uma parte específica do texto e tocar o áudio naquele exato momento, controlar a velocidade do áudio, buscar e trocar palavras nos textos e fazer *download* de pequenos áudios de uma parte do texto selecionado. Além disso, o sistema é integrado a outros softwares de edição de áudio e vídeo como *Adobe Audition Integration* e *Adobe Premiere Integration*. Apesar de mostrar eficiência para processar arquivos de áudio, não há evidências de que este software seja capaz de escalar para suportar o processamento de grandes massas de dados.

A empresa Microsoft lançou em 2008 o sistema chamado *Microsoft Audio Video Indexing Service* (MAVIS) (MAVIS 2020). O MAVIS permitia buscar por palavras faladas e reduzia os erros expandindo o vocabulário, além de permitir gravar palavras alternativas usando uma técnica desenvolvida pela Microsoft para melhorar a precisão. Em 2014 a Microsoft anunciou o MAVIS publicamente como *Microsoft Azure Media Services Indexer* (Azure Media Services (AMS) 2020). Este serviço permite construir diferentes *workflows* em um ambiente em nuvem e tem suporte a diversas linguagens como .NET, Python, Go, Java entre outras. O sistema permite identificar pessoas e objetos nos vídeos, tem a capacidade de captar textos que aparecem na tela, analisar os vídeos para extrair *insights* como nível de satisfação do cliente e construir *dashboards* a partir do texto extraído. O sistema da Microsoft usa Azure CDN para distribuir o conteúdo a ser processado para permitir escalabilidade no processamento e oferece suporte a processamento de um grande volume de dados. Entretanto, este é um sistema proprietário que exige que as ferramentas da Microsoft sejam utilizadas e também envolve um custo para sua utilização.

Speechmatics (Speechmatics 2020) é o último software encontrado que se assemelha ao projeto desta dissertação. Este software faz conversão de fala em texto em tempo real e pode ser implementado em plataformas físicas ou virtuais dependendo das necessidades do usuário. O suporte a plataformas baseadas em *containers* garante que o sistema possa crescer sob demanda para processar volumes crescentes de dados. Entretanto, ele não menciona se tem algum sistema de busca e indexação dos dados, além de ser também um sistema pago.

2.3. Discussão

Os trabalhos e softwares correlatos que foram encontrados demonstram que existem pesquisas e softwares nesta área que tornam possível a extração do conteúdo falado de um arquivo de áudio ou vídeo, e que, em alguns casos, permitem realizar buscas no conteúdo extraído. Além disso, este estudo mostra a escassez de pesquisas e projetos dentro dessa área; a data de produção de alguns dos resultados encontrados não é recente.

Com base nas referências encontradas percebe-se que a maioria das pesquisas atende parcialmente ao problema proposto nesta dissertação e acabam deixando alguns pontos a serem melhorados ou explorados. Entre os pontos, como por exemplo, a utilização de técnicas de escalabilidade do processamento para permitir processar um grande volume de dados, indexar e pesquisar o conteúdo extraído, e não serem softwares *open source*.

Alguns dos projetos encontrados não têm a preocupação de trabalhar com uma grande quantidade de dados, enquanto outros são específicos para alguns cenários como processar alguns arquivos específicos de um determinado local. Além disso, as soluções que permitem realizar buscas em áudio ou vídeos não são comuns.

A Tabela 1 apresenta se os trabalhos e softwares encontrados atendem ou não os requisitos propostos como ideais para resolver o problema mencionado. Os valores da Tabela 1 são compostos pela seguinte legenda:

- Sim – O projeto atende ao requisito;
- Não – O projeto não tem suporte ao requisito;
- Talvez – O projeto não oferece dados suficientes para afirmar categoricamente.

Tabela 1. Matriz projeto x requisitos

Projeto	**Fonte de dados diversificada?**	**Suporte a grande quantidade de dados?**	**Suporte a escalabilidade?**	**Uso de software de reconhecimento de fala?**	**Indexação e pesquisa dos dados?**	***Open source*?**
(Azure Media Services (AMS) 2020)	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	Não
(Lawto, *et al.* 2011)	**Sim**	Talvez	Talvez	**Sim**	**Sim**	Não
(Nouza, Cerva, *et al.* 2014)	Não	Não	Não	**Sim**	**Sim**	Não
(Topkara, *et al.* 2010)	Não	Não	Talvez	**Sim**	**Sim**	Não
(Speechmatics 2020)	**Sim**	**Sim**	**Sim**	**Sim**	Não	Não
(Sonix 2020)	**Sim**	Não	Talvez	**Sim**	**Sim**	Não
(Eljazzar, Hassan e Al-Sharkawy 2017)	Não	Não	Talvez	**Sim**	Talvez	Não
(Sigcha *et al.* 2017)	Não	Não	Não	**Sim**	Talvez	Não
(Chaudhary, *et al.* 2017)	Não	Não	Não	**Sim**	Talvez	Não

3. Arquitetura Proposta

A arquitetura da solução proposta é de propósito geral e apresenta os componentes principais necessários para a construção de um sistema completo e robusto. A proposta é resolver os problemas apresentados para processar e indexar os arquivos de áudio e vídeo, de modo a permitir ao usuário realizar pesquisas nos conteúdos que são falados dentro dos arquivos.

Para a construção da arquitetura da solução proposta, cinco principais serviços ou componentes são sugeridos: fonte de dados, plataforma de processamento de *streams*, software de reconhecimento de fala, ferramenta de indexação e interface de pesquisa.

O **primeiro** componente é a **fonte de dados**, que diz respeito ao local de armazenamento dos arquivos de áudio e vídeo.

O **segundo** componente é a **plataforma de processamento de *streams***, que está relacionado em como os arquivos serão gerenciados e processados em paralelo. Este componente precisa ter a capacidade de escalar o processamento se necessário para suportar altas cargas de trabalho.

O **terceiro** componente é o **software de reconhecimento de fala**, peça fundamental para a extração das palavras faladas nos arquivos processados e que será utilizado pela ferramenta de processamento de *streams*.

O **quarto** componente é a **ferramenta de indexação**, que é o serviço necessário para salvar e indexar o conteúdo extraído dos arquivos de uma forma otimizada, de modo a deixar o conteúdo disponível para as buscas.

Por último o **quinto** componente é a **interface de pesquisa**, ferramenta que apresentará os resultados consultados na base indexada para o usuário.

Estes são os componentes sugeridos da arquitetura de solução para a construção de um sistema que esteja relacionado com o tema apresentado. Além disso, algumas técnicas e conceitos podem ser utilizados para deixar a arquitetura mais robusta e confiável. Estes podem estar incorporados ou serem aplicados a um ou mais dos componentes apresentados na solução, como por exemplo:

- ***Clustering***: Arquitetura de computação escalável, capaz de agrupar vários computadores para trabalharem em conjunto e melhorar o desempenho na execução de processos de alta complexidade ou carga;
- **Disponibilidade de dados ou tolerância a falhas**: Garantir que os dados sempre estarão disponíveis. Caso algum sistema ou servidor fique indisponível ele é substituído por outro para continuar a disponibilidade ou processamento dos dados;
- **Balanceamento de carga**: Capacidade de balancear a carga de trabalho para dividir e equilibrar o processamento pelos processadores, desta forma evitando que algum recurso fique ocioso enquanto outro está sobrecarregado.

A solução proposta não limita a adição de mais componentes ou serviços na arquitetura, o que permite adicionar e prover novos recursos ou funcionalidades não descritas nesta dissertação. Portanto, a arquitetura deixa livre a escolha se serão utilizados serviços existentes no mercado, sejam eles proprietários ou de livre uso, como também a decisão de desenvolvimento de um dos componentes descritos ou algum novo.

A Figura 3 apresenta a arquitetura geral da solução proposta para o processamento, indexação e pesquisa de grandes massas de dados em formato de áudio e vídeo. Todos os componentes descritos neste capitulo estão representados na Figura 3. Além disso, é possível visualizar a comunicação que existe entre os componentes.

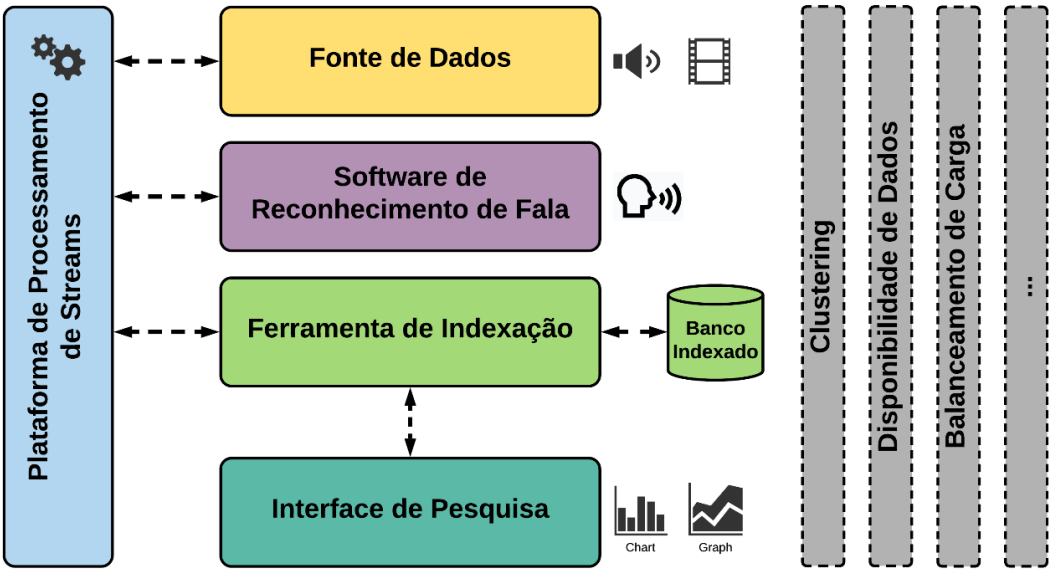

Figura 3. Arquitetura Geral Proposta.

Portanto, a arquitetura em questão é composta por cinco principais componentes: fonte de dados, plataforma de processamento de *streams*, software de reconhecimento de fala, ferramenta de indexação e interface de pesquisa. Os componentes estão descritos a seguir.

3.1. Fonte de Dados

A fonte de dados é o local de armazenamento dos arquivos de áudios e vídeos. Eles serão coletados pelo sistema que irá processa-los para extrair o texto. Este local deve permitir que o sistema tenha acesso remoto aos arquivos, de modo a evitar a necessidade da instalação do sistema no mesmo lugar no qual o repositório de arquivos se encontra.

A Figura 4 apresenta algumas possibilidades de como os arquivos de áudios e vídeos podem ser gerados, como também vários locais onde os arquivos podem ser armazenados.

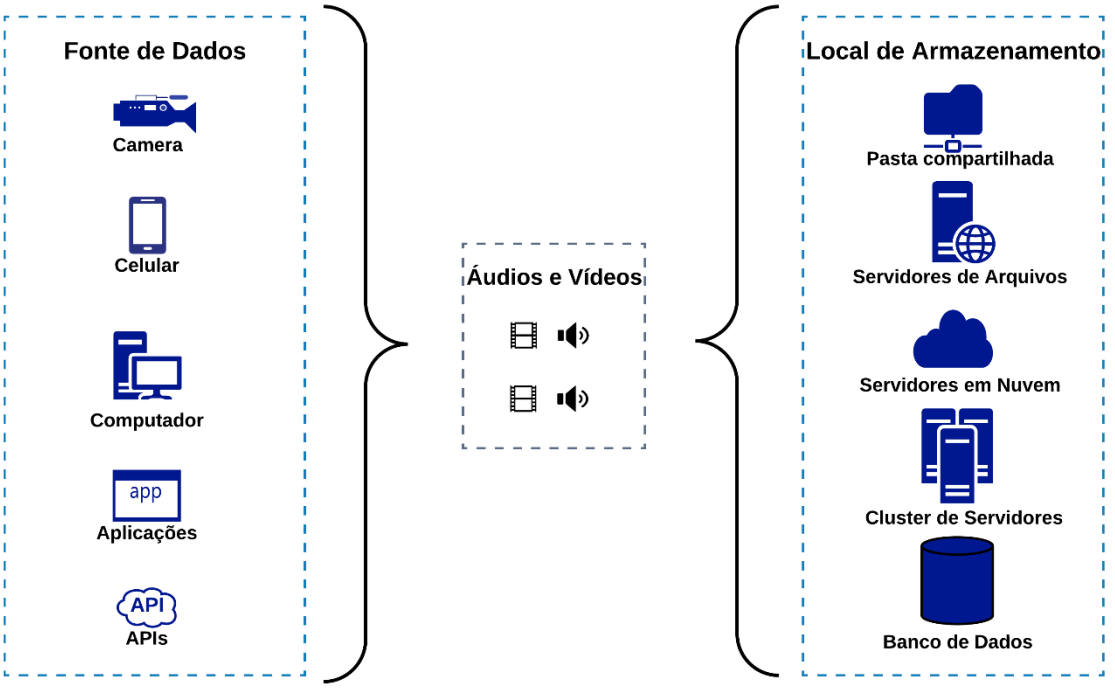

Figura 4. Sistema de armazenamento distribuído.

A arquitetura proposta suporta a utilização ou o desenvolvimento de vários conectores que podem coletar arquivos de áudio e vídeo diretamente do local de armazenamento para serem processados pelo sistema.

A proposta é a construção de conectores que possam recuperar os arquivos diretamente da fonte de dados. Por sua vez, que seja capaz de enviar o arquivo para a plataforma de processamento em um único formato comum, desta forma facilitando o processamento ao padronizar o formato a ser processado. Este conector pode ser desenvolvido ou dependendo das ferramentas utilizadas, a plataforma de processamento pode prover alguma API o mecanismo para facilitar a operação.

Os detalhes de processamento estão descritos no próximo capítulo.

3.2. Plataforma de Processamento de *Streams*

Plataformas de processamento de *streams* são ferramentas que fazem parte de um grupo de tecnologias desenvolvidas para trabalhar com *Big Data*. Estas ferramentas já contam com uma série de mecanismos para suportar, analisar, escalar, manter a disponibilidade dos dados e processar os dados.

Hu *et al*. em (Hu, et al. 2014) descrevem que, de acordo com o tempo de resposta do processamento destes dados, é possível categorizar o processamento em dois paradigmas, processamento de *streaming* e processamento em *batch*:

- **Processamento de *streaming* de dados**: Processa e analisa os dados assim que possível, geralmente com tempo de resposta de milissegundos. O conjunto de dados é infinito. Segundo Lopez, Lobato e Duarte (Lopez, Lobato e Duarte 2016), o processamento é utilizado em aplicações de baixa latência e com tempo de resposta em milissegundos. Souza (Souza 2015) afirma que o os dados não estão armazenados ou apenas uma parcela está salva em memória;
- **Processamento em *batch***: Os dados são primeiramente armazenados e depois analisados, com tempo de resposta mais demorado. Geralmente é um conjunto de dados finito. Lopez, Lobato e Duarte (Lopez, Lobato e Duarte 2016), descrevem que esta técnica tem grande latência e o tempo de resposta pode demorar alguns segundos. Souza (Souza 2015) afirma que o processamento passa várias vezes pelo dado e o tempo de resposta pode levar horas ou dias.

A utilização de uma ferramenta de processamento de *streams* permite que o sistema possa processar um dado em tempo real, à medida que o arquivo de áudio ou vídeo é gerado no sistema. Embora nada impeça a utilização de uma plataforma de

processamento em batch, que também seja capaz de organizar e distribuir os arquivos para serem processados.

Segundo Stonebraker, Çetintemel e Zdonik (Stonebraker, Çetintemel e Zdonik 2005), os sistemas de processamento de *stream* de dados atendem alguns requisitos principais que o difere dos sistemas de processamento em *batch*, estes requisitos estão descritos a seguir:

- **Manter os dados em movimento**: Para conseguir ter baixa latência dos dados, o sistema deve conseguir processar os dados sem ter uma operação de armazenamento que seja custosa. Os dados devem ser processados ”*in-stream*” conforme eles são gerados;
- **Consulta usando SQL nos *streams***: Nestas aplicações, algum mecanismo de consulta deve ser usado para encontrar eventos específicos ou computar análises em tempo real a partir dos *streams* de dados;
- **Lidar com inconsistências de *streams***: Capacidade de tratar os dados que são entregues atrasados, que estão ausentes ou fora de ordem. Tratar esses problemas sem bloquear o sistema é fundamental. Adicionalmente, faz com que se tenha baixa latência, já que os dados não estão salvos em um local de armazenamento e a infraestrutura permite que o processamento dos dados possa continuar mesmo com dados parciais;
- **Garantia de resultados previsíveis e com tolerância a falhas**: Diferentes execuções do mesmo processamento devem reproduzir o mesmo resultado, isto é importante no ponto de vista da tolerância e recuperação de falhas. Se caso o processamento não puder ser realizado por algum motivo, ele pode ser refeito em outra máquina e terá o mesmo resultado;
- **Integrar dados armazenados e dados de *streaming***: O sistema deve ter a capacidade de armazenar, acessar e modificar dados armazenados e combiná-los com dados de *streaming*. Uma tarefa muito comum é comparar dados do presente com o passado;
- **Garantia de segurança e disponibilidade de dados**: É preciso garantir a integridade dos dados e a alta disponibilidade dos dados. O sistema deve ter a habilidade de recuperação de uma falha e manter os dados consistentes apesar de

falhas. Com isso é preciso utilizar uma solução de alta disponibilidade para que o sistema sempre esteja disponível;

- **Particionar e escalar aplicações automaticamente**: Particionar e distribuir os dados para o processamento em múltiplas máquinas para escalabilidade sem que o desenvolvedor escreva código específico para que isso ocorra. O sistema de processamento de *stream* deve suportar operações de *multi-thread* e multiprocessamento, facilitando assim a baixa latência;
- **Processar e responder instantaneamente**: Com o fluxo de baixa latência e com alta disponibilidade dos dados, o sistema deve permitir o processamento de grandes volumes de dados em tempo real.

Para integrar as entidades geradoras e consumidoras de dados, plataformas de processamento de *streams* utilizam o paradigma *publish/subscribe*. De acordo com Eugster *et al*. (Eugster, *et al.* 2003) *subscribers* tem a capacidade de expressar seu interesse em um evento, ou um padrão de eventos, a fim de serem notificados posteriormente quando estes eventos acontecerem. *Publishers* criam eventos a partir dos dados que recebem. Cada *subscriber* recebe somente os eventos para os quais ele está registrado.

Estes eventos podem ser caracterizados e agrupados em tópicos ou assuntos (Eugster, *et al.* 2003). Por exemplo, os eventos podem ser categorizados como "áudios de MP3", "reportagens", ou simplesmente algo mais genérico como "áudio" ou "vídeos". Desta forma pode ser criado, por exemplo, um tópico que contenha somente arquivos de áudios ou algum grupo específico de arquivos.

Um *subscriber* pode, por exemplo, querer receber somente arquivos de áudio, então basta ele se registrar para receber somente eventos deste tópico. Quando um *publisher* dispara um evento deste tipo, ele será encaminhado a todos os *subscribers* que estiverem registrados, como mostrado na Figura 5.

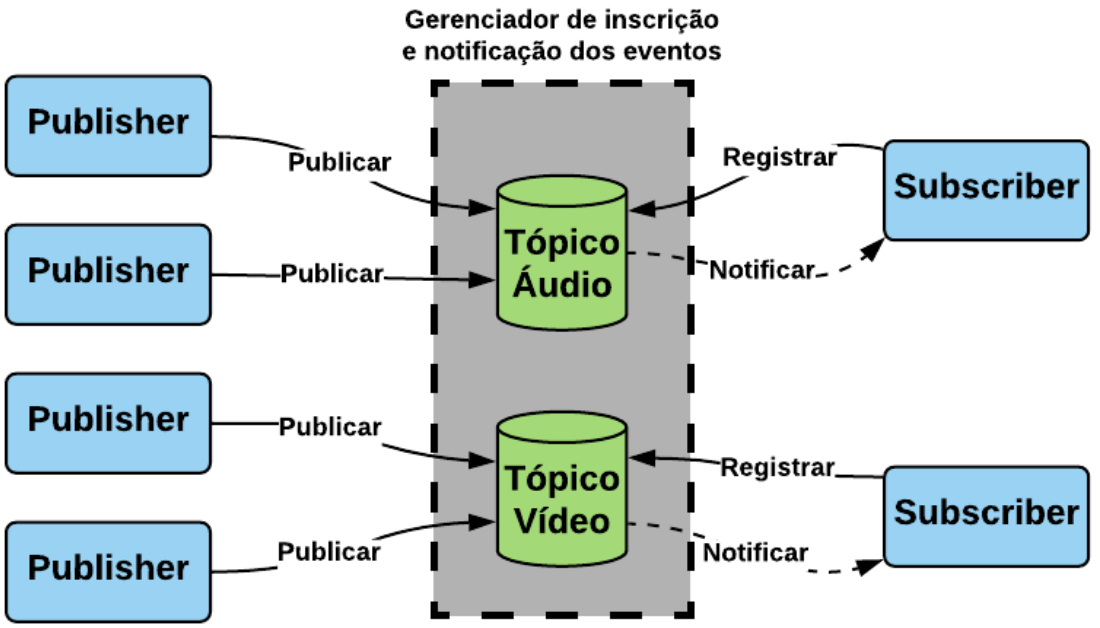

Figura 5. Solução *publish/subscribe*.

Os arquivos de áudios e vídeos podem já estar previamente armazenados em uma fonte de dados (processamento em *batch*), como podem estar sendo gerados no mesmo momento em que os processamentos estão ocorrendo (processamento de *stream*). A plataforma de processamento de *streams* pode tratar ambos cenários e deixar a coleta dos dados transparente para quem vai processar os dados, removendo a necessidade de se preocupar em como recuperar os dados para serem processados e em como processar dados que já existem ou novos dados que estão sendo criados.

A Figura 6 apresenta uma arquitetura que utiliza a plataforma de processamento de *streams* no padrão *publish/subscribe* que estão descritos como produtor/consumidor. Desta maneira o produtor não precisa se preocupar se o dado foi entregue para o consumidor para ser processado, esta responsabilidade é da plataforma de processamento de *stream*.

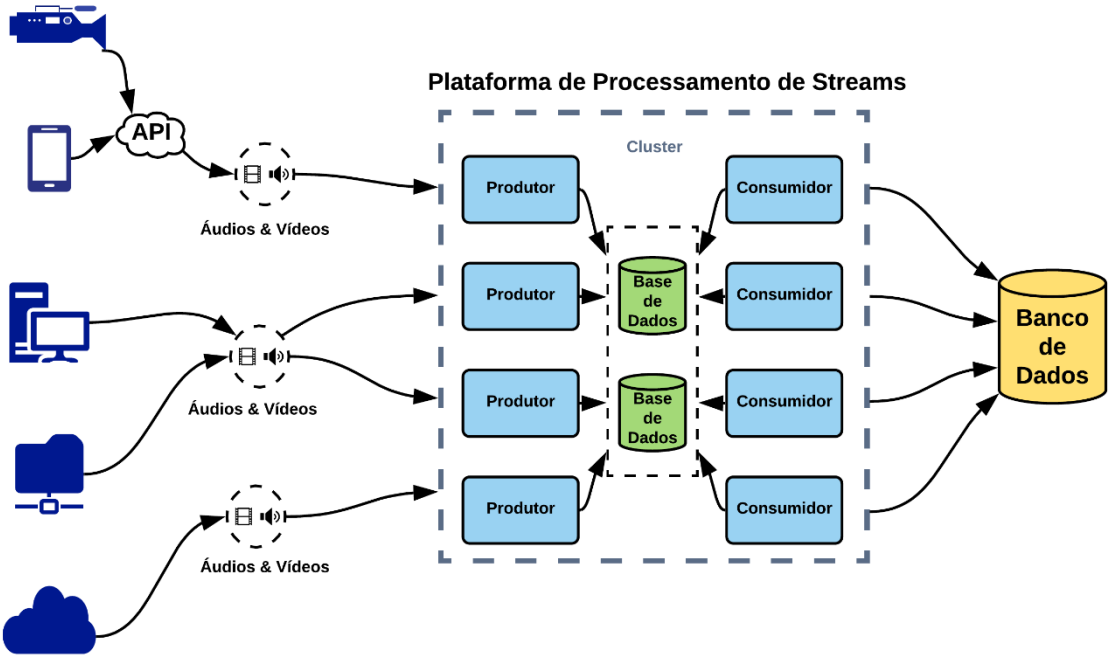

Figura 6. Arquitetura com a plataforma de processamento de *streams*.

O sistema pode ter vários produtores e vários consumidores o que permite escalar o componente de processamento dos dados. Os produtores e consumidores processam os dados dos mesmos ou de diferentes tópicos contendo uma parcela dos dados para serem processados. Além disso, a plataforma distribui os dados para serem processados por todos os consumidores garantindo o seu processamento por pelo menos um consumidor, sem se preocupar em gerenciar a fila de processamento.

Este tipo de plataforma pode funcionar em um cluster com várias máquinas unidas de modo a garantir a alta disponibilidade do serviço. Funciona com baixa latência de dados e conta com a habilidade de recuperação de falhas; caso alguma máquina ficar indisponível outra assume o lugar.

Os produtores podem utilizar diferentes conectores para coletar os dados de diferentes fontes de dados. Esta plataforma permite homogeneizar o formato e o local onde os dados estão armazenados. Desta forma os dados são processados sem a necessidade de criar diferentes consumidores com códigos customizados para tratar os diferentes tipos de arquivos.

Os consumidores criados são programas responsáveis por utilizar o software de reconhecimento de fala para extrair o texto falado nos arquivos de áudio e vídeo e enviar o conteúdo para um banco indexado.

A ferramenta de processamento de *streams* torna possível processar os arquivos de áudio e vídeo em tempo real ou quase em tempo real à medida que eles são gerados e estão disponíveis para o processamento. Além disso, permite processar arquivos em *batch* que já foram criados previamente.

Alguns exemplos de ferramentas que podem ser utilizadas para realizar o processamento dos dados em *batch* ou *stream* são: Hitachi Content Intelligence (HCI) (Hitachi 2017), Pentaho Data Integration (PDI) (Casters, Bouman e Dongen 2010), Apache Flink (Carbone, *et al.* 2015), Apache Spark (Penchikala 2017), Apache Samza (Noghabi, *et al.* 2017), Apache Storm (Toshniwal, *et al.* 2014) e Amazon Kinesis (Amazon Web Services (AWS) 2017), Apache Kafka (Garg 2013) entre outros.

3.3. Software de Reconhecimento de Fala

Ferramentas de reconhecimento automático de fala também são conhecidas como ASR (*Automatic speech recognition*), TTS (*Text to speech*) ou STT (*Speech to text*) (Lakdawala, *et al.* 2018). Elas podem converter texto em fala, fala em texto e realizar reconhecimento de fala em tempo real.

Telmem e Ghanou (Telmem e Ghanou 2018) afirmam que ASR é uma técnica computacional para transcrever um sinal de fala em texto. Entretanto, o sinal de fala é um dos mais complexos e difíceis de transcrever para texto, devido à qualidade dos arquivos, diferentes sotaques dos locutores, vocabulários, gírias, entre outros. Com isso várias técnicas e métodos matemáticos foram desenvolvidos para resolver este problema.

Martins (Martins 1998) categoriza os sistemas reconhecedores em três grandes classes com base na técnica utilizada para o reconhecimento: reconhecedores baseados na análise acústico-fonética, reconhecedores empregando inteligência artificial e reconhecedores por comparação de padrões.

