UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ARTHUR DO REGO BARROS MENDONÇA

Avaliação Experimental de uma Arquitetura de Microsserviços para o Gerenciamento de Notas Fiscais Eletrônicas

Recife

2022

ARTHUR DO REGO BARROS MENDONÇA

Avaliação Experimental de uma Arquitetura de Microsserviços para o Gerenciamento de Notas Fiscais Eletrônicas

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco como requisito parcial para a obtenção do título de Mestre.

Área de Concentração: Banco de Dados
Orientador (a): Valéria Cesário Times

Recife

2022

Catalogação na fonte
Bibliotecária Nataly Soares Leite Moro, CRB4-1722

M539a Mendonça, Arthur do Rego Barros
Avaliação experimental de uma arquitetura de microsserviços para o gerenciamento de notas fiscais eletrônicas / Arthur do Rego Barros Mendonça. – 2022.
117 f.: il., fig., tab.

Orientadora: Valéria Cesário Times.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2022.
Inclui referências e apêndices.

1. Banco de dados. 2. NoSQL. 3. REST. 4. API. 5. Nota fiscal eletrônica. I. Times, Valéria Cesário (orientadora). II. Título

025.04 CDD (23. ed.) UFPE - CCEN 2022 – 108

ARTHUR DO REGO BARROS MENDONÇA

"Avaliação Experimental de uma Arquitetura de Microsserviços para o Gerenciamento de Notas Fiscais Eletrônicas"

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Banco de Dados.

Aprovado em: 08/03/2022.

BANCA EXAMINADORA

Profa. Dra. Ana Carolina Brandão Salgado

Centro de Informática / UFPE

Prof. Dr. Ricardo Rodrigues Ciferri

Departamento de Computação / UFSCar

Profa. Dra. Valéria Cesário Times
Centro de Informática / UFPE

(Orientador)

A todos que acreditaram.

AGRADECIMENTOS

Agradeço a meus pais, sem quem nada disso teria sido possível e a Rebeca, pelo apoio de sempre. A Valéria, que indicou caminhos inexplorados e foi fonte de conhecimento fundamental para o desenvolvimento deste trabalho. A Rodrigo Mateus, meu co-orientador informal, e a Vitória, que não me deixou perder nenhum prazo. Também agradeço a Bethânia, Fábio e ao restante dos colegas do TCE-PE, pelo apoio e auxílio na viabilização desta pesquisa.

Even beauty diminishes with study. It is better to glance than gawk. (BANCROFT, 2013)

RESUMO

Notas fiscais eletrônicas são documentos digitais no formato XML que registram operações de circulação de mercadorias ou prestação de serviços. Por meio de convênios de cooperação, os órgãos de arrecadação e controle fiscal têm intensificado o compartilhamento das notas fiscais que envolvem compras governamentais com os órgãos de controle da Administração Pública, como tribunais de contas e o Ministério Público. No entanto, o gerenciamento desses dados em SGBDs relacionais se mostra desafiador, principalmente pelo volume de dados gerado e pela variedade de formatos da NF-e, em cujo leiaute estão previstos campos multivalorados e opcionais. O leiaute é frequentemente modificado, o que leva à necessidade de retrabalho na modelagem dos dados. Neste trabalho, se descreve a arquitetura ControleNF, uma arquitetura que utiliza microsserviços, uma REST API e SGBD NoSQL para o gerenciamento das notas fiscais por órgãos de controle. A arquitetura é avaliada do ponto de vista qualitativo, através dos critérios de manutenibilidade e portabilidade, previstos no ISO/IEC 25010, e quantitativo, em que os aspectos de desempenho e escalabilidade são mensurados através de uma avaliação experimental. Embora a avaliação qualitativa aponte possíveis ganhos relativos à facilidade de manutenção e à portabilidade da arquitetura, o desempenho mensurado no experimento foi consideravelmente inferior àquele observado na arquitetura tradicional que utiliza SGBDs relacionais. Uma investigação detalhada é realizada e possíveis causas da perda de desempenho são relatadas.

Palavras-chave: microsserviços; NoSQL; REST; API; XML; nota fiscal eletrônica; NF-e.

ABSTRACT

Nota Fiscal Eletrônica (NF-e) is a kind of electronic invoice used in Brazil for registering the sale of goods or the providing of services. NF-e are stored as XML documents and are mainly used for tax collection purposes, but are also useful as secondary information sources for oversight activities conducted by control institutions such as public audit institutions and the Public Ministry. However, the management of NF-e data in traditional architectures that use relational DBMS can be challenging, due mainly to the volume of generated data and the variety of formats in which NF-e can be structured, since there are many multivalued and optional fields in its official layout. This layout is also frequently changed, which might lead to remodeling of the data schema. In this work, software architecture ControleNF is described, which is composed by microservices, a REST API and a NoSQL DBMS in order to manage NF-e in control institutions. The architecture is evaluated with relation to qualitative criteria, more specifically maintenability and portability, which are present in ISO/IEC 25010 software quality model. It is also evaluated with regards to the quantitative aspects of performance and scalability, which are measured in an experimental analysis. Although the initial qualitative analysis shows possible benefits in maintanability and portability, the performance of queries and insertions measured in the experimental analysis was considerably worse than traditional architectures that use relational databases. A detailed investigation is conducted in order to find possible causes for this loss of performance and its results are listed in this work.

Keywords: microservices; NoSQL; REST; API; XML; nota fiscal eletrônica; NF-e; electronic invoice.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CF/88	Constituição Federal de 1988
DSRF	*Design Science Research Framework*
ENCAT	Encontro Nacional de Coordenadores e Administradores Tributários Estaduais
ICMS	Imposto sobre Circulação de Mercadorias e Prestação de Serviços
IDL	linguagem de descrição de interface
IPI	Imposto sobre Produtos Industrializados
MOC	Manual de Orientação ao Contribuinte
MVC	Model-View-Controller
NF-e	Nota Fiscal Eletrônica
ORM	Object Relational Mapper
PIB	Produto Interno Bruto
REST	Representational State Transfer
RFB	Receita Federal do Brasil
RPC	*Remote Procedure Call*
SEFAZ	Secretaria da Fazenda
SEFAZ-PE	Secretaria da Fazenda do Estado de Pernambuco
SGBD	Sistema de Gerenciamento de Banco de Dados
SIASG	Sistema Integrado de Administração de Serviços Gerais
SOA	Service-oriented Architecture
TCE-PE	Tribunal de Contas do Estado de Pernambuco
UASG	Unidade de Administração de Serviços Gerais
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Na organização federativa brasileira, consagrada pela Constituição Federal de 1988 (CF/88) (BRASIL, 1988), a União, os Estados, o Distrito Federal e os Municípios são entes dotados de autonomia política, administrativa e financeira. Nesse sentido, os entes têm autonomia para criar e regular seus tributos, arrecadá-los e empregar os recursos oriundos da arrecadação. Nesse contexto de descentralização fiscal, o projeto da Nota Fiscal Eletrônica surgiu de um esforço de modernização e integração da Administração Tributária brasileira, ao prever a implantação de um modelo único nacional de documento fiscal em formato eletrônico (RECEITA FEDERAL DO BRASIL, 2022).

A Nota Fiscal Eletrônica (NF-e) é um documento estruturado no formato Extensible Markup Language (XML), que contém um conjunto de informações suficientes para descrever uma operação comercial. A NF-e substitui a emissão do documento fiscal em papel e é assinada digitalmente pelo seu emissor, contribuindo para a simplificação do processo de fiscalização tributária e redução de custos no processo de emissão e controle das notas. Após sua emissão por algum aplicativo gerador, as notas são assinadas pelo fornecedor do bem ou prestador do serviço em questão e são posteriormente enviadas à Secretaria da Fazenda (SEFAZ) da jurisdição do emitente, esta responsável pela validação da estrutura da NF-e e pela autorização de uso. As secretarias de fazenda, neste processo, armazenam em seus bancos de dados as notas fiscais por elas autorizadas.

As notas fiscais, por descreverem de forma completa as operações comerciais e suas partes, têm forte valor como fonte secundária de informação para órgãos de controle externo da Administração Pública, como os Tribunais de Contas, órgãos responsáveis pela fiscalização contábil, financeira, orçamentária, operacional e patrimonial dos entes federativos. Esses órgãos, dentre outras atribuições determinadas pela CF/88, são responsáveis por verificar a legalidade, a legitimidade e a economicidade dos atos e contratos administrativos dos órgãos e entidades submetidos a seu controle de contas. Dentre esses atos e contratos, se encontram as contratações de pessoas físicas ou jurídicas privadas, para a aquisição de bens ou prestação de serviços, operações que são registradas em notas fiscais.

Nesse sentido, vários Tribunais de Contas têm firmado convênios de cooperação com as

secretarias de fazenda dos seus respectivos estados para a obtenção das notas fiscais cujo destinatário é algum de seus órgãos ou entidades jurisdicionados. Entre os Tribunais de Contas que contam com esse compartilhamento por parte das SEFAZ, estão Tribunal de Contas do Estado do Mato Grosso (SEFAZ-MT, 2008), Tribunal de Contas do Estado da Paraíba (SEFAZ-PB, 2017), o Tribunal de Contas do Estado de Minas Gerais (SIMõES; CASTRO, 2017) e o Tribunal de Contas do Estado de Pernambuco (TCE-PE, 2016). A partir da disponibilização dessas notas, o órgão passa a ter uma fonte confiável e tempestiva de informações sobre as aquisições realizadas pelos seus jurisdicionados, servindo de base para seus trabalhos de fiscalização. Os auditores podem analisar diferenças entre preços praticados entre órgãos, verificar a realização de aquisições sem lastro em licitação, confrontar as informações fiscais com aquelas declaradas pelos órgãos contratantes em seus portais da transparência, dentre outras possibilidades de análise (SIMõES; CASTRO, 2017; LEAL; MAHLER, 2016).

Embora sejam capazes de auxiliar os órgãos de controle na fiscalização das compras públicas, as notas fiscais também podem representar um desafio do ponto de vista do gerenciamento de dados. Apesar de existir uma estrutura única para a escrituração, ou seja, para a emissão das notas, não existe uma arquitetura padronizada para o gerenciamento das notas fiscais compartilhadas, processo que inclui o compartilhamento entre a SEFAZ e o órgão de controle correspondente, além das atividades de inserção, remoção, alteração e consulta das notas em um banco de dados (BD).

Esse gerenciamento deve levar em consideração alguns aspectos inerentes às notas fiscais, como o volume de dados e a variedade de formatos com que elas podem se apresentar. Cada licitação conduzida por um órgão público pode resultar em múltiplas aquisições, e cada aquisição terá uma nota fiscal correspondente. O leiaute da nota fiscal ainda prevê um conjunto extenso de elementos XML, que podem ou não estar presentes em cada nota a depender das regras de preenchimento, da natureza da operação, do tipo de tributação incidente sobre o produto comercializado, entre outras dezenas de regras consolidadas no Manual de Orientação ao Contribuinte (ENCAT, 2020).

Em sistemas computacionais tradicionais de gerenciamento de notas fiscais, como o que é utilizado pelo TCE-PE, por (ANDRADE, 2013) e (SANTOS NETO, 2018), é comum a transformação dos dados XML das notas fiscais para o formato de tuplas a serem inseridas em tabelas de um SGBD relacional (SGBDR). Essa utilização de SGBDs relacionais traz algumas limitações relativas às características das notas fiscais, como a falta de flexibilidade de esquema para se adaptar às mudanças de leiaute da NF-e. Também é possível citar problemas inerentes

aos SGBDs relacionais, como a falta de escalabilidade horizontal e perdas de desempenho em consultas que envolvem muitas junções (LEAVITT, 2010).

Neste trabalho, investiga-se a adoção e o desempenho de uma arquitetura de microsserviços chamada ControleNF, que utiliza uma REST API para comunicação com aplicações externas e a utilização de um SGBD NoSQL orientado a documentos para armazenamento de notas fiscais. A arquitetura objetiva reduzir o esforço de manutenção de soluções existentes, ao evitar modificações e reprocessamentos a cada mudança de leiaute da nota fiscal, e ampliar sua portabilidade, permitindo que seja aplicada em diversos órgãos de controle sem que isso implique grandes adaptações. Essas características são avaliadas individualmente em relação a cada arquitetura (tradicional e ControleNF), com a utilização do modelo de qualidade de software ISO/IEC 25010 (ISO/IEC, 2011). Após esta avaliação qualitativa preliminar, que indica benefícios em relação aos quesitos avaliados, é realizada a implementação de um protótipo da arquitetura e uma avaliação experimental, com o objetivo de se verificar o desempenho e a escalabilidade da arquitetura de microsserviços investigada.

1.2 MOTIVAÇÃO

De acordo com o Portal da Nota Fiscal Eletrônica (RECEITA FEDERAL DO BRASIL, 2022), até fevereiro de 2022, mais de 32 bilhões de notas fiscais, emitidas por cerca de 2 milhões de fornecedores distintos, já haviam sido autorizadas pelos órgãos fazendários. Dentre essas notas, uma quantidade considerável corresponde a operações de venda de mercadorias e prestação de serviços a órgãos e entidades da Administração Pública, uma vez que a emissão desse tipo de documento é obrigatória para esse tipo de operação (CONFAZ, 2009) e que compras governamentais ocorrem em grande volume, correspondendo a cerca de 12,5% do Produto Interno Bruto (PIB) brasileiro (IPEA, 2018).

Além do aspecto volume, tem-se que o leiaute padrão da NF-e também sofre frequentes alterações. Esse leiaute, ao final de 2021, estava na versão 4.00 (ENCAT, 2020). Esta versão da NF-e compreende mais de 400 campos distintos, que podem ou não estar presentes nas notas, de acordo com o tipo da operação, tributação, emitente, destinatário, dentre outras regras. As versões numeradas dos leiautes compreendem conjuntos de alterações nos campos e regras da NF-e descritas em diversas Notas Técnicas, publicadas periodicamente pela Coordenação Técnica do Encontro Nacional de Coordenadores e Administradores Tributários Estaduais (ENCAT). Esses normativos são frequentes e sofrem alterações ao longo do tempo.

Em 2020, por exemplo, foram publicadas sete notas, de numeração 2020.001 a 2020.007. A Nota Técnica nº 2020.005, desde então, já teve publicadas suas versões 1.00, 1.10, 1.20 e 1.21, esta última tendo sido emitida em outubro de 2021, o que ilustra a frequência com que alterações são realizadas.

O volume e a variedade de formatos das notas fiscais trazem dificuldades em seu gerenciamento por SGBDs relacionais. As regras de normalização empregadas em esquemas de BD relacionais implicam a criação de múltiplas tabelas, o que torna o modelo mais complexo do que aquele em que as notas são armazenadas em formato nativo. Além disso, cada alteração realizada no leaiute implica a alteração das tabelas correspondentes e do procedimento de carga, além da necessidade do reprocessamento das notas fiscais para a carga dos novos campos ou de regras no SGBD.

Na literatura pesquisada, não foi identificada qualquer arquitetura padrão para o gerenciamento de dados de notas fiscais eletrônicas por órgãos de controle. Identificou-se alguns trabalhos relativos ao gerenciamento de dados de notas fiscais (DEBASTIANI, 2017; GOMES, 2018; SANTOS NETO, 2018; Brandão Filho; ALVES; BERTONCINI, 2018; SANTOS, 2015; ANDRADE, 2013) e que alguns deles usam SGBDs NoSQL e REST APIs para o seu gerenciamento (DEBASTIANI, 2017; GOMES, 2018). No entanto, nenhum deles descreve uma arquitetura de microsserviços ou realiza uma análise comparativa de desempenho que leve em consideração arquiteturas existentes.

1.3 OBJETIVOS

Esta pesquisa tem como objetivo geral investigar o desempenho da criação e do processamento de consultas sobre diferentes volumes de bases de NF-es, utilizando arquiteturas baseadas em microsserviços, sistemas de banco de dados orientados a documentos, e arquiteturas de SGBD relacionais tradicionais. Para alcançar esse objetivo geral, os seguintes objetivos específicos foram definidos:

1. Uso do Design Science Research Framework (DSRF) para seguir as etapas recomendadas na condução de uma pesquisa científica cujo objeto é o design de um sistema de informação. Nesta dissertação, deseja-se investigar o design de um sistema computacional para gerenciamento de NF-es.

2. Especificação e implementação de uma arquitetura de software, chamada de ControleNF, para o gerenciamento de NF-es em órgãos de controle, que facilite a comunicação com

aplicações externas e que use microsserviços e sistemas de BD NoSQL para obtenção de adaptabilidade e flexibilidade da manutenção de esquemas de dados de NF-es.

3. Construção de um gerador de NF-es para avaliação experimental de aplicações de gerenciamento de NF-es mantidas por órgãos de controle. Esse gerador deve produzir diferentes volumes de bases de NF-es semelhantes às NF-es reais.

4. Avaliação da arquitetura de software que tem sido adotada em um órgão de controle do Estado de Pernambuco e que se baseia em SGBDR e sua comparação com a arquitetura ControleNF. Avaliar essas arquiteturas de software em termos de manutenibilidade, portabilidade, desempenho e escalabilidade.

1.4 ESTRUTURA DO DOCUMENTO

Além desta introdução, este documento tem a seguinte estrutura. No Capítulo 2, são apresentados os fundamentos teóricos sobre as tecnologias de microsserviços, REST APIs, RPCs, NoSQL e o modelo C4 para visualização de software. Além disso, é feita uma introdução ao projeto da Nota Fiscal Eletrônica e às características desses documentos. No Capítulo 3, são apresentados trabalhos semelhantes ao objeto desta dissertação. Já no Capítulo 4, o processo de pesquisa do framework Design Science Research é introduzido e seu artefato, a arquitetura ControleNF, é apresentado. No Capítulo 5, a avaliação experimental da arquitetura é realizada, em comparação com a solução existente no ambiente do TCE-PE. Por fim, o Capítulo 6 contém as considerações finais a respeito da pesquisa aqui desenvolvida e indica direções para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo fornecer ao leitor o embasamento teórico necessário à compreensão dos conceitos tratados nesta dissertação, possibilitando o entendimento do trabalho realizado.

Em primeiro lugar, a Seção 2.1 define o que são microsserviços e como as arquiteturas compostas por microsserviços estão organizadas. A Seção 2.2, por sua vez, discute sobre o que são APIs e como elas se encaixam nas arquiteturas de microsserviços, além de especificar o que são as REST APIs. A Seção 2.3 intruduz o conceito de RPCs e protobufs, utilizados na comunicação entre microsserviços. Em seguida, a Seção 2.4 apresenta os sistemas de bancos de dados NoSQL e suas características principais, além de realizar uma breve introdução ao MongoDB, sistema de gerenciamento de bancos de dados orientado a documentos. A Seção 2.5 descreve o modelo C4, utilizado para representar arquiteturas de software. A Seção 2.6 descreve o modelo de qualidade de software ISO/IEC 25010, utilizado na avaliação da arquitetura descrita nesta pesquisa. Por fim, a Seção 2.7 descreve as características do projeto da NF-e, cujos dados serão gerenciados no âmbito da arquitetura descrita neste documento.

2.1 MICROSSERVIÇOS

Microsserviços, segundo (NEWMAN, 2015), são pequenos serviços autônomos que trabalham de forma conjunta. Essa definição destaca três aspectos principais que caracterizam esses serviços.

Em primeiro lugar, um microsserviço é um serviço pequeno, ou seja, é um serviço de escopo limitado, especializado na realização de uma tarefa específica. Newman se preocupa em indicar que o termo "pequeno" está relacionado não somente ao número de linhas de código, mas principalmente à coesão. Essa coesão se refere à imposição de limites claros ao escopo dos serviços. Assim, um microsserviço deve prover uma funcionalidade bem delimitada, sem extrapolar um campo restrito de ação.

Pode-se destacar também o aspecto da definição que descreve os microsserviços como autônomos. Esse termo ressalta o fato de que os microsserviços presentes em uma arquitetura devem possuir um baixo grau de acoplamento e de dependência entre si, o que permite que eles possam evoluir e serem implantados de forma relativamente independente. Cabe ressaltar que

essa autonomia e a independência dos microsserviços costumam se traduzir em implementações do tipo caixa preta, ou seja, aquelas em que os demais componentes conhecem e acessam os microsserviços somente em função de suas entradas e saídas, sem a exposição de suas estruturas internas.

Outra característica importante é que os microsserviços trabalham de forma conjunta. Isso é necessário para que os pequenos serviços que executam atividades especializadas possam ter suas capacidades combinadas, provendo a funcionalidade global do sistema.

Para complementar o entendimento do que são os microsserviços, pode-se também considerar a definição de Dragoni et al., que definem um microsserviço como um processo independente e coeso que realiza interações via mensagens (DRAGONI et al., 2017). Essa definição é similar à anterior no sentido de que também foca nos aspectos de independência e coesão dos microsserviços. Um serviço que é independente tem baixo acoplamento em relação aos demais e pode funcionar de forma isolada. Já a coesão pode ser entendida como aquela característica da arquitetura em que todas as partes de código fortemente relacionadas entre si estão agrupadas em um só serviço. Além disso, o mecanismo de comunicação entre os serviços se dá por meio de troca de mensagens. Para essas trocas de mensagens, costuma-se utilizar o protocolo HTTP ou sistemas leves de filas de mensagens assíncronas (LEWIS; FOWLER, 2014).

A arquitetura de microsserviços também pode ser definida quando tomada em contraste com o estilo arquitetural tradicional monolítico, aquele em que as aplicações são construídas como uma só unidade (LEWIS; FOWLER, 2014). A utilização de monólitos está comumente associada à arquitetura tradicional cliente-servidor, em que o servidor corresponde a uma unidade lógica que contém toda a funcionalidade do negócio.

Arquiteturas de microsserviços se distinguem das arquiteturas monolíticas, mesmo aquelas orientadas a serviços, de várias maneiras, tais como (DRAGONI et al., 2017):

- Microsserviços implementam um escopo limitado de funcionalidades, o que os torna mais fáceis de se manter. Dada essa limitação e o tamanho do código, que tende a ser bem menor do que o de um monólito, *bugs* podem ser encontrados com maior facilidade e testes podem ser facilmente realizados em funcionalidades específicas de forma isolada.

- A arquitetura de microsserviços possibilita uma espécie de integração contínua no seu processo evolutivo, o que ocorre quando se implanta versões distintas do mesmo serviço lado a lado. Assim, a transição de uma versão para a outra pode ser feita de forma gradual, conectando-se os demais módulos pouco a pouco à nova versão.

- Enquanto a escalabilidade de monólitos tende a ser realizada com a replicação de todos os seus componentes, em uma arquitetura de microsserviços é possível adaptar somente funcionalidades específicas à escala adequada, se implantando ou removendo instâncias dos microsserviços correspondentes conforme a necessidade requerida.

- Microsserviços estão comumente associados à utilização de contêineres, o que permite a utilização de ambientes com diferentes configurações para diferentes serviços que tenham requisitos diversos. De maneira similar, microsserviços distintos em uma mesma arquitetura podem utilizar frameworks, bibliotecas e até mesmo linguagens de programação diferentes. A única restrição tende a ser a utilização do mesmo protocolo de comunicação entre os diferentes serviços.

Ainda segundo (DRAGONI et al., 2017), pode-se resumir que as arquiteturas de microsserviços apresentam três principais características: flexibilidade (microsserviços devem se adaptar facilmente a mudanças no ambiente de negócio), modularidade (os sistemas de microsserviços são compostos por componentes individuais isolados que, de forma conjunta, provêm a capacidade completa do sistema) e evolução (deve ser fácil realizar a manutenção e adição de novas funcionalidades no sistema).

É relevante ressaltar que muitos dos benefícios das arquiteturas de microsserviços aqui elencados estão relacionados à própria definição de serviços. O estilo arquitetural Service-oriented Architecture (SOA) precede o conceito de microsserviços, sendo descrito pelo Open Group como "um estilo arquitetural que suporta orientação a serviços"(THE OPEN GROUP, 1995). Essa definição elenca características de serviços já associadas a microsserviços neste texto, como o caráter autocontido dos serviços, seu escopo delimitado e seu tratamento como caixa preta dentro das arquiteturas.

Essas semelhanças entre os conceitos de arquiteturas de microsserviços e SOA levaram a diversas comparações entre eles na literatura. Zimmermann (2017) realiza uma revisão dos trabalhos relacionados a ambos os conceitos e relaciona os princípios dos microsserviços às características de SOA. A partir da revisão, o estudo conclui que microsserviços seriam uma implementação de SOA em que geralmente são utilizadas tecnologias e práticas de engenharia de software modernas, como computação na nuvem, NoSQL, contêineres e DevOps.

Independentemente da posição adotada, se microsserviços representam um novo estilo arquitetural ou apenas uma abordagem para a implementação de SOA, pode-se concluir que a adoção de uma arquitetura de microsserviços pode apresentar benefícios em relação a uma

arquitetura monolítica, principalmente em relação à facilidade de manutenção e evolução do sistema.

Por essa razão, a proposta de uma arquitetura para gerenciamento de NFes baseada em microsserviços foi escolhida como linha base de comparação com a arquitetura baseada em SGBDR que é usada atualmente no Tribunal de Contas do Estado de Pernambuco. A realização dessa comparação constitui o foco dessa pesquisa.

2.2 API

Programas de aplicação necessitam de mecanismos para interagir e trocar mensagens com outros programas. Em arquiteturas modulares, como aquelas compostas por microsserviços, a necessidade de um mecanismo eficaz de comunicação é clara, uma vez que cada um dos módulos precisa realizar a troca de dados e acessar as funcionalidades dos demais módulos a ele conectados. Nesse sentido, Sommerville (2011) define Application Programming Interfaces (APIs) como interfaces, especificadas geralmente como conjuntos de operações, que permitem o acesso a uma funcionalidade da aplicação. Assim, essas interfaces possibilitam que serviços ou programas externos possam enviar requisições e receber respostas da aplicação, utilizando mensagens de formatos predefinidos pela API.

Uma das principais características das arquiteturas de microsserviços, como já citado, é a segregação das funções entre os microsserviços, que provêm conjuntos bem delimitados de funcionalidades. Nesse caso, a utilização de APIs pode ser a abordagem utilizada para permitir que os diferentes microsserviços interajam entre si, permitindo que um componente interaja com os demais, ou com aplicações externas, garantindo o acesso dessas aplicações às funcionalidades da arquitetura sem o conhecimento de sua estrutura interna.

A utilização de APIs contribui com a separação entre as partes pública e privada de cada componente, permitindo a independência entre a API e a implementação de seus métodos e funções (De Souza et al., 2004). Isso favorece a evolução do sistema que utiliza essa abordagem, já que os mecanismos internos de cada componente podem ser alterados sem afetar a comunicação do sistema com os programas externos ou mesmo a comunicação entre os componentes internos do sistema. APIs são amplamente utilizadas no desenvolvimento de aplicações, sendo a qualidade destas interfaces apontada como fator fundamental para o sucesso dos programas que as utilizam (LINARES-VÁSQUEZ et al., 2013).

2.2.1 REST API

Representational State Transfer (REST) é um estilo de arquitetura de software proposto por Fielding (2000) que combina elementos de arquiteturas em rede com restrições adicionais que, segundo o autor, definem uma interface uniforme de conectores. Fielding afirma que toda a comunicação no REST deve ser sem estado (*stateless*), de forma que todas as requisições enviadas de cliente para servidor devem ser autocontidas, possuindo todas as informações necessárias para a compreensão da requisição, sem que seja necessária a utilização de contexto armazenado no servidor.

Para compreender o estilo REST, é importante definir dois conceitos: recursos e representações. Recursos podem se referir a objetos físicos ou conceitos abstratos, sendo um conceito das arquiteturas REST utilizado para representar basicamente qualquer coisa que tenha importância suficiente para ser referenciada na arquitetura. Representações, por sua vez, são todas aquelas informações úteis a respeito do estado de um recurso (XINYANG; JIANJING; YING, 2009).

O diferenciador central do estilo REST é o foco em uma interface uniforme entre componentes. As interfaces entre os componentes da arquitetura são genéricas, o que simplifica a arquitetura e melhora a capacidade de evolução e a escalabilidade, ao desacoplar a implementação do serviço provido do formato de sua requisição. Para definir essa interface uniforme, Fielding determina que o estilo REST seja definido por quatro restrições de interface: identificação de recursos, manipulação de recursos através de representações, mensagens autodescritivas e hipermídia (*hypermedia*) como o motor do estado da aplicação. Hipermídia é um termo definido pela representação de informações de controle da aplicação juntamente com a própria informação que está sendo comunicada.

A mais conhecida implementação de REST é o protocolo HTTP(XINYANG; JIANJING; YING, 2009). As requisições no HTTP podem seguir quatro formatos: GET, POST, PUT e DELETE. GET é um comando que retorna dados a respeito de um recurso existente, PUT e POST criam um recurso ou atualizam um recurso existente e DELETE remove um recurso. Uma API REST (ou *RESTFul*) é uma API que segue os padrões REST. Em uma API deste tipo, a troca de mensagens entre os componentes do sistema frequentemente se dá através de HTTP.

2.3 RPCS E *PROTOCOL BUFFERS*

Em uma arquitetura de microsserviços, a comunicação síncrona entre serviços se dá principalmente através de REST, conforme descrito na na Seção 2.2, ou de Remote Procedure

Calls (RPCs) (KLEPPMANN, 2017). RPC consiste em um serviço (cliente) realizar a chamada de algum método de um outro serviço (servidor) através de um canal de comunicação. Há inúmeras abordagens possíveis para a implementação de RPCs, que consistem em diferentes maneiras de codificar as chamadas para envio pela rede, em um processo conhecido como serialização. Algumas dessas abordagens se baseiam na utilização de linguagem de descrição de interface (IDL), que é um tipo de linguagem em que são definidas as estruturas das mensagens e chamadas que podem ser realizadas através do protocolo RPC adotado. Muitos dos mecanismos de RPC contam inclusive com ferramentas de geração automática de código, o que permite que o usuário construa aplicações de forma veloz, já que uma vez definidas quais mensagens serão trocadas através de RPC, a ferramenta se encarregará de construir o código para cliente e servidor na linguagem de programação desejada, com base nas informações contidas na definição escrita em IDL.

