UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Bruno de Lima Palhares

# Software como Serviço Multilocatário: Desenvolvimento, Migração e Avaliação

São João del-Rei

2021

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Bruno de Lima Palhares

# Software como Serviço Multilocatário: Desenvolvimento, Migração e Avaliação

Dissertação apresentada como requisito para obtenção do título de mestre em Ciências no Curso de Mestrado do Programa de Pós Graduação em Ciência da Computação da UFSJ.

Orientador: Elder José Reioli Cirilo

Universidade Federal de São João del-Rei – UFSJ

Mestrado em Ciência da Computação

São João del-Rei

2021

Ficha catalográfica elaborada pela Divisão de Biblioteca (DIBIB)
e Núcleo de Tecnologia da Informação (NTINF) da UFSJ,
com os dados fornecidos pelo(a) autor(a)

Palhares, Bruno de Lima .
P161s Software como Serviço Multilocatário:
Desenvolvimento, Migração e Avaliação / Bruno de Lima
Palhares ; orientador Elder José Reioli Cirilo. --
São João del-Rei, 2021.
86 p.

Dissertação (Mestrado - Ciência da Computação) --
Universidade Federal de São João del-Rei, 2021.

1. SaaS. 2. Multilocação. 3. SaaS Multilocatário .
I. Cirilo, Elder José Reioli , orient. II. Título.

Bruno de Lima Palhares

# Software como Serviço Multilocatário: Desenvolvimento, Migração e Avaliação

Dissertação apresentada como requisito para obtenção do título de mestre em Ciências no Curso de Mestrado do Programa de Pós Graduação em Ciência da Computação da UFSJ.

São João del-Rei, 05 de agosto de 2021:

**Elder José Reioli Cirilo**
Orientador
Universidade Federal de São João del Rei

---

**Davy de Medeiros Baia**
Membro Externo
Universidade Federal de Alagoas

---

**Teresinha Moreira de Magalhães**
Membro Externo
Instituto Federal Sudeste de Minas Gerais

---

**Flávio Luiz Schiavoni**
Membro Interno
Universidade Federal de São João del Rei

São João del-Rei
2021

# Agradecimentos

Em primeiro lugar gostaria de agradecer à Deus pelo dom da vida. Agradeço aos meus pais, César e Elza, meu irmão Vitor, minha esposa Aline e meu filho Augusto, por todo o apoio e amor ao longo dos anos. Agradeço especialmente ao meu orientador Elder Cirilo. Aos meus colegas da Universidade Federal de São João del Rei (UFSJ), obrigado por todo o apoio. Obrigado a todos que me ajudaram a desenvolver esse trabalho de alguma forma.

# Resumo

Software como Serviço (SaaS) é uma estratégia de mercado que fornece software empresarial escalonável como um serviço na Internet. Dentre os seus principais conceitos está a multilocação, ou seja, a habilidade que clientes, organizações e consumidores possuem de compartilhar infraestrutura e bancos de dados para aproveitar as vantagens de preço e desempenho que vêm quando se economiza escala. Neste trabalho realizou-se uma revisão da literatura para fornecer uma compreensão aprofundada de pesquisas que vêm sendo realizadas e suas tecnologias no âmbito do desenvolvimento, migração e evolução de SaaS multilocatário. Investigou-se 53 de 666 artigos recuperados e como resultado observou-se e comparou-se as principais propostas para o desenvolvimento e migração de aplicações SaaS. O trabalho também sintetiza as principais vantagens e desvantagens das aplicações Saas.

**Palavras-chaves**: Software como Serviços, Multilocatário, Desenvolvimento de Software, Banco de Dados

# Abstract

Software as a Service (Saas) is a market strategy to offer scalable enterprise software as an Internet service. Multi-tenancy, one of SaaS's central concepts, gives us the ability to share infrastructure and databases among clients, organizations, and consumers. In this work, we conducted a systematic literature review that aims to provide a deep understanding of the research realized involving the development, migration, and evaluation of Multi-tenant SaaS applications. We analyzed 53 (out of 666 papers), and as a result, we observed the leading development and migration techniques and approaches. We also synthesized the main advantages and disadvantages of SaaS applications.

**Key-words**: Software as Services, Multi-tenant, Software Development, Database

# Lista de ilustrações

# Lista de tabelas

# Sumário

# 1 Introdução

O SaaS é uma abordagem que fornece software empresarial escalonável como um serviço na Internet. Dentre os seus principais conceitos está a multilocação, ou seja, a habilidade que clientes, organizações e consumidores possuem de compartilhar infraestrutura e bancos de dados para aproveitar as vantagens de preço e desempenho que vêm quando se economiza escala. Os inquilinos, como são chamados os usuários, podem compartilhar o hardware no qual suas máquinas virtuais ou servidores são executados, ou podem compartilhar tabelas de bancos de dados onde os dados do cliente A estão em uma linha e os do cliente B em outra. Além do compartilhamento dos recursos de hardware a multilocação traz outras vantagens como, por exemplo, alta disponibilidade.

De fato, a multilocação acompanhada pela escalabilidade e a configurabilidade são os três principais atributos do SaaS que, de acordo com (HUR; KANG, 2011), podem ser alcançados através de um modelo de maturidade estruturado em quatro níveis. O nível um é o mais baixo dos níveis e requer um servidor dedicado e uma instância de serviço para cada inquilino. Nesse nível o custo de manutenção do provedor de serviços é alto, pois várias instâncias diferentes para diferentes inquilinos são exigidas. O atributo configurabilidade está disponível no nível dois que apesar de ainda exigir um servidor dedicado para cada inquilino, fornece um recurso para que eles possam configurar alguns aspectos do software SaaS de acordo com suas necessidades. Isso permite que instâncias idênticas possam ser utilizadas. O nível três, por sua vez, engloba o atributo multilocação que permite que o aplicativo SaaS atenda a vários inquilinos utilizando uma única instância de serviço. Isso significa que apesar de estarem compartilhando uma única instância de serviço e um único servidor, os inquilinos sentem como se estivessem utilizando um servidor dedicado. É nesse nível, portanto, que se pode, em termos de custo operacional, alcançar o benefício da economia de escala. O quarto e mais alto nível compreende o atributo escalabilidade que é adicionada através de uma arquitetura multicamadas com balanceamento de carga. A capacidade do sistema, portanto, pode ser aumentada ou diminuída por meio da adição ou remoção de servidores.

Neste trabalho apresentamos um levantamento da literatura para fornecer uma compreensão aprofundada da pesquisa que tem sido realizada sobre o SaaS e suas tecnologias de suporte a adoção desse modelo. Para selecionar e analisar os artigos nossa revisão da literatura seguiu um método sistemático. Primeiramente determinamos e executamos quatro strings de busca para consultar três bases de dados que foram escolhidas devido à relevância dos acervos para a engenharia de software (ACM Digital Library, IEEE Xplore Digital Library e Springer Link). O segundo passo foi a elaboração de uma estratégia de busca com base em três questões de pesquisa. O primeiro questionamento abordou os

métodos adotados pelos autores para apoiar a criação de aplicações multilocatárias. Já a segunda questão de pesquisa investigou as abordagens utilizadas pelos autores para realizar a migração de aplicações de inquilino único para aplicações multilocatárias. A terceira questão, por sua vez, procurou evidenciar as vantagens e desvantagens da utilização do recurso da multilocação.

Como resultado, 53 (de 666 artigos recuperados) foram incluídos e analisados em nosso trabalho. Consideramos que esse levantamento é nossa principal contribuição para os interessados nesse assunto. Além disso, consideramos os resultados do trabalho relevante, pois SaaSs são na realidade majoritariamente oferecidos pela Computação em Nuvem. Essa arquitetura computacional vem revolucionando a maneira de se entregar serviços de TI ao oferecer pela Internet os mais variados tipos de recursos como, por exemplo, servidores, armazenamento, bancos de dados, rede, software e diversos outros. A Computação em Nuvem, portanto, contribui imensamente na oferta de inovações mais rápidas e recursos mais flexíveis.

O restante dessa dissertação está organizado da seguinte forma. No capítulo 2 fornecemos um histórico do SaaS multilocatário, detalhando seu diferencial para o modelo de inquilino único, seus benefícios para os usuários e suas estratégias com relação ao armazenamento de dados. No capítulo 3 apresentamos a metodologia para realizar nossa revisão sistemática. No capítulo 4 sintetizamos os 36 artigos que descrevem as propostas para o desenvolvimento de aplicativos SaaS (sessão 4.1), bem como os 11 artigos que relatam a migração de aplicativos do modelo de inquilino único para o modelo SaaS multilocatário (sessão 4.2). No capítulo 5 estão sintetizados 6 artigos que discorrem acerca das vantagens e desvantagens dos aplicativos SaaS multilocatários. Finalmente, no capítulo 6, realizamos uma análise dos principais pontos observados na maioria dos artigos e nossas conclusões no capítulo 7.

# 2 Referencial Teórico

A computação em nuvem (CN), uma das tecnologias mais populares e amplamente utilizadas hoje, é um modelo de serviço que permite aos usuários acessar, de forma eficiente, através de uma plataforma de serviços hospedada na Internet, recursos como, por exemplo, softwares e armazenamento de dados.

Dentre as diversas modalidades compreendidas pela CN está o SaaS (Software as a Service) que consiste na disponibilização de serviços por parte de empresas (como a Netflix, Spotify, Uber, etc.) que cobram um valor mensal ou anual de seus clientes para utiliza-los. Os clientes, por sua vez, só pagam pelos serviços que, de fato, necessitam, já que a oferta dos mesmos ocorre sob demanda, ou seja, à medida que o usuário necessita de mais recursos, estes são prontamente disponibilizados (acarretando, evidentemente, aumento da mensalidade). Esta mudança na maneira de se distribuir um software representa um grande avanço tecnológico. Atualmente o usuário pode aproveitar toda uma gama de ferramentas apenas com um navegador de Internet. Diferente do que ocorria alguns anos atrás, quando para se utilizar um software era necessário adquirir uma licença do fabricante e instala-lo (geralmente através de um CD) no computador do usuário. Na visão de (YOUNAS, 2014), "em um futuro previsível, os aplicativos SaaS se integrarão com a Internet das Coisas, Computação Móvel, Big Data, Redes de Sensores Sem Fio e muitas outras tecnologias de computação e comunicação para fornecer serviços inteligentes e personalizáveis para uma vasta população".

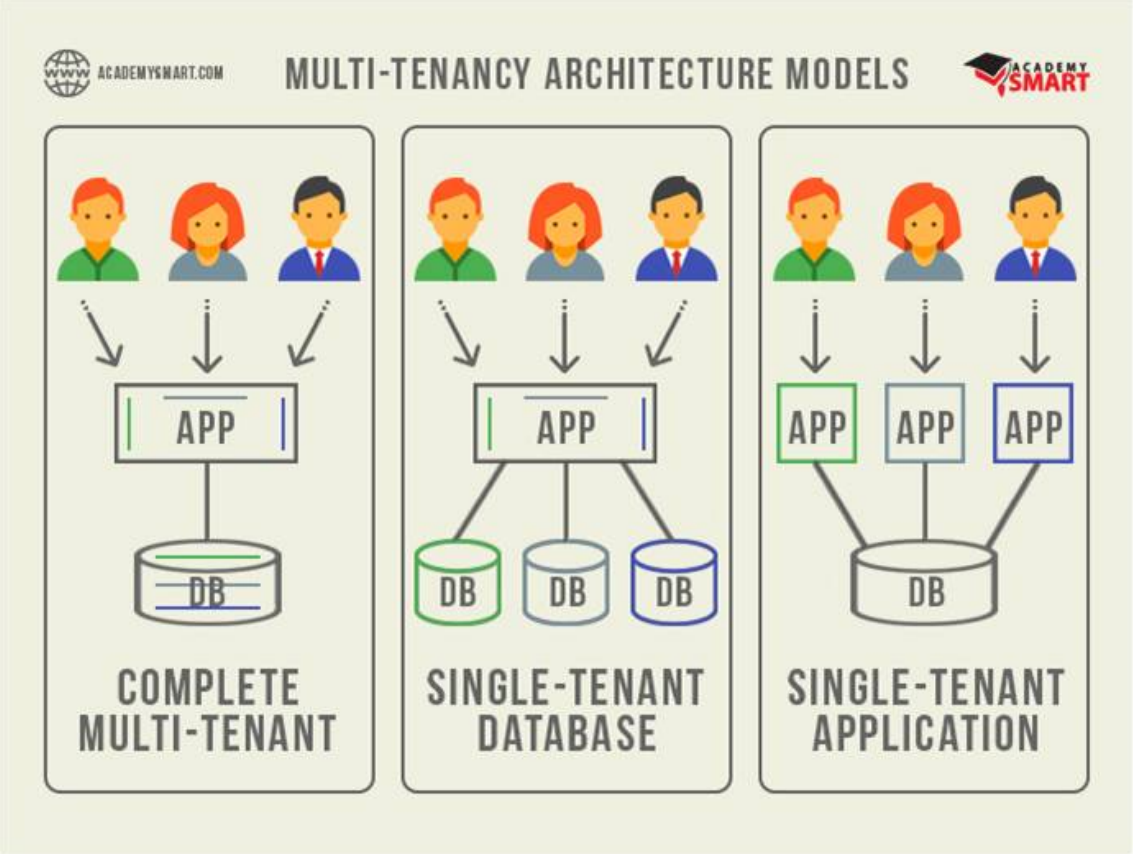


Figura 1 – Modelos de arquitetura multilocatária

Fonte: Academy Smart (2020)

Diferente do modelo single-tenant onde cada cliente (também chamado de inquilino) possui sua própria instância da aplicação e da base de dados, a arquitetura SaaS está geralmente atrelada ao modelo Multi-tenant onde segundo (BANKAR, 2013) os inquilinos consomem os serviços da mesma plataforma, compartilhando todos os componentes na pilha de tecnologia, incluindo o modelo de dados, servidores e camadas de banco de dados. Este compartilhamento de recursos se traduz em benefícios significativos para os inquilinos, incluindo: 1 - Escalabilidade: uma infraestrutura multilocatária facilita a ampliação da capacidade do sistema quando mais poder de processamento for exigido. O dimensionamento de uma plataforma SaaS multilocatária é, em muitos casos, simplesmente uma questão de se conectar mais hardware aos diferentes elementos contidos na pilha de tecnologia. Ao adicionar novo hardware à plataforma, a capacidade total de todo o ambiente aumenta, tornando-se mais escalonável não apenas para um único inquilino, mas para toda a gama de inquilinos. 2 - Proteção de dados: um arquiteto SaaS é responsável por construir a proteção de dados adequada, bem como vários níveis de defesa que se complementam para combater as ameaças internas e externas. A proteção de dados pode ser implementada por meio de listas de controle de acesso, criptografia, filtros ou firewalls. 3 - Infraestrutura compartilhada: a maioria das pessoas pensa que os servidores são os principais responsáveis pelo custo da infraestrutura, mas os custos relacionados

às licenças dos diversos softwares necessários para manter o sistema, como, por exemplo, bancos de dados, podem ser ainda mais altos. Levando-se ainda em conta as despesas com o pessoal de operações: DBAs, administradores de sistema, administradores de rede, etc., a conta final pode ficar realmente alta. Como em um sistema multilocatário toda a infraestrutura é compartilhada com dezenas, centenas ou até mesmo milhares de inquilinos, o provedor do sistema tem condições de cobrar um valor bem mais baixo pelo uso do mesmo. 4 - Compartilhamento de recursos de hardware: como já dito anteriormente, diferente do modelo single-tenant onde cada inquilino possui seu próprio servidor, no modelo multi-tenant todos os inquilinos compartilham os mesmos recursos de hardware. Uma das vantagens ao se colocar vários inquilinos no mesmo servidor é que a utilização do mesmo pode ser aprimorada. Embora isso também possa ser alcançado por meio da virtualização, a mesma impõe um limite muito menor no número de inquilinos por servidor devido aos altos requisitos de memória para cada servidor virtual. A maior utilização dos servidores existentes resultará em custos gerais mais baixos do aplicativo, pois a quantidade total de hardware necessária é menor. 5 - Alta disponibilidade: um elemento crítico da implantação do banco de dados em um ambiente SaaS multilocatário é a garantia da disponibilidade dos dados. Para tanto cada objeto de banco de dados deve ser replicado em tempo hábil. A replicação pode ser orientada a eventos ou programada. Uma desvantagem da abordagem de esquemas separados (mencionada no próximo parágrafo) é que em caso de alguma falha, os dados do inquilino são mais difíceis de serem restaurados, pois a restauração do banco de dados inteiro significaria substituir os dados de todos os inquilinos no mesmo banco de dados pelos dados de backup.

Com relação ao armazenamento de dados, (AULBACH; JACOBS, 2007), afirmam que o modelo Multi-tenant define duas estratégias: 1 - aplicação compartilhada, base de dados separada: todos os inquilinos utilizam a mesma instância do software, porém cada um deles possui sua própria base de dados separada. 2 - aplicação compartilhada, base de dados compartilhada: os inquilinos compartilham tanto a mesma instância do software quanto a mesma base de dados. Esta segunda estratégia ainda se subdivide em outras duas: 2.1 - base de dados compartilhada, esquemas separados: cada inquilino possui suas próprias tabelas de banco de dados separadas dentro da base de dados compartilhada. Esta técnica, apesar de exigir que um grande número de tabelas seja mantido, facilita a manutenção, pois caso algum inquilino necessite modificar algo em sua regra de negócios, o que certamente acarretaria modificações em sua base de dados, a atualização ocorreria apenas no esquema do inquilino em questão. 2.2: base de dados compartilhada, esquemas compartilhados: os inquilinos utilizam as mesmas tabelas de banco de dados dentro da base de dados compartilhada e são identificados por meio de um ID. Esta técnica é considerada a mais eficiente em termos de custo de manutenção e uso do hardware. Porém, a implementação se torna mais complexa, pois outras técnicas devem ser utilizadas para garantir que não haja perda de desempenho para os demais inquilinos quando, por exem-

plo, houver atualizações em uma determinada tabela, ou mesmo conjuntos de tabelas, devido às mudanças nas regras de negócios de um inquilino específico. Outro fator de complexidade no desenvolvimento dessa técnica é a questão da segurança dos dados uma vez que um inquilino de modo algum poderá ter acesso aos dados de outro.

# 3 Revisão Sistemática da Literatura

Uma revisão sistemática da literatura (RSL) visa identificar, avaliar e interpretar todas as pesquisas disponíveis sobre um tópico de interesse. Este tipo de revisão é caracterizado por uma especificação transparente do procedimento de estudo para minimizar vieses de pesquisa e apoiar a reprodutibilidade (DAVILA, 2020). Neste capítulo, descrevemos o protocolo de nossa RSL (Seção 3.1), seguindo as diretrizes propostas por Kitchenham e Charters (2007). Nós também fornecemos detalhes da execução do protocolo especificado (Seção 3.2) que nos permitiu identificar a literatura investigada em nossa RSL.

## 3.1 Protocolo de Revisão Sistemática

O objetivo deste estudo é identificar as principais pesquisas sobre Padrões de "Software as a Service"(SaaS) multilocatário. Dado o crescente interesse e a quantidade de pesquisas sobre o tema, pretendemos gerar uma visão coletiva dos resultados de diferentes estudos realizados e que tipo de suporte foi desenvolvido para o SaaS Multilocatário, identificando seus pontos fortes e limitações. Esse trabalho será útil para que futuros pesquisadores possam compreender o que já foi explorado e, deste modo, questionar. Também será de grande valor para que os praticantes aprendam como alcançar melhores resultados da tecnologia através das soluções existentes que podem ser adotadas. A seguir detalhamos nosso protocolo de revisão sistemática, descrevendo as estratégias para recuperar o máximo de contribuições literárias possíveis para atingir nosso objetivo de pesquisa.

### 3.1.1 Questões de pesquisa

Com base em nosso objetivo, nossa principal questão de pesquisa é: qual é o estado da arte da pesquisa sobre Padrões de SaaS multilocatário? Mais especificamente, pretendemos responder as seguintes questões de pesquisa (QP's):

- QP-1: Quais são os tipos de abordagens/métodos propostos para apoiar a criação de aplicações multilocatárias?
- QP-2: Quais são os tipos de abordagens/métodos propostos para apoiar a migração de aplicações multilocatárias?
- QP-3: Quais são as evidências das vantagens e desvantagens do uso de multilocação?

| **Base de dados** | **URL** | **#Trabalhos** |
|---|---|---|
| ACM Digital Library | portal.acm.org | 156 |
| IEEE Xplore Digital Library | ieeexplore.ieee.org | 234 |
| Springer Link | link.springer.com | 313 |
| Duplicados | | 37 |
| Total (incluindo duplicados) | | 703 |
| Total (excluindo duplicados) | | 666 |

Tabela 1 – Resultados das pesquisas por fonte.

### 3.1.2 Estratégia de Busca

A estratégia de busca define como recuperar os trabalhos primários de uma RSL com base em suas questões de pesquisa. Para encontrar os estudos relevantes para o nosso trabalho, selecionamos as três bases de dados apresentadas na Tabela 1. Essas bases de dados armazenam trabalhos publicados em muitas conferências e periódicos importantes de engenharia de software. Nossa pesquisa não inclui outras fontes, como Google Scholar e arXiv, porque fornecem indicações para trabalhos já armazenados nas bases de dados pesquisadas ou incluem trabalhos não revisados por pares. Nós nos concentramos em trabalhos que passaram por um processo de seleção por revisão por pares como forma de obter evidências da qualidade de suas pesquisas.

Para identificar o estado da arte dos padrões de SaaS multilocatário, especificamos quatro strings de busca para consultar as bases de dados selecionadas e recuperar os trabalhos. Na base de dados da IEEE Xplore, por exemplo, uma única string de busca foi utilizada para buscar os trabalhos tanto pelo título quanto pelo abstract. A base de dados da ACM, por sua vez, exigiu duas strings de busca separadas; uma para o título e outra para o abstract. Finalmente, na base da Springer Link, bastou a utilização de uma consulta menos complexa. As strings de busca para cada base de dados são as seguintes:

- IEEE Xplore: ((("Document Title":multi OR "Document Title":single) AND ("Document Title":tenancy OR "Document Title":tenant) AND ("Document Title":saas) AND ("Document Title":architecture OR "Document Title":database OR "Document Title":schema)) OR (("Abstract":multi OR "Abstract":single) AND ("Abstract":tenancy OR "Abstract":tenant) AND ("Abstract":saas) AND ("Abstract":architecture OR "Abstract":database OR "Abstract":schema)))

- ACM pelo título: [Publication Title: "multi"or "single"] AND [Publication Title: "tenancy"or "tenant"] AND [[Publication Title: "software as a service"] OR [Publication Title: or "service"or "saas"]] AND [Publication Title: "application"or "architecture"or "database"or "schema"or "table"]

- ACM pelo abstract: [Abstract: "multi"or "single"] AND [Abstract: "tenancy"or

"tenant"] AND [[Abstract: "software as a service"] OR [Abstract: or "service"or "saas"]] AND [Abstract: "application"or "architecture"or "database"or "schema"or "table"]

- Springer Link: '(multi OR single) AND (tenancy OR tenant) AND (software OR "service"OR "saas") AND ("database"OR "schema"OR "table")'

| **Critérios** | **#Trabalhos** |
|---|---|
| CI-1 - O trabalho apresenta propostas de desenvolvimento de aplicações multilocatárias? | 36 |
| CI-2 - O trabalho apresenta propostas de migração para aplicações multilocatárias? | 11 |
| CI-3 - O trabalho apresenta evidências das vantagens e/ou desvantagens do uso de multilocação? | 6 |
| CE - 1 Não são estudos primários | 0 |
| CE - 2 Não estão em inglês ou português | 0 |
| CE - 3 Não são revisados por pares | 0 |
| CE - 4 Usam bancos de dados / aplicativos multilocatários, mas não descrevem o método para desenvolver / criar bancos de dados / aplicativos multilocatários. | 154 |
| CE - 5 Usam qualquer tipo de ferramenta, mas não descrevem a ferramenta em si. | 0 |
| CE - 6 Não estão relacionados a bancos de dados / aplicativos multilocatários. | 459 |
| **Trabalhos selecionados** | **53** |

Tabela 2 – Critérios de inclusão (CI), critérios de exclusão (CE) e o número de trabalhos que os atendem.

### 3.1.3 Critérios de Seleção

Os trabalhos recuperados por meio de consultas aos bancos de dados de pesquisa foram filtrados usando critérios de seleção utilizados para identificar estudos primários relevantes para responder às nossas questões de pesquisa. Assim sendo, especificamos três critérios de inclusão (CI) e seis critérios de exclusão (CE), conforme resumido na Tabela 2.

Para ser selecionado, um estudo primário deve satisfazer pelo menos um critério de inclusão e nenhum critério de exclusão. Cada uma de nossas QP's tem um CI correspondente, o que leva a três tipos gerais de estudos: (i) PROPOSTAS DE DESENVOLVIMENTO de soluções que auxiliem a implementação de aplicações multilocatárias, como, por exemplo, uma ferramenta, técnica ou método (CI-1); (ii) PROPOSTAS DE MIGRAÇÃO que permitam analisar (através de uma ferramenta, técnica ou método) a viabilidade de se migrar uma aplicação single-tenant para uma aplicação multilocatária (CI-2); e (iii) EVIDENCIAS DAS VANTAGENS E/OU DESVANTAGENS que demons-

trem a possibilidade de se utilizar as técnicas de múltiplos inquilinos, ponderando suas vantagens e desvantagens (CI-3). Ao longo da dissertação, os termos destacados são usados para se referir a esses tipos de estudos.