3.3.1. Reconhecedores Baseados na Análise Acústico-Fonética

Os reconhecedores baseados na análise acústico-fonética são aqueles que decodificam o sinal de fala com base nas características acústicas e fonéticas. Este método tenta identificar unidades fonéticas que compõe a palavra a ser reconhecida, e a partir da concatenação dessas unidades, reconhecer a fala (Martins 1998). Martins e Bresolin (Martins 1998) e (Bresolin 2003) descrevem três etapas envolvidas neste método de reconhecimento.

A primeira etapa para o método de reconhecimento por análise acústico-fonética é a análise do sinal de voz ou análise espectral, que envolve a segmentação do sinal de voz em regiões discretas no tempo, esta etapa também está presente em todos os outros métodos de reconhecimento de fala. Este tipo de análise utiliza algumas técnicas bastante conhecidas, por exemplo, banco de filtros e transformada discreta de Fourier.

A segunda etapa é detectar as características do sinal, convertendo as medidas da análise espectral em várias características que descrevem as unidades fonéticas: presença ou não de ressonância nasal, presença ou não de excitação da fala, classificação com ou sem voz, dentre outras características.

A terceira etapa é a fase de segmentação e rotulação do sinal de fala. O sistema procura encontrar e rotular regiões segmentadas de acordo com as unidades fonéticas individuais, para a escolha da palavra que melhor corresponde à sequência de unidades. Nesta etapa, são utilizados vários controles estratégicos para limitar a gama de pontos de segmentação e probabilidades de rótulo e assim, melhorar e facilitar o reconhecimento.

A Figura 7 apresenta um diagrama de blocos do sistema de reconhecimento de fala acústico-fonética.

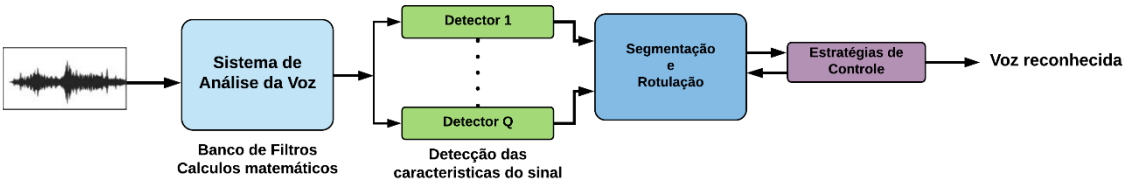

Figura 7. Reconhecimento de fala acústico-fonética.

3.3.2. Reconhecedores por Inteligência Artificial

Os reconhecedores por inteligência artificial podem utilizar redes neurais na sua implementação. Este método utiliza várias fontes de conhecimento acústico, léxico, sintático, concorrente e pragmático das sentenças (Bresolin 2003):

- **Conhecimento acústico:** Detecta evidência ou não do som. O sinal é medido de modo a detectar a presença ou não das características do som;
- **Conhecimento léxico:** Detecta evidência acústica do som de palavras sendo faladas. Utiliza um mapa léxico de palavras;
- **Conhecimento sintático:** Detecta a combinação das palavras na forma gramaticalmente correta nas sentenças ou frases;
- **Conhecimento semântico:** Detecta padrões de concorrência das palavras e se o seu entendimento está de acordo com a sentença;
- **Conhecimento pragmático:** Refere-se à habilidade para resolver casos de ambiguidade de significado.

Este método utiliza mecanismos para aprendizagem e adaptação, que são implementados através de redes neurais, com o objetivo de adquirir e refinar o conhecimento necessário para reconhecer o que é falado no áudio.

A rede neural é um processador paralelo e distribuído, composto por unidades de processamento chamadas neurônios. Os neurônios estão interligados por conexões com pesos que armazenam o conhecimento (Haykin 2001).

3.3.3. Reconhecedores por Comparação de Padrões

Os sistemas reconhecedores por comparação de padrões realizam um treinamento do sistema para o reconhecimento de padrões de fala. Segundo Bresolin (Bresolin 2003), este método utiliza dois passos principais:

- **Treinamento de padrões de fala**: A máquina aprende através de exemplos os padrões a serem reconhecidos. Por exemplo, utilizar uma base de dados com arquivos de áudio com diferentes locutores falando o texto específico;
- **Reconhecimento**: Realiza uma comparação entre o padrão desconhecido e os padrões de referência gerados durante o processo de aprendizagem. Nesta etapa,

ocorre um cálculo de medida de similaridade com o objetivo de encontrar o padrão que melhor corresponde ao padrão desconhecido.

Martins (Martins 1998) e Bresolim (Bresolin 2003) citam que os reconhecedores por comparação de padrões são os mais utilizados e com melhores resultados. Podem utilizar diferentes algoritmos, podem ser aplicáveis a diferentes vocabulários e regras de decisão e apresentam alto desempenho de processamento. Os Modelos Ocultos de Markov (HMM) são modelos estatísticos para reconhecimento de fala que utilizam este tipo de metodologia.

Rabiner e Juang (Rabiner e Juang 1986) definem os Modelos Ocultos de Markov como um processo duplamente estocástico. O que significa que é composto por dois conjuntos com estados indeterminados, com origem em eventos aleatórios. Dos dois conjuntos, um é composto por um modelo estocástico que não é observável (que está oculto), mas que só pode ser observado através de outro conjunto de processos estocásticos que produzem uma sequência de símbolos observados.

Espindola (Espindola 2009) apresenta um exemplo para facilitar o entendimento do HMM. Suponha uma cadeia de estados relacionada à condição do tempo em um determinado dia definida como {S1= chuvoso, S2 = nublado, S3 = ensolarado}. Esta cadeia de estado está oculta do observador, entretanto é possível observar o comportamento de um trabalhador ao se locomover para o trabalho em função do tempo ou de sua previsão e, através de padrões, dizer a probabilidade do tempo naquele dia.

Por exemplo, o trabalhador geralmente vai ao trabalho de bicicleta, mas costuma pegar um taxi em dias chuvosos. Consequentemente, se ele for ao trabalho de bicicleta em um determinado dia, existe uma probabilidade maior de que o dia esteja ensolarado do que chuvoso, embora ainda exista a possibilidade de ocorrência de chuva. Portanto, com um modelo que está oculto como a condição do tempo e a partir de outro conjunto observável como o comportamento de um trabalhador, foi possível dizer qual é a probabilidade de estado do tempo naquele dia.

Atualmente, existem sistemas de reconhecimento de fala com um amplo vocabulário; os LVCSR (Large Vocabular Continuous Speech Recognition). Neste caso os padrões a serem reconhecidos podem ser sentenças ou frases, ao invés de reconhecer apenas uma palavra isolada (Tevah 2006).

A arquitetura destes sistemas que usam reconhecimento de padrões é composta pelos seguintes componentes (Tevah 2006): interface de captação e extração de parâmetros do sinal de fala (conhecido como *front-end* ou pré-processamento), modelo acústico, modelo léxico ou dicionário fonético, modelo de linguagem e um decodificador. A Figura 8 apresenta a arquitetura descrita.

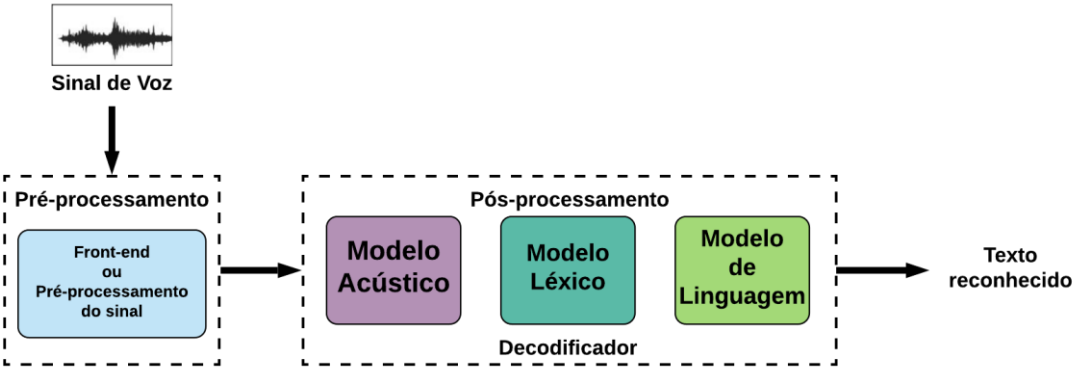

Figura 8. Reconhecimento de fala por comparação de padrões.

3.3.3.1. Pré-Processamento ou Front-End

O pré-processamento/*front-end* é responsável por coletar, anotar e processar os dados de entrada contendo a fala. Ele deve converter a onda da fala em uma representação paramétrica mais fácil de ser analisada posteriormente (Vimala e Radha 2012).

Tevah (Tevah 2006) descreve que a principal função desta etapa é dividir o sinal de fala em blocos menores. Normalmente, o espaçamento entre os blocos e o tamanho dos mesmos para análise é de 10ms e 25ms respectivamente. Com esta divisão é gerado um vetor acústico para análise.

As anotações fornecidas pelo *front-end* também incluem o início e o final de um segmento de dados. Algumas operações podem ser executadas a fim de melhorar os dados para análise. Estas operações podem ser: cancelamento de ruídos, controle automático de ganho, finalização, aplicações de filtros e análises matemáticas (Telmen 2018).

3.3.3.2. Pós-Processamento

Depois do pré-processamento vem a etapa de pós-processamento. Esta etapa envolve os componentes: modelo acústico, modelo léxico/dicionário fonético, modelo de língua e o decodificador para reconhecer a palavra falada (Vimala e Radha 2012).

Tevah (Tevah 2006) e Su (Su 2018) descrevem o método estatístico utilizado para o reconhecimento das palavras. Para uma sequência de observação acústica X (forma de onda de áudio), o sistema utiliza a probabilidade máxima posteriori que é baseada na inferência bayesiana, para encontrar a melhor estimativa W de sequência de palavras pronunciadas. Este método é representado na equação (1).

$$\arg\max\left[P(W \mid X)\right] = P(X \mid W)P(W) \tag{1}$$

Esta equação está dividida e descrita a seguir, na qual:

- **W** é uma sequência de palavras;
- **X** é a forma da onda acústica;
- **P (X|W)** é a probabilidade de observar a evidência acústica **X** quando a sequência **W** é pronunciada, **modelo acústico**;
- **P (W)** refere-se às palavras que poderiam ter sido pronunciadas, **modelo de língua**.

O modelo acústico **P (W|X)** identifica os padrões de fonemas presentes no áudio e, com esses fonemas, através do modelo léxico, identifica as palavras que foram pronunciadas. Então o modelo acústico **P (W|X)** é calculado pelas sequências das subunidades (fonema) geradas. Desta forma, pode ser introduzido no modelo uma sequência de **fonemas Y** usados para definir a **palavra W**; este componente é chamado de **modelo léxico** ou **dicionário fonético**. Desta forma, a fórmula final para incluir o conceito de fonemas é apresentada em (2).

$$\arg\max\left[P(W \mid X)\right] = P(X \mid Y)P(Y \mid W)P(W) \tag{2}$$

Esta equação está dividida e descrita a seguir, na qual:

- **W** é uma sequência de palavras;
- **X** é a forma da onda acústica;

- **Y** é uma sequência de fonemas;
- **P (X|Y)** é a probabilidade de observar a evidência acústica **X** quando a sequência de fonema **Y** é pronunciada, **modelo acústico**;
- **P (Y|W)** é a probabilidade de observar uma sequência de fonema **Y** quais são as possíveis palavras **W** formadas pelos fonemas, **modelo léxico**;
- **P (W)** refere-se às palavras que poderiam ter sido pronunciadas, **modelo de linguagem**.

A título de exemplo, vamos utilizar a seguinte frase em inglês, "***this is speech***", como parâmetro de entrada para o sistema.

- **Modelo acústico** identifica o seguinte o padrão de fonemas:
  - (th-ih-s-ih-z-s-p-ih-ch)
- **Modelo léxico** identifica palavras para o padrão encontrado:
  - (th-ih-s) → this
  - (ih-z) → is
  - (s-p-ih-ch) → speech
- **Modelo de linguagem** identifica as probabilidades do que foi falado:
  - (this) – (is) – (speech)
  - P(this) P(is | this) P(speech | this | is)
- O **decodificador** realiza a busca final do que foi falado e apresenta o texto final "***this is speech***".

3.3.3.2.1. Modelo Acústico

O modelo acústico provê um método que calcula a semelhança de qualquer sequência de vetores de parâmetros X (forma da onda acústica), que foi extraído de um sinal de voz através do processamento digital, com a sequência correspondente de palavras W mais provável de ocorrer (Tevah 2006).

Tevah também descreve que a distribuição de probabilidade de P (X|W), pode ser modelada através de várias palavras e o cálculo estatístico de sequências de vetores correspondentes. Entretanto, esse método é impraticável para sistemas com amplo vocabulário, e ao invés disso, as palavras são modeladas pelos seus respectivos fonemas.

A modelagem acústica busca criar representações estatísticas para as sequências do vetor de parâmetros X geradas a partir da forma de onda da fala. Este modelo abrange a modelagem de pronúncia das palavras, que descreve como uma ou várias sequências de unidades de fala (fonemas) são usadas para gerar unidades de fala maiores. Como por exemplo, palavras ou frases que são objetos de reconhecimento de fala. Em outras palavras, esta modelagem tenta identificar quais são as sequências de fonemas a partir da forma da onda reconhecida (Vimala e Radha 2012).

3.3.3.2.2. Modelo Léxico / Dicionário Fonético

O modelo léxico/dicionário fonético, também conhecido como dicionário de pronúncia, contém detalhes de como as palavras são pronunciadas. Este modelo contém todas as palavras que o reconhecedor de palavras pode identificar.

O dicionário contém as palavras associadas com suas respectivas sequências fonéticas. Como algumas palavras podem ter várias pronuncias, o arquivo de dicionário pode conter mais de uma sequência fonética associada à mesma palavra (Vimala e Radha 2012).

Este componente faz a ponte entre as palavras na forma escrita e a sua forma acústica. Com o dicionário é possível reconhecer palavras que não aparecem no conjunto que foi previamente treinado para ser reconhecido (Su 2018). No exemplo a seguir, extraído de um dicionário em inglês utilizado pela ferramenta CMU Sphinx, há palavras com mais de uma sequência fonética. Especificamente, a palavra ***finance*** contém três possíveis sequências fonéticas.

…
finals F AY N AH L Z
finamore F IH N AH M AO R
finan F IH N AH N
finance F AH N AE N S
finance(2) F IH N AE N S
finance(3) F AY N AE N S
financed F IH N AE N S T
financement F IH N AE N S M AH N T

…

3.3.3.2.3. Modelo da Língua

Um modelo comum da língua é prover um mecanismo que estime a probabilidade de encontrar a palavra W em uma sentença em função das palavras que ocorrem em conjunto. Este modelo é utilizado para restringir a pesquisa de palavras, ele ajuda o reconhecedor de fala a descobrir uma probabilidade de uma sequência de palavras independente de sua acústica.

O modelo contém o conhecimento prévio sobre uma linguagem, como as possíveis sequências de palavras ou a frequência com que elas ocorrem na linguagem. Este modelo normalmente é treinado a partir de uma grande quantidade de fontes de dados contendo áudios e seus textos transcritos em que cada palavra falada deve existir no dicionário.

3.3.3.2.4. Decodificador

O decodificador é responsável por realizar a busca em todas as possíveis sequências de palavras. Este componente utiliza e combina os resultados do modelo acústico e do modelo de linguagem; as sequências com as maiores probabilidades são produzidas como resultado do reconhecimento.

Segundo Tevah (Tevah 2006) este processo busca por todos os modelos das palavras formadas por seus respectivos modelos de fonemas, os compara com uma sequência de vetores acústicos que contém as possíveis palavras para os fonemas. Este processo de busca pode ser muito custoso em termos computacionais, pois o número de modelos possíveis cresce com o vocabulário existente. Em sistemas de reconhecimento de fala continua, é esta etapa que define a velocidade final do sistema.

Algumas ferramentas de reconhecimento de fala que podem ser utilizadas na arquitetura proposta são CMU Sphinx (Lamere, *et al.* 2003), Kaldi (Povey, *et al.* 2011), HTK (Young, *et al.* 2009), IBM Watson (High 2012), DeepSpeech (Mozilla 2020) e Google Speech API (Adorf 2013), entre outros.

3.4. Ferramenta de Indexação

O objetivo deste componente é realizar a indexação e permitir a pesquisa textual do conteúdo extraído dos arquivos de áudios e vídeos. Desta forma, a ferramenta permite pesquisar por palavras chaves ou frases contidas no texto extraído pela ferramenta de reconhecimento de fala.

O resultado final do processamento dos arquivos nesta arquitetura é um grande conjunto de documentos textuais. Estes documentos precisam ser armazenados e organizados em algum local ou estrutura que seja eficiente para realizar buscas no conteúdo.

Ferramentas com a finalidade de armazenar e realizar buscas em textos são chamadas de sistema de banco de dados de texto ou também de motores de busca (Zobel e Moffat 2006). Estes sistemas funcionam similarmente aos bancos de dados tradicionais, onde os documentos são armazenados e uma indexação para otimização da busca é criada e mantida.

Ramakrishnan e Gehrke (Ramakrishnan e Gehrke 2000) definem indexação como uma técnica geral que pode acelerar a recuperação de registros com determinados valores no campo de pesquisa. A aplicação da técnica de indexação é necessária na arquitetura proposta para permitir buscas eficientes no conteúdo que foi previamente extraído.

A ferramenta de indexação e pesquisa precisa ter suporte para armazenamento de uma grande quantidade de dados. O objetivo é resolver o problema existente da indexação e pesquisa de texto em vários documentos, com o intuito de encontrar qual documento contém uma determinada palavra ou um conjunto delas, sem que demore muito tempo para retornar o resultado.

Realizar uma busca para encontrar todos os textos que citam uma palavra específica, utilizando um método de busca tradicional, pode ser custoso e levar muito tempo para retornar o resultado caso exista uma grande quantidade de arquivos. Uma busca tradicional pode ser realizada através da busca de palavra por palavra ou até mesmo letra por letra, dentro de todos os arquivos existentes.

Devido à grande quantidade de arquivos existentes e à busca por palavras dentro do conteúdo textual, é preciso utilizar métodos e técnicas de armazenamento e de busca para que a busca não seja um gargalo no sistema.

A seguir serão apresentadas duas estratégias que ajudam no processo de busca por palavras dentro do conteúdo textual: remover *stop words* e armazenar os dados em índices invertidos.

3.4.1. Índice Invertido

Índice invertido, conhecido como *inverted list*, *inverted index* ou até mesmo *inverted file*, é uma estratégia de indexação que permite a realização de buscas precisas e rápidas em conteúdo de texto. Zobel e Moffat (Zobel e Moffat 2006) afirmam que esta estratégia de indexação é uma das mais eficientes para busca em texto.

O índice invertido é uma estrutura de índice que permite a recuperação rápida de todos os documentos que contêm um determinado termo de consulta (Ramakrishnan e Gehrke 2000). O índice invertido é composto por dois principais componentes: a estrutura de pesquisa ou vocabulário e uma lista invertida (Zobel e Moffat 2006).

- **Estrutura de pesquisa ou vocabulário:** Contém todos os termos distintos que são indexados. Para cada termo distinto é salva uma frequência da ocorrência do termo, para saber em quantos documentos o termo aparece e, um ponteiro para o início da lista invertida;
- **Lista Invertida:** Para cada termo existe uma lista invertida que contém os identificadores dos documentos que contém o termo e a frequência com que o termo aparece no documento.

Em algumas implementações do índice invertido, cada termo distinto no vocabulário recebe um ID (Identificação) único do tipo inteiro. Para cada ID existe uma lista invertida que apresenta em quais documentos e a posição onde o termo aparece, este ID faz a ligação entre os dois componentes (Transier 2010).

Zobel e Moffat (Zobel e Moffat 2006) citam que a estrutura de índice invertido pode conter várias outras estruturas, como por exemplo, incluir um mapeamento do

documento com sua localização no disco. Os autores também afirmam que essas estruturas não são limitadas somente às estruturas que foram descritas.

A Figura 9 apresenta uma estrutura de índice invertido com base na junção das estruturas descritas em (Zobel e Moffat 2006) e (Transier 2010):

- **Vocabulário:** A estrutura de busca contém todos os termos **(t)** existentes, que estão referenciados por um **ID** único para sua identificação e a frequência **F (t)** com que os termos aparecem em todos os documentos. A utilização de um ID inteiro para referenciar as palavras deixa a busca mais rápida, por comparar um número ao invés de um texto;
- **Lista invertida:** Esta estrutura contém o **ID** do termo do vocabulário com uma lista contendo informações do ID do documento (**Documento ID**), frequência **F (t)** com que o termo aparece no documento e uma lista ([...]) com as posições em que o termo aparece no documento.

Vocabulário

ID	Termo (t)	Frequência F (t)
0	a	1
1	de	2
2	descreve	2
3	documento	3
4	estratégia	1
5	exemplo	1
6	indexação	4
7	indexar	1
8	invertida	2
9	para	1
10	que	2
11	um	1
12	uma	1

Lista Invertida

ID	(Doc ID, F (t), [posições]), ...
0	(1, 1, [4])
1	(2, 1, [3]), (3, 1, [6])
2	(1, 1 [3]), (3, 1, [3])
3	(1, 1, [1]), (3, 2, [1, 11])
4	(3, 1, [5])
5	(2, 1, [2])
6	(1, 1, [5]), (2, 2, [1, 4]), (3, 1, [7])
7	(3, 1, [9])
8	(1, 1, [6]), (2, 1, [5])
9	(3, 1, [8])
10	(1, 1, [2]), (3, 1, [2])
11	(3, 1, [10])
12	(3, 1, [4])

Figura 9. Estrutura do Índice Invertido.

Está estratégia inverte a maneira em que o sistema realiza a busca. Por exemplo, ao realizar uma busca pelo termo “reportagem”, o sistema pode abrir em uma base de dados, documento por documento, e realizar uma varredura no conteúdo para encontrar o termo de busca. Entretanto, ao invés disso o sistema busca pelo termo diretamente no vocabulário, que por sua vez vai conter a referência para todos os documentos no qual o termo aparece com a sua frequência e posições nos documentos.

3.4.2. Remover *Stop Words*

Wilbur e Sirotkin em (Wilbur e Sirotkin 1992) definem *stop words* como palavras que não são relevantes para a consulta, mas que têm a mesma ou uma maior probabilidade de ocorrência das palavras relevantes nos documentos.

Em técnicas de indexação de documentos, cada palavra é considerada como um potencial termo de índice de busca para os documentos. Várias destas palavras não são relevantes e podem trazer documentos que não são relevantes para a busca, por serem termos comuns, portanto, é muito comum remover esses termos para não serem indexados.

Remover esses termos pode ajudar a economizar espaço nos índices e em alguns casos podem melhorar o desempenho na busca dos documentos.

A Figura 10 apresenta a mesma estrutura utilizada na Figura 9 ao aplicar a técnica de remoção das palavras de paradas, é possível visualizar que somente as palavras relevantes serão indexadas reduzindo a quantidade de palavras a serem indexadas. Desta forma, a indexação fica mais eficiente e precisa e com base no exemplo apresentado na Figura 9 e Figura 10 é possível ver que o vocabulário inicial de 13 palavras foi reduzido para 7.

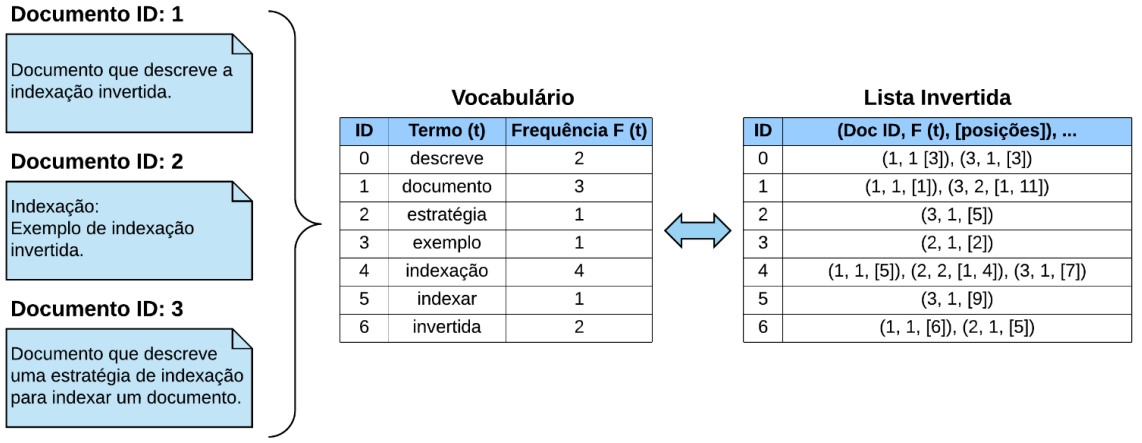

Figura 10. Estrutura do Índice Invertido com as Palavras de Parada Removidas.

A ferramenta de indexação provê comumente mecanismos para realizar a busca nos dados que estão indexados através de conectores e APIs para integrar com interfaces de pesquisas. Em alguns casos as próprias ferramentas já contam com uma interface de pesquisa acoplada em sua arquitetura.

Alguns exemplos de ferramentas para indexação de dados são: Elasticsearch (Gormley e Tong 2015), Apache Solr (Shahi 2015), Sphinx Search Engine (Ali 2011), RediSearch (RediSearch 2020), Algolia (Algolia 2020) e Amazon CloudSearch (Amazon CloudSearch 2020), entre outros.

3.5. Interface de Pesquisa

Este é o último componente que completa a arquitetura proposta, uma interface de pesquisa para o usuário. O objetivo é utilizar uma ferramenta que faça a interface entre os dados que estão sendo gerados pelo sistema e as pesquisas que o usuário deseja realizar.

A interface de pesquisa deve utilizar conectores ou APIs para conectar com a base de dados indexada. Além disso, deve permitir a customização da sua interface sem necessitar o desenvolvimento de código específico para isso.

No contexto deste projeto, a interface de pesquisa deve retornar como resultado a referência para todos os arquivos de áudios e vídeos nos quais aparecem uma ou mais das palavras que foram pesquisadas com suas posições em horas, minutos e segundos dentro dos arquivos.

A Figura 11 apresenta um esboço de uma interface de pesquisa. Neste exemplo, existe um campo de texto para pesquisar por alguma palavra, um botão de pesquisar, uma tabela apresentando os resultados com os detalhes, e dois gráficos gerados com base no conteúdo que foi indexado. Um gráfico de barra horizontal que apresenta as palavras com mais ocorrência por áudio e vídeo, e um segundo gráfico de pizza que mostra a porcentagem dos arquivos agrupados por tipo de arquivo.