Uma das bibliotecas disponíveis para realizar a serialização dessas chamadas remotas é *Protocol Buffers* (*protobuf*)[1], que é mantida pelo Google e possui uma ferramenta de geração de código em diversas linguagens, incluindo Python, C# e Java. *Protobuf* é utilizado no *gRPC*[2], que é um framework de código aberto criado pelo Google para a implementação de RPC em uma grande variedade de ambientes. O *gRPC* utiliza HTTP/2 para transmitir as mensagens serializadas com o *protobuf*. O *protobuf* permite serializar mensagens com alta eficiência, o que reduz a carga geral da arquitetura na rede se comparada à utilização de JSON ou XML para o formato das mensagens. Nesse sentido, Kleppmann (2017) demonstra que uma mesma mensagem no formato JSON de 81 bytes teria somente 32 bytes se serializada com *protobuf*.

Há uma série de críticas à utilização de RPC para a comunicação entre serviços na literatura, como mencionam Newman (2015) e Kleppmann (2017). Esses problemas geralmente estão relacionados à falta de garantias decorrentes da comunicação na rede, que tornam as chamadas remotas menos confiáveis do que chamadas de métodos locais. Os problemas relativos a comunicação em rede tornam a utilização de RPCs mais adequada para a comunicação entre processos que funcionam em um mesmo ambiente, como ocorre em um centro de dados de uma organização.

Newman e Kleppmann também mencionam problemas de interoperabilidade nas chamadas remotas, como a necessidade de traduzir tipos de dados na comunicação entre serviços de

[1] <https://developers.google.com/protocol-buffers>
[2] <https://grpc.io>

linguagens diferentes ou atrelar alguns mecanismos RPCs a plataformas específicas. Esses problemas são endereçados em alguns mecanismos de RPC modernos, como é o caso do *gRPC*, que é capaz de gerar código automaticamente em diferentes linguagens de programação e realizar comunicação em arquiteturas poliglotas (que utilizam serviços escritos em múltiplas linguagens).

2.4 NOSQL

O modelo relacional foi proposto há mais de 40 anos por Codd (1970). Desde então, dezenas de Sistemas de Gerenciamento de Banco de Dados (SGBDs) foram desenvolvidos com base nesse modelo, que obteve popularidade quase imediata devido à sua simplicidade e fundação matemática sólida (ELMASRI; NAVATHE, 2010).

Embora tenham enorme relevância para o armazenamento de dados transacionais até os dias de hoje, esses SGBDs relacionais apresentam limitações para determinados ramos de aplicação, principalmente quando surge a necessidade de se armazenar e processar grandes volumes de dados que não são necessariamente estruturados. Elmasri e Navathe afirmam que o excesso de funcionalidades dos sistemas SQL (relacionais) e a rigidez de esquema do modelo relacional podem trazer empecilhos ao bom funcionamento em determinados segmentos de aplicação que tratam de grandes conjuntos de dados.

Leavitt (2010) também expõe essas limitações e as sintetiza em quatro pontos principais:

- **Escalabilidade:** Segundo Leavitt, SGBDs relacionais possuem certas limitações para atingir a escalabilidade horizontal. Para escalar uma aplicação de SGBDR, ocorre a implementação de diversas instâncias do SGBD de forma distribuída em servidores distintos, que não compartilham recursos como CPU e memória. A dificuldade maior em se obter esse tipo de escalabilidade reside na complexidade de se realizar junções de dados presentes em tabelas armazenadas em nós distintos.

- **Complexidade:** Em SGBDs relacionais, os dados são armazenados em tabelas, formadas por colunas (atributos) de tipos fixos e predeterminados. Isso pode representar problemas para armazenar dados não estruturados, como dados multimídia (imagem, áudio, vídeo), por exemplo. Leavitt cita que, quando dados não se encaixam de forma trivial em uma estrutura tabular, a estrutura do banco de dados pode se tornar complexa e difícil de se trabalhar.

- **SQL:** A linguagem SQL oferece uma gama de comandos voltados para consulta em dados estruturados em tabelas. Quando se trata de outros tipos de informação, ela pode não ser o instrumento mais adequado para a extração de dados.
- **Grande conjunto de funcionalidades:** Sistemas de BD relacionais possuem muitas funcionalidades e garantias em relação à integridade dos dados. Esses recursos, que representam complexidade adicional e têm um custo computacional, podem não ser necessários em muitas aplicações onde há a demanda de ser mais flexível com relação a essas restrições e garantias de integridade.

NoSQL é uma categoria ampla que inclui uma série de SGBDs distintos. O termo NoSQL denota "não relacionais"ou "não apenas SQL", na sigla em Inglês, e claramente se refere a uma classe de SGBDs modernos e mais flexíveis, que não são relacionais, mas não existe consenso quanto a uma definição precisa do termo (CATTELL, 2010). Nesse caso, as definições de NoSQL costumam ser realizadas em função das características geralmente presentes nesse tipo de sistema. Em seu levantamento, Cattell define NoSQL com base em seis dessas características:

- A habilidade de escalar a taxa de transferência (*throughput*) de forma horizontal entre vários servidores.
- A habilidade de particionar dados (replicá-los e distribuí-los) ao longo de vários servidores.
- Uma interface ou protocolo simples em nível de chamada, em contraste a uma vinculação à linguagem SQL.
- Um modelo de concorrência mais simples do que as propriedades ACID das transações dos SGBDs relacionais.
- O uso eficiente de índices distribuídos e de memória RAM para armazenamento de dados.
- A habilidade de adicionar novos atributos de forma dinâmica aos registros de dados.

Elmasri e Navathe (2010) fazem referência também ao modelo de escalabilidade horizontal, à utilização de fragmentação, ou particionamento horizontal de dados ao longo de múltiplas coleções, a modelos de replicação de dados e à utilização de linguagens de programação de alto nível que implementam um conjunto menor de funcionalidades que a linguagem SQL, utilizada nos SGBDs relacionais. Quanto a este último aspecto, é comum nos sistemas NoSQL

que não sejam implementadas operações de junção (*joins*) entre coleções distintas de dados, como ocorre nos SGBDR. O acesso a dados em NoSQL é frequentemente realizado através de APIs que possuem conjuntos de comandos predefinidos, e não de consultas construídas pelo usuário.

Sistemas NoSQL também têm sido avaliados com base no teorema CAP, que determina que um SGBD distribuído não pode prover, ao mesmo tempo, as características de consistência, disponibilidade e partição tolerante a falhas. Elmasri e Navathe (2010) mencionam que muitos sistemas NoSQL têm como objetivo prover disponibilidade contínua, e, dados os requisitos impostos pelo processamento de grandes volumes de dados, acabam adotando modelos de consistência eventual, em que as atualizações de registros replicados não necessariamente irão ocorrer de imediato em todas as cópias, reduzindo o tempo de processamento das transações.

NoSQL, no entanto, são bastante heterogêneos e podem atender ou não a essa classificação. O MongoDB, por exemplo, possui configurações de consistência chamadas *read concern* e *write concern*[3] que permitem ajustar os níveis de consistência e disponibilidade de acordo com as características requeridas pela aplicação. Assim, percebe-se que até mesmo instâncias diferentes de um mesmo SGBD podem priorizar a disponibilidade ou consistência em diferentes níveis, de acordo com os parâmetros de configuração adotados.

Há dezenas de SGBDs NoSQL heterogêneos que seguem diferentes modelos de armazenamento de dados. Dentre eles, uma característica predominante é a ausência de necessidade de esquema, o que possibilita o armazenamento de registros de dados de diferentes formatos em uma mesma coleção. Esses SGBDs também costumam facilitar a escalabilidade horizontal e possuem modelos de consistência mais flexíveis. Esse conjunto de características torna essa classe de SGBDs adequada para o armazenamento de grandes volumes de dados que se apresentam em formatos diversos.

2.4.1 SGBDs Orientados a Documentos

Dentre os vários tipos de SGBDs NoSQL existentes, cabe descrever de forma breve os SGBDs que são considerados orientados a documentos ou baseados em documentos. Esses SGDBs geralmente armazenam dados no formato de coleções de documentos. Esses documentos costumam ser armazenados em formatos semiestruturados como XML ou JSON, permitindo a representação de dados em formatos autodescritivos, em que os dados são re-

[3] https://docs.mongodb.com/manual/reference/read-concern/#read-concern-levels

presentados junto a metadados que descrevem suas estruturas. Esse caráter autodescritivo permite a representação de dados que seguem estruturas irregulares, ou seja, aqueles em que itens de dados similares pertencentes à mesma coleção podem ser representados por conjuntos de atributos diferentes uns dos outros.

Um exemplo de SGBD orientado a documentos é o MongoDB, cujo modelo de dados consiste em armazenar documentos no formato JSON. Esse formato permite o armazenamento de dados de vários tipos, incluindo tipos complexos como *arrays* e objetos aninhados (MongoDB Inc., 2021). Como já expresso anteriormente, este tipo de formato semiestruturado permite também o armazenamento de dados com flexibilidade de esquema, sem que haja um modelo fixo e predefinido para os registros de dados.

2.5 MODELO C4

Modelos de arquitetura de software se destinam a representar, de forma padronizada, como os sistemas de software estão organizados. Um estilo de arquitetura, segundo Garlan e Shaw (1993), pode ser entendido como o vocabulário de componentes e conectores que, juntamente com um conjunto de restrições, podem representar os sistemas de computador.

Nesse sentido, o modelo C4 é uma notação para representação de arquitetura de software que se baseia na criação de diagramas em quatro níveis diferentes de abstração: os diagramas de contexto, contêineres, componentes e código. O modelo oferece essas quatro visões hierarquizadas da arquitetura de software representada, permitindo a disponibilização de diagramas adequados a públicos diferentes que tenham requisitos distintos de visualização.

Para construir esses diagramas, o C4 utiliza um conjunto de abstrações para descrever os diversos aspectos da arquitetura de um sistema (BROWN, 2018):

- **Pessoas** são os seres humanos que irão utilizar e interagir com o sistema de software.
- **Sistemas de software** correspondem ao maior nível de abstração do modelo. Essa abstração é utilizada tanto para representar o sistema cuja arquitetura está sendo especificada, quanto os outros sistemas que possuem relação de dependência ou interação com o sistema que está sendo modelado.
- **Contêineres** servem para representar aplicações ou estruturas de armazenamento de dados. Existem vários exemplos de conceitos e aplicações que podem ser representadas

por contêineres, como aplicações web, aplicações móveis, aplicações desktop, bancos de dados, sistemas de arquivos, scripts *shell*, dentre outros.

Em suma, um contêiner deve ser entendido como um contexto ou limite em que algum código é executado ou dados são armazenados - cada contêiner deve ser capaz de ser executado ou implementado separadamente, geralmente rodando como um processo individualizado.

- **Componentes** são as partes que compõem um contêiner e são definidos como agrupamentos de funções relacionadas e encapsuladas por meio de uma interface bem definida.
- **Código** diz respeito à visão de como cada componente é implementado na linguagem de programação adotada. Essa abstração inclui elementos como classes, interfaces, objetos e funções.

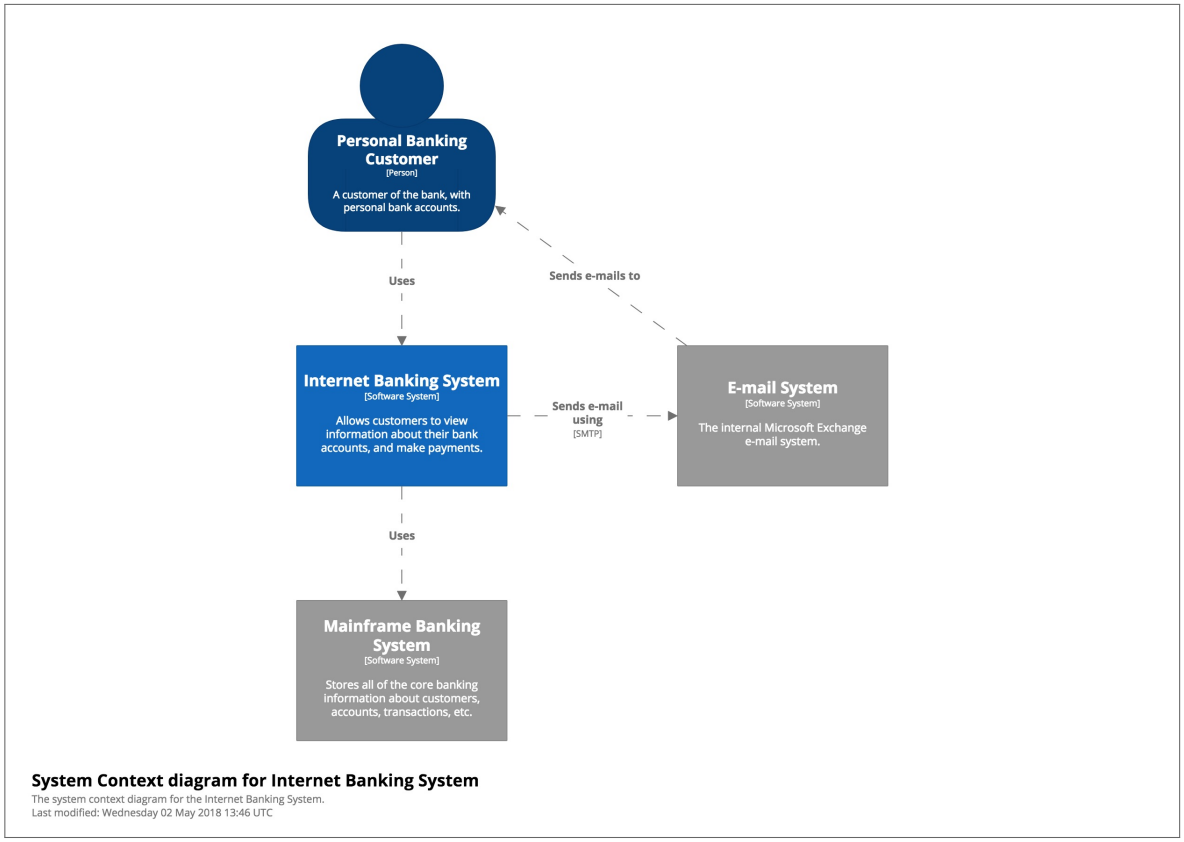
Aqui cabe destacar que a definição de contêiner no modelo C4 se mostra alinhada com as definições de microsserviços apresentadas anteriormente. Um microsserviço pode ser visto como um processo independente e coeso, com um conjunto de funcionalidades bem definido e limitado. Como microsserviços são serviços coesos e independentes, cada unidade considerada como um microsserviço distinto deverá ser representada em um contêiner de forma individual.

Como já citado, o modelo C4 organiza os conceitos e estruturas apresentadas até aqui em diagramas de diferentes níveis de abstração. Diferentes diagramas podem servir a diferentes propósitos e usuários, em diferentes etapas e tarefas do desenvolvimento e manutenção do software.

Os tipos de diagramas no modelo C4 são os seguintes:

- **Nível 1 - Diagrama de contexto do sistema:** Neste nível há uma representação de alto nível do sistema a ser construído, na forma de uma caixa preta, ou seja, sem detalhes a respeito dos componentes e fluxos internos de informação no sistema. O sistema é representado no contexto em que está inserido, acompanhado de representações das pessoas que o utilizam e os outros sistemas com os quais ele interage.

Figura 1 – Exemplo de diagrama de contexto no modelo C4



Fonte: (BROWN, 2018)

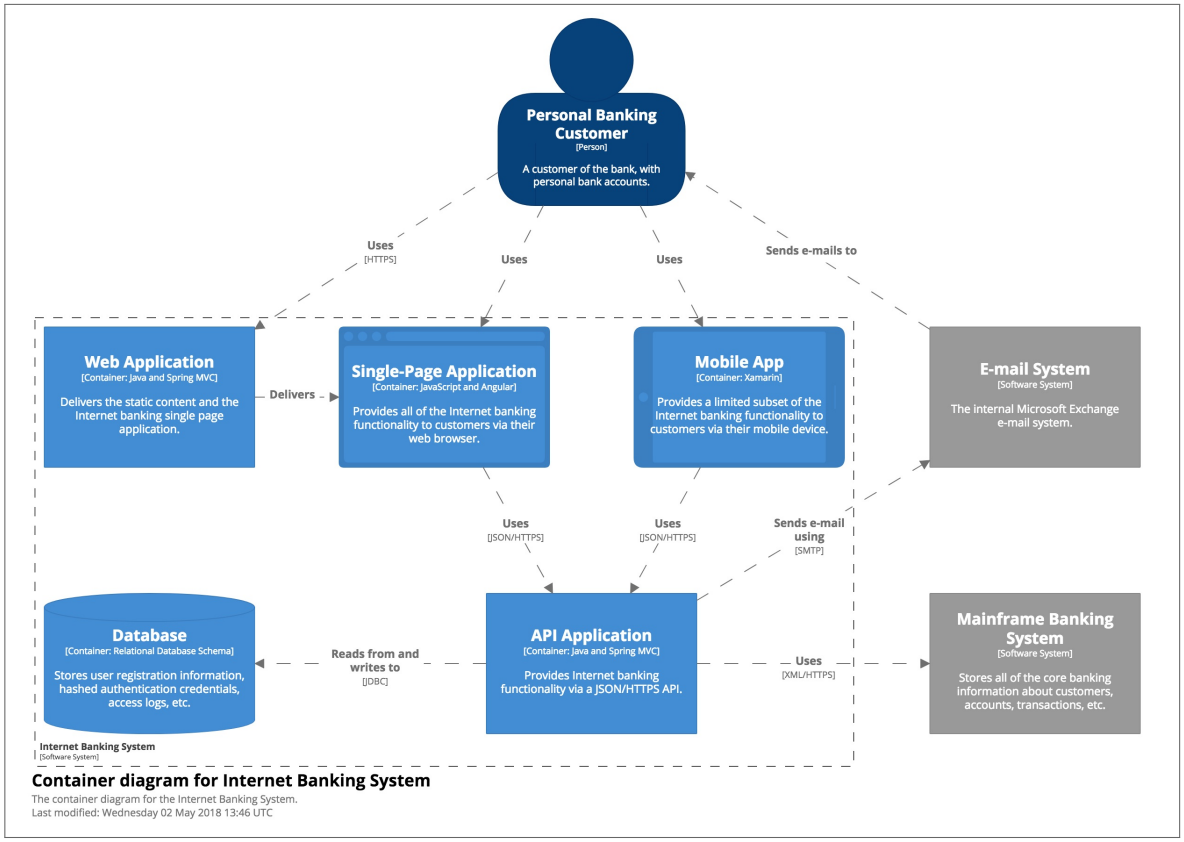
- **Nível 2 - Diagrama de contêineres:** Um diagrama de contêineres irá exibir os contêineres que compõem o sistema de software da camada anterior. Como já definido anteriormente, os contêineres são aquelas unidades independentes que executam um conjunto bem definido de funcionalidades ou armazenam dados. Em uma arquitetura de microsserviços, cada microsserviço estará presente neste diagrama como um contêiner independente.

- **Nível 3 - Diagrama de componentes:** Diagramas de componentes detalham os componentes que fazem parte de um determinado contêiner, descrevendo-os e exibindo as interações entre eles e os contêineres externos. Os componentes, como já definimos, são agrupamentos de código com funções relacionadas e mantidas dentro de um contêiner.

- **Nível 4 - Diagrama de código:** Por fim, o modelo C4 prevê a existência de diagramas de código, que permitem detalhar como determinado componente é implementado em linguagem de programação. A criação de modelos neste nível de detalhe é desaconselhada, a menos que seja necessária, e somente para a definição de componentes mais

complexos ou importantes.

Figura 2 – Exemplo de diagrama de contêineres no modelo C4



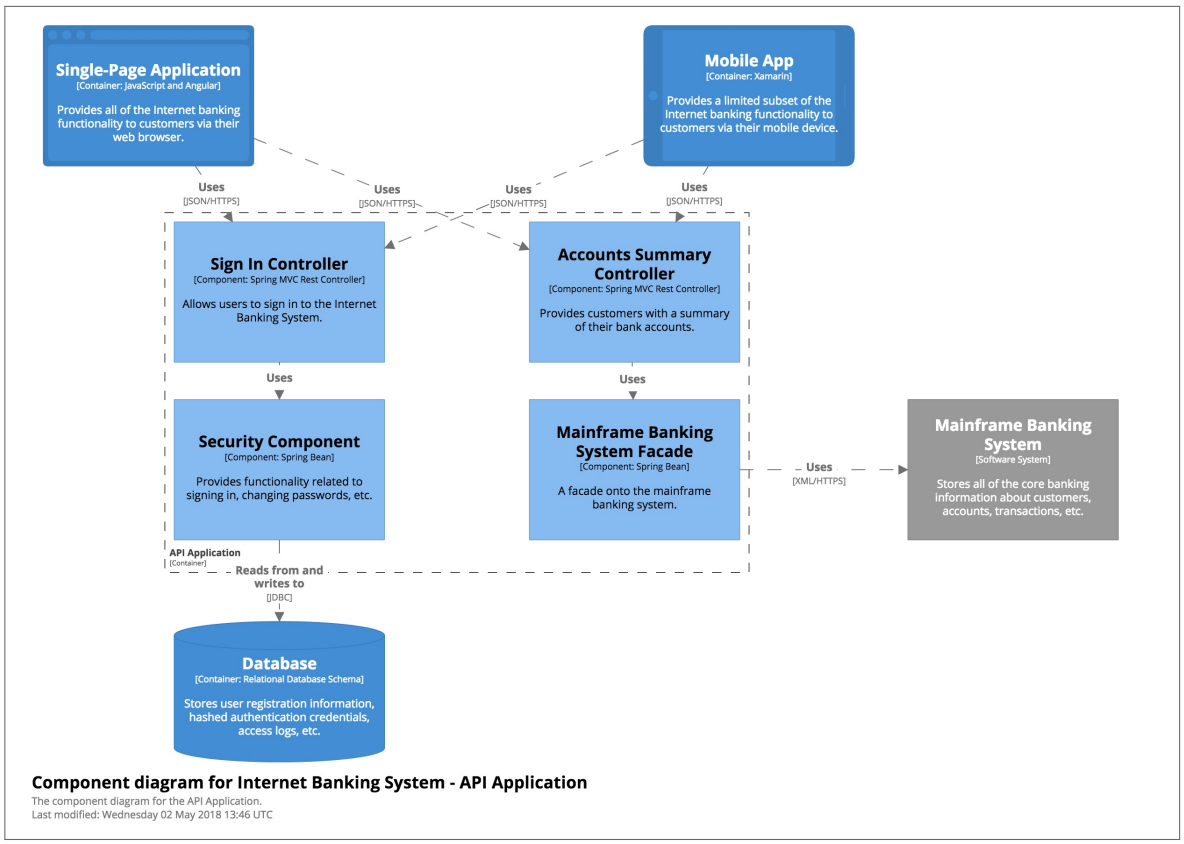
Fonte: (BROWN, 2018)

Os diagramas do modelo C4 seguem uma organização hierárquica, permitindo a evolução do modelo de forma relativamente independente. Por exemplo, uma alteração na disposição dos componentes internos de um contêiner não necessariamente irá afetar o diagrama de contêineres pertinente.

Cabe ressaltar que o modelo C4 não definem qualquer tipo de representação gráfica específica. Os diagramas podem apenas implementar os conceitos definidos e expostos neste capítulo, como contêineres e componentes, sem que seja necessário adotar um conjunto de símbolos predefinidos para representá-los.

C4 representa um mecanismo flexível de se representar arquiteturas de software. Os diferentes tipos de diagramas previstos podem atender às necessidades de grupos de usuários distintos, para que toda a comunidade de usuários seja capaz de compreender os aspectos da arquitetura de software pertinentes.

Figura 3 – Exemplo de diagrama de componentes no modelo C4



Fonte: (BROWN, 2018)

2.6 ISO/IEC 25010

O ISO/IEC 25010 é um modelo de qualidade de software que visa determinar quais características devem ser levadas em consideração na análise das propriedades de um sistema de software (ISO/IEC, 2011). O padrão inclui oito características de qualidade, e divide cada uma em duas ou mais subcaracterísticas. As características são (em tradução livre): aptidão funcional, eficiência de desempenho, compatibilidade, usabilidade, confiabilidade, segurança, manutenibilidade e portabilidade.

Para esta pesquisa, foram considerados os aspectos de manutenibilidade e portabilidade, que são definidos como requisitos para o desenvolvimento de um artefato no processo do *framework Design Science Research*, conforme descrito na Seção 4.1.1. Essas características são detalhadas a seguir.

Em primeiro lugar, a manutenibilidade é definida como a característica que representa o grau de efetividade e eficiência com que um produto ou sistema pode ser modificado para ser melhorado, corrigido ou adaptado para mudanças no ambiente ou nos requisitos. A manute-

nibilidade possui as seguintes subcaracterísticas:

- **Modularidade:** Grau com que um sistema ou programa de computador é composto por componentes discretos, de forma que uma alteração em um componente tem impacto mínimo em outros componentes.
- **Reusabilidade:** Grau com que um ativo pode ser usado em mais de um sistema, ou na construção de outros sistemas.
- **Analisabilidade:** Grau de eficácia e eficiência com que é possível verificar o impacto em um produto ou sistema da alteração em uma ou mais de suas partes, ou com que é possível diagnosticar um produto com deficiências ou causas de erros, ou identificar partes a serem modificadas.
- **Modificabilidade:** Grau com que um produto pode ser modificado, de forma eficaz e eficiente, sem introduzir defeitos ou degradar a qualidade do produto.
- **Testabilidade:** Grau de eficácia e eficiência com que critérios de teste podem ser estabelecidos para um sistema, produto ou componente, e como esses testes podem ser desempenhados para determinar se estes critérios foram atingidos.

Já a portabilidade é definida no ISO/IEC 25010 como o grau de efetividade e eficiência com que um produto, sistema ou componente pode ser transferido para um outro hardware, software ou outro ambiente de utilização. Ela é composta pelas seguintes subcaracterísticas:

- **Adaptabilidade:** Grau de efetividade e eficiência com que um produto ou sistema pode ser adaptado para hardware, software ou outros ambientes operacionais diferentes.
- **Instalabilidade:** Grau de efetividade e eficiência com que um produto ou sistema pode ser instalado ou desinstalado com sucesso em um ambiente específico.
- **Substituição:** Grau com que o produto pode substituir outro produto para o mesmo propósito em um mesmo ambiente.

A avaliação dessas características do ISO/IEC 25010, realizada na Seção 5.3, é utilizada para determinar em que grau a arquitetura descrita nesta dissertação atende aos requisitos de manutenibilidade e portabilidade definidos no processo de *design* da arquitetura.

2.7 NOTA FISCAL ELETRÔNICA

A NF-e foi instituída pelo Ajuste SINIEF 07/05, instrumento celebrado pela União, os Estados e o Distrito Federal e publicado no Diário Oficial da União em setembro de 2005. Uma NF-e pode ser definida como um documento eletrônico que se destina a registrar operações relativas à circulação de mercadorias ou à prestação de serviços, possibilitando o recolhimento de tributos, como o Imposto sobre Circulação de Mercadorias e Prestação de Serviços (ICMS) e o Imposto sobre Produtos Industrializados (IPI). A NF-e foi desenvolvida de forma conjunta pela Receita Federal do Brasil (RFB), juntamente com as Secretarias de Fazenda dos Estados (RECEITA FEDERAL DO BRASIL, 2022).

O Ajuste SINIEF 07/05 estabelece o padrão XML como o formato em que deverão ser gerados os documentos digitais das NF-e. Esses documentos devem seguir um modelo pre-estabelecido, com seus campos definidos no Manual de Orientação ao Contribuinte (MOC), documento oficial produzido no âmbito do ENCAT, evento quadrimestral que reúne representantes dos órgãos fazendários dos Estados (ENCAT, 2020).

O MOC, cuja versão 7.00 foi publicada em novembro de 2020, compreende as alterações realizadas no formato da NF-e realizadas pela Receita Federal por meio de suas Notas Técnicas. Essas Notas Técnicas são normativos publicados periodicamente pela RFB e compreendem ajustes e modificações em aspectos como leiaute e regras de validação dos campos da NF-e. Tais alterações ocorrem com certa frequência: em 2020, foram publicadas sete notas técnicas (de 2020.001 a 2020.007), tendo algumas delas sido publicadas em múltiplas versões, com alterações nas regras de validação ou no leiaute dos documentos em cada uma das versões. A Nota Técnica nº 2020.005, por exemplo, já teve publicadas as versões 1.00, 1.10, 1.20 e 1.21 até o final de 2021.

O leiaute da NFe especificado no Anexo I do MOC 7.00 compreende mais de 400 campos distintos. Cada campo tem regras de validação diferentes, incluindo regras a respeito da obrigatoriedade do seu preenchimento. Um mesmo campo poderá ser obrigatório ou opcional em notas distintas, a depender do tipo de NF-e e da operação que está sendo registrada. Como as NF-e são geradas no padrão XML, formato semiestruturado, eventuais campos que não sejam obrigatórios para o tipo de operação registrado podem estar ausentes do documento.

A NFe funciona numa estrutura hierárquica, em que elementos que dizem respeito a uma determinada informação são agrupados e dispostos como subelementos de um elemento pai. Os campos de uma NFe também podem ser multivalorados. Na versão 7.00 do MOC, por

exemplo, é especificado que uma mesma NFe pode conter de 1 até 990 itens de produtos ou serviços, listados sob o campo *prod* no documento.

A NF-e representou um avanço na desburocratização do Estado ao substituir a emissão do documento fiscal em papel pelo formato eletrônico, permitindo o registro e o acompanhamento em tempo real através do Portal da Nota Fiscal Eletrônica[4]. Até fevereiro de 2022, mais de 32 bilhões de notas fiscais emitidas por cerca de 2 milhões de emissores distintos já haviam sido autorizadas pelos órgãos fazendários no Brasil.