Os critérios de exclusão CE-1 a CE-6 garantem que sejam selecionados apenas trabalhos escritos em Inglês ou em Português que sejam revisados por pares e que descrevam a ferramenta ou método utilizado para desenvolver ou criar aplicações/bancos de dados multilocatários.

## 3.2 Execução da Revisão

Nosso protocolo de revisão detalhado acima foi executado para recuperar os estudos primários investigados. Os resultados da nossa pesquisa e seleção são detalhados a seguir.

### 3.2.1 Execução da Pesquisa

Cada base de dados de destino possui uma sintaxe específica no mecanismo de pesquisa. Portanto, a string de busca foi personalizada para cada uma das bases de dados selecionadas. As pesquisas foram realizadas principalmente nos títulos e nos resumos das publicações. No entanto, devido a limitações na base de dados Springer Link, que não permite a realização de pesquisas nos resumos, formulamos uma string de busca menos complexa como já descrito na seção 3.1.2. A consulta nas três bases de dados selecionadas foi realizada em 14 de agosto de 2020. Os trabalhos recuperados e selecionados foram analisados ao longo de quatro meses. Como resultado, obtivemos seiscentos e sessenta e seis trabalhos, excluindo duplicatas.

### 3.2.2 Seleção dos trabalhos primários

Com base nos resultados da execução da pesquisa, trabalhos primários relevantes foram selecionados através da análise do título e do resumo de cada trabalho recuperado com o objetivo de encontrar qualquer evidência de que o trabalho, de fato, correspondesse a, pelo menos, um dos critérios de inclusão, mas nenhum dos critérios de exclusão. O trabalho poderia satisfazer a mais de um critério de inclusão ao mesmo tempo, desde que, como já dito, nenhum critério de exclusão fosse satisfeito. Havendo correspondência, o trabalho era então considerado válido. Dos seiscentos e sessenta e seis trabalhos recuperados inicialmente foram validados cinquenta e três após o processo de seleção. A tabela 3 apresenta todos os trabalhos selecionados para análise.

### 3.2.3 Método de Análise

Primeiramente foram analisados os títulos dos trabalhos. Se o título não contivesse termos como "SaaS", "aplicações/banco de dados multilocatário", o trabalho era descartado devido o critério E6. Caso contrário, analisamos o resumo para nos certificarmos de que o trabalho, de fato, apresenta alguma ferramenta, técnica ou método para desenvolver uma aplicação/banco de dados multilocatário ou para migrar para uma aplicação/banco de dados multilocatário. Não havendo como chegar a uma conclusão apenas pela análise do resumo, o trabalho era, então, analisado na íntegra e o mesmo descartado devido o critério E4 caso não apresentasse as características descritas anteriormente.

| **PROPOSTAS DE DESENVOLVIMENTO** |
|---|
| Kang et al. (2011), Aghera and Chaudhary and Kumar (2012), Khamis and Samir and Shahin (2013), Chauvel and Solberg (2018), Elkheir and Fouad and Saleh (2014), Leymann et al. (2009), Abu-Matar et el. (2014), Abu-Matar et al. (2017), Han et al. (2009), Damasceno and Leite and Rodrigues (2012), Gao et al. (2007), Huang et al. (2013), Cai et al. (2009), Abmann et al. (2012), Matsuo and Nishikawa and Oki (2012), Almorsy and Grundy and Ibrahim (2012), Younas and Yousef and Zhu (2014), Pinto et al. (2019), AZEEZ et al. (2010), Elkhatib andJumagaliyev and Whittle (2016), Chen et al. (2014), Joosen et al. (2015), Borger et al. (2015), Gey and Joosen and Landuyt (2015), Fan et al. (2011), Feuerlicht and Goyal and Yaish (2013), Kushwaha and Pippal (2013), Feuerlicht and Goyal and Yaish (2011), Goyal and Yaish (2013), Li and Xu and Zhao (2011), Joosen et al. (2017), Indrani and Radhakrishnan and Shiri (2013), Brodt et al. (2011), Wang et al. (2020), Chen et al. (2015), Meinel and Saleh and Takouna (2013) |
| **PROPOSTAS DE MIGRAÇÃO** |
| Cai and Wang and Zhou (2010), Hohenstein and Koka (2016), Bezemer and Zaidman (2010), Bezemer et al. (2010), Nugraheni (2013), Almorsy and Grundy and Ibrahim (2012), Chen et al. (2010), Sakamoto (2009), An et al. (2014), Du and Zheng (2014), Cao et al. (2012) |
| **EVIDENCIAS DAS VANTAGENS E/OU DESVANTAGENS** |
| Ojala (2013), An et al. (2008), Aulbach et al. (2009), Krebs and Momm (2011), Bhuvaneswari and Saraswathi (2014), Bayley et al. (2015) |

Tabela 3 – Trabalhos primários selecionados

Agora que introduzimos os trabalhos utilizados em nossa RSL e o método usado para analisá-los, passamos aos resultados. Os próximos dois capítulos apresentam os resultados de nossas análises.

# 4 Propostas de Desenvolvimento e Migração

Neste capítulo procuramos responder as seguintes questões de pesquisa abordadas no capítulo anterior: 1) quais são os tipos de abordagens/métodos propostos para apoiar a criação de aplicações multilocatárias? e 2) quais são os tipos de abordagens/métodos propostos para apoiar a migração de aplicações multilocatárias?

Dos trabalhos primários identificados, trinta e seis são classificados como PROPOSTAS DE DESENVOLVIMENTO. Estes trabalhos visam comparar as diversas abordagens existentes para apoiar a criação de aplicações e bancos de dados multilocatários. Outros onze trabalhos estão relacionados às PROPOSTAS DE MIGRAÇÃO do modelo de locatário único para o modelo multilocatário. Ambas as propostas serão analisadas tanto em relação às aplicações como em relação aos bancos de dados. A seguir, apresentamos uma visão geral dos diferentes tópicos investigados e, após uma análise aprofundada, discutimos o que foi descoberto.

## 4.1 Propostas de Desenvolvimento

Nós identificamos trinta e seis trabalhos que abordam o desenvolvimento de aplicativos SaaS multilocatários. Nove desses trabalhos apresentam soluções para a customização dos aplicativos (KANG et al., 2011; AGHERA; CHAUDHARY; KUMAR, 2012; KHAMIS; SAMIR; SHAHIN, 2013; CHAUVEL; SOLBERG, 2018; ELKHEIR; FOUAD; SALEH, 2014; LEYMANN et al., 2009; ABU-MATAR et al., 2014; ABU-MATAR et al., 2017; HAN et al., 2009). Cinco trabalhos relatam as propostas de arquiteturas para o desenvolvimento dos aplicativos (DAMASCENO; LEITE; RODRIGUES, 2012; GAO et al., 2007; HUANG et al., 2013; CAI et al., 2009; ABMANN et al., 2012). Dois trabalhos descrevem soluções voltadas para a segurança dos aplicativos (MATSUO; NISHIKAWA; OKI, 2012; ALMORSY; GRUNDY; IBRAHIM, 2012). Outros dois trabalhos propõem soluções para a recuperação de falhas dos aplicativos (YOUNAS; YOUSEF; ZHU, 2014; PINTO et al., 2019). Um trabalho apresenta uma arquitetura para executar aplicativos atuais em um ambiente multilocatário (AZEEZ et al., 2010). Outro trabalho propõe o gerenciamento de recursos SaaS através de técnicas de modelagem (ELKHATIB; JUMAGALIYEV; WHITTLE, 2016). Finalmente quatro trabalhos apresentam o desenvolvimento de Middlewares voltados para: (1) desenvolvimento e implementação de aplicativos SaaS corporativos (CHEN et al., 2014); (2) gerenciamento de dados orientado por políticas (JOOSEN et al., 2015); (3) isolamento de desempenho (BORGER et al., 2015); (4) atualização de aplicativos (GEY; JOOSEN; LANDUYT, 2015).

Também identificamos doze trabalhos que se dedicam ao desenvolvimento de ban-

cos de dados multilocatários (FAN et al., 2011; FEUERLICHT; GOYAL; YAISH, 2013; KUSHWAHA; PIPPAL, 2013; FEUERLICHT; GOYAL; YAISH, 2011; GOYAL; YAISH, 2013; LI; XU, ZHAO, 2011; JOOSEN et al., 2017; INDRANI; RADHAKRISHNAN; SHIRI, 2013; BRODT ET AL., 2011; WANG et al., 2020; CHEN ET AL., 2015; MEINEL; SALEH; TAKOUNA, 2013).

| **Tipo de pesquisa** | **Trabalhos de referência** | **#Trabalhos** |
|---|---|---|
| Customização | Kang et al., 2011; Aghera; Chaudhary; Kumar, 2012; Khamis; Samir; Shahin, 2013; Chauvel; Solberg, 2018; Elkheir; Fouad; Saleh, 2014; Leymann et al., 2009; Abu-Matar et el., 2014; Abu-Matar et al., 2017; Han et al., 2009 | 9 |
| Arquitetura | Damasceno; Leite; Rodrigues, 2012; Gao et al., 2007; Huang et al., 2013; Cai et al., 2009; Abmann et al., 2012 | 5 |
| Segurança | Matsuo; Nishikawa; Oki, 2012; Almorsy; Grundy; Ibrahim, 2012 | 2 |
| Recuperação de falhas | Younas; Yousef; Zhu, 2014; Pinto et al., 2019 | 2 |
| Execução de aplicativos atuais em ambiente multilocatário | AZEEZ et al., 2010 | 1 |
| Gerenciamento de recursos | Elkhatib; Jumagaliyev; Whittle, 2016 | 1 |
| Middleware | Chen et al., 2014; Joosen et al., 2015; Borger et al., 2015; Gey; Joosen; Landuyt, 2015 | 4 |
| Banco de dados | Fan et al., 2011; Feuerlicht; Goyal; Yaish, 2013; Kushwaha; Pippal, 2013; Feuerlicht; Goyal; Yaish, 2011; Goyal; Yaish, 2013; Li; Xu, Zhao, 2011; Joosen et al., 2017; Indrani; Radhakrishnan; Shiri, 2013; Brodt et al., 2011; Wang et al., 2020; Chen et al., 2015; Meinel; Saleh; Takouna, 2013 | 12 |

Tabela 4 – Propostas de desenvolvimento: Autores e a quantidade de trabalhos realizados por assunto

### 4.1.1 Customização

Embora a maioria das empresas utilize um aplicativo SaaS padronizado e implementado por um desenvolvedor SaaS, o trabalho produzido por (HUR; KANG, 2011), apesar de apenas conceitual, visa o futuro desenvolvimento de aplicativos SaaS multilocatários e, principalmente, configuráveis.

O objetivo do trabalho é permitir que cada um dos inquilinos atrelados ao sistema possa, de acordo com seus próprios requisitos, configurar quatro aspectos importantes do software SaaS: 1) estrutura organizacional: segundo os autores, ser capaz de configurar as permissões do sistema é uma característica essencial para um aplicativo SaaS, pois trata-se de um aspecto muito exclusivo dos inquilinos que pode ser alterado com frequência e é, praticamente, impossível considerá-las no estágio de desenvolvimento. Nem todo usuário pode ter acesso a todas as funcionalidades do sistema. Dados sigilosos, por exemplo, podem ser acessados por executivos, mas não por funcionários. O processo da folha de pagamento não pode ser acessado por um membro do departamento de suporte. Dados que não sejam abertos aos fornecedores não podem ser visualizados na página do fornecedor; 2) interface do usuário (IU): os autores afirmam que é de extrema importância que a IU seja de fácil utilização por parte dos inquilinos. Por essa razão a configuração da IU visa alterar a aparência e o comportamento da mesma. As cores dos componentes das páginas podem ser alteradas, os layouts reestruturados, hiperlinks e componentes adicionados ou removidos, uma nova coluna deve ser adicionada a um componente de IU de grade para mostrar os campos de um objeto de dados caso um novo campo seja adicionado ao objeto. Da mesma forma, se qualquer objeto de dados for excluído do banco de dados, o componente deverá refletir essa mudança; 3) modelo de dados: essa configuração, de acordo com os autores, significa a capacidade de adicionar ou excluir objetos de dados, adicionar ou excluir campos de dados em objetos de dados existentes e alterar o nome do campo, tipo e assim por diante. O inquilino pode acessar o SGBD através da plataforma e gerenciar o modelo de dados do software SaaS; 4) fluxo de trabalho: como diferentes organizações possuem estruturas organizacionais, processos de tomada de decisão ou regras de negócios exclusivos, os fluxos de trabalho devem ser configurados para cada inquilino. Os autores apresentam uma ferramenta chamada Process Designer para que o inquilino possa configurar vários componentes do processo de negócios em termos de fluxo de trabalho, tipo de atividade e regras de negócios.

O recurso de múltiplos inquilinos, como afirmam os autores, exige que a arquitetura da plataforma permita a autoconfiguração por parte de um inquilino individual sem que haja alteração do código-fonte do aplicativo SaaS para os demais inquilinos e, além disso, garanta a não paralisação do serviço durante a configuração em tempo de execução. Para implementar essas características os autores utilizaram uma arquitetura orientada a metadados. Neste tipo de arquitetura, como explicam os autores, todos os

aspectos do aplicativo SaaS que podem ser configurados pelos inquilinos são armazenados no banco de dados de metadados. Quando um inquilino configura o aplicativo, as informações configuradas são armazenadas separadamente nos metadados específicos do inquilino. O mecanismo de tempo de execução gera um aplicativo polimórfico para cada inquilino individual usando a base de código do aplicativo e os metadados específicos do inquilino. Através desse aplicativo polimórfico, os inquilinos têm a percepção de que estão utilizando seu próprio aplicativo de negócios, enquanto a instância de serviço é compartilhada por todos eles.

Com relação à segurança dos dados os autores garantem que apesar do compartilhamento dos mesmos, o aplicativo polimórfico acessa os dados de forma independente por meio de uma consulta otimizada para cada inquilino individualmente.

A plataforma SaaS proposta pelos autores é composta por quatro componentes principais: 1) aplicativo cliente: por meio de um navegador web os inquilinos são capazes de utilizar os serviços SaaS disponibilizados pela plataforma; 2) configurador: O Configurador é um aplicativo web utilizado pelo inquilino para configurar, através de uma interface gráfica com o usuário, os quatro principais aspectos do aplicativo SaaS mencionados no início do parágrafo; 3) mecanismo de tempo de execução: como já explicado anteriormente, os aspectos do aplicativo SaaS configurados pelos inquilinos são armazenados como metadados no banco de dados de metadados, enquanto a base de código implementada pelo desenvolvedor é armazenada no banco de dados da aplicação juntamente com os dados gerados pelo usuário. O mecanismo de tempo de execução é responsável por gerar os aplicativos específicos para cada inquilino, de acordo com as configurações feitas por cada um deles, por meio da utilização da base de código e dos metadados; 4) sistema de gerenciamento de metadados: responsável por oferecer suporte a multilocação, o sistema converte a solicitação feita pelo navegador web em uma consulta otimizada ao banco de dados para recuperar páginas e dados específicos de cada inquilino.

Os autores (AGHERA, 2012) afirmam que para atingir um modelo de maturidade mais elevado, a arquitetura SaaS requer um certo nível de configuração e personalização. Baseados nisso os autores propõem uma arquitetura orientada por metadados que permite ao inquilino monitorar, gerenciar, administrar e configurar o software, além de serviços de gerenciamento de grandes volumes de dados. Embora haja diversas maneiras de se desenvolver e implantar um aplicativo SaaS com base no ambiente de hospedagem (Salesforce.com, Google AppEngine, Microsoft Azure, etc.), na arquitetura a ser seguida e nas tecnologias de desenvolvimento disponíveis (Java, .NET), o conceito proposto pelos autores é de uma plataforma SaaS independente da infraestrutura subjacente para que o aplicativo possa ser migrado de uma nuvem para outra sem que para isso seja necessário a alteração do código.

A arquitetura proposta foi dividida em três partes: bancos de dados, fluxo ló-

gico e pooling da aplicação. A primeira parte (bancos de dados) se subdivide em outras quatro: banco de dados de metadados, banco de dados da aplicação, banco de dados de monitoramento e interface de serviço da web.

Na arquitetura orientada por metadados, o banco de dados exerce uma função dupla. No banco de dados de metadados ficam armazenadas as informações de configuração da aplicação com base nos requisitos específicos de cada inquilino, ou seja, dados sobre dados. O banco de dados da aplicação, por sua vez, armazena os dados produzidos pelos usuários. Um terceiro banco de dados, chamado pelos autores de banco de dados de monitoramento, mantém os logs de acesso ao sistema e os registros de SLA (Service Level Agreement). O SLA define um contrato formal entre o provedor SaaS e o consumidor, incluindo critérios relacionados à segurança de dados, privacidade, propriedade, disponibilidade, nível de desempenho, backup de dados, recuperação de falhas, entre outros. Finalmente o serviço web é fornecido como uma interface de acesso aos bancos de dados com o intuito de ocultar o acesso direto aos mesmos. A interface dispõe das principais transações para a manipulação de dados, como inserção, atualização e exclusão. O ID do inquilino é utilizado como parâmetro para acessar os dados.

Uma das principais atribuições do sistema de fluxo lógico da aplicação é realizar o processamento da URL do navegador web com o objetivo de extrair dela o ID do inquilino, através do qual o sistema realizará o controle de acesso, acessará todas as configurações de metadados e exibirá dinamicamente a página customizada do inquilino requisitante.

A tarefa do pooling da aplicação (última parte da arquitetura) é, essencialmente, implementar o isolamento de processos visando o bom desempenho da aplicação caso um grande número de inquilinos acesse o sistema ao mesmo tempo. O pooling também exerce o gerenciamento de prioridades do sistema uma vez que alguns inquilinos requerem maior prioridade do que outros.

Apesar de não implementarem a arquitetura por eles proposta os autores garantem que conseguiram mostrar uma maneira mais fácil de desenvolver aplicativos SaaS sem que seja necessário a utilização de frameworks proprietários. Além disso, afirmam que todos os aspectos do desenvolvimento de SaaS foram devidamente abordados, como: configuração, personalização, gerenciamento de inquilinos, especificação de SLA e monitoramento com nível de desempenho satisfatório.

Segundo (KHAMIS, 2013) os aplicativos SaaS devem ser customizáveis para atender aos diversos requisitos funcionais dos inquilinos individuais. Ainda segundo os autores, os elementos de um aplicativo que precisam ser customizados incluem: Interface Gráfica do Usuário (GUI), fluxo de trabalho (lógica do processo de negócios), serviço e dados. Os autores, no entanto, se preocupam apenas com a customização do fluxo de trabalho e das camadas de serviço.

Uma das abordagens mais notáveis para alcançar a customização específica do inquilino, de acordo com os autores, é fornecer um modelo de aplicativo com partes não especificadas que podem ser customizadas selecionando componentes de um conjunto de componentes fornecidos. Essas partes não especificadas são chamadas de pontos de customização de um aplicativo.

Os autores se deparam com quatro importantes questões que precisam ser consideradas: 1) como modelar os pontos de customização e variações?; 2) como descrever as relações entre as variações?; 3) como validar customizações realizadas pelos locatários?; 4) como associar e desassociar variações dos / para pontos de customização durante o tempo de execução?

Para solucionar as questões acima mencionadas os autores apresentam o Orthogonal Variability Model (OVM) and Metagraphs. Uma abordagem orientada a aspectos para personalizar aplicativos SaaS. O OVM é usado para modelar pontos de customização e variações e para descrever os relacionamentos entre variações. Um algoritmo baseado em Metagraph foi desenvolvido para validar as personalizações dos inquilinos. Finalmente, uma extensão orientada a aspectos para Business Process Execution Language (AO4BPEL) é usada para associar e desassociar variações dos / para pontos de customização durante o tempo de execução.

Para demonstrar a abordagem os autores utilizam um aplicativo simplificado de uma agência de viagens que inclui apenas as atividades de reserva e pagamento. Essas atividades são, na verdade, dois locais personalizáveis. Cada local personalizável no fluxo de trabalho do aplicativo pode ser substituído por dois subfluxos de trabalho. O cenário então é o seguinte: a reserva pode ser feita online ou presencial. O pagamento, por sua vez, pode ser feito através de cartão de crédito ou em dinheiro.

Os autores (CHAUVEL; SOLBERG, 2013) descrevem uma técnica para personalização profunda de aplicativos SaaS multilocatários utilizando microsserviços intrusivos. Essa técnica consiste em implementar o código personalizado como um microsserviço isolado e autocontido em execução ao lado do serviço principal e utilizar o código de retorno de chamada para executar consultas ou comandos intrusivamente dentro do serviço principal. Um aplicativo de compras online de código aberto chamado MusicStore é utilizado pelos autores para demonstrar a personalização profunda em aplicativos SaaS multilocatários. O aplicativo MusicStore simula as funcionalidades essenciais das compras online, como gerenciamento de usuários, catálogo, carrinho de compras e caixa, utilizando álbuns de música como produtos de amostra.

Após implementarem as modificações no aplicativo MusicStore, os autores concluem que a personalização profunda é realmente viável para aplicativos SaaS multilocatários. Duas desvantagens, no entanto, devem ser observadas: 1) o código personalizado está fortemente acoplado ao serviço principal e, portanto, as atualizações do serviço prin-

cipal podem quebrar algum código personalizado; 2) a personalização profunda em teoria permite que os clientes façam qualquer alteração no serviço compartilhado, o que pode causar graves problemas de segurança.

Os autores admitem que a abordagem proposta por eles não resolve os dois problemas mencionados acima e, por isso, pretendem gerar casos de teste para verificar a compatibilidade do código personalizado após as atualizações do serviço principal, analisar o código estático no serviço principal e o código personalizado para fins de segurança.

Os autores (SALEH, 2014) se preocuparam em formular, analisar, classificar e priorizar os requisitos de software em ambientes de aplicativos multilocatários. Eles propuseram uma nova classificação para os requisitos de software de multilocação dependendo do volume da demanda e da natureza da variação entre esses requisitos e forneceram diretrizes para que os arquitetos e os desenvolvedores sejam capazes de selecionar apropriadamente tanto a arquitetura de software quanto as técnicas de realização de variabilidade em cada classe de requisitos.

A variabilidade do software, como explicam os autores, se refere à capacidade do software de ser estendido, alterado, customizado ou configurado de forma eficiente para uso em um determinado contexto e se divide em duas partes: 1) Configurabilidade: é a capacidade de um sistema de ser alterado dentro de um escopo predefinido. Por exemplo, por meio de assistentes e definições de configuração; 2) Personalização: é a capacidade de um sistema de ser alterado adicionando código personalizado para estender o aplicativo com funcionalidade personalizada.

A seleção de uma solução de variabilidade, de acordo com os autores, depende da análise dos requisitos dos inquilinos do sistema. Essa análise segue uma série de passos descritos a seguir: 1) Definição dos requisitos específicos de domínio: são os requisitos comuns ou compartilhados que devem ser alcançados por todos os inquilinos que assinam o software multilocatário. Os requisitos definem as metas e objetivos comuns do negócio no nível do domínio; 2) Definição dos Requisitos de Segmento: um grupo de requisitos que são de interesse de um grupo de inquilinos semelhantes no mesmo segmento de mercado ou um conjunto de metas ou objetivos compartilhados de um determinado grupo de inquilinos; 3) Definição dos requisitos do inquilino: um conjunto de requisitos que são de interesse de um inquilino exclusivo ou um conjunto de inquilinos; 4) Definição dos Requisitos do Usuário: os requisitos de layout pessoal do usuário final. Esses requisitos podem variar entre os usuários finais na mesma organização, de acordo com seu estilo individual.

Nenhuma avaliação da técnica proposta pelos autores foi apresentada por eles, mas os mesmos esperam ter conseguido, através da análise da variação nos requisitos de vários inquilinos, implementar técnicas de realização de variabilidade dependendo dos níveis de tais requisitos.

Os autores (MIETZNER, 2009) descrevem como as técnicas de modelagem de variabilidade, bem conhecidas na Engenharia de Linha de Produto de Software (ELPS), podem ser usadas para modelar a variabilidade em aplicativos SaaS e aplicam os conceitos de variabilidade externa e interna ao problema da personalização e suporte de implantação para aplicativos SaaS.

A proposta dos autores, como já mencionado, é modelar a variabilidade de aplicações SaaS explorando as técnicas de ELPS. Na ELPS dois tipos de variabilidade se destacam: 1) variabilidade externa: é a variabilidade que é comunicada ao cliente da linha de produtos; 2) variabilidade interna: só é visível para os desenvolvedores da linha de produtos. Os autores, portanto, seguiram essa distinção ao construir os modelos de variabilidade SaaS e foram capazes de raciocinar sobre esses modelos a fim de apoiar o processo de decisão para construir, customizar e implantar aplicativos SaaS.