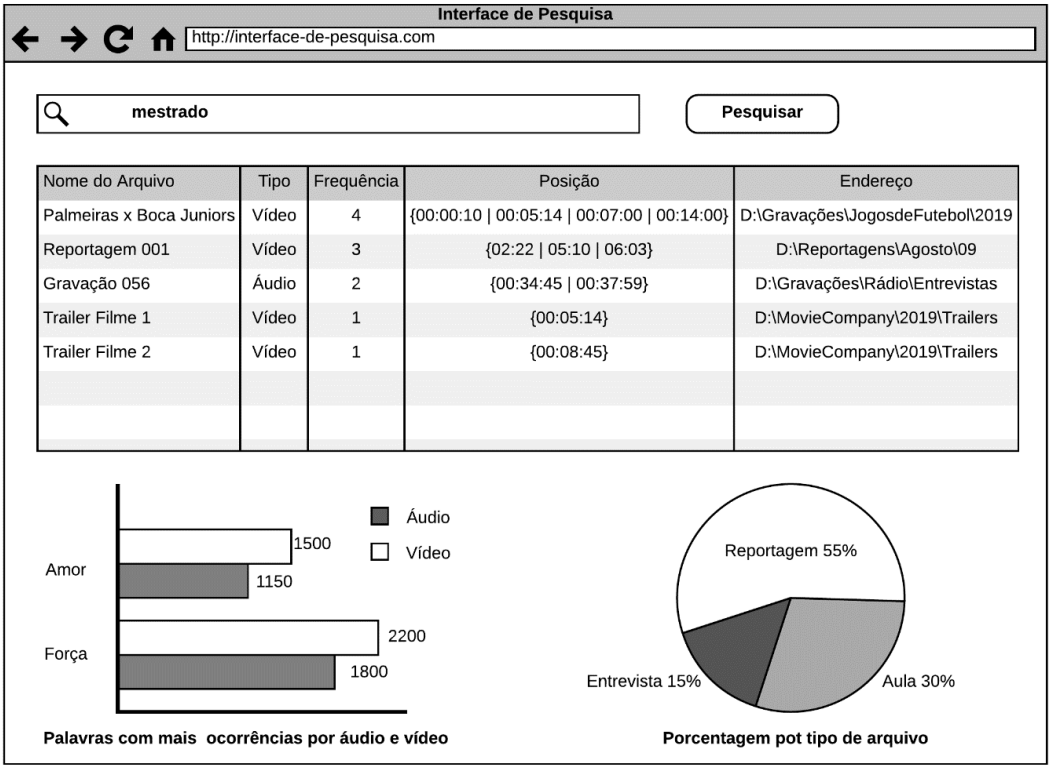

Figura 11. Exemplo de Interface de Pesquisa.

Alguns exemplos de ferramentas que podem ser utilizadas para a construção da interface são: Kibana (Gupta 2015), Grafana (Grafana 2020), Splunk (Carasso 2012), Sisense (Sisense 2020), QlikView (Troyansky, Gibson e Leichtweis 2015) e Tableau (Murray 2013), entre outros.

3.6. Síntese do Capítulo

A Figura 12 apresenta detalhes internos adicionais à arquitetura de solução para sumarizar alguns aspectos descritos. Como por exemplo, as diversas fontes de dados que podem ser utilizadas, o conceito de produtor e consumidor, como o software de reconhecimento de fala é utilizado pela ferramenta de processamento. A figura também apresenta o uso de *clustering* na plataforma de processamento de streams e na ferramenta de indexação para aumentar o desempenho e o poder computacional. Além disso a figura apresenta o fluxo dos dados na arquitetura proposta.

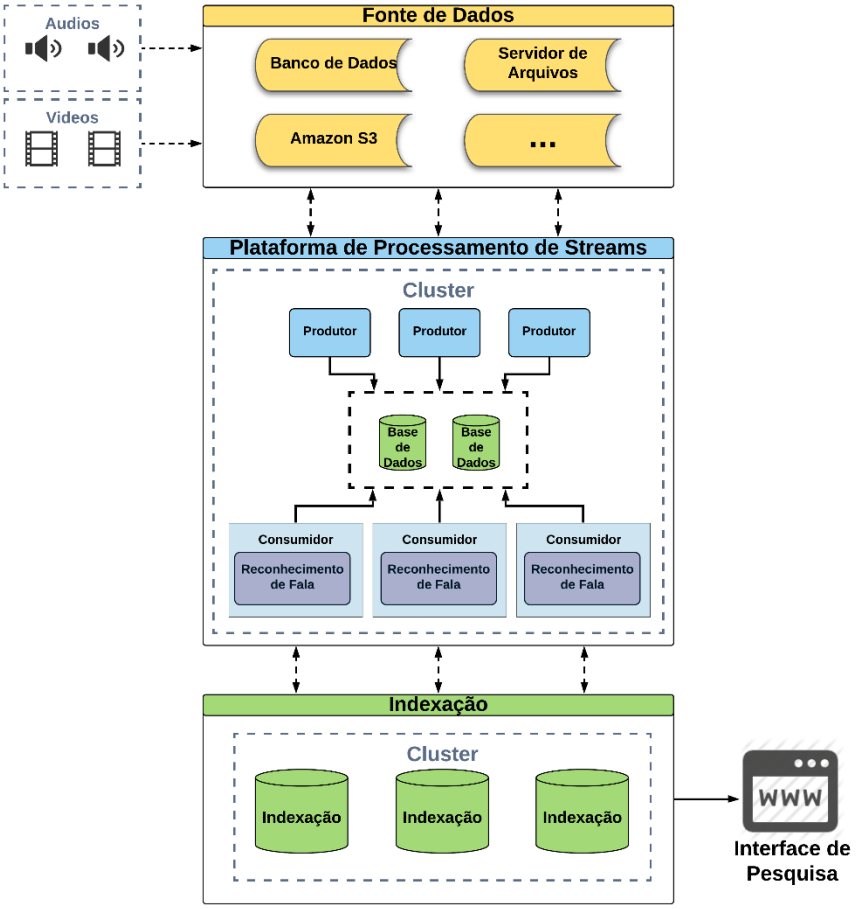

Figura 12. Fluxo dos dados na Arquitetura Proposta.

A Figura 12 apresenta o fluxo dos dados durante o processamento, descrito a seguir:

1. Os arquivos de áudio e vídeo estão sendo gerados e salvos na fonte de dados;
2. Os produtores estão recuperando os arquivos da fonte de dados através de diversos conectores, e envia os dados para a base de dados da plataforma de processamento de *streams*;
3. Os consumidores que estão registrados para processar os dados recuperam os dados da base de dados da plataforma de *streams* e através do software de reconhecimento de fala, extraem o conteúdo falado nos arquivos para enviar à ferramenta de indexação dos dados;
4. A ferramenta de indexação recebe o conteúdo extraído dos arquivos e os armazena de uma maneira otimizada permitindo consultas em sua base de dados;
5. Por fim a interface de pesquisa conecta com a base indexada para permitir ao usuário realizar consultas sobre o conteúdo extraído dos arquivos.

4. Implementação do Protótipo

Este capítulo apresenta os detalhes da implementação do protótipo, seguindo as diretrizes da arquitetura proposta. São apresentados alguns detalhes e explicações técnicas da construção do protótipo e dos componentes escolhidos. O objetivo é provar na prática que a arquitetura de solução pode resolver o problema proposto nesta dissertação.

As tecnologias apresentadas foram escolhidas por serem *open source* e por existir bastante conteúdo de fácil acesso na internet para apoiar os desenvolvedores. Além disso, são ferramentas que atendem os requisitos apresentados na arquitetura de solução. Existem várias opções de ferramentas que podem ser utilizadas, não estando limitadas somente as que foram utilizadas nesta dissertação.

A Figura 13 apresenta as ferramentas escolhidas para implementação do protótipo de acordo com a arquitetura de solução proposta. Estas ferramentas são listadas a seguir:

- **Fonte de Dados**: Servidor de Arquivos;
- **Plataforma de Processamento de *Streams***: Apache Kafka;
- **Software de Reconhecimento de Fala**: CMU Sphinx;
- **Ferramenta de Indexação**: Elasticsearch;
- **Interface de Pesquisa**: Kibana e Aplicação desenvolvida em HTML e JavasScript.

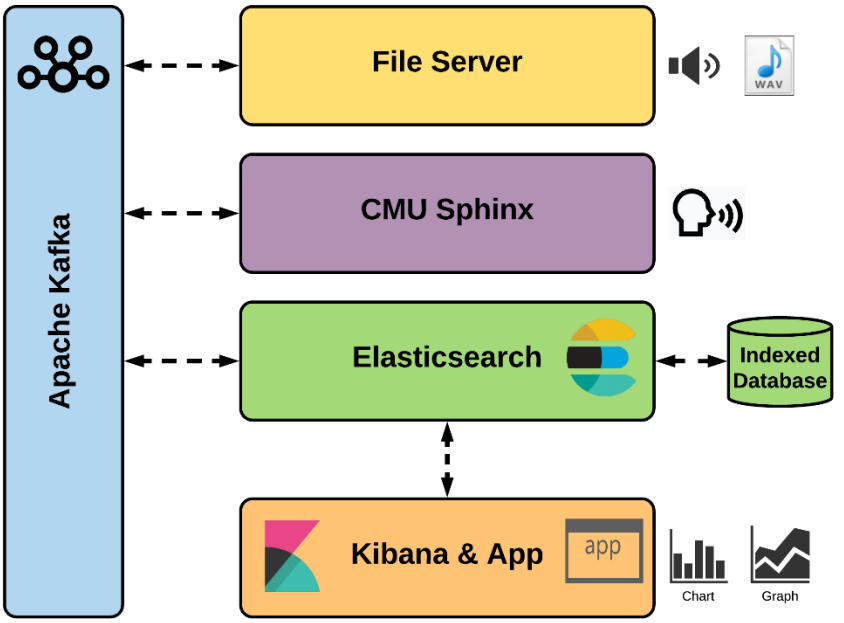

Figura 13. Implementação da Arquitetura.

Uma vantagem das ferramentas escolhidas é que elas podem ser configuradas e implementadas tanto em maquinas Windows e/ou Linux. Além de serem ferramentas de livre uso, não envolvendo custos adicionais para sua implementação.

Para a redução dos custos e facilitar a implementação e a execução dos testes deste protótipo, foram criadas máquinas virtuais Linux em nuvem. Máquinas virtuais criadas em nuvem são de fácil configuração, tem baixo custo, permitem escalar horizontalmente e verticalmente se necessário e, além disso, é possível automatizar sua criação para atender a qualquer necessidade de configuração de ambiente.

As máquinas virtuais removem a necessidade de ter uma máquina física local para o desenvolvimento e execução do protótipo. As plataformas em nuvem foram criadas no Google. O objetivo através de seu uso é demonstrar o poder computacional que a arquitetura pode atingir.

O código foi desenvolvido em JAVA, e foi utilizado o ambiente de desenvolvimento integrado (IDE) do Eclipse. O Apache Maven, ferramenta para o gerenciamento de dependências de bibliotecas, foi utilizado para gerenciar a biblioteca do software de reconhecimento de fala, o CMU Sphinx.

As ferramentas escolhidas para a implementação do protótipo estão descritas a seguir.

4.1. Fonte de dados

Um **servidor de arquivos** contendo arquivos de áudios no formato Waveform Audio File (WAV) foi utilizado como fonte de dados. Os arquivos estão em um diretório compartilhado que aponta para um armazenamento local em um servidor de arquivos. O diretório compartilhado é de fácil configuração e pode ser subdividido em subdiretórios de modo a distribuir os arquivos para serem processados por diferentes produtores se necessário.

O servidor de arquivos utilizado não precisa de nenhuma configuração especifica. Os requisitos exigidos são uma conexão com a rede onde o protótipo está rodando e ter espaço em disco suficiente para alocar os arquivos a serem processados. A **Tabela 2** apresenta as especificações técnicas da máquina utilizada no protótipo.

Tabela 2. Detalhes da Máquina 01 - Servidor de Arquivos e Produtor

Componente	Configuração
Tipo de Máquina	N1-standard-2
Sistema Operacional	Linux Ubuntu 18.04
CPU	Intel ® Xeon® CPU @ 2.30GHz
vCPUs	2
Memória	7,5 GB
Tamanho em Disco	30 GB
Zona	us-central1-a

Os arquivos utilizados estão no formato WAV. A escolha do formato se deve a sua compatibilidade com o software de reconhecimento de fala utilizado. Isto facilita a implementação do protótipo, pois não é necessária a implementação de um conversor de formatos nos produtores.

A Tabela 3 apresenta os detalhes da fonte de dados processada pelo protótipo. O arquivo de áudio utilizado para processamento tem aproximadamente 22 segundos de duração, com um tamanho total de 700 KB e contém 59 palavras de uma conversa entre dois nativos norte-americanos. Para este protótipo foram utilizados 16364 copias deste arquivo, totalizando aproximadamente 100 horas de áudio a serem processadas. O

arquivo foi escolhido por conter uma boa qualidade de gravação e ser um arquivo de uma aula de Inglês, para este trabalho foi escolhido processar 100 horas de áudios para coletar os resultados.

Tabela 3. Detalhes dos Arquivos de Áudios

Detalhe	Valor
Duração de 1 arquivo	22 segundos
Quantidade de palavras	59 palavras
Tamanho de 1 arquivo	700 KB
Quantidade total de arquivos	16364
Duração total de arquivos	~100 horas
Quantidade total de palavras	965.476 palavras
Tamanho total dos arquivos	10.9 GB
Formato dos arquivos	Waveform Audio File (WAV)

4.2. Plataforma de Processamento de *Streams*

O **Apache Kafka** versão 2.12-2.2.0 é a ferramenta escolhida para o processamento das *streams* de dados. Apache Kafka é um sistema de eventos *publish-subscribe*, onde os arquivos de áudios e vídeos são considerados como *streams*/mensagens que são persistidos em uma fila através de um produtor (*publisher*). Consumidores (*subscriber*) podem se inscrever a uma ou mais filas e consumir todas as *streams*/mensagens destas filas em paralelo.

O Apache Kafka é responsável por receber os arquivos de áudios e vídeos de diferentes fontes de dados e uni-los em uma única fonte de dados distribuída. Além disso, ele tem a responsabilidade de encaminhar os arquivos de áudio, quando requisitado, para diferentes instâncias do software de reconhecimento de fala que produzirão a lista de palavras faladas em cada um dos arquivos.

O fluxo básico do Apache Kafka pode ser resumido em três principais ações:

- **Produzir** uma mensagem;
- **Anexar** essa mensagem em um tópico;

- **Consumir** essa mensagem de um tópico.

A **Figura 14** apresenta as três principais ações do fluxo básico do Apache Kafka.

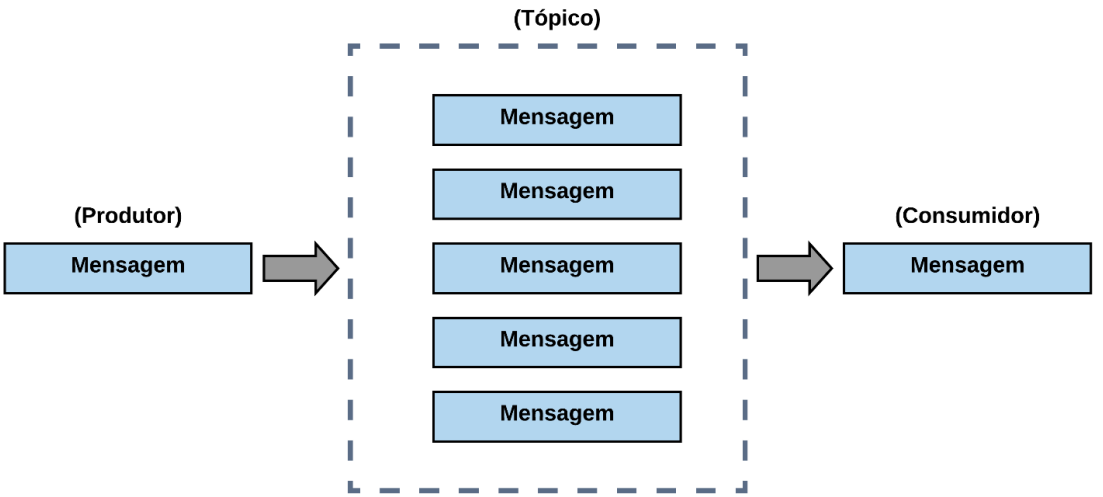

Figura 14. Fluxo Básico do Apache Kafka

4.2.1. Mensagem

A **mensagem** é o stream que está sendo processado e transferido pelo sistema. Neste protótipo a mensagem é um conteúdo em JSON com informações do arquivo e o arquivo de áudio no formato binário. As informações dos arquivos são nome, endereço físico, tamanho, dono do arquivo, hora de processamento, identificador único universal que permite rastrear o arquivo durante o processamento.

Uma mensagem que é enviada para o Apache Kafka pode ser definida por um valor composto por chave/valor, na chave foi definido o conteúdo em JSON contendo as informações do arquivo e no valor contém o arquivo no formato binário.

Com o objetivo de salvar mais detalhes sobre o arquivo processado para extrair métricas, foi criada uma estrutura em JSON salva na chave. O formato da estrutura é descrito a seguir:

- **fileName:** O nome completo do arquivo salvo no diretório.
- **filePath:** Caminho completo onde o arquivo se encontra.
- **creationTime:** Data de criação do arquivo na máquina.
- **size:** Tamanho total em bytes do arquivo.

- **owner:** Proprietário do arquivo.
- **UUID:** Identificador único universal do arquivo, este valor é utilizado para rastrear o arquivo e gerar métricas durante o processamento do arquivo até o destino final. Como o sistema é distribuído, o processamento completo do arquivo pode ocorrer em mais de uma máquina. Este valor é usado para gerar arquivos de logs contendo o tempo de duração de uma determinada tarefa, por exemplo, duração de conversão do arquivo para binário, duração de envio do conteúdo para o servidor do Apache Kafka, entre outros.

A **Tabela 4** apresenta o formato da mensagem composta pela chave/valor.

Tabela 4. Formato da Mensagem.

Chave	Valor
{ "fileName": "audio.wav", "filePath": "C:\\Audio\\Aulas\\audio.wav", "creationTime": "2020-04-03T00:47:35.672434Z", "size": 716460, "owner": "Company-PC\\michelbatista", "UUID": "d592c687-8f2b-44f8-85ed-c138931f33fb" }	Arquivo WAV convertido em formato binário.

4.2.2. Produtor

O **produtor** é um executável desenvolvido em JAVA que utiliza as bibliotecas nativas do Apache Kafka. Estas bibliotecas permitem que o sistema estabeleça conexão e consiga enviar comandos para o Apache Kafka. O produtor está rodando diretamente no servidor de arquivos. É o responsável por pegar todos os arquivos no formato WAV de um determinado diretório, coletar e salvar as informações do arquivo em uma estrutura em JSON, converter o arquivo para o formato binário para que possa ser trafegado e por fim enviar o conteúdo para um tópico específico do Apache Kafka.

Neste produtor podemos realizar alguns tratamentos nos arquivos se necessário. Podemos citar, por exemplo, extrair o áudio de um vídeo ou aplicar algum filtro para melhorar a qualidade do conteúdo a ser extraído, ou conectar em alguma outra fonte de dados.

O produtor realiza quatro principais ações:

1. Coletar informações do arquivo de áudio e salvar em uma estrutura em JSON.
2. Converter o arquivo WAV para binário.
3. Enviar o conteúdo e as informações para um tópico no Apache Kafka.
4. Gerar logs de arquivos com o tempo de duração de execução, para converter o arquivo para binário e para enviar o conteúdo para o tópico do Apache Kafka.

Os 16364 arquivos de áudio foram processados em uma instância do produtor, rodando no mesmo servidor dos arquivos. Com isto, o produtor não depende de uma rede para recuperar os arquivos e será mais rápido do que em um modelo onde o produtor precise recuperar os arquivos através da rede.

No produtor desenvolvido para processar arquivos de um servidor de arquivos é possível configurar algumas variáveis no arquivo de configuração, descritas a seguir:

- **kafka-broker**: Configura o endereço para conectar no Apache Kafka;
- **kafka-topic**: Configura o nome do tópico do Apache Kafka para enviar as mensagens;
- **windows.statistics.path** ou **linux.statistics.path**: Configura o endereço do diretório para salvar os logs das estatísticas com o tempo de processamento;
- **windows.file.path** ou **linux.file.path**: Configura o endereço onde os arquivos de áudios estão para serem processados.

4.2.3. Tópico

O **tópico** fica no servidor do Apache Kafka e é responsável por agrupar as mensagens recebidas pelo produtor, ordenar essas mensagens por ordem de chegada e distribuí-las para os consumidores que estão requisitando mensagens para processar. Um tópico mantém as mensagens ordenadas e as organiza em diferentes partições.

Um tópico pode conter uma quantidade qualquer de **partições** para receber as mensagens. O Apache Kafka distribui as mensagens recebidas nestas partições, que por sua vez as divide entre os consumidores que estão assinados para consumir as mensagens deste tópico. Portanto, utilizar mais partições significa um maior número de paralelização e uma maior taxa de transferência de dados a serem processados.

O Apache Kafka permite escalar horizontalmente o processamento através de um conceito chamado de grupo de consumidores; os consumidores agrupados consomem as mensagens de um mesmo tópico. Desta maneira é possível agrupar vários consumidores para processar todas as mensagens de um tópico em paralelo. Além disso garantir que uma mensagem será processada somente uma única vez.

Para este protótipo foi criado um único tópico com 150 partições. Portanto, 16364 arquivos de áudios foram processados por um único produtor. O Apache Kafka distribuiu as mensagens aleatoriamente nestas 150 partições para serem processadas por um grupo de 100 consumidores em paralelo. Como 100 consumidores foram criados para realizar o processamento, o número de partições criadas foi maior para se caso algum consumidor terminar o seu processamento ele possa pegar arquivos de uma outra partição.

O ***broker*** é a ferramenta do Apache Kafka que gerencia e organiza as partições dos tópicos. Ele é responsável por salvar as mensagens em cada partição, manter sua ordenação como também ligar as partições nos consumidores. Para este protótipo apenas uma instância de Apache Kafka foi necessária, ou seja, apenas um *broker* está trabalhando para gerenciar o tópico criado.

Cada servidor de Apache Kafka contém um *broker*, e se houver a necessidade de escalar o sistema e/ou replicar as partições para evitar perda de dados, basta rodar outra instância do Apache Kafka e configura-las para rodarem em um cluster. Desta forma dois *brokers* estariam gerenciando todos os tópicos e as partições existentes.

4.2.4. Consumidor

Consumidor é um executável desenvolvido em JAVA que utiliza a biblioteca do Apache Kafka que pode ser executado em uma máquina qualquer de processamento. O consumidor é responsável por conectar no servidor Apache Kafka, requisitar as mensagens, realizar o processamento dos arquivos de áudios, utilizar o software de reconhecimento de fala e enviar o resultado para o Elasticsearch que é a base de dados que vai indexar os dados.

O consumidor realiza quatro principais ações:

1. Conecta no Apache Kafka e requisita uma mensagem, converte a mensagem recebida para uma estrutura em JSON contendo as informações do arquivo e o binário em um arquivo de áudio em memória.
2. Realiza o processamento do arquivo de áudio utilizando o software de reconhecimento de fala para extrair o texto falado do arquivo.
    - Estrutura o texto extraído em um formato JSON que é o formato recebido pelo Elasticsearch.
    - Remove as *stop words* para salvar somente o conteúdo necessário para facilitar a busca das palavras, reduzir o espaço consumido e organizar os dados.
3. Envia a estrutura em JSON para o Elasticsearch (ferramenta de indexação dos dados).
4. Gera logs de estatísticas com o tempo de execução para extrair o texto do áudio e envia os dados para o Elasticsearch.

Os consumidores não precisam se preocupar em gerenciar a fila de processamento ou dividir o processamento entre outros consumidores. Ele apenas requisita para o Apache Kafka uma mensagem para ser processada e o *broker* do Apache Kafka se encarrega de enviar a mensagem correta.

No momento da criação da conexão com o *broker* do Apache Kafka é possível alterar alguns parâmetros de configuração para poder especificar como a conexão deve se comportar. Na Tabela 5, estão detalhadas as principais configurações implementadas neste protótipo.

Tabela 5. Principais Configurações de Conexão com o Apache Kafka.

Configuração	Valor
group.id	mestrado.projeto.processor
enable.auto.commit	True
key.deserializer	org.apache.kafka.common.serialization.StringDeserializer
value.deserializer	org.apache.kafka.common.serialization.ByteArrayDeserializer
auto.offset.reset	Earliest
fetch.message.max.bytes	7340032

max.poll.records	1

Esta configuração faz com que o consumidor conecte no Apache Kafka e requisite somente uma única mensagem para processar e desconecte do Apache Kafka liberando a partição para outro consumidor se necessário.

- ***group.id***: Este é o valor que o Apache Kafka utiliza para agrupar os consumidores, todos os consumidores que tiverem o mesmo agrupador de consumidor vão processar todas as mensagens uma única vez, evitando que a mesma mensagem seja processada por mais de um consumidor. Para este protótipo todos os consumidores utilizam o mesmo valor de group.id para poder agrupar e dividir as mensagens entre os consumidores.
- ***enable.auto.commit***: Este valor diz para o Apache Kafka se a mensagem pode ser considerada processada assim que o consumidor receber a mensagem, caso esteja configurada como "True". Existem casos onde o processamento pode falhar e a mensagem deverá ser processada por outro consumidor; o valor configurado deve ser "False". O protótipo está configurado como "True" o que permite marcar a mensagem como processada assim que ela for recebida pelo consumidor. Desta forma, a partição fica liberada para ser conectada em outro consumidor e processar a próxima mensagem. O processamento do arquivo pode demorar alguns segundos ou até mesmo alguns minutos, e isso pode impedir o processamento da próxima mensagem da partição até que a mensagem que está em processamento seja completamente processada, fazendo com que alguns consumidores possam ficar ociosos. Em casos de acontecer algum problema o protótipo realizar 3 tentativas de reprocessamento do arquivo antes de registrar algum erro.
- ***key.deserializer***: Configuração do formato de dado da chave enviada para o Apache Kafka. Neste protótipo é enviado um texto no formato JSON, então o valor configurado é *StringDeserializer*.
- ***value.deserializer***: Configuração do formato de dado do valor enviado para o Apache Kafka. Neste protótipo, o arquivo de áudio é convertido para binário, então o valor configurado é *ByteArrayDeserializer*.

- ***auto.offset.reset***: Está configuração diz se ao conectar a uma partição do Apache Kafka o consumidor deve receber os dados do início da partição sem considerar se os dados já foram processados ou se deve pegar o último valor a ser processado. Este protótipo está configurado para recuperar sempre a última mensagem a ser processada.
- ***fetch.message.max.bytes***: Configura o tamanho máximo de uma mensagem que pode ser buscada pelo consumidor. Para este protótipo foi configurado até aproximadamente 7 GB.
- ***max.poll.records***: Configura o número máximo de mensagens retornadas em uma chamada do consumidor para o Apache Kafka. Este protótipo está processando uma mensagem por vez.

Assim que o consumidor recebe a mensagem do Apache Kafka contendo a chave e o valor, esse conteúdo é convertido para os formatos conhecidos para o processamento.

A chave é convertida em um objeto no formato JSON para poder recuperar as informações do arquivo de áudio e salvá-las na estrutura final, que por sua vez será enviada para o banco que indexa as informações no Elasticsearch.

O valor, o áudio em binário, é convertido para o formato WAV. Após esse processo, este arquivo é processado pelo software de reconhecimento de fala e o resultado do processamento é tratado para remover algumas palavras que não são normalmente utilizadas para realizar uma busca por conteúdo (*stop words*). Ambas as técnicas de processamento do arquivo e tratamento do resultado estão implementadas dentro do consumidor.

Após o processo de recuperar as informações e processar o conteúdo, o resultado do processamento é organizado em uma estrutura JSON para ser enviado para o Elasticsearch.

