INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ
CAMPUS TERESINA CENTRAL
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ARTUR OLIVEIRA CARVALHO

ANÁLISE E APLICAÇÃO DE TÉCNICAS DE RESILIÊNCIA DE *SOFTWARE* À BIBLIOTECA DF-E

TERESINA, PIAUÍ
2023

ARTUR OLIVEIRA CARVALHO

ANÁLISE E APLICAÇÃO DE TÉCNICAS DE RESILIÊNCIA DE *SOFTWARE* À BIBLIOTECA DF-E

Trabalho de Conclusão de Curso (Relatório técnico) apresentado como exigência parcial para obtenção do diploma do Curso de Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Teresina central.

Orientador: Prof. M^{e}. Ely da Silva Miranda

TERESINA, PIAUÍ
2023

ARTUR OLIVEIRA CARVALHO

ANÁLISE E APLICAÇÃO DE TÉCNICAS DE RESILIÊNCIA DE *SOFTWARE* À BIBLIOTECA DF-E

Trabalho de Conclusão de Curso (Relatório técnico) apresentado como exigência parcial para obtenção do diploma do Curso de Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Teresina central.

Aprovado pela banca examinadora em 13 de Dezembro de 2023.
BANCA EXAMINADORA:

Prof. Me. Ely da Silva Miranda
Instituto Federal de Educação, Ciência e Tecnologia do Piauí - IFPI

Prof. Me. Rogerio da Silva
Instituto Federal de Educação, Ciência e Tecnologia do Piauí - IFPI

Prof. Me. Fernando Castelo Branco Goncalves Santana
Instituto Federal de Educação, Ciência e Tecnologia do Piauí - IFPI

Prof. Me. Helcio de Abreu Soares
Instituto Federal de Educação, Ciência e Tecnologia do Piauí - IFPI

Dedico este trabalho a todos os que me ajudaram ao longo desta caminhada.

AGRADECIMENTOS

Agradeço à minha mãe por cuidar de mim e por seu constante incentivo. Também dedico uma lembrança especial ao meu pai, que, embora não esteja mais conosco, tenho a certeza de que estaria orgulhoso do meu progresso e das minhas conquistas até o momento.

Aos meus amigos Gustavo, Bruno, Mikael, José Carlos, Paulo e todos os outros que não mencionei, quero expressar minha sincera gratidão. Suas amizades têm sido um grande apoio em minha vida, e sou profundamente grato por ter amigos tão incríveis ao meu lado. Obrigado por confiarem em mim, acreditarem no meu potencial e por sempre desejarem o melhor para mim. Vocês são parte fundamental da minha jornada, e valorizo cada momento compartilhado com todos vocês.

Agradeço à todos os meus professores, pois cada um deles desempenhou um papel fundamental na minha jornada acadêmica. Gostaria de destacar alguns deles que tiveram um impacto notável na minha formação:

Rogério Silva, que me introduziu ao fascinante mundo da computação por meio de suas aulas de algoritmos. Suas instruções foram o ponto de partida para minha paixão por programação.

Ely Miranda, que aprofundou ainda mais meus conhecimentos, ensinando-me sobre Programação Orientada a Objetos e Programação para Internet. Fiquei fascinado por web scraping durante esse período.

Thiago Elias, por compartilhar seu conhecimento e habilidades em consultas SQL, permitindo-me dominar a arte de criar as melhores consultas.

Hélcio Abreu, que guiou meu aprendizado em Engenharia de *Software* durante quatro longos períodos. Suas lições foram fundamentais para minha compreensão das práticas de desenvolvimento de *software*.

Gostaria de estender meus agradecimentos aos meus chefes, Higo e Rafael, que me proporcionaram uma oportunidade valiosa em um momento crucial da minha carreira. Agradeço por acreditarem em mim e pelo apoio constante.

Aos meus amigos da RevGás, quero expressar minha profunda gratidão. Vocês são parte fundamental da minha jornada, obrigado por estarem ao meu lado, por compartilharem risadas e por tornarem cada dia de trabalho mais alegre e divertido.

Por último, mas não menos importante, quero prestar uma homenagem ao meu querido gato Sigma. Embora ele não esteja mais ao meu lado, as lembranças dos momentos felizes que compartilhamos sempre aquecem meu coração. Sigma foi um companheiro leal e uma fonte constante de alegria em minha vida. Obrigado, Sigma, por ter sido um amigo fiel e por ter enriquecido minha vida de maneiras inesquecíveis.

A primeira regra do Clube da Luta é:
você não fala sobre o Clube da Luta.
Tyler Durden

RESUMO

A resiliência de *software* é a capacidade de um sistema se manter operacional mesmo diante de falhas ou adversidades, desempenhando um papel fundamental na garantia da disponibilidade e confiabilidade de aplicações críticas. Em aplicativos fiscais, essa característica assume um papel de extrema importância, pois assegura a continuidade ininterrupta das operações, preservando a disponibilidade e integridade das informações fiscais, ao mesmo tempo em que cumpre prazos legais, mantém a confiança dos contribuintes e protege dados sensíveis. A análise e aplicação de padrões de resiliência à biblioteca DF-e visaram melhorar sua estabilidade e eficácia. A resiliência foi abordada em várias áreas, como a disponibilidade do *software*, armazenamento de XML e comunicação com serviços externos. Os resultados destacam ganhos significativos, como a implementação do *autoscaling*, que elevou a disponibilidade do Serviço Emissor de 90% para 100%, otimizando também os custos de infraestrutura. No armazenamento de XML, a adoção de estratégias resilientes reduziu drasticamente as intervenções manuais e melhorou a satisfação do cliente. A introdução do padrão *Circuit Breaker* mitigou falhas nos autorizadores fiscais, estabilizando o uso de recursos. Estratégias avançadas de retentativas na comunicação com serviços externos resultaram em uma recuperação eficaz em falhas temporárias. Em resumo, a implementação desses padrões aprimorou a resiliência da biblioteca DF-e, garantindo maior confiabilidade, eficiência e satisfação do usuário.

Palavras-chaves: resiliência de *software*; *circuit breaker*; *autoscaling*.

ABSTRACT

Software resilience is the ability of a system to remain operational in the face of failures or adversities, playing a crucial role in ensuring the availability and reliability of critical applications. In fiscal applications, this characteristic takes on extreme importance, as it ensures the uninterrupted continuity of operations, preserving the availability and integrity of fiscal information while meeting legal deadlines, maintaining taxpayer trust, and protecting sensitive data. The analysis and application of resilience patterns to the DF-e library aimed to improve its stability and effectiveness. Resilience was addressed in various areas, including software availability, XML storage, and communication with external services. The results highlight significant gains, such as the implementation of autoscaling, which increased the availability of the Emission Service from 90% to 100%, also optimizing infrastructure costs. In XML storage, the adoption of resilient strategies drastically reduced manual interventions and improved customer satisfaction. The introduction of the Circuit Breaker pattern mitigated failures in fiscal authorizers, stabilizing resource usage. Advanced retry strategies in communication with external services resulted in effective recovery from temporary failures. In summary, the implementation of these patterns enhanced the resilience of the DF-e library, ensuring greater reliability, efficiency, and user satisfaction.

Keywords: software resilience; circuit breaker; autoscaling.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AWS	*Amazon Web Services*
CNPJ	Cadastro Nacional de Pessoa Jurídica
CSC	Código de Segurança do Contribuinte
MOC	Manual de Orientação ao Contribuinte
MTBF	Tempo médio entre falhas
MTTR	Tempo médio de recuperação
NF-e	Nota Fiscal Eletrônica
QoS	*Quality of Service* (Qualidade de serviço)
SDK	*Software Development Kit*
SOAP	*Simple Object Access Protocol*
SVCAN	Sefaz Virtual de Contingência do Ambiente Nacional
SVCSP	Sefaz Virtual de Contingência de São Paulo
SVRS	Sefaz Virtual do Rio Grande do Sul
Uptime	Tempo de disponibilidade
UF	Unidade Federativa
WSDL	*Web Services Description Language*

SUMÁRIO

1 INTRODUÇÃO

No cenário atual, o uso de *software* é indispensável para a humanidade. Várias áreas, como o setor financeiro, entretenimento e a indústria, dependem extensivamente de *softwares* em suas operações (SOMMERVILLE, 2011). A disponibilidade e confiabilidade se tornaram requisitos essenciais em todas as formas de *software* (BURNS, 2018). Portanto, a resiliência dos sistemas é crucial para garantir que permaneçam robustos e acessíveis, mesmo diante de desafios (NYGARD, 2007).

A criação de *softwares* resilientes exige um *design* cuidadoso desde o início, isso envolve a identificação de pontos de falha potenciais, a criação de estratégias de redundância e a definição de mecanismos de recuperação. Em sistemas complexos, os componentes muitas vezes dependem uns dos outros, a resiliência requer a gestão dessas interdependências para garantir que a falha em um componente não cause uma cascata de falhas em todo o sistema (NYGARD, 2007). Além disso, a resiliência deve ser adaptável a diferentes cenários e tipos de falha, o que pode ser desafiador devido à imprevisibilidade de eventos adversos.

No entanto, a criação de sistemas distribuídos e resilientes é intrinsecamente complexa e muitas vezes não se beneficia de uma solução genérica. Em vez disso, requer uma abordagem personalizada e única, o que pode aumentar significativamente a complexidade do desenvolvimento em comparação com sistemas de arquitetura mais simples (BURNS, 2018). Felizmente, com o avanço da engenharia de *software*, surgiram técnicas de resiliência que simplificam o desenvolvimento de aplicações resilientes, tornando o processo mais eficiente e consistente.

Dada a crescente demanda por alta disponibilidade em aplicações de uso simples, torna-se ainda mais crucial garantir a resiliência em aplicações complexas e críticas. Um exemplo notável é o *software* fiscal, onde erros são inadmissíveis, pois podem resultar em implicações legais severas, incluindo sanções e multas. O uso de *softwares* fiscais é essencial devido ao aumento da regularização fiscal brasileira (COSTA, 2023), e também à complexidade das operações realizadas, desde a comunicação com os servidores fiscais até o cálculo de impostos. Diante disso, a resiliência aplicada a esses sistemas deve garantir que todas as operações sejam executadas com total precisão, segurança e eficiência.

O Manual de Orientação ao Contribuinte (MOC) é a base de desenvolvimento de um *software* fiscal definindo especificações de comunicação com os servidores fiscais brasileiros. Além disso, ele fornece orientações sobre padrões e sugestões abrangentes para o uso da arquitetura como um todo. Vale ressaltar que o MOC inclui uma seção dedicada à utilização da contingência, um servidor alternativo projetado para entrar em operação em caso de falha do servidor principal. Essa abordagem

assegura que as operações continuem sem interrupções, mesmo diante de possíveis falhas no sistema principal.

Dessa forma, existem bibliotecas que abstraem as operações fiscais, facilitando a utilização dos mesmos, como é o caso do DF-e, desenvolvido pelo autor deste trabalho. Essa biblioteca é amplamente utilizada nas operações fiscais do sistema Emissor DF-e, pertencente à empresa Tartigrado Tecnologia LTDA, que realiza diariamente um volume massivo de operações fiscais para clientes em todo o Brasil, com exceção do estado do Acre.

Embora o DF-e seja uma solução estável, ele carece de métodos específicos de resiliência para *software* fiscal. Além disso, são escassos os trabalhos relacionados a esse tema na literatura. Portanto, este trabalho tem como objetivo demonstrar os possíveis problemas relacionados à resiliência em *software* fiscal, bem como apresentar as melhores práticas para lidar com falhas. O trabalho também oferece métricas de confiabilidade, como o Tempo médio de recuperação (MTTR) e o Tempo médio entre falhas (MTBF), para validar as informações fornecidas.

Estas métricas foram adquiridas por meio da análise dos *logs* dos servidores que empregam o DF-e, possibilitando a extração do tempo médio de resposta, do histórico de falhas e erros antes e depois da implementação dos padrões de resiliência. Além disso, a ferramenta de atendimento ao cliente *Zendesk* foi empregada para obter informações relacionadas aos atendimentos ao usuário final.

1.1 OBJETIVOS

A meta principal deste trabalho é identificar os principais pontos críticos de falha e aplicar técnicas de resiliência à *softwares* fiscais, em específico a biblioteca DF-e. É importante destacar que os conceitos e métodos discutidos neste trabalho podem ser aplicados a uma ampla gama de componentes e sistemas de *software*, não limitadas à aplicações fiscais.

1.1.1 Geral

O objetivo principal deste trabalho é analisar as técnicas de resiliência de *software* e aplicá-los à biblioteca fiscal DF-e, buscando aprimorar significativamente a qualidade e a confiabilidade da aplicação.

1.1.2 Específicos

- Identificar os principais pontos críticos de vulnerabilidade e falhas existentes que podem afetar a resiliência do sistema, levando em consideração aspectos como o armazenamento de dados, o uso do servidor e a confiabilidade das operações críticas;

- Estudar as técnicas de resiliência que melhor se adequam ao contexto da biblioteca DF-e, adaptando-as às necessidades específicas do cenário fiscal, com foco na mitigação dos cenários de risco identificados;
- Extrair métricas de confiabilidade da resiliência de *software* que comprovem a eficácia da aplicação dos padrões implementados, além de apresentar gráficos comparativos para assegurar que a resiliência não comprometa significativamente o desempenho da aplicação.

1.2 ORGANIZAÇÃO

Este trabalho está organizado em três capítulos da seguinte maneira. Na Capítulo 2, são explorados fundamentos teóricos relacionados ao conceito de resiliência de software, destacando padrões como *Circuit Breaker*, *Retry Pattern* e *Auto Scaling*. Além disso, é apresentada uma explicação sobre a biblioteca DF-e, incluindo detalhes sobre sua arquitetura e componentes.

No Capítulo 3, a estrutura é dividida em duas seções. A primeira aborda questões relacionadas ao DF-e, incluindo a alta disponibilidade do serviço, o armazenamento de XML e a indisponibilidade de autorizadores, explorando como os padrões de resiliência podem abordar esses desafios. A segunda seção apresenta métricas e resultados provenientes da implementação desses padrões de resiliência. Por fim, na Capítulo 4, são apresentadas as considerações finais sobre este trabalho, oferecendo um resumo por meio da recapitulação dos conteúdos abordados e a reapresentação dos resultados obtidos.

2 FUNDAMENTAÇÃO TEÓRICA

Nessa seção serão abordados aspectos fundamentais relacionados à resiliência de *software*, bem como a arquitetura da biblioteca DF-e e seu uso de modo geral.

2.1 RESILIÊNCIA DE *SOFTWARE*

A resiliência de *software* desempenha um papel fundamental no cenário atual de desenvolvimento de sistemas de informação. Ela diz respeito à capacidade de um sistema de *software* se adaptar e manter seu desempenho, confiabilidade e disponibilidade, mesmo diante de adversidades e eventos inesperados (CUSICK, 2020). Esse conceito desafia a visão tradicional de que um sistema de *software* deve ser apenas confiável e estável. Em vez disso, envolve a incorporação de mecanismos de autodescoberta, autorrecuperação e autoajuste nos sistemas. Os sistemas resilientes são projetados para identificar proativamente ameaças potenciais, se adaptar a mudanças no ambiente e continuar operando de maneira eficaz, em vez de lidar apenas com falhas quando elas ocorrem (FIRESMITH, 2019b).