O autores implementaram um protótipo baseado no tempo de execução da arquitetura de componente de serviço (Service Component Architecture - SCA) Apache Tuscany3 que, como explicam os autores, usa um registro que captura informações sobre partes já implantadas do aplicativo SaaS. A infraestrutura de provisionamento então consulta esse registro para decidir se um serviço precisa ser implantado ou não ou se é suficiente para implantar dados de configuração.

Devido ao fato dos aplicativos SaaS multilocatários convencionais fornecerem o mesmo conjunto de serviços para todos os inquilinos resultando, dessa forma, em aplicativos únicos, os autores (ABU-MATAR, 2014) observam que soluções de SaaS personalizáveis são necessárias, uma vez que os inquilinos podem ter requisitos diferentes. Além disso, observam também que os requisitos dos inquilinos estão em constante evolução e, por essa razão, a instância SaaS deve evoluir sistematicamente.

Pelos motivos apresentados acima os autores apresentam uma plataforma de evolução SaaS multilocatária de instância única com base em linhas de produtos de software. A plataforma, como explicam os autores, especifica um conjunto de regras de evolução com base na modelagem de recursos que governam as decisões de evolução.

A arquitetura proposta pelos autores fornece uma solução orientada a recursos que permite aos provedores de serviços gerenciar a variabilidade em um nível mais alto de abstração e especifica diferentes cenários que podem levar a uma evolução de uma instância SaaS. São eles: 1) instanciando um kernel SaaS: neste cenário ocorre a geração de um modelo independente de plataforma do kernel que consiste nos recursos principais compartilhados entre todos os inquilinos; 2) inquilinos integrados: um novo inquilino inicia uma solicitação para assinar os serviços SaaS. Supondo que as funcionalidades a serem solicitadas pelo inquilino sejam parte do modelo de recurso existente, o inquilino começa selecionando o recurso desejado no modelo de recurso; 3) personalização de inquilinos existentes: como os serviços SaaS são acessados com base no pagamento conforme o uso,

os inquilinos existentes podem solicitar a personalização de suas próprias configurações de recursos com base em seus requisitos mutáveis ao longo do tempo. Por meio de uma fase de nova seleção de recursos, um inquilino seleciona o conjunto de recursos a ser personalizado. O inquilino pode solicitar adicionar ou remover recursos, alterar mapeamentos de recursos ou alterar o modelo de dados do recurso; 4) removendo locatários: como sugere o modelo de custo dos aplicativos SaaS, os inquilinos podem assinar ou cancelar a assinatura com base em suas necessidades. Em um determinado momento o inquilino pode decidir cancelar a assinatura de todos os serviços que solicitou anteriormente. Dependendo da configuração do inquilino que estiver saindo uma evolução SaaS pode ou não ser necessária para remover o inquilino.

As necessidades evolutivas dos inquilinos dos aplicativos SaaS multilocatários exigem que a instância SaaS se adapte dinamicamente. Por essa razão os autores (ABU-MATAR, 2017) apresentam uma plataforma integrada que facilita a adaptação dinâmica de aplicativos SaaS multilicatários e oferece suporte à personalização de configurações dos inquilinos em tempo de execução. Os autores basearam a plataforma em três conceitos: Orientação a Serviços, Linhas de Produtos de Software e Arquitetura Orientada a Modelos. Além disso, a solução proposta abrange duas dimensões: 1) gerenciamento de variabilidade de nível de recurso: essa fase tem por objetivo empregar um modelo de recurso de um aplicativo SaaS para derivar configurações específicas do inquilino, modelar composições de serviço e construir a arquitetura da instância que será usada durante a adaptação do tempo de execução; 2) gerenciamento de variabilidade de tempo de execução: essa fase visa gerenciar a adaptação da instância SaaS em tempo de execução de acordo com as solicitações de adaptação. Isso exige o monitoramento das solicitações de entrada para detectar as adaptações desejadas instantaneamente e implementá-las de acordo e com cuidado para evitar inconsistências de tempo de execução.

Para a realização dos testes do protótipo os autores utilizaram 24 serviços de um estudo de caso e simularam a integração, customização e remoção de inquilinos com diferentes configurações. Ao invés de utilizar apenas 10 inquilinos para avaliar a abordagem proposta, como ocorreu na maioria das pesquisas semelhantes, os autores utilizaram 50 inquilinos, pois na visão dos mesmos esse número é considerado mais razoável para ser suportado por provedores SaaS reais.

Os resultados da avaliação, de acordo com os autores, demonstraram que os pedidos de adaptação dos inquilinos afetam apenas os inquilinos solicitantes. No entanto, a escalabilidade da plataforma pode ser afetada pelo design do serviço. Os serviços precisam ser projetados da maneira mais granulada possível, sem afetar a flexibilidade do aplicativo para minimizar o número de serviços necessários e maximizar seu efeito.

Os autores (HAN, 2009) reconhecem que a capacidade de fornecer diferentes níveis de desempenho com base em contratos de nível de serviço (Service Level Agreements -

SLAs) específicos para inquilinos é um requisito fundamental para aplicativos multilocatários, porém um objetivo difícil de ser atingido devido as variações da carga de trabalho e o amplo compartilhamento de recursos entre os inquilinos.

O problema supracitado é abordado pelos autores através de um regulador de desempenho com base no controle de feedback. O regulador tem uma estrutura hierárquica com a qual um controlador de alto nível gerencia as taxas de admissão de solicitação para evitar sobrecarga e um controlador de baixo nível gerencia a alocação de recursos para solicitações admitidas para rastrear um nível específico de diferenciação de serviço entre os inquilinos.

O objetivo do alto nível do regulador de desempenho, de acordo com os autores, é garantir a aderência dos objetivos de desempenho para cada inquilino com base na alocação de recursos no nível do aplicativo e algoritmos de controle cuidadosamente projetados. Especificamente os objetivos do projeto do regulador são os seguintes: 1) tempo médio de resposta garantido: as prioridades do pool de threads são ajustadas dinamicamente de acordo com as variações da carga de trabalho com o objetivo de manter o tempo de resposta em um valor especificado para cada inquilino; 2) alta utilização da CPU: o controlador baseado em feedback continua tentando aumentar a taxa de admissão até que os objetivos de tempo de resposta sejam alcançados; 3) políticas flexíveis de resolução de conflitos: as metas de desempenho de diferentes inquilinos podem entrar em conflito entre si especialmente durante a sobrecarga do sistema. Dependendo dos requisitos de negócios as políticas de resolução de conflitos possíveis podem incluir: garantir menos solicitações descartadas de um inquilino do que dos outros durante a sobrecarga, garantir um determinado resultado final de rendimento para um inquilino específico e assim por diante.

Para demonstrar a eficácia do regulador de desempenho os autores utilizaram três máquinas. Uma para o servidor Web e de aplicativos. Outra para o servidor de banco de dados e outra máquina cliente que foi utilizada para conduzir o sistema. Após a realização dos experimentos os autores concluíram que apesar das variações da carga de trabalho, tanto a prevenção de sobrecarga quanto a diferenciação de serviço foram alcançadas com sobrecarga insignificante.

### 4.1.2 Arquitetura

O trabalho produzido por (DAMASCENO, 2012) tem por objetivo prestar auxílio aos desenvolvedores e engenheiros de software no desenvolvimento de aplicações SaaS multilocatário. Para isso os autores descrevem, implementam, implantam e apresentam os resultados da arquitetura por eles proposta.

Havia um problema a ser solucionado: o desenvolvimento de um aplicativo para

fornecer suporte ao desenvolvimento de LPS (Linhas de Produtos de Software), pois alguns estagiários de uma empresa precisavam definir e documentar várias LPS para diferentes empresas.

Para resolver o problema acima mencionado os autores partiram para a escolha da arquitetura a ser utilizada e das tecnologias capazes de desenvolvê-la. Os principais componentes da arquitetura definida são: 1) autenticação: uma única instância da aplicação multiusuário, bem como do banco de dados é compartilhada por todos os inquilinos do sistema. Por essa razão, na visão dos autores, a autenticação é de extrema importância para garantir que cada inquilino somente tenha acesso a seus próprios dados, além de fornecer aos inquilinos um ambiente customizado; 2) configuração: a configuração do sistema deve garantir que o inquilino tenha a percepção de que está trabalhando em seu próprio ambiente personalizado. Para que isso seja possível os autores afirmam que os seguintes tipos de configuração devem ser disponibilizados pelo sistema: estilo de Layout, configurações gerais definidas pelo usuário, configurações de localização de arquivos e configurações de fluxos de trabalho; 3) banco de dados: o isolamento dos dados é crucial para uma aplicação multiusuário, pois, como já dito, todos os inquilinos compartilham a mesma instância do banco de dados. É preciso se certificar de que os inquilinos tenham acesso somente a seus próprios dados. Os autores dizem que os Sistemas de Gerenciamento de Banco de Dados (SGBD) atuais não possuem esse recurso e que uma nova camada deve ser criada para implementar a criação de novos inquilinos no banco de dados, a execução de consultas no SGBD e o balanceamento de carga.

Para o desenvolvimento da arquitetura os autores optaram pelas tecnologias da plataforma Java: JSF, Struts 2, Spring Framework e Grails. A escolha se deu principalmente por causa do framework Grails que fornece uma arquitetura baseada em plugins. Um repositório, mantido pela comunidade dos desenvolvedores, fornece plugins que implementam várias funcionalidades como, por exemplo, testes, buscas, geração de relatórios, segurança, etc. Isso permite que funcionalidades extremamente complexas do ponto de vista da implementação possam simplesmente ser adicionadas a aplicação de uma maneira bastante simples. Esse foi o caso, por exemplo, do plugin Multi-Tenant Core que implementou dois dos componentes da arquitetura: acesso ao banco de dados e autenticação. O plugin associa um usuário da aplicação a um inquilino específico e filtra os dados provenientes de uma consulta ao banco de dados para garantir que o usuário somente possa ter acesso aos dados vinculados ao seu inquilino. A autenticação foi fornecida através da integração do plugin com um framework de controle de acesso chamado Spring Security que, segundo os autores, é bastante utilizado em aplicações Java. Com relação ao terceiro componente da arquitetura (configuração), este foi implementado por meio de outro plugin, pertencente ao repositório Grails, chamado Dynamic Domain Class.

Antes da implementação da aplicação real os autores avaliaram um protótipo con-

tendo uma pequena parte dos requisitos funcionais da aplicação. Uma tabela com doze atributos de qualidade foi criada para servir de parâmetro durante a avaliação do protótipo. Os atributos são os seguintes: disponibilidade, integridade conceitual, flexibilidade, interoperabilidade, manutenibilidade, gerenciabilidade, performance, reusabilidade, escalabilidade, segurança, testabilidade e usabilidade.

Os autores, após a avaliação do protótipo, verificam que dos doze atributos de qualidade propostos, oito deles passaram no teste satisfatoriamente sendo que os outros quatro (disponibilidade, gerenciabilidade, performance e escalabilidade) não puderam ser avaliados.

Com este resultado os autores afirmam que a avaliação foi considerada satisfatória. Na visão dos mesmos o fato de não terem conseguido avaliar inicialmente os quatro atributos não impede o desenvolvimento da primeira versão real da aplicação, o que, de fato, ocorreu logo em seguida.

A principal contribuição do trabalho de (GAO, 2007), segundo os mesmos autores, é propor uma arquitetura que simplifique e agilize o desenvolvimento de aplicações multilocatárias nativas de alta qualidade, através da exploração de diversas abordagens que proporcionam suporte a melhores isolamentos entre os inquilinos no tocante a diferentes aspectos, como segurança, desempenho, disponibilidade e administração.

A arquitetura proposta conta com a seguinte estrutura: 1) camada de ativação de multilocação: é o núcleo de toda a estrutura. A camada permite que os locatários aproveitem os benefícios da multilocação nativa enquanto esconde as complexidades de habilitar o recurso tanto dos inquilinos quanto dos desenvolvedores de lógica de negócios. Cinco subdivisões integram a camada: 1.1) é necessário considerar que outros inquilinos podem introduzir ricos de segurança adicionais, além dos mecanismos tradicionais de segurança como autenticação, autorização, auditoria, etc., uma vez que compartilham a mesma instância de aplicativo e recursos; 1.2) isolamento de desempenho: os comportamentos de um inquilino não podem afetar o desempenho de outros inquilinos; 1.3) isolamento de disponibilidade: quando muitos inquilinos compartilham a mesma instância de aplicativo, é fundamental para o ambiente detectar quaisquer falhas e interromper a propagação das mesmas o mais rápido possível; 1.4) isolamento de administração: os consoles administrativos devem ser isolados para cada inquilino para que eles possam visualizar e alterar os dados operacionais e de negócios que sejam específicos de suas próprias organizações; 1.5) customização on-the-fly: a personalização do aplicativo é necessária para que se possa atender as necessidades e situações particulares do inquilino. A fim de minimizar o tempo de inatividade do sistema, a personalização geralmente é realizada enquanto a instância do aplicativo está em operação.

Os autores observam que aplicaram partes do design e dos princípios da arquitetura em um aplicativo multilocatário real. A experiência prática os ajudou a identificar mais

tópicos de pesquisa, especialmente nas áreas de proteção de informações, desempenho e isolamento de disponibilidade. Na visão dos autores é necessário desenvolver algoritmos de criptografia híbrida de alta eficiência para obter melhor compensação entre segurança e desempenho.

Os autores (HUANG, 2013) apresentam um modelo de software SaaS multilocatário para grandes organizações. Segundo os autores o desenvolvimento desse tipo de software para grandes empresas é mais complicado, pois existem muitas suborganizações hierárquicas que não apenas têm o requisito de gerenciamento personalizado e isolamento de dados, mas também necessitam de compartilhamento de recursos e interação de dados entre si. Por essa razão os autores pretendem explorar um modelo SaaS em nível de sistema que não apenas esteja em conformidade com o requisito do inquilino na configuração personalizada e isolamento de dados, mas também implemente de forma eficaz a colaboração entre inquilinos e o compartilhamento de recursos para grandes organizações.

A descrição do modelo apresentado pelos autores é a seguinte: 1) estrutura organizacional comum: é o arranjo formal de cargos dentro de uma organização. Muitos tipos de estruturas organizacionais foram projetadas para satisfazer os requisitos práticos do trabalho de gerenciamento organizacional. As mais populares são: 1.1) estrutura simples: um projeto organizacional com baixa departamentalização, ampla amplitude de controle, autoridade centralizada e pouca formalização. É popular entre organizações pequenas e iniciantes; 1.2) estrutura funcional: um projeto organizacional que agrupa especialidades ocupacionais semelhantes ou relacionadas; 1.3) estrutura divisional: uma estrutura organizacional composta de unidades ou divisões semiautônomas separadas. Concentra-se em resultados, os gerentes são responsáveis pelo que acontece com seus produtos e serviços; 1.4) estrutura matricial: Uma estrutura organizacional que atribui especialistas de diferentes departamentos funcionais para trabalhar em um ou mais projetos. O gerente de projeto tem autoridade sobre os membros da equipe no projeto, mas todos os membros da equipe ainda estão sujeitos aos seus departamentos funcionais; 2) organização e funcionário: a estrutura básica do modelo é baseada no tipo misto de estrutura organizacional introduzida. Organização e funcionário são dois fatores importantes do modelo. 2.1) organização: as organizações podem ser divididas em organização real (OR) e organização virtual (OV). A OR existe no projeto organizacional da estrutura funcional e divisional, podendo ter funcionários e organizações subordinadas. Os serviços de OV estão relacionados à estrutura matricial; 3) inquilino: o inquilino representa o consumidor independente no SaaS. Pode ser uma pessoa, uma organização ou um aplicativo. Neste modelo SaaS para grandes organizações, o inquilino é definido como uma organização ou grupo de organizações específico que compartilha um ambiente de serviço separado no SaaS; 4) dados: por se tratar do bem mais valioso que qualquer empresa possui, o trabalho de gestão dos dados é indispensável no dia a dia das organizações. No modelo descrito existem dois tipos de dados: 4.1) dados de negócios: são os dados relacionados ao trabalho diário do

gerenciamento de negócios. Sua permissão de acesso é restrita a funcionários autorizado; 4.2) dados de recursos: referem-se a vários recursos carregados ou produzidos online pelos inquilinos como vídeos, documentos e fotos.

Como estudo de caso os autores implantaram a arquitetura proposta na Chinese Academy of Sciences (CAS), uma grande organização composta de mais de cem institutos. Os autores reconhecem que apesar de ter satisfeito boa parte dos recursos propostos ainda existem alguns pontos em que a arquitetura precisa sofrer melhorias como, por exemplo, o armazenamento de todos os dados de recursos no mesmo esquema pode trazer problemas de desempenho no futuro quando a quantidade de dados armazenados se tornar muito grande.

Os autores (CAI, 2009) descrevem uma metodologia de ponta a ponta junto com um kit de ferramentas que oferece suporte ao mecanismo de multilocação de granulação fina, ou seja, uma única instância do aplicativo oferece suporte a vários inquilinos ao mesmo tempo.

A descrição da metodologia fornecida pelos autores é a seguinte: 1) principais conceitos: a multilocação de granulação fina é uma evolução natural dos modelos de aplicativos da web tradicionais que modifica alguns componentes-chave do servidor de aplicativos da web e adiciona alguns manipuladores de multilocação e ferramentas de facilitação. As principais mudanças são as seguintes: 1) um método deve realizar a correspondência entre uma solicitação da web e seu inquilino correspondente; 2) mudanças devem existir nos componentes do servidor de aplicativos da web, de modo que a solicitação e a resposta da web possam ser isoladas e customizadas; 3) devem ocorrer mudanças na fonte de dados, uma vez que o banco de dados pode ser compartilhado por diversos inquilinos.

Com relação aos conceitos de multilocação de granulação fina estes são os principais: 1) ID do inquilino: um identificador exclusivo do inquilino que revela sua identidade no sistema; 2) arquivos de configuração do inquilino: eles armazenam o lado do aplicativo e o lado do recurso das informações de um inquilino; 3) contexto do inquilino: é um elemento de contexto de tempo de execução que carrega a configuração do inquilino nos arquivos de configuração durante a fase de inicialização do aplicativo.

Sobre a questão dos pontos de isolamento os autores esclarecem que os mesmos devem ser gerenciados tanto pela aplicação quanto pelos servidores da aplicação. 1) gerenciamento dos pontos de isolamento pela aplicação: os desenvolvedores são os responsáveis por identificar esses pontos de isolamento que incluem alguns tipos como: constante, campo, método e classe. No entanto, os autores se limitaram a estudar apenas o isolamento de um ponto constante no ambiente de multilocação; 2) gerenciamento dos pontos de isolamento pelos servidores da aplicação: diferentes inquilinos podem ter configurações diferentes de recursos que normalmente são incorporados aos arquivos de configuração dos servidores da aplicação.

O objetivo principal dos autores ao propor o método de multilocação de granulação fina é economizar recursos substanciais do sistema e para testar a metodologia proposta os autores transformaram uma aplicação web desenvolvida em Java, chamada Petstore, em uma versão multilocatária. Segundo os autores foram realizadas pequenas modificações no código original, mas diversas alterações nas configurações do sistema. Na versão dos autores cada inquilino tem seu próprio banco de dados, índice de produto, layout de página da Web e domínios de autenticação.

Os autores simularam cinco usuários simultâneos para acessar a página do produto, a página de pesquisa e a página de upload de dados do sistema Java Petstore. Os resultados alcançados demonstram que os objetivos foram satisfeitos, uma vez que, a utilização do método de multilocação reduziu o uso da memória original em pelo menos 75%.

As várias partes envolvidas em um sistema SaaS, segundo os autores (AßMANN, 2012), possuem requisitos e interesses diferentes ou mesmo contraditórios. Um provedor, por exemplo, está interessado em minimizar os custos operacionais com o intuito de maximizar seu lucro. Um inquilino, por sua vez, está interessado em oferecer funcionalidades personalizadas aos seus usuários. Já o usuário foca na qualidade do aplicativo escolhendo, por exemplo, o algoritmo disponível mais rápido para um cálculo em tempo de execução.

O trabalho desenvolvido pelos autores apresenta a visão dos mesmos com relação a uma arquitetura SaaS variável e adaptativa. Os autores identificam requisitos que uma arquitetura de referência SaaS com reconhecimento de multilocação deve oferecer tanto em tempo de design como em tempo de execução. Em tempo de design, como explicam os autores, certas informações do aplicativo precisam ser modeladas. Essas informações incluem os componentes de software e hardware e suas propriedades, bem como configurações do inquilino que definem quais desses componentes estarão disponíveis para seus usuários e como eles são limitados. Por sua vez os requisitos de tempo de execução incluem a possível avaliação das informações modeladas para gerenciar os inquilinos e seus usuários, bem como para auto-otimizar o ambiente de tempo de execução de acordo com a configuração de um inquilino e as demandas de seus usuários. A arquitetura proposta, chamada pelos autores de arquitetura variável, é descrita a seguir: 1) infraestrutura de nuvem: um cluster de servidor em que os nós do servidor são distribuídos em vários locais e são capazes de se comunicar uns com os outros. Se um servidor falhar, os aplicativos afetados serão automaticamente alternados para outro nó; 2) cluster de servidor escalonável: o cluster de servidor suporta escalonamento horizontal, ou seja, os nós do servidor são adicionados ou removidos de acordo com sua utilização do cluster; 3) propriedades não funcionais mensuráveis: capacidade de medir as propriedades não funcionais de todos os recursos na nuvem a qualquer momento; 4) aplicativo de instância única: a multilocação exige uma estratégia de instância única configurável no nível do aplicativo, enquanto a instância do aplicativo é compartilhada por vários inquilinos, mas seus componentes per-

mitem configuração e personalização específicas do inquilino; 5) Aplicativo distribuído: Uma instância do aplicativo multilocatário é distribuída em vários nós do servidor físico no cluster de servidor; 6) Aplicativo escalonável em tempo de execução: No tempo de design de um aplicativo multilocatário as configurações concretas do inquilino que estarão contidas no aplicativo de instância única são desconhecidas. Considerando que no tempo de execução do aplicativo novas configurações do inquilino são criadas e adicionadas à instância do aplicativo.

### 4.1.3 Segurança

Os autores (MATSUO, 2012) apresentam um framework SaaS voltado para a segurança de dados. O framework utiliza o que os autores chamam de Gateway de Informações (GI). Ao ser configurado no ambiente do inquilino, o GI passa a monitorar os dados de acordo com se eles possuem ou não informações sigilosas. Somente os dados protegidos são enviados para a aplicação SaaS na nuvem.

Quando se trata de informações confidenciais a aplicação SaaS não pode ser utilizada, segundo afirmam os autores, por ser tratar de uma aplicação que armazena os dados dos inquilinos na nuvem. Por essa razão os autores foram estimulados a desenvolver o GI com o intuito de permitir o uso do SaaS de forma segura, ou seja, sem o envio de informações confidenciais diretamente para a nuvem. As funcionalidades do GI são as seguintes: 1) criptografia de dados (descriptografia): baseado em uma política de dados o GI realiza a criptografia das informações confidenciais e as envia para o aplicativo SaaS na nuvem. O GI também percorre o caminho inverso, ou seja, a descriptografia quando o inquilino necessita obter resultados de processamento da nuvem; 2) determinação dinâmica do local de execução: se acontecer da aplicação SaaS não estiver disponível para receber os dados criptografados, o GI deverá mudar, de forma dinâmica, o local de execução para o ambiente do cliente onde o pré-processamento será realizado com segurança por meio de uma estrutura chamada pelos autores de sandbox de implantação. Ao término do pré-processamento somente as informações confidenciais serão enviadas para o aplicativo SaaS na nuvem; 3) montagem de dados para evidências de auditoria: para se certificar de que os dados do cliente sejam processados de maneira correta no ambiente do cliente, tanto os logs de comunicação com o aplicativo SaaS quanto os logs de acesso a arquivos no ambiente do cliente devem ser armazenados pelo GI. Se houver algum tipo de problema no ambiente do cliente, o inquilino deve ser capaz de aceitar esses logs mesclados com os logs da nuvem sem influenciar o serviço da mesma, como a evidência de auditoria; 4) portabilidade para aplicações existentes: integrar recursos já desenvolvidos por outras aplicações na aplicação SaaS é uma característica importante para que os desenvolvedores possam se dedicar apenas a descrever a lógica de negócios. Para isso, como dizem os autores, a lógica de roteamento deve ser expressa de forma intuitiva e abrangente.

A arquitetura proposta pelos autores possui três componentes essenciais: 1) estrutura de integração: trata-se do componente principal que viabiliza a utilização de uma aplicação SaaS sem que seja necessário o envio de informações confidenciais para a nuvem; 2) sandbox estendida: os autores aperfeiçoaram as funções da JVM (Java Virtual Machine) para que pudessem obter acesso aos recursos locais e a comunicação com a aplicação SaaS na nuvem, uma vez que, por padrão, a sandbox limita o acesso a certos recursos, como dados e aplicações do cliente, de uma aplicação de terceiros. Os logs obtidos são mantidos no GI e nunca enviados para a nuvem; 3) gerenciador de segurança de dados: decide onde o GI deve realizar o processamento ou se os dados devem ser criptografados ou não.

