UNIVERSIDADE DO VALE DO RIO DOS SINOS – UNISINOS

UNIDADE ACADÊMICA DE EDUCAÇÃO ONLINE

ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

Everton Jeferson da Silva

BANCO DE DADOS EM APLICAÇÕES *Software as a Service*: Estudo de caso com *Azure* SQL *Database Elastic Pool*

Porto Alegre

2019

UNIVERSIDADE DO VALE DO RIO DOS SINOS – UNISINOS

UNIDADE ACADÊMICA DE EDUCAÇÃO ONLINE

ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

Everton Jeferson da Silva

BANCO DE DADOS EM APLICAÇÕES Software as a Service:
Estudo de caso com *Azure* SQL *Database Elastic Pool*

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do título de Especialista em Engenharia de Software, pelo curso de Pós-Graduação Lato Sensu em Engenharia de Software da Universidade do Vale do Rio dos Sinos – UNISINOS.
Orientador: Prof. Dra. Josiane B. Porto

Porto Alegre

2019

Banco de dados em aplicações *Software as a Service*: estudo de caso com *Azure SQL Database Elastic Pool*

Everton J. da Silva[1]

[1]Universidade do Vale do Rio do Sinos – UNISINOS - Porto Alegre – RS - Brasil

evertonjdasilva@yahoo.com.br

Abstract. *With the advent of the internet, software development has undergone major changes in its architectures. Most modern applications are already projects to work exclusively via the internet. This type of architecture brings a number of advantages such as high-scale selling. However, this type of architecture can cause problems, especially in relation to databases. The paper presents a case study using Azure Sql Database Elastic Pool as a database for SaaS applications. By presenting comparative tests among other approaches, the results show how the tool behaved and in which scenarios it can be treated with a solution.*

Resumo. *Com o advento da internet, o desenvolvimento de software vem sofrendo grandes mudanças em suas arquiteturas. A maioria das aplicações modernas já são projetos para funcionar exclusivamente via internet. Este tipo de arquitetura traz uma série de vantagens, como a venda em alta escala. Contudo, esse tipo de arquitetura pode gerar problemas, principalmente em relação aos bancos de dados. O trabalho apresenta um estudo de caso utilizando o Azure Sql Database Elastic Pool como banco de dados para aplicações SaaS. Apresentando testes comparativos entre outras abordagens, os resultados mostram como a ferramenta se comportou e em quais cenários ela pode ser tratada com uma solução.*

1. Introdução

Nos últimos anos a área da computação vem sofrendo uma grande mudança impulsionada pela a chamada *cloud computing* ou 'computação em nuvem', termo surgido em 2006 [AYMERICH, 2008] e que se refere ao modelo para permitir acesso de rede onipresente, conveniente e sob demanda a uma rede compartilhada de recursos de computação configuráveis que pode ser provisionado e liberado rapidamente com esforço mínimo de gerenciamento ou interação com o provedor de serviços. [NIST, 2011]. Este tipo de modelo traz uma série de benefícios como alta flexibilidade no uso de recursos, custo de inicial de utilização mais baixo e pagamento conforme a utilização [KIM, 2009]

Com a computação em nuvem também surgiram alguns outros termos, sendo um deles o *software as a service* ou SaaS, este modelo se refere a softwares pessoais ou empresariais que são licenciados como um serviço, deste modo eliminando a necessidade de instalar e executar aplicação nos computadores locais do cliente [WANG, 2008].