Esse volume de dados e as constantes alterações no modelo do documento representam desafios para as arquiteturas e ferramentas que se destinem a gerenciar dados das NF-e. Como exposto na Seção 2.4 deste capítulo, SGBDs relacionais apresentam escalabilidade horizontal mais limitada, o que pode levar a dificuldades no processamento de uma grande quantidade de NF-e. Além disso, os SGBDs relacionais armazenam dados no formato de tabela, cuja estrutura é fixa e predefinida, tornando alterações de esquema necessárias a cada mudança de formato das NF-e.

Por outro lado, sistemas de BD orientados a documentos facilitam alterações de esquemas, facilitando o gerenciamento de mudanças no formato das NF-es, mas podem ter um impacto negativo no tempo processamento de documentos, especialmente se a quantidade elevada de navegações em documentos não for gerenciada. Por esse motivo, esta pesquisa investiga o desempenho de diferentes tipos de arquiteturas de software para gerenciamento de NF-es.

2.8 CONCLUSÕES

Foram apresentados até aqui conceitos importantes para a compreensão da arquitetura descrita neste trabalho. A arquitetura é composta por microsserviços, que foram definidos aqui como pequenos serviços coesos que se comunicam entre si, permitindo o desenvolvimento de sistemas com pouco acoplamento e boa capacidade de evolução.

Já APIs foram definidas neste capítulo como interfaces que permitem que aplicações ou componentes externos acessem as funcionalidades de determinado componente de um sistema de software, como um microsserviço. Elas podem ser do tipo REST API, caso em que seguem restrições e características particulares do estilo REST, o que inclui o envio de mensagens sem estado e a representação de objetos de interesse através dos chamados recursos. REST permite arquiteturas escaláveis e com baixo grau de acoplamento, dada a uniformidade das in-

[4] Acessível em <https://www.nfe.fazenda.gov.br>

terfaces utilizadas. A arquitetura investigada nesta dissertação utiliza o protocolo HTTP, uma implementação de REST, para a comunicação entre os microsserviços. Também foram descritos aspectos de RPCs, chamadas de procedimentos remotos, que são o meio de comunicação adotado para a troca de mensagens entre os microsserviços da arquitetura.

NoSQL representa uma categoria de sistemas de bancos de dados que não seguem o modelo relacional, abarcando diversos modelos de dados distintos. Esses sistemas, em geral, se apresentam como mais flexíveis do que aqueles relacionais, podendo armazenar dados sem esquema fixo, e podem apresentar um bom desempenho para processar grandes volumes de dados em formatos variados. Dentre esses modelos, foi descrito o modelo orientado a documentos, adotado pelo sistema MongoDB. SGBDs NoSQL se mostram adequados para o armazenamento e processamento de notas fiscais eletrônicas, que são geradas e transmitidas no formato semiestruturado XML e seguem um modelo de dados que sofre frequentes alterações. No entanto, é necessário avaliar se o desempenho é mantido constante ou mesmo melhorado em relação a SGBDs relacionais, o que é objeto da avaliação experimental realizada nesta dissertação.

Ainda, foram expostos os conceitos associados ao modelo C4, modelo de arquitetura de software que divide os sistemas em quatro níveis: contexto, contêineres, componentes e código. Esse modelo, livre de uma notação específica, preza pela simplicidade e facilidade de comunicação entre diversos públicos para representar as arquiteturas de software. Ele é utilizado para descrever a arquitetura investigada e discutida neste trabalho.

Em seguida, o modelo de qualidade de software ISO/IEC 25010 foi descrito e suas características listadas. Esse modelo visa determinar quais aspectos devem ser levados em consideração na avaliação da qualidade de sistemas de software. Dentre todas as características do modelo, foram destacadas as definições de manutenibilidade e portabilidade, que são utilizadas para avaliação da arquitetura de software investigada nesta pesquisa.

Por fim, discutiu-se a respeito do projeto da Nota Fiscal Eletrônica, iniciativa de modernização do serviço público que permitiu a digitalização das notas fiscais, agora emitidas em documentos no formato XML. Foram expostas as características dos documentos XML das NF-e, sua estrutura e os desafios de se trabalhar com este tipo de dado nos SGBDs relacionais.

No Capítulo 3, serão descritos os trabalhos correlatos encontrados na literatura. Esses trabalhos descrevem arquiteturas de microsserviços que, dentre suas diversas funcionalidades, realizam o gerenciamento de dados em bancos de dados NoSQL e utilizam REST APIs para a troca de mensagens. Também serão descritos e avaliados trabalhos que tratem de quaisquer

arquiteturas de software que se destinam ao gerenciamento de notas fiscais eletrônicas.

3 TRABALHOS CORRELATOS

A literatura traz diversos trabalhos que envolvem arquiteturas de software que realizam o gerenciamento de dados em SGBDs NoSQL e que utilizam microsserviços, além de trabalhos que tratam dos aspectos relativos ao gerenciamento de notas fiscais eletrônicas. Neste capítulo, são descritos os trabalhos que apresentem correlação com a pesquisa descrita no presente documento.

Na Seção 3.1, primeiramente são listados os trabalhos que descrevem arquiteturas e microsserviços que realizem o gerenciamento de dados em SGBDs NoSQL. Já na Seção 3.2, são descritos e avaliados trabalhos que tratam do gerenciamento de notas fiscais eletrônicas. A Seção 3.3 apresenta as considerações finais sobre os trabalhos correlatos descritos e conclui o capítulo.

3.1 ARQUITETURAS DE GERENCIAMENTOS DE DADOS

Nesta seção, são detalhados trabalhos que descrevem arquiteturas de microsserviços que realizam o gerenciamento de dados em SGBDs NoSQL, com propósitos e ramos de aplicação diversos. Alguns dos trabalhos utilizam REST APIs para a comunicação com clientes ou entre o serviços. Ao final da seção, uma avaliação comparativa dos trabalhos em relação à pesquisa descrita nesta dissertação é realizada.

3.1.1 Arquiteturas de Microsserviços e NoSQL com Avaliação Experimental de Desempenho

3.1.1.1 O Trabalho de Mendes et al.

O trabalho de (MENDES et al., 2019) trata de uma arquitetura de gerenciamento de dados oriundos de dispositivos IoT (*Internet of Things*) com base em microsserviços. A arquitetura inclui a coleta de dados produzidos por vários sensores que compõem uma camada de IoT, o pré-processamento desses dados em dispositivos da borda da rede (edge), e o posterior envio para uma camada de nuvem, que é composta por vários microsserviços com funcionalidades distintas.

A aplicação em que a arquitetura é posta em prática, chamada VITASENIOR-MT, tem como objetivo realizar o monitoramento de dados biométricos e ambientais de sensores em

estabelecimentos de residência de longa permanência para idosos de forma eficiente. Os dispositivos da borda da rede, denominados VITABOX, são responsáveis pela validação e agregação dos dados dos sensores, e pela comunicação com a nuvem. A camada de nuvem, por sua vez, é onde são executadas as atividades relacionadas à manipulação e à disponibilização dos dados para os atores interessados, com destaque para os cuidadores dos idosos.

A camada de nuvem funciona sobre o serviço de PaaS (Platform as a Service) provido pela IBM Cloud. Nessa plataforma há vários nós de processamento, com cada tipo de nó executando um microsserviço de funcionalidade diferente.

No fluxo de dados da aplicação, os dados coletados pelos sensores IoT são convertidos pelo VITABOX para o formato JSON, para fornecer uma abstração para os diferentes formatos de dados produzidos pelos diferentes sensores. Dessa forma, os formatos diferentes são unificados em um só padrão, o que também possibilita a agregação desses registros antes do envio para a camada de nuvem quando necessário para aliviar o tráfego de rede.

Após a conversão e agregação, os dados processados pelo VITABOX podem ser transmitidos para os microsserviços da nuvem para que seja realizado o processamento relativo ao objetivo da aplicação. Para o armazenamento desses registros, é adotado um sistema de gerenciamento de bancos de dados orientado a documentos, uma vez que os dados estão no formato JSON. Na aplicação descrita no artigo, é utilizado o MongoDB. A arquitetura também prevê a utilização de um SGBD relacional para o armazenamento de dados sensíveis (como aqueles relativos à saúde de idosos) e contábeis.

Com o objetivo de prover interoperabilidade, a arquitetura possui um microsserviço destinado à manutenção de uma interface de programação de aplicação (API) para o acesso aos diferentes SGBDs integrantes da arquitetura. Este serviço gerencia uma API que provê comandos para as operações CRUD (criação, leitura, atualização e remoção) de registros de dados nos diversos modelos adotados. A API possibilita a tradução dos comandos específicos de cada modelo em comandos genéricos através da utilização de Representational State Transfer (REST).

O microsserviço responsável pela API também atua na comunicação com o SGBD, incluindo o processo de autenticação. Através de um sistema de fila de mensagens (na aplicação VISTASENIOR-MT é utilizado o RabbitMQ), os demais microsserviços enviam comandos HTTP ao nó que contém a API. Este, por sua vez, envia as requisições no formato adequado para o SGBD correspondente, efetivamente centralizando toda a comunicação do restante da arquitetura com os sistemas de gerenciamento de bancos de dados. Dessa maneira, os demais

microsserviços podem acessar os diferentes bancos de dados de forma transparente, sem que seja necessário definir, para cada microsserviço, os comandos da linguagem de acesso a dados ou credenciais de autenticação.

Para mensurar o desempenho da arquitetura, uma análise experimental é realizada para medir os tempos de resposta a requisições enviadas à API. O trabalho compara uma arquitetura monolítica, em que as funcionalidades de negócio da aplicação e o gateway da API funcionam em um só serviço, à arquitetura de microsserviços, em que as funcionalidades são separadas em serviços distintos. Nos resultados, o desempenho observado da arquitetura de microsserviços foi considerado superior.

3.1.1.2 O Trabalho de Lu, Xu e Lan

O trabalho em questão trata de uma plataforma baseada em arquitetura de microsserviços, denominada SOOCP. A plataforma visa permitir o gerenciamento e a análise de dados de características óticas de objetos espaciais (SOOC - *Space Object Optical Characteristics*). Esses dados podem ser de diversas naturezas e se apresentar em diversos formatos, o que leva à necessidade de ferramentas adequadas para se lidar com essa heterogeneidade. Os dados podem ser tanto estruturados quanto não estruturados, a depender da característica e da fonte de informações utilizada.

A arquitetura do SOOCP se destina a atender a uma série de requisitos. No nível de dados, é necessário permitir aos operações de entrada e extração de dados, a sua integração e visualização, permitindo a análise e a realização de experimentos. No nível de algoritmos, é necessário permitir a integração de algoritmos para a realização de experimentos, bem como o provimento de serviços online e a personalização do funcionamento da arquitetura. Por fim, no nível de desenvolvimento, é necessário permitir a integração da arquitetura a serviços preexistentes, fornecer uma API de acesso unificada e prover as capacidades de balanceamento de carga e descoberta de serviços.

A exemplo dos demais trabalhos descritos nesta seção, o SOOCP utiliza uma arquitetura de microsserviços para atingir as suas funcionalidades. Na arquitetura, a comunicação entre os serviços e entre os clientes e o SOOCP se dá através de REST API.

Nessa arquitetura, as requisições dos clientes são processadas pelo *gateway* da API, que provê uma interface unificada de acesso aos serviços da arquitetura. Dessa maneira, as funcionalidades de armazenamento, caching, recuperação e integração de dados podem ser aces-

sadas, sem que os clientes precisem conhecer as mensagens internas trocadas pelos serviços ou as consultas executadas nos bancos de dados SQL/NoSQL utilizados.

O serviço de gerenciamento de dados é um serviço híbrido, ou seja, um só serviço que é capaz de gerenciar os dados armazenados tanto em bancos de dados relacionais (SQL) como NoSQL. Há um modelo genérico de dados que permite representar os diferentes tipos de características óticas armazenadas e processadas pelo sistema, o que favorece a interoperabilidade entre os diferentes tipos de bases de dados utilizados.

A respeito dos bancos de dados, o trabalho classifica aqueles usados na arquitetura em três tipos. O primeiro deles é o banco de dados de *cache*. Na implementação do SOOCP, os bancos de dados de cache são do tipo NoSQL e servem para armazenar dados intermediários e dados que devem ser compartilhados com os demais bancos de dados. Por exemplo, ao se aplicar um algoritmo que realize algum pré-processamento dos dados espaciais para análise, os resultados intermediários da operação podem já ser armazenados em um banco de dados de cache, o que torna desnecessários novos acessos ao banco de dados que contém os dados originais. Isso permite uma execução mais célere das operações que utilizem esses dados e alivia a carga nos repositórios que contêm os dados originais.

O segundo tipo de banco de dados utilizado no SOOCP são os tradicionais bancos de dados relacionais (SQL). Esses bancos de dados são utilizados para armazenar dados brutos, os resultados dos processamentos e também metadados para os dados não estruturados, quando estes últimos estão armazenados na forma de arquivos em um sistema de arquivos.

O terceiro tipo compreende os bancos de dados não estruturados. Esse tipo de banco de dados se destina a prover um acesso de alta performance a dados não estruturados, como imagens e textos. Para prover suporte a esses formatos não estruturados, são utilizados bancos de dados NoSQL. No protótipo construído no trabalho, são utilizados Redis e MongoDB.

O SOOCP é avaliado em função de aspectos quantitativos e qualitativos em um estudo de caso. No aspecto qualitativo, são avaliadas as funcionalidades providas pela arquitetura em relação aos requisitos predefinidos. Quanto ao aspecto quantitativo, são comparados os tempos de inserção e extração de grandes volumes de dados em relação àqueles obtidos realizando as mesmas operações no MySQL. O esquema híbrido de acesso aos dados do SOOCP oferece uma performance superior ao MySQL nos experimentos realizados, além de atender melhor às necessidades dos usuários ao agregar os benefícios das tecnologias NoSQL à robustez dos bancos de dados relacionais.

3.1.2 Frameworks e Middlewares para Arquiteturas de Microsserviços com Avaliação Experimental de Desempenho

3.1.2.1 O trabalho de Viennot et al.

Esta arquitetura baseada em microsserviços tem como objetivo realizar a replicação de dados entre sistemas de bancos de dados heterogêneos, mantendo representações dos mesmos dados em diferentes formatos e modelos. Nesse sentido, a ideia é permitir que os diferentes serviços que compõem cada aplicação estejam associados a seus próprios SGBDs. Esses SGBDs, então, poderão armazenar o mesmo conjunto de dados, mas armazená-los de formas diversas, podendo diferir em termos de esquema, layout, engine ou mesmo na composição de seus índices.

No momento da publicação do trabalho, o Synapse (nome da arquitetura), oferecia suporte para a utilização de bancos de dados relacionais (PostgreSQL, MySQL, Oracle), de documentos (MongoDB,TokuMX, RethinkDB), colunares (Cassandra), de busca (Elasticsearch) e baseados em grafos (Neo4j).

Para gerar a camada de abstração que permite a interoperabilidade dos serviços em relação ao acesso a dados, o Synapse faz uso do paradigma Model-View-Controller (MVC). De modo geral, esse paradigma separa logicamente as representações de dados em estruturas chamadas modelos e a lógica de negócio da aplicação em controladores. Os modelos são mapeados a cada SGBD através de estruturas chamadas Object Relational Mappers (ORMs), que são processos responsáveis por permitir o acesso aos dados para realização das operações de inserção, leitura, modificação e remoção de dados através de APIs.

O Synapse está organizado no modelo publicador/assinante, com um gerenciador de fila de mensagens como Kafka[1], RabbitMQ[2] ou SQS[3] fazendo a ligação entre publicadores e assinantes. Publicadores são os nós que iniciam a transação e assinantes os demais nós que devem registrar as réplicas do banco de dados em seus formatos e modelos específicos. Para garantir a interoperabilidade, os valores dos atributos envolvidos em cada transação realizada por um publicador são transmitidos no formato JSON e os comandos convertidos para o formato adequado a cada SGBD associado a cada um dos assinantes através dos ORMs.

O trabalho é implementado em ambientes de produção e avaliado em termos do *overhead* introduzido pela arquitetura. Os testes mostram que a arquitetura apresentou *overhead* de

[1] <https://kafka.apache.org>
[2] <https://www.rabbitmq.com>
[3] <https://aws.amazon.com/pt/sqs/>

comunicação médio de 8% nos ambientes avaliados, o que foi considerado um bom resultado pelos autores. Também é realizado um teste de estresse na forma de uma simulação de tráfego de uma aplicação de rede social, em que ficou demonstrada uma boa escalabilidade da arquitetura para até 60.000 operações de atualização de dados por segundo.

3.1.2.2 O trabalho de Kathiravelu et al.

O trabalho em questão apresenta como principal contribuição um *middleware* cujo objetivo é expor dados armazenados em diversas fontes heterogêneas através de uma REST API a aplicações, sem introduzir um gargalo de performance considerável. A arquitetura proposta é de microsserviços, cuja modularidade permite a evolução das aplicações para adaptação a novos tipos de fontes de dados e funcionalidades.

O *middleware* Bindaas fornece ferramentas para que os usuários definam APIs para acesso a diferentes fontes de dados, o que torna a arquitetura extensível e configurável para cada cenário de utilização. AS APIs criadas são organizadas em estruturas chamadas projetos, que podem compreender um ou mais provedores de dados, estes últimos sendo as estruturas responsáveis pelo acesso ao mecanismo de armazenamento de dados, geralmente um SGBD. Os provedores de dados do Bindaas abarcam as ações padrão HTTP POST, GET, PUT e DELETE para realizar as operações conhecidas como CRUD: criação, leitura, atualização e remoção de registros de dados.

O nome Bindaas está relacionado às *bind variables* (variáveis de ligação) que são suportadas pela arquitetura para a definição de consultas a dados nas APIs criadas pelos usuários. Assim, as consultas que façam parte da API podem conter variáveis dentro do código para permitir a passagem de parâmetros na chamada de API correspondente. Aqueles usuários que estiverem interagindo com a API poderão, então, passar parâmetros para consulta sem precisarem ser expostos ao código específico que cada provedor de dados irá utilizar para consultar o banco de dados correspondente.

O Bindaas também permite a extensão das APIs com o auxílio de módulos que se dividem em plug-ins e modificadores. Os plug-ins podem adicionar funcionalidades como limitação de banda e balanceamento de carga nos resultados recebidos dos provedores de dados. Já os modificadores permitem que os usuários definam comportamentos que serão aplicados às mensagens de consultas e de resultados de consultas trocadas entre os provedores de dados e os bancos de dados, além de poderem atuar para modificar o próprio conteúdo de uma

requisição POST (a chamada *payload*, ou carga útil).

Os modificadores de consultas atuam nas consultas enviadas pela REST API ao banco de dados, podendo inspecionar ou alterar seu código antes que este seja enviado ao banco de dados. Os autores exemplificam uma potencial aplicação deste mecanismo ao sugerir um modificador de consulta que inspecione um código SQL à procura de heurísticas de SQL Injection. Já os modificadores de resultados de consulta podem realizar algum processamento no resultado das consultas executadas ao banco de dados, como salvar imagens retornadas em objetos binários (BLOBs) diretamente em um outro formato desejado. Um modificador de *payload* de submissão, por fim, é o mecanismo que permite alterar o corpo de uma requisição POST. Os autores exemplificam uma aplicação deste último tipo de modificador àquelas aplicações que lidam com dados sensíveis, que precisam passar por um processo de anonimização e limpeza de dados antes de serem transmitidos.

Para avaliar a performance do middleware, os autores realizaram uma simulação de várias requisições ao banco de dados MongoDB, comparando a utilização do Bindaas com a utilização de um cliente JDBC. O trabalho não identificou um aumento significativo de latência com a utilização do Bindaas em relação ao cliente JDBC convencional.

3.1.3 Arquiteturas de Microsserviços e NoSQL sem Avaliação Experimental de Desempenho

3.1.3.1 O Trabalho de Braun et al.

O trabalho intitulado *A Generic Microservice Architecture for Environmental Data Management* (BRAUN et al., 2017) visa propor uma arquitetura genérica para o gerenciamento de dados, especialmente voltada para o processamento de grandes volumes de dados originados de sensores, em ramos de aplicação diversos. A arquitetura utiliza microsserviços com o objetivo de prover escalabilidade e reusabilidade de seus componentes.

A arquitetura prevê a utilização de diversos microsserviços separados em camadas de acordo com suas funcionalidades. Na camada de gerenciamento dos microsserviços, estão presentes serviços que regem o funcionamento da arquitetura como um todo, como funções de autenticação, configuração geral da aplicação, roteamento e descoberta e um *gateway*, que recebe as requisições por parte dos clientes – as aplicações – e repassa os comandos pertinentes para os microsserviços da camada interna da arquitetura.

O *gateway* deve conter a API da arquitetura, o que permite o recebimento das chamadas

das aplicações e a tradução em comandos que podem ser interpretados pelos microsserviços responsáveis pelo gerenciamento de dados. O envio de mensagens para os demais componentes da arquitetura é facilitado pelo serviço de roteamento e descoberta, que mantém um registro das instâncias de serviços disponíveis.

A camada interna da arquitetura é onde ficam os microsserviços associados ao gerenciamento de dados de diversas naturezas. Cada microsserviço se destina ao gerenciamento de algum tipo de dado, mas nenhum deles está atrelado a algum SGBD específico. Eles permanecem genéricos para favorecer a interoperabilidade com o maior número de sistemas possível. Os serviços de gerenciamento de dados não se conectam diretamente à camada de persistência, composta pelos SGBDs, devendo se manter genéricos e, portanto, sem dependências a tecnologias específicas. Para realizar a ligação entre essas duas camadas, a arquitetura prevê uma série de adaptadores, de modo a mapear cada serviço ao SGBD adotado para armazenar os dados.

O trabalho avalia a arquitetura através da descrição de dois protótipos, já implementados, mas não realiza nenhum tipo de avaliação experimental de desempenho ou escalabilidade.

3.1.3.2 O Trabalho de Jita e Pieterse

O trabalho em questão apresenta uma arquitetura de microsserviços voltada para o gerenciamento de dados de sensores em um cenário de assistência médica domiciliar, referido no texto como *In Home Medical Care Services (IHMCS)*. No trabalho, os autores visam definir uma arquitetura que permita a coleta de dados de diferentes sensores e a sua comunicação com possíveis partes interessadas, como médicos, farmacêuticos, cuidadores e parentes dos pacientes.

O objetivo principal da arquitetura é permitir o monitoramento contínuo do estado de saúde do paciente, fornecendo alertas quando as medições se desviarem de padrões estabelecidos, além de permitir o diagnóstico e atendimento remoto no caso de enfermidades e outros eventos adversos. Os dados também podem ser usados com a finalidade de melhorar a qualidade de vida do paciente, ao fornecer informações e conselhos relativos ao seu bem-estar (relativas a exercícios físicos, qualidade do sono, alimentação), e também prover outros tipos de funcionalidades, como aquelas relativas a entretenimento e ao gerenciamento de outros aspectos da vida cotidiana do assistido, como suas finanças.

A arquitetura prevê a utilização de um microsserviço distinto para cada funcionalidade

do sistema. Não existe um repositório central de dados. Na arquitetura, cada microsserviço gerencia o armazenamento do conjunto de dados referente à funcionalidade que desempenha. Na prova de conceito realizada no trabalho, os microsserviços utilizam instâncias do Cassandra, sistema de banco de dados NoSQL colunar.

Dentre os microsserviços, podemos destacar o *vitals*, que coleta e armazena dados de sensores relativos à saúde do paciente, como pressão sanguínea, frequência cardíaca e temperatura, e o serviço *analysis*, que realiza análises de dados, como a comparação dos dados coletados pelo serviço *vitals* com os valores de referência. O serviço *patients* gerencia dados pessoais do paciente, enquanto o serviço *subscribers* gerencia os dados dos usuários que receberão alertas no caso de anomalias detectadas nos dados de saúde. Por fim, os microsserviços *diagnosis* e *treatment* servem para gerenciar dados de diagnósticos e tratamentos prescritos pelos profissionais de saúde responsáveis pelo paciente.

Os microsserviços se comunicam para atingir as funcionalidades globais desejadas. Por exemplo, o serviço *analysis* consulta os dados armazenados pelo serviço *vitals* para obter os dados mensurados pelos sensores e só então poder compará-los com os valores de referência. Para o protocolo de troca de mensagens entre os serviços, a arquitetura prevê várias possibilidades, como fila de mensagens e REST, mas não define nenhum padrão. A arquitetura é implementada em um protótipo como prova de conceito, mas nenhuma avaliação de desempenho é realizada.

3.1.3.3 O Trabalho de Azevedo et al.

Este trabalho apresenta uma arquitetura de microsserviços e persistência poliglota desenvolvida para armazenar, processar e consultar dados geológicos no campo da indústria de óleo e gás. Esses dados representam grandes volumes e são obtidos através de diversas fontes e estruturados em diferentes formatos.

Segundo os autores, nenhum SGBD único atendia a todos os requerimentos para o armazenamento e processamento dos dados utilizados. Os dados utilizados nos sistemas se dividem em dados geológicos, como dados brutos a respeito de medições sísmicas e aqueles referentes a mensurações de poços; e ontologias geológicas, que incluem a representação de relacionamentos entre os dados geográficos e entidades como bacias sedimentárias, litologias, dentre outros conceitos específicos do domínio do conhecimento representado.

Por esse motivo, foi adotada uma solução de persistência poliglota, que se caracteriza pela

utilização de múltiplas tecnologias de bancos de dados para gerenciar os dados do domínio em questão. Na arquitetura proposta, foi realizada a utilização do MongoDB para armazenar os dados geológicos e o AllegroGraph, que é um Triplestore, banco de dados orientado ao armazenamento de Triplas, para armazenar as ontologias geológicas. Para suportar essa heterogeneidade de modelos dos dados armazenados, os autores adotaram uma arquitetura composta por microsserviços, em que cada microsserviço contém uma API RESTful e gerencia um tipo de armazenamento de dados distinto.

Os serviços adotados foram implementados na linguagem Python. Cada serviço foi encapsulado em um contêiner Docker e a ferramenta de orquestração utilizada foi a do Kubernetes. Os serviços fazem uso de servidores NGINX para prover as funções de criptografia, caching, controle de acesso, balanceamento de carga e monitoramento.

O trabalho avalia a arquitetura proposta por meio de uma prova de conceito, realizando quatro consultas associadas a casos de uso comuns da arquitetura. Os resultados são expressos por meio de observações e lições aprendidas durante o processo de desenvolvimento. Desses resultados, pode-se destacar a importância encontrada na utilização de uma camada de serviços de integração de dados para encapsular as regras de integração, abstraindo-se as diferenças entre os diferentes mecanismos de armazenamento utilizados. Em relação ao aspecto do desenvolvimento de arquiteturas de microsserviços, os autores também destacam a importância de se utilizar tecnologias padrão para a especificação de contratos de serviços.

3.1.3.4 O Trabalho de Maraver et al.

O trabalho propõe uma arquitetura de microsserviços aplicada ao gerenciamento de dados de publicações científicas. A ideia central é permitir a pesquisa e o gerenciamento de publicações em várias fontes de pesquisa distintas, como aquelas fornecidas por Elsevier, Wiley, Springer e Nature, dentre outras, além de automatizar o monitoramento contínuo das buscas por novas publicações.

A aplicação descrita no trabalho, chamada PaperBot, consiste numa arquitetura de microsserviços, com cada uma das funcionalidades de negócio sendo provida através de um serviço coeso. Os serviços são implementados na linguagem Java e a comunicação entre os serviços se dá através de uma REST API.

Os microsserviços presentes na arquitetura podem ser implementados de forma independente uns dos outros e cada um se comunica diretamente com o banco de dados ou aplicação

web associada. As funcionalidades dos microsserviços incluem o gerenciamento de dados - mais especificamente a realização das operações CRUD (Create, Read, Update e Delete) sobre os registros - de palavras chave, de publicações (artigos) e de metadados.

Esses dados são armazenados em repositórios do MongoDB, sistema de gerenciamento de banco de dados NoSQL orientado a documentos, escolhido devido a suas propriedades de escalabilidade em clusters, flexibilidade de esquema e capacidade de agregação de dados.

A avaliação do projeto é realizada de forma quantitativa, comparando-se o PaperBot ao procedimento convencional de revisão sem a utilização do PaperBot. Foi observada um aumento considerável do número de artigos processados durante a revisão da literatura com a utilização do PaperBot, ao passo em que a carga de trabalho da equipe de avaliação foi mantida constante. Essa diferença se deveu predominantemente à automação da geração e aplicação de combinações de palavras chave, da execução de pesquisas periódicas, detecção de resultados duplicados e extração das informações bibliográficas (metadados) dos trabalhos. A ferramenta permitiu, então, quintuplicar a quantidade de artigos identificados e eliminar o envolvimento humano em atividades automatizáveis.

3.1.3.5 O Trabalho de Ye et al.

A plataforma proposta neste trabalho tem como objetivo integrar diversos sistemas de bancos de dados NoSQL do tipo time series database (TSDBs) em uma ferramenta de *benchmarking*. A plataforma faz parte de uma arquitetura que envolve a utilização de microsserviços para a comunicação de dados e a mensuração dos *benchmarks*.

A arquitetura envolve cinco camadas: infraestrutura, armazenamento de dados, transporte de mensagens, implementação de cargas de trabalho (workloads) e interface de usuário. A camada de infraestrutura é a camada de *hardware* ou de máquinas virtuais em que os bancos de dados irão ser implementados. A camada de armazenamento de dados é composta pelos TSDBs que serão integrados na plataforma. Apesar de ser voltada para a utilização dessas TSDBs, a plataforma também é capaz de integrar outros bancos de dados NoSQL, como aqueles orientados a documentos (MongoDB, CouchBase), chave-valor (Redis, Memcached) e colunares (Cassandra, HBase).