Por fim os autores apresentaram um estudo de caso onde adotaram um serviço que analisa códigos legados e retorna os resultados da análise para o usuário cliente. Quando o usuário acessa o serviço de análise na nuvem usando o GI, o pré-processamento e a criptografia são realizados no ambiente do cliente e o processamento restante continua na nuvem. Os autores então afirmam que o framework por eles proposto funciona satisfatoriamente, uma vez que permitiu um serviço de análise perfeito sem enviar as informações confidenciais para a nuvem.

Os autores (ALMORSY, 2012) apresentam a TOSSMA, uma arquitetura de gerenciamento de segurança SaaS orientada ao inquilino que permite que os próprios inquilinos definam, personalizem, apliquem e monitorem seus requisitos de segurança sem dependerem dos desenvolvedores de aplicativos para manutenção ou personalizações de segurança. Além disso, permite que os provedores de aplicativos SaaS gerenciem o isolamento de segurança entre seus inquilinos de serviço.

Os componentes que integram a arquitetura são os seguintes: 1) banco de dados de descrição da arquitetura do aplicativo: a TOSSMA consegue manipular vários aplicativos SaaS hospedados na mesma plataforma em nuvem. O provedor que quiser acoplar seu aplicativo SaaS com a TOSSMA deve fornecer o arquivo de descrição da arquitetura do seu aplicativo que descreve os principais componentes do aplicativo, arquivos de configuração dos mesmos e pacotes de implantação. O banco de dados de descrição da arquitetura do aplicativo armazena esses arquivos e a TOSSMA os utiliza para realizar engenharia reversa nas classes e métodos e configurar o sistema; 2) console de gerenciamento de segurança do aplicativo: Os inquilinos selecionam na descrição do sistema (componentes, classes, métodos, etc.) os pontos críticos com os quais estão preocupados e especificam seus requisitos de segurança; 3) banco de dados de requisitos de segurança multilocatário: para que os inquilinos e provedores possam visualizar, manter e aplicar seus próprios requisitos de segurança sem interferir nos requisitos de outros inquilinos e provedores, os requisitos de segurança especificados são armazenados no banco de dados de requisitos de segurança multilocatário; 4) invólucro do sistema: é o modulo responsável por aplicar novos requisitos de segurança, definidos pelos inquilinos, em um determinado componente

do sistema. Se, por exemplo, o inquilino especificou algum requisito de segurança em algum componente, então todos os métodos deste componente devem ser interceptados para que o requisito de segurança seja imposto; 5) banco de dados de controles de segurança: inquilinos e provedores de serviços podem estabelecer controles de segurança que podem ser usados para proteger os dados dos inquilinos de SaaS enquanto são armazenados, processados e transmitidos; 6) ponto de aplicação de segurança: as solicitações a um recurso crítico do sistema são interceptadas pelo invólucro do sistema e enviadas ao ponto de aplicação de segurança. Isso carrega os requisitos de segurança especificados para o recurso interceptado, de acordo com o inquilino requisitante, usando o banco de dados de requisitos de segurança multilocatário.

Para testar a arquitetura TOSSMA os autores utilizaram dois aplicativos recém-desenvolvidos, GalacticERP e PetShop, além de outros dois aplicativos web já existentes SplendidCRM e KOOBOO. A intenção dos autores era avaliar, em aplicativos novos e existentes, alguns recursos da arquitetura. Os autores afirmam que a TOSSMA logrou êxito em capturar e impor diferentes atributos de segurança (gerenciamento de identidade, autenticação, autorização, criptografia, assinatura digital e validação de entrada) para vários locatários em tempo de execução e na mesma instância de serviço.

Os autores também identificaram uma limitação na arquitetura. Segundo eles, antes de utilizar a TOSSMA é necessário que aplicativos que possuam segurança integrada sejam modificados para desabilitar a segurança, pois caso contrário, eles continuarão a aplicar os antigos e os novos requisitos de segurança.

No geral, os autores afirmam que a TOSSMA consegue cumprir satisfatoriamente o que foi proposto, ou seja, fornecer uma arquitetura de gerenciamento de segurança SaaS orientada ao inquilino.

### 4.1.4 Recuperação de falhas

Preocupados com a questão da recuperação de falhas em sistemas SaaS multilocatário, os autores (YOUNAS, 2014) analisam uma abordagem de recuperação de falhas chamada checkpoint e afirmam que as técnicas tradicionais dessa abordagem estão enfrentando um grande desafio quando aplicadas a sistemas SaaS multilocatário devido à grande escala do estado do sistema e à diversidade de requisitos dos usuários sobre a qualidade dos serviços.

O artigo desenvolvido pelos autores propõe uma nova técnica de checkpoint chamada de checkpoint de nível de inquilino. A ideia dessa técnica é extrair e salvar os dados que pertencem a um inquilino específico para cada chamada da operação de checkpoint. A principal vantagem é que cada operação de checkpoint pode ser feita em um momento apropriado para que se tenha efeito mínimo sobre o desempenho de todo o sistema. Todo

o estado do sistema pode ser salvo por meio de uma série de invocações de operações de checkpoint, por exemplo, inquilino por inquilino. Outra vantagem da abordagem proposta é que a frequência dos pontos de verificação pode ser adaptada de acordo com os requisitos do inquilino sobre confiabilidade e qualidade dos serviços. Além disso, após uma interrupção do sistema, a reversão também pode ser realizada gradualmente de modo inquilino a inquilino para que o tempo total de interrupção do sistema possa ser reduzido. No caso de falha parcial do sistema, a reversão também pode ser realizada reiniciando apenas os inquilinos afetados.

O emprego da tecnologia de Big Data na implementação de operações de checkpoint é a segunda ideia básica da abordagem proposta, pois ao utilizar essa tecnologia os autores afirmam que a latência de tempo e outras sobrecargas serão ainda mais reduzidas. Outro benefício da tecnologia de Big Data, na visão dos autores, é o aproveitamento das vantagens do poder de processamento paralelo da infraestrutura em nuvem e a distribuição de dados de checkpoint em um cluster de servidores.

A abordagem proposta foi implementada pelos autores em um protótipo chamado Tench (Tenant level Checkpointing) que foi desenvolvido utilizando o Eclipse 3.7.2 com a biblioteca Couchbase Java SDK. O protótipo foi implantado e executado em um pequeno cluster de oito computadores.

Os dois experimentos realizados pelos autores tiveram como objetivo responder as seguintes questões de pesquisa: 1) como a latência do checkpoint varia com o tamanho dos dados do checkpoint? 2) como o número de inquilinos afeta a latência do checkpoint?

A descrição dos experimentos é a seguinte: no primeiro experimento a operação de checkpoint foi aplicada aos estados do sistema onde o tamanho dos dados do checkpoint de um inquilino varia de 10.000 registros a 1.000.000 de registros, onde cada registro é um documento JSON. Cada etapa aumenta o tamanho em 10.000 registros. Cada registro tem 128 bytes. No segundo experimento o tamanho dos dados a serem verificados é fixo, mas o número de inquilinos no sistema varia. - Para cada um dos dois experimentos, um banco de dados NoSQL (Couchbase) é populado com dados aleatórios de documentos JSON.

Os resultados dos experimentos obtidos pelos autores são descritos a seguir: 1) latência de diferentes tamanhos de checkpoint: para a abordagem baseada em disco, a latência do checkpoint aumenta linearmente com o aumento do tamanho dos dados do checkpoint. Uma observação semelhante é feita nos experimentos com checkpoints baseados em banco de dados NoSQL. Em ambas as abordagens, a latência aumenta de forma linear com o aumento do número de registros; 2) Latência de diferentes números de inquilinos: a latência do checkpoint varia pouco com o aumento do número de inquilinos no sistema se a quantidade total de dados no sistema e a quantidade de dados a serem verificados permanecerem iguais. No entanto, nos checkpoints baseados em banco de dados NoSQL, a variação de latência é muito maior do que nos checkpoints baseados em

arquivos. Isso acontece devido a estrutura de cache de vários níveis dos bancos de dados NoSQL.

Uma taxonomia de falha preliminar para sistemas SaaS multilocatários é proposta pelos autores (PINTO, 2019) considerando as perspectivas industrial e de pesquisa.

Para investigar a perspectiva industrial os autores desenvolveram e aplicaram um questionário aos desenvolvedores e testadores de sistemas SaaS visando identificar falhas comuns encontradas tanto durante o desenvolvimento como na fase de testes desses sistemas. O questionário utilizado pelos autores é composto por sete questões: 1) Como você avalia sua experiência com sistemas SaaS multilocatários? 2) Com base na classificação de possíveis falhas, você teria sugestões? Descreva abaixo, isso será muito útil. 3) Quais são as tecnologias de desenvolvimento utilizadas por você? 4) Que tipo de testes são realizados em sua empresa ou grupo de pesquisa? (Teste de unidade, teste de sistema, teste baseado em cobertura, automação de teste, etc.). Diga-nos qual ferramenta de teste você utilizou. 5) Quais são as falhas frequentemente encontradas durante os testes? 6) Em sua opinião, existem limitações na atividade de teste (práticas, ferramentas, etc.) neste contexto? Quais? 7) Você teria alguma sugestão para melhorar o processo de teste? Por favor, descreva-os.

Após a análise das respostas dos quarenta e sete profissionais que responderam o questionário os autores descrevem as conclusões para cada uma das questões. Para a questão 1 estavam disponíveis três opções: iniciante (menos de 1 ano), intermediário (quase 2 anos) ou especialista (mais de 2 anos). O resultado foi que 95% dos participantes poderiam ser considerados especialistas. O restante possuía uma experiência intermediária. Os autores afirmam que esta questão é importante para estabelecer certo nível de confiabilidade dos dados com base na experiência dos participantes. Com relação à questão 2, os autores disponibilizaram uma versão preliminar da classificação de falhas desenvolvida por eles para que os profissionais pudessem avaliá-la e sugerir melhorias. Dentre as propostas a maioria dos entrevistados apontou falhas que poderiam ser mescladas ou refinadas. Na terceira questão a maioria dos entrevistados respondeu Java com EclipseLink como tecnologias empregadas para desenvolver aplicações multilocatárias. Os demais entrevistados mencionaram: Spring Boot, Python / Django, Rails / Heroku e node js, html5, css3 e javaScript. Na questão 4 os seguintes tipos de testes foram mencionados: teste de unidade, teste de integração, teste funcional, teste de usabilidade e teste de carga. Com relação às ferramentas de teste, os entrevistados citaram: RSpec, JUnit, Cucumber, JBehave, Selenium, pyunit e CI com Hudson. As falhas mais frequentes, mencionadas pelos entrevistados na questão 5, estão relacionadas a baixos requisitos de qualidade e problemas relacionados ao Service Level Agreement (SLA) e Quality of Service (QoS). Em geral a principal preocupação dos testadores é em relação à segurança para evitar ataques de hackers. Na sexta questão os entrevistados discutiram sobre a falta de um processo de teste

orientado ao inquilino e padrões para teste e desenvolvimento. Finalmente na questão 7 sugestões de melhorias no processo de teste foram apresentadas e de acordo com a maioria dos entrevistados, o principal problema é que ainda não há uma atividade de teste em conformidade com as novas tendências e necessidades do ciclo de vida de construção de software como entrega contínua, implantação e integração.

Com relação à perspectiva da pesquisa uma revisão sistemática da literatura foi realizada pelos autores com o objetivo de coletar todos os estudos relevantes que apresentam contribuições relacionadas às técnicas de teste para sistemas SaaS.

### 4.1.5 Execução de aplicativos atuais em ambiente multilocatário

Os autores (AZEEZ, 2010) implementaram uma plataforma de Arquitetura Orientada a Serviços (Service Oriented Architecture - SOA) multilocatário que permite aos usuários executar seus aplicativos atuais em um ambiente multilocação. Os aplicativos SOA, segundo os autores, geralmente consistem em coleções de artefatos que incluem: 1) implementações de serviço: lógica de negócios, acesso a dados, interface de serviço, etc.; 2) processos de negócios: orquestrações de serviços e tarefas de interação humana que devem ser concluídas para alcançar uma interação de negócios específica; 3) Aplicativos da Web: aplicativos baseados na Web que permitem aos usuários interagir com serviços corporativos e realizar transações comerciais.

A solução proposta pelos autores é implementada sobre a plataforma WSO2 Carbon, uma estrutura de middleware de código aberto para a construção de servidores escaláveis e de alto desempenho que consiste em três componentes: 1) núcleo WSO2 Carbon: é o motor principal do WSO2 Carbon e gerencia o carregamento, vinculação, execução e gerenciamento dos componentes da plataforma; 2) framework do console de gerenciamento da interface do usuário: seu principal recurso é o framework do console da interface do usuário (IU) que fornece um modelo de IU para todos os consoles de gerenciamento do servidor WSO2 Carbon; 3) todos os outros componentes que são executados dentro do framework.

No que diz respeito à multilocação para SOA os autores descrevem os diferentes aspectos da arquitetura do sistema: 1) implantação de serviço: os componentes de hospedagem de serviço do WSO2 Carbon têm o Apache Axis2 como mecanismo de execução subjacente. A arquitetura do Apache Axis2 foi projetada para oferecer suporte a implantações multilocatário. O Axis2 é um sistema que possui todos os estados de configuração armazenados em uma árvore de configuração chamada AxisConfiguration e todos os estados de tempo de execução armazenados em outra árvore chamada ConfigurationContext. Todos os elementos de configuração, como serviços e módulos, são criados no escopo de um AxisConfiguration; 2) envio de mensagens: quando um cliente envia uma mensagem endereçada ao serviço de um determinado inquilino essa solicitação deve indicar o inquilino de

alguma maneira. A abordagem padrão no WSO2 Carbon é adicionar o nome do inquilino a URL da seguinte forma: http://exemplo.com/t/nome-do-inquilino/servicos/nome-do-servico; 3) segurança: um desafio para habilitar a multilocação é fornecer mecanismos para que os inquilinos tenham controle de segurança sobre seus serviços. Os requisitos incluem ter uma base de usuários específica do inquilino que suporte políticas de segurança específicas do inquilino; 4) execução de serviço: a execução de serviço por inquilino em uma única JVM é uma tarefa desafiadora, pois o WSO2 Carbon precisa garantir que o código malicioso de um inquilino não consiga acessar os dados de outro inquilino. Isso se estende não apenas aos serviços, mas a qualquer outro artefato que possa ser implantado como parte do aplicativo WSO2 Carbon. O isolamento total das implantações desses diferentes inquilinos é um requisito primário e obrigatório; 5) acesso a dados: a estrutura WSO2 Carbon fornece um repositório que abstrai a camada de banco de dados relacional, ao mesmo tempo que oferece suporte a uma API orientada a recursos mais simples para armazenar e recuperar dados persistentes.

Alguns experimentos iniciais foram conduzidos pelos autores com o intuito de medir o impacto da multilocação no desempenho geral do sistema. Para a configuração do experimento os autores utilizaram um serviço simples de cotação de ações que retorna uma cotação de ações gerada aleatoriamente. O serviço foi implantado nas versões multilocatário e não multilocatário do WSO2 Web Services Application Server, um servidor de execução de serviço baseado no WSO2 Carbon.

Por fim os autores apresentaram os resultados dos experimentos. De acordo com eles durante a execução do teste a utilização da CPU foi de quase 90% e a da rede foi de aproximadamente 95%. A máquina foi capaz de manter um heap de memória quase constante enquanto aumentava a carga e a simultaneidade. Como os autores já esperavam houve, de fato, uma sobrecarga introduzida pela multilocação. No entanto, a variação dessa sobrecarga foi de 10% a 20% apenas e, mesmo com a multilocação, o servidor foi capaz de sustentar uma respeitável taxa de transferência de 600 solicitações por segundo. Apesar da sobrecarga imposta pela multilocação os autores chegaram à conclusão de que as medições iniciais atestaram que o sistema resultante é eficaz.

### 4.1.6 Gerenciamento de recursos

A multilocação, segundo os autores (ELKHATIB, 2016), apesar de seus benefícios, apresenta desafios adicionais como particionamento, extensibilidade e personalização durante o desenvolvimento do aplicativo e, após sua implantação, novos requisitos dos clientes e mudanças no ambiente de negócios resultam na evolução do aplicativo. Essa evolução constante, exigida por clientes individuais, apesar de adicionar cada vez mais complexidade ao aplicativo não deve de forma alguma afetar a disponibilidade, a segurança e o desempenho do aplicativo para outros clientes.

Na visão dos autores o gerenciamento de toda essa complexidade requer abordagens e ferramentas adequadas. Por essa razão os mesmos propõem técnicas de modelagem de linhas de produtos de software (Software Product Lines - SPL) e engenharia orientada a modelos (Model-Driven Engineering - MDE) para gerenciar a variabilidade e apoiar a evolução dos aplicativos multilocatários e seus requisitos.

A abordagem dos autores, baseada no trabalho de Jayaraman et al., mantém a separação de recursos e a detecção de dependências estruturais e conflitos entre recursos durante a análise e a modelagem do projeto. Recursos ou grupos de recursos são modelados usando a linguagem UML. Uma linguagem de composição de modelo chamada MATA (Modeling Aspects using a Transformation Approach) detecta relacionamentos e conflitos.

Para explorar a abordagem proposta os autores apresentam um estudo de caso do serviço Surveys da Microsoft. Surveys é um aplicativo SaaS multilocatário para criar e gerenciar pesquisas online. Os inquilinos podem criar ou publicar pesquisas e analisar resultados.

### 4.1.7 Middleware

Os autores (CHEN, 2014) relatam o design de um middleware habilitado para vários inquilinos que visa dar suporte ao desenvolvimento e implementação de aplicativos SaaS corporativos.

O Middleware de serviço proposto aborda três aspectos essenciais de design: 1) Serviço de gerenciamento de contexto do inquilino: neste aspecto os autores explicam que uma abordagem comum para os clientes de um aplicativo SaaS baseados na Web é armazenar os contextos do inquilino em uma implementação de sessão HTTP. Os autores alertam, no entanto, que essa abordagem só se aplica a clientes baseados na Web. Para sanar esse problema os autores desenvolveram um mecanismo independente de plataforma que permite que a lógica do programa acesse os contextos do inquilino a partir de interfaces de usuário, lógicas de negócios e dados sem depender da sessão HTTP; 2) Serviço de mapeamento de esquema multilocatário: os autores descrevem o serviço proposto de mapeamento de esquema multilocatário baseado em Tabela Universal. A Tabela Universal é um armazenamento genérico que consiste em um GUID (Global Unique Identifier), um ID de locatário e um número fixo de colunas de dados genéricos. No centro do serviço está um conjunto de regras de reescrita de consultas que especificam as transformações de consultas lógicas em consultas físicas por meio da álgebra relacional; 3) Serviço de mapeamento de objeto relacional multilocatário: os autores nos explicam que as aplicações corporativas normalmente acessam o banco de dados por meio de um framework de mapeamento de objeto relacional (Object-Relational Mapping - ORM). Há, no entanto, uma incompatibilidade dos mecanismos de regravação de SQL com o ORM. Por essa razão os autores propõem uma abordagem para integrar estes mecanismos em um framework

ORM. Esta abordagem consiste em definir os mapeamentos entre os campos do objeto e os campos do banco de dados. Esses mapeamentos são geralmente especificados por anotações no código-fonte. Os autores utilizam a anotação "@MultiTenantCapable"para indicar que o objeto anotado será mapeado para um banco de dados multilocatário. Para que a classe Produto, por exemplo, seja capaz de lidar com vários inquilinos basta adicionar a anotação da seguinte forma: @MultiTenantCapable Public class Product {...}.

Conforme os autores (JOOSEN, 2015) os aplicativos SaaS multilocatário são cada vez mais desenvolvidos com base em combinações de tecnologias e provedores de armazenamento em nuvem em uma configuração chamada multi-nuvem. Nesse ambiente de várias nuvens os dados do aplicativo são distribuídos e replicados em vários sistemas de armazenamento em nuvem, cada um com diferentes modelos de dados suportados, APIs de desenvolvimento, desempenho, escalabilidade, disponibilidade e durabilidade.

Para gerenciar essa arquitetura de armazenamento em várias nuvens que na prática não é nada trivial os autores apresentam um Middleware de gerenciamento de dados orientado por políticas que 1) faz abstração tanto dos provedores quanto das diferentes tecnologias de armazenamento em nuvem; 2) segue uma abordagem orientada por políticas para tomar decisões de colocação de dados e obter os benefícios do multi-armazenamento e configuração de várias nuvens; e 3) permite a especificação de configurações de armazenamento e políticas avançadas de armazenamento de dados, tanto pelo provedor de serviços quanto pelos inquilinos.

A arquitetura do Middleware proposto pelos autores é dividida em quatro camadas: 1) camada de multilocação; 2) camada de aplicativo SaaS; 3) camada de gerenciamento de dados do Middleware; 4) camada de armazenamento distribuído descentralizado.

O núcleo do Middleware e, portanto, o foco dos autores é a camada de gerenciamento de dados do Middleware que é composta por três componentes: 1) Middleware de acesso a dados: os aplicativos SaaS são desenvolvidos com base no componente Middleware de acesso a dados, independentemente das tecnologias subjacentes suportadas por vários provedores de armazenamento em nuvem; 2) gerenciamento de configuração: responsável por armazenar detalhes de configuração de persistência dos inquilinos e provedores SaaS sobre os sistemas de armazenamento de back-end; 3) drivers de armazenamento: oculta a complexidade dos sistemas back-end distribuídos em uma configuração de várias nuvens fornecendo uma API uniforme.

Para validar o Middleware proposto os autores implementaram um protótipo sobre o qual construíram um aplicativo SaaS multilocatário e multi-nuvem. O protótipo foi implementado sobre o framework Kundera, um projeto de código aberto que fornece drivers para diferentes sistemas de armazenamento.

Após a realização dos diversos testes conduzidos pelos autores os mesmos afirmam

que os resultados da avaliação experimental demonstram que os benefícios do Middleware proposto são alcançados com sobrecarga de desempenho mínima.

O trabalho desenvolvido pelos autores (BORGER, 2015) apresenta um middleware de isolamento de desempenho adaptável que visa permitir que os provedores de SaaS gerenciem e apliquem restrições de desempenho concorrentes em aplicativos SaaS multilocatários. O Middleware, segundo os autores, consegue gerenciar uma combinação de restrições de desempenho em termos de latência, taxa de transferência e prazos, permitindo uma resposta rápida em circunstâncias variáveis, enquanto preserva a eficiência do uso dos recursos de multilocação.

Os quatro componentes do Middleware proposto assumem, cada um deles, um papel essencial para atingir o objetivo de atender a vários inquilinos ao mesmo tempo em que atende às diferentes restrições de desempenho concorrentes e preserva o alto desempenho e a eficiência de custo. Os autores descrevem esses componentes da seguinte maneira: 1) camada de gerenciamento de aplicativos e configuração: mantém uma visão geral das configurações dos inquilinos correspondentes, incluindo as restrições de desempenho específicas do inquilino; 2) serviço de monitoramento: coleta dados sobre os trabalhos ativos e tarefas por inquilino, bem como do sistema para avaliar a carga de trabalho; 3) serviço de Priorização: toma decisões que resultam em prioridades. As decisões são baseadas nas tarefas que devem ser entregues, nas diferentes restrições de desempenho aplicáveis e na carga de trabalho real; 4) camada execução adaptativa: gerencia todas as tarefas que devem ser despachadas rapidamente para os trabalhadores multilocatários no ambiente de nuvem. Esta camada incorpora uma fila adaptável que pode reconfigurar dinamicamente a sequência de tarefas que devem ser executadas com base nas prioridades atribuídas. De acordo com os autores é nessa camada que ocorre a operação básica do Middleware que consiste em vários subserviços multilocatários recuperando tarefas da fila e as executam. Ao aplicar as prioridades atribuídas pelo serviço de priorização, a ordem das tarefas enfileiradas é adaptada em tempo de execução e, portanto, influencia a seleção da próxima tarefa. Por sua vez, o serviço de priorização consulta o serviço de monitoramento e a camada de gerenciamento de configuração para tomar as decisões adequadas em tempo de execução com base em um algoritmo de priorização ou escalonamento.

Com o objetivo de avaliar o Middeware proposto os autores desenvolveram um protótipo baseado em Java sobre uma plataforma de nuvem privada baseada em OpenStack e JBoss. A avaliação, conforme os autores, demonstrou a eficácia e a flexibilidade da solução proposta ao garantir a conformidade com as restrições de desempenho concorrentes. Além disso, o Middleware apresentou apenas uma sobrecarga de desempenho limitada e, portanto, preservou os benefícios da multilocação em nível de aplicativo.

A evolução ou mesmo a atualização de um aplicativo SaaS multilocatário sem afetar profundamente a continuidade do serviço, na visão dos autores (GEY, 2015), é

sempre problemática, pois nem todos os inquilinos são iguais e, para algumas organizações, as interrupções custam mais caro do que para outras.

Visando solucionar o problema acima exposto os autores apresentam o design e a implementação de um Middleware aprimorado para a evolução de aplicativos SaaS multi-locatários que permite a adaptação em tempo de execução por meio da ativação gradual de upgrades, inquilino a inquilino. O mecanismo de adaptação, conforme os autores, suporta a configuração com base nas entradas do administrador do inquilino e outras partes interessadas e é automatizado ao máximo. O Middleware aplica internamente uma sequência de manipulações a instâncias de composição de serviço específicas limitando o impacto na continuidade do serviço.