No consumidor desenvolvido também é preciso configurar algumas variáveis no arquivo de configuração de modo a ter o software devidamente configurado e em funcionamento, descritas a seguir:

- ***kafka-broker***: Configura o endereço para conectar no Apache Kafka;

- ***kafka-topic***: Configura o nome do tópico do Apache Kafka que vai receber as mensagens;
- ***windows.statistics.path*** ou ***linux.statistics.path***: Configura o endereço do diretório para salvar os arquivos logs para coletar estatísticas de tempo de execução;
- ***windows.acoustic.model.path*** ou ***linux.acoustic.model.path***: Endereço do diretório onde o modelo acústico está salvo;
- ***windows.lexicon.model.path*** ou ***linux.lexicon.model.path***: Endereço do diretório onde o modelo léxico está salvo.
- ***windows.language.model.path*** ou ***linux.language.model.path***: Endereço do diretório onde o modelo de linguagem está salvo.
- ***elasticsearch.url***: Configura o endereço para conectar no Elasticsearch.
- ***elasticsearch.port***: Configura a porta para conectar no Elasticsearch.
- ***elasticsearch.protocol***: Configura o protocolo de comunicação com o Elasticsearch HTTP ou HTTPS.
- ***elasticsearch.index***: Configura o índice onde os dados extraídos serão indexados no Elasticsearch.

4.2.5. Apache Zookeeper

O Apache Zookeeper é um serviço centralizado para manter informações de configurações, sincronização e nomenclaturas entre serviços distribuídos (Apache Zookeeper 2020). O Apache Kafka utiliza o Apache Zookeeper para sincronizar as configurações entre diferentes clusters.

Para rodar o Apache Kafka é preciso rodar também o Apache Zookeeper para gerenciar o Apache Kafka. O Apache Zookeeper controla o status dos nós do cluster do Apache Kafka e também acompanha os tópicos e partições.

O Apache Zookeeper garante a alta disponibilidade do cluster do Apache Kafka. Caso algum nó do cluster caia ou fique indisponível, o Apache Zookeeper garante que os outros nós estejam funcionando e replicando os dados. Sua função também é garantir que sempre tenha um nó mestre no cluster, e caso ele caia outro seja eleito como o nó mestre.

Algumas informações importantes como quantas mensagens cada consumidor consome, local das partições e as configurações dos tópicos também podem ser salvas no Apache Zookeeper. Ele também garante que o estado do Apache Kafka não seja perdido em caso de falhas.

Para este propósito não é necessária nenhuma configuração extra de cluster, pois é usada uma instância do Apache Kafka.

4.2.6. Organização Geral

A Figura 15 mostra a organização da estrutura do Apache Kafka em relação ao produtor, consumidores, tópico e partições.

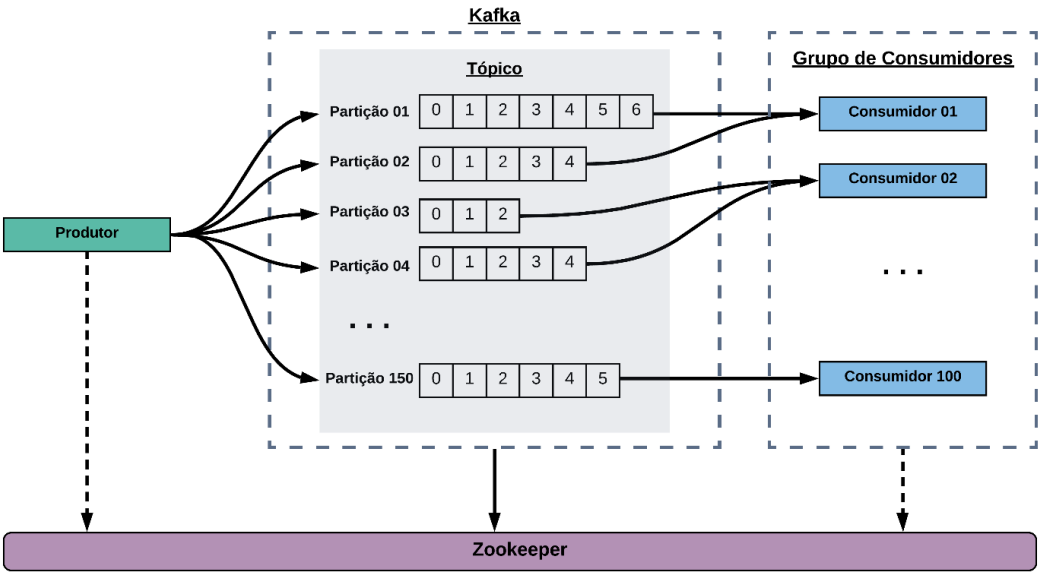

Figura 15. Configuração do Ambiente do Apache Kafka.

Portanto, para a implementação do protótipo foi necessária uma única instância do produtor, 100 instâncias do consumidor, uma instância do Apache Kafka e apenas um tópico para receber todos os arquivos de áudio em formato de mensagens para ser processados.

Neste protótipo, os consumidores realizam o processamento das mensagens do Apache Kafka em 20 máquinas. Em cada máquina foram executadas cinco instâncias do consumidor, desta forma foram criadas 100 instâncias de consumidores para realizar o processamento das mensagens que estão nas 150 partições do tópico criado.

O Apache Kafka é responsável por dividir as 150 partições entre as 100 instâncias de consumidores e garantir que as mensagens sejam processadas. Ele também é responsável por realocar as partições do tópico para outros consumidores caso algum consumidor fique indisponível ou caso algum novo consumidor entre em execução.

O Apache Zookeeper roda para suportar o Apache Kafka salvando o status e informações de tópicos, partições e mensagens processadas.

Para visualizar todos os comandos utilizados no Apache Kafka e suas descrições, verificar o Apêndice I – Script para a Configuração do Apache Kafka.

4.3. Software de Reconhecimento de Fala

O **CMU Sphinx** versão 5 é uma ferramenta de código aberto para o reconhecimento de fala. A escolha do software foi baseada nos resultados obtidos em outros trabalhos acadêmicos: Gaida *et al*. (Gaida, *et al.* 2014) e Matarneh *et al*. (Matarneh, *et al.* 2017), em que o CMU Sphinx é utilizado. Os testes preliminares foram realizados com alguns arquivos e foi obtido um índice de pelo menos 72% de acerto na extração dos textos.

O CMU Sphinx é um reconhecedor de fala por comparação de padrões. É uma ferramenta de fácil utilização.

4.3.1. Implementação e Execução do CMU Sphinx

O **Primeiro passo** para criar o consumidor é criar um projeto em JAVA e exportar as bibliotecas do CMU Sphinx. Neste protótipo, foi utilizada a ferramenta de desenvolvimento Eclipse para o desenvolvimento, e para gerenciar e organizar as bibliotecas externas foi utilizado o Maven no Eclipse.

Ao criar o projeto Maven no Eclipse basta editar o arquivo de configuração do Maven “pom.xml” e incluir as seguintes dependências necessárias para utilização do CMU Sphinx. A Tabela 6 mostra a configuração da dependência necessária.

Tabela 6. Configuração de Dependência do CMU Sphinx.

```
<!-- CMU Sphinx dependência -->
<dependency>
  <groupId>edu.cmu.sphinx</groupId>
  <artifactId>sphinx4-core</artifactId>
  <version>5prealpha-SNAPSHOT</version>
  <scope>compile</scope>
</dependency>

<!-- CMU Sphinx dependência para usar os modelos acústico e de linguagem
padrões em Inglês US-->
<dependency>
  <groupId>edu.cmu.sphinx</groupId>
  <artifactId>sphinx4-data</artifactId>
  <version>5prealpha-SNAPSHOT</version>
  <scope>compile</scope>
</dependency>
```

Segundo passo: Configurar o sistema para dizer quais serão os modelos utilizados para realizar o reconhecimento da fala; o modelo acústico, modelo léxico e o modelo de linguagem.

O projeto CMU Sphinx já vem com vários modelos acústicos de alta qualidade que podem ser utilizados. Entretanto, estes modelos são para o Inglês. Todos os modelos utilizados neste protótipo são fornecidos pela comunidade do CMU Sphinx, e são estes modelos que definem o percentual de acerto das palavras reconhecidas.

Portanto, para melhores resultados é preciso treinar mais o sistema e utilizar arquivos de áudio com diferentes locutores. Esta implementação utiliza os modelos padrões já fornecidos e o código necessário para a configuração dos modelos está descrito na Tabela 7.

Tabela 7. Código para Configuração dos Modelos.

```
Configuration config = new Configuration();

// Configura o caminho para o modelo acústico.
config.setAcousticModelPath("file:/cmusphinx/models/en-us/en-us");

// Configura o caminho para o modelo léxico.
config.setDictionaryPath("file:/cmusphinx/models/en-us/cmudict-en-
```

```
us.dict");

// Configura o caminho para o modelo de linguagem.
config.setLanguageModelPath("file:/cmusphinx/models/en-us/en-
us.lm.bin");
```

Terceiro passo: Garantir o formato do arquivo que será decodificado pelo CMU Sphinx, que deve ser um áudio no formato WAV de 16 bits e 16 KHz ou 8 KHz. A Tabela 8 apresenta o formato completo requerido pelo software.

Tabela 8. Formatos de Áudio Aceito pelo CMU Sphinx.

Formatos de Áudio Aceitos pelo CMU Sphinx
RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 16000 Hz
RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 8000 Hz

Para este protótipo os arquivos já estão no formato necessário para o processamento. Caso estejam em outro formato será preciso realizar uma conversão de formato de arquivo para um dos formatos aceitos pelo CMU Sphinx.

Quarto e último passo: Executar o sistema para decodificar o arquivo de áudio e extrair o conteúdo falado. A Tabela 9 apresenta o código necessário para este fim (Código apresentado apenas para demonstra a facilidade de utilização da ferramenta).

Tabela 9. Código para Decodificar o Áudio.

```
StreamSpeechRecognizer recognizer = new
StreamSpeechRecognizer(config);

recognizer.startRecognition(new
FileInputStream("ArquivoDeAudio.wav"));

SpeechResult result = null;

While(result = recognizer.getResult() != null){

   // Palavras reconhecidas.

}

recognizer.stopRecognition();
```

A variável do tipo *SpeechResult* provê acesso a várias partes do resultado do reconhecimento. Podemos citar, por exemplo, uma sentença completa contendo várias

palavras reconhecidas ou uma lista de palavras com o horário de início e fim de suas ocorrências que é exatamente o objetivo desta implementação.

À medida em que o software decodifica o arquivo e reconhece as palavras, o protótipo salva as palavras em uma lista de estruturas contendo a palavra identificada, uma lista contendo o horário de início da ocorrência e a quantidade de ocorrências.

Esta estrutura reduz o espaço em memória necessário para salvar todas as ocorrências das palavras. Não precisa salvar a palavra a cada vez que ela for encontrada, e também não é preciso saber o horário de término da pronúncia da palavra, basta saber onde ela começa. A Tabela 10 apresenta um exemplo do resultado da estrutura descrita após o reconhecimento de um áudio de teste de cerca de 4 segundos contendo a seguinte frase "*Test, this is a test example*".

Tabela 10. Exemplo da Estrutura com as Palavras Reconhecidas.

Palavra	Quantidade de Ocorrências	Ocorrências
test	2	00:00:00.766, 00:00:02.975
this	1	00:00:01.869
is	1	00:00:02.380
a	1	00:00:02.758
example	1	00:00:03.397

Como resultado 6 palavras são encontradas e uma delas contém duas ocorrências, então o resultado da decodificação é organizado e estruturado como apresentado na Tabela 10, contendo uma lista com a quantidade de ocorrência da palavra encontrada no arquivo e os horários da ocorrência no seguinte formato ***horas:minutos:segundos.milissegundos***.

O processo de decodificação pelo CMU Sphinx utiliza bastante recursos da máquina, como CPU e memória, para realizar o reconhecimento dos padrões, cálculos estatísticos e encontrar a palavra correta. Durante os testes preliminares, ao utilizar os modelos padrões, cada execução de consumidor utilizando o CMU Sphinx durante a decodificação consumiu em média de 1 GB a 1,5 GB de memória do computador.

O resultado dos testes preliminares foi o que resultou na configuração do ambiente com no máximo 5 instâncias de consumidores por máquina. Mais detalhes sobre os testes estão no capítulo de análise e resultados.

4.3.2. Remoção das *Stop Words*

Após o processo de decodificação e ter a estrutura contendo todas as palavras reconhecidas, o protótipo remove todas as *stop words*, que são aquelas palavras não relevantes para as consultas e que estão presentes em praticamente todos os documentos. Remover estas palavras ajuda a limpar a base de dados a ser indexada e economizar espaço utilizado, e desta forma ajuda a otimizar as buscas.

Para a implementação deste protótipo foi utilizado um arquivo de texto (txt), contendo uma lista com 127 *stop words* em inglês, que é carregado para a memória durante a execução do consumidor. Após todas as palavras serem reconhecidas pelo software de reconhecimento o consumidor remove todas as *stop words* com base no arquivo de texto.

A Tabela 11 apresenta o resultado das palavras encontradas após a remoção das *stop words*, utilizando o mesmo arquivo usado no resultado da Tabela 10. O resultado foi que apenas 2 palavras foram encontradas com 3 ocorrências no total.

Tabela 11. Resultado Após a Remoção das Palavras de Parada.

Palavra	Quantidade de Ocorrências	Ocorrências
test	2	00:00:00.766, 00:00:02.975
example	1	00:00:03.397

Caso seja preciso incluir ou remover alguma *stop word* basta editar o arquivo de texto. Alguns exemplos de *stop words* em inglês estão listados a seguir:

- *Have*
- *Here*
- *I*
- *Me*
- *Should*
- *This*

4.4. Indexação

A funcionalidade de indexação no protótipo é fornecida pelo **Elasticsearch**, uma ferramenta que permite indexar os documentos e realizar buscas em uma grande quantidade de dados em tempo real. O Elasticsearch é escalável e particularmente eficiente para fazer pesquisas textuais em grandes volumes de dados. O Elasticsearch é *open source* e tem uma grande comunidade de discussão.

O Elasticsearch faz parte de um conjunto de ferramentas *open source*, ELK (Elasticsearch, Logstash e Kibana), para ingestão, armazenamento, análise e visualização de dados.

Está ferramenta ainda conta com APIs REST que são leves e simples, permitem gerenciar os dados e realizar buscas complexas. Estas APIs facilitam a integração para que outros sistemas possam conectar no Elasticsearch.

Para a implementação deste protótipo foi necessário uma única instância do Elasticsearch na versão 7.6.2, além de algumas configurações para preparar a base para que a ingestão de dados seja feita da melhor forma e mais otimizada possível. A ingestão de dados é o processo onde os dados brutos são analisados, normalizados e enriquecidos antes de serem indexados no Elasticsearch. Estes passos são executados pelo próprio Elasticsearch a partir da configuração do ambiente.

Após a execução do Elasticsearch todas as configurações e consultas são realizadas através das APIs REST disponibilizadas pelo sistema: criação de um índice, mapeamento da definição da estrutura dos documentos que serão salvos, queries para recuperar o conteúdo salvo.

4.4.1. Estrutura do Elasticsearch

O Elasticsearch é orientado a documentos no qual ele armazena objetos ou documentos inteiros. Esta ferramenta também utiliza a técnica de lista invertida para salvar, organizar os documentos salvos e classificar e filtrar os documentos e os dados.

Os dados são indexados no formato JSON. Cada documento salvo pertence a um tipo, que por sua vez pertencem a um índice. A Tabela 12 apresenta um comparativo de estrutura entre um banco de dados relacional tradicional e o Elasticsearch.

Tabela 12. Comparativo de Estrutura de um Banco de Dados Relacional vs Elasticsearch

Banco de dados relacional	Elasticsearch
Databases (Banco de dados)	Indexes (Índices)
Tables (Tabelas)	Types (Tipos)
Rows (Linhas)	Documents (Documentos)
Columns (Colunas)	Fields (Campos)

Este comparativo apresentado é apenas para facilitar o entendimento dos conceitos. Nas versões mais novas, acima da versão 7.0.0, o mapeamento dos tipos foi removido, devido a que o tipo do Elasticsearch seria equivalente a uma tabela do banco de dados relacional. Entretanto, dentro de um mesmo índice não é permitido ter duas colunas com o mesmo nome em tipos diferentes, pois internamente o Elasticsearch trata ambos como o mesmo campo e tipo.

Para este protótipo foi criada a seguinte estrutura para armazenar os resultados extraídos dos arquivos de áudio:

- Um documento é indexado por arquivo de áudio processado.
- Cada documento é do tipo "_doc" e recebe um identificador.
- Os documentos estão salvos em um único índice chamado audio-index.
- Os tipos dos campos deste documento estão mapeados para facilitar a busca e funções de agregações.

4.4.2. Criação do Índice

Um único índice chamado audio-index foi criado e armazenado em um único *shard* (o *shard* representa a quantidade de nós em execução do Elasticsearch em um servidor ou em um cluster de servidores, para poder distribuir os dados armazenados). Este índice é responsável por armazenar todo o conteúdo extraído em texto dos arquivos de áudio.

O ponto de partida é ter um índice criado para armazenar os dados. Um índice é apenas um espaço lógico que aponta a um ou mais *shards* físicos. *Shards* são unidades de trabalho de baixo nível que contêm uma parte de todos os dados do índice.

Um *shard* é uma instância de um índice do Apache Lucene Index que é a base usada no Elasticsearch. O Apache Lucene Index é um mecanismo de pesquisa, que indexa e manipula as consultas em um cluster do Elasticsearch. Em um ambiente de cluster é possível ter vários *shards* onde os dados do índice poderão estar distribuídos e replicados se necessário, para este protótipo somente um servidor do Elasticsearch foi necessário para testar o protótipo.

4.4.3. Mapeamento dos Campos do Documento

Os campos do documento no formato JSON são compostos por chave e valor. A chave é o nome do campo ou da propriedade. O valor é o conteúdo que pode ser de vários formatos, por exemplo, texto, número, booleano, outro objeto, uma coleção de valores ou até mesmo um tipo especializado representando uma data ou uma geolocalização.

O Elasticsearch gera automaticamente um mapeamento dos campos do documento com base no documento que é salvo no índice. Entretanto, para esta implementação já existe uma definição dos campos do documento, o que cada campo representa e qual o seu tipo de dado. Desta forma, foi configurado o mapeamento dos campos para este índice com o objetivo de configurar cada campo com o tipo correto de dado para facilitar a execução das pesquisas e a utilização de funções de agregações como soma ou cálculo de média.

Os seguintes tipos e propriedades de dados foram utilizados no documento:

- ***Text***: Para indexar valores de texto completo.
- ***Integer***: Tipo inteiro entre os valores $-2^{31}-1$ e $2^{31}-1$.
- ***Date***: Tipo data e o seu padrão no Elasticsearch é definido como yyyy-mm-ddThh:MM:ss, exemplo 2021-03-17T15:30:55.

- ***Keyword***: Para indexar um conteúdo estruturado, como Ids, endereços de email, nome de hosts, códigos de status ou tags e etc. Este tipo de campo é filtrado apenas pelo valor exato.
- ***Object***: Um objeto que pode conter objetos internos.
- ***Nested***: Um tipo aninhado, versão do tipo *object*. Permite que uma coleção de objetos seja indexada e permita ser consultada independentemente uma da outra.
- ***Fielddata***: Atributo adicional utilizado quando classificações ou agregações podem ser utilizadas no campo, por exemplo, realizar uma soma ou média de valores.

A estrutura do documento criado para este protótipo é composta por três partes:

1. A primeira parte é composta por um campo data para saber a hora que o documento entrou na base de dados.
2. A segunda parte é um objeto contendo informações do arquivo como, nome do arquivo (*fileName*), caminho do arquivo (*filePath*), data de criação (*creationTime*), tamanho em bytes (*size*), proprietário (*owner*), tipo do arquivo áudio ou vídeo (*fileType*), tag que é um valor que pode ser configurado no consumidor para agrupar os arquivos (*tag*) e duração em segundos (duration).
3. A terceira parte é composta por uma coleção de objetos com a palavra reconhecida (*word*), o horário de sua ocorrência (*occurrences*) e a quantidade de ocorrências no arquivo (*occurrenceQuantity*).

A Tabela 13 apresenta a estrutura do mapeamento do documento em JSON.

Tabela 13. Mapeamento dos Campos do Documento.

Mapeamento dos Campos em JSON

```
{
   "properties": {
      "time": { "type": "date" },
      "fileAttributes": {
         "type": "object",
         "properties": {
            "fileName": { "type": "text", "fielddata": true },
            "filePath": { "type": "text" },
            "creationTime": { "type": "text" },
            "size": { "type": "integer" },
            "owner": { "type": "keyword" },
            "uuid": { "type": "keyword" },
            "fileType": { "type": "keyword" },
```

```
            "tag": { "type": "keyword" },
            "duration": { "type": "integer" }
         }
      },
      "words": {
         "type": "nested",
         "properties": {
            "word": { "type": "keyword" },
            "occurrences": {  "type": "text", "fielddata": true },
            "occurrenceQuantity": { "type": "integer" }
         }
      }
   }
}
```

A Tabela 14 apresenta um exemplo de um documento que é salvo no Elasticsearch.

Tabela 14. Exemplo da Estrutura do Documento Salvo no Elasticsearch.

Exemplo da Estrutura do Documento Salvo no Elasticsearch

```
{
   "time": "2020-04-20T10:30:00",
   "fileAttributes": {
      "fileName": "test.wav",
      "filePath": "C:\\Reportagens\\audios\\reportagem001.wav",
      "creationTime": "2019-09-24T00:31:53.141187Z",
      "size": 716460,
      "owner": "michel-batista\\pessoal",
      "fileType": "audio",
      "tag": "reportagem",
      "duration": "22"
   },
   "words": [
      { "word": "big", "occurrences": [ "00:00:00.180" ],
"occurrenceQuantity": 1 },
      { "word": "show", "occurrences": [ "00:00:00.470", "00:00:07.760" ],
"occurrenceQuantity": 3 },
      { "word": "look", "occurrences": [ "00:00:04.320" ],
"occurrenceQuantity": 1 }
      ...
   ]
}
```

4.4.4. APIs e Queries de Busca

Com o Elasticsearch, o índice e o mapeamento de campos devidamente configurados, tudo está pronto para poder receber os documentos e permitir realizar as consultas.

Existem alguns métodos fornecidos pelo Elasticseach para realizar a comunicação entre qualquer sistema externo e a base de dados indexada. É possível utilizar vários meios para poder interagir com as APIs do Elasticsearch. Por exemplo, pode ser utilizado diretamente a API RESTful do Elasticsearch, uma API do Java, ou usar linhas de comandos diretamente da máquina através do comando CURL (Client URL é uma biblioteca de linha de comando para transferir dados usando vários protocolos).

Este protótipo utiliza a própria API do Elasticsearch e uma API do Java como métodos de comunicação com a base indexada. A API do Java é utilizada nos consumidores para enviar o conteúdo extraído para o Elasticseach. A API do Elasticsearch é chamada através da interface de Pesquisa Kibana e pela biblioteca em JavaScript chamada JQuery. Esta API é utilizada pela interface de pesquisa desenvolvida para realizar as buscas nos conteúdos indexados.

A estrutura da API do Elasticsearch para realizar uma chamada está descrita a seguir:

- **Verb**: Verbo da solicitação para a API.
- **URL: <PROTOCOL>://<HOST:PORT>/<PATH>/?QUERY_STRING**
- **Body**: Corpo da solicitação para API se necessário.

Os valores necessários para chamar a API estão descritos na Tabela 15:

Tabela 15. Itens Necessários para Chamar a API.

Item	Descrição
Verb	Método HTTP: GET, POST, PUT, HEAD ou DELETE
Protocol	Protocolo de comunicação HTTP ou HTTPS
Host	Endereço do cluster do Elasticsearch ou do endereço do nó que está em execução do Elasticseach
Port	Porta que está rodando o Serviço HTTP do Elasticsearch, o valor padrão é 9200
Path	Caminho que contém o método do Elasticsearch e/ou o index
QUERY_STRING	Parâmetros opcionais da URL para realizar uma chamada
Body	Se caso estiver enviando algum documento para o Elasticsearch é preciso preencher o Body com o conteúdo do documento no formato em JSON. Por

	exemplo, para passar o documento a ser indexado no Elasticsearch

4.4.4.1. Criação de Índice e Mapeamento de Campos

Para a preparação do Elasticsearch dois métodos são utilizados para a criação do índice e o mapeamento dos campos. Estes dois métodos são executados através do comando CURL, que podem ser executados a partir de qualquer máquina, contanto que tenha acesso ao endereço do Elasticsearch.

A estrutura do comando utilizado segue o mesmo padrão apresentado na Tabela 15:

```
curl -X<VERB> '<PROTOCOL:PORT>://<HOST>/<PATH>?<QUERY_STRING>' –d '<BODY>'
```

Para o protótipo o índice está sendo criado do modo mais simples, sem passar qualquer comando para criar configurações avançadas. Para criar o índice é preciso utilizar o verbo PUT e passar o nome do índice na URL da API, se o Elasticsearch estiver rodando em um ambiente local. A Tabela 16 apresenta o exemplo da chamada para API para criar um índice chamado audio-index e o retorno de sucesso da chamada.

Tabela 16. Chamada para Criar um Índice e o Retorno da Chamada.

Item	Resultado
Chamada	**curl** -XPUT http://localhost:9200/**audio-index**

Resultado da chamada	{
	"acknowledged": true,
	"shards_acknowledged": true,
	"index": "audio-index"
	}

Para realizar o mapeamento dos campos é preciso fornecer as seguintes informações:

- Utilizar o verbo PUT;
- Fornecer o índice onde os dados serão salvos;
- O parâmetro *_mapping* pois está sendo realizada uma operação de mapeamento;
- O tipo do conteúdo (fornecer o *content-type* do tipo *application/json*), pois está sendo utilizado um arquivo no formato JSON;

- E no corpo da chamada a estrutura do mapeamento em JSON como apresentado na Tabela 13.

A Tabela 17 apresenta a chamada e o retorno da chamada ao criar um mapeamento no índice.

Tabela 17. Chamada para Inserir um Mapeamento de Campos.

Item	Resultado
Chamada	**curl** –XPUT -H "Content-Type: application/json" http://localhost:9200/**audio-index/_mapping** –d {Mapeamento em JSON}

Resultado da chamada	<pre>{
	  "acknowledged": true,
	}</pre>

Os comandos de criação de índice e o de criação de mapeamento são executados diretamente no servidor do Elasticsearch e o deixa preparado para receber os documentos.

4.4.4.2. Envio dos Documentos para o Elasticsearch

Os consumidores utilizam a API nativa em JAVA do Elasticsearch para enviar os documentos para serem indexados. A API do JAVA abstrai qualquer dificuldade de desenvolvimento, facilita a chamada e a interação com o Elasticsearch. Para sua utilização basta referenciar a biblioteca no projeto do Java, instanciar o objeto para criar a comunicação, adicionalmente, fornecer a URL, a porta de comunicação da API do Elasticsearch, o protocolo de comunicação (HTTP ou HTTPS) e o documento no formato em JSON como apresentado na Tabela 14.

4.4.4.3. Consultas no Elasticsearch

Com a base de dados devidamente configurada e com os textos extraídos dos arquivos de áudio processado, o Elasticsearch está pronto para realizar pesquisas na base de dados e procurar pelo conteúdo já processado.

Para esta tarefa este protótipo utiliza uma ferramenta de busca nativa que se comunica com o Elasticsearch. Adicionalmente, utiliza chamadas assíncronas diretamente na API nativa do Elasticsearch utilizando a biblioteca de JavaScript JQuery.