Um aspecto crucial da resiliência de *software* é a consideração de cenários adversos que vão além de falhas técnicas óbvias, como erros de programação. Isso inclui ameaças cibernéticas, desastres naturais, sobrecarga inesperada de tráfego e até mesmo erros humanos. Um sistema resiliente deve ser capaz de lidar com todos esses desafios sem comprometer sua funcionalidade. Além disso, a resiliência de *software* está intimamente ligada à experiência do usuário. Sistemas capazes de se adaptar e continuar operando de maneira confiável oferecem aos usuários uma experiência mais consistente e satisfatória, mesmo em condições adversas. No contexto da qualidade de *software*, a resiliência pode ser entendida em relação a diversos aspectos (FIRESMITH, 2019a):

- **Adaptabilidade:** Refere-se à capacidade de um sistema de se ajustar a mudanças no ambiente, tais como o uso de balanceadores de carga, reconfiguração e reestruturação, permitindo a recuperação eficaz de eventos adversos;
- **Disponibilidade:** Indica a capacidade de uma aplicação de se recuperar prontamente de eventos adversos. Um sistema resiliente deve manter a disponibilidade de seus serviços críticos mesmo sob condições inesperadas;
- **Manutenibilidade:** Um sistema é considerado mais resiliente quando consegue realizar autorreparos, evitando a necessidade de intervenção humana. Além disso, é considerado um pouco mais resiliente se suporta manutenções corretivas;

- **Desempenho:** Altos níveis de resiliência em termos de detecção de falhas podem resultar em uma diminuição do desempenho em relação à taxa de transferência e tempo de resposta, o que pode ser inaceitável em aplicações críticas;
- **Confiabilidade:** A baixa confiabilidade leva a múltiplas falhas e erros na aplicação, tornando necessária uma maior resiliência, especialmente em relação à tolerância a falhas. Assim, um sistema resiliente deve manter sua confiabilidade mesmo diante de falhas;
- **Capacidade de Reparo:** Um sistema capaz de se recuperar automaticamente, como a substituição automática de subsistemas com falhas, tem a capacidade de se recuperar mais eficazmente após um evento adverso.

A importância da resiliência de *software* nunca foi tão evidente quanto nos dias de hoje, com a crescente dependência de sistemas digitais em praticamente todos os aspectos da vida moderna (BURNS, 2018). Seja em sistemas fiscais, financeiros, de saúde, de transporte ou até mesmo em casas inteligentes, a confiabilidade desses sistemas é crítica. O reconhecimento dessa importância levou ao desenvolvimento de conceitos e estratégias dedicados à resiliência de *software*.

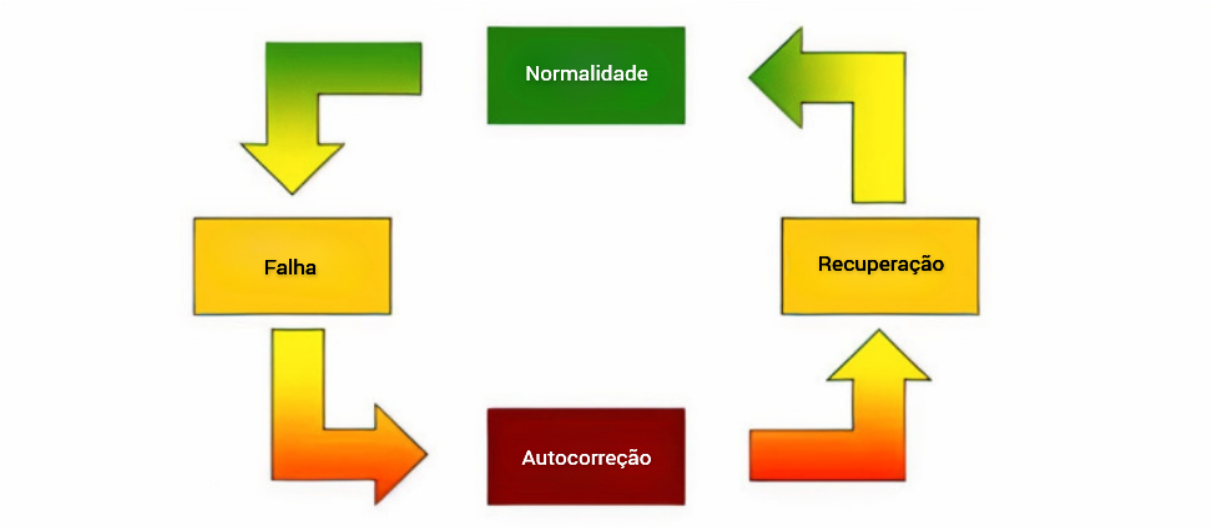
Para uma implementação bem-sucedida dos mecanismos de resiliência em um *software*, é crucial realizar uma análise abrangente de seu funcionamento. Em muitos casos, a combinação de vários mecanismos se mostra a abordagem mais eficaz para garantir a resiliência do *software*. Alguns dos principais mecanismos de resiliência de *software* incluem (FIRESMITH, 2020):

- **Alertas e Alarmes:** Os alertas desempenham um papel fundamental na notificação dos administradores de *software* sobre condições inesperadas e falhas na aplicação, permitindo que eles tomem as medidas necessárias;
- **Assertividade:** Uma aplicação pode verificar a validade de certas condições por meio de assertivas, com o propósito de validar erros na entrada, processamento e/ou saída de dados;
- ***Auto Scaling*:** O aumento dos recursos de *software*/*hardware*, muitas vezes relacionado à quantidade de servidores disponíveis, visa evitar erros de sobrecarga durante períodos de alta demanda na aplicação;
- **Cache de Conteúdo:** Quando o conteúdo permanece inalterado, estratégias de cache ajudam a evitar o uso desnecessário de recursos para recuperar os dados;
- ***Retry/Backoff*:** A retentativa (Retry) de operações, com intervalos de espera (Backoff), previne a congestão da rede e aumenta a taxa de sucesso em operações que falharam devido a erros de conexão ou problemas similares;

- ***Circuit Breaker Pattern*:** Este padrão visa evitar a propagação de falhas em cascata, interrompendo temporariamente uma parte do sistema quando uma falha ocorre, permitindo sua recuperação antes de retomar as operações;
- **Containerização:** O uso de contêineres ajuda a limitar a propagação de falhas, uma vez que cada contêiner pode ser isolado, além de facilitar a recuperação, pois são facilmente escaláveis;
- **Balanceamento de Carga:** O balanceamento de carga distribui as operações de maneira equitativa entre os sistemas para evitar sobrecarga;
- **Tratamento de Exceções:** O tratamento de exceções fornece informações corretas ao usuário final sobre falhas e, em alguns casos, orienta sobre como resolver o erro;
- ***Failover*:** Em caso de falha do *software* principal, um *software* redundante assume o controle das operações, reduzindo o tempo de inatividade do sistema;
- ***Health Check*:** As verificações de integridade atuam como um mecanismo para informar o status de um recurso, gerando alertas em caso de erros inesperados;
- ***Immutable Server*:** Em caso de falhas, é recomendável reconstruir o servidor a partir de uma versão testada e confiável, em vez de tentar aplicar correções para resolver o problema;
- ***Timeout*:** Quando as solicitações de serviço demoram para serem concluídas, de modo que as *threads* ultrapassem o limite de conexões disponíveis, os *timeouts* evitam esse bloqueio, permitindo o uso de novas conexões.

A Figura 1 ilustra um modelo de resiliência (SERULLE, 2010) que funciona de forma cíclica, passando diferentes estados.

Figura 1 – Modelo de resiliência. Tradução própria



Fonte: (SERULLE, 2010).

O ciclo é simples: um sistema em seu estado normal pode experimentar uma falha de qualquer natureza. Após a detecção da falha, o sistema passa por uma etapa de autocorreção, corrigindo a falha por conta própria, e retorna ao estado de recuperação, finalmente, voltando à normalidade. O ponto crucial é que o sistema sofre degradação e se recupera, mas não falha (CUSICK, 2020).

2.1.1 Circuit Breaker

O *Circuit Breaker Pattern* é um dos padrões de resiliência de *software* amplamente adotados para gerenciar falhas e aprimorar a estabilidade de sistemas distribuídos. Este padrão foi originalmente introduzido por Michael Nygard em seu livro "*Release It*!" como uma metáfora para a proteção de sistemas de *software* contra falhas catastróficas, assim como disjuntores elétricos protegem circuitos elétricos.

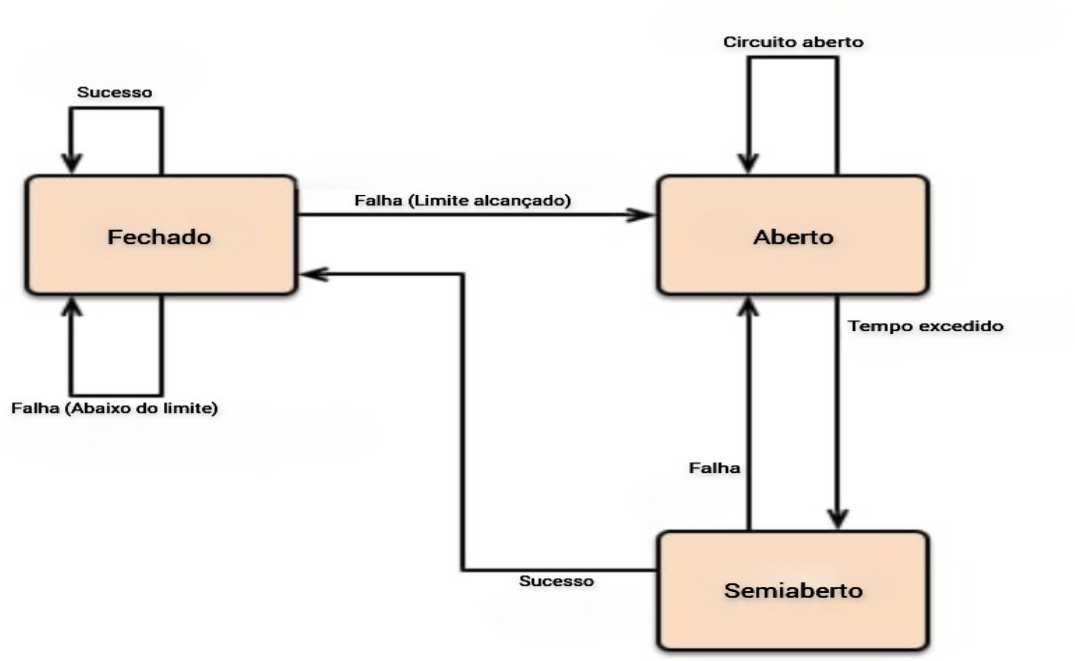
Quando se utiliza serviços ou componentes externos, é essencial que a aplicação esteja pronta para lidar com possíveis indisponibilidades. Se os componentes não estão preparados para enfrentar essa situação, a falha de um deles pode afetar todos os outros, o que é conhecido como uma falha em cascata (MONTESI; WEBER, 2016).

O *Circuit Breaker Pattern* é projetado para evitar a propagação de falhas em cascata dentro de um sistema distribuído. Funciona como um interruptor que pode abrir ou fechar, dependendo das condições do sistema. Quando o *Circuit Breaker* está fechado, as solicitações são permitidas a passar normalmente. No entanto, quando o sistema detecta uma quantidade excessiva de falhas ou tempo de resposta lento, o *Circuit Breaker* é aberto, bloqueando temporariamente as solicitações para evitar que a

situação piore. Além do estado de aberto e fechado, existe o estado “semiaberto” que tem como objetivo a própria recuperação do circuito, onde algumas solicitações são permitidas, e conforme eventuais sucessos e/ou falhas, o circuito pode ser fechado e aberto, conforme especificado por Martin Fowler (FOWLER, 2014).

A Figura 2 oferece uma representação visual da implementação do padrão Circuit Breaker e ilustra como ocorre a variação de estados do circuito.

Figura 2 – Circuit Breaker. Tradução própria



Fonte: (FOWLER, 2014).

No estado fechado do circuito, as solicitações bem-sucedidas não têm impacto sobre o estado do do mesmo, enquanto as solicitações que resultam em falhas são registradas. Quando o número de falhas atinge um limite predefinido, o circuito entra no estado aberto. Durante esse período, todas as solicitações subsequentes são bloqueadas, impedindo que elas sejam executadas enquanto o circuito permanece neste estado. Posteriormente, quando o tempo de espera do circuito aberto for excedido, o circuito transita para o estado semiaberto, onde solicitações bem-sucedidas têm o efeito de fechar o circuito novamente, enquanto solicitações que falham irão abrir o circuito. Esse processo de transição entre os estados “aberto” e “semiaberto” permite o controle dinâmico da disponibilidade de um componente ou serviço, protegendo-o de falhas em cascata.

O *Circuit Breaker* é uma ferramenta fundamental não apenas para garantir a estabilidade e resiliência de sistemas de *software* diante de falhas em serviços externos, mas também para proteger componentes internos que desempenham um papel crítico no funcionamento do sistema. Sua versatilidade o torna uma escolha valiosa em

diversos contextos, contribuindo para a manutenção da disponibilidade e confiabilidade de sistemas distribuídos e aplicativos em geral.

2.1.2 Retry Pattern

O *Retry Pattern*, em tradução livre “Padrão de Retentativa”, é um padrão de resiliência de *software* que visa aprimorar a capacidade de um sistema em lidar com falhas transitórias em operações. Esse padrão permite que o sistema reexecute automaticamente uma operação que falhou temporariamente por um número predefinido de vezes, em intervalos específicos, na esperança de que a falha seja apenas passageira e que a operação seja eventualmente bem-sucedida.

O *Retry Pattern* é fundamentado em princípios-chave que incluem tentativas controladas, onde a aplicação realiza retentativas controladas após falhas, estabelecendo um limite máximo de tentativas para evitar ciclos infinitos de retentativas sem sucesso e a implementação de intervalos exponenciais entre as tentativas subsequentes. Isso significa que as primeiras retentativas ocorrem em intervalos curtos, enquanto os intervalos aumentam progressivamente, permitindo uma abordagem mais eficaz na recuperação de operações que inicialmente falharam (EKUAN, 2023).