No modelo tradicional, um software é vendido como um produto, onde o cliente paga por ele e deve fornecer os recursos de hardware para sua execução, no modelo como serviço, toda responsabilidade de fornecer os recursos necessários para a execução são transferidos para o fornecedor. Este modelo se torna muito vantajoso para o cliente, pois traz vantagens como o baixo custo inicial de operação. Já no lado da empresa fornecedora de software também é possível ver vantagens, como uma venda em escala mais rápida devido a implantação ser simplificada [BEZEMER e ZAIDMAN, 2010]. Contudo, surgem problemas, sendo um deles o custo de hospedagem. Como a maioria das empresas desenvolvedoras de software não tem sua própria infraestrutura de nuvem, é necessário que isso seja terceirizada para empresas de grande porte, como *Azure* (*Azure*, 2019) e *Amazon* (Amazon 2019). Devido a este contexto surgiu um novo tipo de arquitetura, a chamada multi-inquilino (BEZEMER e ZAIDMAN, 2010]) (do inglês, *multi-tenancy*) com o intuito de minimizar os custos de hospedagem e de manutenção.

A maioria das aplicações comerciais hoje são formadas por pelo menos 2 componentes, uma é a interface e a outra é o banco de dados. A interface é a parte onde o usuário interage e o banco de dados é componente necessário para a persistência dos dados. Nas aplicações *multi-tenancy*, a interface é tratada de forma mais natural, porque mesmo que o layout seja diferente para cada cliente, sempre é o mesmo código que é executado, sendo assim é possível compartilhar este recurso de forma mais simples. Já quando se fala em banco de dados, essa afirmação não é verdadeira. Cada cliente possuí seus dados e esses dados devem ser mantidos isolados entre os clientes. Isso faz com que cada cliente tenha que ter um banco de dados próprio ou esquemas compartilhados. [TSAI, BAI, HUANG, 2014]

Os bancos de dados em aplicações *multi-tenancy* se tornaram um problema, pois conforme dito anteriormente, as empresas desenvolvedoras terceirizam os recursos de nuvem para empresas maiores e este custo deve ser passado para o cliente. Hoje o custo do banco de dados pode chegar a 50% do valor pago pelo cliente em algumas aplicações, isso se deve ao fato que a arquitetura de banco de dados multi-inquilinos ideal é a com bancos de dados separados, sendo ela a ideal pois garante mais segurança, isolamento e personalização, mas isso implica em custos mais altos [SALEH, FOUAD, ABU-ELKHEIR, 2014].

Outras arquiteturas de bancos multi-inquilinos possuem um custo geralmente menor, como é o caso dos modelos de banco de dados com tabelas separadas e com tabelas compartilhadas, mas em contraponto, um modelo prejudica a escalabilidade do sistema e o desempenho, respectivamente [SALEH,FOUAD, ABU-ELKHEIR, 2014].

Devido a essa problemática os chamados *pool* de recursos tornam-se atrativos. *Pool* de recursos são recursos compartilhados que ao mesmo que compartilham o hardware entre si, também garantem um nível de isolamento, o que ajuda a melhorar a utilização dos recursos físicos, simplifica a administração e reduz custos para as empresas [SOUZA, 2013].

Algumas empresas fornecedoras de serviços *cloud* já fornecem esse tipo de recurso para banco de dados, como é o caso do *Azure Database Elastic Pool*, ferramenta da Microsoft projetada para atender a necessidade de aplicações multi-inquilinos e que se propõe a atender as exigências desse tipo de aplicação, garantido

economia no custo de operação e manutenção das bases de dados [MICROSOFT, 2018].

Mas será que esta ferramenta realmente consegue entregar o melhor custo benefício em bancos de dados em uma aplicação *multi-tenancy*? O objetivo do presente trabalho é responder essa questão. Desta forma, a utilização da ferramenta do Azure foi avaliada, com base em uma aplicação *multi-tenancy* específica e por meio de métricas de desempenho, uso de recursos, isolamento e custo, a fim de verificar se realmente são melhores ou equivalentes, quando comparado a bancos de *dados on-premise,* se tornando assim uma alternativa viável à problemática apresentada.

Por tanto, tem-se como objetivo central do trabalho analisar a problemática das arquiteturas dos bancos de dados em aplicações SaaS verificando a utilização da ferramenta Azure SQL Elastic Pool como repositório das bases de dados, contribuindo assim para o tema, gerando artefatos de comparação e uma análise para a produção de possíveis soluções eficazes, em nível de desempenho e custo, entre outras métricas relevantes.