Para mostrar a flexibilidade do Middleware proposto no contexto de um provedor SaaS do mundo real que fornece um serviço de processamento de documentos os autores implementaram três atualizações diferentes para um aplicativo SaaS multilocatário que está constantemente ocupado processando solicitações.

O protótipo, como afirmam os autores, demonstrou uma melhoria significativa na continuidade do serviço durante a aprovação da atualização, impondo sobrecarga de desempenho insignificante em operação normal.

### 4.1.8 Banco de dados

O trabalho dos autores (AN, 2011) propõe, após analisar os requisitos e as lacunas no banco de dados tradicional ao oferecer suporte ao cenário SaaS, uma nova estrutura de banco de dados para resolver os desafios de projetar e construir um banco de dados multi-locatário econômico, seguro, personalizável, escalonável e não intrusivo que acelera muito a migração e o desenvolvimento de aplicativos SaaS. Os autores, além de discutir e comparar diferentes abordagens de implementação, também identificam algumas abordagens de otimização de desempenho de banco de dados no cenário multilocatário.

A estrutura de banco de dados multilocatário proposta pelos autores possui os seguintes componentes detalhados pelos mesmos: 1) aplicativo/agente cliente: Obtém a identidade do inquilino do usuário atual e propaga para o serviço de tempo de execução. Fornece interface de programação transparente e visão lógica. O desenvolvedor pode usar a estrutura do banco de dados multilocatário sem alterar muito seus aplicativos; 2) serviço de tempo de execução: interage com o agente cliente e mapeia a visão lógica do inquilino para o modelo de armazenamento físico real no tempo de execução com base na configuração do inquilino no repositório de metadados. Também executa roteamento de solicitação de dados do inquilino em tempo de execução, aplicação de segurança e isolamento de desempenho; 3) serviço operacional e console de administração: gerencia o aplicativo e o ciclo de vida do locatário (registro ou atualização do aplicativo), pool de

recursos de banco de dados, escalabilidade e configuração de disponibilidade, monitoramento, operações orientadas ao inquilino (backup, restauração, migração de dados, etc.); 4) Meta Repositório MMT: armazena informações de metadados do sistema de banco de dados multilocatário, incluindo informações relacionadas ao esquema do aplicativo, recursos de banco de dados físicos subjacentes, configuração de uso de recursos do aplicativo e do inquilino, informações de monitoramento, etc.

O sistema implementado pelos autores, como os mesmos afirmam, já foi usado em dezenas de compromissos reais com clientes na China e nos Estados Unidos. Ele ajudou a migrar muitos aplicativos legados para modelos SaaS e várias operadoras de SaaS a construir e operar centros de dados SaaS.

Os autores, através da experiência prática que tiveram, identificaram mais tópicos de pesquisa, especialmente nas áreas de customização, isolamento de segurança flexível e otimização de desempenho e pretendem dar continuidade ao estudo desses itens visando aprimorar cada vez mais o trabalho realizado.

Um novo design de banco de dados personalizável para aplicativos multilocatários é proposto pelos autores (FEUERLICHT, 2011). O projeto apresenta uma Tabela de Extensão Elástica (TEE) que consiste em Tabelas de Inquilinos Comuns (TIC) e Tabelas de Extensão Virtual (TEV). Esse design permite que os inquilinos criem seu próprio esquema de banco de dados elástico em tempo de execução do aplicativo multilocatário para satisfazer suas necessidades de negócios. Os autores afirmam que em comparação com as técnicas de tabelas de extensão já propostas, a TEE exibe uma maneira flexível de construir um esquema de banco de dados fornecendo alta extensibilidade para os bancos de dados multilocatários, melhora o desempenho ao eliminar valores NULL, atribuir chaves primárias para as colunas exclusivas, fornecer índices para as colunas, criar relacionamento de banco de dados entre tabelas virtuais e comuns e armazenar dados BLOB e CLOB em tabelas designadas.

Como já dito a técnica TEE consiste em dois tipos de tabelas: TIC e TEV. As tabelas TEVs podem ser criadas para atender às necessidades específicas de cada inquilino. A tabela "inquilino", por exemplo, é uma das TICs que pode ser utilizada e compartilhada por todos os inquilinos para armazenar suas informações, enquanto a TEV será uma extensão da tabela "inquilino" e incluirá colunas extras que não fazem parte da tabela comum "inquilino".

O trabalho desenvolvido pelos autores (FEUERLICHT, 2013) propõe uma tecnologia denominada pelos mesmos de EET (Elastic Extension Tables), um serviço de proxy de banco de dados multilocatário que combina tabelas relacionais e tabelas relacionais virtuais fazendo-as trabalhar juntas e atuar como um banco de dados para cada inquilino. Este banco de dados combinado permite que os inquilinos projetem seu banco de dados e configurem automaticamente seu comportamento em tempo de execução do aplicativo.

Segundo os autores os aplicativos desenvolvidos por meio da abordagem proposta permitem recuperar dados dos inquilinos através de chamadas de funções o que evita o gasto de dinheiro e o esforço para escrever consultas SQL e códigos de gerenciamento de dados de back-end.

Quatro tipo de experimentos foram realizados pelos autores com o objetivo de verificar a viabilidade do serviço proposto. As classificações dos experimentos foram denominadas simples, simples a médio, médio e complexo, dependendo da complexidade das consultas utilizadas. Os quatro experimentos mostram comparações entre o tempo de resposta de recuperação de dados de CTTs (Common Tenant Tables), VETs (Virtual Extension Tables) ou ambas as CTTs e VETs.

Através dos resultados experimentais os autores descobriram que o desempenho médio das CTTs, VETs e CTTs-e-VETs para as consultas simples e as consultas simples para médias são consideradas quase iguais. Constataram também que o desempenho médio das consultas médias para VETs e CTT-e-VET é considerado ligeiramente superior ao das CTTs, mas as VETs são as mais elevadas entre os três tipos de tabelas. Os resultados experimentais das consultas complexas demonstraram que o desempenho médio das CTTs-e-VETs é considerado superior ao das CTTs em cerca de 1,2 segundos e para as VETs é considerado superior ao das CTTs em cerca de 1,5 segundos.

A proposta dos autores (GOYAL; YAISH, 2013) é um projeto de arquitetura para construir uma camada de banco de dados intermediária para ser usada entre aplicativos de software e sistemas de gerenciamento de banco de dados relacional visando armazenar e acessar dados de vários inquilinos no esquema de banco de dados multi-inquilino Elastic Extension Table (EET). Os autores explicam que essa camada de banco de dados combina tabelas relacionais multilocatárias e tabelas relacionais virtuais e as faz trabalhar juntas para atuar como um banco de dados para cada inquilino. Ainda segundo os autores este projeto de arquitetura de banco de dados multilocatário pode ser usado como uma base tanto para a construção de aplicativos de software em geral como também para aplicativos de Software como Serviço (SaaS).

Os autores reconhecem que projetar e desenvolver um banco de dados multilocatário configurável para gerar e executar consultas de inquilinos usando uma solução de base de código e converter consultas multilocatárias em consultas regulares de banco de dados é difícil, complicado e requer tempo e trabalho para ser realizado.

Visando solucionar o problema supracitado os autores apresentam um design de arquitetura para simplificar e acelerar o desenvolvimento de soluções de banco de dados multilocatário que permite aos provedores de serviços de banco de dados criarem um único aplicativo de banco de dados que ofereça suporte a vários inquilinos na mesma infraestrutura de hardware e software. A camada de banco de dados oferecida pela arquitetura proposta pode ser usada sozinha como serviço de camada de banco de dados multiloca-

tário para aplicativos móveis, web e autônomos ou pode ser considerada uma camada de banco de dados multilocatário para aplicativos SaaS multilocatários.

A avaliação do desempenho de recuperação e armazenamento de dados dos inquilinos no EET bem como o dimensionamento do design de arquitetura de banco de dados multilocatário em várias instâncias de servidor será, segundo os autores, o próximo trabalho dos mesmos.

Uma arquitetura de banco de dados multilocatário simples, robusta, eficiente em consultas, escalonável e com economia de espaço é proposta pelos autores (KUSHWAHA; PIPPAL, 2013). Segundo os autores uma das vantagens do banco de dados multilocatário é que ele pode acomodar um número maior de inquilinos, pois uma única instância de banco de dados é usada para armazenar os dados de vários inquilinos. Outra vantagem do trabalho proposto, ainda de acordo com os autores, é que os inquilinos podem criar múltiplas tabelas que agregam flexibilidade em termos de um conjunto variado de atributos conforme especificamente requerido para sua aplicação.

A abordagem proposta foi implementada com sucesso utilizando o banco de dados MySQL no Ubuntu. Para os testes e geração de relatórios o Python foi usado como linguagem de script.

Através dos resultados dos testes os autores observaram os seguintes pontos com relação a abordagem proposta: 1) a arquitetura do banco de dados multilocatário é altamente eficiente em termos de execução de consultas, economia de espaço e alteração no número de atributos; 2) o desempenho é moderado e comparável com a abordagem da tabela de extensão para solicitações simultâneas; 3) há uma melhoria de 20% a 30% nas operações de inserção, exclusão e atualização; 4) a abordagem da tabela de extensão é menos eficiente do que a abordagem proposta com relação ao desempenho da consulta quando o número de atributos aumenta na tabela; 5) o gerenciamento de dados e escalabilidade proposto é simples de implementar e eficiente devido ao conceito usado para desacoplar o gerenciador de banco de dados e o gerenciador de transações.

Os autores (LI, 2011) propõem uma solução para resolver o problema do modo de entrega tradicional de sistemas de informação educacional, onde aplicativos e bancos de dados são estabelecidos para cada faculdade nas universidades de forma independente. Segundo os autores isso causa uma série de problemas como, por exemplo, inconsistência e redundância de dados, desperdício de recursos, etc.

A solução proposta pelos autores é um esquema de arquitetura de banco de dados distribuído nos sistemas de informação educacional multilocatários baseados em SaaS. Utilizando a tecnologia J2EE os autores também implementaram um módulo de processo de acesso ao banco de dados distribuído que por meio da análise e do ajuste da semântica do idioma de acesso ao banco de dados na camada de arquitetura do sistema redireciona o

destino de acesso ao banco de dados de cada inquilino. Assim sendo o módulo de processo analisa a instrução da consulta e processa os resultados da mesma com todo o processo sendo transparente para os inquilinos.

A plataforma proposta pelos autores é composta por três camadas: 1) camada de aplicativo: contém principalmente diferentes sistemas de aplicativos, vários inquilinos e manipulador de consultas. Na arquitetura, cada sistema de aplicativo pode ser usado por vários inquilinos e cada inquilino pode se registrar em vários sistemas de aplicativos. O gerenciador de consulta recebe a solicitação de acesso ao banco de dados do aplicativo e a envia para os servidores de destino após processá-la; 2) camada de processador: é o componente principal da estrutura. O analisador de configuração analisa as informações de configuração relacionadas e armazena os resultados em estruturas de dados específicas para que outros componentes possam usar os resultados facilmente. O processador de idioma do banco de dados repassa o idioma de acesso ao banco de dados e ajusta sua semântica de acordo com as informações de configuração; 3) camada de unidade de execução: o roteador de consulta e o gerenciador de fonte de dados. O roteador de consulta é responsável por enviar a instrução de acesso ao banco de dados para os servidores de banco de dados de destino. O gerenciador de fonte de dados gerencia as conexões entre aplicativos e bancos de dados.

Apesar de não apresentarem resultados de testes os autores afirmam que a plataforma de acesso a banco de dados distribuída baseada em SaaS proposta por eles atende aos requisitos de informações educacionais e a consideram prática, versátil, conveniente, segura e estável. Por fim eles apresentam três características principais da plataforma: 1) transparente para os inquilinos; 2) aprimora bastante a expansão do SaaS e evita a modificação dos códigos existentes na camada de lógica de negócios; 3) analisa o idioma de acesso ao banco de dados diretamente e fornece alta eficiência.

Os autores (JOOSEN, 2017) observam que no contexto de aplicativos SaaS multilocatários o suporte à confidencialidade de dados está cada vez mais sendo oferecido a partir da camada de aplicativo, em vez da camada de banco de dados, para acomodar os requisitos em constante mudança de vários inquilinos. Conforme os autores as plataformas de Middleware de gerenciamento de dados em nível de aplicativo estão se tornando cada vez mais atraentes para lidar com a complexidade de uma arquitetura de armazenamento em nuvem, bem como aplicativos SaaS multilocatários. Porém, na visão dos autores, essas plataformas normalmente oferecem suporte a estratégias tradicionais de mapeamento de dados que são criadas sob a suposição de um esquema de banco de dados fixo e rigoroso. Este cenário, apesar de oferecer suporte a diversos requisitos de confidencialidade de dados, leva à fragmentação dos dados em nós de armazenamento distribuído e isso introduz uma sobrecarga de desempenho significativa no nível de transações de banco de dados individuais e afeta negativamente a escalabilidade geral.

Com o objetivo de amenizar o problema descrito acima os autores apresentam uma estratégia de mapeamento de dados dedicada que alavanca a flexibilidade do esquema de dados dos bancos de dados NoSQL colunares para realizar criptografia de dados dinâmica e refinada de maneira mais eficiente e escalonável.

No contexto de um aplicativo SaaS multilocatário industrial os autores validaram a estratégia de mapeamento de dados proposta conduzindo uma avaliação de desempenho abrangente. Os resultados dessa avaliação, de acordo com os autores, confirmaram que a abordagem proposta realmente produz escalabilidade e melhorias de desempenho.

O objetivo dos autores (INDRANI, 2013) é melhorar ainda mais o tempo de processamento e a utilização do espaço do modelo de tabela compartilhada. Eles então apresentam um novo modelo de dados denominado "Modelo de Tabela Índice"(MTI) e comparam seu desempenho com o "Modelo de Tabela Chunk"(MTC). A abordagem dos autores inclui uma coluna de índice adicional na tabela base que faz referências cruzadas por meio do índice nas tabelas de suporte.

Para avaliar o desempenho da abordagem proposta os autores criaram dados aleatoriamente grandes para o MTC, a partir do qual construíram as tabelas para o MTI. Também compilaram três tipos de consultas utilizadas na avaliação: consulta tipo 0: consultas suportadas pelo MTC e pelo modelo proposto MTI; consulta tipo 1: consultas encontradas e coletadas na literatura relacionada; consulta tipo 2: consultas suportadas apenas pelo MTI.

Os resultados obtidos pelos autores mostraram que os tempos de processamento de consulta da MTI para consultas do tipo 0 melhoraram em uma ordem de magnitude. Já os tempos de processamento de consultas para consultas do tipo 1 são quase os mesmos. Os autores também compilaram um conjunto exaustivo de consultas do Tipo 2. Essas consultas não podem ser suportadas pelo MTC por causa de sua inflexibilidade e de seu design deficiente. Como o MTI é um esquema particionado verticalmente pode-se aproveitar as vantagens dos recursos de armazenamento de coluna. Isso é uma vantagem, pois a maioria das consultas para SaaS são colocadas em colunas individuais.

Finalmente os autores concluem que o MTI é mais expressivo do que o MTC e resulta em muitas melhorias no desempenho em termos de utilização de espaço por meio da compressão e é mais expressivo para suportar uma gama mais ampla de tipos de consulta.

Os autores (BRODT, 2011) contribuem com os recursos para que um Sistema de Gerenciamento de Banco de Dados (SGBD) ofereça suporte nativo ao gerenciamento de dados com reconhecimento de inquilino. Eles apresentam inquilinos como objetos de banco de dados de primeira classe e propõem o conceito de um contexto de inquilino para isolar um inquilino de outros inquilinos. Além disso, apresentam também um conceito

de herança de esquema que permite o compartilhamento de um esquema de aplicativo principal entre inquilinos, ao mesmo tempo em que permite extensões de esquema por inquilino.

A abordagem proposta pelos autores, segundo os mesmos, facilita o desenvolvimento de aplicativos SaaS multilocatários que dependem de um SGBD devido a uma série de fatores descritos a seguir: 1) evita a implementação de um reescritor de consulta complexo sobre o SGBD; 2) facilita a manutenção central do esquema principal do aplicativo e a manutenção individual dos esquemas dos inquilinos; 3) como o SGBD conhece e controla os esquemas principais do aplicativo, bem como os esquemas dos inquilinos, ele pode otimizar o acesso e processamento de dados; 4) evita redundância, pois compartilha o esquema central do aplicativo entre os inquilinos.

Para a implementação de um protótipo da abordagem apresentada para suporte a multilocação nativa em um SGBD os autores optaram por utilizar o PostgreSQL 8.4 como ponto de partida, pois, segundo eles, o PostgreSQL está disponível como código aberto, apresenta um design limpo e seu código-fonte é bem documentado.

Baseados nos resultados dos testes que foram conduzidos através do protótipo os autores afirmam que o compartilhamento do esquema central do aplicativo diminui consideravelmente o consumo de memória principal e os tempos de pesquisa do dicionário de dados em comparação com esquemas totalmente dedicados por inquilino.

Geralmente, como explicam os autores (WANG, 2020), um banco de dados habilitado para multilocação é implementado em cima do banco de dados SQL convencional e a Tabela Universal (TU) é uma das técnicas de implementação mais comumente usadas. Essa técnica visa o mapeamento de todos os dados do inquilino em uma única tabela e em nível de aplicativo esse mapeamento de esquema é implementado pela tradução de instruções SQL.

A proposta do autores é um método que suporta a tradução da sintaxe SQL completa, incluindo a linguagem de manipulação de dados (DML) e a linguagem de definição de dados (DDL). Eles se baseiam na tradução direcionada por sintaxe na gramática livre de contexto da sintaxe SQL e implementam o método como uma ferramenta de tradução automática baseada no framework ANTLR. Esse framework traduz instruções DDL e DML SQL comuns da tabela de origem lógica para a TU com apenas a predefinição de algumas metainformações necessárias.

A fim de avaliar a eficácia, a eficiência, o desempenho e a simultaneidade do método proposto uma série de experimentos foram conduzidos pelos autores e os resultados, segundo os mesmos, demonstraram que o método é escalonável para a quantidade de inquilinos e o tamanho do banco de dados.

A técnica de mapeamento de esquema denominada Tabela Universal é, de acordo

com os autores (CHEN, 2016), comumente usada em ambientes SaaS para projetar e realizar o mapeamento de esquema de vários inquilinos. Apesar disso os autores observam que pouca pesquisa tem sido dedicada ao design e realização de um esquema de reescrita de consulta para a Tabela Universal. Por essa razão os autores apresentam uma coleção de esquemas de reescrita de consulta geral para Tabela Universal que podem transformar de forma transparente consultas lógicas específicas do inquilino em consultas físicas correspondentes.

Para verificar a viabilidade da abordagem proposta os autores desenvolveram um Middleware de mapeamento de esquema baseado em Java e um aplicativo Saas de compra online. Os autores então conduziram uma série de experimentos e concluíram que os resultados estavam de acordo com suas previsões analíticas demonstrando que os esquemas desenvolvidos são escaláveis para o número de inquilinos e o número de conexões de banco de dados simultâneas.

Os autores (MEINEL, 2013) reconhecem que graças a natureza do SaaS e da nuvem em geral, onde os dados estão fora do controle do usuário, a privacidade e a segurança são considerados fatores vitais e as maiores preocupações na adoção da computação em nuvem. Em aplicativos SaaS multilocatários, conforme os autores, os inquilinos se preocupam com a confidencialidade de seus dados, uma vez que vários inquilinos compartilham a mesma infraestrutura. Inevitavelmente algumas questões surgem, como, por exemplo, como proibir um inquilino de acessar os dados de outros?; Como garantir que os dados do inquilino estejam disponíveis apenas para usuários autenticados?

Visando sanar as questões acima mencionadas os autores apresentam um mecanismo, projetado para facilitar o processo de proteção de dados armazenados na nuvem, denominado SignedQuery. A tecnologia garante a confidencialidade dos dados evitando que qualquer inquilino acesse os dados de outros inquilinos de forma acidental ou maliciosa. SignedQuery assina a solicitação do inquilino para que o servidor possa reconhecer o solicitante e garantir que os dados a serem acessados pertençam a este inquilino. Ao interceptar os objetos de solicitação HTTP na rede interna do inquilino, SignedQuery cria a assinatura e a anexa aos cabeçalhos da solicitação. Em seguida, envia a solicitação ao provedor SaaS onde a assinatura é validada.

SignedQuery foi implementado como um proxy HTTP instalado na rede interna do inquilino e no lado do provedor de SaaS. Para a realização dos experimentos os autores utilizaram um aplicativo denominado OrangeHRM. Conforme os mesmos esse é o aplicativo de gerenciamento de recursos humanos de código aberto mais popular do mundo, com mais de um milhão de usuários globalmente e mais de 600.000 downloads.

Os resultados dos testes demonstraram que a abordagem proposta pelos autores é, de fato, viável e incorre em uma sobrecarga desprezível.

## 4.2 Propostas de Migração

Identificamos onze trabalhos que abordam a questão da migração de aplicativos de inquilino único para aplicativos SaaS multilocatários. Dois desses trabalhos apresentam abordagens que dispensam a reescrita do código-fonte original (CAI; WANG; ZHOU, 2010; HOHENSTEIN; KOKA, 2016). Outros dois trabalhos relatam abordagens que, ao contrário dos dois primeiros, utilizam técnicas de reengenharia para ativar o recurso de multilocação nos aplicativos (BEZEMER; ZAIDMAN, 2010; BEZEMER et al., 2010). Dois trabalhos propõem o desenvolvimento de frameworks para realizar a migração dos aplicativos (IBRAHIM NUGRAHENI, 2013; ALMORSY; GRUNDY, 2012). Três trabalhos relatam métodos próprios para migrar os aplicativos (CHEN et al., 2010; SAKAMOTO, 2009; AN et al., 2014). Um trabalho apresenta uma arquitetura de migração (DU; ZHENG, 2014). Um último trabalho propõe um método para que os bancos de dados possam oferecer suporte a multilocação (CAO et al., 2012).

| **Tipo de pesquisa** | **Trabalhos de referência** | **#Trabalhos** |
|---|---|---|
| Propostas que dispensam reengenharia | Cai; Wang; Zhou, 2010; Hohenstein; Koka, 2016 | 2 |
| Propostas que utilizam reengenharia | Bezemer; Zaidman, 2010; Bezemer et al., 2010 | 2 |
| Framework | Nugraheni, 2013; Almorsy; Grundy; Ibrahim, 2012 | 2 |
| Métodos próprios | Chen et al., 2010; Sakamoto, 2009; An et al., 2014 | 3 |
| Arquitetura | Du; Zheng, 2014 | 1 |
| Banco de dados | Cao et al., 2012 | 1 |

Tabela 5 – Propostas de migração: Autores e a quantidade de trabalhos realizados por assunto

### 4.2.1 propostas que dispensam reengenharia

Os autores (CAI, 2010) descrevem uma abordagem para transformar aplicativos Web existentes em aplicativos com suporte a multilocação hospedados em uma nuvem pública, sem que para isso seja necessário reescrever o código-fonte original.

Segundo os autores a tipologia dos aplicativos Web modernos tem a seguinte configuração: um cluster de servidor Web, um cluster de servidor de aplicativos Web, um servidor de banco de dados, um servidor LDAP (Lightweight Directory Access Protocol - um protocolo para acessar e manter serviços de informação de diretório distribuído) e às vezes um cluster de cache distribuído.

Para que os aplicativos SaaS sejam habilitados para multilocação, os autores afirmam haver duas necessidades principais: o isolamento e a personalização. Segundo os

mesmos autores, o isolamento é a base de todas as tecnologias de ativação. Um ponto de isolamento em um aplicativo Web, por exemplo, significa que neste aplicativo existe uma classe, um método ou um campo cujo comportamento ou valor é específico de um inquilino. Esta classe, método ou campo, portanto, precisa ser isolado para um inquilino específico. A personalização, por sua vez, fornece ao inquilino comportamentos exclusivos.

Outra exigência para fazer com que os aplicativos Web existentes se tornem aplicativos de multilocação, de acordo com os autores, é a necessidade de propagar as informações de contexto do inquilino para os sistemas distribuídos que oferecem recursos remotos que os aplicativos Web usarão. Para que isso seja possível é necessário um modelo conceitual que possa capturar os artefatos mais críticos e comuns para habilitar a multilocação. O modelo conceitual de multilocação SaaS proposto pelos autores inclui o seguinte: interceptor multilocatário, contexto de inquilino, mapa compartilhado para armazenar informações de inquilino, lógica de aplicativo original (usando threads), Java Virtual Machine (JVM) ou APIs de contêiner da Web e servidores remotos.

O funcionamento do modelo conceitual descrito acima, como explicam os autores, se dá da seguinte forma: o interceptor de multilocação pode ser implementado com um filtro da Web e capturar informações do inquilino sabendo a qual inquilino esta solicitação da Web se destina. O contexto do inquilino é um objeto para armazenar informações do inquilino e pode ser vinculado a um encadeamento e usado pelos códigos de lógica de negócios do aplicativo. A chamada de JVM e APIs de contêiner da Web será interceptada e adicionada com alguma pré ou pós-operação específica de multilocação. As informações de contexto do inquilino também podem ser propagadas por um cliente (por exemplo, cliente JDBC) de serviço remoto para o servidor (servidor de banco de dados) desse serviço.