Erros transitórios são comuns em um ambiente de nuvem e a aplicação deve tratá-los de forma simples e transparente, onde este tratamento minimiza os efeitos que estas falhas podem ocasionar ao sistema. Caso a aplicação detecte uma falha ao realizar uma solicitação à um serviço, ela pode tentar uma das seguintes soluções:

- **Cancelar:** A aplicação deve interromper a operação caso seja identificado que esta não aparenta ser uma falha transitória, ou não ser passível de retentativa;
- **Retentar:** A aplicação pode solicitar imediatamente caso a falha for incomum, assumindo que a mesma irá ser bem sucedida;
- **Esperar e retentar:** A aplicação deve esperar antes de realizar uma nova solicitação caso a falha for ocasionada por um problema de rede, ou um sistema de *throttling* (limitação de solicitações).

Uma aplicação deve ser resistente a falhas transitórias. Falhas, como erros de rede, *locks* de banco de dados e a indisponibilidade temporária de um serviço, podem ser tratadas com tentativas subsequentes, o que aumenta a probabilidade de chamadas bem-sucedidas (MENDONCA *et al.*, 2020). A Figura 3 oferece uma representação simplificada do funcionamento do *retry pattern*, ilustrando o conceito fundamental de esperar e tentar novamente.

Figura 3 – Retry Pattern. Tradução própria

Fonte: (EKUAN, 2023).

- **Passo 1**: A aplicação realiza uma requisição para o serviço, no entanto, essa solicitação enfrenta um problema e o servidor do serviço responde com o código de resposta HTTP 500, indicando um erro interno no servidor;
- **Passo 2**: Em resposta à falha anterior, a aplicação espera um curto período de tempo e faz uma nova tentativa para realizar a mesma operação. Entretanto, esta segunda tentativa também não é bem-sucedida e resulta em um código de resposta HTTP 500;
- **Passo 3**: Após a segunda falha, o aplicativo decide esperar um intervalo mais longo antes de tentar novamente. Quando realiza a terceira tentativa, a operação é finalmente bem-sucedida, e o servidor responde com um código de resposta HTTP 200, indicando que a solicitação foi completada com sucesso, demonstrando desta forma que falhas eventuais podem ser contornadas utilizando este padrão.

O Retry Pattern é particularmente valioso em ambientes distribuídos, nos quais a conectividade de rede e a disponibilidade de serviços podem ser menos previsíveis. Em tais cenários, a aplicação precisa ser resiliente a falhas transitórias para garantir a continuidade das operações. É importante ajustar a política de retentativa de acordo com os requisitos da aplicação e a natureza das falhas. Uma política de retentativa agressiva pode afetar negativamente o desempenho ou a confiabilidade da aplicação, especialmente se ela estiver constantemente tentando operações que têm alta probabilidade de falha.

2.1.3 Auto Scaling

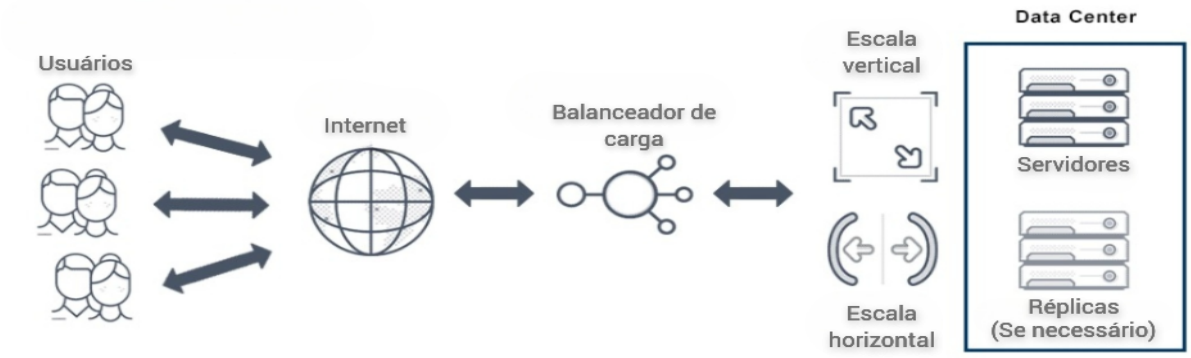
Auto Scaling, em tradução livre dimensionamento automático, é uma prática essencial em arquiteturas de computação em nuvem (KHALEQ; RA, 2021). Ele se refere à capacidade de ajustar dinamicamente a capacidade de recursos de uma aplicação

ou serviço em nuvem para corresponder à demanda variável, de forma automática e elástica. Isso é feito com base em métricas específicas, como a carga de trabalho, utilização de recursos (CPU, memória, rede), métricas de aplicação ou qualquer outro fator relevante. Sua importância na resiliência de sistemas em nuvem é definida pelos seguintes pontos-chave:

- **Disponibilidade e Confiabilidade**: O *Auto Scaling* permite que os sistemas em nuvem mantenham alta disponibilidade e confiabilidade. Quando a demanda aumenta, o sistema pode escalar automaticamente para acomodar o aumento de carga, evitando interrupções nos serviços e garantindo que os usuários possam acessar os recursos quando necessário;
- **Economia de recursos**: O *Auto Scaling* também é crucial para economizar recursos. Quando a demanda diminui, o sistema pode escalar para baixo, liberando recursos não utilizados. Isso economiza custos de infraestrutura, o que é particularmente importante em ambientes de nuvem onde os recursos são pagos com base no uso;
- **Atendimento a Requisitos de QoS**: Sistemas críticos têm requisitos estritos de *Quality of Service* (Qos), como o tempo de resposta. O *Auto Scaling* desempenha um papel crítico ao garantir que esses requisitos sejam atendidos. Quando a carga aumenta, o sistema precisa escalar para garantir que o tempo de resposta seja mantido dentro dos limites aceitáveis;
- **Adequação às Mudanças de Carga**: Os sistemas em nuvem estão sujeitos a flutuações na carga de trabalho, que podem ser sazonais, diárias ou mesmo imprevisíveis. O *Auto Scaling* permite que os sistemas se adaptem rapidamente a essas mudanças, sem intervenção manual. Isso é vital para manter a resiliência do sistema em face de variações na demanda.

Existem duas abordagens principais para o Auto Scaling: vertical e horizontal. O Auto Scaling vertical, também conhecido como dimensionamento vertical, envolve adicionar ou remover recursos computacionais dentro da mesma instância (ou máquina virtual) para atender às necessidades de carga variável. O Auto Scaling horizontal, também conhecido como dimensionamento horizontal, envolve adicionar ou remover instâncias (ou máquinas virtuais) em resposta à demanda, mantendo a mesma capacidade de recursos em cada instância. A Figura 4 mostra de maneira simplificada o funcionamento do Auto Scaling.

Figura 4 – Auto Scaling



Fonte: Produzido pelo autor.

Os usuários acessam um sistema pela internet e fazem solicitações que são direcionadas por um balanceador de carga para servidores. Nesse cenário, o balanceador de carga é responsável por escalonar os servidores de duas maneiras: horizontalmente, em resposta a uma alta demanda de operações, e verticalmente, quando operações específicas exigem mais recursos. Por outro lado, em situações de menor tráfego, o número de servidores é reduzido, desempenhando um papel fundamental na resiliência de sistemas em nuvem, garantindo que eles possam se adaptar dinamicamente às mudanças na demanda, ao mesmo tempo em que atendem aos requisitos de qualidade de serviço (KHALEQ; RA, 2021).

2.2 DF-E

A biblioteca DF-e é um *software* fiscal desenvolvido em Java pelo autor deste trabalho, sua principal função é a emissão de documentos fiscais eletrônicos brasileiros, seguindo os padrões do MOC, que definem os detalhes de implementação dos serviços fiscais oferecidos por diversos autorizadores no Brasil.

O DF-e foi projetado para ser uma ferramenta versátil e útil, oferecendo diversas interfaces que abstraem os serviços fiscais e os processos relacionados, como a assinatura de documentos usando certificados digitais modelo A1, validação de XMLs conforme os padrões definidos pelo governo federal e serviços de armazenamento de XMLs, entre outros. Embora forneça implementações padrão para essas funcionalidades, ele também permite personalizações por meio de extensões, adaptando-se às necessidades específicas de cada usuário. O uso da biblioteca do DF-e é baseada em componentes-chave, que desempenham papéis fundamentais em seu funcionamento.

2.2.1 Configuração

O componente de configuração tem como principal finalidade armazenar os dados do usuário final na memória e é empregado de forma específica por cada módulo.

Os métodos definidos são essenciais para direcionar o fluxo da aplicação como um todo. Além disso, esta classe apresenta extensões específicas, que são utilizadas em diferentes contextos, como NF-e (*NfeConfig*), NFC-e (*NfceConfig*), CT-e (*CteConfig*) e MDF-e (*MdfeConfig*). A Tabela 1 fornece uma lista dos métodos disponíveis juntamente com suas respectivas descrições e *interfaces*.

Tabela 1 – Descrição dos Métodos da Interface *Config*

Método	**Descrição**	**Interface**
uf()	Unidade Federativa (UF) onde está sendo realizada a operação	*Config*
cnpj()	Cadastro Nacional de Pessoa Jurídica (CNPJ) do usuário	*Config*
environment()	Ambiente fiscal (Produção ou Homologação) a ser utilizado na operação atual	*Config*
info()	Informações do certificado digital do usuário	*Config*
webServiceUF()	UF do serviço fiscal de destino	*Config*
emission()	Tipo de Emissão	Todas
send()	Tipo de Envio da NF-e/NFC-e	*NfeConfig*, *NfceConfig*
csc()	Código de Segurança do Contribuinte (CSC)	*NfceConfig*
cscId()	Identificador do CSC	*NfceConfig*

Fonte: Produzido pelo autor.

O Tipo de Emissão varia conforme a *interface*, já que cada documento fiscal possui valores distintos, impossibilitando a presença desse método na *interface* principal. Além disso, sua utilização na aplicação difere entre NF-e e NFC-e, onde é empregado na criação da chave de acesso, o identificador exclusivo para um documento fiscal, enquanto no CT-e é utilizado para gerar o *QR Code*.

O CSC é um código alfanumérico exclusivo atribuído a cada contribuinte e é empregado na geração do QR Code da NFC-e. O Identificador do CSC é um valor incremental que aumenta a cada criação de um novo CSC pelo contribuinte.

Na especificação dos demais componentes do DF-e, serão destacados os usos específicos dos métodos implementados nesse componente. Isso ajudará a elucidar como esses métodos desempenham um papel fundamental nas funcionalidades de cada componente, proporcionando uma compreensão mais clara de como são empregados em contextos específicos.

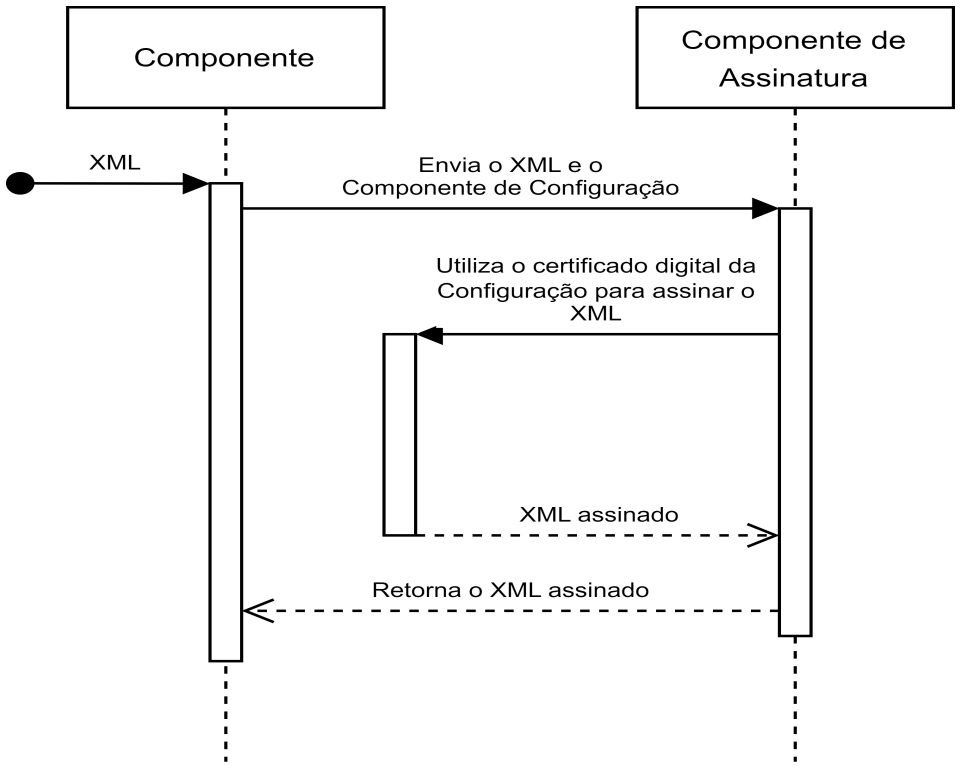
2.2.2 Assinatura Digital

A assinatura digital é uma tecnologia utilizada para garantir a integridade, autenticidade e não repúdio de um documento XML. Ela é baseada em criptografia assimétrica e envolve a geração de um par de chaves: uma privada, que é mantida em sigilo pelo signatário, e uma pública, que pode ser compartilhada com terceiros.

O MOC especifica que as mensagens enviadas para os autorizadores que são referentes às transmissões de documentos fiscais e/ou eventos devem ser assinadas por um certificado digital emitido pela Autoridade Certificadora credenciada pela Infraestrutura de Chaves Públicas Brasileira – ICP-Brasil, tipo A1 ou A3.

O Componente de Assinatura é encarregado de assinar um arquivo XML por meio do componente de configuração, que armazena informações relativas ao certificado digital do contribuinte. A implementação atual do DF-e utiliza exclusivamente certificados A1, pois, devido ao caráter físico dos certificados A3, seriam necessárias arquiteturas adicionais para viabilizar a assinatura digital. No entanto, a flexibilidade e extensibilidade do DF-e desempenham um papel fundamental na possibilidade de implementação de outros componentes de assinatura personalizados. A Figura 5 apresenta um diagrama de sequência que ilustra, de forma simplificada, o fluxo de funcionamento do componente de assinatura implementado no DF-e.

Figura 5 – Componente de Assinatura.



Fonte: Produzido pelo autor.

O primeiro objeto, referido como “Componente”, deve enviar o XML e a configu-

ração para o Componente de Assinatura. Posteriormente, o Componente de Assinatura processará o XML, assinando-o por meio do Componente de Configuração, que armazena os dados do certificado digital. O resultado retornado será o XML assinado. É relevante ressaltar que esse fluxo é interno e foi rigorosamente testado, excluindo chamadas a componentes externos para prevenir falhas imprevistas.