Para tanto utilizou-se os seguintes objetivos específicos para alcançar o objetivo central: i) Analisar os aspectos de uma aplicação SaaS; ii) Analisar os aspectos do banco de dados em aplicações SaaS; iii) Criar cenários de testes para fins de avaliação; iv) Realizar os testes com a ferramenta proposta nos cenário projetados; v) Analisar os resultados dos testes, por meio de métricas referentes ao assunto.

A realização desses objetivos, bem como a análise das métricas, irá contribuir para o entendimento do problema e por sua vez gerar materiais essenciais para comparações de opções viáveis para o uso de banco de dados em arquiteturas multi-*tenancy*. Possibilitando assim, que esse tipo de arquitetura, emergente e essencial para softwares modernos, possa ser cada vez mais difundida e usada em aplicações comerciais, justificando assim, todo o empenho e importância do trabalho aqui apresentado.

O trabalho está organizado da seguinte forma: na seção 2 é apresentado o referencial teórico. Esta seção tem o objetivo de apresentar uma visão geral de cada tema que está ligado com o assunto principal do trabalho. Para isso, o referencial apresenta os conceitos de computação em nuvem, arquitetura SaaS, aplicações multi-tenants e banco de dados relacionais. Na sessão 3 são apresentados as técnicas e métodos utilizados no delineamento da pesquisa. A sessão 4 apesenta os detalhes do estudo de caso utilizado, bem como detalhes de configuração e do processo de testes. Os resultados obtidos são apresentados na sessão 5. Nesta sessão são vistos os compilados e os comparativos dos cenários e modelos propostos. As considerações finais, referentes a pesquisa, são vistos na sessão 5.

2. Fundamentação teórica

2.1. Computação em nuvem

A computação em nuvem vem se apresentando uma das maiores mudanças na área de Tecnologia da Informação (TI) após os anos 2000, principalmente sendo impulsionado pelas grandes empresas de tecnologia do mundo, como *Google* e *Amazon*. Segundo o

International Data Corporation (IDC) os gastos com computação em nuvem em 2019 devem chegar em US $ 210 bilhões, um aumento de 23,8% em relação a 2018 e até 2022 este número deve chegar a US $ 370 bilhões (IDC, 2019). Estes números demonstram a importância que o tema tem ganhado com o passar do tempo. A definição formal do termo 'computação em nuvem' ou *cloud computing* é definido por NIST (2011) como:

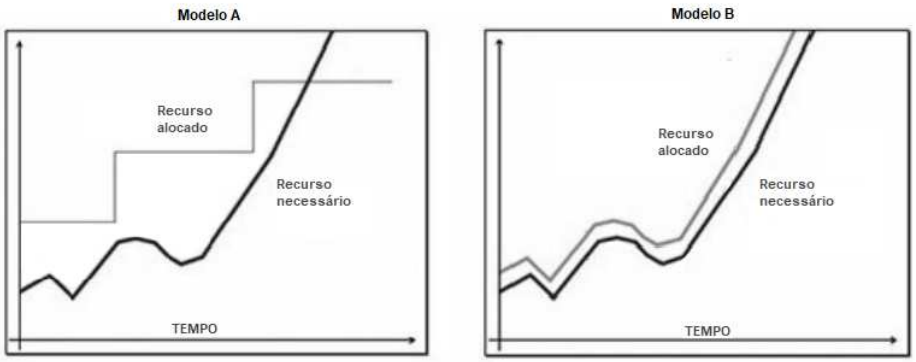
> Um modelo para permitir acesso de rede onipresente e conveniente a um conjunto compartilhado de recursos de computação configuráveis (por exemplo, redes, servidores, armazenamento, aplicativos e serviços) que podem ser rapidamente provisionados e liberados com esforço mínimo de gerenciamento ou interação com o provedor de serviços.