Os autores não apresentam nenhum tipo de avaliação ou demonstração da abordagem proposta.

Os autores (HOHENSTEIN; KOKA, 2016) se concentram na migração de aplicativos legados de um único inquilino para aplicativos totalmente multilocatários. Eles explicam que embora haja algumas abordagens e metodologias que demonstram como converter aplicativos legados em software multilocatário, elas geralmente exigem a reengenharia do código-fonte legado. A abordagem proposta pelos autores, no entanto, consiste em simplesmente adicionar componentes ao aplicativo legado, sem tocar explicitamente no código-fonte do aplicativo.

Os autores basearam a abordagem na linguagem AspectJ, uma extensão da linguagem Java que introduz um novo conceito de aspecto no Java. Um aspecto altera a estrutura dinâmica de um programa ao interceptar certos pontos do fluxo do programa (pontos de junção). Esses pontos de junção podem ser chamados de métodos ou construtores, execuções, acessos a campos e exceções, etc. A proposta dos autores é que vários

componentes sejam implementados como aspectos no AspectJ e adicionados ao arquivo WAR do aplicativo.

A abordagem dos autores, segundo os mesmos, se preocupou em atingir três questões principais: 1) isolamento do inquilino para diferentes estratégias e servidores de banco de dados; 2) personalização de comportamento específico do inquilino; 3) monitoramento das atividades dos usuários dos inquilinos para fins de cobrança.

A validação da abordagem foi feita pelos autores através de um aplicativo Java industrial existente que fornece aos clientes serviços REST. O aplicativo é executado em um servidor de aplicativos Tomcat e usa um banco de dados Oracle para o armazenamento de dados.

De acordo com os autores os serviços REST são mais fáceis de manipular do que os aplicativos com uma interface gráfica, pois há código Java puro sem nenhuma parte em HTML ou Javascript.

### 4.2.2 propostas que utilizam reengenharia

Alguns dos principais desafios da implementação de multilocação são identificados pelos autores (BEZEMER; ZAIDMAN, 2010) que, além disso, apresentam uma abordagem de reengenharia conceitual para oferecer suporte à migração de aplicativos de um único inquilino para aplicativos de vários inquilinos. As três principais camadas da abordagem proposta são: 1) autenticação; 2) configuração; 3) banco de dados.

A camada de autenticação fornece um mecanismo para identificar cada inquilino no aplicativo gerando um ticket de inquilino (token) após um login bem sucedido. Esse token pode ser usado em todo o aplicativo para carregar a configuração correspondente para um inquilino.

A camada de configuração se subdivide em outras quatro: 1) layout: permite o uso de temas específicos para cada inquilino; 2) configuração geral: é a configuração específica do inquilino como, por exemplo, configurações de banco de dados, configurações de chave de criptografia e detalhes de perfil pessoal; 3) entrada e saída (E/S) de arquivo: permite a especificação de caminhos de arquivo específicos do inquilino que podem ser usados para, por exemplo, geração de relatório; 4) fluxo de trabalho: permite a configuração de fluxos de trabalho específicos do inquilino. Um exemplo de um aplicativo no qual a configuração do fluxo de trabalho é necessária é um aplicativo de planejamento de recursos corporativos (ERP), no qual o fluxo de trabalho das solicitações pode variar significativamente para diferentes empresas.

A camada de banco de dados, por sua vez, é de extrema importância, pois, segundo os autores, a diferença mais importante entre um aplicativo de inquilino único e um aplicativo multilocatário é o gerenciamento e isolamento de dados. Os autores afirmam

que os SGBDs atuais não são capazes de lidar com multilocação por si próprios e que isso deve ser feito em uma camada entre a lógica de negócios e o pool de banco de dados do aplicativo.

Como estudo de caso os autores transformaram o sistema wiki ScrewTurn (um aplicativo wiki de código escrito em C# e baseado na plataforma ASP.NET 3.5) em uma versão multilocatária e comprovaram que a abordagem por eles proposta é aplicável e executável.

Os autores (BEZEMER, 2010) relatam suas experiências com a reengenharia de um sistema de software industrial de inquilino único em um multilocatário utilizando uma abordagem de reengenharia leve. O sistema em questão é o Codename, desenvolvido por uma empresa de software holandesa chamada Exact. A empresa é especializada em software de planejamento de recursos empresariais (ERP), gestão de relacionamento com o cliente (CRM) e administração financeira.

O padrão de reengenharia utilizado pelos autores, de acordo com os mesmos, é um processo orientador que permite transformar rápida e eficientemente um aplicativo de um único inquilino em um multilocatário. O padrão, ainda segundo os autores, fornece recursos para estilos de layout específicos do inquilino, configuração e gerenciamento de dados.

O estudo conduzido pelos autores demonstrou alguns pontos positivos com relação à abordagem proposta: 1) é uma abordagem leve uma vez que a implementação foi feita em cerca de 100 linhas de código e levou apenas cinco dias para ser finalizada; 2) é uma abordagem transparente, pois a aparência do aplicativo não precisa ser alterada ec o usuário final não sabe que o aplicativo é multilocatário; 3) a abordagem não exige que os desenvolvedores envolvidos no projeto sejam treinados em multilocação, pois as alterações no código são mínimas e limitadas a algumas pequenas partes.

O padrão de reengenharia de multilocação proposto pelos autores leva em consideração alguns dos principais aspectos da multilocação: 1) Possibilidade de compartilhamento de recursos de hardware, possibilitando redução de custos; 2) Alto grau de configurabilidade, permitindo que cada cliente crie sua própria aparência e fluxo de trabalho; 3) Um aplicativo compartilhado e instância de banco de dados, facilitando a manutenção.

Com relação aos objetivos do padrão proposto os autores destacam dois principais: 1) migrar um aplicativo de inquilino único para um aplicativo multilocatário com pequenos ajustes na lógica de negócios existente; 2) separar claramente os componentes de multilocação, para que os mecanismos de monitoramento e balanceamento de carga possam ser integrados futuramente.

### 4.2.3 Framework

A autora (NUGRAHENI, 2013) apresenta um framework para migrar um aplicativo Web de inquilino único denominado SIMA para um aplicativo multilocatário. O SIMA é responsável por registrar o inventário dos bens públicos do governo da Indonésia e, de acordo com a autora, sua reengenharia para o modelo de multilocação é necessária para que várias agências governamentais possam utilizá-lo sem que para isso seja necessário que cada agência mantenha sua própria infraestrutura.

A configuração de um aplicativo multilocatário exige uma lógica de negócios muito mais complexa do que um aplicativo de inquilino único. No caso do SIMA, no entanto, a autora observa que devido ao fato do sistema ter adotado as regulamentações governamentais relativas ao sistema de inventário de bens públicos, a configuração necessária para a lógica de negócios é bastante semelhante entre os inquilinos.

A abordagem proposta pela autora apresenta as seguintes fases: 1) um novo módulo para lidar com os processos de registro dos inquilinos e seus usuários deve ser desenvolvido; 2) melhoria do sistema de banco de dados para permitir a geração de um banco de dados para cada inquilino registrado; 3) uma camada de personalização adicional deve ser adicionada à medida que os usuários se registram no sistema. Cada inquilino que efetua login no aplicativo terá um "tenant-id"que pode ser usado para configuração e personalização do aplicativo; 4) fornecer a cada inquilino uma página para configurar e personalizar alguns componentes da interface, como, por exemplo, logotipo, cor, aparência, etc.; 5) desenvolvimento de um painel para que o administrador do sistema possa monitorar o desempenho e o status de cada inquilino.

A investigação dos aspectos de segurança do isolamento de dados será conduzida pela autora em um trabalho futuro. Esse tópico, segundo a mesma, é de extrema importância, uma vez que as violações de segurança podem levar ao vazamento de dados dos inquilinos. Outra pretensão da autora é realizar um estudo de comparação entre o uso do banco de dados compartilhado e a abordagem multiesquema com o intuito de descobrir qual tecnologia se mostra mais eficiente para o armazenamento de dados.

A reengenharia de aplicativos para que possam oferecer suporte a multilocação na visão dos autores (ALMORSY, 2012) é uma tarefa complexa e desafiadora por requerer um conhecimento profundo do aplicativo em questão e, além disso, exigir que quase todos os módulos do sistema sejam revisados. Com o objetivo de auxiliar os engenheiros de software no processo de migração de seus aplicativos os autores apresentam o SMURF, um framework baseado no conceito de reaspectos. Um reaspecto captura a assinatura de entidades do sistema a serem modificadas em termos de restrições OCL (ações a serem aplicadas, incluindo inserir, substituir, modificar e excluir código) e o código a ser aplicado. O SMURF então utiliza as assinaturas desses reaspectos para aplicar as ações especificadas

para cada um deles.

Para validar o SMURF os autores selecionaram um conjunto de cinco aplicativos de código aberto desenvolvidos na linguagem .NET e tamanhos diferentes. São eles: Galactic, PetShop, SplendidCRM, NopCommerce e BlogEngine.

Os resultados, de acordo com os autores, mostram que o SMURF foi aplicado com sucesso para migrar a maioria dos aplicativos envolvidos nos experimentos. O desempenho do tempo (em milissegundos) total gasto pelo SMURF para localizar possíveis correspondências das assinaturas especificadas também foi avaliado: Galactic (8), PetShop (5), SplendidCRM (90), NopCommerce (205), BlogEngine (15).

Por fim os autores destacam uma diferença na abordagem proposta para as abordagens tradicionais. Segundo os autores o SMURF se concentra na atualização do aplicativo original para oferecer suporte a multilocação real em vez de usar plataformas SaaS que apenas simulam a multilocação, o que resulta em travamento do aplicativo.

### 4.2.4 Métodos próprios

Devido à falta de abordagens e ferramentas de migração adequadas, os autores (CHEN, 2010) propõem um método sistemático para migrar e evoluir aplicativos Web de locação única em aplicativos habilitados para multilocação. De acordo com os autores a abordagem proposta se baseia principalmente em três aspectos: 1) migração do modelo de dados: este aspecto se subdivide em outros dois: 1.1) alteração do banco de dados (BD) do aplicativo legado: o BD legado armazena os dados de um único inquilino, devendo ser estendido para conter os dados de vários inquilinos. A segurança e o isolamento dos dados são tópicos que devem ser levados em consideração; 1.2) Criação do banco de dados de inquilinos: é a base do gerenciamento de inquilinos. Armazena não apenas as informações básicas do inquilino, mas também o endereço do esquema específico (fonte de dados) de cada um. Assim um inquilino pode localizar seu próprio esquema no banco de dados do aplicativo para acessar seus dados de negócios; 2) migração de controle de acesso: quando um aplicativo da web legado é transformado em multilocatário as políticas de controle de acesso originais devem ser ajustadas. O controle de acesso aprimorado foi elaborado a partir de três aspectos: autenticação de identidade, controle de acesso a dados e controle de acesso de escopo global; 3) gerenciamento de inquilinos: sua função básica é adicionar, excluir, atualizar e consultar inquilinos. Quando um novo inquilino é inserido no banco de dados do inquilino, um novo esquema no banco de dados do aplicativo é criado para armazenar os dados de negócios do inquilino. Essas duas etapas devem ser consideradas como uma única transação e não podem ser separadas.

Um experimento foi realizado pelos autores visando ajustar a abordagem e avaliar a aplicabilidade e o impacto no desempenho do método de migração. Os mesmos acreditam

que para os aplicativos legados com arquitetura semelhante a ferramenta pode fornecer diretrizes e transformação semiautomática para multilocação.

O autor (SAKAMOTO, 2009) descreve como se deu a transformação de um aplicativo pertencente à empresa Fujitsu denominado “Internet Navigware” da versão de inquilino único para SaaS multilocatário.

O “Internet Navigware”, conforme o autor, está disponível no Japão desde 1998, mas o trabalho em sua conversão para um aplicativo SaaS começou no início de 2008. Trata-se de um aplicativo de gerenciamento de treinamento que oferece diversos recursos, desde o desenvolvimento de materiais didáticos até o aprendizado e gerenciamento de resultados. Além das funções de aprendizagem eletrônica o aplicativo oferece diversas outras funções, como, por exemplo, gerenciamento de treinamento em grupo, quadros de avisos e funções de comunicação para perguntas e respostas.

Para a conversão do "Internet Navigware"em um aplicativo SaaS levou-se em consideração três motivos principais que segundo o autor são os seguintes: 1) o desejo dos clientes de utilizar os serviços em formato SaaS como uma alternativa aos serviços baseados em intranet; 2) o interesse da Fujitsu em possuir uma fonte estável de receita uma vez que os clientes SaaS podem ser cobrados mensalmente de acordo com o uso do aplicativo; 3) os recursos de desenvolvimento podem ser consolidados. Em sistemas tradicionais os desenvolvedores devem oferecer suporte a vários sistemas operacionais e bancos de dados. No caso do SaaS, no entanto, eles precisam oferecer suporte a apenas um sistema operacional e um banco de dados, o que reduz as horas de trabalho envolvidas no desenvolvimento e nos testes.

O formato de se ter várias organizações compartilhando um servidor, conforme o autor, já era utilizado pelo "Internet Navigware". Por essa razão sua conversão para SaaS não exigiu a implementação de uma versão completamente nova. O próprio aplicativo forneceu uma base para um sistema multilocatário.

Finalmente o autor chama atenção para um recurso diferenciado na nova versão do "Internet Navigware". Geralmente nos sistemas SaaS os usuários são cobrados somente pelo que, de fato, utilizaram do sistema. No "Internet Navigware", no entanto, existe a opção do cliente definir um orçamento mensal fixo. Esse é um recurso interessante para clientes que não podem exceder determinado valor.

Para habilitar o recurso de multilocação em um aplicativo é necessário, conforme os autores (AN, 2016), encontrar e processar um tipo especial de entidades de dados chamadas Pontos de Isolamento Global (PIGs). Porém é um verdadeiro desafio encontrar todos os PIGs de um aplicativo uma vez que o método tradicional de se fazer isso é navegar manualmente no código do aplicativo, exigindo muito esforço humano.

Com o intuito de ajudar a localizar e processar os PIGs de um aplicativo os autores

apresentam um kit de ferramentas denominado Auto-MT. Eles afirmam que o Auto-MT é capaz de encontrar novos PIGs com base em suas relações com PIGs conhecidos.

Para demonstrar a eficácia do Auto-MT os autores o implementaram como um plugin do Eclipse e o utilizaram para transformar um aplicativo real para ser habilitado para multilocação. O aplicativo escolhido foi um servidor de blog da Web de código aberto desenvolvido pela Apache Software Foundation denominado Roller. O aplicativo é usado por centenas de empresas e organizações como, por exemplo, a Oracle e a IBM.

Os resultados experimentais obtidos pelos autores demonstraram que o Auto-MT, de fato, economiza esforço humano substancial e acelera o processo de transformação de aplicativos para serem habilitados para multilocação.

### 4.2.5 Arquitetura

Os autores (ZHENG J.; DU, 2014) propõem uma arquitetura de serviço para migrar facilmente aplicativos cliente-servidor convencionais para o modelo SaaS multilocatário de computação em nuvem. Na visão dos autores esse modelo está se tornando a tendência da nova geração de desenvolvimento de software devido ao seu baixo investimento, flexibilidade e acessibilidade.

Utilizando o mecanismo de nuvem Amazon EC2 para implementar e verificar a arquitetura de serviço proposta, denominada serviços A2SF, os autores citam as quatro partes que compõem a mesma: 1) serviços de reconhecimento de inquilino; 2) serviço de gerenciamento de inquilino; 3) serviço de implantação automática de aplicativo; 4) serviço de gerenciamento de recursos em nuvem.

Um aplicativo cliente-servidor do mundo real, denominado SugarCRM, foi implantado no Amazon EC2 com base em um protótipo A2SF implementado. O SugarCRM é um dos aplicativos de gerenciamento de relacionamento com o cliente mais populares atualmente. Ele é implementado na linguagem de programação PHP e oferece suporte a vários tipos de bancos de dados, incluindo MySQL.

Uma série de experimentos sobre a latência de tempo médio do serviço A2SF foi conduzida pelos autores que mediram os seguintes segmentos de tempo: o primeiro tempo de resposta, o tempo médio da resposta normal, o tempo de processamento do proxy de serviço e o tempo de processamento do proxy de dados. A medição foi realizada através do uso de 10 computadores com endereços IP diferentes que foram escolhidos para simular 10 inquilinos.

Os autores chegaram as seguintes conclusões: 1) a latência de tempo de uso do A2SF é de cerca de 70 milisegundos (ms); 2) no melhor caso o primeiro tempo de resposta do sistema SaaS migrado foi de 3417 ms e no pior caso foi de 65741 ms; 3) o tempo médio de geração da instância do aplicativo está relacionado ao tamanho do pool da

máquina virtual e ao tamanho do aplicativo de destino. Adicionar um pool de máquina virtual melhoraria a eficiência da geração de instância do aplicativo; 4) quando o tamanho do pool de máquina virtual em A2SF foi definido como 2, o tempo médio da primeira resposta tornou-se 39.705,6 ms, o que é significativamente menor do que o tempo de primeira resposta do pior caso; 5) o tempo médio de processamento do proxy de serviço é de cerca de 65 ms, já o tempo médio de processamento do proxy de dados é de cerca de 27 ms. Sob a mesma capacidade de computação, o tempo de processamento do proxy de serviço e o tempo de processamento do proxy de dados são, portanto, estáveis.

### 4.2.6 Banco de dados

Transformar um banco de dados de locação única em um modelo de multilocação é fundamental na grande maioria dos aplicativos SaaS migrados. Por essa razão os autores (CAO, 2012) propõem um método para migrar um modelo de dados de locação única para um grupo de modelo de dados de multilocação configurável que atende às necessidades de personalização do modelo de dados e obtém segurança e sistema de banco de dados escalonável.

O método de migração de dados proposto pelos autores é composto por quatro etapas que são descritas pelos mesmos da seguinte forma: 1) implementação de personalização do modelo de dados: essa etapa se subdivide em outras três: 1.1) modificar todas as tabelas adicionando a coluna "TenantID"e aplicar a nova estrutura a todas as instâncias; 1.2) definir um nome único como "TableAddr"para cada serviço de banco de dados; 1.3) criar uma nova instância de banco de dados chamada "TenantDB"e uma tabela de dados chamada "Tenant Management Table"; 2) Acesso ao banco de dados multilocatário: o procedimento de migração consiste em duas partes: 2.1) como as informações personalizadas do banco de dados para inquilinos são armazenadas no "TenantDB", a primeira etapa da transformação na camada de código é adicionar códigos para concluir o acesso ao "TenantDB"; 2.2) encontrar o endereço de dados personalizados usando as funções na etapa 2.1 e, em seguida, modificar os códigos para voltar à fonte de dados correta e executar um comando real nela; 3) implementação de segurança: a primeira camada é a parede de filtragem de informações do inquilino e a outra é uma camada de validação de autoridade; 4) implementação de escalabilidade: a escalabilidade geralmente depende da configurabilidade. Como o Grupo de Modelo de Dados de Multilocação Configurável alcançou a configurabilidade em nível de tabela os autores apenas precisaram se dedicar à configurabilidade em nível de banco de dados.

Não havendo um método de avaliação apropriado para determinar a viabilidade de uma abordagem de migração, os autores utilizaram um aplicativo B2C baseado na Web como um estudo de caso para mostrar as etapas realizadas pelos métodos de migração do banco de dados multilocatário. O sistema B2C é um sistema de informação da Web

implementado através dos frameworks de código aberto Struts, Spring e Hibernate. Jboss e Jetty atuaram como o servidor de aplicativos e servidor Web, enquanto o sistema de banco de dados original é o Oracle 11g. Os autores o migraram para um ambiente de multilocação com uma variedade de modelos de dados.

De acordo com os resultados obtidos os autores chegaram à conclusão que o método por eles proposto pode atender às necessidades de personalização do modelo de dados e obter segurança e sistema de banco de dados escalonável.

# 5 Evidências das Vantagens e/ou Desvantagens

Neste capítulo respondemos a última questão de pesquisa abordada no capítulo três: quais são as evidências das vantagens e desvantagens do uso da multilocação?

Foram identificados seis trabalhos que abordam as vantagens e desvantagens dos aplicativos SaaS multilocatários. Um trabalho faz uma comparação entre os modelos de receita SaaS (OJALA, 2013). Um trabalho avalia os principais padrões de implementação de multilocação na camada de dados (AN et al., 2008). Outro trabalho compara algumas técnicas de implementação de esquemas flexíveis para SaaS (AULBACH et al., 2009). Dois trabalhos avaliam algumas opções de implementação de multilocação (KREBS; MOMM, 2011; BHUVANESWARI; SARASWATHI, 2014). Um último trabalho relata algumas previsões sobre o futuro do SaaS - Big SaaS (BAYLEY et al., 2015).

| **Tipo de pesquisa** | **Trabalhos de referência** | **#Trabalhos** |
|---|---|---|
| Modelos de receita SaaS | Ojala, 2013 | 1 |
| Implementação de multilocação na camada de dados | An et al., 2008 | 1 |
| Esquemas flexíveis para SaaS | Aulbach et al., 2009 | 1 |
| Opções de implementação de multilocação | Krebs; Momm, 2011; Bhuvaneswari; Saraswathi, 2014 | 2 |
| Big SaaS | Bayley et al., 2015 | 1 |

Tabela 6 – Vantagens e desvantagens do SaaS: Autores e a quantidade de trabalhos realizados por assunto

## 5.1 Modelos de receita SaaS

A computação em nuvem, na visão do autor (OJALA, 2013), mudou a forma como o software é vendido, entregue e usado. No modelo SaaS os clientes podem acessar o software online em vez de instalá-lo em seus computadores. A vantagem disso é que eles sempre utilizarão a versão mais recente do software e não será mais necessário se preocupar com as especificações técnicas ou capacidade de armazenamento do computador. Esse modelo, no entanto, também pode causar riscos relacionados à segurança dos dados quando os mesmos são armazenados em um servidor de nuvem pública ou quando os aplicativos exigem uma conexão confiável com a Internet.

O autor se dispõe a apresentar uma visão geral dos dois principais modelos de receita SaaS: 1) pagamento por uso; 2) aluguel de software. Além disso, o autor ainda

foca nas vantagens e desvantagens para provedores de SaaS e seus clientes e identifica o modelo de receita mais eficaz para situações específicas.

Para explicar o exposto acima o autor tenta responder as seguintes questões: 1) quando os fornecedores de software devem manter seu modelo de licenciamento tradicional em vez de oferecer o SaaS? 2) qual modelo de receita SaaS é mais lucrativo: aluguel ou pagamento por uso? 3) para os clientes, quais são as compensações entre o licenciamento tradicional, o aluguel e o pagamento por uso?

Os softwares, segundo o autor, podem ser vendidos tanto na forma material (CD, DVD) como na forma intangível pela Internet. Assim sendo, eles podem ser utilizados com diversos modelos de receita. O termo modelo de receita é usado pelo autor para se referir a como uma empresa coleta receita de seus clientes. O termo, portanto, está relacionado às várias opções que uma empresa pode oferecer aos clientes que desejam comprar seu software. O autor, no entanto, se concentra na análise, como já dito anteriormente, dos dois principais modelos que podem ser fornecidos por SaaS: pagamento por uso e aluguel de software. Além disso, o autor também apresenta uma comparação entre esses dois modelos de receita e o modelo tradicional.

No modelo de receita pagamento por uso, segundo o autor, o cliente é cobrado de acordo com as unidades por ele utilizadas. A unidade medida pode ser baseada no período de tempo que o software está em execução, o número de vezes que uma sub-rotina principal é chamada, o número de transações tratadas ou alguma combinação destes. Este modelo permite que o fornecedor disponibilize seu produto para clientes sem recursos financeiros para comprar uma licença de software tradicional. O modelo também é adequado quando o cliente precisa do software apenas ocasionalmente ou para algum propósito específico.

Outra vantagem considerada pelo autor é em relação à pirataria de software. Segundo ele a execução de software em um servidor em nuvem torna a prática quase impossível.

O modelo de pagamento por uso também apresenta desvantagens para os fornecedores de software. Duas delas são apontadas pelo autor: 1) como os clientes não assinam contratos de longo prazo para utilizar o software, se torna barato e fácil mudar de fornecedor caso apareçam aplicativos alternativos a um preço mais baixo; 2) com os preços baseados na utilização do software, o fornecedor pode precisar manter um documento detalhado sobre o uso de cada cliente, aumentando o trabalho administrativo e exigindo um processo confiável para a entrega de tais documentos.

Analisando as vantagens do ponto de vista do cliente o autor afirma que para empresas menores, sem orçamento para o tipo de investimento exigido pelo licenciamento de software tradicional, o modelo de pagamento por uso é uma boa opção. Assim como também é mais vantajoso para o cliente que necessita do software apenas ocasionalmente,

uma vez que é possível testar e avaliar o software para ver se ele atende às necessidades da empresa. Como já dito, é mais fácil mudar de fornecedor se a qualidade ou funcionalidade do software não for adequada. Outra vantagem é que o cliente não precisa instalar, manter e atualizar o software o que diminui a necessidade de pessoal de TI.