O núcleo da arquitetura é a camada de transporte. Esta consiste na utilização de microsserviços que se comunicam através do mecanismo de fila de mensagens Kafka. Todos os bancos de dados NoSQL utilizados têm interface com o Kafka e podem, então, receber e responder

aos comandos enviados através das camadas superiores. A plataforma utiliza o Kafka para simular fluxos de dados reais de sensores, permitindo a execução dos *benchmarks* desejados.

A camada de implementação de cargas de trabalho contém os mecanismos necessários para a realização dos cenários de testes requeridos para o processo de *benchmarking*. Isso inclui a inserção de carga, a modificação de quantidade de requisições e de dados e a execução de consultas. A camada de interface, por fim, permite a interação do usuário com as ferramentas de *benchmarking*, incluindo a escolha dos bancos de dados e operações a serem utilizadas e a exibição dos resultados.

A avaliação do projeto inclui a execução de experimentos de *benchmarking* com um conjunto de TSDBs. Os resultados são medidos em termos de utilização de CPU e memória e desempenho de consultas. A arquitetura também é avaliada de um ponto de vista qualitativo, sendo comparada com a utilização de ferramentas de *benchmarking* como YCSB e BigDataBench em termos de características como extensibilidade, usabilidade, maturidade de software, suporte a diferentes bancos de dados NoSQL e quantidade de *benchmarks* e métricas de performance suportadas.

3.1.4 Estudo Comparativo

Todos os trabalhos descritos nesta seção promovem o gerenciamento de dados em SGBDs NoSQL através de microsserviços, com alguns deles utilizando REST APIs. Alguns deles têm como objeto a própria arquitetura de microsserviços e também realizam avaliações experimentais, assim como feito na pesquisa descrita nesta dissertação. Esse é o caso de (MENDES et al., 2019), que propõe uma arquitetura de microsserviços para o gerenciamento de dados de IoT no ramo da medicina. O trabalho realiza uma avaliação experimental de desempenho e escalabilidade em relação a uma arquitetura monolítica, tendo resultados favoráveis à de microsserviços. De forma semelhante, Lu, Xu e Lan (2019) descreve uma plataforma cuja finalidade é o gerenciamento de dados de características óticas de objetos espaciais. A arquitetura é de microsserviços e utiliza REST API para comunicação com os clientes. A arquitetura, diferente da que é proposta nesta dissertação, utiliza um mecanismo híbrido de armazenamento de dados, parte em um SGBD relacional, parte em sistemas NoSQL. Uma avaliação experimental utilizando MySQL, MongoDB e Redis é descrita e demonstra uma melhora de desempenho no armazenamento híbrido em relação ao relacional.

Uma outra categoria de trabalhos envolve *frameworks* ou outros tipos de ferramenta que

se destinam à criação de componentes para arquiteturas de microsserviços. Kathiravelu et al. (2019) descrevem um *middleware* para a criação de REST APIs em uma arquitetura de microsserviços destinada ao gerenciamento de dados em diversas fontes. O trabalho, a exemplo do objeto desta dissertação, realiza uma avaliação de desempenho para mensurar o desempenho da arquitetura, através de um protótipo que utiliza o MongoDB. Em uma linha semelhante, (VIENNOT et al., 2015) propõem um *framework* para criação de microsserviços com persistência poliglota, que permitem o armazenamento e gerenciamento de um mesmo conjunto de dados em vários SGBDs diferentes. Uma avaliação do desempenho em aplicações criadas com o *framework* foi realizada com o objetivo de mensurar o *overhead* introduzido pela aplicação. No entanto, as avaliações discutas aqui não fazem uso de NF-es.

Outros pesquisadores adotaram arquiteturas de microsserviços, NoSQL e REST, mas não realizaram avaliações experimentais de desempenho. Braun et al. (2017) tratam de dados de sensores, propondo uma arquitetura genérica de microsserviços para o gerenciamento de dados ambientais. A arquitetura proposta é semelhante àquela que foi desenvolvida para o estudo desta dissertação, na medida em que também contém uma REST API para comunicação com aplicações externas e um serviço de gerenciamento de dados associado a cada tipo de SGBD específico. Para este trabalho de 2017, um protótipo foi construído, mas nenhuma avaliação experimental de desempenho foi realizada. Azevedo et al. (2019) também propõem uma arquitetura de microsserviços com persistência poliglota, desta vez visando armazenar, processar e consultar dados geológicos da indústria de óleo e gás. A arquitetura também utiliza REST APIs e microsserviços individuais associados ao gerenciamento de dados de um SGBD específico. O trabalho avalia o processo de desenvolvimento e a adequação do trabalho a um conjunto de requisitos funcionais, mas não realiza avaliação de desempenho.

Maraver et al. (2019) propõem uma aplicação para auxiliar no gerenciamento de dados de publicações científicas, que inclui uma arquitetura de microsserviços, com a comunicação entre eles ocorrendo através de REST API e o armazenamento de dados utilizando o MongoDB. A avaliação, no entanto, é realizada em termos do esforço de equipes de pesquisa com e sem a utilização da ferramenta, e não em termos de desempenho.

Em Ye et al. (2020), há a descrição uma plataforma construída em uma arquitetura de microsserviços para gerenciamento de dados em NoSQL, com o objetivo de se realizar o *benchmarking* de bancos de dados de séries temporais (TSDBs). A arquitetura, diferentemente da proposta neste trabalho, utiliza um sistema de fila de mensagens para a comunicação entre os serviços. A avaliação consiste na criação de um protótipo e na comparação das funcionalidades

e características da plataforma de *benchmarking* em relação a outras plataformas existentes, além da mensuração da utilização de recursos de memória e processador. Por fim, Jita e Pieterse (2018) propõe a utilização de uma arquitetura de microsserviços para o gerenciamento de dados de sensores em cenários de assistência médica domiciliar. Na arquitetura, os dados são armazenados em um SGBD NoSQL colunar, como o CassandraDB. A arquitetura é implementada em um protótipo como prova de conceito, mas nenhuma avaliação de desempenho foi realizada.

3.2 NOTAS FISCAIS ELETRÔNICAS

Nesta seção, são descritos e analisados trabalhos que tratam de sistemas de software que de alguma forma realizam o gerenciamento de dados de notas fiscais.

3.2.1 REST APIs

O trabalho de Debastiani (2017) propõe uma REST API escrita na linguagem Java, em uma arquitetura de microsserviços, que contém funcionalidades associadas à emissão de notas fiscais eletrônicas, processo não descrito nesta dissertação. A API se destina a extrair informações de um banco de dados para a geração das notas fiscais eletrônicas, sua assinatura eletrônica e envio para o webservice órgão fiscal para autorização. Além disso, permite a realização de outras operações de gerenciamento das notas, como o cálculo da DIFAL (diferença de alíquotas do ICMS entre estados) e o cancelamento de uma nota fiscal emitida.

Já o trabalho de Gomes (2018) consiste na proposta de uma plataforma colaborativa para o armazenamento e consulta de notas fiscais. A plataforma utiliza uma REST API para permitir o *upload* de notas fiscais em um servidor *web*, a consulta dos produtos contados nas notas e a listagem dos produtos existentes. A exemplo do protótipo da arquitetura produzido no âmbito desta pesquisa, a plataforma utiliza um servidor web produzido com o *framework* Flask e utiliza o SGBD MongoDB para o armazenamento das notas fiscais.

3.2.2 Aplicações de Análise de Dados e *Business Intelligence*

O trabalho de SANTOS NETO (2018) utiliza os dados de notas fiscais eletrônicas para a extração de informações mercadológicas através de mineração de dados. O trabalho se vale de ferramenta do site da Secretaria da Fazenda do Estado de Pernambuco (SEFAZ-PE) que permite a extração de notas fiscais a partir de sua chave, mediante a solução de um CAPTCHA.

O mecanismo de extração consiste basicamente em um gerador de chaves de notas fiscais e de uma ferramenta de quebra de CAPTCHA. Cabe ressaltar que a ferramenta não poderia ser reutilizada no presente trabalho para a etapa de geração do banco de dados de notas fiscais, uma vez que a SEFAZ-PE alterou o mecanismo de CAPTCHA para o reCAPTCHA do Google, desde então. Após a extração, os dados são armazenados em um SGBD relacional e então minerados para a extração das informações de interesse do autor.

O trabalho de Brandão Filho, Alves e Bertoncini (2018) realizou um processo de análise de notas fiscais com o objetivo de se extrair informações a respeito do transporte urbano de cargas na cidade de Fortaleza-CE. As notas fiscais foram extraídas de um banco de dados que continha notas cujo emitente ou destinatário era situado na capital cearense. A arquitetura descreve o processo de ETL (extração, transformação e carga) realizado através do software QlikSense e descreve as conclusões relativas à dinâmica do transporte de cargas na cidade de Fortaleza.

Já (SANTOS, 2015) tem finalidade semelhante, a extração de informações de notas fiscais para subsidiar o planejamento do transporte urbano de cargas. O autor utiliza um banco de dados de notas fiscais da Secretaria da Fazenda do Distrito Federal para extrair dados de 25 campos nas NF-es, com o auxílio da ferramenta QlikView. De forma semelhante ao que foi realizado neste trabalho (Seção 5.4.2), o autor complementa o banco de dados das notas fiscais com outros bancos de dados que contêm informações sobre os emitentes das notas. No trabalho, é usado o cadastro fiscal de contribuintes do Distrito Federal no lugar do banco de dados do CNPJ.

Por fim, o trabalho de Andrade (2013) propõe o sistema GANFE (Gerenciamento e Auditoria de Notas Fiscais Eletrônicas), que utiliza uma arquitetura monolítica e um SGBD relacional para permitir o gerenciamento de notas fiscais, com a finalidade de organizá-las em um *data warehouse*, para que possam ser utilizadas como base para a tomada de decisão. O GANFE permite a execução de uma série de operações predefinidas e a interação com o usuário se dá através de uma interface de usuário contida em uma página HTML disponibilizada através de um servidor web.

3.2.3 Aplicações de Rastreabilidade e Segurança das NF-e

Guilhen et al. (2021) propõem aliar a utilização de *blockchain* à escrituração da NF-e para permitir que se rastreie mercadorias da linha de produção até o consumidor final, de modo a

coibir furtos e desvios. Ele consiste em se extrair o código de barras do produto manufaturado e comercializado da NF-e, inserindo-o em uma *blockchain*, que serviria para o monitoramento das etapas seguintes do ciclo de vida do produto. O trabalho, a exemplo deste, utiliza o método *Design Science Research (DSR)* para a elaboração do sistema.

Denny, Paulo e Neves (2021), em linha similar, sugerem a integração de tecnologias de contabilidade distribuída (distributed ledger technologies - DLT), como *blockchain* ao sistema da Nota Fiscal Eletrônica. O trabalho cita como objetivos o combate à evasão fiscal, a otimização dos mecanismo de substituição tributária e a viabilização de um mercado de comercialização de crédito tributário. O trabalho, no entanto, somente sugere a adoção da tecnologia e apresenta argumentos para tal, sem descrever uma arquitetura ou processo para essa implementação.

Por fim, (BARBOZA, 2018) propõe uma modificação no processo de assinatura digital da NF-e, com a adoção de uma nova metodologia baseada na autenticação multifatorial de *hardware* biométrico e *smart cards*. Tal modificação tem como base uma avaliação que aponta para a ausência de garantias relativas à autenticidade e o não-repúdio das NF-es emitidas no processo corrente.

3.3 CONCLUSÕES

Neste capítulo, foram descritos e analisados trabalhos recentes que tratam de arquiteturas de microsserviços que realizam o gerenciamento de dados em bancos de dados NoSQL. Essas arquiteturas são semelhantes àquela que é descrita e avaliada nesta dissertação, que é de microsserviços e utiliza uma REST API e um SGBD NoSQL para gerenciar notas fiscais eletrônicas. Também foram apresentados trabalhos que tratavam de qualquer aspecto relativo ao gerenciamento de notas fiscais eletrônicas. Esses trabalhos são relevantes pois retratam características e dificuldades encontradas no processo de obtenção e gerenciamento de NF-es.

Constatou-se que há diversos trabalhos e aplicações presentes na literatura que tratam dos tópicos de microsserviços, NoSQL, REST APIs e Nota Fiscal Eletrônica, mas nenhum deles realiza uma análise experimental comparativa do gerenciamento de NF-es, confrontando arquiteturas tradicionais com arquiteturas de microsserviços que utilizam NoSQL.

No Capítulo 4, será descrito o processo de pesquisa realizado para a identificação de problemas com arquiteturas atuais de gerenciamento de NF-e e para o desenvolvimento de uma arquitetura de gerenciamento que apresente soluções a esses problemas. O capítulo também inclui uma descrição da arquitetura tradicional do TCE-PE e da arquitetura ControleNF, pro-

posta neste trabalho, utilizando o modelo C4 descrito na Seção 2.5.

4 A ARQUITETURA CONTROLENF

Neste capítulo, será descrito o desenvolvimento de uma arquitetura de software que endereça os problemas descritos nas seções 1.2 e 2.7, decorrentes do gerenciamento de notas fiscais eletrônicas por parte de órgãos de controle. Na Seção 4.1, as três primeiras etapas do *Design Science Research Framework* (DSRF) são descritas. Elas incluem a identificação do problema e a formalização dos princípios e requisitos definidos para o desenvolvimento de uma solução, além da descrição do processo de desenvolvimento. Na Seção 4.2 é descrito o artefato produzido no DSRF, que consiste na arquitetura de microsserviços chamada de ControleNF, destinada ao gerenciamento de dados de notas fiscais eletrônicas em órgãos de controle. A Seção 4.3 conclui o capítulo.

4.1 ESPECIFICAÇÃO DA ARQUITETURA CONTROLENF

A arquitetura ControleNF foi elaborada com a utilização de princípios do DSRF (HEVNER et al., 2004). Esse *framework* funciona como guia para a aplicação da ciência do *design* aos sistemas de informação, com o objetivo de gerar artefatos inovadores que contribuam para o entendimento e para a solução de determinado problema. O *framework* consiste em um processo de seis passos, que vão desde a identificação do problema e da motivação, passando pela definição dos objetivos, especificação da solução, incluindo seu *design* e desenvolvimento, e finalizando com as etapas de demonstração, avaliação e comunicação (HEVNER; CHATTERJEE, 2010).

A seguir, será descrito como foi conduzido o processo de criação de ControleNF, conforme as atividades do DSRF.

4.1.1 Identificação do problema e motivação

O primeiro passo no DSRF consiste em definir o problema de pesquisa e justificar o valor de sua solução.

A demanda pela produção de uma arquitetura de gerenciamento de NF-e surgiu da utilização das notas fiscais pelo TCE-PE, após convênio (TCE-PE, 2016) firmado com a SEFAZ-PE para compartilhamento desses dados entre os dois órgãos. Como parte deste convênio, a SEFAZ-PE envia diariamente as NF-es emitidas para os órgãos jurisdicionados do TCE-PE,

como prefeituras municipais, secretarias, autarquias e outros órgãos e entidades do Estado de Pernambuco fiscalizados pelo Tribunal.

Os documentos XML recebidos pelo TCE-PE são então extraídos de arquivos compactados (extensão .zip) e transformados através de procedimentos armazenados (*stored procedures*), para então serem inseridos nas tabelas de um SGBD relacional. A partir daí, as notas ficam disponíveis para utilização das equipes de fiscalização, que as acessam através dos sistemas de informação do Tribunal, que acessam os dados através de conexões ODBC.

No banco de dados relacional, os dados das notas fiscais são divididos em quatro tabelas principais, que foram elaboradas de acordo com as regras de normalização de SGBDs relacionais: Notas, Itens, Emitentes e Destinatários. Estas tabelas armazenam, respectivamente, os campos de identificação e caracterização das NF-es, os dados detalhados a respeito dos itens adquiridos em cada nota, dados relativos aos emitentes (empresas que fornecem os produtos e serviços) e dados dos órgãos e entidades adquirentes dos produtos ou serviços.

Os dados completos das notas fiscais no formato XML ficam armazenados em um coluna do tipo XML de uma tabela auxiliar, para caso seja necessário o reprocessamento futuro. Essa necessidade surge principalmente quando se percebe a necessidade de incorporar algum campo das notas fiscais ao esquema de dados que não havia sido previamente incluído na carga. Neste caso, os dados em XML são reprocessados para extrair o valor do campo de todos os documentos que o contenham.

Como já discutido na Seção 2.4, os bancos de dados relacionais são adequados para um grande número de aplicações, porém apresentam limitações importantes em determinados cenários de utilização. Quando se trata do gerenciamento de NF-e, pode-se destacar, principalmente, a falta de flexibilidade de esquema para lidar com o formato hierárquico dos arquivos XML, com a presença de campos multivalorados e opcionais, e com as frequentes alterações no leiaute da NF-e, com a adição periódica de novos campos. Somando-se a essa inflexibilidade, as consultas a bancos de dados relacionais também podem apresentar perda de desempenho quando é necessário realizar junções entre tabelas, o que pode tornar lento o processo da análise desses dados por parte dos auditores do TCE-PE em determinados cenários.

A manutenção da lógica de extração e processamento das notas fiscais na forma de procedimentos armazenados em um SGBD também tem a desvantagem de tornar a arquitetura pouco portátil. Não seria possível reaproveitar o código utilizado no TCE-PE em outros órgãos de controle que também tenham convênio com os respectivos órgãos fazendários, a não ser que estes utilizassem exatamente o mesmo SGBD (Microsoft SQL Server), visto que até

mesmo os dialetos SQL de SGBDs relacionais distintos possuem diferentes funções e tipos de dados para se trabalhar dados XML.

Nesse cenário, tem-se que o desenvolvimento de diferentes rotinas e sistemas para a realização das mesmas atividades em instituições do setor público vai de encontro à lógica da economicidade, princípio fundamental da administração pública consagrado na Constituição Federal. O reaproveitamento e desenvolvimento conjunto de soluções por órgãos de diferentes esferas contribuem com a integração do serviço público e representam ganhos de eficiência para os órgãos que participam dessas iniciativas. Com base nesses aspectos, uma eventual solução para o problema do gerenciamento de notas fiscais requer três requisitos de *design*:

1. Uma arquitetura para gerenciamento de NF-e deve ser capaz de armazenar os documentos com flexibilidade de esquema.

2. A arquitetura deve ser de fácil manutenção e ser portátil para diferentes ambientes, de modo a permitir sua aplicação em diferentes órgãos fiscais e de controle.

3. A arquitetura deve ter bom desempenho nas operações de inserção e consulta de notas fiscais.

Esses requisitos podem se traduzir, de forma resumida, nas características de portabilidade, manutenibilidade, desempenho e escalabilidade. Essas características irão guiar a elaboração dos princípios de *design* da solução proposta, que corresponde à atividade seguinte do DSRF.

4.1.2 Definição dos objetivos da solução almejada

A etapa seguinte do DSRF consiste em estabelecer os objetivos da solução proposta, inferidos a partir dos requisitos anteriormente definidos. Esses objetivos podem ser expressos na forma de princípios de *design*, que demonstram como efetivamente os problemas apresentados serão atendidos pela solução.

Nesta etapa, foram definidos os seguintes princípios de *design*:

1. A solução adotada deve ser uma arquitetura de microsserviços.

2. A arquitetura deve utilizar padrões abertos nas interfaces de comunicação com aplicações externas.

3. Deve-se utilizar um SGBD NoSQL para armazenamento das notas fiscais completas, em formato semiestruturado.

4. A solução deve apresentar tempo de inserção e consultas de notas fiscais igual ou inferior a soluções existentes.

A arquitetura de microsserviços está relacionada às propriedades de portabilidade e manutenibilidade, além da escalabilidade. A separação das funcionalidades em pequenos serviços permite incluir ou modificar serviços de acordo com a necessidade de cada ambiente em que a arquitetura é implementada, sem que seja necessário realizar alterações nos demais. Os microsserviços também podem permitir a escalabilidade horizontal, com o balanceamento de carga ocorrendo entre réplicas do mesmo serviço.

A utilização de padrões abertos na comunicação de sistemas e usuários externos com a arquitetura contribui para a sua portabilidade, já que não há a necessidade de aprendizado ou aquisição de alguma tecnologia proprietária por parte das equipes que irão implementá-la em seus ambientes.

O SGBD NoSQL orientado a documentos capaz de armazenar as notas em formato semiestruturado (como XML ou JSON), por sua vez, contribui para a melhoria da manutenibilidade, ao permitir o armazenamento das notas em sua integridade em formato análogo ao que é disponibilizado pelo fisco. Isso também remove a necessidade de se adaptar os dados ao modelo relacional, o que causa um impacto negativo de desempenho tanto no carregamento das notas fiscais - devido às transformações necessárias para o formato de tabelas, quanto no acesso às notas - devido à necessidade de junções entre as tabelas no momento da consulta.

Por fim, o princípio que trata dos tempos de inserção e de consulta das NF-es se destina a restringir as possíveis soluções àquelas que tragam algum ganho de desempenho e escalabilidade, ou, em último caso, não representem uma perda nesses aspectos em relação a soluções existentes.

4.1.3 *Design* e desenvolvimento da arquitetura ControleNF

Nesta etapa do DSRF há o desenvolvimento do artefato propriamente dito, que é o objeto que representa uma contribuição científica e contribui para a solução do problema de *design* apresentado. O termo objeto, aqui, é utilizado em sentido amplo e pode significar um método, modelo, ou constructo, dentre outras possibilidades (HEVNER et al., 2004).

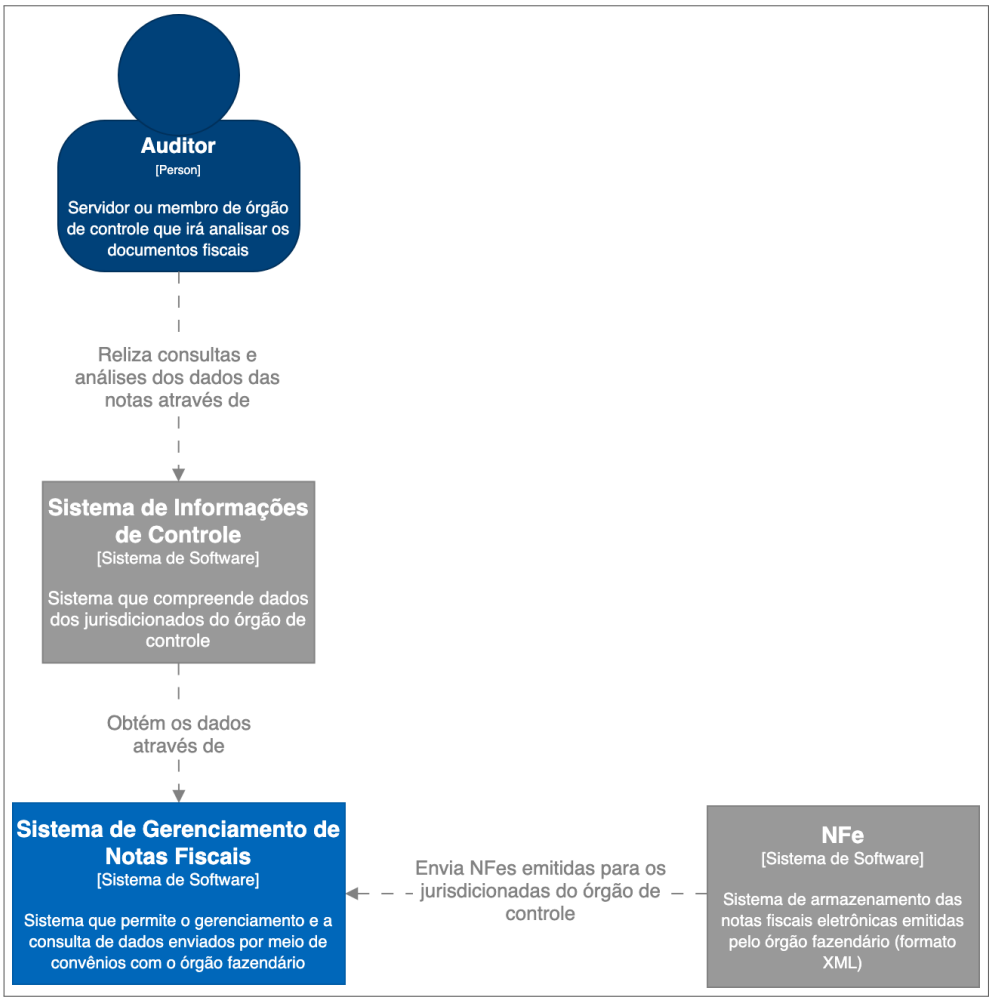
Antes de adentrar na especificação do artefato criado neste processo, é importante descrever o processo tradicional de compartilhamento e gerenciamento das notas fiscais utilizado no cenário observado, que é aquele que envolve SEFAZ-PE e TCE-PE. Neste sentido, foi uti-

lizado o modelo C4 para descrever tanto o processo tradicional quanto o artefato resultante do processo de pesquisa do DSRF.

O primeiro diagrama previsto nesse modelo, conforme descrito na Seção 2.5 desta dissertação, é o diagrama de contexto do sistema. Este diagrama é útil para identificar quais são os componentes e atores que interagem com o sistema e descrever o contexto em que ele está inserido, sem se preocupar com os componentes internos.

No contexto do TCE-PE, as notas fiscais são enviadas pela SEFAZ-PE através da rede da Agência Estadual de Tecnologia da Informação do Estado (ATI). Elas são extraídas do sistema de gerenciamento de NF-e da SEFAZ-PE e enviadas diariamente em arquivos compactados, para então serem carregadas no sistema de gerenciamento de notas fiscais eletrônicas do TCE-PE, que na solução tradicional é um SGBDR.

Figura 4 – Diagrama de contexto no TCE-PE



Fonte: Elaborada pelo autor (2022)

Esses dados são consultados pelos auditores do TCE-PE através, principalmente, do sis-

tema Tome Conta[1], que é um sistema que agrega informações de diversas fontes que podem ser úteis às atividades de fiscalização do Tribunal. Os dados também são ocasionalmente disponibilizados na forma de painéis de BI, que provêm outras perspectivas de análise sobre esses mesmos dados.

Esse sistema de gerenciamento de NF-e, em azul no diagrama de contexto do TCE-PE (Figura 4), será substituído pelo artefato desenvolvido na pesquisa descrita nesta dissertação, com base no DSRF. Nesse nível, portanto, o mesmo diagrama de contexto pode ser utilizado tanto para representar a arquitetura tradicional, quanto a arquitetura desenvolvida.

O próximo diagrama do modelo C4 é o diagrama de contêiner (Figura 5). Nesse diagrama, o sistema de software é detalhado, havendo a separação dos componentes internos da arquitetura. Na arquitetura tradicional, o sistema de gerenciamento de NF-es é dividido em três partes: a rotina de carga de dados, o sistema de arquivos que armazena as notas fiscais compactadas e o SGBD relacional utilizado para processar e armazenar as notas fiscais.

No contexto do TCE-PE, as notas fiscais são enviadas pela SEFAZ-PE através da rede da Agência Estadual de Tecnologia da Informação do Estado (ATI). Os conjuntos de notas são recebidos e salvos em um diretório de um servidor local através de operações contidas em um arquivo *batch*, programado para ser executado diariamente.

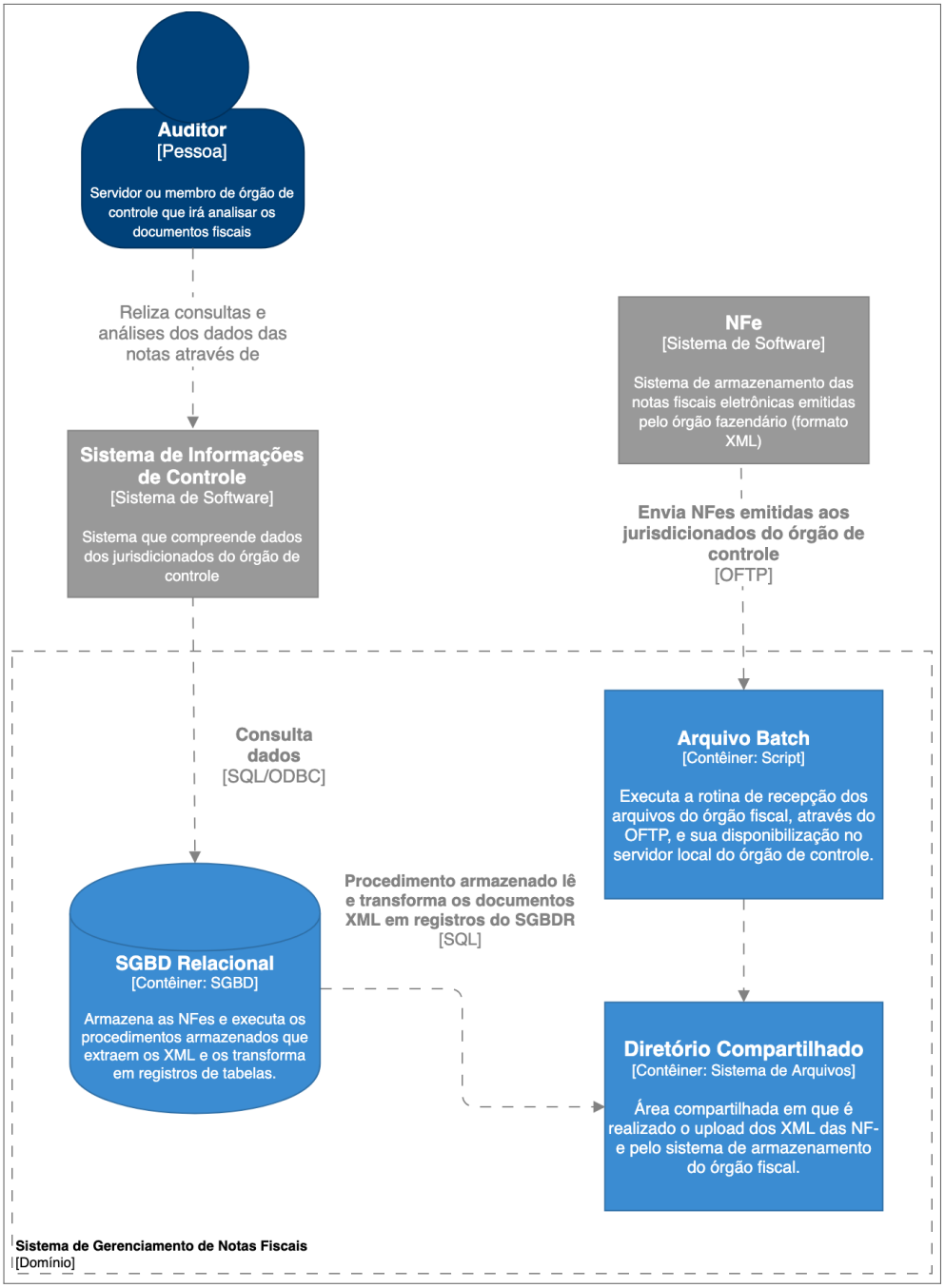
As notas fiscais são disponibilizadas em arquivos compactados (.zip), cada um contendo todas as notas fiscais emitidas para os órgãos juridiscionados do TCE-PE em uma determinada data, no formato original XML. As notas então ficam disponíveis para que a rotina diária de processamento, na forma de procedimentos armazenados pelo SGBD relacional, possa executar as etapas de extração, transformação e carga dos dados nas tabelas relacionais.