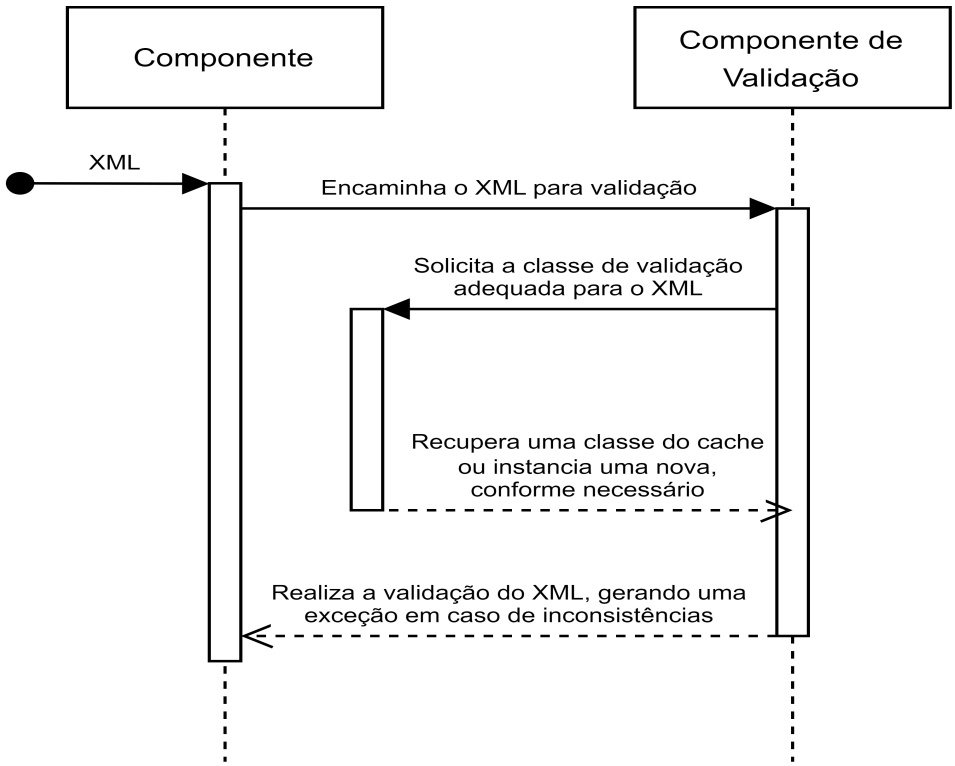
2.2.3 Validação

O XSD (XML Schema Definition) é uma linguagem de definição de esquema XML usada para descrever a estrutura e as restrições de um documento XML. Um arquivo XSD define os elementos, atributos e tipos de dados usados em um documento XML e fornece um conjunto de regras que validam a conformidade do documento XML com a estrutura definida.

As mensagens transmitidas para os servidores fiscais devem ser validadas através de um XSD específico fornecido pelas mesmas, de modo com que mensagens com erros; como, por exemplo, de formatação ou caracteres inválidos, não sejam enviadas.

O componente de validação realiza a verificação da integridade do XML por meio do XSD, antes de enviar requisições para os autorizadores. A criação de instâncias das classes que validam o XML via XSD pode ser custosa em termos de uso de CPU, devido à necessidade de interpretar todas as especificações do arquivo. No entanto, essas classes são *threadsafe*, ou seja, podem ser usadas por vários processos simultaneamente. Portanto, o componente de validação emprega a estratégia de *cache* para evitar a alocação desnecessária de recursos. A Figura 6 apresenta um diagrama de sequência que ilustra, de forma simplificada, o fluxo de funcionamento do componente de validação implementado no DF-e.

Figura 6 – Componente de Validação.



Fonte: Produzido pelo autor.

O primeiro objeto, chamado “Componente”, envia o XML para ser validado pelo Componente de Validação. O Componente de Validação procura a classe de validação correspondente ao XML fornecido. Essa busca é realizada de forma sincronizada no próprio componente. Se a classe correspondente já existe, o componente simplesmente a reutiliza, uma vez que, como mencionado anteriormente, é uma classe *threadsafe* que pode ser compartilhada entre processos. Caso a classe não exista, o componente a instanciará com o XSD específico para aquele XML e a armazenará, permitindo que validações posteriores aproveitem essa instância. Uma vez obtida a classe de validação, o XML é validado. Se o XML não for válido, uma exceção será lançada, fornecendo detalhes que indicam em qual linha do arquivo ocorreu o erro de validação.

2.2.4 Busca do Serviço Fiscal

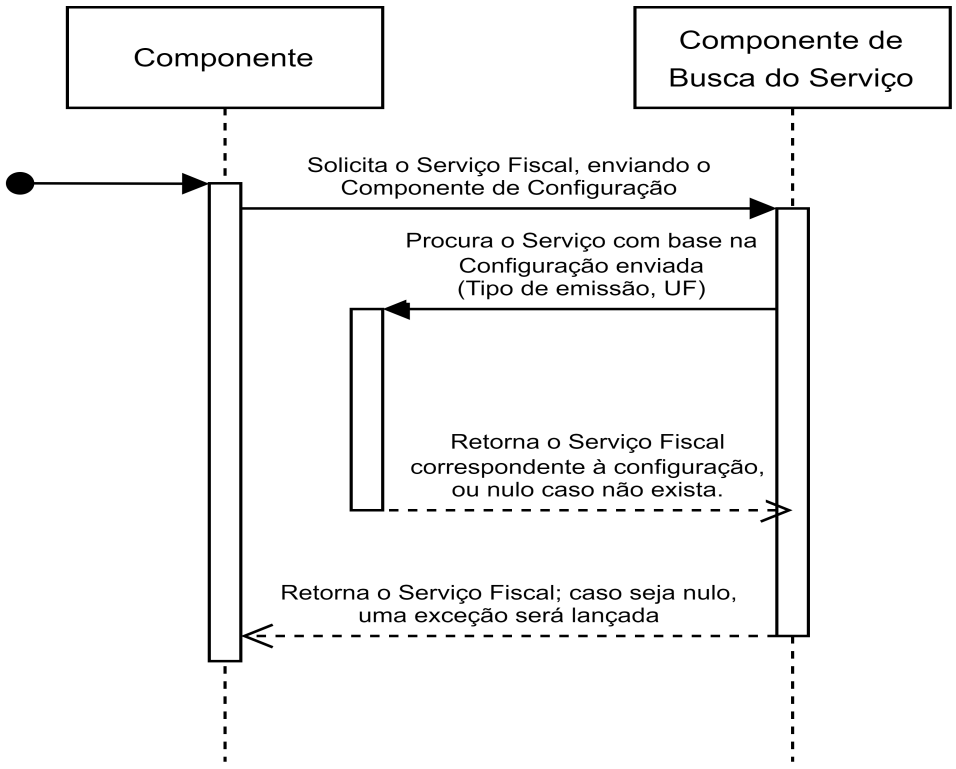
O Serviço Fiscal, também conhecido como Autorizador ou Sefaz, é um serviço externo ao qual o DF-e envia requisições. Esse serviço desempenha um papel crucial na aceitação das operações fiscais realizadas diariamente pelos contribuintes. Existem vários autorizadores, e eles podem aceitar requisições de contribuintes de uma única unidade federativa (UF) ou de várias UFs, sendo estes últimos chamados de Sefaz Virtual. O ponto fundamental a ser destacado é a diversidade de autorizadores, que podem lidar com as requisições de maneiras distintas e possuir arquiteturas diversas.

Embora o Manual de Orientação ao Contribuinte (MOC) especifique como o autorizador deve responder, a implementação real muitas vezes difere das expectativas.

Um exemplo notável é o autorizador do Mato Grosso, onde o envio de mensagens contendo acentuação resulta em rejeição. O motivo da rejeição é apresentado como “280 - Certificado Transmissor inválido”, embora, de acordo com o MOC, a mensagem de validação apropriada deveria ser “402 - XML da área de dados com codificação diferente de UTF-8.” Isso ilustra como a implementação real dos autorizadores pode variar em relação às especificações previstas.

O Componente de Busca de Serviço Fiscal tem como principal objetivo determinar qual é o serviço fiscal apropriado, utilizando o Componente de Configuração como guia. Esse componente é de extrema importância, pois a utilização inadequada de um serviço pode resultar em rejeições e até mesmo no bloqueio do uso do serviço pelo contribuinte. A Figura 7 apresenta um diagrama de sequência que ilustra, de forma simplificada, o fluxo de funcionamento do componente de busca de serviço implementado no DF-e.

Figura 7 – Componente de Busca do Serviço Fiscal.



Fonte: Produzido pelo autor.

O primeiro objeto, denominado “Componente”, envia a configuração do contribuinte para o Componente de Busca de Serviço. O Componente de Busca de Serviço, por sua vez, determina qual serviço é adequado para a configuração fornecida. Por exemplo, se a configuração indicar que a autorização de NF-e no Piauí com tipo de

emissão “Normal” é necessária, o serviço apropriado será o da Sefaz Virtual do Rio Grande do Sul (SVRS). No entanto, se o tipo de emissão for “Contingência”, o serviço adequado será o da Sefaz Virtual de Contingência de São Paulo (SVCSP). Se não for encontrado um serviço adequado para a operação solicitada, uma exceção será lançada, informando que o serviço solicitado não foi implementado ou não existe. É importante ressaltar que a variedade de autorizadores disponíveis adiciona uma complexidade adicional à aplicação, especialmente quando existem regras específicas para cada UF.

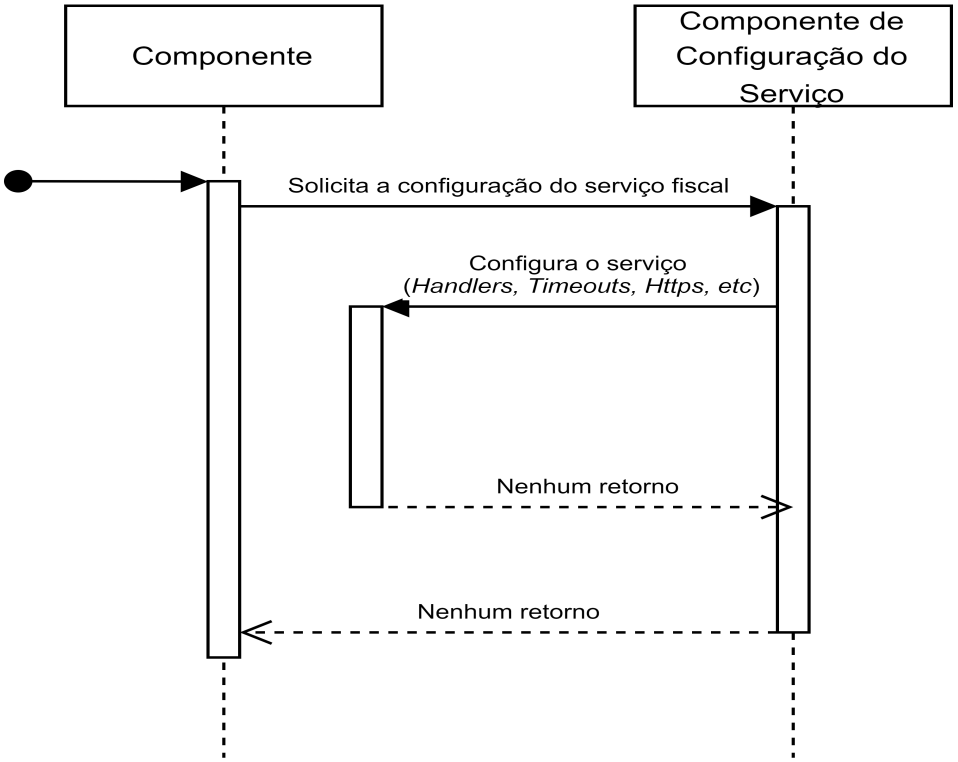
2.2.5 Configuração do Serviço Fiscal

Web Services Description Language (WSDL) é uma linguagem de descrição de serviços da web que é usada para definir a funcionalidade de um serviço da web. Ela descreve os métodos disponíveis, os parâmetros que podem ser passados para esses métodos, os tipos de dados envolvidos e como acessar o serviço.

No contexto do DF-e, os Serviços Fiscais são implementados com base nos WSDL fornecidos pelos autorizadores fiscais. No entanto, podem ocorrer situações em que esses WSDL estejam incorretos ou enfrentem adversidades. Um exemplo é o serviço de recepção do CT-e em Mato Grosso, onde a URL do serviço especificada utiliza *http* em vez de *https*. A utilização de *http* pode levar a erros de conexão, uma vez que *https* é uma conexão segura e é mais adequada para a troca segura de informações sensíveis, como dados fiscais.

O componente de Configuração do Serviço Fiscal funciona para configurar estas falhas, bem como quaisquer customizações que possam ser feitas no Serviço, bem como *timeouts*, *handlers* de requisição e resposta. A Figura 8 apresenta um diagrama de sequência que ilustra, de forma simplificada, o fluxo de funcionamento do componente de busca de serviço implementado no DF-e.

Figura 8 – Componente de Configuração do Serviço Fiscal.



Fonte: Produzido pelo autor.

O primeiro objeto, denominado “Componente”, envia o serviço a ser configurado, bem como a configuração do contribuinte, para o Componente de Configuração do Serviço Fiscal. Este último realizará configurações no serviço, como o uso do *https*, a utilização do certificado da configuração do contribuinte na requisição, a criação padrão de *handlers* de requisição e resposta para fins de monitoria, além de definir um *timeout* nas requisições. E, por fim, nenhum valor será retornado, uma vez que o objeto do serviço foi alterado e não foi criado um novo objeto.

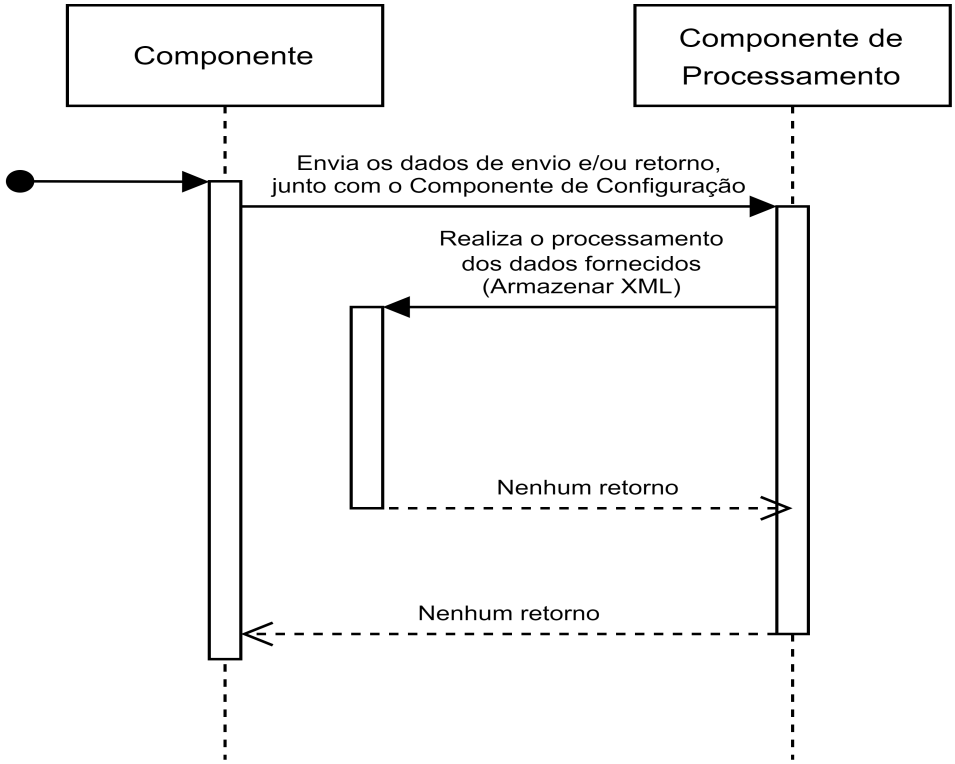
2.2.6 Processamento

No Brasil, a legislação estabelece que os arquivos eletrônicos fiscais devem ser armazenados pelo prazo de cinco anos, a contar da data de emissão do documento fiscal. Essa prática assegura a conformidade fiscal, facilita consultas e auditorias, além de contribuir para a gestão financeira da empresa. O Componente de Processamento foi concebido com o objetivo de realizar ações antes e após operações fiscais específicas, e sua implementação padrão no ambiente do DF-e é focada no armazenamento do XML de documentos fiscais.