As desvantagens para o cliente com relação ao modelo de pagamento por uso, na visão do autor, estão relacionadas aos seguintes fatores: 1) preço do software: o preço, na maioria dos casos, é fixo. Portanto não há negociações de preço com o fornecedor do produto; 2) estimativa da necessidade real de se utilizar o software: é difícil estimar com antecedência quanto o software será realmente usado. Outro modelo de receita, portanto, pode ser mais apropriado se o software for utilizado de forma mais contínua; 3) segurança dos dados: se os dados de uma empresa são armazenados em uma nuvem pública, a empresa pode não saber exatamente onde os dados estão sendo armazenados ou se estão sendo misturados com outros dados; 4) permanência do provedor do software no mercado: a existência contínua do provedor de nuvem é vital. Se o provedor desaparecer do mercado, o efeito sobre o cliente será imediato e possivelmente catastrófico.

Outro modelo de receita abordado pelo autor é o modelo aluguel de software. Nesse modelo, de acordo com o mesmo, o cliente paga uma taxa de assinatura negociada para usar a licença de software por um tempo limitado. Para o fornecedor o aluguel de software oferece mais flexibilidade de preços do que o modelo de pagamento por uso. O preço do aluguel pode ser baseado na duração do contrato, no número de usuários na organização do cliente, nas funcionalidades do software ou no tamanho da empresa. Portanto o software pode ser mais barato para clientes menores do que para empresas maiores.

O modelo de aluguel, ainda conforme o autor, é mais fácil de aplicar do que o modelo de pagamento por uso que exige o monitoramento constante da utilização do software. Além disso, no longo prazo, o modelo de aluguel pode gerar mais receita do que os outros modelos, desde que o fornecedor mantenha relacionamentos leais com o cliente.

Com relação às desvantagens do modelo de aluguel de software para o fornecedor o autor explica que são bastante semelhantes às do modelo de pagamento por uso. No entanto, no modelo de aluguel não há a necessidade de se manter registros auditáveis da utilização do software.

Do ponto de vista do cliente o aluguel de software oferece mais possibilidades de negociações de preços e de variação dos termos de utilização e duração do contrato. Os custos totais do software também são previsíveis e definidos contratualmente. O cliente, portanto, sabe com antecedência quais recursos financeiros devem ser alocados durante o contrato.

Mais uma vez as desvantagens são semelhantes às do modelo de pagamento por

uso, a diferença é que no modelo de aluguel o cliente paga utilizando ou não o software.

Além dos dois modelos de receita de software SaaS o autor ainda apresenta o modelo tradicional no qual o software é comumente vendido como pacote ou licenciamento baseado em servidor. A diferença é que no licenciamento por pacote um cliente compra uma única licença para um único usuário ou computador. Já no licenciamento baseado em servidor o software é comprado para um determinado número de processadores que o executam. Sendo assim, o software só pode ser executado em um número limitado de computadores.

No modelo tradicional, conforme o autor, os clientes geralmente devem pagar, além de uma taxa de licença inicial, uma taxa de manutenção que inclui suporte técnico e atualizações de versão.

Os fatores que devem ser levados em consideração para que a escolha do modelo de receita seja o mais adequado são apresentados pelo autor. O mesmo afirma que o mais importante deles é o tamanho da empresa. Empresas maiores podem estar mais dispostas a comprar uma licença de software e usar o software em seu próprio data center, especialmente se o software for necessário para processos de negócios críticos. As grandes empresas também têm mais recursos e podem proteger seus dados em sua própria infraestrutura de TI. Por outro lado, empresas menores com recursos financeiros limitados podem achar mais econômico usar o modelo de aluguel ou pagamento por uso. Se o segmento alvo do software for restrito, o modelo tradicional ou o aluguel de software podem funcionar bem. O modelo de pagamento por uso, por sua vez, é mais adequado para um grupo grande de clientes.

Por fim o autor expõe algumas conclusões de sua pesquisa. Para ele, em alguns casos, o modelo tradicional ainda tem vantagens sobre o modelo SaaS, pois pode atender clientes grandes e pequenos e segmentos-alvo amplos e estreitos. Além disso, ajuda a proteger contra a pirataria de software e fornece um fluxo de receita previsível e menos arriscado. Contudo o autor deixa claro que o modelo tradicional não é o mais vantajoso em todos os casos e que os fornecedores de software podem aumentar a lucratividade e expandir sua base de clientes fornecendo uma licença tradicional para grandes clientes que fazem uso extensivo do software em seu negócio principal, um modelo de aluguel para usuários de classe média e um modelo de pagamento por uso para usuários ocasionais.

O autor ainda afirma que o modelo de aluguel de software, do ponto de vista do cliente, pode oferecer uma compensação entre o licenciamento tradicional e o modelo de pagamento por uso, pois tem a vantagem particular de tornar possível estimar os custos do software e comprá-lo sem orçamentos separados ou processos de decisão complicados.

## 5.2 Implementação de multilocação na camada de dados

Os autores (AN, 2008) se propuseram a estudar as tecnologias para construir uma infraestrutura multilocatária econômica, segura e escalonável, especialmente na camada de dados. Eles exploraram e avaliaram (através de uma série de experimentos) os principais padrões de implementação de multilocação da camada de dados com relação ao isolamento, segurança, personalização e escalabilidade. Além disso, identificaram os gargalos de desempenho em potencial e resumiram as abordagens de otimização correspondentes e as melhores práticas de implementação para diferentes modelos de uso de negócios multilocatários.

A infraestrutura multilocatária, segundo os autores, deve se preocupar principalmente com os seguintes aspectos: 1) isolamento de recursos: separar a alocação e o uso de recursos entre os inquilinos; 2) segurança: impedir o acesso a recursos inválidos e, consequentemente, a um potencial ataque malicioso; 3) personalização: oferecer suporte a recursos específicos do inquilino por meio de configurações; 4) escalabilidade: dimensionar a infraestrutura de entrega do aplicativo SaaS para suportar o número crescente de inquilinos, garantindo desempenho e disponibilidade.

Com relação ao padrão de isolamento de recursos os autores afirmam que na camada de dados de um aplicativo multilocatário existem vários graus de isolamento de dados e descrevem três deles: 1) totalmente isolado (padrão de banco de dados dedicado): cada inquilino possui um banco de dados separado; 2) parcialmente compartilhado (tabela dedicada/padrão de esquema): vários inquilinos compartilham um banco de dados, mas cada inquilino possui tabelas/esquema separados; 3) totalmente compartilhado (padrão de tabela/esquema de compartilhamento): vários inquilinos compartilham o mesmo banco de dados e compartilham as mesmas tabelas/esquema.

Nos experimentos conduzidos pelos autores, visando avaliar os padrões de isolamento descritos acima, cada inquilino possui em média 4.000 registros de pedidos na tabela “SalesOrder” com 20 colunas de dados em cada registro. Os resultados desses experimentos mostram que para os inquilinos maiores, com TPS (Transações por Segundo - o mínimo aceitável por inquilino varia de 0,1 a 5) relativamente alto, como 5, o padrão de banco de dados dedicado é o mais adequado, pois pode fornecer melhores recursos de isolamento sem introduzir sobrecarga de desempenho significativa. Já para os inquilinos menores, com TPS relativamente baixo, é preferível utilizar a tabela dedicada/padrão de esquema ou o padrão de tabela/esquema de compartilhamento, pois eles podem obter vantagens de desempenho óbvias.

Outra avaliação realizada pelos autores é com relação ao isolamento de segurança de dados entre os inquilinos, ou seja, impedir que um usuário obtenha privilégios para acessar dados pertencentes a outros inquilinos. Os autores destacam a existência de dois

padrões para realizar os mecanismos de segurança de dados: 1) padrão baseado em filtro no nível do aplicativo: ao adicionar o filtro no nível do aplicativo a cada solicitação do inquilino do usuário, os dados do inquilino podem ser acessados apenas pelo próprio inquilino; 2) padrão baseado em permissão em nível de SGBD: cada inquilino é atribuído a uma conta de acesso ao banco de dados dedicada que só tem permissão para acessar seus próprios recursos.

A comparação da diferença de desempenho dos dois padrões citados acima mostra que no banco de dados dedicado o mecanismo de autorização baseado no esquema do banco de dados não introduzirá sobrecarga de desempenho. Já no padrão "esquema de compartilhamento"a tecnologia CABR (Controle de Acesso Baseado em Rótulo) resultará em uma redução de desempenho de cerca de 15%, especialmente para as operações de inserção.

Partes dos resultados dos estudos conduzidos pelos autores já foram aplicadas no projeto e implementação de um aplicativo multilocatário real. Os autores esperam que as experiências práticas os ajudem a abordar mais tópicos de pesquisa sobre otimização de desempenho e aspectos de escalabilidade na camada de dados, como, por exemplo, equilíbrio de carga de reconhecimento de comportamento de inquilino em ambiente de cluster de banco de dados distribuído.

Outro objetivo dos autores é explorar tecnologias para adequar o SGBD tradicional para ambientes multilocatários. Eles pretendem iniciar a pesquisa com servidores de banco de dados de código aberto como, por exemplo, mySQL e Derby.

Os autores acreditam no surgimento futuro de um novo projeto de SGBD com multilocação nativa que oferecerá suporte a desenvolvedores de aplicativos SaaS e provedores de serviços.

## 5.3 Esquemas flexíveis para SaaS

Os autores (AULBACH, 2009) afirmam que um sistema de banco de dados multilocatário para SaaS deve oferecer esquemas flexíveis, pois podem ser estendidos para diferentes versões do aplicativo e modificados dinamicamente enquanto o sistema está on-line. Uma comparação experimental de cinco técnicas de implementação de esquemas flexíveis para SaaS é apresentada pelos autores com o objetivo de definir a melhor opção entre elas.

Em três das cinco técnicas analisadas pelos autores a estrutura do Banco de Dados (BD) é definida explicitamente em DDL. Nesse tipo de BD geralmente são utilizados dois tipos de mapeamentos: um mais comum em que cada inquilino recebe suas próprias tabelas privadas e outro que emprega colunas esparsas no Microsoft SQL Server. Os

autores reconhecem que essas técnicas têm um bom desempenho, mas observam que elas oferecem suporte limitado para a evolução do esquema.

Nas outras duas técnicas o esquema é mapeado em estruturas genéricas no BD. Por exemplo, XML no DB2 e tabelas dinâmicas no HBase. Os autores consideram que essas técnicas fornecem ao aplicativo controle completo sobre a evolução do esquema, porém podem produzir uma diminuição significativa no desempenho a menos que partes significativas do BD sejam reimplementadas externamente.

Após concluírem que o BD ideal para SaaS ainda não foi desenvolvido os autores oferecem algumas sugestões de como ele deveria ser projetado. Os mesmos acreditam que o sistema deveria ser baseado no mapeamento de tabelas privadas, uma vez que a intercalação de inquilinos que ocorre em todos os outros mapeamentos divide o particionamento natural dos dados, forçando assim a importação e exportação de dados do inquilino que devem ocorrer para backup, restauração e migração. Com as tabelas privadas, no entanto, os dados de cada inquilino são agrupados no disco para que possam ser manipulados de forma independente.

Ainda como parte da sugestão dos autores, no sistema de BD o DDL deveria suportar explicitamente a extensão do esquema. O esquema base e suas extensões deveriam ser registrados como modelos. Cada inquilino deveria ser capaz de selecionar várias extensões para o esquema base. O sistema não deveria apenas carimbar uma nova cópia do esquema para cada inquilino, mas deveria manter a estrutura de compartilhamento. Essa estrutura deveria ser utilizada para aplicar alterações de esquema a todos os inquilinos relevantes. Além disso, essa estrutura reduziria a quantidade de metadados gerenciados pelo sistema.

Por fim os autores esclarecem que as sugestões por eles oferecidas podem ser aplicadas sem maiores problemas aos bancos de dados de memória principal, já que os avanços nas tecnologias de armazenamento de dados estão gradativamente tornando esse tipo de banco de dados mais atraente.

## 5.4 Opções de implementação de multilocação

De acordo com os autores (MOMM; KREBSC, 2011) várias opções para a implementação de multilocação estão disponíveis, desde partes únicas compartilhadas da pilha de aplicativos, como infraestrutura ou Middleware, até instâncias de aplicativos compartilhados. Os autores afirmam que a próxima grande etapa de evolução para muitos aplicativos existentes é justamente a ativação de multilocação. Por essa razão os autores apresentam um modelo de custo simples para avaliar diferentes opções de implementação de multilocação dependendo do aplicativo em questão, o custo de operação, gerenciamento do ciclo de vida, o número esperado de inquilinos e o período de uso esperado.

O objetivo dos autores é auxiliar os tomadores de decisão na comparação de diferentes opções de implementação de multilocação para um determinado cenário de SaaS em relação ao custo total de propriedade resultante. Para atingir a meta proposta os autores primeiramente apontam os principais requisitos do cliente, bem como as opções mais importantes para a habilitação de multilocação em um aplicativo existente. Em um segundo momento eles apresentam, como já dito anteriormente, um modelo de custo simples que explica os principais direcionadores de custos, tanto fixos quanto variáveis. Os autores ainda discutem a respeito das diferentes alternativas de implementação em relação aos custos fixos necessários para atender às expectativas do cliente e os custos variáveis de operação.

Alguns desafios da multilocação assim como seus principais requisitos são identificados pelos autores: 1) provisionamento: define o tempo para provisionar uma nova instância de serviço para um cliente. Assim, os clientes esperam obter uma instância totalmente operacional oferecida dentro de minutos a horas, não horas a dias; 2) capacidade de resposta: para um determinado limite de carga de trabalho do cliente o aplicativo não deve cair abaixo de um certo tempo de resposta; 3) escalabilidade: os clientes esperam das ofertas de SaaS que elas se adaptem aos requisitos de carga de trabalho em constante mudança de uma forma muito flexível; 4) tempo de inatividade planejado: como o software ainda evolui com o tempo, o provedor requer tempo para manutenção regular e atividades de atualização; 5) disponibilidade (tempo de inatividade não planejado): refere-se ao tempo de atividade do serviço sem tempos de inatividade planejados; 6) segurança de dados: mecanismos adequados devem estar disponíveis para evitar a perda de dados. Devem existir mecanismos especiais de backup para oferecer suporte a uma reversão de dados específica do inquilino sem sobrescrever os dados não afetados, garantindo o isolamento dos dados; 7) segurança/privacidade: Em uma configuração multilocatária o provedor deve garantir um isolamento lógico dos dados entre os inquilinos. Uma vez que os inquilinos podem compartilhar o mesmo banco de dados ou a mesma tabela, a complexidade da lógica do aplicativo pode aumentar.

Três opções para implementar uma solução multilocatária são apresentadas pelos autores: 1) infraestrutura compartilhada: apenas a infraestrutura é compartilhada entre os inquilinos que usam tecnologias de virtualização; 2) Middleware compartilhado: envolve uma instância de Middleware compartilhada em um sistema operacional compartilhado, que pode ser colocado em um hardware físico ou virtualizado; 3) aplicativo compartilhado: permite menos sobrecarga por inquilino, uma vez que a instância do aplicativo já é compartilhada entre os inquilinos. No entanto essa opção provavelmente envolve a maioria das atividades de reengenharia.

A complexidade da ativação de SaaS multilocatário, concluem os autores, pode ser reduzida pelo desenvolvimento de técnicas de migração e reengenharia adequadas. Eles

acreditam que o aprimoramento dos modelos de Middleware e programação existentes melhoraria o desenvolvimento de novos aplicativos para o modelo SaaS e criaria novas opções para a migração de aplicativos existentes.

A multilocação na visão dos autores (BHUVANESWARI; SARASWATHI, 2014) oferece várias vantagens, incluindo economia de custos, utilização de recursos, controle de versão e muito mais. Por essa razão os autores se dedicam a discutir duas opções para a implementação de multilocação: 1) uma abordagem compartilhada de esquema eficiente para banco de dados multilocatário baseado em nuvem com estrutura de autenticação e autorização; 2) um banco de dados multilocatário não intrusivo para aplicativos de grande escala.

A primeira opção inclui o conceito de arquitetura com suporte para multilocação, isolamento de dados entre locatários, autenticação de dados e estrutura de autorização para segurança. O isolamento de dados é feito através de um esquema de banco de dados otimizado, já os recursos de autenticação e autorização são implementados utilizando os conceitos do Kerberos que utiliza criptografia de chave simétrica para a autenticação das entidades seguindo as seguintes etapas: 1) o cliente envia a solicitação ao servidor de autenticação; 2) o servidor de concessão de tickets recebe o ticket do servidor de autenticação; 3) enviar a resposta do servidor de concessão de tickets para o servidor de concessão de serviço; 4) obtendo acesso ao servidor: utilizando o algoritmo Kerberos os usuários desfrutam de desempenho otimizado para operações de inserção, atualização e exclusão.

A segunda opção, por sua vez, utiliza a tecnologia de ativação de banco de dados multilocatário, ou seja, conceitos de isolamento de dados que oferecem suporte a todos os três modelos de banco de dados (descritos no próximo parágrafo). Ela também suporta clustering de banco de dados e mecanismos de roteamento e balanceamento de carga para escalabilidade.

Três abordagens para o gerenciamento de dados multilocatários no banco de dados em nuvem são apresentadas pelos autores da seguinte forma: 1) armazenar os dados de cada inquilino em bancos de dados separados. Esta é a abordagem mais simples para o isolamento de dados; 2) alojar vários inquilinos no mesmo banco de dados com cada inquilino tendo seu próprio conjunto de tabelas agrupadas em um esquema criado especificamente para o inquilino; 3) utilizar o mesmo banco de dados e o mesmo conjunto de tabelas para hospedar dados de vários inquilinos.

## 5.5 Big SaaS

Os autores (BAYLEY, 2015) afirmam que em um futuro previsível, os aplicativos SaaS se integrarão à Internet das coisas, computação móvel, big data, redes de sensores

sem fio e muitas outras tecnologias de computação e comunicação para fornecer serviços inteligentes personalizáveis a uma vasta população. Os autores preveem que toda essa integração iniciará uma era de sistemas denominados Big SaaS de complexidade e escala sem precedentes. Esses sistemas, ainda na previsão dos autores, terão um grande número de inquilinos inter-relacionados de maneiras complexas. O código, por sua vez, também será complexo, mas fornecerá grande valor para o cliente.

Juntamente com os benefícios que esses sistemas proporcionarão os autores chamam a atenção para os grandes riscos e desvantagens que também surgirão e discutem como lidar com esses problemas em todos os estágios do ciclo de vida do software.

Os principais desafios para o desenvolvimento de aplicativos Big SaaS na visão dos autores são os seguintes: 1) desafios sociais: para a sociedade como um todo - aceitar as mudanças em vários aspectos empresariais, financeiros, jurídicos, éticos e morais; 2) desafios técnicos: para a indústria e pesquisadores - desenvolver novas técnicas e novas aplicações de técnicas existentes; 3) desafios de engenharia: para engenheiros e metodologistas - desenvolver novos processos, métodos e ferramentas para produzir aplicativos de forma sistemática, eficiente e até automática.

Com relação à engenharia propriamente dita os autores citam os seguintes fatores como entraves para o avanço do Big SaaS: 1) riscos sociais: para um aplicativo SaaS, o Risco SaaS de falha é: Risco SaaS = R × T × C, onde T é o número de inquilinos que residem no sistema; R é a taxa de falha do sistema; C é a consequência média de uma falha por inquilino. Para um sistema de aplicativo de software de propriedade do cliente, o risco total RiscoWS de falha global é: RiscoWS = R' × C' × S, onde S é o número de cópias do sistema em execução ao mesmo tempo globalmente; R' é a taxa de falha do sistema e C' é a consequência média de uma falha para o cliente que executa uma cópia do software. Suponha que cada locatário execute uma cópia do sistema (ou seja, T = S), e que o SaaS tenha o mesmo nível de confiabilidade do software de propriedade do cliente (ou seja, R = R'). Então, temos que Risco SaaS = RiscoWS, se C = C'; 2) crowdsourcing confiável: quando há um grande número de inquilinos é altamente desejável que um aplicativo SaaS suporte a personalização para que as necessidades específicas dos clientes e seus usuários possam ser atendidas. No entanto, para o Big SaaS essa customização não pode ser feita pelo provedor de serviços manualmente. Uma solução adotada por quase todos os aplicativos SaaS existentes de sucesso é o crowdsourcing. Isso significa que os próprios clientes realizam a customização; 3) evolução contínua: a evolução contínua tem sido aplicada à prática de desenvolvimento de software para sistemas baseados na web como parte de metodologias ágeis. Nessa abordagem, um sistema de software é revisado, testado e atualizado com tanta frequência que a noção de versões e lançamentos não faz mais sentido. Além disso, a evolução contínua também exige que tais atualizações e lançamentos sejam aplicados sem qualquer interrupção do serviço. Isso acaba se tornando

um desafio para o Big SaaS devido sua complexidade sem precedentes; 4) integridade conceitual: a integridade conceitual é um dos principais recursos de um bom design de software. Significa que existe um modelo conceitual simples do sistema no qual sua estrutura, funcionalidade e comportamento dinâmico podem ser compreendidos. O design de um bom modelo conceitual para um aplicativo Big SaaS e a manutenção de sua integridade desempenham um papel crucial no desenvolvimento e na manutenção. Eles também desempenham um papel na personalização e na evolução contínua do sistema.

Como já dito os autores discutem soluções para os problemas por eles apresentados em todos os estágios do ciclo de vida do software. Na fase de especificação, por exemplo, eles enumeram os benefícios que o uso de uma linguagem de especificação algébrica pode trazer: 1) apoiar o desenvolvimento formal de sistemas orientados a serviços para melhorar a confiabilidade; 2) manter a integridade conceitual por fornecer uma definição formal do modelo conceitual; 3) oferecer suporte à personalização baseada em crowdsourcing, vinculando a especificação formal à descrição ontológica dos serviços; 4) automatizar o teste com base em especificações algébricas. Isso, segundo os autores, também ajuda na evolução contínua.

Na fase de projeto arquitetônico os autores sugerem a instalação de um ponto de verificação no nível do inquilino. Isso, de acordo com os mesmos, pode desempenhar um papel significativo na redução dos riscos sociais.

Já para a fase de implementação os autores afirmam que é desejável um novo paradigma de programação. Por essa razão os mesmos se dedicam a exploração do potencial de uma linguagem de programação orientada a agentes.

Por fim os autores reconhecem que a automação é essencial na fase de testes e afirmam que a especificação formal tornará isso possível.

# 6 Discussão

Com o ganho crescente de popularidade da computação em nuvem nos últimos anos, o desenvolvimento de aplicativos SaaS tem sido amplamente investigado na academia. Portanto, para se ter uma visão abrangente do que tem sido feito nessa área, apresentamos nesta dissertação os resultados de uma revisão sistemática da literatura sobre SaaS multilocatário que inclui 53 estudos primários. Nós identificamos três categorias principais de estudos: 1) propostas de desenvolvimento; 2) propostas de migração e 3) vantagens e desvantagens.

A primeira categoria é composta por 36 artigos e foi dividida em 8 subcategorias: customização (9 artigos); arquitetura (5 artigos); segurança (2 artigos); recuperação de falhas (2 artigos); execução de aplicativos atuais em ambiente multilocatário (1 artigo); gerenciamento de recursos (1 artigo); Middleware (4 artigos) e banco de dados (12 artigos). A segunda categoria composta por onze artigos, por sua vez, foi dividida em 6 subcategorias: propostas que dispensam reengenharia (2 artigos); propostas que utilizam reengenharia (2 artigos); framework (2 artigos); métodos próprios (3 artigos); arquitetura (1 artigo) e banco de dados (1 artigo). Já a terceira categoria abriga 6 artigos e foi dividida em 5 subcategorias: modelos de receita SaaS (1 artigo); implementação de multilocação na camada de dados (1 artigo); esquemas flexíveis para SaaS (1 artigo); opções de implementação de multilocação (2 artigos) e Big SaaS (1 artigo).

O primeiro trabalho relacionado ao desenvolvimento do SaaS foi publicado no ano de 2007 e abriu as portas para os demais que foram publicados nos anos de 2009 (3 trabalhos), 2010 (1 trabalho), 2011 e 2012 (5 trabalhos), 2013 (7 trabalhos), 2014 e 2015 (4 trabalhos), 2016 (1 trabalho), 2017 (2 trabalhos), 2018, 2019 e 2020 (1 trabalho). Ocorreu no ano de 2009 a publicação do primeiro trabalho relativo a segunda categoria. As publicações continuaram nos anos de 2010 (4 trabalhos), 2012 (2 trabalhos), 2013 (1 trabalho), 2014 (2 trabalhos) e uma última publicação em 2016.

A respeito das vantagens e desvantagens do SaaS, os anos de 2008, 2009, 2011, 2013, 2014 e 2015 testemunharam a publicação de um trabalho por ano. A seguir destacamos alguns trabalhos relevantes publicados ao longo desses 13 anos e que abordam as categorias supracitadas (veja 2).