Após carregados nas tabelas do SGBD, os dados podem ser consultados pelos auditores através de algum sistema que agrega informações de controle da administração pública. No caso do TCE-PE, o sistema Tome Conta é o principal instrumento que desempenha essa função.

O sistema de informações, por sua vez, consulta os dados do banco de dados através de uma conexão ODBC. Ocasionalmente, usuários avançados podem consultar os dados diretamente através de consultas SQL executadas pelo SGBD, mas esse tipo de acesso é considerado exceção, já que a concessão de permissões para utilização do SGBD para execução de consultas não é amplamente realizada para usuários finais.

[1] Disponível em <https://sistemas.tce.pe.gov.br/tomeconta>

Figura 5 – Diagrama de contêiner - Arquitetura tradicional do TCE-PE



Fonte: Elaborada pelo autor (2022)

O diagrama de código do modelo C4, que é o mais detalhado previsto na especificação do modelo, não foi elaborado para este trabalho, uma vez que é opcional e geralmente adotado somente para componentes mais complexos. Como a arquitetura de microsserviços separa as funções do sistema em componentes simples, que executam uma funcionalidade bem definida e não um grande conjunto de operações, julgou-se desnecessária sua inclusão.

Na seção seguinte, será realizada a descrição do artefato produzido neste processo de

pesquisa. O artefato tem a função do Sistema de Gerenciamento de Notas Fiscais apresentado nos diagramas anteriores e utiliza uma arquitetura de microsserviços, além de uma REST API e um SGBD NoSQL para apresentar soluções aos problemas identificados com a arquitetura tradicional.

4.2 DESCRIÇÃO DA ARQUITETURA CONTROLENF

No processo do DSRF, o artefato produzido foi uma arquitetura de software destinada ao gerenciamento de notas fiscais eletrônicas e seu compartilhamento entre órgãos fazendários e órgãos de controle, denominada aqui de ControleNF. ControleNF segue os princípios de *design* estabelecidos anteriormente, sendo uma arquitetura de microsserviços que fornece uma REST API para realização das operações CRUD (criação, leitura, atualização e remoção) das notas fiscais eletrônicas, com o armazenamento de dados em SGBD NoSQL orientado a documentos.

Na ControleNF, a interação de usuários e sistemas com o banco de dados se dá através de uma REST API. O ponto de entrada da API (*gateway*) é um microsserviço escrito em Python com a utilização dos pacotes *Flask*[2] e *Connexion*[3]. O *framework Flask* serve para a criação de um servidor web, enquanto o *Connexion* é um módulo que permite a utilização de uma especificação de API no formato do *Swagger*/*OpenAPI*[4].

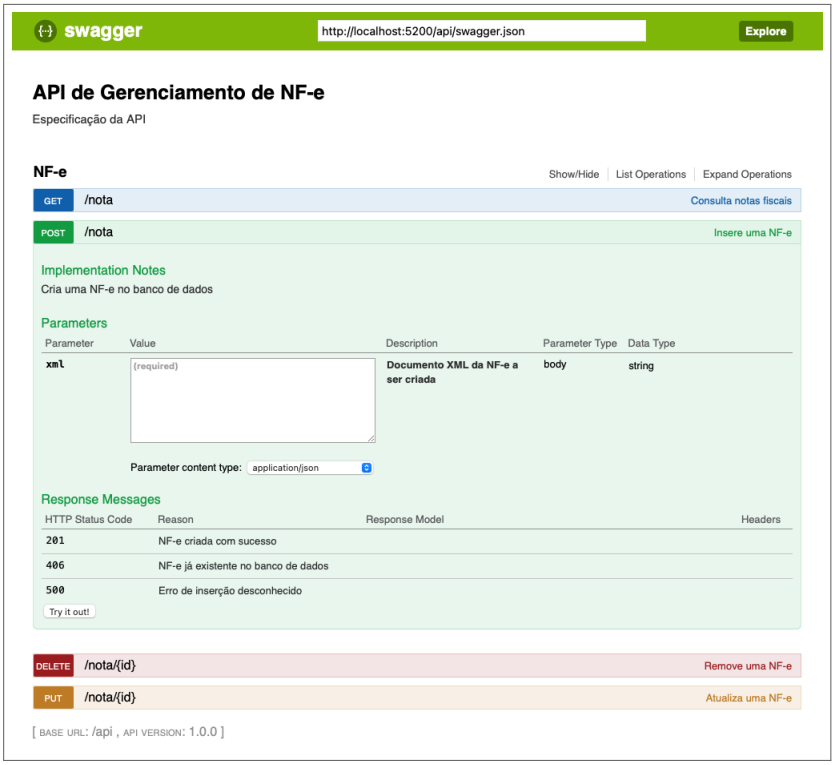
O *Swagger*, renomeado para *OpenAPI* em versões mais recentes, permite a especificação da API através de um só arquivo de configuração no formato YAML ou JSON, que contém as informações necessárias para que a API possa realizar validação dos formatos das requisições e seus parâmetros, das respostas da API e da interface de usuário. A interface é gerada automaticamente com base na configuração, e exibe aos usuários as operações disponibilizadas pela API, além dos formatos das requisições e respostas. A interface funciona como uma documentação dinâmica e interativa da API, o que contribui para a adaptabilidade da arquitetura a outros cenários de aplicação.

A API funciona como um servidor que recebe requisições HTTP de clientes. Um cliente pode ser tanto uma aplicação do órgão fazendário, que realiza o envio das notas fiscais, ou alguma aplicação interna do próprio órgão de controle que realize operações de leitura ou escrita sobre as NF-es.

---

[2] <https://flask.palletsprojects.com/en/2.0.x/>
[3] <https://connexion.readthedocs.io/en/latest/>
[4] <https://swagger.io>

Figura 6 – Captura de tela: Interface do *Swagger*



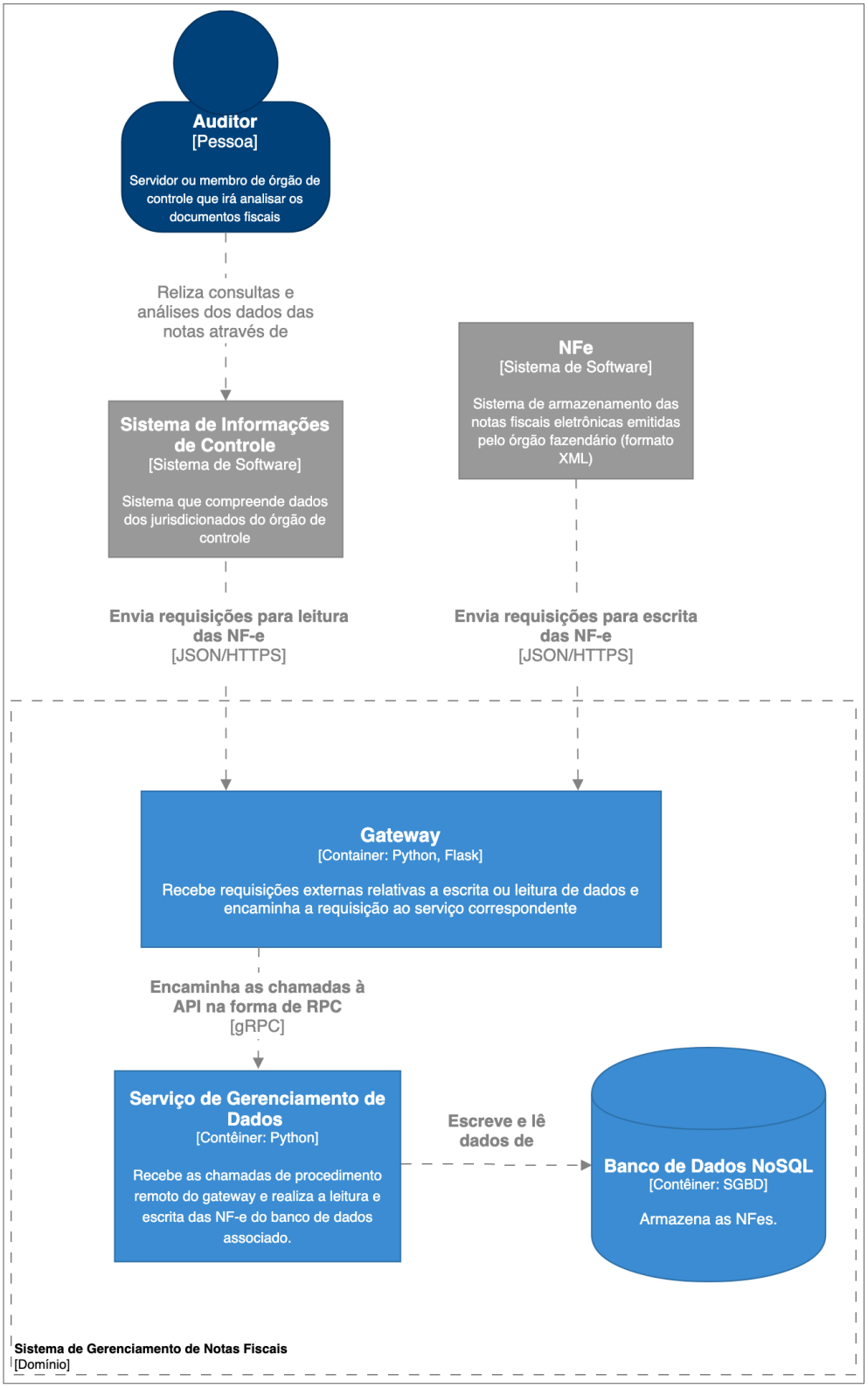
Fonte: Elaborada pelo autor (2022)

Dada a natureza das notas fiscais, que registram alguma operação, como venda de mercadoria, prestação de serviço, importação ou exportação, não costuma haver necessidade de se realizar atualizações de dados, já que os órgãos de controle não têm qualquer tipo de atribuição que implique na modificação dos dados constantes das NF-e.

As operações realizadas com mais frequência são as de inserção ou consulta, que são realizadas por meio dos métodos HTTP PUT e GET, respectivamente. No entanto, a arquitetura também prevê a possibilidade de atualizar os campos de uma nota, através do método POST. Atualizações podem ser eventualmente necessárias para marcar o cancelamento da nota ou devolução do produto, caso o modelo de dados preveja esse tipo de operação. Também é possível remover a nota do banco de dados, através do método DELETE da API.

Essas operações são executadas no banco de dados através de um serviço de gerenciamento de dados, que é um microsserviço que recebe o comando de inserção, remoção, alteração ou leitura de notas fiscais e realiza a comunicação com o SGBD, enviando o comando correspondente e recebendo uma resposta. Esse serviço de gerenciamento de dados é único para cada SGBD utilizado.

Figura 7 – Diagrama de contêiner - Arquitetura do ControleNF



Fonte: Elaborada pelo autor (2022)

Na demonstração do artefato no ambiente do TCE-PE, o serviço de gerenciamento de

dados é escrito em Python e utiliza a biblioteca *PyMongo*[5] para se conectar ao SGBD MongoDB. O MongoDB foi adotado por ser um SGBD orientado a documentos, que é capaz de armazenar notas fiscais em seu formato semiestruturado sem a necessidade de modificações em sua estrutura, sendo um dos mais proeminentes SGBDs do tipo citados na literatura.

A comunicação entre os serviços se dá através do *gRPC*, *framework* de RPC descrito na Seção 2.3. Foi utilizado um arquivo na IDL do *protobuf* (chamado de arquivo proto ou *proto file*) que define os métodos que podem ser chamados e os formatos das mensagens que podem ser trocadas entre os serviços.

Na implementação criada para o experimento realizado nesta pesquisa, os métodos do RPC correspondem aos mesmos métodos da API, sendo responsáveis pela inserção, atualização, remoção e consulta das notas fiscais. Como um dos pontos da utilização da arquitetura de microsserviços é permitir a evolução do sistema, a definição do RPC pode ser expandida para permitir trocas de outros tipos de mensagens e chamadas de novos métodos entre os serviços.

Também é possível a adição de novas funcionalidades à arquitetura através da incorporação de novos microsserviços. Por exemplo, pode-se adicionar um serviço de *caching* entre o *gateway* da API e o serviço de gerenciamento de dados para reduzir o tempo de resposta a consultas frequentes ou um serviço de ofuscamento de dados para que determinados grupos de usuários possam consultar as notas sem visualizar os dados de seus emitentes.

4.2.1 Detalhamento dos componentes de ControleNF

O modelo C4 também prevê a possibilidade da elaboração de diagramas de componentes, que são diagramas que visam detalhar os componentes internos dos contêineres. Na descrição da arquitetura ControleNF, foram elaborados dois diagramas de componentes, referentes aos contêineres do *gateway* da API e do serviço de gerenciamento de dados. Esses diagramas permitem a compreensão do fluxo interno dos dados nos contêineres e a visualização das dependências entre os componentes.

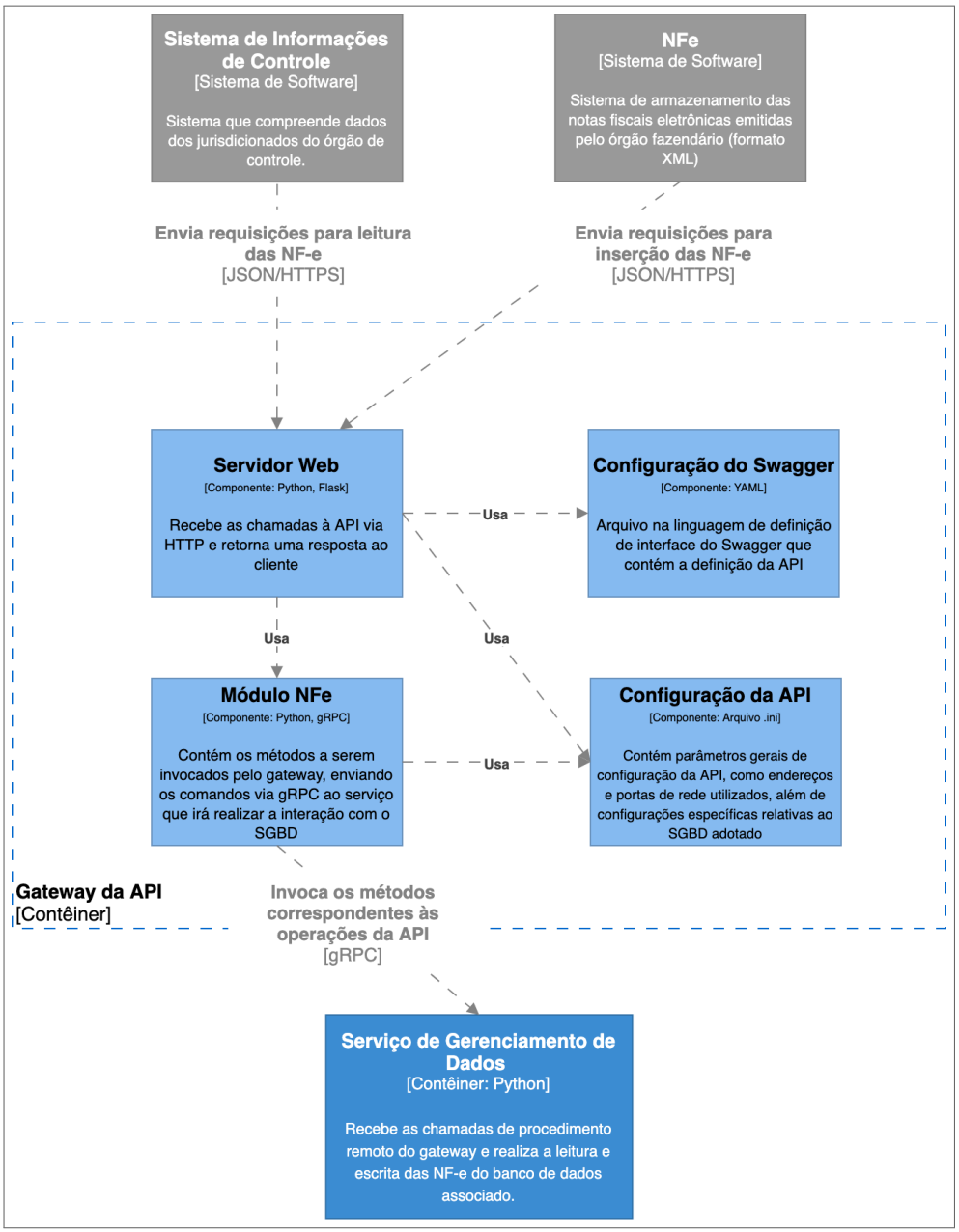
O primeiro dos dois diagramas é o do *gateway* da API, conforme mostra a Figura 8. Essa figura exibe os componentes do servidor web, o módulo NFe e os arquivos de configuração do *Swagger* e da API como um todo. As requisições à API são enviadas pelos clientes ao servidor web, que é um componente escrito em Python que faz uso do *framework Flask* para criar um servidor que recebe e responde a requisições HTTP. O servidor usa o pacote *Connexion* e se

[5] <https://pymongo.readthedocs.io/en/stable/>

baseia no arquivo de configuração do *Swagger* para criar a API no servidor.

O módulo NFe, por sua vez, é um *script* em Python que contém os códigos dos métodos invocados pela API. No protótipo desenvolvido para essa pesquisa, os métodos basicamente convertem as requisições HTTP para o formato do *gRPC* e fazem as chamadas de procedimentos remotos ao serviço de gerenciamento de dados. Em um ambiente de produção, estes métodos poderiam utilizar outros microsserviços para realizar algum tipo de processamento nas chamadas da API ou nas respostas, na hipótese da extensão da arquitetura para adição de novas funcionalidades.

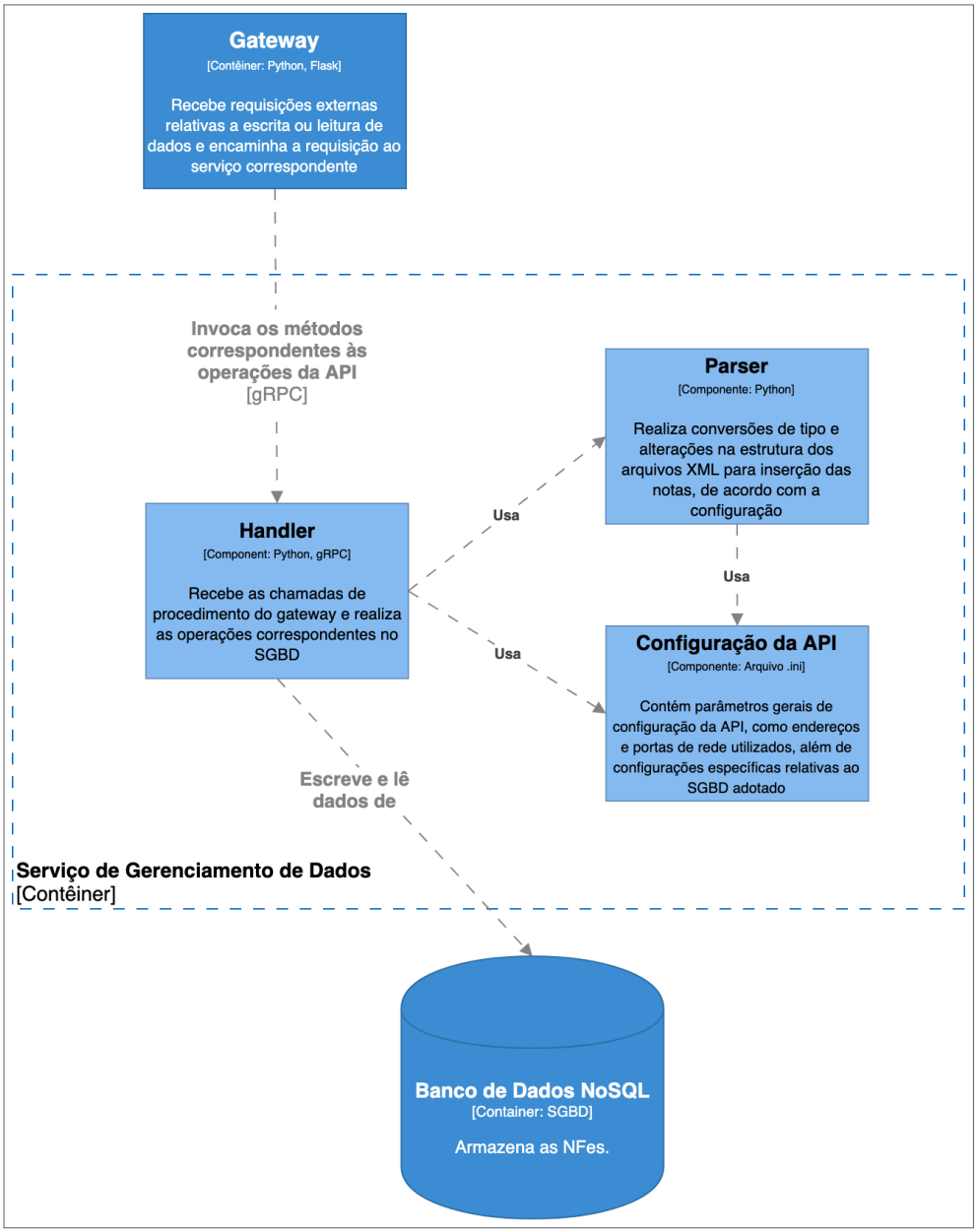
Figura 8 – Diagrama de componentes - Contêiner do *gateway* da API



Fonte: Elaborada pelo autor (2022)

O arquivo de configuração do *Swagger* é um arquivo YAML na linguagem de definição de interface do *Swagger*, que permite especificar os métodos fornecidos pela API, os parâmetros permitidos para cada chamada e as respostas possíveis para cada uma. O *Connexion* gera automaticamente uma interface gráfica no servidor com as informações da API como exemplificado na Figura 6.

Figura 9 – Diagrama de componentes - Contêiner do serviço de gerenciamento de dados



Fonte: Elaborada pelo autor (2022)

O arquivo de configuração da API, por sua vez, é um arquivo no formato .ini criado para ser lido com a biblioteca *ConfigParser* da linguagem Python. Ele contém parâmetros de configuração gerais da aplicação, como endereços IP e portas usadas na definição dos

servidores e clientes, além de configurações internas do *gRPC*, como o tamanho máximo das mensagens que podem ser enviadas e recebidas por cada serviço.

O segundo diagrama de componentes é o diagrama do serviço de gerenciamento de dados. Neste diagrama, são incluídos os componentes do *handler*, do *parser* e o arquivo de configuração da API, este último já descrito no diagrama do gateway. O *handler* é a parte do serviço que lida com as chamadas ao *gRPC*, contendo os comandos a serem realizados no âmbito de cada operação da API.

O *parser*, por sua vez, é a parte do script que é responsável por modificar a estrutura das NF-es para se adaptar ao esquema de dados adotado, antes de ser realizada a inserção. Esse componente pode ou não estar presente na arquitetura. Caso não seja de interesse do órgão que o implementa, as NF-e podem ser inseridas no SGBD em sua integridade e sem modificações de tipos de dados. O órgão também pode optar por incluir as conversões de tipos e seleção de campos em uma etapa posterior à inserção das notas no banco. No caso do protótipo produzido, que utiliza o MongoDB, SGBD orientado a documentos, o parser altera alguns campos para tipos numéricos e outros para tipo *datetime* antes de inseri-los no SGBD. O *parser* também cria um campo chamado _id, que funciona como identificador único em cada coleção, atribuindo a ele a chave da nota fiscal, composta pelo termo NFe seguido de um código numérico único de 47 algarismos.

4.3 CONCLUSÃO

Neste capítulo foi descrito o processo de desenvolvimento do artefato que visa solucionar problemas relativos ao gerenciamento de notas fiscais eletrônicas realizado por órgãos de controle. Esse processo foi realizado com base no *framework Design Science Research* e culminou com a descrição da arquitetura, realizada com o auxílio do modelo C4 para visualização de software.

Os capítulos seguintes representam as próximas etapas do DSRF. O Capítulo 5 está relacionado à tarefa de demonstração (*demonstration*) do *framework*. Nele, o uso do artefato para resolver o problema de pesquisa será testado em uma simulação do cenário do TCE-PE.

5 DEMONSTRAÇÃO E AVALIAÇÃO DA ARQUITETURA CONTROLENF

5.1 INTRODUÇÃO

No capítulo anterior, foram descritas as primeiras três atividades previstas no DSRF, que são: a identificação do problema e motivação, a definição dos objetivos para uma solução e a etapa de *design* e desenvolvimento. Neste capítulo, serão abordadas a terceira e quarta atividades do *framework*, que tratam da demonstração e da avaliação do artefato, respectivamente.

Na Seção 5.2 é descrita a metodologia utilizada para avaliação da arquitetura ControleNF em comparação às arquiteturas tradicionais utilizadas para o gerenciamento de notas fiscais eletrônicas. Essa avaliação consiste em uma análise comparativa qualitativa em relação aos requisitos de manutenibilidade e portabilidade, que é detalhada na Seção 5.3, e de uma avaliação quantitativa experimental que busca mensurar os aspectos de desempenho e escalabilidade, que é descrita na Seção 5.4. Por fim, na Seção 5.5, são discutidos os resultados observados em ambas as formas de avaliação, apresentando a conclusão do capítulo.

5.2 METODOLOGIA DE AVALIAÇÃO

De acordo com o DSRF (HEVNER; CHATTERJEE, 2010), após a etapa de *design* e desenvolvimento do artefato há a atividade de demonstração. Essa atividade consiste em demonstrar a utilização do artefato para resolver uma ou mais instâncias do problema observado. Conforme exposto na Seção 4.1.1, foram estabelecidos três requisitos para o desenvolvimento da solução, que foram resumidos nas características de manutenibilidade, portabilidade, desempenho e escalabilidade. A demonstração da arquitetura, portanto, se destina a avaliar em que grau a arquitetura ControleNF atende a esses requisitos, em comparação com arquiteturas tradicionais existentes. Para esta avaliação, foi adotada como instância do problema o cenário do TCE-PE, que recebe notas fiscais eletrônicas da SEFAZ-PE, conforme descrito na Seção 4.1.3. Esse cenário caracteriza uma configuração de teste desta pesquisa e é chamada de arquitetura tradicional. Ela será comparada com a arquitetura de microsserviços que utiliza os conceitos de REST API e NoSQL.

A avaliação foi dividida em duas partes, uma avaliação qualitativa e uma quantitativa. A avaliação qualitativa se destina a descrever e comparar as duas arquiteturas em função dos requisitos de manutenibilidade e de portabilidade. Ela utiliza o padrão ISO/IEC 25010 para

definir e expandir esses dois requisitos, para que se possa aferir o grau de adequação de cada uma das duas arquiteturas a essas características. Já a avaliação quantitativa é realizada através de um experimento e se destina a mensurar os requisitos de desempenho e escalabilidade. No experimento, são realizadas operações de escrita e leitura de notas fiscais eletrônicas nos bancos de dados, com diferentes volumes de documentos, mensurando-se o tempo de resposta de cada conjunto de operações. Os tempos médios de processamento são coletados para ambas arquiteturas e comparados em termos de uma variação percentual e uma análise sobre os resultados é realizada.

5.3 AVALIAÇÃO QUALITATIVA DAS ARQUITETURAS

O objetivo da avaliação qualitativa é comparar a arquitetura tradicional utilizada no TCE-PE para o gerenciamento das NF-es recebidas da SEFAZ-PE com a arquitetura ControleNF, descrita na Seção 4.1.3, em termos das características de manutenibilidade e portabilidade, que foram identificadas como requisitos no processo do DSRF descrito no Capítulo 4.

Para a avaliação dessas características, foi utilizado o padrão ISO/IEC 25010 (ISO/IEC, 2011), que define um modelo para avaliação de qualidade de software. O padrão lista oito características fundamentais que devem ser levadas em consideração na avaliação da qualidade de sistemas de software, dentre as quais se encontram a manutenibilidade e a portabilidade, que foram detalhadas na Seção 2.6.

Para a estratégia de avaliação desses quesitos, foi adotada uma abordagem semelhante àquela observada em (VILLAÇA; AZEVEDO; BAIO, 2018). A metodologia consiste em se atribuir o valor 1 para cada subcaracterística percebida como um ponto positivo de cada arquitetura, e o valor 0 quando a subcaracterística não se adequar à arquitetura em questão. Ao final, é calculado o percentual de pontos obtidos por cada arquitetura quanto à manutenibilidade e à portabilidade. É importante ressaltar que o resultado não tem a pretensão de ser uma métrica definitiva para avaliação das características, uma vez que a avaliação, apesar de utilizar critérios definidos em um modelo de qualidade de software estabelecido, está sujeita à subjetividade do autor. Os resultados podem ser tomados como um ponto de partida para a análise de viabilidade da arquitetura, para posterior experimentação prática em trabalhos futuros. Os resultados da avaliação são detalhados a seguir.