Segundo Gormley e Tong (Gormley e Tong 2015), uma grande vantagem no Elasticsearch é que cada campo no documento pode ser indexado e consultado durante uma consulta. O Elasticsearch pode usar todos os índices existentes para retornar os resultados em uma excelente velocidade, o que normalmente em um banco de dados tradicional não acontece.

A busca mais básica da API é a busca vazia, onde não se especifica nenhuma query de busca e todos os documentos de todos os índices ou de um índice específico são retornados. A Tabela 18 apresenta um exemplo de busca vazia para entender o retorno da chamada, as queries utilizadas neste protótipo seguem a mesma estrutura e a única coisa que difere é a query de busca utilizada para realizar filtros e ordenações.

Tabela 18. Query de Busca Vazia.

Item	Resultado
Chamada	**curl** –XGET http://localhost:9200/**audio-index/_search**

Resultado da chamada	{
	"took": 151,
	"timed_out": false,
	"_shards": { "total": 1, "successful": 1, "skipped": 0, "failed": 0 },
	"hits": {
	"total": { "value": 10000, "relation": "gte" },
	"max_score": 1,
	"hits": [
	{
	"_index": "audio-index",
	"_type": "_doc",
	"_id": "m3b0jnEBBeqWuw43UXjn",
	"_score": 1,
	"_source": {
	"time": "2020-04-18T15:35:00",
	"fileAttributes": {
	"fileName": "teste.wav",
	"filePath": "C:\arquivos\audios\teste.wav",
	"creationTime": "2019-09-24T00:31:53.141187Z",
	"size": 716460,
	"owner": "michieru-PC\michel",
	"fileType": "audio",
	"tag": "audio-aula",
	"duration": "60"
	},

	"words": [{ "word": "big", "occurrences": ["00:00:00.180"], "occurrenceQuantity": 1 }] } }] } }

A Tabela 18 apresenta uma chamada vazia e algumas informações importantes no retorno da chamada como *took*, *timed_out*, *shards* e *hits*. As informações de retorno estão descritas a seguir:

- ***Took***: Este valor diz quantos milissegundos durou a busca completa para ser executada.
- ***Timed_out***: Este valor informa se a consulta teve o tempo máximo esgotado. É possível especificar o tempo de timed_out na URL da chamada (exemplo para configurar o *timed out* como 10 milissegundos "/_search?timeout=10ms").
- ***Shards***: Este valor informa a quantidade de *shards* envolvidos na busca, quantos obtiveram sucesso e quantos falharam em sua execução. O Elasticsearch faz buscas e se algum *shard* falhar ele busca em sua cópia, mesmo se ambos falham o Elasticsearch retorna o resultado e reporta o *shard* como falha.
- ***Hits***: Esta sessão contém todos os documentos que correspondem à consulta realizada. Para cada documento ele contém algumas informações como *_index*, *_type*, e *_id* do documento, e no *_source* contém a estrutura do documento. Por padrão, o Elasticsearch contém valor de *_score* que significa o quanto o documento é relevante em relação à consulta realizada, e os resultados são retornados em ordem do mais relevante primeiro.

Para este protótipo essas informações não estão sendo consideradas para tomar nenhuma ação, entretanto elas podem ser utilizadas para fornecer mais detalhes da execução ou até mesmo realizar uma nova tentativa em caso de falhas.

Para realizar uma busca com filtros ou funções de agregação que é o objetivo deste protótipo, é preciso fornecer uma query no corpo da chamada para a API do Elasticsearch.

O Elasticsearch fornece uma estrutura de consulta completa chamada de DSL (*Domain Specific Language*) que é baseado em JSON. DSL é uma linguagem que o

Elasticsearch utiliza para expor todo o seu poder de recursos através de uma simples interface em JSON.

A Tabela 19 apresenta um exemplo de como realizar uma busca utilizando filtros na consulta.

Tabela 19. Estrutura da Consulta Utilizando Filtro de Busca.

Item	Resultado

Chamada	**curl** -XPOST http://localhost:9200/**audio-index/_search –d "CONSULTA"**
	
	CONSULTA:
	{
	"query": SUA_QUERY_AQUI
	
	}

Resultado da chamada	O mesmo retorno descrito na Tabela 18, entretanto somente com os dados referente a consulta.

O Elasticsearch contém muitas possibilidades e opções de cláusulas para realizar buscas. O foco deste protótipo não é explicar todas as possibilidades, mas sim uma combinação necessária para realizar a busca nos documentos indexados, por uma ou mais palavras faladas nos arquivos de áudio. Para mais detalhes sobre as possibilidades pode ser encontrado em o livro "*Elasticsearch: The Definitive Guide"* (Gormley e Tong 2015).

A Tabela 20 apresenta uma consulta completa para buscar pelas palavras "*big"* e "*city"*, e realizar uma ordenação pela quantidade de ocorrência da palavra nos documentos indexados.

Tabela 20. Exemplo da Consulta Completa no Elasticsearch.

Exemplo da Consulta Completa no Elasticsearch
{
"size": 100,
"query": {
"bool": {
"must": [
{
"nested": {
"path": "words",
"query": { "match": { "words.word": "big" } }
}
},
{

```
                "nested": {
                    "path": "words",
                    "query": { "match": { "words.word": "city" } }
                }
            }
        ]
    }
},
"sort": [
    {
        "words.occurrenceQuantity": {
            "order": "desc",
            "nested": {
                "path": "words",
                "filter": { "match": { "words.word": "big" } }
            }
        }
    },
    {
        "words.occurrenceQuantity": {
            "order": "desc",
            "nested": {
                "path": "words",
                "filter": { "match": { "words.word": "city" } }
            }
        }
    }
]
}
```

A Tabela 21 apresenta a descrição dos parâmetros utilizados na consulta descrita na Tabela 20. A consulta apresentada é uma das várias possibilidades que o Elasticsearch provê para buscar os arquivos.

Tabela 21. Parâmetros Utilizados na Consulta.

Parâmetro	Descrição
Size	Indica a quantidade de documentos que devem ser retornados, o valor padrão é 10.
Query	Elemento principal que contém toda estrutura da busca.
Bool	Esta clausula permite combinar outras clausulas, por exemplo, realizar uma busca por documentos que contenham duas palavras diferentes.
Must	Clausula composta que pode ser combinada com outras clausulas e também pode conter outras clausulas aninhadas. *Must* significa que a palavra fornecida na busca deve existir no documento. Existem outras clausulas como *should* (pode conter o valor ou não) e *must_not* (que não deve conter).
Nested	Para realizar busca em estruturas de documento que estão aninhados (Elemento JSON dentro de outro JSON), é preciso utilizar a clausula *nested* porque estes objetos são indexados como documentos ocultos separados, então não é possível realizar uma busca diretamente nestes campos. Por exemplo, a estrutura que contém as palavras

	extraídas estão em um objeto aninhado chamado *words* (ver Tabela 18).
Path	É o caminho da estrutura aninhada (nested) que está sendo realizada a consulta.
Match	É uma consulta padrão para procurar por um valor de texto em algum campo.
Sort	Por padrão o resultado é ordenado por ordem de relevância, entretanto a consulta utilizada precisa ser ordenada pela quantidade de ocorrência em que a palavra aparece no documento. Este campo é o "*words.occurrenceQuantity*"
Order	Está clausula diz como os dados devem ser ordenados, em ordem crescente ou decrescente.
Filter	Está clausula está sendo utilizada junto com a ordenação para saber o que deve ser ordenado, neste caso a mesma palavra utilizada na busca.

Está é a consulta principal utilizada pelo protótipo. Outras consultas que serão realizadas para construir alguns gráficos informativos serão responsabilidade da própria ferramenta de consulta do conjunto ELK; o Kibana. O Kibana também realiza a chamada da API do Elasticsearch.

Para visualizar todos os comandos utilizados no Elasticsearch, consultar o Apêndice II – Script para a Configuração do Elasticsearch.

4.5. Interface de Pesquisa

4.5.1. Kibana

O **Kibana** é uma interface de visualização de dados gratuita e de código aberto capaz de gerar gráficos, relatórios e *dashboards* com facilidade. No protótipo implementado, o Kibana é usado como *front-end* para o usuário realizar as pesquisas e visualizar seus resultados. Como vantagem adicional neste caso, o Kibana foi projetado para integrar nativamente com Elasticsearch, que está sendo utilizada como ferramenta de indexação.

Todas as consultas são realizadas diretamente nos dois índices criados para salvar o conteúdo extraído do áudio e dos logs de execução do protótipo.

Para sua configuração é preciso utilizar a mesma versão do Elasticsearch, e em seu arquivo de configuração precisa configurar o endereço de rede do Elasticsearch. Os passos para começar as consultas após o Kibana estar rodando com a configuração inicial serão divididos em três etapas:

1. Configuração inicial da base de dados no Kibana;
2. Pesquisa e criação de gráficos;
3. Criação de *dashboards*.

4.5.1.1. Configuração Inicial da Base de Dados no Kibana

A primeira etapa é realizar a configuração inicial no Kibana, para que ele possa encontrar e filtrar o índice criado no Elasticsearch que será utilizado. O primeiro passo é criar um padrão de índice, este padrão permite filtrar as buscas por um ou mais índices. O nome do padrão é composto por parte ou o nome inteiro do índice que se deseja buscar. Por exemplo, para o protótipo foi criado "audio-" que contém parte do nome do índice criado no Elasticsearch: "audio-index".

O segundo passo desta etapa é definir algum campo do tipo "data do documento" que será indexado, para permitir realizar ordenação dos dados por este campo. Esta não é uma configuração obrigatória, entretanto se não for configurada não será possível filtrar e ordenar os documentos pela data e hora que os registros entraram no sistema.

O terceiro passo é melhorar e enriquecer a visualização dos dados que estão salvos na base indexada. O Kibana permite realizar uma formatação nos dados antes de serem apresentados aos usuários. Por exemplo, configurar um campo que salva seus valores em bytes ou segundos, para que possa apresentar e converter os dados em algo que possa ser lido pelo usuário, ou seja, ao invés de mostrar 512000 bytes e 3600 segundos o que é apresentado ao usuário é 50 megabytes e 1 hora como informação.

Além de configurar a formatação dos campos também é possível dizer se o campo é pesquisável, o que significa se o campo é considerado ou não nas buscas. Também é possível configurar se o campo é do tipo agregável, que permite a utilização de funções de agregações, por exemplo, soma e média.

Após estes passos, o Kibana está pronto para permitir ao usuário criar gráficos dinâmicos e realizar pesquisas nos documentos que estão sendo inseridos no Elasticsearch. Para mais detalhes de como realizar a configurar inicial consultar o Apêndice III – Configuração Inicial da Base de Dados no Kibana.

4.5.1.2. Pesquisa e Criação de Gráficos

O Kibana permite procurar diretamente nos documentos salvos no Elasticsearch. Através dos campos mapeados, o Kibana permite criar queries no Formato Kibana Query Language (KQL); linguagem simplificada mais fácil do que usar a API nativa do Elasticsearch.

A Figura 16 mostra a opção de realizar as buscas no Kibana que se encontra no menu lateral com o nome de *discover*. Basta escolher entre os padrões de índices existentes, por exemplo, audio-index. No campo principal de consulta KQL é possível realizar as consultas. Por exemplo, para procurar por documentos que tem a tag "vídeo-aula" e são arquivos são do tipo ".avi", a consulta utilizada seria:

```
fileAttributes.tag : "video-aula" and fileAttributes.fileName : "*.avi"
```

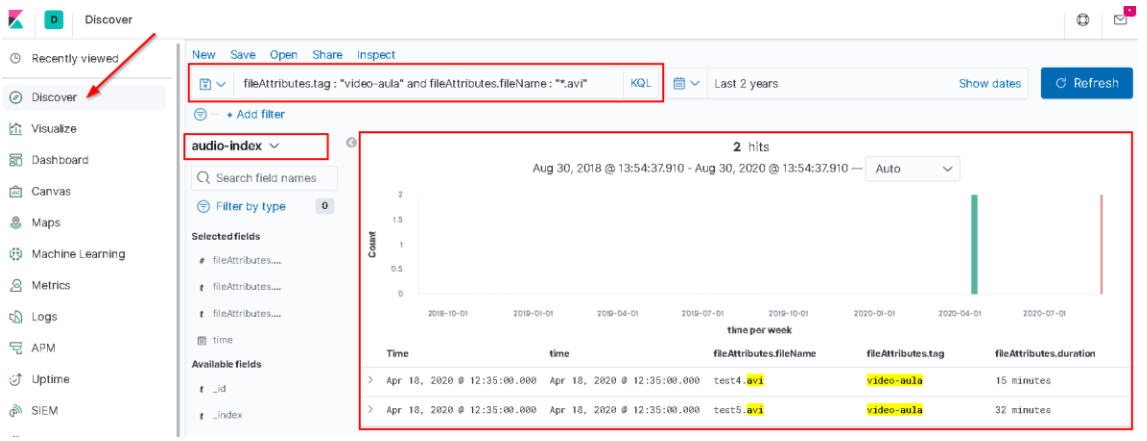

Figura 16. Pesquisa no Kibana.

Outro exemplo, para realizar uma busca por duas palavras especificas (look e big), a seguinte consulta de KQL é utilizada:

```
words:{ word : "look" } and words:{ word : "big" }
```

A consulta retornará todos os campos presentes no documento, mas é possível escolher quais campos serão apresentados na tabela. Com a consulta pronta também é possível salvar a consulta da tabela para ser utilizada em algum *dashboard*.

O Kibana permite criar visualizações a partir de várias opções pré-existentes, por exemplo, gráficos de barra, pizza, gauge, métrica, mapas ou visualizações customizadas

usando sua ferramenta de criação chamada vega. A Figura 17 apresenta algumas opções de visualizações que podem ser utilizadas.

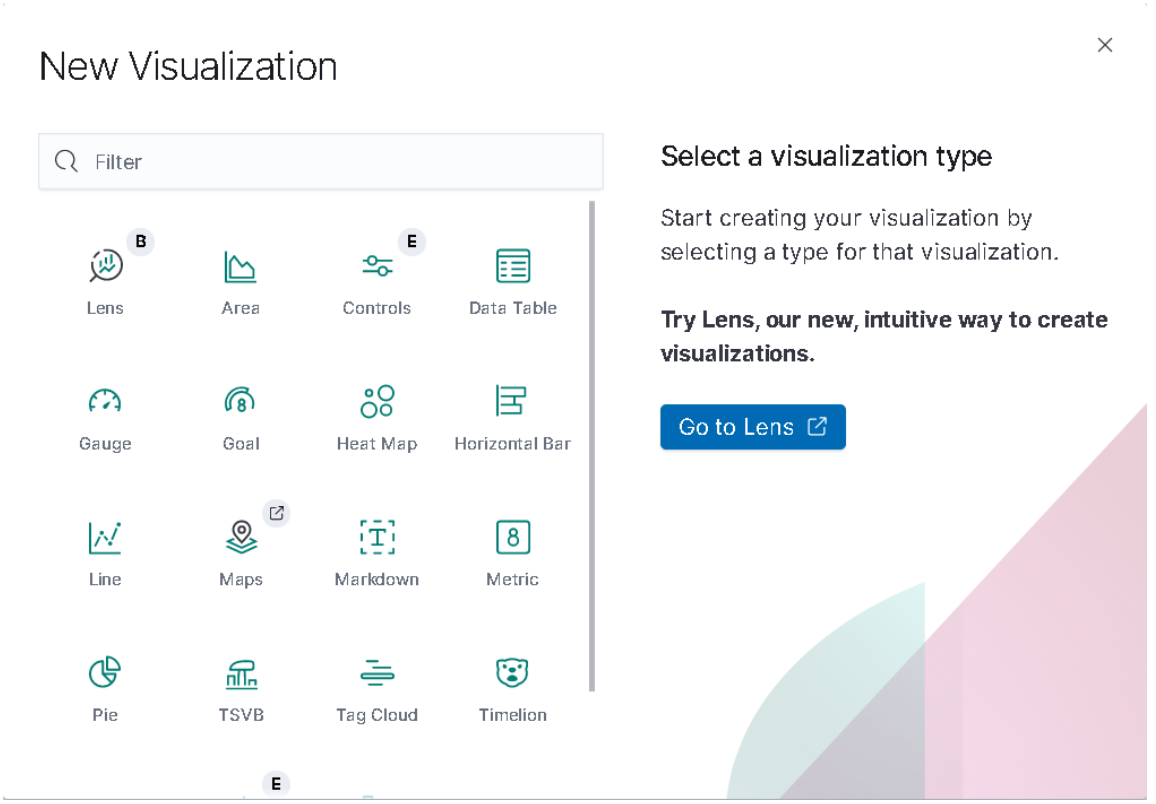

Figura 17. Opções de Visualizações.

Para o protótipo algumas visualizações foram criadas para apresentar o resultado dos arquivos processados:

- **Métrica única (*Metric*)**: é um tipo de visualização que mostra um cálculo ou um número único. Além disso, permite usar funções de agregações como soma, média, mínimo, máximo entre outros. Por exemplo, apresentar a soma em bytes dos arquivos de áudio processados, ou a duração média de processamento de cada arquivo.
- **Gráfico de barra**: é um tipo de visualização que apresenta os dados em forma de barras em um gráfico. Por exemplo, apresentar as 5 *tags* que mais tem documentos indexados, ou a quantidade de arquivos processador por consumidores.
- **Gráfico de pizza**: é um tipo de visualização parecido com o de barra, mas que apresenta os dados no formato de uma pizza. Por exemplo, apresentar a quantidade de arquivos processados por tipo (áudio e vídeo).

Para mais detalhes de como configurar os gráficos e os *dashboards* consultar o Apêndice IV – Pesquisa e Criação de Gráficos no Kibana. A Tabela 22 apresenta a descrição de todas as visualizações criadas em cima dos dados processados e salvos nos índices do Elasticsearch.

Tabela 22. Visualizações Criadas em cima dos Arquivos Processados.

Nome da Visualização	Tipo de Visualização	Descrição
Quantidade de Arquivos	Métrica	Apresenta a quantidade total de arquivos processados.
Quantidade de Dados	Métrica	Apresenta a soma total do tamanho dos dados processados.
Duração Total	Métrica	Apresenta a soma total do tempo de duração dos arquivos processados.
Quantidade de Arquivos por Tipo	Gráfico de Pizza	Apresenta a quantidade de arquivos agrupado por tipo (áudio e vídeo).
Quantidade de Arquivos por *Tag*	Gráfico de Pizza	Apresenta a quantidade de arquivos agrupado por *tags*.
Quantidade de Dados por Tag	Gráfico de Barra	Apresenta a soma de dados dos arquivos agrupado por tag.

A Tabela 23 apresenta a descrição de todas as visualizações criadas em cima dos logs de processamento dos arquivos gerados pelos produtores e consumidores.

Tabela 23. Visualizações Criadas em cima dos Logs de Processamento dos Arquivos.

Nome da Visualização	Tipo de Visualização	Descrição
Quantidade de Produtores	Métrica	Apresenta a quantidade de produtores envolvidos no processamento.
Quantidade de Consumidores	Métrica	Apresenta a quantidade de consumidores envolvidos no processamento.
Duração Média de Processamento pelos Produtores	Métrica	Apresenta a duração média de processamento total de um arquivo pelos produtores.
Duração Média de Processamento pelos Consumidores	Métrica	Apresenta a duração média de processamento total de um arquivo pelos consumidores.
Duração Média da Conversão para Binário	Métrica	Apresenta a duração média de processamento para converter o arquivo de áudio para binário.
Duração Média de	Métrica	Apresenta a duração média de processamento para enviar o

Envio para o Tópico do Apache Kafka		arquivo no formato binário para o tópico do Apache Kafka.
Duração Média para Extrair o Texto	Métrica	Apresenta a duração média de processamento para extrair o texto falado no arquivo utilizando a ferramenta de reconhecimento de fala.
Duração Média de Envio para o Índice do Elasticsearch	Métrica	Apresenta a duração média de processamento para enviar o conteúdo extraído para o índice do Elasticsearch.
Quantidade de Arquivos Processados Pelos Produtores	Gráfico de Barra	Apresenta a quantidade de arquivos processados agrupado pelos produtores.
Quantidade de Arquivos Processados Pelos Consumidores	Gráfico de Barra	Apresenta os 30 primeiros consumidores com mais quantidade de arquivos processados.

4.5.1.3. Criação de *Dashboards*

Com todos os gráficos necessários criados, o Kibana permite agrupa-los em uma única ou mais visualizações chamadas de *dashboards*. Os *dashboards* são uma coleção de gráficos, tabelas, métricas, mapas e buscas que foram coletados juntos e estão agrupados em um único painel. Dois *dashboards* foram criados para apresentar os dados processados.

O primeiro *dashboard* apresenta métricas dos dados dos arquivos de áudio processados. Como por exemplo: detalhes de quantos arquivos foram processados; quantidade de dados em bytes processada; duração total dos arquivos entre outras métricas relacionadas aos arquivos e seus dados coletados. O *dashboard* contém as visualizações e gráficos que foram criados e apresentados na Tabela 22.

O segundo *dashboard* agrupa gráficos e informações relacionadas ao processamento dos arquivos de áudio, como a duração de processamento para cada etapa do processamento do arquivo e informações sobre os produtores e consumidores. O *dashboard* com os dados de processamento contém as visualizações e gráficos apresentados na Tabela 23.

Os *dashboards* criados no Kibana podem ser compartilhados com outros usuários e incorporado a outros sistemas através de um link de compartilhamento. Além disso, o

Kibana permite filtrar os dados de todos os gráficos de uma só vez por data e também permite que sejam atualizados em tempo real.

Esta ferramenta permite enriquecer e transformar os dados coletados em informações que sejam uteis para visualização da saúde do sistema e para tomada de decisões. Permitir saber a velocidade de processamento dos dados, saber qual o progresso do processamento, e também se é preciso escalar o sistema e colocar mais processadores em funcionamento. Tudo com base nos dados organizados em visualizações e *dashboards* que foram extraídos durante o processamento.

4.5.2. Interface Desenvolvida para Pesquisa de Palavras

Com as API disponíveis pelo Elasticsearch a construção e a integração de uma ferramenta de busca por palavras se torna mais fácil. O Kibana é uma excelente ferramenta para conectar no Elasticsearch, entretanto se precisar de alguma visualização customizada ou mais avançada, por exemplo, visualizar somente as palavras pesquisadas e o tempo em que as palavras ocorrem, pode ser necessária a construção de uma nova ferramenta.

A estrutura do documento salvo no Elasticsearch para este projeto utiliza *nested objects* como mencionado anteriormente. A versão 7.6.2 utilizada neste protótipo não permite visualizar ou filtrar somente os *nested objects*, de uma forma mais clara para facilitar visualização somente dos dados que o usuário quer visualizar.

Por exemplo, para uma busca de uma palavra especifica, o Kibana vai filtrar todos arquivos que contenham a palavra. Entretanto, a estrutura que contém todas as palavras não será filtrada para apresentar somente a palavra que foi buscada. Serão retornados os documentos encontrados e em cada documento a lista com todas as palavras existentes.

A Figura 18 mostra como uma busca pelos documentos que contenham a palavra “big” é apresentada para o usuário. Todas as palavras são apresentadas no formato JSON e o documento completo também é visualizado. O que não fica fácil de visualizar é saber em qual momento do arquivo acontece a ocorrência da palavra buscada.

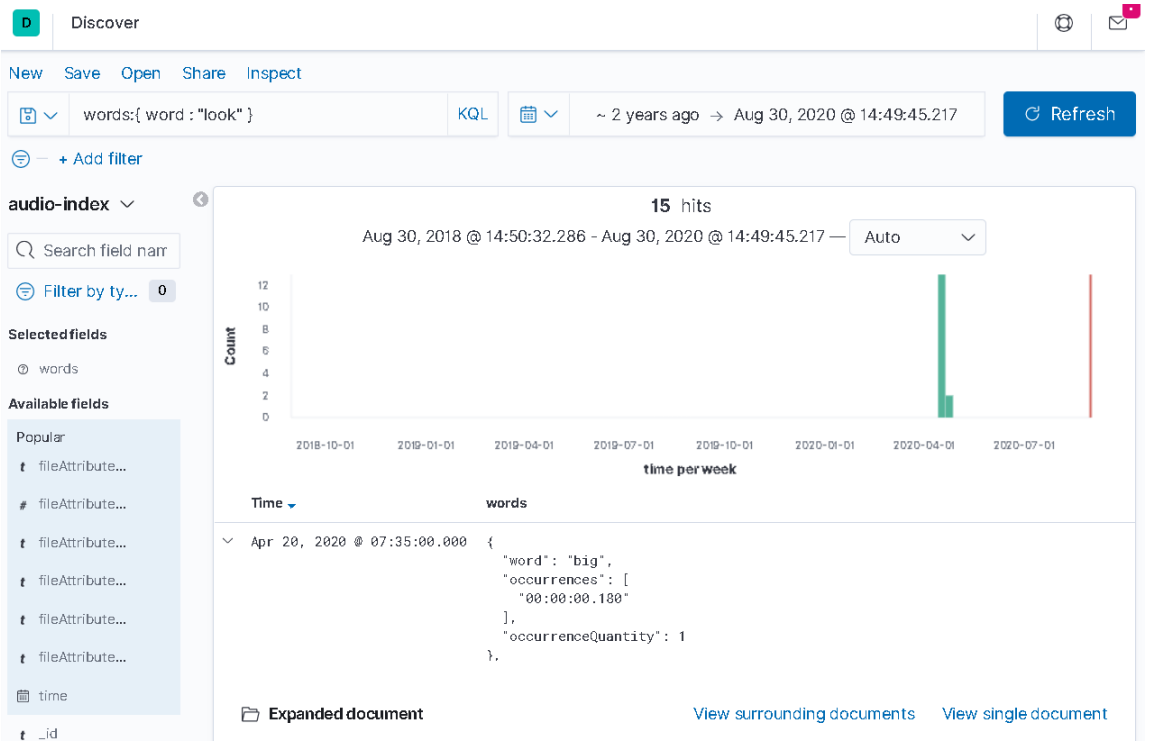

Figura 18. Busca por palavras no Discover do Kibana.

A ferramenta *Discover* do Elasticsearch atende ao propósito deste protótipo, ele permite buscar e visualizar os dados. Entretanto, para melhor apresentar os resultados da busca, com o foco em apresentar somente as palavras que o usuário quer visualizar, foi desenvolvido um protótipo de uma ferramenta web para apresentar os dados de uma forma mais amigável ao usuário.

As principais ferramentas utilizadas para a construção do protótipo de interface de pesquisa das palavras foram:

- **JQuery versão 3.5.0:** Biblioteca em JavaScript responsável por controlar ações que acontecem na tela como realizar uma operação busca, navegar nas telas disponíveis e por fazer as chamadas para a API do Elasticsearch afim de recuperar os dados;
- **Tabulator versão 4.7:** Biblioteca em JavaScript para construção de tabelas dinâmicas automaticamente com várias funções, como por exemplo ordenação e busca, construída com JavaScript e HTML;
- **HTML5:** Para a criação dos componentes como botões e menus;
- **CSS:** Para a estilização dos componentes criados em HTML5 e alterações de cores na página.

Devido à API do Elasticsearch, a construção de um novo sistema se torna mais fácil e abstrai a complexidade de realizar as pesquisas nos índices da base de dados. A estrutura das consultas realizadas estão apresentadas na Tabela 20. Com esta premissa, a interface web construída se limita apenas em realizar a chamada para a API e apresentar os dados para o usuário.