Apesar de sua simplicidade aparente, o Componente de Processamento se destaca pela notável flexibilidade que oferece. Essa flexibilidade possibilita a execução de uma ampla gama de ações personalizadas. É possível realizar validações específicas para cada Unidade Federativa (UF) antes da conclusão de uma operação, ou, após a emissão de uma nota fiscal, encaminhá-la automaticamente por e-mail ao destinatário. Dessa forma, inúmeras regras de negócio podem ser implementadas de acordo com as necessidades da empresa. Essa versatilidade torna o Componente de Processamento uma ferramenta essencial para a adaptação e otimização de processos fiscais, agregando valor significativo às operações das empresas.

O componente de processamento se divide em duas interfaces fundamentais: uma para o pré-processamento dos dados antes do envio da requisição, que inclui a mensagem e a configuração do contribuinte, e outra para a execução de ações posteriores à requisição, onde são fornecidos a mensagem de envio, a resposta do autorizador e a configuração do contribuinte. Essa abordagem se mostra essencial devido à necessidade de compor um XML que contenha tanto a mensagem do contribuinte quanto a resposta do autorizador, garantindo assim sua validade jurídica. A Figura 9 oferece um diagrama de sequência simplificado, proporcionando uma visão do funcionamento do componente de processamento implementado no DF-e

Figura 9 – Componente de Processamento.



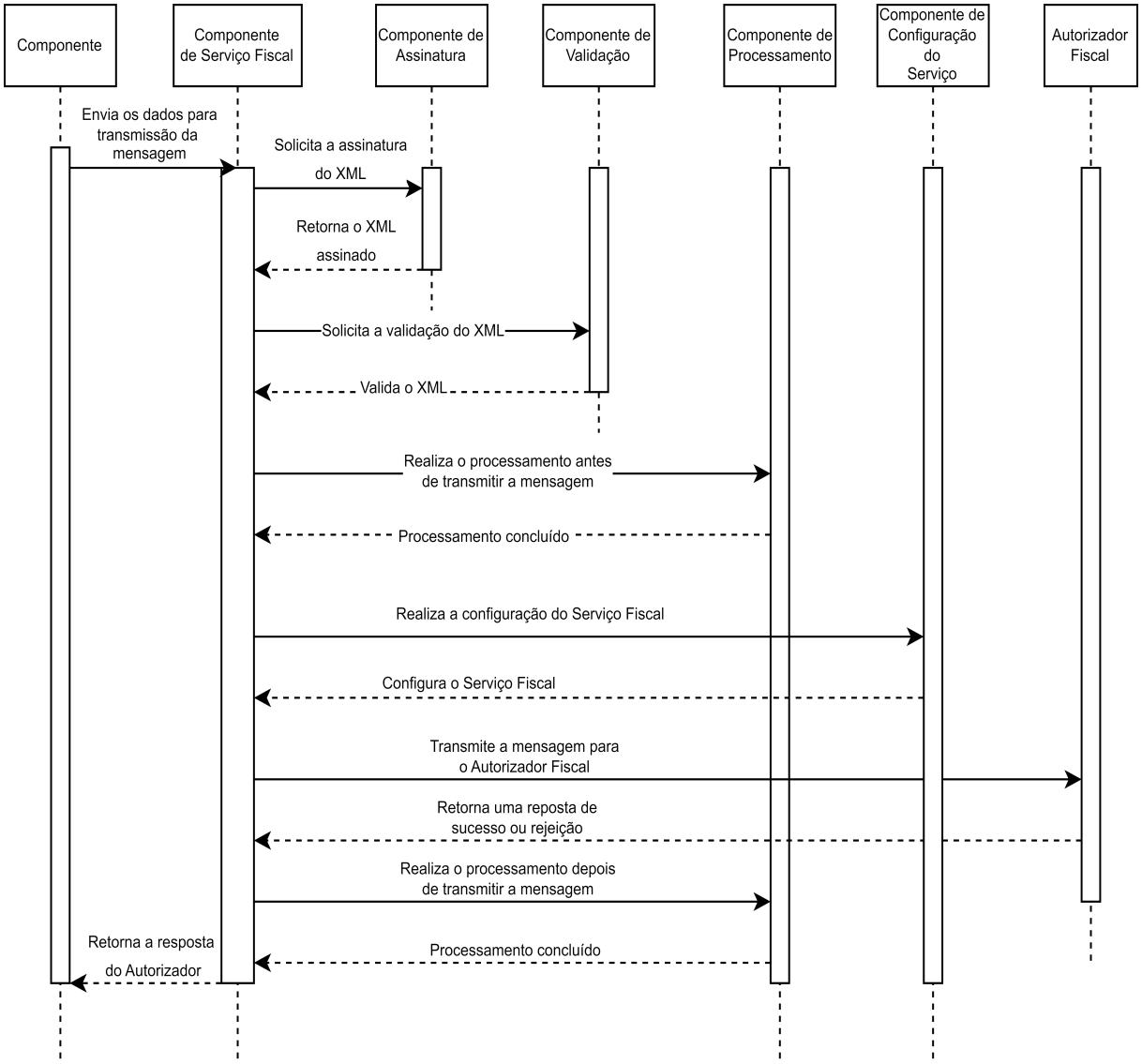
Fonte: Produzido pelo autor.

O primeiro objeto, denominado “Componente”, transmite os dados de envio do contribuinte e, quando aplicável, os dados de resposta do autorizador, juntamente com a configuração do contribuinte, ao Componente de Processamento. Este último executa o processamento dos dados, que, na implementação padrão, consiste na simples tarefa de armazenar o XML para fins de auditoria. No caso de qualquer erro durante esse processo, uma exceção é lançada para interromper o fluxo.

2.2.7 Serviço Fiscal

O Componente de Serviço Fiscal se vale dos componentes de configuração, assinatura, validação, processamento e configuração do serviço fiscal para efetuar a transmissão de mensagens aos servidores de autorização. A Figura 9 fornece um diagrama de sequência simplificado que oferece uma visão de como esse componente desempenha sua função e como se integra com os demais componentes do sistema.

Figura 10 – Componente de Serviço Fiscal.



Fonte: Produzido pelo autor.

O processo se inicia com o primeiro objeto, denominado “Componente”, que executa uma operação fiscal por meio do Componente de Serviço Fiscal. Em seguida, a solicitação de assinatura do XML é realizada, caso necessário. Após a assinatura, o XML passa por uma etapa de validação pelo componente de validação. Posteriormente, após a conclusão da etapa de pré-processamento do XML, a configuração do serviço fiscal é estabelecida, e a chamada para o autorizador fiscal é feita. Quando o autorizador responde, ocorre o pós-processamento da mensagem, e a resposta do autorizador é então retornada ao primeiro componente.

2.2.8 Serviço DF-e

O Serviço DF-e desempenha o papel de um *wrapper* que engloba todos os outros componentes, tornando o uso deles mais acessível por meio da adição

de métodos auxiliares. Um exemplo disso, como mencionado anteriormente, é a necessidade de fornecer todos os outros componentes ao utilizar o serviço fiscal. Ao instanciar o Serviço DF-e, ele já inclui todos os componentes padrões. Portanto, para enviar uma mensagem ao autorizador, basta enviá-la ao Serviço DF-e, que cuidará de efetuar a chamada ao serviço fiscal apropriado e transmitir todos os parâmetros necessários. Isso simplifica significativamente o processo, tornando-o mais prático e eficiente.

Além disso, o Serviço DF-e é altamente extensível. Se houver a necessidade de integrar serviços externos de autorização ou etapas de processamento customizadas, basta fornecê-los ao Serviço DF-e, que os utilizará de maneira adequada. Essa flexibilidade permite a expansão das funcionalidades do sistema de forma personalizada, atendendo às necessidades específicas do ambiente de emissão de documentos fiscais eletrônicos.

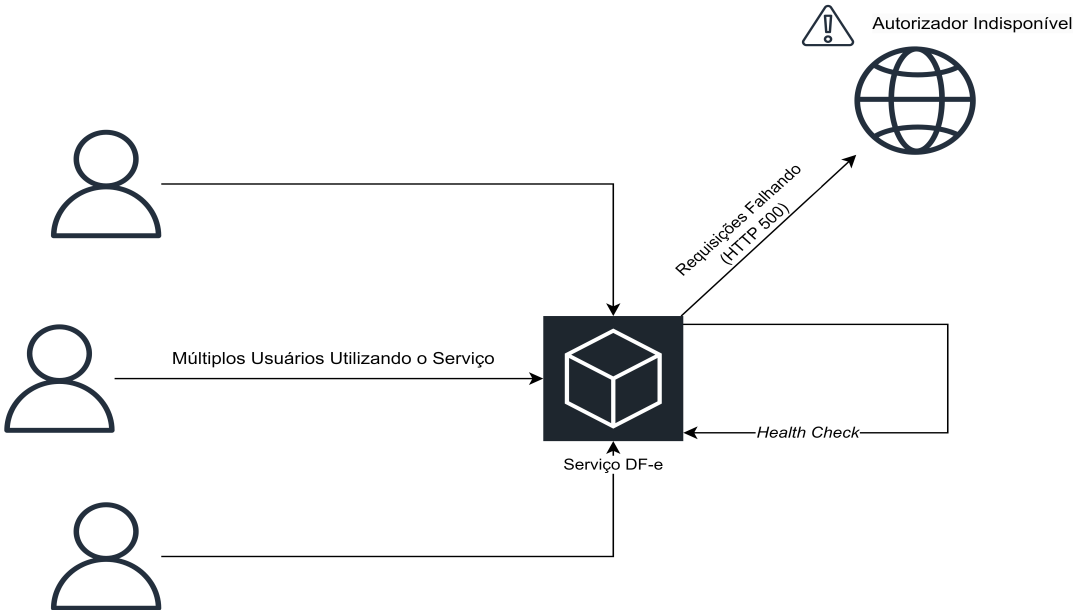
3 DESCRIÇÃO DE ATIVIDADES

Nesta seção são apresentadas as atividades realizadas para a análise e aplicação de padrões de resiliência à biblioteca DF-e, desde o planejamento inicial até a implementação prática dos padrões. O objetivo é oferecer uma visão abrangente das ações realizadas para melhor compreensão da pesquisa conduzida e dos benefícios alcançados ao tornar a biblioteca mais resiliente.

3.1 APLICAÇÃO DOS PADRÕES DE RESILIÊNCIA EM RELAÇÃO À BIBLIOTECA DF-E

O DF-e, como qualquer outro *software*, está sujeito a falhas. Mesmo com implementação de testes abrangentes, situações imprevistas podem ocorrer. É nesse contexto que a resiliência de *software* desempenha um papel fundamental. A primeira etapa para tornar um *software* resiliente é identificar os pontos críticos de falha que têm potencial para afetar o sistema de maneira abrangente. Isso envolve analisar como as funcionalidades implementadas podem impactar o *software* em produção. A Figura 11 ilustra um exemplo concreto de como a resiliência de *software* tem impactos significativo na utilização de um serviço.

Figura 11 – *Health Check* em um Serviço Fiscal.



Fonte: Produzido pelo autor.

Considerando um cenário em que a biblioteca DF-e está sendo executada em um ambiente que realiza verificações de integridade. Essas verificações são essenciais para garantir que o sistema funcione sem problemas. Elas operam por meio de códigos de resposta HTTP, onde um código de erro de servidor (um código HTTP igual ou superior a 500) sinaliza que algo está errado no ambiente. Agora, supondo que, em um momento de grande demanda, um dos autorizadores fique temporariamente indisponível. Nessa circunstância, todas as solicitações para este autorizador feitas ao sistema resultariam em erros de servidor, tornando o ambiente inacessível. Isso ocorre porque a resiliência não foi considerada, e o sistema não é capaz de se adaptar a essa situação desafiadora.

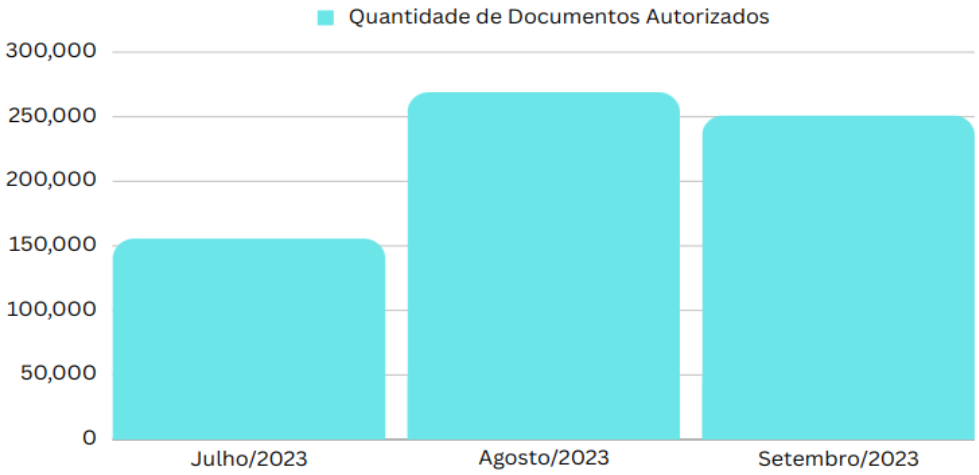
Neste cenário, a importância da resiliência fica clara. Ter a capacidade de lidar com falhas momentâneas ou interrupções no serviço é fundamental para manter a disponibilidade e o desempenho do sistema, mesmo quando ocorrem contratempos imprevistos. É como garantir que, mesmo em tempestades, a estrada continue segura e trafegável para todos os motoristas.

Um fator importante é a aplicação da resiliência de *software* às operações que são classificadas como críticas para um sistema. Isso se refere às funções ou processos que desempenham um papel vital na operação contínua do sistema, impactando diretamente a disponibilidade, a confiabilidade e o desempenho do mesmo. Assegurar a resiliência nesses pontos críticos é essencial para manter a integridade do sistema e a satisfação dos usuários, especialmente em ambientes onde a estabilidade e a confiabilidade são fundamentais, como é o caso das operações fiscais em sistemas como o DF-e.