5.3.1 Resultados da avaliação qualitativa

Em relação à manutenibilidade, foram observadas as seguintes pontuações:

- **Modularidade:** A arquitetura tradicional mantém toda a lógica de carga e leitura das das notas fiscais em estruturas do SGBD relacional. A arquitetura ControleNF é de microsserviços, modular por definição. (Tradicional: 0, ControleNF: 1)

- **Reusabilidade:** Na arquitetura tradicional, os procedimentos e tabelas utilizados para gerenciar notas fiscais são de propósito único, limitando a utilização da arquitetura a funções de leitura e escrita de dados em um banco de dados. Já a arquitetura de microsserviços com REST API, escrita em uma linguagem de propósito geral como Python, permite a incorporação de novas funcionalidades através da adição de novos microsserviços ou até o reuso da estrutura da API para outras funções, substituindo o serviço de gerenciamento de dados por algum que tenha outro propósito. (Tradicional: 0, ControleNF: 1)

- **Analisabilidade:** Por ser uma arquitetura centralizada no SGBD relacional, com todo o processo de carga ficando centralizado em apenas dois procedimentos armazenados, a arquitetura tradicional tende a facilitar a análise na busca por causas de eventuais erros. A modularidade da arquitetura ControleNF torna este processo mais complexo, já que um erro na arquitetura pode ser oriundo de qualquer um de seus componentes discretos. (Tradicional: 1, ControleNF: 0)

- **Modificabilidade:** Como a arquitetura tradicional concentra os processos de carga e leitura das notas fiscais no banco de dados relacional, qualquer alteração em um procedimento armazenado ou tabela pode comprometer alguma outra etapa do gerenciamento das notas. Já a ControleNF, por ser modular, permite a alteração de componentes individuais sem que estas alterações impliquem a necessidade de modificar os demais, desde que as trocas de mensagens entre eles não sejam afetadas. (Tradicional: 0, ControleNF: 1)

- **Testabilidade:** A definição de testabilidade do ISO/IEC 25010 trata tanto da definição de critérios de teste, como da capacidade de aplicá-los ao sistema, produto ou componente. Nesta avaliação, não foram encontradas diferenças significativas entre ambas as arquiteturas. Os critérios de teste da arquitetura tendem a se concentrar nos aspec-

tos de desempenho e escalabilidade, conforme detalhado na Seção 5.4. (Tradicional: 1, ControleNF: 1)

Já em relação à portabilidade, os resultados são listados como se segue:

- **Adaptabilidade:** A arquitetura tradicional apresenta adaptabilidade limitada, uma vez que só pode ser implementada em algum outro ambiente que utilize o mesmo SGBD (Microsoft SQL Server) e o sistema operacional Windows. Já a arquitetura ControleNF pode ser implementada em uma variedade maior de sistemas operacionais, já que tanto a linguagem Python quanto o SGBD MongoDB podem ser executados em diferentes plataformas, incluindo Windows, MacOS e Linux. Durante o desenvolvimento da arquitetura, ControleNF foi instalada e executada tanto no MacOS quanto no Windows sem a necessidade de nenhuma alteração de código ou configuração. (Tradicional: 0, ControleNF: 1)

- **Instalabilidade:** Nenhuma das arquiteturas apresenta obstáculos significativos para a instalação. Na arquitetura tradicional, basta a instalação do cliente OFTP, do SQL Server e a criação das estruturas do banco de dados. Já na arquitetura ControleNF, é necessário somente instalar o Python, juntamente com uma lista de pacotes utilizados e o MongoDB. (Tradicional: 1, ControleNF: 1)

- **Substituição:** Esta subcaracterística é melhor avaliada no caso de componentes específicos da arquitetura. Como na presente avaliação está se considerando a arquitetura inteira, nenhum aspecto das duas arquiteturas representa alguma dificuldade excessiva para a substituição de alguma outra solução existente, então ambas recebem a pontuação neste quesito. (Tradicional: 1, ControleNF: 1)

Os resultados dessas avaliações são sumarizados nas Tabelas 1 e 2. A qualidade percebida da arquitetura ControleNF foi superior à da arquitetura tradicional quanto aos aspectos de manutenibilidade e portabilidade. A modularidade da arquitetura de microsserviços e a flexibilidade decorrente da ausência de esquema proporcionada pelo SGBD NoSQL reduzem, em teoria, o esforço necessário para a manutenção do artefato. Já a adoção de ferramentas multiplataforma permite que a portabilidade seja alcançada com maior sucesso pela ControleNF.

Tabela 1 – Resultado da avaliação qualitativa de manutenibilidade

Manutenibilidade		
	Tradicional	**ControleNF**
Modularidade	0	1
Reusabilidade	0	1
Analisabilidade	1	0
Modificabilidade	0	1
Testabilidade	1	1
Resultado:	**40%**	**80%**

Fonte: Elaborada pelo autor (2022)

Tabela 2 – Resultado da avaliação qualitativa de portabilidade

Portabilidade		
	Tradicional	**ControleNF**
Adaptabilidade	0	1
Instalabilidade	1	1
Substituição	1	1
Resultado:	**67%**	**100%**

Fonte: Elaborada pelo autor (2022)

5.4 AVALIAÇÃO QUANTITATIVA - EXPERIMENTO

Na Seção 4.1.1, foi estabelecido como requisito de *design* da arquitetura que esta deveria ter bom desempenho nas operações de inserção e consulta de notas fiscais. Esse requisito resultou no princípio de *design* nº 4, estabelecido na Seção 4.1.2, que determina que a arquitetura ControleNF deveria apresentar tempo de inserção e consultas de notas fiscais igual ou inferior a soluções existentes.

Para avaliar se o requisito foi satisfeito, foi desenvolvido um experimento com o objetivo de mensurar os tempos de resposta da arquitetura ControleNF para a execução de comandos de inserção de conjuntos de notas fiscais e de consultas sobre tais notas. Para isso, foi construído um gerador de notas fiscais sintéticas que gera notas semelhantes às originais. Mais detalhes sobre o gerador de notas fiscais construído para essa pesquisa são dados na Seção 5.4.2.

5.4.1 Configuração do Ambiente de Teste

O experimento foi conduzido em uma infraestrutura de rede local, com um laptop atuando como cliente e outro como servidor. No caso das inserções de notas fiscais, a máquina cliente representa o ambiente da SEFAZ-PE, que realiza o envio das notas fiscais para o ambiente do servidor, que processa os comandos de inserção em um SGBD. Já nas operações de consulta, a máquina cliente faz o papel das aplicações internas do TCE-PE, que enviam consultas às notas fiscais e recebem os resultados através da rede.

Como servidor, foi utilizado um computador Windows equipado com processador Intel Core i7-8565U, com 4 *cores*, 16GB de memória principal e SSD NVMe de 128GB. Já para a máquina cliente, foi utilizado um MacBook Pro (2018) com o processador Intel Core i5-8259U de 4 *cores*, 16GB de memória principal e um SSD NVMe de 500GB.

Em relação aos SGBDs adotados, na arquitetura tradicional foi utilizado o SQL Server 2019 (15.0.2000.5), versão Developer, que é equivalente à versão Enterprise, mas destinada a desenvolvimento e testes. Na arquitetura ControleNF, foi utilizado o MongoDB Community Edition 5.0.5. Os códigos desenvolvidos para a API e para a execução do experimento foram escritos em Python e executados na versão 3.9.6 da linguagem. Não foram criados índices secundários nos SGBDs, somente índices primários para as chaves primárias do esquema relacional e para o campo identificador padrão (_id) do MongoDB.

5.4.2 Geração do Banco de Dados

No planejamento inicial do experimento, havia a intenção de se utilizar notas fiscais reais emitidas para os órgãos e entidades do Estado de Pernambuco enviadas pela SEFAZ-PE ao TCE-PE como parte do convênio de cooperação firmado entre os órgãos. Todavia, ao se realizar a solicitação formal para utilização dos referidos documentos nesta pesquisa, houve resposta negativa por parte da Procuradoria Jurídica do TCE-PE, com base nos termos do convênio firmado e na Lei Geral de Proteção de Dados, o que inviabilizou a utilização destes dados.

Também foram exploradas outras hipóteses de utilização de NF-es reais, como a realização do download das notas disponibilizadas pelo Tribunal de Contas da União[1] ou no Portal da Transparência do Governo Federal[2]. No entanto, estes conjuntos de dados se mostraram pequenos e incompletos. Os dados do TCU só dizem respeito a suas próprias compras, que

[1] <https://contas.tcu.gov.br/ords/f?p=1675:1::::::>
[2] <https://www.portaltransparencia.gov.br/download-de-dados/notas-fiscais>

ocorrem em pequeno volume (menos de 100 NF-e no mês de janeiro de 2022, por exemplo). O Portal da Transparência do Governo Federal, por ter iniciado a publicação das NF-e pouco tempo antes da realização deste experimento, também não apresentava um grande volume de dados no momento do acesso. Em meados de janeiro de 2022 havia cerca de 180 mil notas, emitidas a partir de 28/10/2021. No entanto, a maior limitação dessa fonte de dados consiste no fato de que o Portal da Transparência não disponibilizava os dados completos das notas, sendo provido somente um pequeno subconjunto de campos, insuficiente para a geração de NF-es de acordo com o leiaute oficial.

Após tentativas infrutíferas de obtenção de um conjunto de dados considerado grande o suficiente para realização do experimento, isto é, algo próximo a um milhão de notas para avaliar o desempenho das consultas, procedeu-se à geração de um conjunto de documentos sintéticos, gerados com base nas regras do leiaute da Nota Fiscal Eletrônica.

Para isso, foi utilizado o conjunto de dados abertos de licitações do Governo Federal, disponível no Portal da Transparência[3]. Foram baixados dados de compras governamentais realizadas num período de 3 anos, entre os meses de julho de 2018 e junho de 2021. Esses dados compreendiam a aquisição de 5.206.710 itens[4], em 636.300 licitações. Esses dados foram então carregados em um SGBD relacional para serem transformados, quando necessário, e então servirem de base para a geração das NF-e sintéticas.

Para entender como esses dados foram processados e transformados em notas fiscais, é importante primeiro compreender as informações que devem estar contidas em uma NF-e. Como descrito na Seção 2.7, uma NF-e é um documento XML organizado em uma estrutura hierárquica, com elementos agrupados de acordo com a natureza da informação representada. Para a geração das notas fiscais sintéticas utilizadas nesse experimento, foram adotados os elementos XML obrigatórios presentes no leiaute versão 4.00 da NF-e, presente no Anexo I do Manual de Orientação ao Contribuinte versão 7.00 (ENCAT, 2020) e listados na Figura 10.

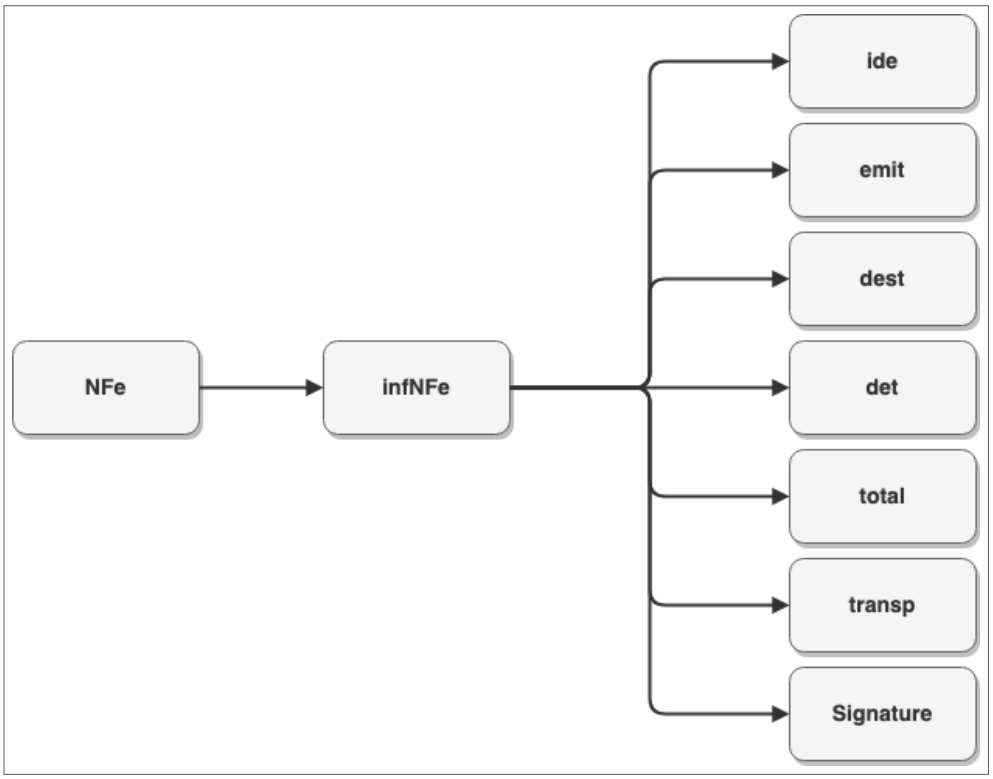
O elemento NFe é o elemento raiz de cada nota fiscal eletrônica. Esse elemento tem um subelemento chamado infNFe que contém a maior parte das informações relevantes para cada nota. Essas informações ficam divididas em vários outros elementos, que servem como grupos de informações de naturezas distintas a respeito de cada nota. Aqueles grupos utilizados no leiaute das notas fiscais geradas para o experimento são listados a seguir:

[3] <https://www.portaltransparencia.gov.br/download-de-dados/licitacoes>

[4] Considera-se como um item adquirido cada produto distinto presente em uma mesma licitação. A quantidade de itens não se confunde com o quantitativo de unidades adquiridas de cada item.

- **ide:** Identificação da NF-e
- **emit:** Emitente da NF-e
- **dest:** Destinatário da NF-e
- **det:** Detalhamento dos produtos ou serviços
- **total:** Valores totais da NF-e
- **transp:** Informações a respeito do transporte dos produtos
- **Signature:** Assinatura digital da NF-e conforme padrão XML Digital Signature

Figura 10 – Estrutura das NF-es sintéticas geradas



Fonte: Elaborada pelo autor (2022)

Com base nas informações da Figura 10, os dados das compras governamentais extraídos do Portal da Transparência foram transformados e complementados com dados de outras fontes, de modo a se reunir informações suficientes para gerar documentos fiéis à realidade. Era necessário obter, no mínimo, um conjunto de dados que trouxesse informações sobre a transação, os produtos adquiridos, o fornecedor (emitente da nota), o destinatário (o órgão

ou entidade comprador do produto ou tomador do serviço), os valores envolvidos e dados da transportadora responsável, além de uma assinatura digital.

Para esse fim, foram extraídos do Portal da Transparência dados de 5.206.710 itens adquiridos por órgãos e entidades federais em 636.300 licitações. Esse conjunto de dados já possuía boa parte das informações necessárias presentes nos grupos **ide**, **det** e **totais** das notas fiscais, sem necessidade de alterações. No entanto, cada aquisição listava apenas o número do CPF ou CNPJ e o nome do fornecedor. Por esse motivo, decidiu-se agregar o banco de dados do CNPJ, disponibilizado em formato aberto pela Receita Federal do Brasil, que contém informações como nome fantasia, endereço e telefone, que são elementos obrigatórios do grupo **emit**.

Em alguns casos, foi necessário remover aquisições de itens do conjunto de dados em que a obtenção de mais informações, como nome, endereço e telefone seria impossível. Foram removidos 5 itens que não possuíam informação sobre fornecedor, 24.845 que tinham conteúdo sigiloso (sem informações a respeito do que foi adquirido ou de quem foi o fornecedor), além de 19.283 que possuíam fornecedores estrangeiros, para os quais não seria possível obter as suas informações no banco de dados do CNPJ, e 127.072 adquiridos de pessoas físicas, cujos dados não são públicos. Após esse processo, restaram 5.035.505 itens adquiridos de 126.997 fornecedores diferentes, para os quais foi possível extrair os dados do CNPJ.

Em relação aos destinatários das NF-e (grupo **dest**), foram utilizados dados dos órgãos e entidades do Governo Federal que realizaram as licitações para aquisição dos itens. Esses órgãos e entidades, no Sistema Integrado de Administração de Serviços Gerais (SIASG), sistema que mantém o seu registro, são conhecidos como Unidades de Administração de Serviços Gerais (UASGs). No conjunto de dados das compras governamentais, a exemplo dos dados dos fornecedores, só havia o código de cada unidade no SIASG e seu nome. Foi então utilizada a API do Comprasnet, sistema de compras governamentais, para se obter os dados detalhados de cada uma das UASGs.

Esses dados, no entanto, não foram suficientes para se obter as informações obrigatórias do grupo **dest** das notas fiscais. Das 3.705 UASGs que realizaram as compras presentes no conjunto de dados considerado, 2.423 não possuíam informação de endereço. Para 2.422 delas, havia a informação do CEP, então se utilizou APIs públicas de consulta de CEPs (www.cepaberto.com e https://viacep.com.br) para se obter os campos do endereço, já que não há a disponibilização de conjunto de dados abertos do CEP por parte dos Correios. Dessas, 437 CEPs não puderam ser encontrados, então atribuiu-se o endereço da sede do governo do

respectivo Estado. Também havia 1.701 UASGs sem número do CNPJ, o que é comum, já que nem toda unidade administrativa que realiza licitações tem um CNPJ. Para esses casos, foi atribuído também o CNPJ e demais dados da sede do governo do estado em que a UASG está localizada.

Em relação aos dados de **transp**, não havia qualquer tipo de informação sobre o transporte dos produtos no banco de dados de compras governamentais, então foram extraídos do banco de dados do CNPJ os dados de 1.000 empresas do Estado de Pernambuco cujo nome continha o texto "Transportadora". Essas transportadoras foram atribuídas aleatoriamente a cada aquisição.

Por fim, para o campo **Signature**, a maior parte dos campos possuem valores fixos no leiaute da NF-e, então foi necessário apenas gerar valores codificados em base64 para dois campos. Para esses, foram utilizados os mesmos valores para todas as notas fiscais geradas, já que a assinatura digital da NF-e é um instrumento de verificação da autenticidade das notas e, como as notas obtidas pelos órgãos de controle vêm diretamente do órgão fazendário responsável pela autorização das notas, não existe a necessidade dessa verificação. Para fins de execução das consultas usadas na análise dos dados, a presença do elemento **Signature** também é irrelevante, mas como esse elemento contém longas cadeias de caracteres codificadas em base64, ele acaba compondo boa parte do tamanho do documento de cada nota, o que é relevante para as análise dos tempos de execução das inserções nas arquiteturas analisadas.

Feitas essas transformações, foi gerada uma NF-e para cada combinação única entre licitação e fornecedor. Ou seja, em vários casos, para uma mesma licitação foram geradas múltiplas notas fiscais, cada uma agrupando os itens fornecidos por uma mesma empresa. A partir desse processo, foram obtidas 931.118 NF-e no formato XML, compreendendo a aquisição de 5.035.505 itens. As notas foram geradas por um script na linguagem Python, a partir de uma mistura de valores obtidos no banco de dados de compras governamentais, valores aleatórios e valores escalares fixos, sempre respeitando as regras estabelecidas no leiaute da NF-e definido no Manual de Orientação ao Contribuinte. O conjunto completo de elementos gerados e os valores atribuídos estão disponíveis nas tabelas do Apêndice A.

5.4.3 Carga de dados

O experimento consistiu em executar as mesmas operações sobre as notas fiscais na arquitetura tradicional e na arquitetura ControleNF, mensurando-se o tempo de resposta de cada

execução com o auxílio da biblioteca *time* da linguagem Python, linguagem de programação adotada para a construção do programa de testes.

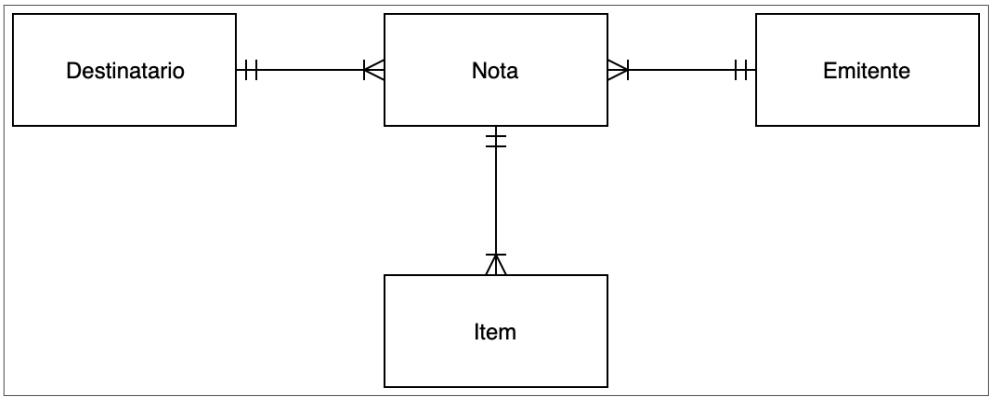
Para se avaliar também a escalabilidade, ou seja, para verificar como as arquiteturas se comportam com o aumento no volume de notas fiscais, utilizou-se subconjuntos progressivamente maiores das notas fiscais geradas, avaliando-se os tempos de execução das operações a cada iteração. Para as operações de inserção, utilizou-se conjuntos de 1.000, 10.000 e 100.000 NF-es. Para as operações de consulta, foram utilizadas as mesmas três cargas, além do conjunto total de 931.118 notas fiscais geradas na etapa de geração do banco de dados, conforme descrito na Seção 5.4.2.

Cada operação foi executada cinco vezes para cada carga de dados. Os tempos de resposta para cada uma das arquiteturas foram medidos em cada execução, e, ao final, as médias dos tempos de execução de cada operação foram extraídas. Essas médias foram comparadas entre as arquiteturas, obtendo-se uma variação percentual que demonstra o ganho ou a perda de desempenho aferido com a utilização da arquitetura ControleNF.

5.4.3.1 Inserções

Para a criação do banco de dados relacional, usado na arquitetura tradicional, foi utilizada a ferramenta de geração de scripts do Microsoft SQL Server para extrair os comandos DDL da linguagem T-SQL[5] de criação de objetos do ambiente de produção do TCE-PE. O banco de dados foi então recriado no ambiente de testes. Os elementos desse banco de dados estão representados no esquema ER da Figura 11.

Figura 11 – Esquema ER da Aplicação de NF-es Avaliada nos Experimentos



Fonte: Elaborada pelo autor (2022)

[5] Dialeto da linguagem SQL utilizado pelo Microsoft SQL Server

O processo de carga de um conjunto de notas fiscais no SGBD relacional consiste basicamente em executar um procedimento armazenado, que objetiva: realizar a chamada em linha de comando do aplicativo 7-Zip para extrair os arquivos compactados que contêm os arquivos XML das notas em um diretório do servidor, carregar os dados das notas em uma coluna do tipo XML de uma tabela auxiliar, utilizar o método *nodes* da linguagem T-SQL para extrair os dados das notas fiscais dessa tabela e por fim inseri-los nas tabelas correspondentes do esquema de dados da Figura 11, realizando-se as conversões de tipo pertinentes. As conversões de tipo são necessárias, uma vez que o XML armazena todos os dados como texto e durante a análise das notas fiscais há a necessidade de se realizar operações como somas de valores numéricos e comparações entre datas.

No caso da arquitetura ControleNF, a inserção de notas é realizada através do chamada da método HTTP POST à API, cujo corpo é composto pelo conteúdo do arquivo XML da nota em formato *string*. Para cada nota inserida é realizada uma chamada distinta do método POST, o que modifica a lógica de processar as inserções em lote. A arquitetura, então, converte a nota do formato texto para o formato dicionário (*dict*) da linguagem Python, e posteriormente realiza as conversões de tipo pertinentes. Para a realização do experimento, as conversões realizadas foram as mesmas daquelas realizadas no processamento da arquitetura tradicional. O dicionário, então, é passado como argumento ao método *insertOne*, que insere um documento em uma coleção do MongoDB.

5.4.3.2 Consultas

Em relação às consultas, buscou-se utilizar um conjunto que refletisse operações realizadas no processo de análise das notas fiscais por parte de auditores em um processo de fiscalização. Foram concebidas seis consultas, que foram escritas na linguagem SQL, para execução na arquitetura tradicional, e na linguagem de consulta do MongoDB (MQL), para execução na arquitura ControleNF. As consultas estão listadas a seguir, juntamente com a cardinalidade (quantidade de registros retornados) para a carga total de 931.118 notas fiscais. Os códigos nas linguagens de consulta estão listados no Apêndice B.

Q1: *Retornar CNPJ e nome de destinatário, chave, número e série das notas fiscais emitidas, em que o estado de emissão da NF-e é um dos estados do Nordeste.* **Cardinalidade:** 161.780

Q2: *Retornar CNPJ e nome de destinatário, chave, número e série das notas fiscais emitidas*

entre janeiro de 2019 e junho de 2020. **Cardinalidade:** 302.717

Q3: *Retornar dados de destinatário, emitente, nota e itens em que a quantidade adquirida é maior que 1.* **Cardinalidade:** 4.394.672

Q4: *Retornar dados de destinatário, emitente e nota, em que há algum item de "locação"ou "serviço".* **Cardinalidade:** 57.839

Q5: *Retornar quantidade de notas fiscais emitidas no estado de São Paulo, agrupadas por CNPJ e nome do destinatário e ordenadas de forma decrescente pela quantidade.* **Cardinalidade:** 271

Q6: *Retornar valor médio dos itens, soma dos valores e contagem de itens, para todos os itens cujo valor de comercialização é maior que 100, agrupados por município do destinatário, descrição do item e unidade do item.* **Cardinalidade:** 277.028

As consultas na linguagem SQL foram executadas na arquitetura tradicional através do pacote *PyODBC* da linguagem Python, sendo passadas no formato *string* direto para o método *cursor.execute(consulta)*, que atribui ao objeto cursor o conjunto de resultados da consulta, permitindo que se itere sobre as linhas retornadas.

No caso da arquitetura ControleNF, as consultas são executadas na forma de chamadas do método HTTP GET da API. No entanto, o microsserviço de gerenciamento de dados, ao acessar o MongoDB para executar as consultas, utiliza os métodos *find* e *aggregate*, que recebem argumentos do tipo dicionário (objeto composto por pares chave-valor) e lista (*array*). Esses argumentos, então, precisam ser serializados para que possam ser transmitidos por HTTP e posteriormente por *gRPC* na arquitetura. Para este fim, foi definida uma sintaxe para a construção das consultas, a serem passadas em formato texto nas URLs de chamadas do método GET. Essa sintaxe consiste em gerar um dicionário com três campos: *op*, *args* e *datetime* e transformá-lo em uma *string* de texto no formato JSON. Esses campos têm os seguintes significados:

- ***op:*** Método de uma coleção do MongoDB a ser executado no banco de dados. Pode ter os valores "find", "aggregate", "distinct"ou "count".
- ***args:*** Dicionário cujas chaves são os argumentos da chamada do método, no formato chave:valor.
- ***datetime:*** Lista de campos da consulta a serem transformados no formato *datetime*. Esse campo é necessário pois o *PyMongo* requer que se utilize objetos do tipo *datetime*

da linguagem Python para realizar consultas que envolvem campos do tipo data/hora do MongoDB. Como não é possível enviar objetos desse tipo através do HTTP, eles são enviados no formato string e então convertidos em datetime pelo microsserviço de gerenciamento de dados.

Para ambas as arquiteturas, o experimento consistiu em remover todos os registros o banco de dados, executar a carga das notas fiscais e executar cada uma das seis consultas em sequência. O procedimento foi realizado cinco vezes para cada uma das três cargas de dados (1.000, 10.000 e 100.000 notas). A carga máxima (931.118) só foi utilizada para mensurar os tempos de execução das consultas, já que, em testes, seu processamento para as inserções na arquitetura tradicional tomou um tempo excessivo, superior a 20h, o que inviabilizou sua execução em tempo hábil para inclusão neste teste. Assim, esse conjunto máximo de notas foi inserido apenas uma vez em cada arquitetura, e os conjuntos de consultas foram executados cinco vezes, com comandos de limpeza de cache sendo executados entre as iterações.

5.4.4 Resultados do experimento

Na arquitetura tradicional, o desempenho da inserção das notas se deteriorou conforme o aumento das cargas, saindo de uma média de cerca de 63 NF-e por segundo na carga de 1.000 notas, para 47,5 NF-e/s na carga de 10.000 notas e finalmente 20 NF-e/s na carga de 100.000 notas. Os tempos de execução detalhados são listados na Tabela 3.

Tabela 3 – Tempos de inserção de NF-es na arquitetura tradicional (em segundos)

Inserções - Arquitetura Tradicional		
1.000 notas	**10.000 notas**	**100.000 notas**
16,036114	206,876255	5042,093781
15,694185	207,785987	4964,854267
15,829663	212,048544	4957,381052
15,702978	212,975551	4990,366191
15,706954	213,377740	4974,644408

Fonte: Elaborada pelo autor (2022)

Já na arquitetura ControleNF, as taxas de inserção se mantiveram constantes, com 35,2 NF-e/s para a carga de 1.000 notas, 34,18 NF-e/s para a carga de 10.000 notas e 33.8 NF-e/s para a carga de 100.000 notas. Os tempos de execução detalhados são listados na Tabela 4.

Tabela 4 – Tempos de inserção de NF-es na arquitetura ControleNF (em segundos)

Inserções - Arquitetura ControleNF		
1.000 notas	**10.000 notas**	**100.000 notas**
29,789750	297,525586	3170,101865
28,843457	303,286637	3227,732006
28,720201	314,192317	3165,260932
28,659094	317,552157	3172,669256
28,658200	312,277079	3111,506918

Fonte: Elaborada pelo autor (2022)

Comparando-se os tempos médios de execução das operações nas duas arquiteturas, tem-se que a arquitetura tradicional obteve melhor desempenho nas cargas de 1.000 e 10.000 notas. Na carga de 1.000 notas, o tempo médio de execução da arquitetura ControleNF foi 79,7% maior do que o tempo correspondente da arquitetura tradicional e na carga de 10.000 notas, 38,9% maior. O cenário se inverteu na carga de 100.000 notas, em que a arquitetura ControleNF apresentou um tempo médio de execução 40,7% menor. Os resultados são detalhados na Tabela 5.