Esta interface construída permite visualizar os resultados das buscas em forma tabular, com isso permite filtrar e utilizar as funcionalidades pré-existentes nas bibliotecas de JavaScript utilizada para ordenar e filtrar os dados diretamente na tela.

Os links dos *dashboards* construídos no Kibana para visualização dos dados e de estatística de processamento foram colocados dentro de páginas da interface, para organizar os resultados em uma única ferramenta.

O protótipo da interface apresenta um menu no topo com 3 links para diferentes seções:

- O primeiro item do menu é a "**Ferramenta de Busca**". Que é a página para realizar busca na base de dados. Está página contém um campo de texto para realizar a pesquisa, e permite colocar uma ou mais palavras separadas por espaço para serem considerados na busca, botão de pesquisa e a tabela que apresenta os dados.
- O segundo item do menu é o "***Dashboard***". Que é um link que carrega o *dashboard* principal criado no Kibana, que apresenta informações de quantos arquivos foram processados, a quantidade de dados, e algumas informações referente aos arquivos.
- O terceiro item do menu é o de "**Estatísticas**". É um link para o segundo *dashboard* criado no Kibana, que apresenta informações referente ao processamento dos arquivos pelo protótipo que extrai o texto dos arquivos.

Esta interface de busca completa e finaliza a implementação do protótipo que utiliza a arquitetura de solução proposta nesta dissertação.

5. Resultados

Este capítulo apresenta os resultados obtidos durante o desenvolvimento do protótipo e dos testes realizados no ambiente com todas as ferramentas da arquitetura. O processo para a coleta dos resultados foi dividido em três etapas: a primeira etapa foram os testes preliminares, a segunda etapa foi a configuração do ambiente de teste e terceira etapa foi a execução e coleta dos resultados dos testes do protótipo.

5.1. Testes Preliminares

O objetivo principal dos testes preliminares foi identificar uma configuração ideal para o processamento final dos arquivos de áudios. Dois testes preliminares foram executados, o primeiro com a finalidade de testar o percentual de acerto da ferramenta de reconhecimento de fala escolhida. O segundo um teste de concorrência, com a finalidade de identificar o número máximo de threads que podem ser executadas ao mesmo tempo, em uma única máquina, sem que ocorra perda de desempenho.

Na Tabela 24, é possível ver as configurações de hardware da máquina utilizadas durante os testes preliminares.

Tabela 24. Configuração da máquina de teste.

Configuração do sistema	Valor
Sistema operacional	Windows 7 - 64bits
Processador	Intel® Core™ i7-3612QM CPU @ 2.10 GHz 2.10 GHz
Quantidade de núcleos	8 núcleos
Memória	8 GB
Disco rígido	1 TB

5.1.1. Teste de Percentual de Acerto do CMU Sphinx

Foi construído um programa em Java que utiliza a ferramenta de reconhecimento de fala CMU Sphinx. Estes testes buscam verificar o percentual de acerto do reconhecimento de palavras de alguns arquivos de áudios aleatórios. Este teste busca

demonstrar que a ferramenta de reconhecimento de fala escolhida tem um percentual de acerto de pelo menos 70%, valor escolhido para nivelar os testes.

O objetivo é testar arquivos de áudios com diferentes locutores (todos nativos) e quantidades de palavras para não ter um teste enviesado. Os arquivos foram escolhidos a dedo com o único requisito de que os locutores tinham que ser diferentes.

Dois arquivos foram extraídos da internet, um de uma aula de inglês e um segundo arquivo com a narrativa de um capítulo da Bíblia. Três arquivos são capítulos de livros narrados da base de áudio livro da Librivox. A Librivox é um projeto para disponibilizar todos os livros de domínio público, narrados por pessoas reais e distribuídos gratuitamente, em formato de áudio na internet (Librivox 2021).

A Tabela 25 apresenta o resultado dos testes de conversão dos arquivos de áudio em texto falado usando o CMU Sphinx. Para conferir o percentual de acerto, os arquivos contam com sua transliteração para checagem. As palavras originais e as palavras reconhecidas foram colocadas manualmente em duas colunas para conferir o resultado de quais palavras foram reconhecidas.

Tabela 25. Resultado dos testes de conversão de fala em texto usando CMU Sphinx.

Nome	Descrição	Duração do áudio	Quantidade de palavras existentes	Quantidade de palavras reconhecidas	Porcentagem de acerto
A big city	**Arquivo falado por um nativo para uma aula de inglês.**	**00:00:22**	**59**	**56**	**94,92%**
Northanger Abbey	Livro de Jane AUSTEN (1775 - 1817), capítulo 01 em áudio livro	00:08:42	1228	1074	87,46%
Gênesis Capítulo 1	**Capítulo 1 do livro de gênesis da Bíblia sagrada. A qualidade do áudio falado não estava muito boa.**	**00:05:21**	**797**	**576**	**72,27%**
The Silence: What It Is, How To Use It	Livro de David Van BUSH (1882 - 1959), capítulo 01 em áudio livro	00:07:46	1171	1017	86,85%
Emma	Livro de Jane AUSTEN (1775 - 1817), capítulo	00:11:31	1722	1469	85,31%

	01 em áudio livro				

A Tabela 25 apresenta algumas informações sobre os arquivos e os resultados do teste, a seguir e detalhado o significado de cada coluna:

- **Nome:** Nome do arquivo de áudio;
- **Descrição:** Breve descrição sobre o arquivo. Por exemplo, qual o capítulo do livro a narração pertence;
- **Duração do áudio:** Duração do total do arquivo de áudio;
- **Quantidade de palavras existentes:** Representa a quantidade de palavras faladas dentro do arquivo de áudio;
- **Quantidade de palavras reconhecidas:** Esta coluna apresenta a quantidade de palavras que foram reconhecidas utilizando o CMU Sphinx;
- **Porcentagem de acerto:** Porcentagem de acerto foi calculada com base na quantidade de palavras reconhecidas.

A partir dos dados obtidos com base na amostra, a menor porcentagem de acerto de palavras foi de 72%, e a maior foi de 94,92%. Para este trabalho foi definido um valor esporádico de 70% para ser o percentual mínimo de resultado. Todos os resultados da amostra atingiram o valor mínimo de porcentagem de acerto definido.

A conversão que teve o pior resultado foi a leitura de um capítulo do livro de Gênesis da Bíblia sagrada, onde o áudio não tem a mesma qualidade de pronúncia durante a leitura em comparação aos outros arquivos. A conversão que obteve o melhor resultado é um arquivo curto gravado para uma aula de inglês, onde existe uma atenção especial em sua pronuncia e na qualidade do arquivo. Já os outros resultados foram extraídos da Librivox, onde os voluntários que gravaram o áudio tiveram um pouco mais de cuidado na pronuncia ao falar o conteúdo durante a criação do áudio livro.

Devido ao resultado atingir o valor mínimo definido para este trabalho e com base em artigos mencionados como base, o CMU Sphinx foi a ferramenta escolhida para realizar o reconhecimento de fala dos arquivos de áudio. Para a construção do protótipo poderia ser utilizado qualquer ferramenta de reconhecimento de fala, o CMU Sphinx foi escolhido devido aos resultados e ao conteúdo disponível na internet para sua utilização.

5.1.2. Teste de Concorrência

O teste preliminar de concorrência foi importante para poder testar e determinar o número máximo de instâncias de processamento que podem ser executadas em uma única máquina ao mesmo tempo. O objetivo é usar ao máximo os recursos do servidor e não ter perda de desempenho.

A Tabela 26 apresenta os dois cenários utilizados nos testes de concorrência: um processamento com 10 arquivos e outro com 100 arquivos. Para facilitar a comparação, um único tipo de arquivo foi utilizado. O arquivo escolhido foi "*A big city*", que teve um percentual de quase 95% de acerto. Este arquivo foi copiado 10 e 100 vezes a fim de realizar os testes.

Tabela 26. Informações sobre os arquivos usados no teste de concorrência.

Quantidade de arquivos	Tamanho total	Duração total
10	6,83 MB	00:03:43
100	68,3 MB	00:37:18

A Tabela 27 apresenta o resultado do teste de concorrência dos dois cenários apresentados na Tabela 26, ao utilizar diferentes números de *threads* durante o processamento. O objetivo é identificar o número máximo de threads que podem estar em execução concorrente e que tenha o menor tempo de duração de processamento. Na Tabela 27 é possível visualizar a quantidade de threads utilizada e a duração de processamento para processar 10 e 100 arquivos de áudios respectivamente.

Tabela 27. Resultado do teste de concorrência.

Número de *threads*	Duração - 10 arquivos	Duração - 100 arquivos
1 Thread	00:05:34	00:49:00
2 *Threads*	00:03:10	00:28:32
3 *Threads*	00:02:40	00:22:51
4 *Threads*	00:02:51	00:21:20
5 *Threads*	**00:02:07**	**00:20:43**
6 *Threads*	00:03:02	00:26:51

7 *Threads*	00:14:41	02:18:22
8 *Threads*	00:18:17	02:46:03

Os resultados da Tabela 27 demonstram que aumentar o número de threads melhora o tempo final de processamento até certo ponto. Após essa quantidade máxima ser atingida o desempenho começa a piorar, quando a ferramenta CMU Sphinx é utilizada para o processamento em um ambiente parecido ou igual ao descrito na Tabela 24.

O melhor resultado obtido durante os testes foi usar no máximo 5 *threads* para o processamento dos arquivos. Utilizar um número acima de 5 *threads* fez com que a duração total de processamento começasse a levar mais tempo do que usar um número menor de *threads* na execução.

Um motivo para que ocorra uma piora no desempenho ao utilizar mais de 5 *threads* no processamento é que para processar os arquivos áudios com o CMU Sphinx é utilizado muito recurso de leitura durante a extração do texto falado nos áudios. Isto ocorre quando o software está realizando a consulta e o reconhecimento das palavras. A cada thread iniciada aumenta o consumo de memória e CPU e quando mais de 5 threads são iniciadas o sistema operacional fica com poucos recursos para realizar dividir e continuar o processamento.

Desta forma, o melhor cenário obtido foi ao utilizar 5 *threads* para processar 10 arquivos, com tempo de 2 minutos e 7 segundos e 20 minutos e 43 segundos para processar 100 arquivos. O tempo de processamento com cinco *threads* foi de aproximadamente 40% menor que o tempo de processamento com uma *thread*.

A Figura 19 apresenta um gráfico de linha contendo os valores da Tabela 27 convertidos para decimal. O objetivo é poder observar a curva de processamento e facilitar a visualização de qual cenário teve o melhor resultado.

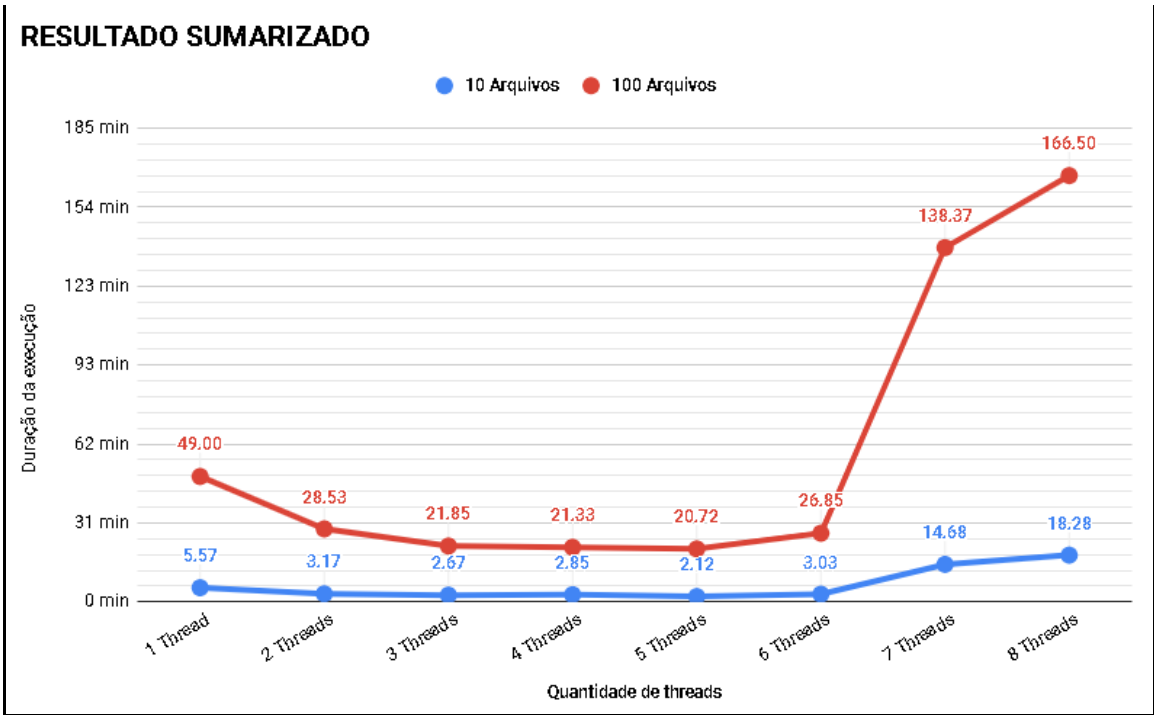

Figura 19. Resultado sumarizado do processamento paralelo.

Portanto, com o resultado dos testes é possível concluir que a ferramenta CMU Sphinx ao processar os arquivos apresentados na Tabela 25, obteve o percentual de acerto das palavras esperado de no mínimo 70% para sua aceitação de utilização no protótipo. Adicionalmente, podemos concluir que o melhor desempenho é obtido quando utilizadas cinco instancias do protótipo ao mesmo tempo no sistema operacional utilizado para os testes.

5.2. Configuração do Ambiente de Teste

Esta etapa é responsável por configurar o ambiente necessário para executar os testes finais do protótipo. Vários servidores em nuvem foram utilizados para hospedar as ferramentas utilizadas neste protótipo e realizar os testes. A Figura 20 apresenta a distribuição e a configuração do ambiente de teste criado para a execução completa do teste final do protótipo.

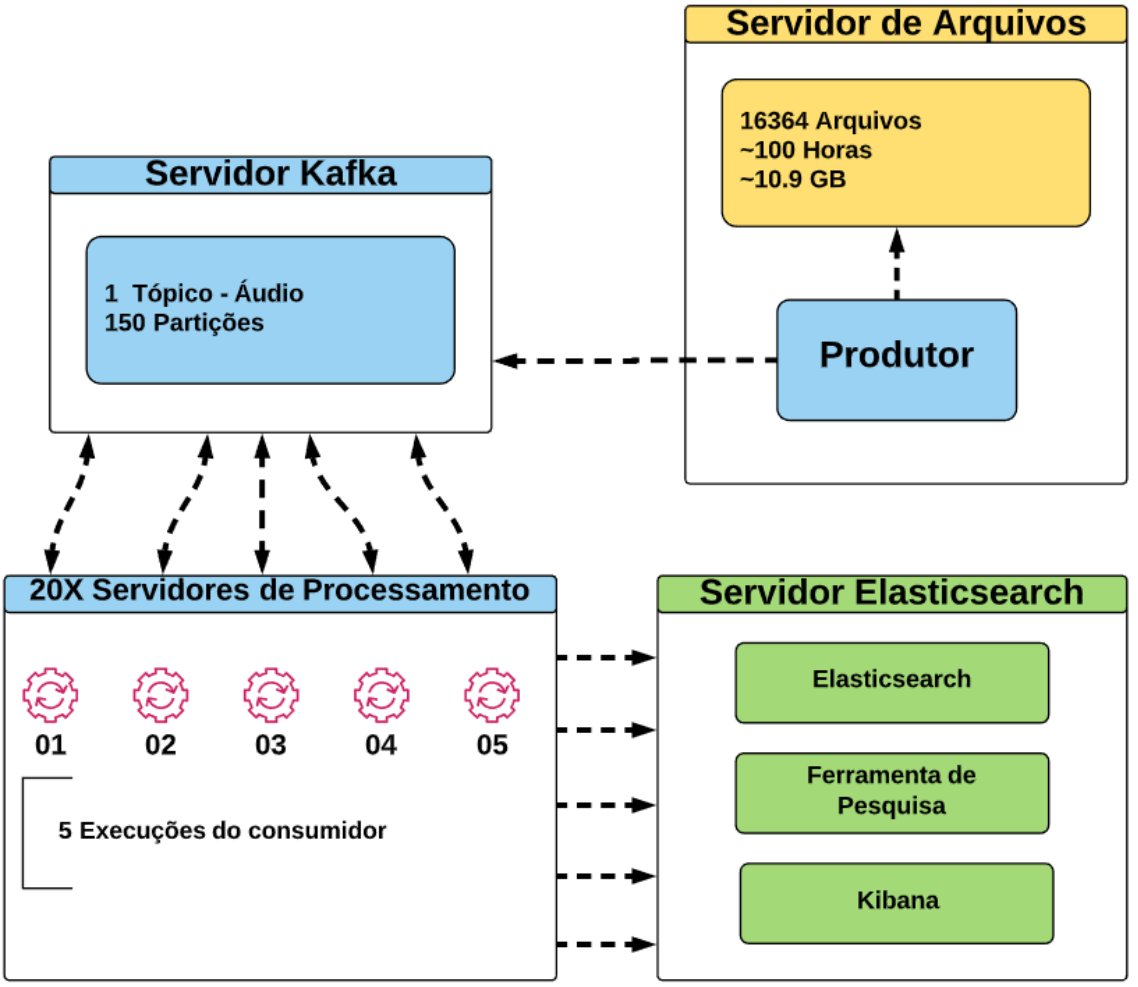

Figura 20. Configuração do Ambiente de Teste.

Foram utilizados 23 servidores para preparar o ambiente para o teste. Um servidor foi dedicado como servidor dos arquivos utilizados nos testes. No total 100 horas de áudio foram armazenadas no servidor. O servidor de arquivos contém uma única instância do produtor em execução para processar e enviar os arquivos de dados para o servidor do Apache Kafka.

Para o Apache Kafka somente um servidor foi utilizado, como descrito no capítulo 4.2.3 e somente um tópico foi criado com 150 partições para receber e dividir os dados recebidos pelo produtor.

Para realizar o processamento dos dados salvos no Apache Kafka, 20 servidores de processamento foram utilizados, no qual, cada servidor contém cinco execuções do consumidor/processador totalizando 100 execuções do consumidor. No momento da criação dos servidores é possível escolher uma região onde o servidor será hospedado.

Os servidores foram distribuídos em diferentes regiões geográficas com o objetivo de demonstrar que não importa a localidade dos servidores e que é possível utilizar uma máquina em qualquer região do planeta que tenha acesso `a internet para ajudar no processamento. A Tabela 28 apresenta a distribuição de servidores de processamento por regiões:

Tabela 28. Distribuição de Servidores de Processamento por Regiões.

Região	Quantidade de Servidores
Leste dos Estados Unidos	4
Central Sul dos Estados Unidos	2
Oeste dos Estados Unidos	4
Central dos Estados Unidos	2
Central Norte dos Estados Unidos	2
Leste da Austrália	2
Central do Canadá	2
Sul do Brasil	2

Por último, um servidor foi utilizado para hospedar o Elasticsearch, Kibana e a ferramenta de pesquisa. No Elasticseach foram criados dois índices, o primeiro chamado audio-index para salvar os textos extraídos dos arquivos de áudio, e um segundo chamado processamento-index para salvar os logs de processamento dos arquivos.

Para mais detalhes, sobre scripts utilizados para automatizar a configuração do ambiente, ver o Apêndice I – Shell Scripts para Construção do Ambiente de Teste em Linux.

5.3. Resultado Final da Execução dos Testes do Protótipo

A Tabela 29 apresenta o resultado final contendo a duração total para processar cerca de 100 horas de áudio distribuído em 16364 arquivos. Os arquivos foram divididos em quatro pastas de modo a simular um cenário: reportagem, vídeo aula, áudio aula e arquivos de entrevistas.

O protótipo realizou todo o processamento em cerca de 2 horas e 20 minutos. Depois de concluído, os dados estavam disponíveis no Elasticsearch para consulta.

Tabela 29. Resultado Final de Processamento.

Item	Valor
Quantidade Total de Arquivos Processados	16364
Duração Total	~100 horas de áudio
Horário de Início	00:30
Horário de Término	02:50
Duração Total de Processamento	**02 horas e 20 minutos**

A Tabela 30 apresenta o custo estimado de utilização dos 23 servidores utilizados no protótipo, com base em valores atuais de 2020 ao utilizar a plataforma em nuvem da Google. A duração estimada entre configuração do ambiente e processamento foi de 3 horas e a soma do custo de utilização dos servidores foi de aproximadamente R$ 175,00. Para os servidores de processamento, por serem maquinas com mais poder computacional, o custo foi maior.

Tabela 30. Custo Estimado de Uso dos Servidores.

Item	Valor
Tempo de Execução dos servidores	~3 horas
Servidor Arquivos, Apache Kafka e Elasticsearch (3 servidores utilizados)	R$ ~5,00
Servidores de Processamento (20 servidores utilizados)	R$ ~170,00
Custo Total Estimado	R$ ~175,00

A Figura 21 apresenta o *dashboard* que foi criado no Kibana e que está sendo apresentado na ferramenta de busca. Os dados apresentados são referentes às 100 horas de áudio processado. Este *dashboard* apresenta os dados relacionados aos arquivos que são processados pelo sistema, como informações de quantos bytes e arquivos foram processados entre outras informações.

Figura 21. *Dashboard* Principal com os Dados dos Arquivos de Áudio.

A Figura 22 apresenta o *dashboard* criado no Kibana a partir das informações de log de processamento dos arquivos de áudio. É possível visualizar informações de quantos produtores e consumidores foram utilizados no processo, como também informações relacionadas à duração média em tempo de processamento de cada etapa envolvida no protótipo.

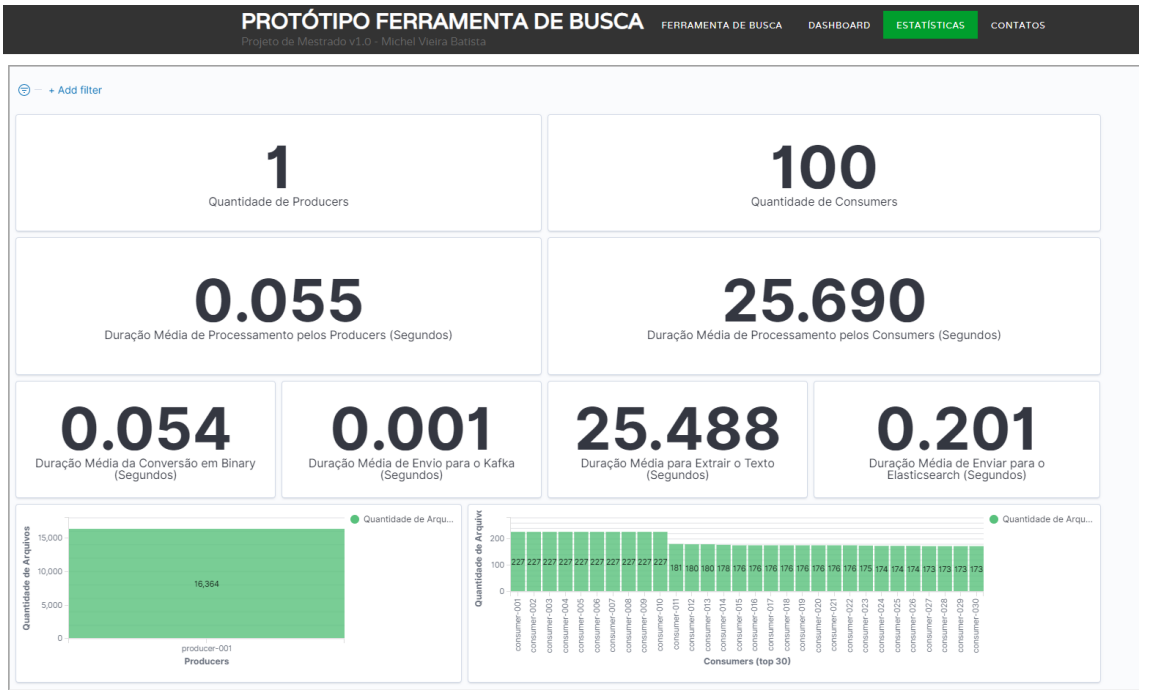

Figura 22. *Dashboard* com os Dados de Processamento.

A Tabela 31 apresenta os dados da Figura 22 em forma tabular para poder detalhar um pouco mais os resultados.

Tabela 31. Duração Média de Processamento por Etapa.

	Etapa de Medição	Duração Média em Segundos	Descrição
Produtor	Conversão em Binário	0.054	Processo para converter o arquivo em binário.
	Envio para o Apache Kafka	0.001	Processo para enviar o arquivo em binário para o Apache Kafka.
	Duração média de processamento pelo Produtor	0.055	Duração média total de processamento pelo produtor.
Consumidor	Extrair o Texto do Áudio	25.488	Processo para extrair o texto dos arquivos.
	Envio para o Elasticseach	0.201	Processo para enviar o texto extraído para o Elasticseach.
	Duração média de processamento pelo Consumidor	25.690	Duração média total de processamento pelo consumidor.
	Duração Média Total	**25.745**	**Duração média total de processamento por arquivo.**

Com base nos dados apresentados na Tabela 31, o maior tempo de processamento está no processo de extrair o texto dos arquivos de áudio que representa quase 99% do tempo total de processamento de um arquivo. Este é o gargalo atual do sistema. Portanto, dependendo da ferramenta de reconhecimento de fala utilizada é possível melhorar o tempo médio de processamento total dos arquivos.

A Figura 23 apresenta o resultado de uma busca pela palavra "*capital*" no protótipo da ferramenta de busca. A ferramenta apresenta informações que foram salvas no Elasticsearch para os arquivos que contém a palavra da busca, informações como tamanho do arquivo, duração, endereço entre outras informações.

Figura 23. Ferramenta de Busca de Palavras.

A Figura 24 apresenta a lista com todas as palavras, quantidade de ocorrências e o momento em que foram encontradas as ocorrências das palavras no arquivo processado. Também é possível filtrar a lista de palavras pela mesma palavra utilizada na busca ou por qualquer outra palavra que possa existir.

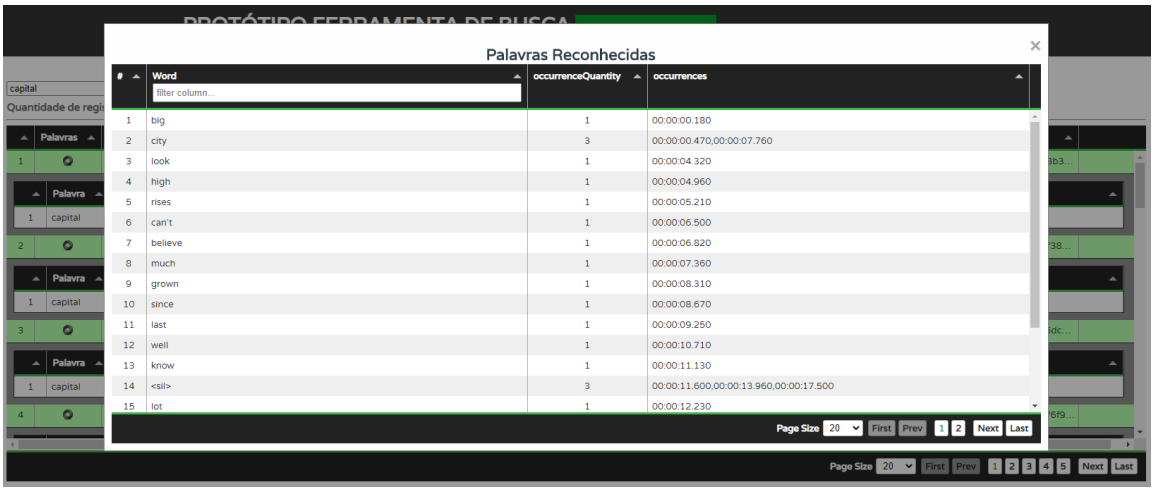

Figura 24. Palavras e suas Ocorrências de um Arquivo.