Veras (2012) apresenta uma definição um pouco diferente, com um foco no processo de utilização:

> (CLOUD COMPUTING) é substituir ativos de TI que precisam ser gerenciados internamente por funcionalidades e serviços do tipo pague-conforme-crescer a preços de mercado. Estas funcionalidades e serviços são desenvolvidos utilizando novas tecnologias como a VIRTUALIZAÇÃO, arquiteturas de aplicação e infraestrutura orientadas a serviço e tecnologias e protocolos baseados na Internet como meio de reduzir os custos de hardware e software usados para processamento, armazenamento e rede.

Veras (2012) também afirma que o grande objetivo da computação em nuvem é o melhor aproveitamento da infraestrutura de TI, sendo que a elasticidade é o componente principal. No modelo de hospedagem própria, os recursos são escalonados de forma segmentada, e não acompanham a real utilização dos recursos. Outro ponto problemático é que os recursos apenas aumentam e dificilmente se consegue baixá-los sem que ocorra desperdício de hardware. Já no modelo de computação em nuvem, a real utilização e a alocação dos recursos ficam próximas uma da outra. A Figura 1 demostra isso:

Figura 1. Alocação de recursos em *cloud*

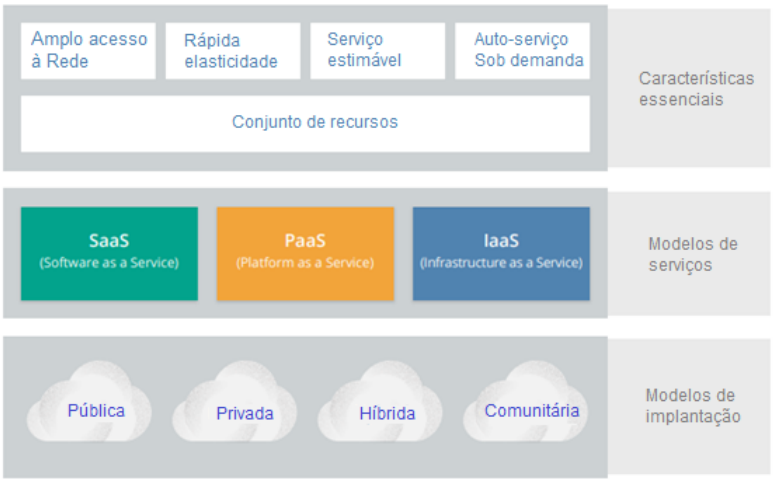


Fonte: Veras (2012)

Como é visto na Figura 1 tem-se o modelo A, baseada em uma infraestrutura dedicada local. Neste modelo é possível ver que o recurso alocado é escalado em etapas e de forma gradual, esse fenômeno ocorre, pois, os *upgrades* de hardware são realizados em espasmos conforme a necessidade de recurso ocorre ou está prestes a alcançar o recurso disponível. Este tipo de modelo gera uma zona de desperdício de infraestrutura grande, pois o recurso está alocado, mas não há uso. Já no modelo B tem-se um modelo baseado nos princípios de nuvem, onde há um crescimento mais elástico entre a necessidade real e o recurso alocado, gerando assim um desperdício mínimo dos recursos.

A Cloud Security Alliance (CSA, 2017) apresenta uma visão geral sobre computação em nuvem, dividindo-a em 3 aspectos essenciais que devem ser entendidos, sendo eles, as características principais, os modelos de serviços e os modelos de implantação. A Figura 2 mostra um quadro resumo das características essenciais, modelos de serviços e modelos de implantação definidos pela *Cloud Security Alliance*, explicadas nos itens seguintes.