Tabela 5 – Análise comparativa dos tempos de inserção

Inserções			
	Tradicional	**ControleNF**	**Variação**
1.000 notas	15,793979	28,384996	79,7%
10.000 notas	210,612815	292,535150	38,9%
100.000 notas	4985,867940	2955,640626	-40,7%

Fonte: Elaborada pelo autor (2022)

Em relação às consultas, os tempos de execução detalhados da arquitetura tradicional estão presentes na Tabela 6, enquanto os resultados da arquitetura ControleNF são listados na Tabela 7. Cada consulta foi executada cinco vezes para carga de dados e seus tempos de execução mensurados e registrados.

Comparando-se os tempos de execução de ambas as arquiteturas, a arquitetura ControleNF apresentou tempos de execução em média 531% maiores que a arquitetura tradicional, considerando todos os tempos de execução aferidos. O desempenho foi inferior em todas as consultas e para todas as cargas, com a exceção da consulta Q5 na carga de 10.000 notas. A avaliação completa das médias consta na Tabela 8.

Tabela 6 – Tempos de execução das consultas na arquitetura tradicional (em segundos)

Consultas - Arquitetura Tradicional					
1.000 notas					
Q1	Q2	Q3	Q4	Q5	Q6
0,015620	0,015623	0,140625	0,046874	0,031234	0,031244
0,015622	0,015671	0,109300	0,046901	0,015620	0,046828
0,015622	0,015621	0,109348	0,046901	0,015624	0,046874
0,015621	0,015584	0,093727	0,062526	0,015620	0,046862
0,015620	0,015621	0,109350	0,046864	0,015666	0,031239
10.000 notas					
Q1	Q2	Q3	Q4	Q5	Q6
0,046902	0,046856	0,959956	0,343628	0,093763	0,234315
0,046865	0,031241	0,921694	0,374916	0,093688	0,234362
0,046863	0,031277	0,921622	0,374914	0,062521	0,234284
0,046894	0,031243	0,980057	0,356205	0,078145	0,234283
0,046821	0,031243	1,077911	0,374871	0,078149	0,234279
100.000 notas					
Q1	Q2	Q3	Q4	Q5	Q6
0,234321	0,318288	10,671042	0,734164	0,171832	0,718622
0,265561	0,333950	9,001670	0,718539	0,193310	0,765444
0,265558	0,296768	10,476825	0,724035	0,187494	0,687339
0,281222	0,328047	9,559659	0,765480	0,187463	0,656093
0,640438	0,499881	10,212376	0,702956	0,203080	0,646292
931.118 notas					
Q1	Q2	Q3	Q4	Q5	Q6
2,132411	2,999727	98,404991	7,617316	0,122506	3,743594
1,966677	2,967223	97,556174	8,103765	0,371605	4,446910
1,941753	3,036772	113,399804	6,481126	0,266520	2,848948
1,993591	2,872386	99,224107	7,872623	0,129656	3,722043
1,965282	2,970976	104,256190	5,986696	0,189982	3,330118

Fonte: Elaborada pelo autor (2022)

Tabela 7 – Tempos de execução das consultas na arquitetura ControleNF (em segundos)

Consultas - Arquitetura ControleNF					
1.000 notas					
Q1	Q2	Q3	Q4	Q5	Q6
0,064116	0,053791	0,615428	0,061763	0,025265	0,109496
0,066043	0,053253	0,624142	0,096938	0,026614	0,120199
0,046909	0,057113	0,602800	0,090485	0,022478	0,105904
0,045409	0,055238	0,602058	0,057512	0,024764	0,133016
0,047091	0,050334	0,604363	0,090973	0,032373	0,118326
10.000 notas					
Q1	Q2	Q3	Q4	Q5	Q6
0,232747	0,348913	6,015618	0,514021	0,043635	0,874462
0,196182	0,312975	5,735074	0,435646	0,048595	0,918859
0,205529	0,313950	5,761626	0,428752	0,050620	0,881787
0,232953	0,319934	5,753791	0,461015	0,040102	0,886962
0,210465	0,332355	6,554762	0,531588	0,041729	0,947583
100.000 notas					
Q1	Q2	Q3	Q4	Q5	Q6
1,928867	2,956097	73,543035	4,891003	0,246367	11,024701
1,898579	2,921091	72,896097	5,270651	0,255970	10,522960
1,942745	2,962396	60,455693	1,686855	0,159923	7,146904
1,907946	2,917218	78,173959	2,332894	0,268835	10,913674
1,913932	2,953550	60,905571	1,673189	0,159708	8,131463
931.118 notas					
Q1	Q2	Q3	Q4	Q5	Q6
26,551493	30,422868	709,508032	18,373041	2,589516	57,579540
22,710800	38,229053	823,352058	44,786576	3,557379	54,395160
24,960230	36,414021	832,265986	42,348289	2,841316	57,623520
25,004017	39,794620	832,708393	42,856108	2,073227	56,761963
26,989638	39,016467	833,759622	45,862690	3,195698	57,864238

Fonte: Elaborada pelo autor (2022)

Tabela 8 – Análise comparativa dos tempos de consulta

Médias de Tempo de Execução - Consulta			
	Tradicional	**ControleNF**	**Variação**
1.000 notas			
Q1	0,015621	0,053914	245%
Q2	0,015624	0,053946	245%
Q3	0,112470	0,609758	442%
Q4	0,050013	0,079534	59%
Q5	0,018753	0,026299	40%
Q6	0,040609	0,117388	189%
		Média da carga:	**203%**
10.000 notas			
Q1	0,046869	0,215575	360%
Q2	0,034372	0,325625	847%
Q3	0,972248	5,964174	513%
Q4	0,364907	0,474204	30%
Q5	0,081253	0,044936	-45%
Q6	0,234304	0,901931	285%
		Média da carga:	**332%**
100.000 notas			
Q1	0,337420	1,918414	469%
Q2	0,355387	2,942070	728%
Q3	9,984315	69,194871	593%
Q4	0,729035	3,170918	335%
Q5	0,188636	0,218161	16%
Q6	0,694758	9,547940	1274%
		Média da carga:	**569%**
931.118 notas			
Q1	1,999943	25,243236	1162%
Q2	2,969417	36,775406	1138%
Q3	102,568253	806,318818	686%
Q4	7,212305	38,845341	439%
Q5	0,216054	2,851427	1220%
Q6	3,618322	56,844884	1471%
		Média da carga:	**1019%**

Fonte: Elaborada pelo autor (2022)

Através dessa análise, observou-se que o desempenho da arquitetura ControleNF foi inferior à arquitetura tradicional em todos os cenários. A arquitetura também apresenta pior escala-

bilidade em relação à quantidade de notas no banco de dados, uma vez que a variação dos tempos de execução entre as duas arquiteturas cresceu conforme a carga foi aumentada. Entre a carga mínima e a máxima, a variação saltou de 203% para 1019%. Esse desempenho indica a inviabilidade da arquitetura da maneira como foi implementada, uma vez que a quantidade de notas fiscais armazenada no banco de dados do TCE-PE tende apenas a crescer ao longo do tempo, conforme mais notas são enviadas pela SEFAZ-PE.

Essa perda de desempenho pode ser explicada parcialmente pelo *overhead* introduzido pela arquitetura. Na ControleNF, é usada a biblioteca *PyMongo* para extrair os resultados do MongoDB, que os retorna no tipo dicionário (*dict*) da linguagem Python. Assim, a cada documento retornado, é necessário serializar o dicionário para um formato que possa ser transmitido através do gRPC e posteriormente por HTTP, para que o resultado possa ser retornado para o cliente da API. No experimento, os resultados foram retornados no formato JSON, codificado em UTF-8. Além disso, é importante observar que a adoção da arquitetura ControleNF implica na comunicação entre quatro componentes para a execução de consultas (cliente - *gateway* da API - serviço de gerenciamento de dados - SGBD), enquanto a arquitetura tradicional permite a conexão direta do cliente ODBC com o SGBD, sem a necessidade de se passar pela API ou por RPCs.

Para identificar o grau com que a perda de desempenho ocasionada pela arquitetura está relacionada a esses fatores ou simplesmente a um menor desempenho do SGBD MongoDB em relação ao SQL Server, testou-se também a utilização do *PyMongo* para a execução das consultas diretamente no servidor MongoDB, sem a utilização da API ou de gRPC. Os resultados desta etapa são listados na Tabela 9.

Tabela 9 – Análise de tempos de consulta - arquitetutra tradicional, ControleNF, *PyMongo*

	Tradicional	**ControleNF**	***PyMongo***
Q1	0,337420	1,918414	0,829073
Q2	0,355387	2,942070	1,437195
Q3	9,984315	69,194871	40,808615
Q4	0,729035	3,170918	1,697112
Q5	0,188636	0,218161	0,311395
Q6	0,694758	9,547940	4,536936

Fonte: Elaborada pelo autor (2022)

Observou-se que os tempos de execução das consultas diretamente ao servidor foram cerca de 34,2% menores, em média, do que aqueles obtidos com a arquitetura ControleNF.

No entanto, os tempos de execução ainda foram em média 251,6% maiores que os valores obtidos na arquitetura tradicional, o que demonstra que a utilização do MongoDB parece ter um papel significativo na redução do desempenho na arquitetura de microsserviços.

De modo a verificar a razão dessa perda de desempenho observada com o MongoDB, as consultas executadas e seus planos de execução foram examinadas. Para se realizar uma avaliação mais detalhada, foi selecionada a carga de 100.000 notas e a consulta Q2, que apresentou a maior variação média de tempo de execução entre as consultas que utilizam o método *find* no MongoDB e a segunda maior em geral. A variação dos tempos dessa consulta em relação à arquitetura tradicional foi de 740%, considerando todas as cargas.

Utilizando o método *explain* no mongosh, cliente de linha de comando do MongoDB, que exibe o plano de execução de uma consulta, observou-se que o tempo estimado para a execução de Q2 foi de 332 milissegundos (0,332 segundo), com a etapa de *COLLSCAN*, a varredura completa da coleção, ocorrendo em 12ms e a etapa de projeção ocorrendo em 35ms. Na prática, ao se executar a consulta no mongosh e atribuir o cursor a uma variável, sem se iterar sobre ele, o tempo de execução total ficou na casa de 170ms em todos os testes. Esse valor foi consideravelmente inferior à média de 1,437s obtida nos testes realizados anteriormente com o cliente *PyMongo*. Isso indica que o gargalo não reside na execução propriamente dita das consultas, mas sim na entrega dos resultados ao cliente pelo servidor do MongoDB.

Para confirmar esse resultado, foram executados novos testes para todas as consultas, com a carga de 100.000 notas, agora mensurando-se dois tempos: T_{ex} (tempo de execução), dado pelo intervalo entre o envio da requisição da consulta e o retorno do cursor de resultados, e T_{it} (tempo de iteração), intervalo entre o retorno do cursor pelo servidor e o final da interação do cliente sobre ele. Os testes foram escritos em Python, utilizando-se novamente a biblioteca *PyMongo*, e foram executados localmente na mesma máquina em que o servidor do MongoDB estava localizado, para eliminar o tempo relativo ao envio dos dados através da rede. Os resultados desse novo experimento são mostrados na Tabela 10.

Tabela 10 – Comparação dos tempos de execução e iteração nas consultas com o MongoDB

Consulta	Nº docs.	T_{ex}(s)	% do total	T_{it}(s)	% do total
Q1	17391	0,000078	0,02%	0,339911	99,98%
Q2	32332	0,000059	0,01%	0,566940	99,99%
Q3	472650	0,006562	0,06%	11,183226	99,94%
Q4	6158	0,000064	$<$0,01%	3,229695	$>$99,99%
Q5	225	0,102405	99,23%	0,000799	0,77%
Q6	60606	2,224956	91,21%	0,214477	8,79%

Fonte: Elaborada pelo autor (2022)

É possível constatar que para as consultas de Q1 a Q4, que retornam dados não agrupados, o gargalo reside na entrega de dados, que compreende a extração dos dados do disco e o envio ao cliente. Já para as consultas Q5 e Q6, que utilizam o estágio *$group* no pipeline de agregação, equivalente ao comando GROUP BY da linguagem SQL, a maior parte do tempo de execução reside no processamento da consulta, e não no consumo dos dados.

Em Q5, isso se explica pela pequena quantidade de documentos retornados pela consulta, apenas 225, o que naturalmente minimiza o tempo T_{it}. Já em relação à consulta Q6, o tempo de processamento acaba sendo a etapa dominante no tempo total de execução. A consulta Q6, cabe mencionar, é aquela em que o desempenho da arquitetura ControleNF apresentou a maior variação em relação à arquitetura tradicional. Analisando-se o plano de execução, constatou-se que as etapas mais custosas de Q6 foram os estágios de *$group* e *$sort*, de agrupamento e ordenação dos documentos, respectivamente, conforme tempos estimados de execução listados na Tabela 11.

Tabela 11 – Tempos estimados de execução dos estágios do pipeline de agregação em Q6

Etapa	Tempo Estimado (ms)
$unwind	835
$match	1327
$group	2083
$sort	2104

Fonte: Elaborada pelo autor (2022)

Por fim, é importante observar que todas as médias de tempo de consulta, mesmo quando realizadas localmente no MongoDB e sem utilização da API, ainda são superiores aos resultados obtidos durante a análise experimental na arquitetura tradicional do TCE-PE, utilizando o Microsoft SQL Server, em que o acesso foi realizado de forma remota (Tabela 12).

Tabela 12 – Médias de tempos totais de execução (s) na arquitetura tradicional e no MongoDB

Consulta	Arq. Tradicional (remoto)	MongoDB (local)
Q1	0,337420	0,339989
Q2	0,355387	0,566999
Q3	9,984315	11,189788
Q4	0,729035	3,229759
Q5	0,188636	0,103204
Q6	0,694758	2,439433

Fonte: Elaborada pelo autor (2022)

Os resultados mostram que o principal fator que ocasiona a perda de desempenho na arquitetura ControleNF foi a de consumo dos dados do servidor, e não o processamento da consulta ou o *overhead* introduzido pela arquitetura. Também observou-se a perda de desempenho do MongoDB quando a consulta envolveu agregação e ordenação dos resultados. Diante de todo o exposto, pôde-se concluir que a perda de desempenho observada na arquitetura ControleNF foi, em grande parte, ocasionada pelo próprio MongoDB. Trabalhos futuros poderão avaliar possibilidades de melhoria de desempenho com a utilização do MongoDB e se há impacto positivo na utilização de outros SGBDs orientados a documentos na arquitetura ControleNF.

5.5 CONCLUSÕES

Neste capítulo, foram realizadas duas avaliações, uma qualitativa e outra quantitativa, com o objetivo de se mensurar o grau de satisfação da arquitetura ControleNF aos requisitos estabelecidos no Capítulo 4.

Embora a avaliação qualitativa realizada indique potenciais benefícios da arquitetura ControleNF, ela é, por natureza, subjetiva, e serve apenas como análise inicial da adequação da arquitetura proposta aos requisitos de manutenibilidade e portabilidade. Os resultados obtidos devem ser confirmados em testes conduzidos em um ambiente análogo ao de produção, com o estabelecimento e aplicação de casos de uso para atividades práticas de manutenção e portabilidade.

Em relação à análise de desempenho e escalabilidade, os testes realizados apontaram um melhor desempenho da arquitetura ControleNF na inserção de notas somente na carga de 100.000 notas, e uma piora considerável no desempenho e na escalabilidade da execução de consultas, com a variação do tempo de execução superando os 1000% em alguns casos.

O experimento indica a inviabilidade da adoção da arquitetura ControleNF sem que sejam realizadas modificações para a melhoria do desempenho.

Em uma avaliação mais detalhada, observou-se que a perda de desempenho pôde ser atribuída em grande parte ao desempenho do SGBD MongoDB no contexto da infraestrutura adotada para a avaliação experimental. Em relação às consultas que não realizavam agrupamentos, o tempo de retorno dos resultados das consultas ao cliente se mostrou responsável por 99,9% do tempo total de execução. Já em relação às consultas que envolviam o agrupamento dos documentos, o tempo de execução da consulta foi responsável pela maior parte do tempo total.

No Capítulo 6, serão discutidas as conclusões obtidas neste trabalho, suas principais contribuições e direções para trabalhos futuros.

6 CONCLUSÃO

6.1 INTRODUÇÃO

Neste capítulo são apresentadas as considerações finais deste trabalho, resumindo suas principais contribuições, e são apontadas direções para trabalhos futuros.

6.2 CONSIDERAÇÕES FINAIS

Nesta dissertação, foi avaliada a arquitetura ControleNF, que se destina ao gerenciamento de notas fiscais eletrônicas compartilhadas entre as secretarias de fazenda e órgãos de controle externo. A arquitetura foi concebida através do framework Design Science Research (DSRF), que guiou a identificação adequada do problema de pesquisa e a elaboração de requisitos e princípios para o desenvolvimento de uma solução. No processo, foi identificado que arquiteturas tradicionais existentes possuem limitações, como a rigidez de esquema trazida por SGBDs relacionais, a dificuldade de manutenção, de escalabilidade e da possibilidade de se transferir a arquitetura para outros cenários semelhantes em outros órgãos. Definiu-se, então, que uma arquitetura a ser proposta deveria apresentar as características de manutenibilidade, portabilidade, desempenho e escalabilidade.

O artefato produzido no âmbito do DSRF, a arquitetura ControleNF, é uma arquitetura de microsserviços, que utiliza uma REST API para se comunicar com aplicações externas e gRPC para a comunicação interna entre os microsserviços. A arquitetura prevê a utilização de um SGBD NoSQL para armazenar as notas fiscais eletrônicas com flexibilidade de esquema. Para a descrição da arquitetura, foi utilizado o modelo C4 para visualização de software, que permitiu a apresentação de sua estrutura em diferentes níveis de detalhe.

Uma vez descrita, a arquitetura foi avaliada quanto aos critérios de manutenibilidade, portabilidade, desempenho e escalabilidade, conforme requisitos anteriormente estabelecidos. A avaliação se deu no cenário do TCE-PE e da SEFAZ-PE, órgãos do Estado de Pernambuco que possuem convênio de cooperação para o compartilhamento das notas fiscais. Para avaliação dos critérios de manutenibilidade e portabilidade, utilizou-se o modelo de qualidade de software definido no ISO/IEC 25010 (ISO/IEC, 2011), atribuindo-se uma pontuação a cada arquitetura de acordo com as características definidas no padrão. Essa avaliação resultou em uma pontuação superior para a arquitetura ControleNF tanto na manutenibilidade (80% con-

tra 40% da arquitetura tradicional) quanto na portabilidade (100% contra 67% da arquitetura tradicional).

Já em relação a desempenho e escalabilidade, foi desenvolvido um experimento que consistia na realização das operações de inserção e consulta de lotes de notas fiscais e a mensuração dos tempos de execução. Uma dificuldade encontrada na realização do experimento foi a impossibilidade de utilização das NF-es reais obtidas pelo TCE-PE, por vedação existente no instrumento do convênio. Para suprir esta necessidade, foi concebido um gerador de notas fiscais sintéticas, criadas com base no conjunto de dados de compras governamentais disponibilizado pelo Governo Federal em seu Portal da Transparência. A esse conjunto de dados, somou-se o banco de dados do Cadastro Nacional de Pessoas Jurídicas (CNPJ), disponibilizado em formato aberto pela Receita Federal, de modo a se obter as informações sobre os contribuintes emitentes das notas e gerar notas fiéis à realidade.

O experimento realizado consistiu em se executar as operações de inserção e consultas em subconjuntos progressivamente maiores das notas fiscais geradas, com cada operação sendo executada cinco vezes, e mensurar os tempos de execução. As médias dos tempos para cada operação e conjunto de dados foram então extraídas e comparadas entre a arquitetura tradicional do TCE-PE e a arquitetura ControleNF. Para as inserções, os resultados mostraram que o desempenho da arquitetura ControleNF foi inferior que o da arquitetura tradicional para as cargas menores, de 1.000 e 10.000 notas, com médias de tempos de execução 79,7% e 38,9% maiores, respectivamente. Já na carga de 100.000 notas, a arquitetura ControleNF teve o melhor desempenho, com tempo de execução 40,7% inferior. Na arquitetura ControleNF, a taxa de notas inseridas por segundo se manteve praticamente constante mesmo com o crescimento das cargas, enquanto a arquitetura tradicional apresentou uma perda de desempenho na ordem de 68% nesta taxa entre a menor carga (1.000 notas) e a maior (100.000 notas).

Em relação às consultas, o desempenho da arquitetura ControleNF foi consideravelmente inferior, com variação média de 531% nos tempos de execução e chegando a 1019% na carga completa das notas fiscais (931.118 notas). De modo a se levantar possíveis causas para a perda de desempenho, também se executou os testes executando-se as consultas diretamente ao SGBD através de um cliente criado através da biblioteca PyMongo da linguagem Python, sem se passar os comandos pela REST API. Os tempos de execução obtidos foram 30% menores que aqueles da arquitetura ControleNF, mas ainda assim 251,6% superiores à arquitetura tradicional, o que mostra que o maior gargalo da arquitetura reside na obtenção dos resultados do banco de dados, e não no *overhead* de comunicação introduzido pela API e pela comunica-

ção entre os microsserviços. Uma avaliação mais detalhada indicou a perda de desempenho do MongoDB no consumo dos dados do disco e entrega ao cliente, naquelas consultas que utilizavam o método *find*, e nas etapas de agrupamento e ordenação, em consultas que utilizavam o método *aggregate*.

Assim, concluiu-se que a arquitetura, da maneira que foi proposta e utilizando as ferramentas e tecnologias usadas no protótipo do experimento, não reúne condições de substituir a arquitetura tradicional para o gerenciamento de notas fiscais. Embora haja potenciais benefícios em relação aos requisitos de manutenibilidade e portabilidade, tal avaliação é limitada pela subjetividade da avaliação e se destinava somente a prover um ponto de partida para a validação da arquitetura. A significativa perda de desempenho na realização de consultas acaba por inviabilizar a implementação da arquitetura sem que sejam realizadas modificações que se destinem a tornar mais ágeis as consultas realizadas.

Com esta dissertação conclui-se também o processo do DSRF com a atividade de comunicação, em que há a exposição do problema e sua importância, do artefato produzido e de sua efetividade na resolução do problema.

6.3 TRABALHOS FUTUROS

Nesta seção são listadas as principais propostas de trabalhos futuros que podem ser seguidas para continuidade do projeto de pesquisa desenvolvido.

6.3.1 Benchmark para Notas Fiscais Eletrônicas

Benchmarks servem como padrões para comparação de desempenho entre diferentes sistemas. Nesta dissertação, para mensurar o desempenho das arquiteturas de gerenciamento de notas fiscais, foram criadas cargas de dados e consultas que se basearam nas atividades e características do ambiente do TCE-PE. No entanto, esses fatores podem não refletir adequadamente a realidade de outros órgãos de controle. A elaboração de um *benchmark* para avaliação de desempenho e escalabilidade em arquiteturas de gerenciamento de notas fiscais eletrônicas visa estabelecer um padrão para se avaliar a eficiência de arquiteturas de gerenciamento de notas fiscais em uma diversidade maior de cenários, contribuindo para a definição de uma arquitetura única a ser utilizada nos diferentes entes da Federação.

6.3.2 Avaliação de Escalabilidade Horizontal de NoSQL

No Capítulo 2, foi exposto que SGBDs NoSQL possuem maior escalabilidade horizontal quando comparados a SGBDs relacionais. Uma possibilidade de trabalho futuro consiste em explorar essa característica, adaptando-se a arquitetura para trabalhar com várias instâncias do SGBD e se avaliando o desempenho da arquitetura quando escala horizontalmente, adicionando novos nós ao sistema de armazenamento distribuído.

6.3.3 Mecanismo de Criptografia para Armazenamento em Nuvem

Um obstáculo para a utilização de computação na nuvem no gerenciamento das notas fiscais eletrônicas é o sigilo fiscal aplicável aos dados dos emitentes. A utilização de criptografia pode viabilizar a utilização da computação em nuvem para o gerenciamento de notas fiscais, trazendo os benefícios de economia e flexibilidade característicos desse modelo de computação. Nesse sentido, é importante avaliar as possibilidades de se criptografar os dados dentro da arquitetura para o seu armazenamento na nuvem, sem introduzir obstáculos significativos em termos de desempenho e facilidade de utilização. Em determinadas aplicações, pode-se avaliar a utilização de criptografia homomórfica, aquela em que operações sobre os dados podem ser executadas sem a necessidade de descriptografá-los.

6.3.4 Avaliação Aprofundada de Desempenho das Consultas no MongoDB

A avaliação negativa do desempenho da arquitetura ControleNF teve como uma de suas causas o desempenho inferior do MongoDB para o processamento das consultas propostas dentro da infraestrutura utilizada. Um trabalho futuro poderá analisar de forma mais detalhada o processamento interno das consultas por parte do MongoDB, identificando gargalos e investigando possíveis soluções, com o objetivo de melhorar o desempenho do SGBD no cenário avaliado.

6.3.5 Avaliação de Desempenho com Diferentes SGBDs

Também é possível a realização de uma avaliação comparativa do desempenho das operações de inserção e consulta de notas fiscais em múltiplos SGBDs orientados a documentos, com o objetivo de identificar quais apresentam melhor desempenho e, portanto, são mais adequados para utilização na arquitetura. Também podem ser explorados outros modelos de SGBD NoSQL que sejam capazes de armazenar as NF-e em seu formato nativo.

6.3.6 Avaliação Prática de Manutenibilidade e Portabilidade

A avaliação de manutenibilidade e portabilidade realizada neste trabalho, apesar de baseada em um modelo de qualidade de software estabelecido, contou com um grau de subjetividade elevado, já que não foram realizados testes estruturados ou avaliações de terceiros que visassem confirmar o diagnóstico realizado na avaliação qualitativa. Um trabalho futuro poderá explorar casos de teste para atividades comuns de manutenção e de portabilidade, de modo a comprovar a adequação da arquitetura ControleNF a essas características.

REFERÊNCIAS

ANDRADE, B. L. F. F. *Gerenciamento e Auditoria Fiscal Eletrônica através do uso de documentos XML como objetivo para tomada de decisão com Business Intelligence*. 2013. Disponível em: <www.cin.ufpe.br/~posgraduacaoRECIFE,JUNHO/2013>.

AZEVEDO, L. G.; Da Silva Ferreira, R.; Da Silva, V. T.; De Bayser, M.; De Soares, E. F.; THIAGO, R. M. Geological data access on a polyglot database using a service architecture. *ACM International Conference Proceeding Series*, p. 103–112, 2019.

BANCROFT, J. *Senlin Ascends*. [S.l.]: Orbit, 2013. ISBN 978-1482590951.

BARBOZA, E. da S. *Autenticação multifatorial em hardware para o processo de assinatura digital da NF-e*. 2018. Disponível em: <https://repositorio.ufpe.br/handle/123456789/32403>.

Brandão Filho, C. A.; ALVES, F. M.; BERTONCINI, B. V. Uso de big data para compreensão do fenômeno do transporte urbano de cargas: Uma análise de geração de entregas. In: . [S.l.: s.n.], 2018.

BRASIL. *Constituição da República Federativa do Brasil*. 1988. Disponível em: <http://www.planalto.gov.br/ccivil_03/constituicao/constituicao.htm>.

BRAUN, E.; SCHLACHTER, T.; DÜPMEIER, C.; STUCKY, K. U.; SUESS, W. A generic microservice architecture for environmental data management. *IFIP Advances in Information and Communication Technology*, Springer International Publishing, v. 507, p. 383–394, 2017. ISSN 18684238. Disponível em: <http://link.springer.com/10.1007/978-3-319-89935-0>.

BROWN, S. *The C4 model for visualising software architecture*. 2018. Disponível em: <https://www.infoq.com/articles/C4-architecture-model/>.

CATTELL, R. Scalable SQL and NoSQL data stores. *SIGMOD Record*, v. 39, n. 4, p. 12–27, 2010. ISSN 01635808.

CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM*, v. 13, n. 6, p. 377–387, jun 1970. ISSN 0001-0782. Disponível em: <https://dl.acm.org/doi/10.1145/362384.362685>.

CONFAZ. *Protocolo ICMS/42*. 2009. Disponível em: <https://www.confaz.fazenda.gov.br/legislacao/protocolos/2009/pt042_09>. Acesso em: 15 fev. 2022.

De Souza, C. R.; REDMILES, D.; CHENG, L. T.; MILLEN, D.; PATTERSON, J. Sometimes you need to see through walls - A field study of application programming interfaces. *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*, v. 6, n. 3, p. 63–71, 2004.

DEBASTIANI, J. A. *Servidor REST para emissão de Nota Fiscal Eletrônica*. 2017.

DENNY, D. M. T.; PAULO, R. F.; NEVES, F. C. Q. Alternativa tecnológica para compensação de créditos de icms: estudo de caso da viabilidade do uso de dlt na nota fiscal eletrônica. *Revista Brasileira de Políticas Públicas*, v. 11, 4 2021. ISSN 2236-1677. Disponível em: <https://www.publicacoes.uniceub.br/RBPP/article/view/6696>.

DRAGONI, N.; GIALLORENZO, S.; LAFUENTE, A. L.; MAZZARA, M.; MONTESI, F.; MUSTAFIN, R.; SAFINA, L. Microservices: Yesterday, today, and tomorrow. *Present and Ulterior Software Engineering*, p. 195–216, 2017.

ELMASRI, R.; NAVATHE, S. B. *Sistemas de Bancos de Dados*. 6ª. ed. São Paulo: Pearson Universidades, 2010. ISBN 9788579360855.

ENCAT. *Sistema Nota Fiscal Eletrônica - Manual de Orientação do Contribuinte (Versão 7.00)*. 2020. Disponível em: <https://www.nfe.fazenda.gov.br/portal/exibirArquivo.aspx?conteudo=LrBx7WT9PuA=>.

FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doutorado) — University of California, Irvine, 2000.

GARLAN, D.; SHAW, M. An Introduction to Software Architecture. In: . [s.n.], 1993. p. 1–39. Disponível em: <http://www.worldscientific.com/doi/abs/10.1142/9789812798039_0001>.

GOMES, D. D. *Plataforma colaborativa para mineração de notas fiscais eletrônicas*. 2018.

GUILHEN, B. A.; SPINOLA, M. de M.; GONçALVES, R. F.; KOFUJI, S. T. Um sistema de rastreabilidade de mercadorias utilizando blockchain, o sped fiscal e a nota fiscal eletrônica. In: . [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2021. p. 481–487. ISBN 9781665441186.

HEVNER; MARCH; PARK; RAM. Design Science in Information Systems Research. *MIS Quarterly*, v. 28, n. 1, p. 75, 2004. ISSN 02767783. Disponível em: <https://www.jstor.org/stable/10.2307/25148625>.

HEVNER, A.; CHATTERJEE, S. *Design Research in Information Systems*. Boston, MA: Springer US, 2010. v. 22. (Integrated Series in Information Systems, v. 22). ISBN 978-1-4419-5652-1. Disponível em: <http://link.springer.com/10.1007/978-1-4419-5653-8>.

IPEA. *Programas visam inserir micro e pequenas empresas nas compras governamentais*. 2018. Disponível em: <https://www.ipea.gov.br/portal/index.php?option=com_content&view=article&id=34435%3Aprogramas-visam-inserir-micro-e-pequenas-empresas-nas-compras-governamentais&catid=6%3Adinte&directory=1&Itemid=1>. Acesso em: 15 fev. 2022.

ISO/IEC. *ISO/IEC 25010*. 2011. Disponível em: <https://www.iso.org/standard/35733.html>.

JITA, H.; PIETERSE, V. A framework to apply the internet of things for medical care in a home environment. *ACM International Conference Proceeding Series*, p. 45–54, 2018.

KATHIRAVELU, P.; SAGHAR, Y. N.; AGGARWAL, T.; SHARMA, A. Data Services with Bindaas: RESTful Interfaces for Diverse Data Sources. In: *Proceedings - 2019 IEEE International Conference on Big Data, Big Data 2019*. [S.l.: s.n.], 2019. p. 457–462. ISBN 9781728108582.

KLEPPMANN, M. *Designing Data-Intensive Applications*. Sebastopol: O'Reilly, 2017. ISBN 9781449373320.

LEAL, J. P.; MAHLER, P. R. Atenção com o preço justo de mercado: Economizando nas compras públicas com o uso da nota fiscal eletrônica. In: . [S.l.: s.n.], 2016.

LEAVITT, N. Will NoSQL Databases Live Up to Their Promise? *Computer*, v. 3, n. 2-3, p. 59–72, 2010. ISSN 0018-9162.

LEWIS, J.; FOWLER, M. *Microservices - A definition of this new architectural term*. 2014. Disponível em: <https://martinfowler.com/articles/microservices.html>.

LINARES-VÁSQUEZ, M.; BAVOTA, G.; BERNAL-CÁRDENAS, C.; Di Penta, M.; OLIVETO, R.; POSHYVANYK, D. API change and fault proneness: A threat to the success of android apps. *2013 9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2013 - Proceedings*, p. 477–487, 2013.

LU, W.; XU, Q.; LAN, C. SOOCP: A platform for data and analysis of space object optical characteristic. *Information (Switzerland)*, v. 10, n. 10, 2019. ISSN 20782489.

MARAVER, P.; ARMAÑANZAS, R.; GILLETTE, T. A.; ASCOLI, G. A. PaperBot: Open-source web-based search and metadata organization of scientific literature. *BMC Bioinformatics*, BMC Bioinformatics, v. 20, n. 1, p. 1–13, 2019. ISSN 14712105.

MENDES, D.; JORGE, D.; PIRES, G.; PANDA, R.; ANTONIO, R.; DIAS, P.; OLIVEIRA, L. VITASENIOR-MT: A distributed and scalable cloud-based telehealth solution. In: *IEEE 5th World Forum on Internet of Things, WF-IoT 2019 - Conference Proceedings*. [S.l.: s.n.], 2019. p. 767–772. ISBN 9781538649800.

MongoDB Inc. *The most popular database for modern apps | MongoDB*. 2021. Disponível em: <https://www.mongodb.com/>.

NEWMAN, S. *Building Microservices*. [S.l.]: O'Reilly, 2015. ISBN 978-1-491-95035-7.

RECEITA FEDERAL DO BRASIL. *Portal da Nota Fiscal Eletrônica*. 2022. Disponível em: <https://www.nfe.fazenda.gov.br/portal/sobreNFe.aspx?tipoConteudo=PEhYdxncZBE= &AspxAutoDetectCookieSupport=1>. Acesso em: 15 jan. 2022.

SANTOS, E. M. dos. *Uso de dados de documentos fiscais eletrônicos para o planejamento do transporte urbano de cargas*. 2015. Disponível em: <http://repositorio.unb.br/handle/10482/21983>.

SANTOS NETO, A. B. d. *Extração de Informações Mercadológicas a partir de Notas Fiscais Eletrônicas*. 2018. Disponível em: <http://www.tede2.ufrpe.br:8080/tede2/handle/tede2/7867>.

SEFAZ-MT. *Sefaz e TCE firmam convênio de compartilhamento de informações - Notícias*. 2008. Disponível em: <http://www5.sefaz.mt.gov.br/-/sefaz-e-tce-firmam-convenio-de-compartilhamento-de-informacoes>. Acesso em: 15 fev. 2022.

SEFAZ-PB. *Receita Estadual libera acesso às notas fiscais eletrônicas para consulta do TCE-PB*. 2017. Disponível em: <https://www.sefaz.pb.gov.br/announcements/4351-receita-estadual-libera-acesso-as-notas-fiscais-eletronicas-para-consulta-do-tce-pb>. Acesso em: 15 fev. 2022.

SIMõES, R. de O. M.; CASTRO, S. H. R. de. Política suricato de fiscalização integrada: Inteligência e inovação. In: . [s.n.], 2017. Disponível em: <http://www.planalto.gov.br/gsi/saei/paginas/inteli>. Acesso em: 15 fev. 2022.

SOMMERVILLE, I. *Engenharia de Software*. 9ª edição. ed. São Paulo: Pearson Prentice Hall, 2011. ISBN 9788579361081.

TCE-PE. *Convênio de cooperação técnica*. 2016. Disponível em: <https://tce.pe.gov.br/internet/docs/convenios/4479/sefaz-pe-compartilhamento-de-dados-1.pdf>. Acesso em: 15 fev. 2022.

THE OPEN GROUP. *SOA Source Book: What is SOA?* 1995. Disponível em: <https://collaboration.opengroup.org/projects/soa-book/pages.php?action=show&ggid=1314>. Acesso em: 15 fev. 2022.

VIENNOT, N.; LÉCUYER, M.; BELL, J.; GEAMBASU, R.; NIEH, J. Synapse: A microservices architecture for heterogeneous-database web applications. *Proceedings of the 10th European Conference on Computer Systems, EuroSys 2015*, 2015.

VILLAÇA, L. H.; AZEVEDO, L. G.; BAIO, F. Query strategies on polyglot persistence in microservices. In: *Proceedings of the ACM Symposium on Applied Computing*. [S.l.: s.n.], 2018. p. 1725–1732. ISBN 9781450351911.

XINYANG, F.; JIANJING, S.; YING, F. REST: An alternative to RPC for web services architecture. *2009 1st International Conference on Future Information Networks, ICFIN 2009*, IEEE, p. 7–10, 2009.

YE, F.; LIU, Z.; ZHU, S.; ZHANG, P.; CHEN, Y. Research of Benchmarking and Selection for TSDB. In: WEN, S.; ZOMAYA, A.; YANG, L. T. (Ed.). *Algorithms and Architectures for Parallel Processing*. Cham: Springer International Publishing, 2020. p. 642–655. ISBN 978-3-030-38961-1.

ZIMMERMANN, O. Microservices tenets: Agile approach to service development and deployment. *Computer Science - Research and Development*, Springer Berlin Heidelberg, v. 32, n. 3-4, p. 301–310, 2017. ISSN 18652042.

APÊNDICE A – REGRAS DO GERADOR DE NF-E

Neste apêndice são listados os critérios utilizados para a geração das notas fiscais sintéticas utilizadas na avaliação experimental empreendida no trabalho. Os valores atribuídos aos campos da nota fiscal podem ser dados reais derivados do banco de dados de compras governamentais construído neste projeto, ou podem ser valores sintéticos atribuídos de forma constante ou aleatória, sempre respeitando as regras do leiaute versão 4.0 da Nota Fiscal Eletrônica.

As tabelas a seguir detalham os campos que são preenchidos pelo gerador de NF-e. Elas contêm o nome e a descrição do campo, o campo pai na hierarquia do documento XML e o critério utilizado para o preenchimento de seu valor.

Tabela 13 – Regras de geração do grupo **ide**

Campo	Descrição	Pai	Valor Adotado
cUF	UF do emitente da nota	ide	UF do fornecedor da compra governamental
cNF	Código numérico da chave da NF-e	ide	Código aleatório de 8 dígitos numéricos
natOp	Natureza da operação	ide	"Venda de mercadoria"
mod	Código do modelo do documento fiscal	ide	"55"
serie	Série do documento fiscal	ide	Valor aleatório entre 0 e 969
nNF	Número do documento fiscal	ide	Valor aleatório entre 1 e 999999999
dhEmi	Data/hora de emissão	ide	Data da licitação
tpNF	Tipo de operação	ide	"1"
idDest	Identificador do local de destino da operação	ide	"1"
cMunFG	Código do município da ocorrência do fato gerador	ide	Código do município do fornecedor, atribuído pela Receita Federal, completado com zeros à esquerda (7 dígitos)
tpImp	Formato de imperessão do DANFE	ide	Valor aleatório entre 0 e 5
tpEmis	Tipo de emissão da NF-e	ide	Valor aleatório entre 1 e 9
cDV	Dígito verificador da chave da NF-e	ide	Valor aleatório entre 1 e 9
tpAmb	Identificação do ambiente	ide	"1"
finNFe	Finalidade da emissão da NF-e	ide	"1"
indFinal	Indicador de operação com consumidor final	ide	"1"
indPres	Indicador da presença do comprador no estabelecimento comercial	ide	"1"
procEmi	Processo de emissão da NF-e	ide	Valor aleatório entre 0 e 3
verproc	Versão do aplicativo emissor da NF-e	ide	Valor aleatório composto por 3 dígitos numéricos, alternados por pontos, no formato X.X.X

Fonte: Elaborada pelo autor (2022)

Tabela 14 – Regras de geração do grupo **emit**

Campo	Descrição	Pai	Valor Adotado
CNPJ	CNPJ do emitente	emit	CNPJ do fornecedor do item
xNome	Razão social do emitente	emit	Nome empresarial (BD do CNPJ)
xFant	Nome fantasia do emitente	emit	Nome fantasia (BD do CNPJ)
enderEmit	Grupo de elementos relativos ao endereço do emitente	emit	
xLgr	Logradouro	enderEmit	Logradouro (BD do CNPJ)
nro	Número	enderEmit	Número (BD do CNPJ)
xCpl	Complemento	enderEmit	Complemento (BD do CNPJ)
xBairro	Bairro	enderEmit	Bairro (BD do CNPJ)
cMun	Código do município	enderEmit	Código do município (BD do CNPJ)
xMun	Nome do município	enderEmit	Nome do município (BD do CNPJ)
UF	Sigla da UF	enderEmit	Sigla da UF (BD do CNPJ)
CEP	CEP	enderEmit	CEP (BD do CNPJ)
cPais	Código do país	enderEmit	Código do país (BD do CNPJ)
xPais	Nome do país	enderEmit	Nome do país (BD do CNPJ)
fone	Telefone	enderEmit	Telefone (BD do CNPJ)
IE	Inscrição estadual do emitente	emit	Código aleatório de 10 dígitos numéricos
CRT	Código de regime tributário	emit	Valor aleatório entre 1 e 3

Fonte: Elaborada pelo autor (2022)

Tabela 15 – Regras de geração do grupo **dest**

Campo	Descrição	Pai	Valor Adotado
xNome	Razão social do destinatário	dest	Nome empresarial (BD do CNPJ)
enderDest	Grupo de elementos relativos ao endereço do destinatário	dest	
xLgr	Logradouro	enderDest	Logradouro (BD do CNPJ)
nro	Número	enderDest	Número (BD do CNPJ)
xCpl	Complemento	enderDest	Complemento (BD do CNPJ)
xBairro	Bairro	enderDest	Bairro (BD do CNPJ)
cMun	Código do município	enderDest	Código do município (BD do CNPJ)
xMun	Nome do município	enderDest	Nome do município (BD do CNPJ)
UF	Sigla da UF	enderDest	Sigla da UF (BD do CNPJ)
CEP	CEP	enderDest	CEP (BD do CNPJ)
cPais	Código do país	enderDest	Código do país (BD do CNPJ)
xPais	Nome do país	enderDest	Nome do país (BD do CNPJ)
fone	Telefone	enderDest	Telefone (BD do CNPJ)
indIEDest	Indicador da IE do Destinatário	dest	"9"(não contribuinte do ICMS)

Fonte: Elaborada pelo autor (2022)

Tabela 16 – Regras de geração do grupo **det** (Parte I)

Campo	Descrição	Pai	Valor Adotado
nItem	(Atributo) Sequencial do item dentro da nota	det	Número sequencial de 1 a 990
prod	Grupo de informações dos produtos e serviços	det	
cProd	Código do produto ou serviço	prod	"CFOP"+ código CFOP
cEAN	GTIN do produto	prod	Código aleatório de 12 dígitos
xProd	Descrição do produto uo serviço	prod	Nome do item adquirido na licitação
NCM	Código NCM	prod	Código aleatório de 8 dígitos
CFOP	Código Fiscal de Operações e Prestações	prod	Código aleatório de 4 dígitos
uCom	Unidade comercial	prod	"UN"
qCom	Quantidade comercial	prod	Quantidade adquirida do item
vUnCom	Valor unitário de comercialização	prod	Valor unitário do item
vProd	Valor total bruto dos produtos ou serviços	prod	Valor total do item (quantidade * valor unitário)
cEANTrib	GTIN da unidade tributável	prod	Mesmo valor de cEAN
uTrib	Unidade tributável	prod	"UM"
qTrib	Quantidade tributável	prod	Meso valor de qCom
vUnTrib	Valor unitário de tributação	prod	Mesmo valor de vUnCom
indTot	Indica se o valor do item (vProd) entra no valor total da NF-e	prod	"1"

Fonte: Elaborada pelo autor (2022)

Tabela 17 – Regras de geração do grupo **det** (Parte II)

Campo	Descrição	Pai	Valor Adotado
imposto	Grupo de informações dos impostos incidentes sobre o item	det	
ICMS	Grupo de informações do ICMS	imposto	
ICMS40	Grupo de tributação do ICMS código 40 (isento)	ICMS	
orig	Origem da mercadoria	ICMS40	Valor aleatório de 0 a 8
CST	Código da situação tributária	ICMS40	"40"(Isento)
motDesICMS	Motivo da desoneração do ICMS	ICMS40	"8"(Motivo - venda a órgão público)
IPI	Grupo de informações do IPI	imposto	OBS: Chance aleatória de 50% de existir na NF-e
cEnq	Código de enquadramento	IPI	"999"(tributação normal)
IPITrib	Grupo de tributação do IPI	IPI	
CST	Código da situação tributária	IPITrib	"99"(outras saídas)
vBC	Valor da base de cálculo	IPITrib	Valor total do item (vProd)
pIPI	Percentual do tributo	IPITrib	8.0
vIPI	Valor do tributo	IPITrib	vBC * pIPI
PIS	Grupo de informações do ICMS	imposto	
PisAliq	Grupo de tributação do PIS por alíquota	PIS	
CST	Código da situação tributária	PisAliq	"01"(operação tributável)
vBC	Valor da base de cálculo	PisAliq	Valor total do item (vProd)
pPIS	Percentual do tributo	PisAliq	1.65
vPIS	Valor do tributo	PisAliq	vBC * pPIS
COFINS	Grupo de informações do COFINS	imposto	
COFINSAliq	Grupo de tributação do COFINS por alíquota	COFINS	
CST	Código da situação tributária	COFINSAliq	"01"(operação tributável)
vBC	Valor da base de cálculo	COFINSAliq	Valor total do item (vProd)
pCOFINS	Percentual do tributo	COFINSAliq	7.6
vCOFINS	Valor do tributo	COFINSAliq	vBC * pCOFINS

Fonte: Elaborada pelo autor (2022)

Tabela 18 – Regras de geração do grupo **total**

Campo	Descrição	Pai	Valor Adotado
ICMSTot	Grupo totais referente ao ICMS	total	
vBC	Base de cálculo do ICMS	ICMSTot	0.00
vICMS	Valor total do ICMS	ICMSTot	0.00
vICMSDeson	Valor total do ICMS desonerado	ICMSTot	0.00
vBCST	Base de cálculo do ICMS ST	ICMSTot	0.00
vST	Valor total do ICMS ST	ICMSTot	0.00
vFCPST	Valor total do fundo de combate à pobreza	ICMSTot	0.00
vFCPSTRet	Valor total do fundo de combate à pobreza retido anteriormente	ICMSTot	0.00
vProd	Valor total dos produtos e serviços	ICMSTot	Soma dos valores dos itens adquiridos
vFrete	Valor total do frete	ICMSTot	0.00
vSeg	Valor total do seguro	ICMSTot	0.00
vDesc	Valor total do desconto	ICMSTot	0.00
vII	Valor total do imposto de importação	ICMSTot	0.00
vIPI	Valor total do IPI	ICMSTot	Soma dos valores do tributo para cada item
vPIS	Valor total do PIS	ICMSTot	Soma dos valores do tributo para cada item
vCOFINS	Valor total da COFINS	ICMSTot	Soma dos valores do tributo para cada item
vOutro	Outras despesas acessórias	ICMSTot	0.00
vNF	Valor total da NF-e	ICMSTot	Soma dos valores dos itens adquiridos
vTotTrib	Valor aproximado total de tributos	ICMSTot	Soma dos valores de IPI, PIS e COFINS

Fonte: Elaborada pelo autor (2022)

Tabela 19 – Regras de geração do grupo **transp**

Campo	Descrição	Pai	Valor Adotado
modFrete	Modalidade do frete	transp	"0"(por conta do emitente)
transporta	Grupo de informações da empresa transportadora	transp	OBS: Selecionados os dados de uma transportadora aleatória do banco de dados do CNPJ
CNPJ	CNPJ	transporta	CNPJ (BD do CNPJ)
xNome	Razão social	transporta	Razão social (BD do CNPJ)
IE	Inscrição estadual	transporta	Inscrição estadual (BD do CNPJ)
xEnder	Endereço (logradouro + número + complemento)	transporta	Endereço (logradouro + número + complemento) (BD do CNPJ)
xMun	Nome do município	transporta	Nome do município (BD do CNPJ)
UF	Sigla da UF	transporta	Sigla da UF (BD do CNPJ)

Fonte: Elaborada pelo autor (2022)

Tabela 20 – Regras de geração do grupo **Signature**

Campo	Descrição	Pai	Valor Adotado
SignedInfo	Grupo de informações da assinatura digital	Signature	
xmlns	(Atributo)	SignedInfo	http://www.w3.org/2000/09/xmldsig#
CanonicalizationMethod	Método de canonicalização	SignedInfo	
Algorithm	(Atributo) Algoritmo	CanonicalizationMethod	http://www.w3.org/TR/2001/REC-xml-c14n-20010315
SignatureMethod	Método de assinatura	SignedInfo	
Algorithm	(Atributo) Algoritmo	SignatureMethod	http://www.w3.org/2000/09/xmldsig#rsa-sha1
Reference	Referência ao objeto	SignedInfo	
URI	(Atributo) Identificador único universal do recurso	Reference	# + chave da Nfe
Transform #1	Elemento multivalorado com algoritmos de transformação	Reference	
Algorithm	(Atributo) Algoritmo	Transforms #1	http://www.w3.org/2000/09/xmldsig#enveloped-signature
Transform #2	Elemento multivalorado com algoritmos de transformação	Reference	
Algorithm	(Atributo) Algoritmo	Transforms #2	http://www.w3.org/TR/2001/REC-xml-c14n-20010315
DigestMethod	Método de digest	Reference	
Algorithm	(Atributo) Algoritmo	DigestMethod	http://www.w3.org/2000/09/xmldsig#sha1
DigestValue	Valor de digest	Reference	vFL68WETQ+mvj1aJAMDx+oVi928=
SignatureValue	Assinatura digital codificada em base64	Signature	Primeiro parágrafo do artigo da Wikipedia sobre NF-e, codificado em base64
KeyInfo	Grupo de informações sobre o certificado	Signature	
X509Data	Informações sobre o certificado no padrão X.509	KeyInfo	
X509Certificate	Certificado codificado em base64	X509Data	Certificado de exemplo no artigo da Wikipedia do X.509, codificado em base64

Fonte: Elaborada pelo autor (2022)

APÊNDICE B – CONSULTAS DO EXPERIMENTO

Q1: *Retornar CNPJ e nome de destinatário, chave, número e série das notas fiscais emitidas, em que o estado de emissão da NF-e é um dos estados do Nordeste.* **Cardinalidade:** 161.780

SQL:

```
SELECT d.CNPJ, d.Razao_Social, n.id, n.numero, n.serie, n.DataEmissao
FROM Nota n JOIN Destinatario d ON n.destinatario = d.Codigo
WHERE d.UF IN ('PE', 'AL', 'BA', 'CE', 'PB', 'PI', 'RN', 'SE', 'MA');
```

MQL:

```
db.nfe.find(
  {
    "NFe.infNFe.dest.enderDest.UF": {
      $in: ["PE", "AL", "BA", "CE", "PB", "PI", "RN", "SE", "MA"],
    },
  },
  {
    "NFe.infNFe.dest.CNPJ": 1,
    "NFe.infNFe.dest.xNome": 1,
    "NFe.infNFe.ide.nNF": 1,
    "NFe.infNFe.ide.serie": 1,
    dhEmi: {
      $dateToString: {
        format: "%Y%m%dT%H%M",
        date: "$NFe.infNFe.ide.dhEmi",
      },
    },
  }
)
```

Q2: *Retornar CNPJ e nome de destinatário, chave, número e série das notas fiscais emitidas entre janeiro de 2019 e junho de 2020.* **Cardinalidade:** 302.717

```
SQL:
SELECT d.CNPJ, d.Razao_Social, n.id, n.numero, n.serie, n.DataEmissao
FROM Nota n JOIN Destinatario d ON n.destinatario = d.Codigo
WHERE n.DataEmissao BETWEEN '20190101' AND '20200630';

MQL:
db.nfe.find(
  {
    "NFe.infNFe.ide.dhEmi": {
      $gte: ISODate('2019-01-01T00:00:00'),
      $lte: ISODate('2020-06-30T00:00:00')
    },
  },
  {
    "NFe.infNFe.dest.CNPJ": 1,
    "NFe.infNFe.dest.xNome": 1,
    "NFe.infNFe.ide.nNF": 1,
    "NFe.infNFe.ide.serie": 1,
    dhEmi: {
     $dateToString: { format: "%Y%m%dT%H%M", date: "$NFe.infNFe.ide.dhEmi" },
    },
  }
)
```

Q3: *Retornar dados de destinatário, emitente, nota e itens em que a quantidade adquirida é maior que 1.* **Cardinalidade:** 4.394.672

```
SQL:
SELECT d.CNPJ, d.Razao_Social, e.CNPJ, e.Razao_Social, e.Endereco, e.Numero,
e.Complemento, e.Bairro, e.Municipio, e.UF, n.id, n.numero,
```

```
n.serie, n.DataEmissao, i.descricao, i.unidade, i.quantidade,
i.vUnCom, i.vProd
FROM Nota n
JOIN Itens i ON i.nota = n.codigo
JOIN Emitente e ON n.emitente = e.Codigo
JOIN Destinatario d ON n.destinatario = d.Codigo
WHERE i.quantidade > 1;

MQL:
db.nfe.aggregate([
    {"$unwind":"$NFe.infNFe.det"},
    {"$match":{"NFe.infNFe.det.prod.qCom":{"$gt":1}}},
    {"$project":{"NFe.infNFe.dest.CNPJ":1,
                 "NFe.infNFe.dest.xNome":1,
                 "NFe.infNFe.emit.CNPJ":1,
                 "NFe.infNFe.emit.xNome":1,
                 "NFe.infNFe.emit.enderEmit.xLgr":1,
                 "NFe.infNFe.emit.enderEmit.nro":1,
                 "NFe.infNFe.emit.enderEmit.xCpl":1,
                 "NFe.infNFe.emit.enderEmit.xBairro":1,
                 "NFe.infNFe.emit.enderEmit.xMun":1,
                 "NFe.infNFe.emit.enderEmit.UF":1,
                 "NFe.infNFe.ide.nNF":1,
                 "NFe.infNFe.ide.serie":1,
                 "dhEmi":{ "$dateToString":{"format":"%Y%m%dT%H%M",
                 "date":"$NFe.infNFe.ide.dhEmi"}},
                 "NFe.infNFe.det.prod.xProd":1,
                 "NFe.infNFe.det.prod.uCom":1,
                 "NFe.infNFe.det.prod.qCom":1,
                 "NFe.infNFe.det.prod.vUnCom":1,
                 "NFe.infNFe.det.prod.vProd":1
                 }}
    ]
```

```
)
```

Q4: *Retornar dados de destinatário, emitente e nota, em que há algum item de "locação"ou "serviço".* **Cardinalidade:** 57.839

```
SQL: SELECT d.CNPJ, d.Razao_Social, e.CNPJ, e.Razao_Social, e.Endereco,
e.Numero, e.Complemento, e.Municipio, e.UF, n.id, n.numero, n.serie,
n.DataEmissao
FROM Nota n
JOIN Emitente e ON n.emitente = e.Codigo
JOIN Destinatario d ON n.destinatario = d.Codigo
WHERE EXISTS (SELECT i.Codigo
FROM itens i
WHERE (i.descricao like '%loca__o%' OR i.descricao like '%servi_o%')
AND i.nota = n.codigo
);

MQL: db.nfe.find(
  {
    $or: [
    { "NFe.infNFe.det.prod.xProd": { $regex: ".*loca..o.*", $options: "i" } },
    { "NFe.infNFe.det.prod.xProd": { $regex: ".*servi.o.*", $options: "i" } },
    ],
  },
  {
    "NFe.infNFe.dest.CNPJ": 1,
    "NFe.infNFe.dest.xNome": 1,
    "NFe.infNFe.emit.CNPJ": 1,
    "NFe.infNFe.emit.xNome": 1,
    "NFe.infNFe.emit.enderEmit.xLgr": 1,
    "NFe.infNFe.emit.enderEmit.nro": 1,
    "NFe.infNFe.emit.enderEmit.xCpl": 1,
    "NFe.infNFe.emit.enderEmit.xBairro": 1,
    "NFe.infNFe.emit.enderEmit.xMun": 1,
```

```
    "NFe.infNFe.emit.enderEmit.UF": 1,
    "NFe.infNFe.ide.nNF": 1,
    "NFe.infNFe.ide.serie": 1,
    dhEmi: {
     $dateToString: { format: "%Y%m%dT%H%M", date: "$NFe.infNFe.ide.dhEmi" },
    },
  }
)
```

Q5: *Retornar quantidade de notas fiscais emitidas no estado de São Paulo, agrupadas por CNPJ e nome do destinatário e ordenadas de forma decrescente pela quantidade.* **Cardinalidade:** 271

```
SQL:
SELECT d.CNPJ, d.Razao_Social, COUNT(0) AS Qtd
FROM Nota n JOIN Destinatario d ON n.destinatario = d.Codigo
WHERE d.uf = 'SP'
GROUP BY d.CNPJ, d.Razao_Social
ORDER BY qtd DESC;

MQL:
db.nfe.aggregate([
  { $match: { "NFe.infNFe.dest.enderDest.UF": "SP" } },
  {
    $group: {
      _id: {
        CNPJ: "$NFe.infNFe.dest.CNPJ",
        Nome: "$NFe.infNFe.dest.xNome",
      },
      qtd: { $count: {} },
    },
  },
  {$sort: { qtd: -1 }},
]);
```

Q6: *Retornar valor médio dos itens, soma dos valores e contagem de itens, para todos os itens cujo valor de comercialização é maior que 100, agrupados por município do destinatário, descrição do item e unidade do item.* **Cardinalidade:** 277.028

SQL:

```
SELECT d.Municipio, i.descricao, i.unidade,
AVG(i.vUnCom) AS ValorMedio,
SUM(i.vProd) AS ValorTotalItens,
COUNT(i.Codigo) AS QtdItens
FROM Nota n JOIN Itens i ON i.nota = n.codigo
JOIN Emitente e ON n.emitente = e.Codigo
JOIN Destinatario d ON n.destinatario = d.Codigo
WHERE i.vUnCom > 100
GROUP BY d.Municipio, i.descricao, i.unidade
ORDER BY d.Municipio;
```

MQL:

```
db.nfe.aggregate(
  [
    { $unwind: "$NFe.infNFe.det" },
    { $match: { "NFe.infNFe.det.prod.vUnCom": { $gt: 100 } } },
    {
      $group: {
        _id: {
          Municipio: "$NFe.infNFe.dest.enderDest.xMun",
          Produto: "$NFe.infNFe.det.prod.xProd",
          Unidade: "$NFe.infNFe.det.prod.uCom",
        },
        ValorMedio: { $avg: "$vUnCom" },
        ValorTotalItens: { $sum: "$vProd" },
        qtd: { $count: {} },
      },
    },
```

```
    {
      $sort: { Municipio: 1 },
    },
  ],
  { allowDiskUse: true, collation: { strength: 1, locale: "pt" } }
)
```