O resultado do protótipo atingiu o objetivo. A arquitetura de solução proposta permitiu ao usuário realizar pesquisas por palavras faladas dentro de arquivos de áudios previamente processados, saber a quantidade de ocorrência e informações adicionais referentes aos arquivos processados.

Como resultado, o protótipo construído foi capaz de processar cerca de 100 horas de áudio em aproximadamente 2 horas e 20 minutos. Durante o processamento dos

arquivos era possível visualizar os *dashboards* para saber quantos arquivos foram processados.

O protótipo utilizou somente ferramentas de livre uso e os únicos custos envolvidos no desenvolvimento foram pela utilização de servidores em nuvem. Isto permitiu demonstrar a capacidade do protótipo construído.

A Tabela 32 apresenta uma projeção linear de tempo estimado para realizar o processamento com uma quantidade maior de dados, com base nos resultados previamente obtidos. Por exemplo, considerando um ano, 10 anos e 100 anos de áudio, se juntar a duração de todos os arquivos que serão processados.

Tabela 32. Projeção de Tempo Estimado de Processamento.

Quantidade de Áudio	Tempo Estimado de Processamento
100 horas (~4 dias de áudio continuo)	~2 horas e 20 minutos
8760 horas (~1 ano de áudio continuo)	~204 horas (~8 dias e 12 horas)
87600 horas (~10 anos de áudio continuo)	~2044 horas (~85 dias/ 2 meses e 25 dias)
876000 horas (~100 anos de áudio continuo)	~20440 horas (~851 dias/ 2 anos e 4 meses)

A Tabela 32 apresenta apenas uma estimativa considerando um cenário parecido com o atual ambiente configurado e arquivos de duração aproximada à utilizada durante os testes. Portanto o tempo de processamento final pode sofrer alterações e ser melhorado à medida em que os componentes da arquitetura ou o ambiente de execução são alterados.

6. Conclusões

O objetivo deste trabalho é apresentar uma arquitetura de solução para a indexação de grandes volumes de áudio e vídeo usando reconhecimento de fala e construir um protótipo com base na arquitetura apresentada. Esta arquitetura permitirá indexar alguns arquivos de áudio, extrair as palavras faladas e deixar disponível para o usuário realizar pesquisas.

As soluções para o mesmo tipo de problema são escassas no mercado. Os sistemas existentes não permitem a liberdade de escolhas das ferramentas ou de suas utilizações. Ferramentas que obtém sucesso nesta área, em sua grande parte, são ou se tornam ferramentas proprietárias.

A arquitetura de solução proposta é de baixo acoplamento e não está amarrada a tecnologias específicas de reconhecimento de fala, indexação, processamento e pesquisa dos dados. Esta arquitetura permite escalar o sistema para poder processar os dados em paralelo em uma ou mais máquinas. Dependendo das ferramentas escolhidas, o sistema pode utilizar máquinas Linux e/ou Windows. Isto dá a capacidade de interoperabilidade ao sistema, por poder se comunicar de uma forma transparente entre diferentes arquiteturas.

A implementação do protótipo com softwares open sources, de fácil integração e modificações, demonstraram que é possível desenvolver uma solução para resolver o problema proposto neste trabalho com baixo custo. É possível utilizar várias ferramentas para resolver parte do problema proposto e reduzir a carga de desenvolvimento. Além disso, é possível construir um novo produto e utilizar ferramentas ou softwares que já existem disponíveis para uso, sendo pagas ou não.

O resultado final foi satisfatório e pode ser colocado em prática para extrair métricas quantitativas. Com pouco desenvolvimento de código foi possível prover um protótipo de uma ferramenta de busca, capaz de realizar buscas pelo conteúdo falado dos arquivos processados. O protótipo foi capaz de processar 100 horas de áudio em aproximadamente 2 horas e 20 minutos.

A principal contribuição deste trabalho é a arquitetura funcional, que pode permitir melhorias futuras por parte da comunidade acadêmica, servindo como base para

outras pesquisas que estejam ou não relacionadas ao tema proposto neste trabalho. Adicionalmente, pode servir como base ou consulta para a construção de um sistema proprietário.

7. Trabalhos Futuros

Para trabalhos futuros é possível destacar o estudo e melhoria para adicionar mais serviços que possam ser usados ou agregar mais funcionalidades na arquitetura de solução proposta. Também realizar mais pesquisas com outras ferramentas de reconhecimento de fala, que permitam melhorar o tempo de processamento final apresentado neste trabalho.

O protótipo construído foi apenas para testar e provar a arquitetura de solução proposta. O protótipo atual processa apenas arquivos de áudio e o tempo de processamento está ligado à duração do arquivo de áudio, o que significa que se dentre as 100 horas de áudio tivesse um único arquivo de 5 horas ou mais, o tempo total de processamento deste arquivo poderia ser próximo de sua duração total. Isto acontece por conta da ferramenta de reconhecimento de fala utilizada não ter mais velocidade em seu processamento, e a falta da construção de um mecanismo para quebrar o arquivo em arquivos menores para ser processado.

Em relação à arquitetura como um todo é possível melhorar a ferramenta de busca, para permitir a busca por sinônimos das palavras utilizadas ou aplicar um mecanismo de inteligência artificial para aplicar semântica nos dados processados. Por exemplo, em uma busca pela palavra “doenças” seria possível achar um arquivo que fala sobre câncer.

Outra possibilidade é a construção de uma ferramenta que realize um teste prévio no servidor ou máquina que será utilizado para o processamento com o intuito de saber o número máximo de instâncias do processador que o servidor suporta.

É possível implementar um mecanismo para permitir configurar e escalonar automaticamente o sistema conforme a necessidade. Isto pode permitir que qualquer máquina seja utilizada para contribuir com o processamento do sistema e facilitar sua configuração.

Por último, criar um mecanismo para evitar carregar ou processar o mesmo arquivo mais de uma vez, por estar em mais de um local armazenado. Por exemplo, poderia gerar um *hash* (valor único gerado) dos arquivos com base no conteúdo binário

ou no texto extraído. Com isso, é possível comparar este valor com uma base de dados dos arquivos já processados e evitar processar arquivos duplicados.

8. Referências Bibliográficas

Adorf, Julius. "Web Speech API." *Relatório Técnico.* Stockholm: KTH Royal Institute of Technology, 2013.

Algolia. *Algolia Documentation.* 2020. https://www.algolia.com/doc/ (acesso em 09 de Agosto de 2020).

Ali, Abbas. *Sphinx Search beginner's Guide.* Packt Publishing Ltd, 2011.

Amazon CloudSearch. 2020. https://aws.amazon.com/pt/cloudsearch/ (acesso em 18 de Maio de 2020).

Amazon Web Services (AWS). "Streaming Data Solutions on AWS with Amazon Kinesis." Julho de 2017.

Apache Zookeeper. *https://zookeeper.apache.org/.* 2020. https://zookeeper.apache.org/ (acesso em 1 de Agosto de 2020).

Azure Media Services (AMS). *Azure Media Services v3 overview.* 2020. https://docs.microsoft.com/en-us/azure/media-services/latest/media-services-overview (acesso em 19 de Setembro de 2020).

Bresolin, Adriano de Andrade. "Estudo do Reconhecimento de Voz para o Acionamento de Equipamentos Elétricos via Comandos em Português." Joinville, Santa Catarina: Universidade de Santa Catarina (UDESC), 2003.

Carasso, David. *Exploring splunk.* New York: CITO Research, 2012.

Carbone, Paris, Stephan Ewen, Seif Haridi, Asterios Katsifodimos, Volker Markl, e Kostas Tzoumas. "Apache flink: Stream and batch processing in a single engine." *IEEE Computer Society Technical Committee on Data Engineering*, 2015: 28.

Casters, Matt, Roland Bouman, e Jos Van Dongen. *Pentaho Kettle Solutions: Building Open Source ETL Solutions with Pentaho Data Integration.* Indianapolis, Indiana: Wiley Publishing, Inc., 2010.

Chaudhary, Aman, Akshatha Kodlekere, Kiran Kodlekere, e Surya Prasad J. “Keyword Based Indexing of a Multimedia File.” *The 19th IEEE International Symposium on Multimedia*, 2017: 573-576.

DOMO. *Data Never Sleeps 8.* 2020. https://www.domo.com/learn/data-never-sleeps-8 (acesso em 10 de Agosto de 2020).

Eljazzar, Maged M., Afnan Hassan, e Amira A. Al-Sharkawy. “Towards a Time Based Video Search Engine for Al Quran Interpretation.” Cairo: Cairo University, 2017.

Espindola, Luciana da Silveira. “Um Estudo sobre Modelos Ocultos de Markov: HMM – Hidden Markov Model.” Porto Alegre: Universidade Católica do Rio Grande do Sul, 2009.

Eugster, Patrick Th., Pascal A. Felber, Rachid Guerraoui, e Anne-Marie Kermarrec. “The many Faces of Publish/subscribe.” *ACM Computing Surveys*, 2003: 114-131.

Gaida, Christian, Patrick Lange, Rico Petrick, Patrick Proba, Ahmed Malatawy, e David Suendermann-Oeft. “Comparing open-source speech recognition toolkits.” 2014.

Gandomi, Amir, e Murtaza Haider. “Beyond the hype: big data concepts, methods, and analytics.” *International Journal of Information Management.* Toronto, Ontario, Canada: Elsevier Ltd., 2014. 137-144.

Garg, Nishant. *Apache Kafka.* Birmingham: Packt Publishing Ltd, 2013.

Gormley, Clinton, e Zachary Tong. *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine.* O’Reilly Media, Inc., 2015.

Grafana. 2020. https://grafana.com/docs/v4.5/ (acesso em 18 de Maio de 2020).

Gupta, Yuvraj. *Kibana Essentials.* Birmingham: Packt Publishing, 2015.

Haykin, Simon. “Redes Neurais: Princípios e Práticas.” Hamilton, Ontário: Bookman, 2001.

High, Rob. *The Era of Cognitive Systems: An Inside Look at IBM Watson and How It Works.* IBM Watson, 2012.

Hitachi. “Hitachi Content Intelligence: Data Transformation and Exploration Solution.” Setembro de 2017.

Hu, Han, Yonggang Wen, Tat-Seng Chua, e Xuelong Li. “Toward Scalable Systems for Big Data Analytics: A Technology Tutorial.” *Index IEEE Access, vol 2.* 2014. 652-687.

IBM. “IBM annual Report.” *What Will We Make of This Moment?* 2013.

International Data Corporation. “Data Age 2025 Whitepaper. The Digitization of the World from Edge to Core.” 2018.

—. “Worldwide Big Data Technology and Services 2012-2015 forecast.” 2012.

Lakdawala, Burhanuddin, Farhan Khan, Arif Khan, Yash Tomar, Rahul Gupta, e Ashfaq Shaikh. “Voice to Text transcription using CMU Sphinx. A mobile application for healthcare organization.” *Proceedings of the 2nd International Conference on Inventive Communication and Computational Technologies.* IEEE Xplore Compliant, 2018. 749-753.

Lamere, Paul, et al. “The CMU SPHINX-4 speech recognition system.” *Conference on Acoustics, Speech and Signal Processing.* Hong Kong: IEEE, 2003. 2-5.

Lawto, Julien, et al. “A scalable video search engine based on audio content indexing and topic segmentation.” *Proceedings of the Networked and Electronic Media (NEM) Summit*, 2011.

Librivox. 2021. https://librivox.org/pages/about-librivox/ (acesso em 01 de 03 de 2021).

Lopez, Martin Andreoni, Antonio Gonzalez Pastana Lobato, e Otto Carlos M. B. Duarte. “A performance comparison of opensource stream processing platforms.” *IEEE Global Communications Conference (GLOBECOM).* Washington, DC, USA: IEEE, 2016. 1-6.

Martins, José Antonio. “Avaliação de Diferentes Tecnicas para Reconhecimento de Fala.” *Tese de Doutorado.* Campinas: Universidade Estadual de Campinas (Unicamp), 1998.

Matarneh, Rami, Svitlana Maksymova, Vyacheslav V. Lyashenko, e Nataliya V. Belova. "Speech Recognition Systems: A Comparative Review." *IOSR Journal of Computer Engineering (IOSR-JCE).* 2017. 71-79.

MAVIS. *Microsoft Audio Video Indexing Service.* Microsoft. 2020. https://www.microsoft.com/en-us/research/project/mavis/ (acesso em 30 de Março de 2020).

Mozilla. *Project deepspeech.* 2020. https://github.com/mozilla/DeepSpeech (acesso em 22 de Março de 2020).

Murray, Daniel G. *Tableau Your Data!: Fast and Easy Visual Analysis with Tableau Software.* Indianapolis, Indiana: John Wiley & Sons Inc, 2013.

Noghabi, Shadi A., et al. "Samza: Stateful Scalable Stream Processing at LinkedIn." *VLDB Endowment.* 2017. 1634-1645.

Nouza, Jan, et al. "Large-Scale Processing, Indexing and Search System for Czech Audio-Visual Cultural Heritage Archives." *IEEE Multimedia Signal Processing (MMSP).* IEEE, 2012. 337-342.

—. "Speech-to-text technology to transcribe and disclose 100,000+hours of bilingual documents from historical Czech and Czechoslovak radio archive." *Interspeech.* Singapore, 2014. 964-968.

Oussous, Ahmed, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, e Samir Belfkih. "Big Data technologies: A survey." *Journal of King Saud University - Computer and Information Sciences.* 2017. 431-448.

Penchikala, Srini. *Big-Data Processing with Apache Spark.* InfoQ.com, 2017.

Povey, Daniel, et al. "The Kaldi Speech Recognition Toolkit." *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding.* IEEE Signal Processing Society, 2011. 1-4.

Rabiner, L. R., e B. H. Juang. "An Introduction to Hidden Markov Models." *IEEE ASSP Magazine.* IEEE, 1986. 4-16.

Ramakrishnan, Raghu, e Johannes Gehrke. *Database Management Systems.* 2. McGraw Hill Companies, 2000.

RediSearch. *Redis Powered Search Engine.* 2020. https://oss.redislabs.com/redisearch/index.html (acesso em 2020 de Maio de 2020).

Saon, George, e Jen-Tzung Chien. "Large-Vocabulary Continuous Speech Recognition Systems: A look at Some Recent Advances." *IEEE Signal Processing Magazine.* 2012. 18-33.

Shahi, Dikshant. *Apache Solr: An Introduction. Apache Solr.* Apress, Berkeley, CA, 2015.

Sigcha, Erik, José Medina, Francisco Vega, Víctor Saquicela, e Mauricio Espinoza. "Automatic Speech-to-Text Transcription in an Ecuadorian Radio Broadcast Context." *Colombian Conference on Computing.* 2017. 695-709.

Sisense. 2020. https://www.sisense.com/whitepapers/ (acesso em 19 de Maio de 2020).

Sonix. 2020. https://sonix.ai/features (acesso em 1 de June de 2020).

Souza, Daniel da cunha Rodrigues de. *Uma arquitetura de referência para o processamento distribuído de stream de dados em soluções analíticas de near real-time.* Dissertação (Mestrado em Engenharia Elétrica), Brasília: Universidade de Brasília, 2015.

Speechmatics. 2020. https://www.speechmatics.com/resources/product-sheets/ (acesso em 5 de July de 2020).

Stonebraker, Michael, Ugur Çetintemel, e Stan Zdonik. "The 8 requirements of real-time stream processing." *ACM SIGMOD Record.* ACM, 2005. 42-47.

Su, Hang. "Combining Speech and Speaker Recognition - A Joint Modeling Approach." Electrical Engineering and Computer Sciences University of California at Berkeley, 10 de Agosto de 2018.

Telmem, Meryam, e Youssef Ghanou. "Amazigh Speech Recognition System Based on CMUSphinx." *Springer International Publishing AG.* 2018. 397-410.

Tevah, Rafael Teruszkin. *Implementação de um Sistema de Reconhecimento de Fala Contínua com Amplo Vocabulário para o Português Brasileiro.* Dissertação de Pós-Graduação de Engenharia Elétrica, Universidade do Rio de Janeiro, 2006.

Topkara, Mercan, Shimei Pan, Jennifer Lai, Stephen P. Wood, e Jeff Boston. *Tag me while you can: Making online recorded meetings shareable and searchable.* IBM Research Division, 2010.

Toshniwal, Ankit, et al. "Storm @Twitter." *ACM SIGMOD Record.* 2014. 47-56.

Transier, Frederik. *Algorithms and data structures for in-memory text search engines.* Tese de PhD, University of Karlsruhe, 2010.

Troyansky, Oleg, Tammy Gibson, e Charlie Leichtweis. *QlikView Your Business: An Expert Guide to Business Discovery with QlikView and Qlik Sense.* Indianapolis: John Wiley & Sons, 2015.

Vimala, C, e V. Radha. "Speaker Independent Isolated Speech Recognition System for Tamil Language using HMM." *Communication Technology and System Design.* Elsevier Procedia Engineering, 2012. 1097-1102.

Wilbur, W. John, e Karl Sirotkin. "The automatic identification of stop words." *Journal of Information Science.* 1992. 45-55.

Young, Steve, et al. *The HTK Book.* Cambridge University Engineering Department, 2009.

Zobel, Justin, e Alistair Moffat Moffat. "Inverted Files for Text Search Engines." Vol. 38. ACM Computing surveys, 2006. p 56.

Apêndice I – Script para a Configuração do Apache Kafka

Shell script de exemplo para realizar a instalação e a configuração do Apache Kafka. O script realiza as configurações necessárias nos arquivos do Apache Zookeeper e do Apache Kafka. Além disso, o script base debian cria um tópico.

```
#! /bin/bash
sudo apt-get -y update
echo 'Install utils'
sudo apt install -y default-jdk
export JAVA_HOME=/usr/lib/jvm/default-java
sudo apt-get install -y nano
sudo apt-get install -y unrar
sudo apt-get install -y rar
echo 'Install KAFKA'
wget http://www-us.apache.org/dist/kafka/2.4.0/kafka_2.13-2.4.0.tgz
tar xzf kafka_2.13-2.4.0.tgz
mv kafka_2.13-2.4.0 /usr/local/kafka


cat <<EOF > /etc/systemd/system/zookeeper.service
[Unit]
Description=Apache Zookeeper server
Documentation=http://zookeeper.apache.org
Requires=network.target remote-fs.target
After=network.target remote-fs.target

[Service]
```

```
Type=simple

ExecStart=/usr/local/kafka/bin/zookeeper-server-start.sh
/usr/local/kafka/config/zookeeper.properties

ExecStop=/usr/local/kafka/bin/zookeeper-server-stop.sh

Restart=on-abnormal


[Install]

WantedBy=multi-user.target

EOF


cat <<EOF > /etc/systemd/system/kafka.service

[Unit]

Description=Apache Kafka Server

Documentation=http://kafka.apache.org/documentation.html

Requires=zookeeper.service


[Service]

Type=simple

Environment="JAVA_HOME=/usr/lib/jvm/default-java"

ExecStart=/usr/local/kafka/bin/kafka-server-start.sh
/usr/local/kafka/config/server.properties

ExecStop=/usr/local/kafka/bin/kafka-server-stop.sh


[Install]

WantedBy=multi-user.target
```

```
EOF


sudo systemctl daemon-reload

sudo systemctl start zookeeper

sudo systemctl start kafka


/usr/local/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 150 --topic audio-topic
```

Apêndice II – Script para a Configuração do Elasticsearch

Shell script base debian de exemplo para configurar o Elasticsearch no servidor. O script realiza a instalação do sistema e algumas configurações extras como: criação do index, configuração do file mapping e configurações para permitir deletar dados do Elasticsearch.

```
#! /bin/bash
sudo apt-get -y update
echo 'Install utils'
sudo apt install -y default-jdk
export JAVA_HOME=/usr/lib/jvm/default-java
sudo apt-get install -y nano
sudo apt-get install -y unrar
sudo apt-get install -y rar
echo 'Elasticsearch install'
sudo apt-get install -y apt-transport-https
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
add-apt-repository "deb https://artifacts.elastic.co/packages/7.x/apt stable main"
sudo apt-get install -y elasticsearch
sed -i 's/#network.host:.*/network.host: 127.0.0.1/g' /etc/elasticsearch/elasticsearch.yml
sed -i 's/#http.port:.*/http.port: 9200\nhttp.host: 0.0.0.0/g' /etc/elasticsearch/elasticsearch.yml
echo 'Start elasticsearch service'
sudo /bin/systemctl enable elasticsearch.service
sudo systemctl start elasticsearch.service
```

```
curl -XPUT -H "Content-Type: application/json" http://localhost:9200/audio-index

curl -XPUT -H "Content-Type: application/json" http://localhost:9200/audio-index/_mapping -d "{\"properties\": { \"time\": { \"type\": \"date\" }, \"fileAttributes\": {\"type\": \"object\",\"properties\": { \"fileName\": { \"type\": \"text\", \"fielddata\": true }, \"filePath\": { \"type\": \"text\" }, \"creationTime\": { \"type\": \"text\" }, \"size\": { \"type\": \"integer\" }, \"owner\": { \"type\": \"keyword\" }, \"uuid\": { \"type\": \"keyword\" }, \"fileType\": { \"type\": \"keyword\" }, \"tag\": { \"type\": \"keyword\" }, \"duration\": { \"type\": \"integer\" }} }, \"words\": {\"type\": \"nested\",\"properties\": { \"word\": { \"type\": \"keyword\" }, \"occurrences\": { \"type\": \"text\", \"fielddata\": true }, \"occurrenceQuantity\": { \"type\": \"integer\" }} }} }"

curl -XPUT -H "Content-Type: application/json" http://localhost:9200/_all/_settings -d "{\"index.blocks.read_only_allow_delete\": null}"
```

Apêndice III – Configuração Inicial da Base de Dados no Kibana

Para a preparação inicial e começar a executar consultas e criar gráficos, é preciso criar os padrões de índices para o Kibana conseguir encontrar e filtrar os índices criados. Esses padrões permitem filtrar por um ou mais índices para poder realizar as buscas e visualizações, desta maneira o Kibana identifica as bases de dados disponíveis no Elasticseach.

A Figura 25 apresenta onde realizar a criação e visualização dos padrões de índices no Kibana.

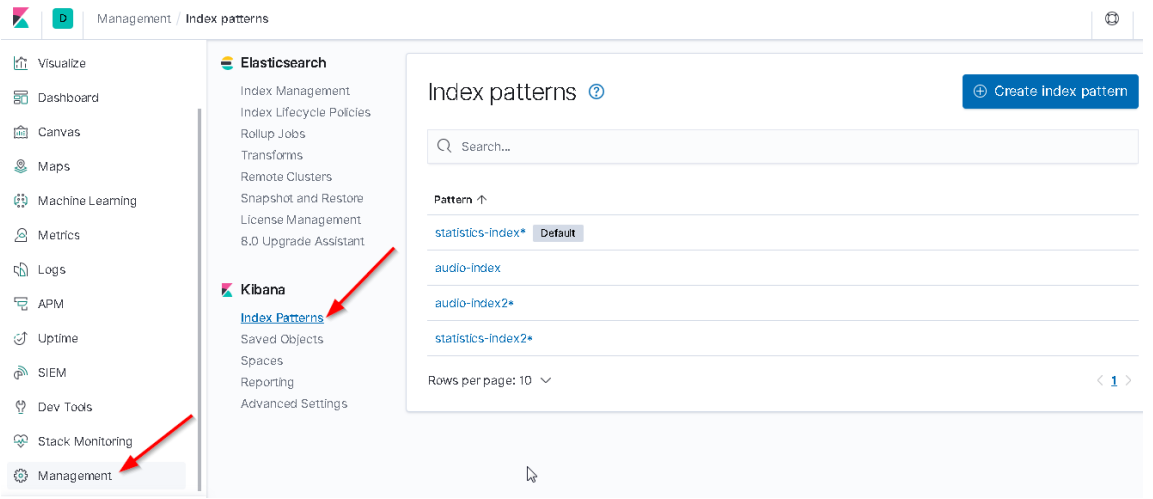

Figura 25. Padrões de Índices.

Ao começar a criar um padrão de índice é preciso fornecer um nome para encontrar o índice criado. Pode ser usado o nome completo ou parcial do índice criado no Elasticsearch. A Figura 26 apresenta a criação de um padrão de índice filtrando por parte do nome do índice “áudio-“.

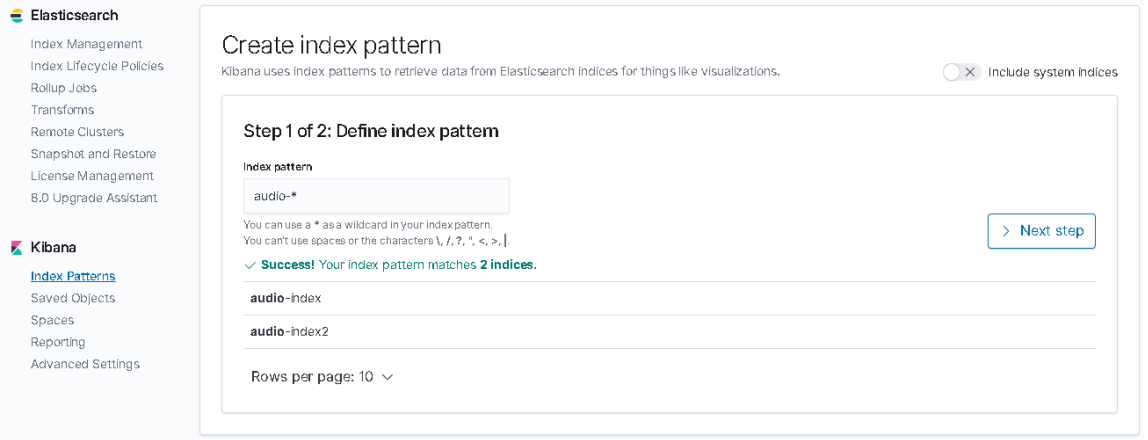

Figura 26. Criação de um Padrão de Índice.

A Figura 27 apresenta o próximo passo para definir um campo para o filtro de tempo, para poder realizar busca e ordenação por este campo. A configuração deste campo não é obrigatória. Entretanto, se não for configurada não será possível filtrar e ordenar os documentos pela data e hora em que os registros entraram no sistema. Para esta solução foi criado um campo chamado time para salvar a data e hora do processamento do arquivo, que pode ser visualizado no mapeamento apresentado na Tabela 13.