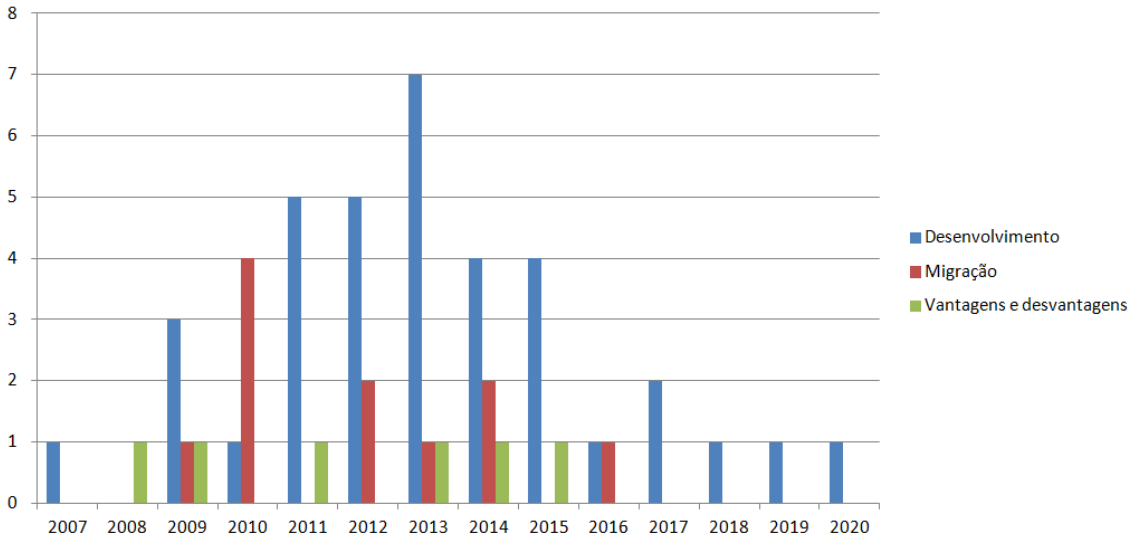


Figura 2 – Número de trabalhos de cada categoria em relação ao ano

Os trabalhos produzidos pelos autores (HUR; KANG, 2011) e (AGHERA, 2012) apresentam uma técnica muita semelhante para realizar a customização dos aplicativos SaaS. Ambos os trabalhos apostam na arquitetura orientada por metadados na qual o banco de dados possui dupla funcionalidade. Enquanto o banco de dados da aplicação mantém os dados gerados pelos usuários, o banco de dados de metadados armazena as informações de configuração da aplicação. A diferença principal entre os dois trabalhos é que os autores (AGHERA, 2012) ainda utilizam um terceiro banco de dados para monitorar os logs de acesso ao sistema e os registros de SLA.

O Acordo de Nível de Serviço (Service Level Agreement - SLA) é levado em consideração não somente pelos autores (AGHERA, 2012), mas, em 2009, já era citado pelos autores (HAN, 2009) que desenvolveram uma abordagem visando fornecer diferentes níveis de desempenho para os inquilinos com base no SLA. Em 2019 os autores (PINTO, 2019) afirmaram, baseados em uma pesquisa, que problemas relacionados ao SLA eram uma das causas de falhas nos aplicativos SaaS multilocatários.

Em 2014 os autores (ABU-MATAR, 2014) apresentaram uma plataforma orientada a recursos cujo objetivo era promover a customização dos aplicativos SaaS por meio de uma arquitetura na qual os provedores de serviços são capazes de gerenciar a variabilidade em um nível mais alto de abstração. Três anos mais tarde, em 2017, os mesmos autores apresentaram outra tecnologia visando facilitar a adaptação dinâmica de aplicativos SaaS multilicatários e oferecer suporte à personalização de configurações dos inquilinos em tempo de execução. Dessa vez a plataforma proposta se baseou nos conceitos de orientação a serviços, linhas de produtos de software e arquitetura orientada a modelos.

Observamos que um dos conceitos mencionados acima, Linhas de Produtos de Software (LPS), também aparece em outros três trabalhos. Em 2009 os autores (MIETZ-

NER, 2009) já propunham a modelagem da variabilidade de aplicações SaaS através da exploração das técnicas de LPS. Em 2012 foi a vez dos autores (DAMASCENO, 2012) apresentarem um aplicativo que fornecesse suporte ao desenvolvimento de LPS. Novamente em 2016, os autores (ELKHATIB, 2016) sugeriram as técnicas de LPS para o gerenciamento da variabilidade e apoio a evolução dos aplicativos multilocatários e seus requisitos.

Entre as tecnologias propostas para desenvolver bancos de dados multilocatários, nos chamou a atenção as Tabelas de Extensão Elástica (TEE). Essa abordagem que consiste na combinação de tabelas relacionais e tabelas relacionais virtuais foi proposta em 2011 pelos autores (FEUERLICHT, 2011). Em 2013 os mesmos autores voltaram a mencionar a abordagem em outro trabalho. Ainda em 2013 os autores (GOYAL; YAISH, 2013), dessa vez sem a participação de FEUERLICHT, publicaram outro trabalho no qual, mais uma vez, utilizaram a tecnologia.

Tanto o trabalho produzido por (CAI, 2010) quanto o produzido por (HOHENSTEIN; KOKA, 2016) se preocupam com as mesmas questões quanto se trata de migrar aplicativos de inquilino único para aplicativos multilocatários sem que seja necessário a reescrita do código-fonte original: isolamento e personalização. A diferença das propostas dos autores é que (HOHENSTEIN; KOKA, 2016) ainda levam em consideração a questão do monitoramento das atividades dos usuários dos inquilinos para fins de cobrança.

O autor (SAKAMOTO, 2009) integrou a equipe responsável por migrar o aplicativo "Internet Navigware", pertencente à empresa japonesa Fujitsu, da versão de inquilino único para a versão multilocatária. Um diferencial nos chamou a atenção na nova versão do aplicativo. O cliente pode definir o valor máximo que pode ser gasto mensalmente para pagar pela utilização do mesmo. Este recurso é interessante para clientes que não podem exceder determinado valor mensal e também é diferente do que tradicionalmente ocorre em aplicativos SaaS.

O trabalho de (OJALA, 2013) é bastante esclarecedor para os interessados em compreender como funciona o sistema de pagamento pela utilização dos aplicativos SaaS, além de suas vantagens e desvantagens. Pagamento por uso e aluguel de software são os dois principais modelos identificados pelo autor. Analisando as características apresentadas pelo autor a respeito de cada um dos modelos, podemos concluir que o recurso de pagamento incluído na nova versão do aplicativo "Internet Navigware", descrito por (SAKAMOTO, 2009), poderia ser incluído no modelo de pagamento por uso, pois o aplicativo só poderia ser utilizado enquanto não excedesse o valor mensal máximo estabelecido pelo cliente.

Por fim destacamos o trabalho dos autores (AN, 2008) que analisou as tecnologias utilizadas na construção de uma infraestrutura multilocatária econômica, segura e escalonável. Através de uma série de experimentos eles avaliaram quatro padrões de imple-

mentação de multilocação: isolamento, segurança, personalização e escalabilidade. Parece haver um consenso entre a maioria dos autores cujos trabalhos foram analisados nessa dissertação de que, de fato, esses são os principais padrões, pois em quase todos os trabalhados os mesmos foram citados, principalmente os padrões de isolamento e personalização que perceptivelmente foram analisados com maior frequência.

# 7 Conclusão

O SaaS é um modelo de serviço de computação em nuvem cuja principal característica é que os clientes podem pagar pelo uso de um aplicativo de software com uma assinatura pré-paga, ou seja, os clientes pagam somente pelos recursos que, de fato, utilizarem. Para ser economicamente sustentável, um aplicativo SaaS deve ser multilocatário. Entre outras vantagens a multilocação permite que um cliente (inquilino) compartilhe recursos de hardware entre diversos outros usuários.

Esta dissertação, consiste em um panorama do atual estado da arte da pesquisa em SaaS multilocatário com uma análise aprofundada dos aspectos pesquisados neste contexto. Dado o crescente interesse e a quantidade de pesquisas sobre o tema, geramos uma visão coletiva dos resultados de diferentes estudos realizados e que tipo de suporte foi desenvolvido para o SaaS Multilocatário, identificando seus pontos fortes e limitações. Mais especificamente, neste trabalho respondemos as seguintes questões de pesquisa: (QP-1) *Quais são os tipos de abordagens/métodos propostos para apoiar a criação de aplicações multi-tenant?*; (QP-2) *Quais são os tipos de abordagens/métodos propostos para apoiar a migração de aplicações multi-tenant?*; e (QP-3) *Quais são as evidências das vantagens e desvantagens do uso de multi-tenant?*. Para isso, analisamos 53 artigos selecionados de diferentes bases de dados (ACM Digital Library, IEEE Xplorer Digital Library e Springer Link). Como resultados, identificamos diversas propostas de apoio ao desenvolvimento de SaaS Multilocatário (36 artigos) envolvendo tópicos como: customização, arquitetura, segurança, recuperação de falhas, gerenciamento de recursos, middleware e banco de dados. Também, identificamos diverssos trabalhos (11 artigos) que propõem soluções para migração de sistemas de software para o modelo SaaS Multilocatário. Finalmente, apresentamos algumas evidências (observadas em 6 artigos) das vantagens e/ou desvantagens do SaaS Multilocatario.

Os resultados deste trabalho podem ser usados por desenvolvedores visando aprender aspectos do SaaS para que eles possam melhorar suas práticas, e por pesquisadores para que possam adquirir uma base sólida neste tópico e sejam capazes de desenvolver abordagens, avaliações e estudos básicos futuros.

# Referências

HUR, S.; KANG, S. A design of the conceptual architecture for a multitenant saas application platform. In: *Proceedings of the 2011 First ACIS/JNU International Conference on Computers, Networks, Systems and Industrial Engineering (CNSI)*. [S.l.: s.n.], 2011. Citado 3 vezes nas páginas 12, 25 e 78.

YOUNAS, M.; YOUSEF, B.; ZHU, H. Tenant level checkpointing of meta-data for multi-tenancy saas. In: *Proceedings of the 2014 IEEE International Workshop on Service-Oriented System Engineering (SOSE)*. [S.l.: s.n.], 2014. Citado 2 vezes nas páginas 14 e 40.

BANKAR, H.; BHUSE, S.; KHATAWKAR, P.; KULKARNI, G.; PALWE, R.; SHELKE, R. Multi-tenant saas cloud. In: *Proceedings of the 2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*. [S.l.: s.n.], 2013. Citado na página 15.

AULBACH, S.; JACOBS, D. Ruminations on multi-tenant databases. In: *Proceedings of the 2007 Business Technology and Web (BTW)*. [S.l.: s.n.], 2007. Citado na página 16.

DAVILA, N. d. C. *Revisão de código moderna: dos estudos básicos às abordagens propostas e sua avaliação*. Dissertação — Universidade Federal do Rio Grande do Sul, Rio Grande do Sul: RS, 2020. Citado na página 18.

AGHERA, P.; CHAUDHARY, S.; KUMAR, V. An approach to build multi-tenant saas application with monitoring and sla. In: *Proceedings of the 2012 International Conference On Communication Systems And Network Technologies (CSNT))*. [S.l.: s.n.], 2012. Citado 2 vezes nas páginas 26 e 78.

KHAMIS, A.; SAMIR, A.; SHAHIN, A. An aspect-oriented approach for saas application customization. In: *Proceedings of the 2013 48th Conference on Statistics, Computer Science and Operations Research*. [S.l.: s.n.], 2013. Citado na página 27.

CHAUVEL, F.; SOLBERG, A. Using intrusive microservices to enable deep customization of multi-tenant saas. In: *Proceedings of the 2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*. [S.l.: s.n.], 2013. Citado na página 28.

SALEH, A.; FOUAD, M.; ABU-ELKHEIR, M. Classifying requirements for variability optimization in multi-tenant applications. In: *Proceedings of the 2014 IEEE 6th International Conference on Cloud Computing Technology and Science*. [S.l.: s.n.], 2014. Citado na página 29.

MIETZNER, R.; METZGER, A.; LEYMANN, F.; POHL, K. Variability modeling to support customization and deployment of multitenant-aware software as a service applications. In: *Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems (PESOS)*. [S.l.: s.n.], 2009. Citado 2 vezes nas páginas 30 e 79.

ABU-MATAR, M.; AL-QUTAYRI, M.; MAHMOUD, Z. A.; MIZOUNI, R.; MOHAMED, F. Saas dynamic evolution based on model-driven software product lines. In: *Proceedings*

*of the 2014 IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom)*. [S.l.: s.n.], 2014. Citado 2 vezes nas páginas 30 e 78.

ABU-MATAR, M.; AL-QUTAYRI, M.; WHITTLE, J.; MIZOUNI, R.; MOHAMED, F. An integrated platform for dynamic adaptation of multi-tenant. In: *Proceedings of the 2017 IEEE 5th International Conference on Future Internet of Things and Cloud, (FiCloud)*. [S.l.: s.n.], 2017. Citado na página 31.

HAN, Y.; LIN, H.; SUN, K.; ZHAO, S. Feedback-control-based performance regulation for multi-tenant applications. In: *Proceedings of the 2009 15th International Conference on Parallel and Distributed Systems (ICPADS)*. [S.l.: s.n.], 2009. Citado 2 vezes nas páginas 31 e 78.

DAMASCENO, J. C.; GARCIA, V. C.; LEITE, A.; MEIRA, S. R. M.; RODRIGUES, J.; SILVEIRA, P. An approach to developing multi-tenancy saas using metaprogramming. In: *Proceedings of the 18th Brazilian symposium on Multimedia and the web*. [S.l.: s.n.], 2012. Citado 2 vezes nas páginas 32 e 79.

GAO, B.; GUO, C. J.; HUANG, Y.; SUN, W.; WANG, Z. H. A framework for native multi-tenancy application development and management. In: *Proceedings of the 9th IEEE International Conference on E-Commerce Technology (CEC)*. [S.l.: s.n.], 2007. Citado na página 34.

HUANG, W.; WANG, Z.; WEI, X.; XIAO, Y.; ZHAO, Y. A multi-tenant software as a service model for large organization. In: *Proceedings of the 2013 International Conference on Cloud and Service Computing (CSC)*. [S.l.: s.n.], 2013. Citado na página 35.

CAI, H.; CAI, J.; GONG, W.; MAO, X.; ZHANG, K.; ZHOU, M. An end-to-end methodology and toolkit for fine granularity saas-ization. In: *Proceedings of the 2009 IEEE International Conference on Cloud Computing*. [S.l.: s.n.], 2009. Citado na página 36.

AßMANN, U.; CECH, S.; GöTZ, S.; SCHROETER, J.; WILKE, C. Towards modeling a variable architecture for multi-tenant saas-applications. In: *Proceedings of the 2012 Sixth International Workshop on Variability Modeling of Software-Intensive Systems*. [S.l.: s.n.], 2012. Citado na página 37.

MATSUO, A.; NISHIKAWA, K.; OKI, K. Saas application framework using information gateway enabling cloud service with data confidentiality. In: *Proceedings of the 2012 19th Asia-Pacific Software Engineering Conference (APSEC)*. [S.l.: s.n.], 2012. Citado na página 38.

ALMORSY, M.; GRUNDY, J.; IBRAHIM, A. S. Tossma: A tenant-oriented saas security management architecture. In: *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*. [S.l.: s.n.], 2012. Citado na página 39.

PINTO, V. H.; SOUZA, P. S.; SOUZA, S. R. A preliminary fault taxonomy for multi-tenant saas systems. In: *Proceedings of the 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. [S.l.: s.n.], 2019. Citado 2 vezes nas páginas 42 e 78.

AZEEZ, A.; FREMANTLE, P.; GAMAGE, D.; LEELARATNE, D.; LINTON, R.; PERERA, S.; SIRIWARDANA, P.; WEERAWARANA, S. Multi-tenant soa middleware for cloud computing. In: *Proceedings of the 2010 3rd IEEE International Conference on Cloud Computing*. [S.l.: s.n.], 2010. Citado na página 43.

ELKHATIB, Y.; JUMAGALIYEV, A.; WHITTLE, J. Evolving multi-tenant saas cloud applications using model-driven engineering. In: *Proceedings of the 10th International Workshop on Models and Evolution*. [S.l.: s.n.], 2016. Citado 2 vezes nas páginas 44 e 79.

CHEN, J. J.; CHEN, K.; LIAO, C. F. Context and data management for multitenant enterprise applications in saas environments: A middleware approach. In: *Proceedings of the 2014 International Conference on Software Technologies (ICSOFT)*. [S.l.: s.n.], 2014. Citado na página 45.

JOOSEN, W.; LAGAISSE, B.; LANDUYT, D. V.; RAFIQUE, A. Policy-driven data management middleware for multi-cloud storage in multi-tenant saas. In: *Proceedings of the 2015 2nd IEEE/ACM International Symposium on Big Data Computing (BDC)*. [S.l.: s.n.], 2015. Citado na página 46.

BORGER, W.; JOOSEN, W.; LAGAISSE, B.; LANDUYT, D. V.; VANBRABANT, B.; WALRAVEN, S. Adaptive performance isolation middleware for multi-tenant saas. In: *Proceedings of the 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*. [S.l.: s.n.], 2015. Citado na página 47.

GEY, F.; JOOSEN, W.; LANDUYT, D. V. Middleware for customizable multi-staged dynamic upgrades of multi-tenant saas applications. In: *Proceedings of the 2015 8th International Conference On Utility And Cloud Computing (UCC)*. [S.l.: s.n.], 2015. Citado na página 47.

AN, W. H.; FAN, L.; GAO, B.; GUO, C. J.; SUN, W.; SUN, X.; WANG, Z. H. A non-intrusive multi-tenant database for large scale saas applications. In: *Proceedings of the 2011 IEEE 8th International Conference on e-Business Engineering (ICEBE)*. [S.l.: s.n.], 2011. Citado na página 48.

FEUERLICHT, G.; GOYAL, M.; YAISH, H. An elastic multi-tenant database schema for software as a service. In: *Proceedings of the 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC)*. [S.l.: s.n.], 2011. Citado 2 vezes nas páginas 49 e 79.

FEUERLICHT, G.; GOYAL, M.; YAISH, H. Proxy service for multi-tenant database access. In: *Proceedings of the 2013 International Conference on Availability, Reliability, and Security*. [S.l.: s.n.], 2013. Citado na página 49.

GOYAL, M.; YAISH, H. A multi-tenant database architecture design for software applications. In: *Proceedings of the 2013 IEEE 16th International Conference On Computational Science And Engineering (CSE))*. [S.l.: s.n.], 2013. Citado 2 vezes nas páginas 50 e 79.

KUSHWAHA, D. S.; PIPPAL, S. K. A simple, adaptable and efficient heterogeneous multi-tenant database architecture for ad hoc cloud. *International Journal of Computer Information Systems and Industrial Management Applications*, v. 2, 2013. Citado na página 51.

LI, X.; XU, J.; ZHAO, X. Design of database architecture in the saas-based multi-tenant educational information system. In: *Proceedings of the 2011 6th International Conference on Computer Science & Education (ICCSE)*. [S.l.: s.n.], 2011. Citado na página 51.

JOOSEN, W.; LANDUYT, D. V.; RAFIQUE, A.; RENIERS, V. Leveraging nosql for scalable and dynamic data encryption in multi-tenant saas. In: *Proceedings of the IEEE 2017 International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. [S.l.: s.n.], 2017. Citado na página 52.

INDRANI, G.; SHIRI, N.; RADHAKRISHNAN, T. A. Flexible data model for multi-tenant databases for software as a service. In: *Proceedings of the 2013 IEEE 16Th International Conference On Computational Science And Engineering*. [S.l.: s.n.], 2013. Citado na página 53.

BRODT, A.; MITSCHANG, B.; SCHILLER, B.; SCHILLER, O. Native support of multi-tenancy in rdbms for software as a service. In: *Proceedings of the 14th International Conference on Extending Database Technology*. [S.l.: s.n.], 2011. Citado na página 53.

WANG, X.; WANG, X.; ZHAO, G.; ZHOU, X. Translation of ddl and dml for constructing flexible schemas in saas applications. In: *Proceedings of the 2020 5th International Conference on Computer and Communication Systems (ICCCS)*. [S.l.: s.n.], 2020. Citado na página 54.

CHEN, J. J.; CHEN, K.; LIAO, C. F.; TAN, D. H. Automatic query rewriting schemes for multitenant saas applications. In: *Proceedings of the 2016 IEEE/ACM 31st International Conference on Automated Software Engineering (ASE)*. [S.l.: s.n.], 2016. Citado na página 55.

MEINEL, C.; SALEH, E.; TAKOUNA, I. Signedquery: Protecting users data in multi-tenant saas environments. In: *Proceedings of the 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. [S.l.: s.n.], 2013. Citado na página 55.

CAI, H.; WANG, N.; ZHOU, M. J. A transparent approach of enabling saas multi-tenancy in the cloud. In: *Proceedings of the 2010 6th World Congress On Services*. [S.l.: s.n.], 2010. Citado 2 vezes nas páginas 56 e 79.

HOHENSTEIN, U.; KOKA, P. Enabling legacy applications for multi-tenancy without reengineering. In: *Proceedings of the International Conference on Software Technologies (ICSOFT)*. [S.l.: s.n.], 2016. Citado 2 vezes nas páginas 57 e 79.

BEZEMER, C. P.; ZAIDMAN, A. C. *Challenges of Reengineering into Multi-Tenant SaaS Applications*. [S.l.]: Delft University of Technology - Faculty of Electrical Engineering, Mathematics and Computer Science. Delft, Holanda, 2010. Citado na página 58.

BEZEMER, C. P.; HART, A.; HURKMANS, T.; PLATZBEECKER, B.; ZAIDMAN, A. Enabling multi-tenancy: An industrial experience report. In: *Proceedings of the 2010 IEEE International Conference on Software Maintenance*. [S.l.: s.n.], 2010. Citado na página 59.

NUGRAHENI, E. Migration of web application sima into multi-tenant saas. In: *Proceedings of the International Conference on ICT For Smart Society (ICISS)*. [S.l.: s.n.], 2013. Citado na página 60.

ALMORSY, M.; GRUNDY, J.; IBRAHIM, A. S. Smurf: Supporting multi-tenancy using re-aspects framework. In: *Proceedings of the 2012 IEEE 17th International Conference on Engineering of Complex Computer Systems*. [S.l.: s.n.], 2012. Citado na página 60.

CHEN, W.; SHEN, B.; TANG, X.; ZHANG, X. From isolated tenancy hosted application to multi-tenancy: Toward a systematic migration method for web application. In: *Proceedings of the 2010 IEEE International Conference on Software Engineering and Service Sciences (ICSESS)*. [S.l.: s.n.], 2010. Citado na página 61.

SAKAMOTO, N. *Construction of SaaS-based e-learning System in Japan*. 2009. Disponível em: <https://www.fujitsu.com/downloads/MAG/vol45-3/paper05.pd>. Citado 2 vezes nas páginas 62 e 79.

AN, W.; FAN, L.; GAO, B.; WANG, Y.; WANG, Z. *A Semi-Automatic Approach of Transforming Applications to be Multi-Tenancy Enabled*. [S.l.]: IEEE, 2016. Citado na página 62.

ZHENG J.; DU, W. Cloud services for deploying client-server applications to saas. In: *2014 International Conference on Internet of Vehicles (ICIOV)*. [S.l.: s.n.], 2014. Citado na página 63.

CAO, J.; JIANG, L.; LI, P.; ZHU, Q. A mixed multi-tenancy data model and its migration approach for the saas application. In: *Proceedings of the 2012 IEEE Asia-Pacific Services Computing Conference (APSCC)*. [S.l.: s.n.], 2012. Citado na página 64.

OJALA, A. *Software-as-a-Service Revenue Models*. [S.l.]: IEEE, 2013. Citado 2 vezes nas páginas 66 e 79.

AN, W. H.; GAO, B.; GUO, C. J.; SUN, W.; WANG, Z. H.; ZHANG, Z. A study and performance evaluation of the multi-tenant data tier design patterns for service oriented computing. In: *Proceedings of the 2008 IEEE International Conference on e-Business Engineering*. [S.l.: s.n.], 2008. Citado 2 vezes nas páginas 70 e 79.

AULBACH, S.; JACOBS, D.; KEMPER, A.; M, S. A comparison of flexible schemas for saas. In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. [S.l.: s.n.], 2009. Citado na página 71.

MOMM, C.; KREBSC, R. A qualitative discussion of different approaches for implementing multi-tenant saas offerings. In: *Proceedings of the 2011 33rd International Conference on Software Engineering (ICSE)*. [S.l.: s.n.], 2011. Citado na página 72.

BHUVANESWARI, T.; SARASWATHI, M. Multitenant saas model of cloud computing: Issues and solutions. In: *2014 International Conference on Communication and Network Technologies (ICCNT)*. [S.l.: s.n.], 2014. Citado na página 74.

BAYLEY, A.; LIGHTFOOT, D.; LIU, D.; YOUNAS, M.; YOUSEF, B.; ZHU, H. Big saas: The next step beyond big data. In: *Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing*. [S.l.: s.n.], 2015. Citado na página 74.