Figura 2. Quadro CSA



Fonte: CSA (2017)

2.1.1. Características essenciais

NIST (2011) define as cinco características essenciais da computação em nuvem, sendo elas amplo acesso à rede, rápida elasticidade, serviços estimáveis, auto-serviço/sob demanda e conjunto de recursos, a saber:

- Amplo acesso à rede: os recursos estão disponíveis na rede e são acessados por meio de mecanismos padrão que promovem o uso por plataformas heterogêneas de *thin-client* (por exemplo, telefones celulares, tablets, laptops e estações de trabalho);
- Rápida elasticidade: os recursos podem ser provisionados e liberados elasticamente, em alguns casos automaticamente, para escalar rapidamente para fora e para dentro de acordo com a demanda. Para o consumidor, os recursos disponíveis para provisionamento geralmente parecem ilimitados e podem ser alocados em qualquer quantidade e a qualquer momento;
- Serviço estimável: os sistemas em nuvem controlam e otimizam automaticamente o uso de recursos aproveitando alguma métrica com nível de abstração apropriado ao tipo de serviço (por exemplo, armazenamento, processamento, largura de banda e contas de usuário ativas). O uso de recursos pode ser monitorado, controlado e divulgado, fornecendo transparência tanto para o provedor quanto para o consumidor do serviço utilizado;
- Auto-serviço/sob demanda. Um consumidor pode provisionar unilateralmente recursos de computação, como tempo de processamento e armazenamento de rede, conforme necessário, automaticamente, sem exigir interação humana com cada provedor de serviços;
- Conjunto de recursos. Os recursos de computação do provedor são agrupados para atender a vários consumidores usando um modelo de multi-locação, com diferentes recursos físicos e virtuais atribuídos e reatribuídos dinamicamente de acordo com a demanda do consumidor. Existe uma sensação de localização independente em que o cliente geralmente não tem controle ou conhecimento sobre a localização exata dos recursos fornecidos, mas pode ser capaz de especificar a localização em um nível mais alto de abstração (por exemplo, país, estado ou datacenter). Exemplos de recursos incluem armazenamento, processamento, memória e largura de banda de rede.

2.1.2. Modelos de implantação

NIST (2011) define que uma nuvem pode ser implantada em quatro maneiras distintas, diferenciando-se pela forma de gerenciamento, operação, propriedade e acesso. Os modelos definidos são: nuvem privada, comunitária, pública e híbrida, com as seguintes características:

- Nuvem privada: a infraestrutura de nuvem é provisionada para uso exclusivo por uma única organização que inclui vários consumidores (por exemplo, áreas de negócios dentro da organização). Ela pode ser de propriedade, gerenciada e operada pela organização, por terceiros ou por alguma combinação deles, e pode existir dentro ou fora das instalações da empresa;

- Nuvem comunitária: a infraestrutura em nuvem é provisionada para uso exclusivo por uma comunidade específica de consumidores de organizações que compartilham preocupações (por exemplo, missão, requisitos de segurança, políticas e considerações de conformidade). Ela pode ser de propriedade, gerenciada e operada por uma ou mais organizações na comunidade, um terceiro ou uma combinação deles, e pode existir dentro ou fora das instalações da comunidade;
- Nuvem pública: a infraestrutura de nuvem é provisionada para uso aberto pelo público em geral. Ela pode ser de propriedade, gerenciada e operada por uma organização comercial, acadêmica ou governamental ou por alguma combinação deles. Ela existe nas premissas do provedor de nuvem;
- Nuvem híbrida: a infraestrutura de nuvem é uma composição de duas ou mais infraestruturas de nuvem distintas (privada, comunitária ou pública) que permanecem como entidades exclusivas, mas são unidas por tecnologia padronizada ou proprietária que permite dados e aplicativos para portabilidade (por exemplo, transbordo de nuvens para balanceamento de carga entre elas).