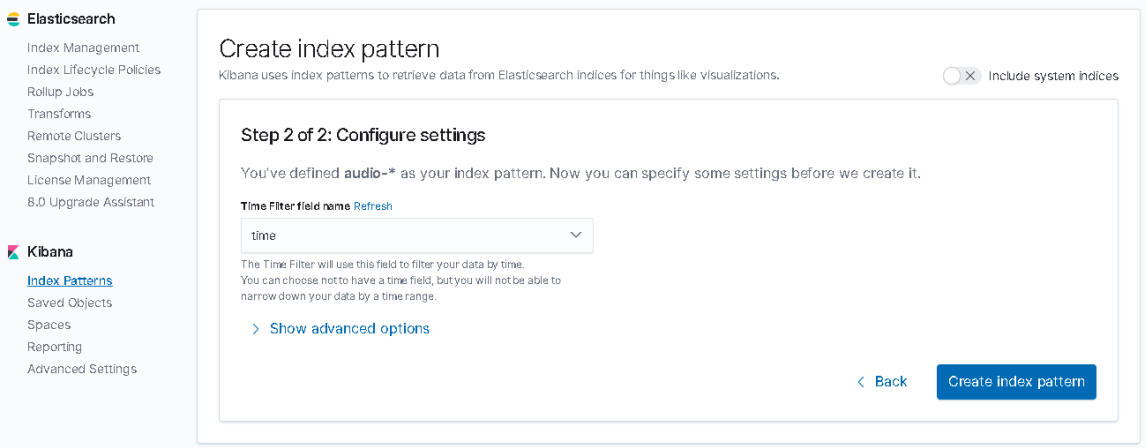

Figura 27 Definição de um Filtro de Tempo.

Após a criação do padrão de índice, é possível visualizar o mapeamento de campos que foi previamente definido.

Os dados salvos no Elasticsearch podem estar salvos em um formato não intuitivo para o usuário ler. Por exemplo, o tempo de processamento está salvo em

segundos e o tamanho dos arquivos processado está em bytes. Para valores grandes pode ficar ruim de ter uma noção da representação desses valores.

O Kibana permite realizar uma formatação dos dados para que na hora que forem apresentados ou visualizados nos gráficos estejam formatados. Por exemplo, ao invés de visualizar 512000 bytes o usuário visualize 50 megabytes. A Figura 28 apresenta o mapeamento e algumas informações como *searchable* (pesquisável), que diz se o campo pode ser pesquisável e *aggregable* (agregável), que diz se o campo permite funções de agregações.

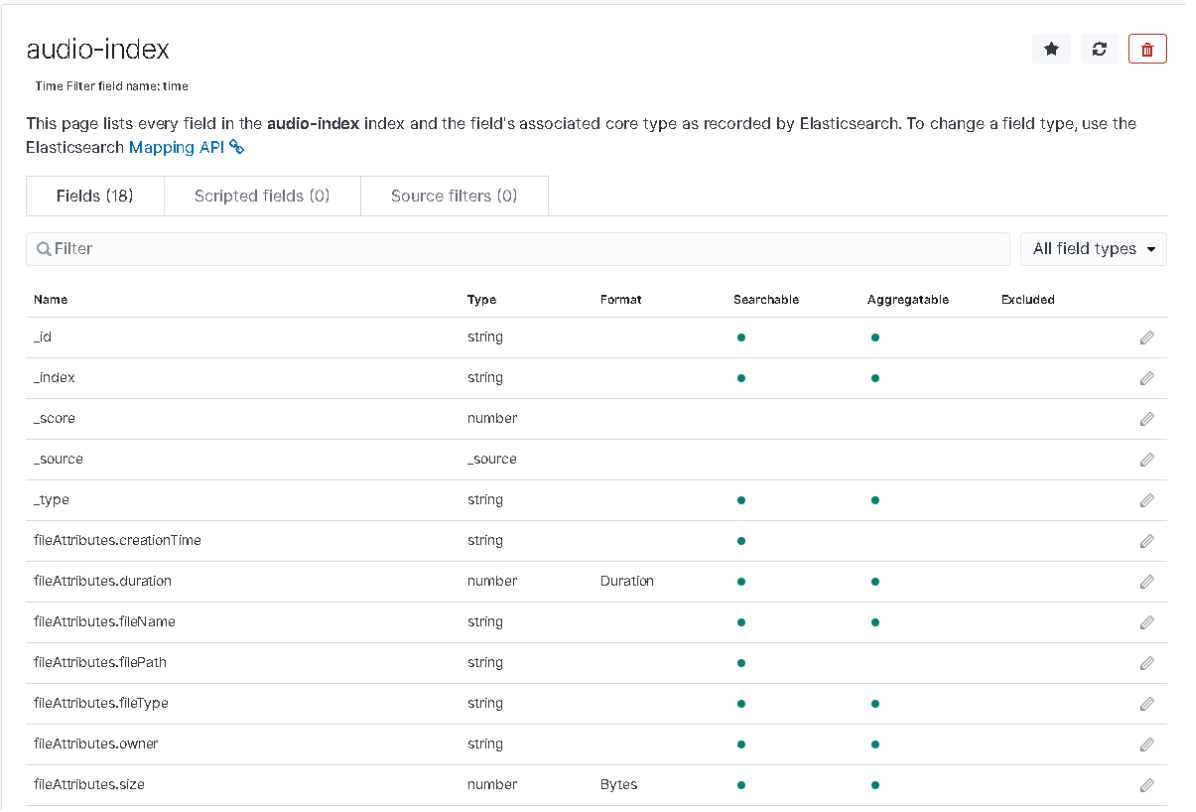

Figura 28. Mapeamento e Formatação dos Campos.

A Figura 29 apresenta a formatação dos campos *duration* (duração) e *size* (tamanho). Ambos os campos em sua forma pura são somente números e não estão formatados.

O campo com o nome *duration* está formatado com o tipo *duration* (duração), e para este formato é necessário fornecer o formato de entrada e de saída do dado (campos *Input format* e *Output format*), para este protótipo o formato de entrada é segundos e output é um formato legível por humanos na Figura 29. O campo *size* está apenas

formatado como bytes. A Figura 29 apresenta alguns exemplos de visualização do dado formatado para o *duration* e o *size*.

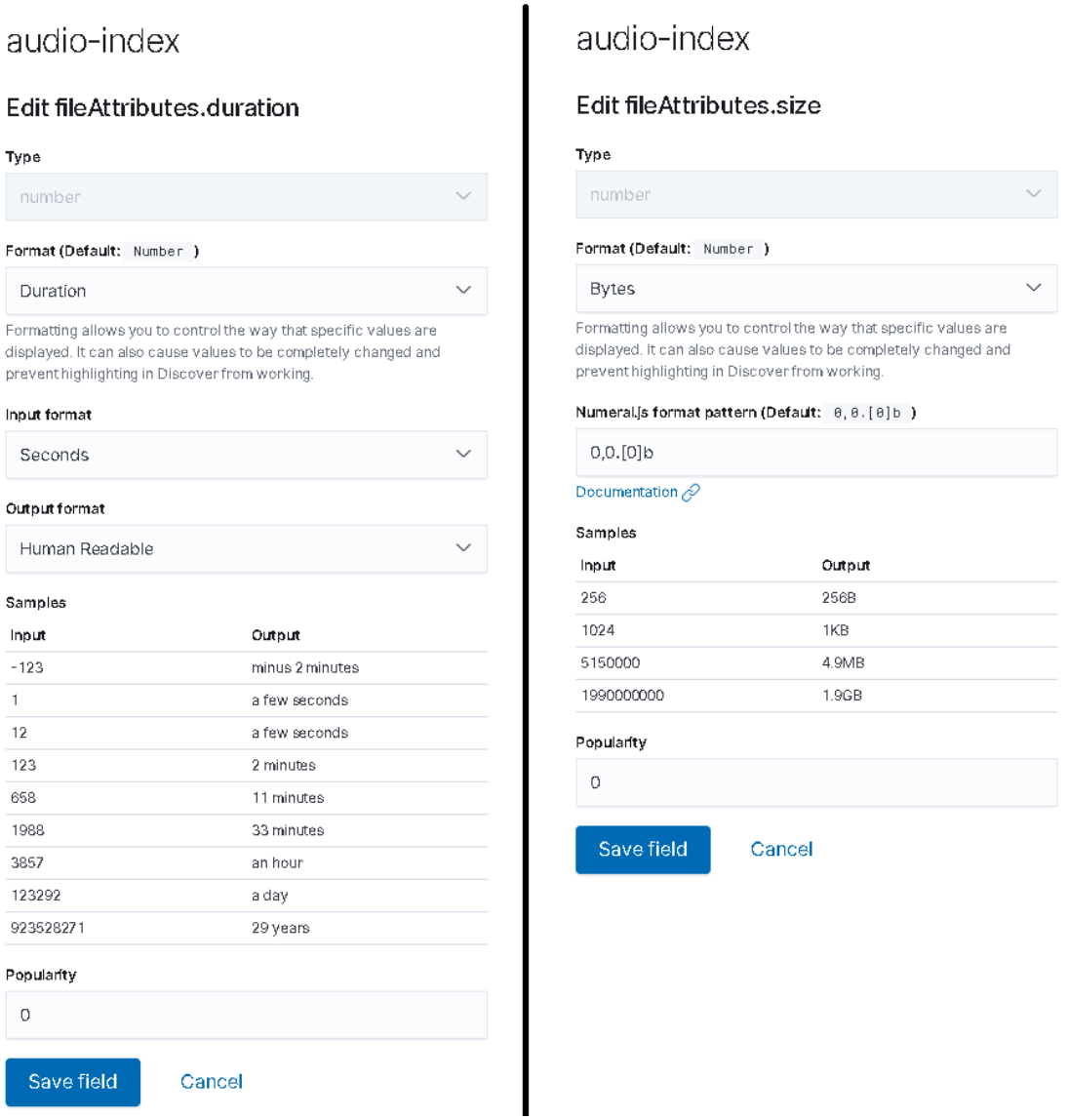

Figura 29. Formatação do campo Duration e do Size.

Nesta etapa o Kibana está pronto para permitir o usuário criar gráficos dinâmicos e realizar pesquisa nos documentos que estão sendo inseridos no Elasticsearch.

Apêndice IV – Pesquisa e Criação de Gráficos no Kibana

O Kibana permite procurar diretamente nos documentos salvos no Elasticsearch. Através dos campos mapeados, o Kibana permite criar queries no Formato Kibana Query Language (KQL); linguagem simplificada mais fácil do que usar a API nativa do Elasticsearch.

A Figura 30 mostra a opção de realizar as buscas no Kibana que se encontra no menu lateral com o nome de *discover*. Basta escolher entre os padrões de índices existentes, por exemplo, audio-index. No campo principal de consulta KQL é possível realizar as consultas. Por exemplo, para procurar por documentos que tem a tag "vídeo-aula" e são arquivos são do tipo ".avi", a consulta utilizada seria:

```
fileAttributes.tag : "video-aula" and fileAttributes.fileName : "*.avi"
```

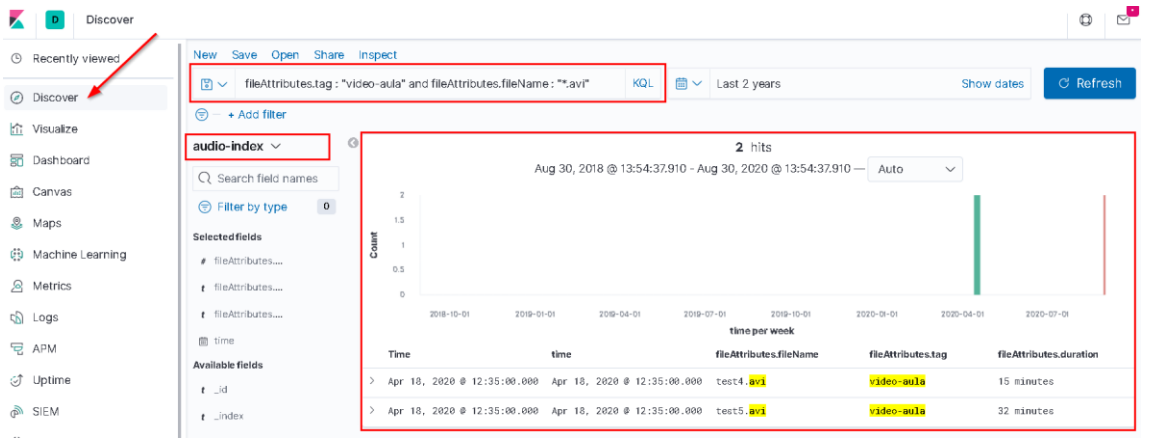

Figura 30. Pesquisa no Kibana.

A consulta retornará todos os campos presentes no documento, mas é possível escolher quais campos serão apresentados na tabela. Com a consulta pronta também é possível salvar a consulta da tabela para ser utilizada em algum *dashboard*.

A Figura 31 mostra como realizar uma busca por duas palavras especificas (look e big) que foram reconhecidas nos arquivos de áudio. Para isto, a seguinte consulta de KQL é utilizada:

```
words:{ word : "look" } and words:{ word : "big" }
```

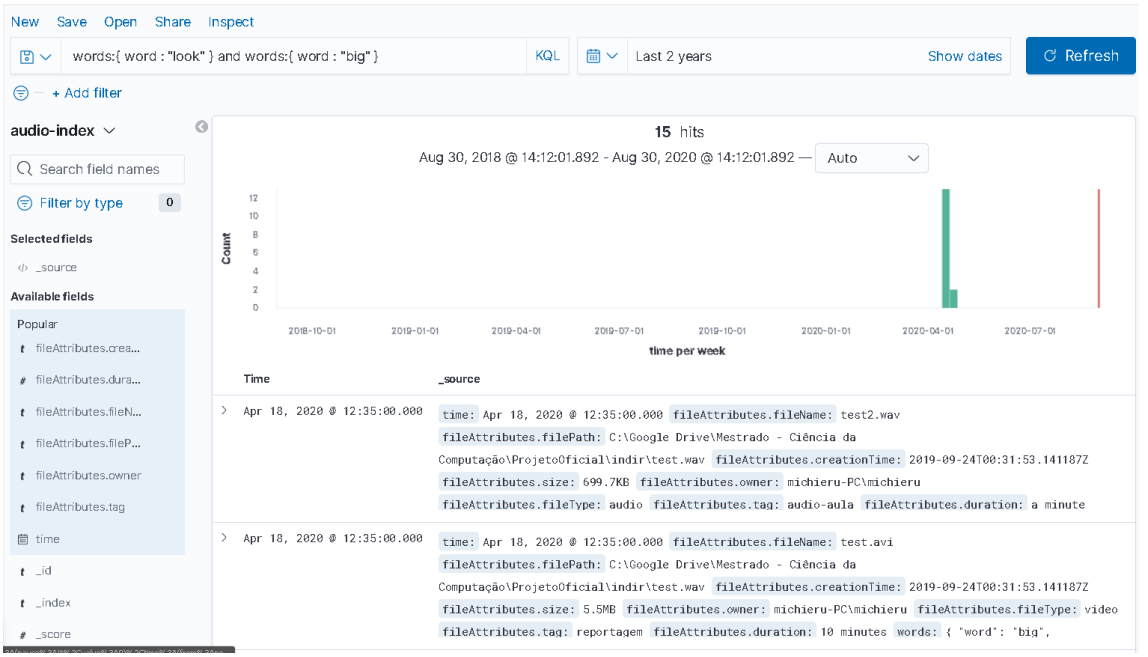

Figura 31. Buscar por Duas Palavras

O Kibana permite criar visualizações a partir de várias opções pré-existentes, por exemplo, gráficos de barra, pizza, gauge, métrica, mapas ou visualizações customizadas usando vega. Para acessar estas opções, basta escolher a opção *visualize* no menu lateral da ferramenta. A Figura 32 apresenta algumas opções de visualizações.

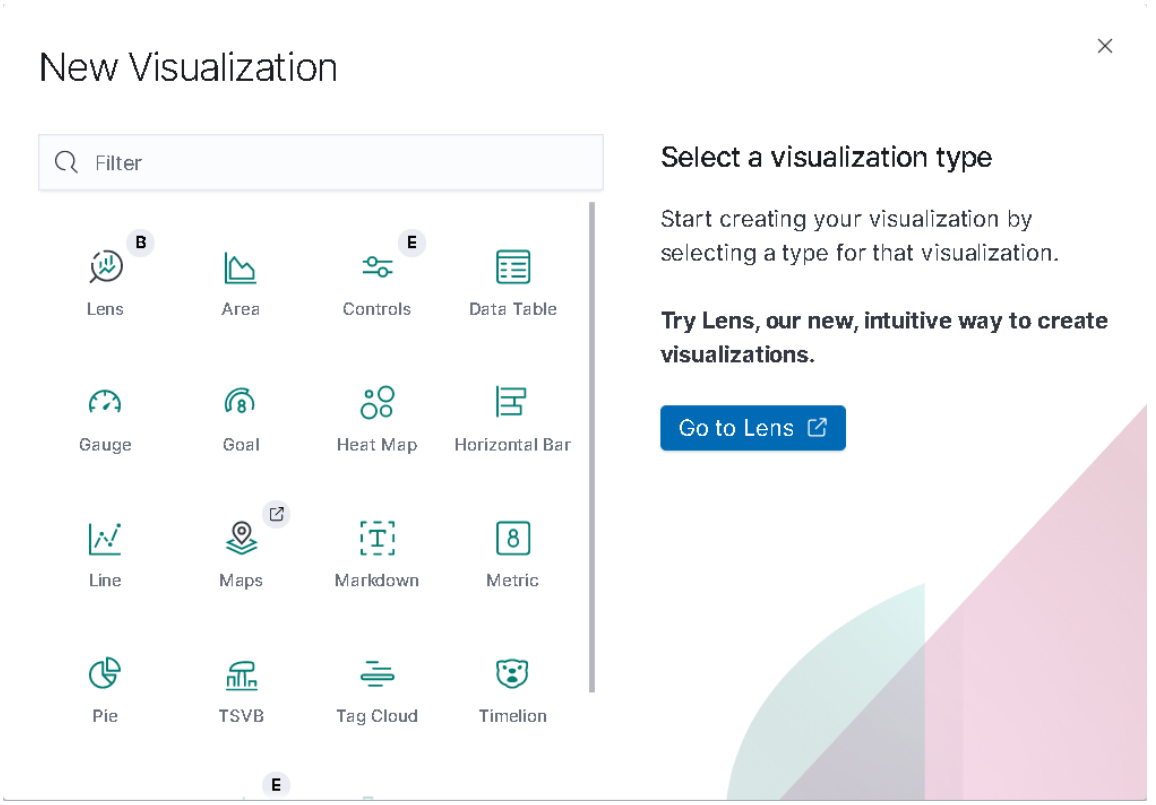

Figura 32. Opções de Visualizações.

Para exemplificar, foi criada uma visualização para somar a quantidade de bytes dos arquivos de áudio presente nos arquivos carregados. M*etric* é um tipo de visualização que mostra um cálculo ou um número único. Além disso, também é possível usar funções de agregações como soma, média, mínimo, máximo entre outros.

A Figura 33 mostra a configuração de uma visualização do tipo *metric*. O primeiro passo é escolher entre um dos padrões de índices já criados. Durante sua criação é possível definir uma data relativa para considerar os documentos, como por exemplo dados de um ano atrás (*a year ago*).

Para realizar a soma em bytes dos arquivos de áudio processados, é preciso selecionar a função de agregação do tipo soma (*sum*), e escolher o campo (*field*) para a função de soma ser aplicada. Neste caso o campo escolhido é o *fileAttributes.size*. Além disso, é possível escolher um nome para o valor, que está presente no campo *Custom Label* como “Soma total em bytes”. Na Figura 33 é possível visualizar as opções descritas.

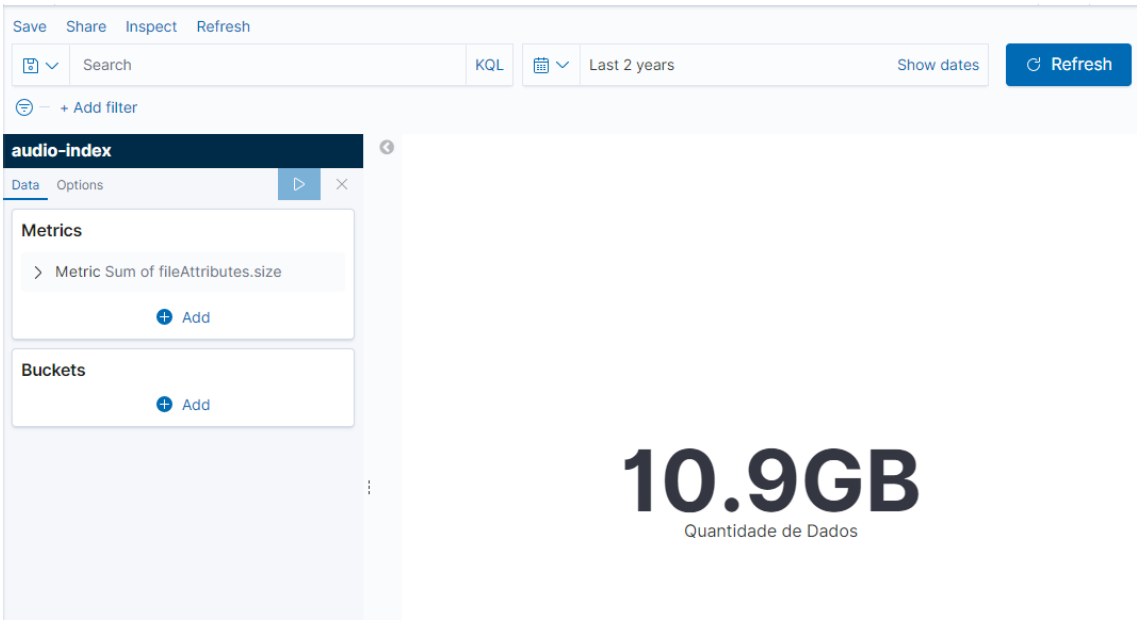

Figura 33. Exemplo de Visualização Metric.

Para o gráfico de barra é possível escolher o tipo de agregação como por exemplo, contagem, soma, média entre outros. Para os gráficos de barra é possível criar as agregações por *buckets* (baldes). Este tipo de agregação não é aplicado simplesmente a um campo, mas sim a documentos agrupados. Por exemplo, criar um gráfico de barras para mostrar as 5 *tags* que mais tem documentos, então ao utilizar *buckets* para cada *tag* é aplicado uma função de agregação de contagem para construir um gráfico de barra. A Figura 34 apresenta a construção do gráfico de barra descrito.

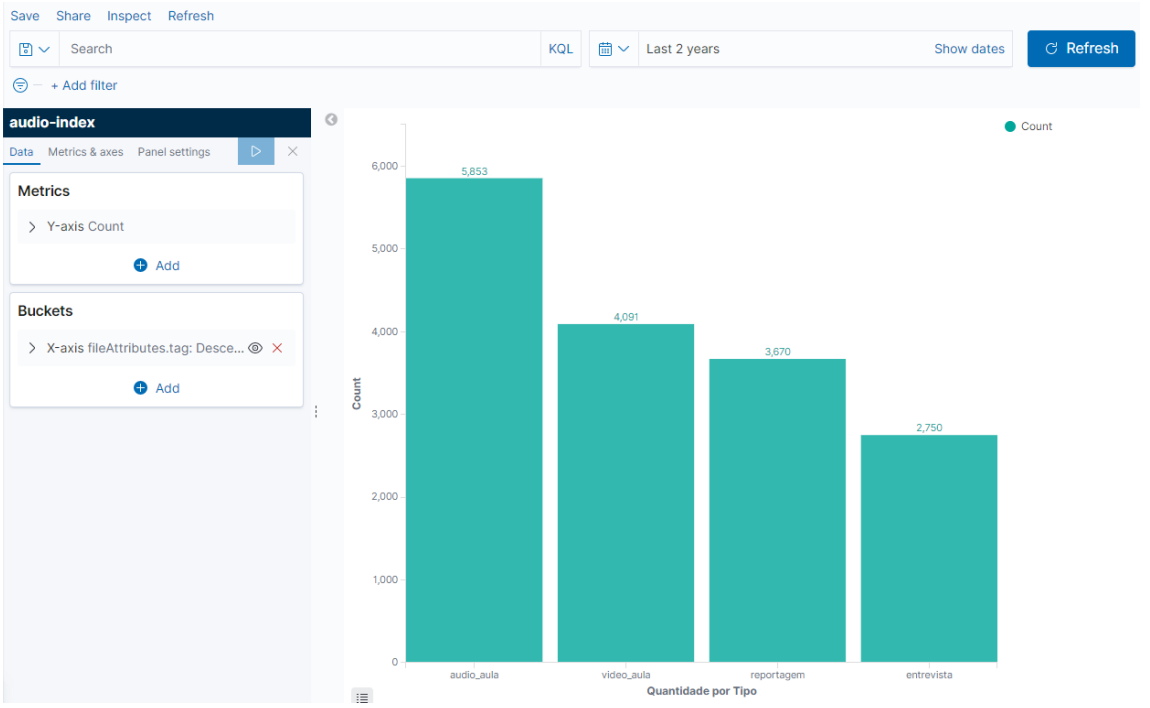

Figura 34. Gráfico de Barra por *Tag*.

O gráfico de pizza segue a mesma estrutura do gráfico de barra de agrupar os dados por *buckets*, para poder aplicar uma função de agregação. Por exemplo a construção de um gráfico de pizza pela quantidade de arquivos processados por tipo (áudio e vídeo). A Figura 35 apresenta um gráfico de pizza com a porcentagem da contagem de arquivos por tipo de arquivo.

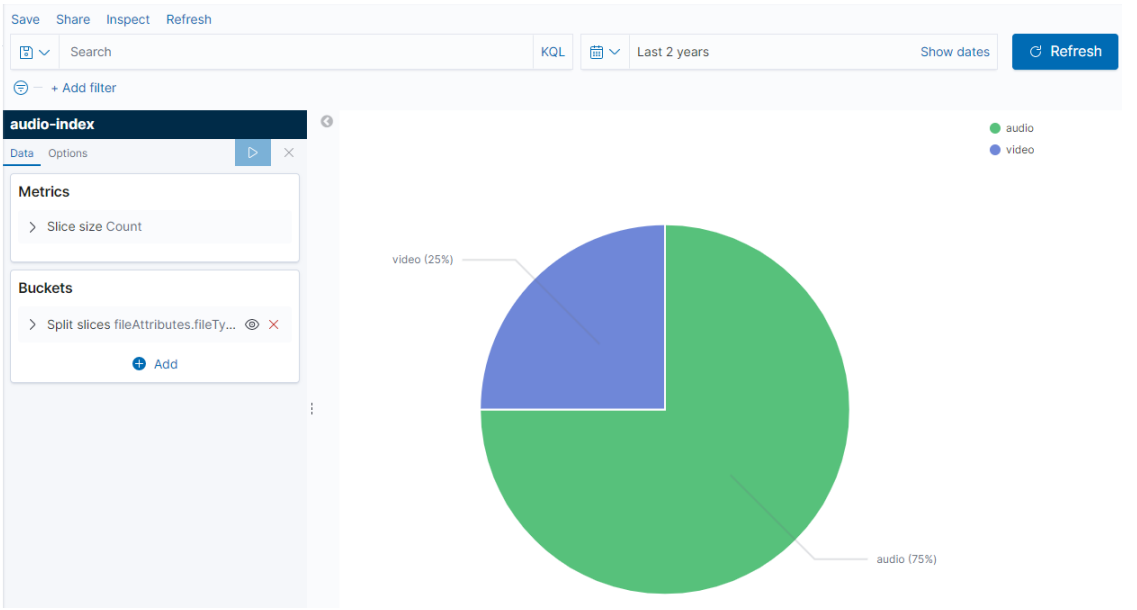

Figura 35. Gráfico de Pizza Quantidade de Arquivos por Tipo.

A pesquisa tabular diretamente no índice, visualização de métrica única, gráfico de barra e o gráfico de pizza apresentados são apenas alguns dos tipos de visualizações disponíveis no Kibana. Entretanto, estes são os tipos de visualizações utilizados neste protótipo, com o objetivo de apresentar o resultado dos arquivos processados.