3.1.1 Uso e disponibilidade do DF-e

A biblioteca DF-e é atualmente empregada no serviço de emissão fiscal da empresa Tartigrado Tecnologia LTDA, desempenhando um papel crucial na emissão de milhares de operações fiscais diariamente. Esse serviço é adotado por contribuintes de todos os estados do Brasil, com a exceção do Acre. Essa ampla utilização evidencia a importância do DF-e em âmbito nacional. O gráfico apresentado na Figura 12 ilustra o volume de documentos autorizados nos três primeiros meses do segundo semestre de 2023, enfatizando ainda mais a relevância do DF-e como uma solução fundamental no contexto fiscal brasileiro.

Figura 12 – Volume de Documentos Autorizados pelo DF-e.



Fonte: Produzido pelo autor.

No mês de julho, observa-se um descompasso em comparação aos meses subsequentes devido à implantação da utilização do DF-e ter ocorrido na metade do mês. No entanto, os próximos meses revelam um volume de mais de duzentos e cinquenta mil documentos autorizados, demonstrando assim o considerável uso do DF-e e sua significativa importância para a autorização de documentos em todo o Brasil.

Manter a disponibilidade contínua de um serviço é essencial para a realização ininterrupta de operações fiscais. No entanto, a natureza dinâmica do sistema fiscal brasileiro, sujeito a mudanças frequentes, bem como as próprias alterações cotidianas no serviço, impõe o desafio da atualização do *software*. Durante essas atualizações, é comum que os servidores fiquem temporariamente indisponíveis. Nesse contexto, surge a pergunta: como garantir que o sistema permaneça acessível e responsivo, mesmo quando sujeito a modificações a qualquer momento?

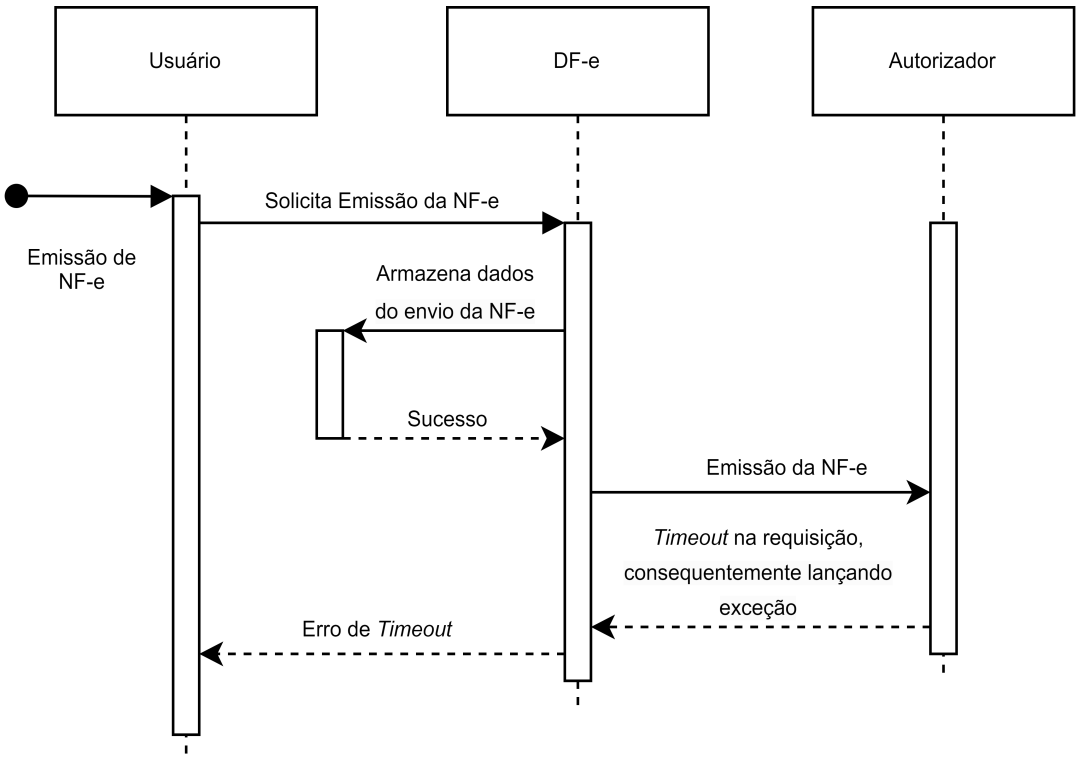
Uma abordagem eficaz para resolver esse desafio é a implementação da técnica de *autoscaling*. Isso envolve configurar o serviço fiscal para manter várias instâncias em operação simultaneamente. Durante atualizações, as instâncias podem ser atualizadas sequencialmente ou em lote, com a precaução de sempre manter uma quantidade adequada de instâncias disponíveis para garantir a operação contínua do *software*. A quantidade de instâncias mantidas em operação pode ser ajustada com base na demanda, assegurando que o sistema permaneça responsivo e resiliente. Essa abordagem permite que as atualizações ocorram de forma controlada, minimizando interrupções e preservando a disponibilidade do serviço fiscal.

3.1.2 Armazenamento de XML

O armazenamento dos XMLs representa uma obrigação legal no Brasil. De acordo com a legislação vigente, os arquivos eletrônicos fiscais devem ser retidos por um período mínimo de cinco anos, a partir da data de emissão do documento fiscal. Portanto, é necessário garantir a resiliência no armazenamento desses XMLs. Em situações de falhas ou interrupções, o DF-e deve ser capaz de empregar estratégias alternativas para a recuperação desses documentos, assegurando o cumprimento das obrigações fiscais estabelecidas.

A adoção de um sistema de armazenamento confiável, a implementação de estratégias de retentativas controladas durante o armazenamento e a utilização de alternativas de armazenamento em caso de falha são abordagens eficazes para incorporar a resiliência de *software* na gestão específica de armazenamento de XML. No entanto, a situação ilustrada na Figura 13 evidencia um cenário no qual a simples aplicação de retentativas pode não ser suficiente para garantir a eficácia.

Figura 13 – Problema de Timeout ao Armazenar o XML.

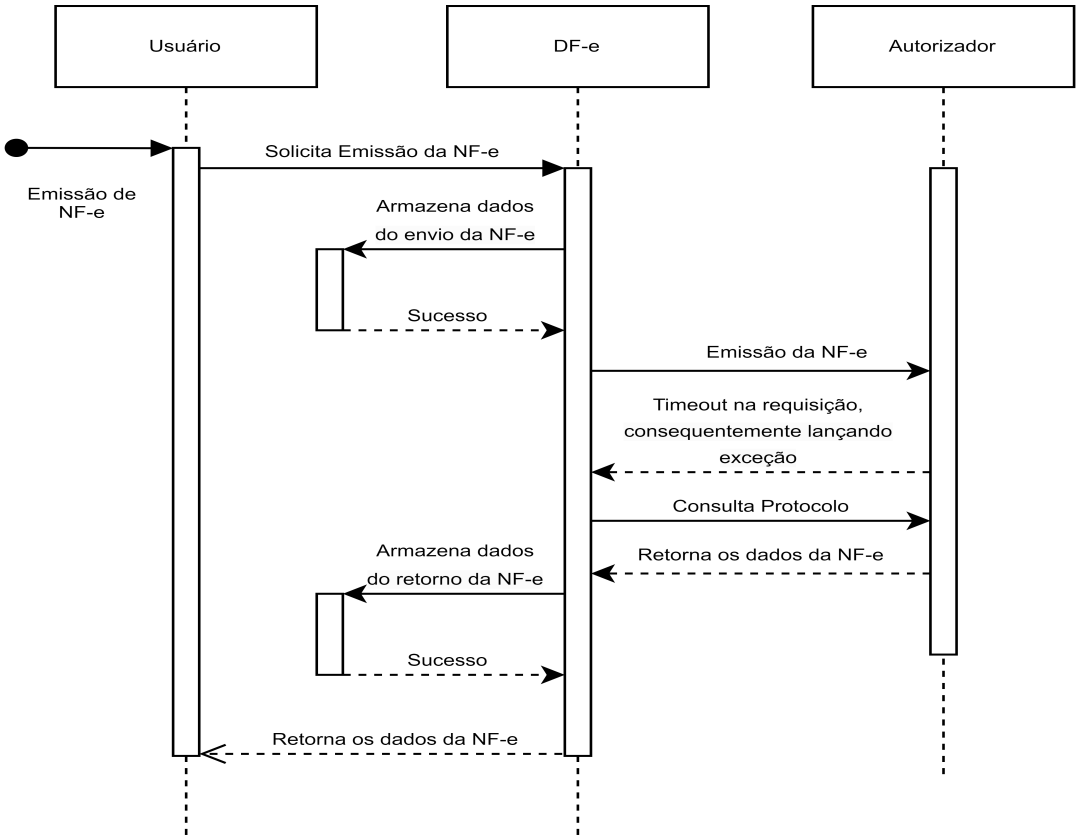


Fonte: Produzido pelo autor.

Nesse contexto, o usuário está realizando a emissão de uma Nota Fiscal Eletrônica (NF-e) e o DF-e irá realizar o armazenamento do XML, tendo em vista que é necessário tanto o envio do contribuinte como o retorno do autorizador. O armazenamento do XML de envio ocorre com sucesso, no entanto, a requisição para o autorizador não é respondida em tempo hábil, resultando em um erro de *timeout*. Vale ressaltar que o erro de *timeout* não implica na não emissão da NF-e, podendo esta ser emitida ou não.

O problema da retentativa nesses casos é que, se a Nota Fiscal Eletrônica (NF-e) for emitida com sucesso, apesar do *timeout*, o contribuinte irá receber a rejeição "Rejeição 539 - Duplicidade de NF-e, com diferença na Chave de Acesso," o que significa que já foi emitida uma NF-e com aquelas informações. Na especificação do MOC, existe um serviço denominado Consulta Protocolo, que deve ser implementado pelos autorizadores e tem como objetivo disponibilizar informações a respeito de uma NF-e e eventos relacionados para o contribuinte. A problemática exposta acima poderia ser resolvida no fluxo determinado pela Figura 14.

Figura 14 – Solução do Problema de Timeout ao Armazenar o XML.



Fonte: Produzido pelo autor.

O fluxo segue conforme especificado na explicação da Figura 13. No entanto, após a requisição de emissão falhar com *timeout*, o DF-e executará a Consulta Protocolo ao Autorizador para verificar se a NF-e foi emitida com sucesso. Em caso

afirmativo, o DF-e armazenará os dados de retorno da NF-e, criando assim um XML com validade jurídica, que será retornado ao usuário.

Usar um serviço de armazenamento confiável, mesclando estratégias de retentativa durante o processo de armazenamento, atrelado ao uso do serviço de Consulta Protocolo durante emissão de NF-e assegura que o DF-e é resiliente no que se diz respeito ao armazenamento de XML, promovento assim a validade jurídica de suas operações fiscais.

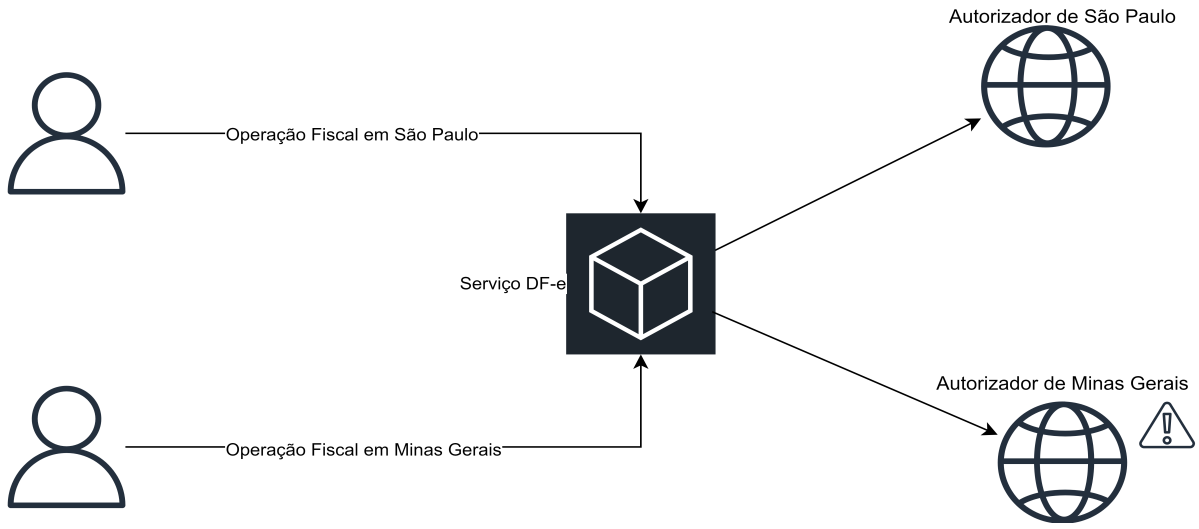
Estratégias de retentativas controladas desempenham um papel fundamental, pois uma aplicação se esforçará ao máximo para gerar uma resposta bem-sucedida para o usuário. No entanto, existe uma situação crítica em que o autorizador fica completamente indisponível e não consegue mais responder a nenhuma solicitação. Além disso, tentar realizar solicitações nesse cenário torna-se prejudicial tanto para o autorizador quanto para o DF-e. Para enfrentar essa situação específica, uma solução especializada será abordada na próxima seção.

3.1.3 Indisponibilidade de autorizadores

A comunicação eficaz com os serviços fiscais é fundamental para o DF-e, que tem como objetivo simplificar tarefas complexas, como a interação via protocolo *Simple Object Access Protocol* (SOAP), assinatura de documentos e validação com XSDs. Garantir a resiliência é essencial, pois situações adversas, como a indisponibilidade dos serviços fiscais, podem ocorrer. Portanto, é necessário adotar estratégias que permitam contornar essas adversidades e garantir o correto funcionamento do sistema mesmo em condições desafiadoras.

Um fator a ser considerado na implementação é a diversidade de autorizadores e serviços fiscais existentes, além disso, a indisponibilidade de um autorizador ou um serviço específico deve ser independente de outros, a Figura 15 demonstra um exemplo concreto da comunicação do DF-e com múltiplos autorizadores.

Figura 15 – Múltiplos Autorizadores.

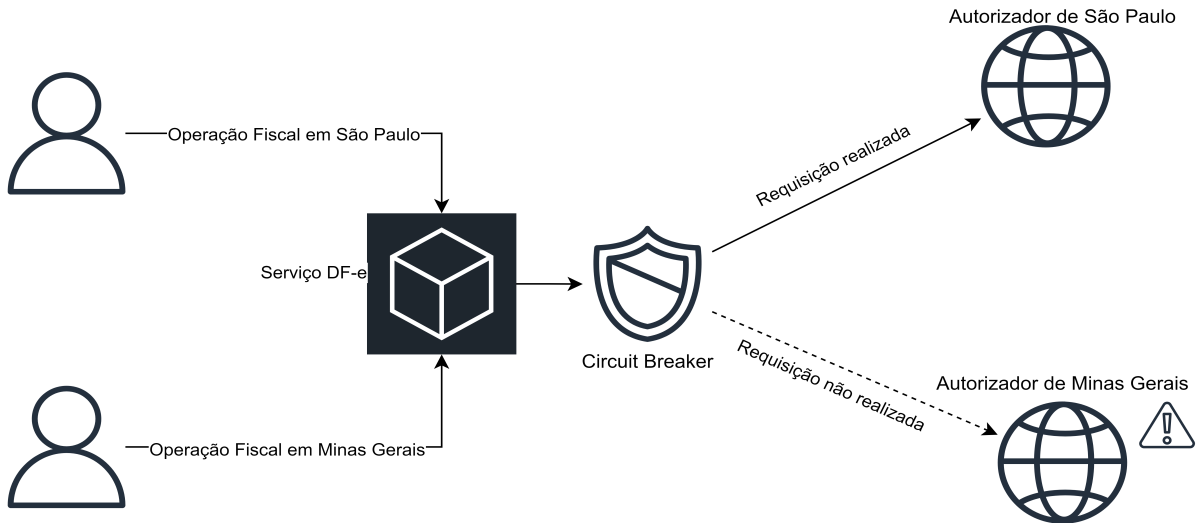


Fonte: Produzido pelo autor.

A imagem ilustra usuários de diferentes estados (UF) realizando operações fiscais. No entanto, observa-se que o autorizador de Minas Gerais está indisponível. O DF-e continua a efetuar solicitações, mesmo estando ciente da indisponibilidade, resultando em falhas constantes. Além disso, recursos do sistema estão sendo alocados para operações que, inevitavelmente, falharão. Além disso, é importante destacar que o DF-e compreende uma variedade de serviços destinados a atender diversas operações fiscais, como emissão de documentos, eventos, consulta de protocolo, entre outros. Seguindo a mesma lógica, a indisponibilidade de um desses serviços não deve ter um impacto direto na utilização de outros, uma vez que cada serviço opera de forma independente e isolada.

O *Circuit Breaker* é um padrão de resiliência ideal para lidar com falhas em serviços externos. Ao aplicá-lo como uma camada intermediária entre o DF-e e o autorizador fiscal, o *Circuit Breaker* se encarrega de gerenciar todas as falhas no autorizador, registrando e verificando a cada operação fiscal. Em situações de falhas persistentes, o *Circuit Breaker* não realizará a operação, em vez disso, lançará uma exceção informando a indisponibilidade do servidor. A Figura 16 apresenta a inclusão da camada do *Circuit Breaker* nesta arquitetura.

Figura 16 – Circuit Breaker em Múltiplos Autorizadores.



Fonte: Produzido pelo autor.

Seguindo o mesmo fluxo apresentado na Figura 15, a diferença crucial agora é a intervenção do *Circuit Breaker*, que impede a realização da operação fiscal quando o Autorizador está indisponível. Além disso, o *Circuit Breaker* notifica o usuário em tempo hábil. Em contrapartida, se não fosse pela presença do *Circuit Breaker*, o tempo de resposta dependeria da rapidez com que o Autorizador falharia. Na ausência de uma política de *timeout*, o DF-e ficaria aguardando indefinidamente por uma resposta que nunca chegaria.

Considerando a escalabilidade do serviço que utiliza o DF-e, torna-se crucial adotar uma estratégia de armazenamento de rápido acesso e acessível entre diversas instâncias, como o *Redis*, na implementação do *Circuit Breaker*. Essa abordagem assegura que, em ambientes com políticas de autoscaling horizontal, todas as instâncias tenham ciência da indisponibilidade de um Autorizador. Dessa forma, evita-se a necessidade de falhar as operações em todas as instâncias para que o circuito seja aberto.

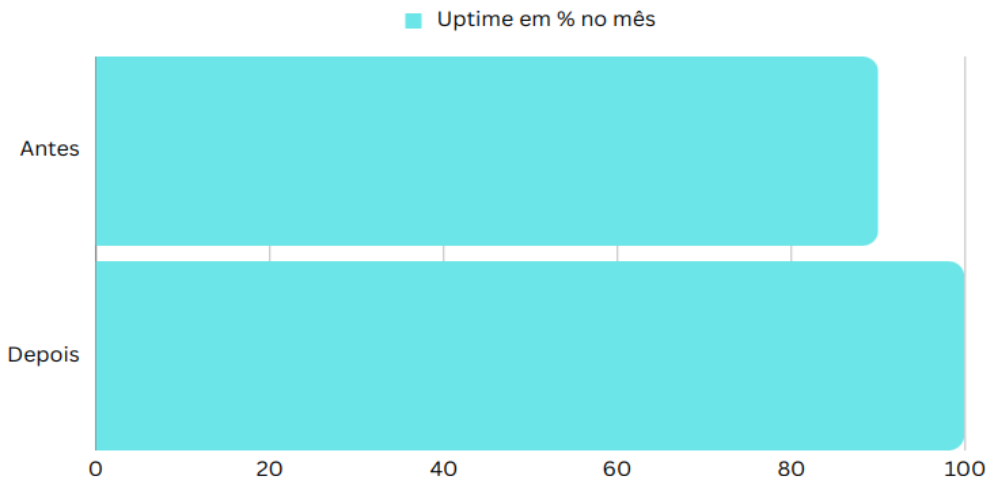
3.2 ANÁLISE DOS RESULTADOS

A implementação das estratégias de resiliência à biblioteca DF-e trouxe resultados significativos que impactaram positivamente a eficácia, confiabilidade e escalabilidade do sistema. Nesta seção, serão apresentados os resultados das estratégias adotadas.

3.2.1 Autoscaling

A implementação do *autoscaling* teve um impacto significativo na disponibilidade do Serviço Emissor, o qual utiliza o DF-e como sua base. Isso garantiu que o sistema continuasse operacional mesmo diante de atualizações e variações na demanda de uso. Essa melhoria é claramente demonstrada na Figura 17.

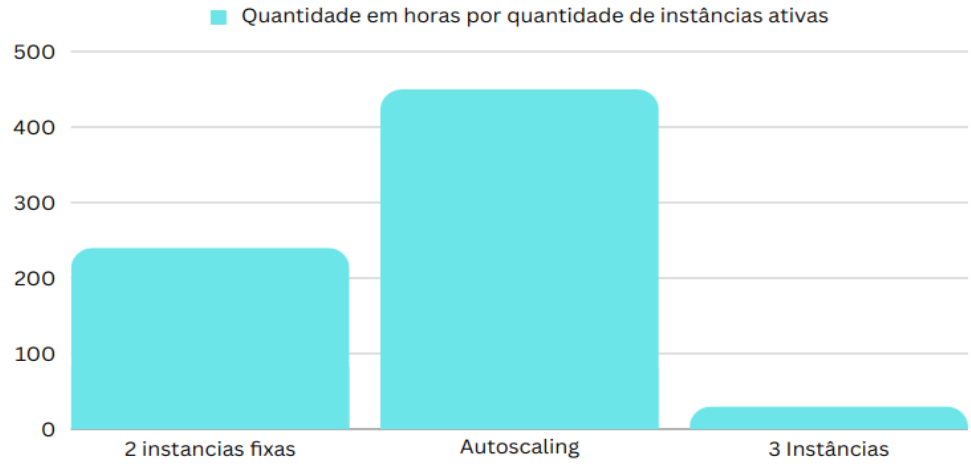
Figura 17 – Disponibilidade do Serviço Emissor.



Fonte: Produzido pelo autor.

A disponibilidade do Serviço Emissor era de 90% antes da implementação das estratégias de autoscaling, e aumentou para 100% após a adoção do autoscaling. Mesmo em horários de alta demanda de operações fiscais, o sistema permaneceu disponível graças ao escalonamento horizontal. A Figura 18 ilustra a quantidade de horas por quantidade de instâncias ao longo de um período de um mês.

Figura 18 – Quantidadade de horas por quantidade de instâncias do Serviço Emissor.

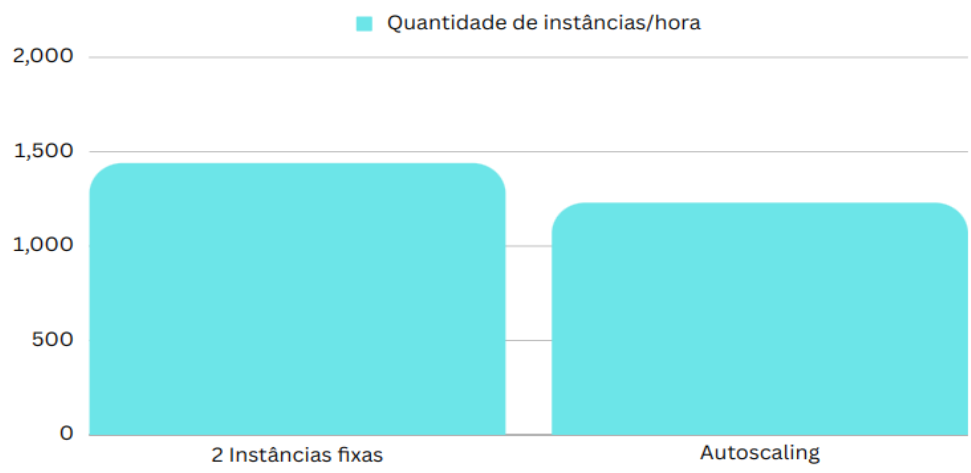


Fonte: Produzido pelo autor.

Durante um mês, observou-se que o *autoscaling* manteve uma instância ativa por 240 horas, correspondentes ao período das 22h às 6h, horário em que as operações fiscais diminuíam. Duas instâncias ativas por 450 horas, o que representava o uso padrão do serviço. E três instâncias ativas por 30 horas, referentes a picos de uso do serviço, quando era necessário adicionar mais uma instância para lidar com a demanda.

Pode-se analisar também que o uso do *autoscaling* resultou em uma redução nos custos da infraestrutura como um todo. Caso fosse empregada uma estratégia com 2 instâncias fixas, os custos seriam incorridos mesmo em períodos em que não seria necessária tanta capacidade de recursos. A Figura 19 ilustra a redução de custos proporcionada pelo *autoscaling* neste cenário, uma vez que as instâncias são cobradas pelo mesmo preço.

Figura 19 – Quantidade de instâncias/hora do Serviço Emissor.



Fonte: Produzido pelo autor.

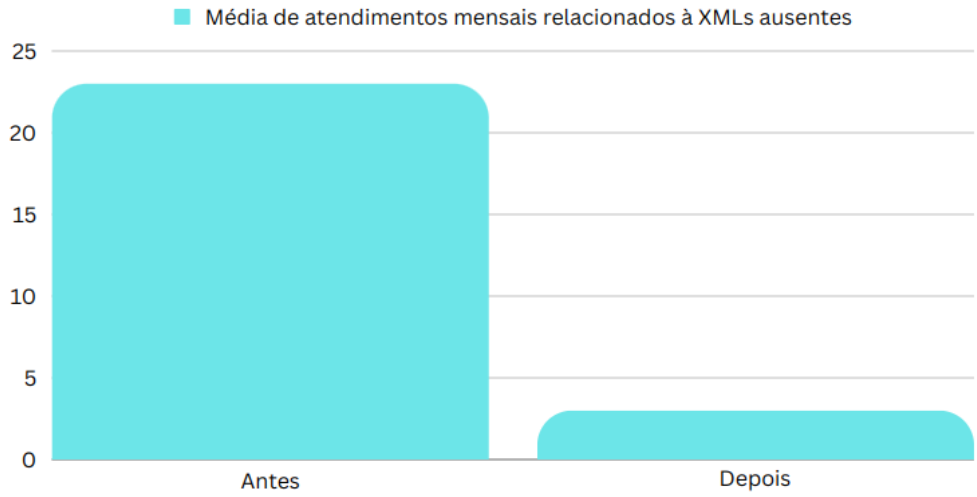
Observa-se que a adoção do *autoscaling* resultou em uma redução de 14.58% no uso de instâncias por hora. Além disso, sem a implementação do *autoscaling* para ampliar o número de instâncias em horários de pico, a fim de acomodar a demanda dos usuários, seria possível que o serviço sofresse com lentidão ou até mesmo se tornasse indisponível. Isso poderia acarretar em consequências que vão além da infraestrutura de sistema, afetando a satisfação dos usuários do serviço.

3.2.2 Armazenamento de XML

Para garantir que os XMLs sejam armazenados de maneira segura e acessível apenas a usuários autorizados, foi utilizado o serviço de armazenamento *S3* da *Amazon Web Services* (AWS). Além disso, o uso Software Development Kit (SDK) da AWS garante a implementação de uma política de retentativa em caso de falhas temporárias no serviço, embora seja importante mencionar que o S3 tem um histórico de confiabilidade, com um *uptime* anual de 99.99%.

Outro detalhe significativo a ser mencionado é o mecanismo de contingência em caso de falha do sistema de armazenamento. Quando ocorre uma falha, o sistema faz uso do armazenamento local da instância e, em seguida, tenta novamente o processo de salvamento do arquivo XML no S3, utilizando uma abordagem baseada em filas. Essa estratégia garante tentativas controladas e evita sobrecarregar o servidor com operações de salvamento em massa. Uma métrica importante, resultado da implementação desse sistema de arquivos resilientes, é a redução significativa do número de atendimentos ao clientes em relação a arquivos XML ausentes, como ilustrado na Figura 20.

Figura 20 – Média de atendimentos mensais relacionados à XMLs ausentes.



Fonte: Produzido pelo autor.

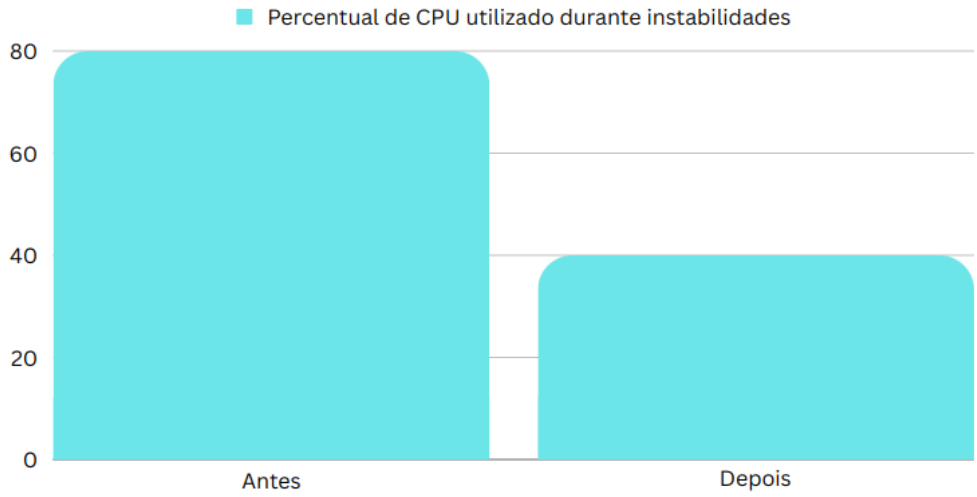
Após a implementação, eventos relacionados à ausência de XML na base de dados eram frequentes, exigindo intervenções manuais para importar os arquivos corretamente. Esses erros ocorriam principalmente quando os clientes executavam operações fiscais em grande volume, resultando em falhas ao salvar os XMLs adequadamente. Com a introdução de um sistema de armazenamento resiliente para XML, esses incidentes diminuíram significativamente. Isso se traduz em um aumento na satisfação do cliente, uma vez que o sistema possui um alto nível de confiabilidade em relação ao armazenamento de XML.

Também foi constatado, após a implementação, que muitos dos atendimentos relacionados à ausência de XML na base de dados estavam ligados a usuários que não tinham acesso ao arquivo, criando uma falsa impressão de que o XML estava ausente. Esse aspecto está mais relacionado à experiência do usuário do que à resiliência do *software*.

3.2.3 Circuit Breaker

As situações em que os autorizadores fiscais ficam indisponíveis são relativamente comuns no cenário atual. Para lidar com essas falhas eventuais, foi implementado o padrão *Circuit Breaker*. Antes da adoção desse padrão, quando os servidores fiscais ficavam indisponíveis, o tempo médio de resposta das requisições aumentava. Além disso, o uso de recursos era mais elevado devido ao bloqueio das *threads*, que continuavam tentando realizar as operações fiscais mesmo quando o autorizador não podia responder. A Figura 21 apresenta o percentual de CPU utilizado durante períodos de instabilidade antes e depois da implementação do *Circuit Breaker*.

Figura 21 – Percentual de CPU utilizada em instabilidades.



Fonte: Produzido pelo autor.

O percentual de CPU quase atingia o limite da instância antes da implementação do *Circuit Breaker*. Isso ocorria devido ao custo de tentar realizar requisições para servidores indisponíveis, o que resultava em um grande consumo de recursos pelas *threads* do servidor, que ficavam esperando por uma resposta de operações que eventualmente falhavam. Após a implementação do *Circuit Breaker*, o percentual de CPU se estabilizou em torno de 40%, que é um valor padrão ao observar todos os dados históricos do servidor.

Adicionalmente, houve uma melhora notável no Tempo médio entre falhas (MTBF), uma vez que, anteriormente à implementação do *Circuit Breaker*, o Serviço Emissor sofria falhas em cascata devido à indisponibilidade dos Autorizadores. Com a implementação, o Serviço Emissor passou a tolerar essas falhas, continuando suas operações sem problemas adicionais. Também, o Tempo médio de recuperação (MTTR) foi reduzido, pois o *Circuit Breaker* permitiu uma recuperação automática mais eficaz em caso de falhas nos Autorizadores. Isso resultou em um serviço mais resiliente, capaz de lidar com falhas de maneira mais eficaz.

4 CONCLUSÃO

O presente trabalho buscou aprimorar a resiliência da biblioteca DF-e, essencial para operações fiscais, através da aplicação de padrões de resiliência em diferentes áreas críticas do sistema. A abordagem adotada envolveu a análise e implementação de estratégias para melhorar a estabilidade, eficácia e confiabilidade do DF-e.

Inicialmente, foram delineados os objetivos do trabalho, centrados na aplicação de padrões de resiliência à biblioteca DF-e. Posteriormente, foram explorados conceitos teóricos de resiliência de software, destacando padrões como o *Circuit Breaker*, *Retry Pattern* e *Auto Scaling*. Além disso, foi apresentada a biblioteca DF-e, com uma visão detalhada de sua arquitetura e componentes.

Iniciou-se a abordagem prática com a identificação de desafios específicos, como alta disponibilidade do serviço fiscal, armazenamento de XML e a indisponibilidade de autorizadores fiscais. A aplicação de padrões como *autoscaling* contribuiu significativamente para elevar a disponibilidade do Serviço Emissor de 90% para 100%, otimizando custos. No armazenamento de XML, estratégias resilientes reduziram intervenções manuais, melhorando a satisfação do cliente. A introdução do *Circuit Breaker* mitigou falhas nos autorizadores, estabilizando o uso de recursos.

Os resultados foram obtidos através da análise de *logs* dos servidores, que permitiu extrair métricas como tempo médio de resposta, histórico de falhas e erros. Além disso, dados relacionados a atendimentos ao cliente foram coletados por meio da ferramenta *Zendesk*. Os ganhos foram expressivos, incluindo uma redução de 14.58% no uso de instâncias por hora com a implementação do *autoscaling*, uma diminuição significativa nos atendimentos relacionados à ausência de XML e uma estabilização do uso de recursos com a introdução do *Circuit Breaker*. A Tabela 2 resume os principais resultados deste trabalho.

Tabela 2 – Resultados da aplicação de padrões de resiliência

Pattern	**Resultado**
AutoScaling	Disponibilidade do Serviço durante atualizações
AutoScaling	Redução de custo em horários de baixa disponibilidade
AutoScaling	Disponibilidade do Serviço durante uso intenso
Retry/Timeout	Evita erros relacionados à armazenamento de XML
Retry/Timeout	Mitigação de erros durante falhas transitórias
Circuit Breaker	Redução do uso de recursos durante instabilidades
Circuit Breaker	Melhoria do tempo de resposta durante instabilidades

Fonte: Produzido pelo autor.

A implementação de diversos padrões de resiliência no software resulta em diversos benefícios relacionados à estabilidade do serviço. Esses benefícios abrangem desde a otimização dos recursos de hardware, com o uso do *Circuit Breaker*, até a redução de custos por meio do *autoscaling*. Além disso, a aplicação da técnica de *retry/timeout* contribui para a satisfação do usuário final, prevenindo erros transitórios

4.1 TRABALHOS FUTUROS

À medida que este projeto estabeleceu uma base sólida na aplicação de padrões de resiliência à biblioteca DF-e, abre-se um caminho para futuras explorações e refinamentos. Os trabalhos futuros são essenciais para a contínua evolução e aprimoramento do sistema, abordando aspectos específicos e aproveitando tecnologias emergentes. Nesta seção, serão delineadas direções que podem ser exploradas para ampliar ainda mais a resiliência e eficácia da biblioteca DF-e.

4.1.1 Otimização de Desempenho

A otimização de desempenho emerge como um tópico essencial para aprimorar ainda mais a biblioteca DF-e. Esta área propõe explorar estratégias e técnicas que visam maximizar a eficiência operacional do sistema, reduzindo tempos de resposta e otimizando o consumo de recursos.

- **Afinamento dos Padrões de Resiliência**: Considerando a diversidade de cenários e operações fiscais, é possível realizar uma análise mais detalhada dos padrões de resiliência aplicados. Isso inclui ajustar parâmetros específicos, como limites de tentativas e períodos de *timeout*, para otimizar o equilíbrio entre a resiliência e a eficiência operacional;
- **Implementação de Cache Estratégico**: A introdução de uma camada de cache estratégico pode ser explorada para armazenar temporariamente resultados de operações frequentes. Isso reduziria a necessidade de consultas repetidas a

certos serviços externos, melhorando significativamente os tempos de resposta e aliviando a carga sobre esses recurso;

- **Otimização de Algoritmos Críticos**: Ao examinar algoritmos essenciais para a funcionalidade da biblioteca DF-e, pode-se explorar opções para otimização. A revisão de algoritmos de processamento de documentos fiscais e verificação de assinaturas digitais, por exemplo, pode resultar em melhorias substanciais no desempenho.

Ao priorizar a otimização de desempenho, a biblioteca DF-e pode não apenas manter sua resiliência aprimorada, mas também oferecer uma experiência mais eficiente e ágil para os usuários, garantindo um serviço de alta qualidade em todos os aspectos.

4.1.2 Segurança e Privacidade

O foco em segurança e privacidade é necessário para consolidar a confiabilidade da biblioteca DF-e. Este campo concentra-se em aprimorar medidas de proteção, garantindo que os dados fiscais permaneçam íntegros, confidenciais e resistentes a ameaças.

- **Implementação de Criptografia Aprimorada**: Explorar algoritmos de criptografia mais robustos e modernos pode elevar o nível de proteção dos dados transmitidos e armazenados. A utilização de criptografia de ponta a ponta, especialmente em comunicações sensíveis, contribui para a segurança abrangente do sistema;

- **Auditoria e Rastreabilidade Aprimoradas**: Reforçar as capacidades de auditoria e rastreamento para monitorar atividades e acessos aos dados. A implementação de registros detalhados de auditoria pode ser vital para identificar e responder a possíveis violações de segurança, proporcionando transparência e responsabilidade;

- **Avaliação de Vulnerabilidades e Testes de Penetração**: Realizar avaliações regulares de vulnerabilidades e testes de penetração ajuda a identificar potenciais pontos fracos no sistema. Adotar uma abordagem proativa para corrigir e mitigar essas vulnerabilidades garante a resistência contínua contra ameaças cibernéticas;

- **Atualizações Contínuas de Segurança**: Manter-se atualizado com as melhores práticas de segurança e aplicar atualizações de segurança de forma regular é essencial. Isso inclui não apenas atualizações de software, mas também a implementação de correções de segurança recomendadas pelas autoridades fiscais e órgãos reguladores;

- **Controle de Acesso Granular**: Aperfeiçoar os controles de acesso, implementando políticas de permissões granulares. Garantir que apenas usuários autorizados tenham acesso a informações específicas, limitando o escopo de acesso conforme as funções e responsabilidades individuais.

Ao priorizar a segurança e a privacidade, a biblioteca DF-e pode assegurar não apenas a conformidade com regulamentações fiscais, mas também a confiança contínua dos usuários, garantindo que seus dados sensíveis estejam protegidos contra ameaças em constante evolução.

4.1.3 Integração de Novas Tecnologias

A integração de novas tecnologias é uma vertente estratégica para manter a biblioteca DF-e atualizada, adaptável e alinhada às mais recentes inovações tecnológicas. Essa abordagem visa potencializar a eficiência operacional, oferecer novas funcionalidades e proporcionar uma experiência aprimorada para os usuários finais.

- **Adoção de Tecnologias de Contêineres**: Explorar o uso de tecnologias de contêineres, como *Docker*, para facilitar a implantação, escalabilidade e gestão de componentes da biblioteca DF-e. Os contêineres oferecem um ambiente isolado e consistente, simplificando processos de desenvolvimento e atualizações;

- **Implementação de Microsserviços**: Avaliar a viabilidade e os benefícios da transição para uma arquitetura baseada em microsserviços. Essa abordagem descentralizada permite maior flexibilidade, escalabilidade independente de componentes e facilita a integração de novas funcionalidades de maneira mais ágil;

- **Exploração de Tecnologias de Inteligência Artificial (IA)**: Incorporar elementos de inteligência artificial, como aprendizado de máquina e processamento de linguagem natural, para aprimorar recursos analíticos, automatizar processos e fornecer *insights* valiosos a partir dos dados fiscais processados pela biblioteca DF-e;

- **Utilização de APIs Modernas**: Investir em APIs modernas e padronizadas para facilitar a integração com outros sistemas e serviços externos. Isso promove a interoperabilidade e possibilita a construção de ecossistemas mais amplos em torno da biblioteca DF-e;

- **Exploração de Tecnologias Emergentes**: Manter-se atualizado com tecnologias emergentes, como blockchain, para avaliar possíveis casos de uso que possam aprimorar a integridade e rastreabilidade dos documentos fiscais. A exploração de novas tecnologias abre portas para soluções mais eficientes e seguras.

Ao integrar novas tecnologias, a biblioteca DF-e não apenas se mantém relevante em um cenário tecnológico dinâmico, mas também abre oportunidades para impulsionar a inovação no âmbito fiscal, proporcionando benefícios tangíveis para os usuários e stakeholders envolvidos no ecossistema fiscal.

REFERÊNCIAS

BURNS, B. **Designing Distributed Systems**. 1. ed. Sebastopol: O'Reilly Media, 2018.

COSTA, A. F. **Receita Federal anuncia aumento de fiscalização de Notas Fiscais, e programas de regularidade fiscal**. 2023. Acesso em: 23 out. 2023. Disponível em: https://www.jusbrasil.com.br/noticias/receita-federal-anuncia-aumento-de-fiscalizacao-de-notas-fiscais-e-programas-de-regularidade-fiscal/123456.

CUSICK, J. J. **Exploring System Resiliency and Supporting Design Methods**. arXiv, 2020. Acesso em: 23 out. 2023. Disponível em: https://arxiv.org/abs/2009.05903.

EKUAN, M. **Retry pattern**. 2023. Acesso em: 23 out. 2023. Disponível em: https://learn.microsoft.com/en-us/azure/architecture/patterns/retry.

FIRESMITH, D. **System Resilience Part 2: How System Resilience Relates to Other Quality Attributes**. 2019. Acesso em: 23 out. 2023. Disponível em: https://insights.sei.cmu.edu/blog/system-resilience-part-2-how-system-resilience-relates-to-other-quality-attributes/.

FIRESMITH, D. **System Resilience: What Exactly is it?** 2019. Acesso em: 23 out. 2023. Disponível em: https://insights.sei.cmu.edu/blog/system-resilience-what-exactly-is-it/.

FIRESMITH, D. **System Resilience Part 5: Commonly-Used System Resilience Techniques**. 2020. Acesso em: 23 out. 2023. Disponível em: https://insights.sei.cmu.edu/blog/system-resilience-part-5-commonly-used-system-resilience-techniques/.

FOWLER, M. **Circuit Breaker**. 2014. Acesso em: 23 out. 2023. Disponível em: https://martinfowler.com/bliki/CircuitBreaker.html.

KHALEQ, A. A.; RA, I. Intelligent autoscaling of microservices in the cloud for real-time applications. **IEEE Access**, IEEE, Piscataway, v. 9, p. 35464–35476, 2021.

MENDONCA, N. C. *et al.* Model-based analysis of microservice resiliency patterns. In: **2020 IEEE International Conference on Software Architecture (ICSA)**. Piscataway: IEEE, 2020. p. 114–124.

MONTESI, F.; WEBER, J. **Circuit Breakers, Discovery, and API Gateways in Microservices**. arXiv, 2016. Acesso em: 23 out. 2023. Disponível em: https://arxiv.org/abs/1609.05830.

NYGARD, M. T. **Release It!: Design and Deploy Production-Ready Software**. 1. ed. Raleigh: Pragmatic Bookshelf, 2007.

SERULLE, N. U. **Transportation Network Resiliency: A Fuzzy Systems Approach**. Tese (Tese (Doutorado em Engenharia Civil)) — Utah State University, Logan, 2010. Disponível em: https://digitalcommons.usu.edu/etd/769.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Education, 2011.