2.1.3. Modelos de serviço

A NIST (2011) defini os modelos de serviço de nuvem em três tipos, sendo eles:

- Software como serviço (SaaS): a capacidade oferecida ao consumidor de usar os aplicativos do provedor em execução em uma infraestrutura de nuvem. O consumidor não gerencia ou controla a infraestrutura de nuvem subjacente, incluindo rede, servidores, sistemas operacionais, e armazenamento;
- Plataforma como serviço (PaaS): a capacidade oferecida ao consumidor de implementar na infraestrutura em nuvem os aplicativos criados ou adquiridos pelo consumidor, criados usando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor. O consumidor não gerencia ou controla a infraestrutura de nuvem subjacente, incluindo rede, servidores, sistemas operacionais ou armazenamento, mas tem controle sobre os aplicativos implantados e possivelmente definições de configuração para o ambiente de hospedagem de aplicativos;
- Infraestrutura como Serviço (IaaS): a capacidade oferecida ao consumidor de usar o processamento, armazenamento, redes e outros recursos fundamentais de computação, nos quais o consumidor é capaz de implantar e executar softwares arbitrários, que podem incluir sistemas operacionais e aplicativos. O consumidor não gerencia ou controla a infraestrutura de nuvem subjacente, mas tem controle sobre sistemas operacionais, armazenamento e aplicativos implantados.

Já, a ISO 17788:2014 (ISO/IEC 17788:2014) tem uma abordagem um pouco diferente, apresentando os três modelos acima, como tipo de capacidades de nuvem ou '*cloud capabilities types*' e apresenta as categorias de serviços como:

- Comunicações como serviço (CaaS): categoria de serviço de nuvem na qual o recurso fornecido ao cliente do serviço de nuvem é interação e colaboração em tempo real;

- Computação como serviço (CompaaS): uma categoria de serviço de nuvem na qual os recursos fornecidos ao cliente do serviço de nuvem são o fornecimento e o uso dos recursos de processamento necessários para implantar e executar um software;
- Armazenamento de dados como um serviço (DSaaS): Uma categoria de serviço de nuvem na qual o recurso fornecido ao cliente do serviço de nuvem é o fornecimento e uso de armazenamento de dados e recursos relacionados;
- Infraestrutura como serviço (IaaS): uma categoria de serviço de nuvem na qual o tipo de capacidade de nuvem fornecido ao cliente de serviço de nuvem é um tipo de capacidade de infraestrutura;
- Rede como serviço (NaaS): uma categoria de serviço de nuvem na qual o recurso fornecido ao cliente do serviço de nuvem é a conectividade de transporte e os recursos de rede relacionados;
- Plataforma como serviço (PaaS): uma categoria de serviço de nuvem na qual o tipo de capacidade de nuvem fornecido ao cliente do serviço de nuvem é um tipo de recurso de plataforma;
- Software como serviço (SaaS): uma categoria de serviço de nuvem na qual o tipo de capacidade de nuvem fornecido ao cliente do serviço de nuvem é um tipo de recurso do aplicativo.

O Quadro 1 mostra um resumo das categorias de serviços em nuvem, apresentando qual os tipos de capacidades que são utilizadas por cada serviço. Sendo que um mesmo serviço pode utilizar uma ou mais capacidades. (ISO/IEC 17788:2014).

Quadro 1. Categorias de serviços por tipos de capacidades

Categorias de serviços em nuvem	**Tipos de capacidades de nuvem**		
	Infraestrutura	**Plataforma**	**Software**
Comunicações (CaaS)		X	X
Computação (CompaaS)	X		
Armazenamento (DSaaS)	X	X	X
Infraestrutura (IaaS):	X		
Rede (NaaS)	X	X	X
Plataforma (PaaS)		X	
Software (SaaS)			X

Fonte: ISO/IEC 17788:2014

2.2. Arquitetura SaaS

Existem muitas definições para SaaS, entre as apresentadas anteriormente, destaca-se a de Carraro e Chong (2006) onde SaaS é um "Software implantado como um serviço hospedado e acessado pela Internet.". Já a definição detalhada do NIST (2011) sobre o termo é:

> A capacidade oferecida ao consumidor de usar um provedor de aplicativos em execução em uma infraestrutura de nuvem. As aplicações são acessíveis a partir de vários dispositivos clientes por meio de uma interface *thin-client*, como um navegador da web (por exemplo, e-mail baseado na web) ou uma interface de programa. O consumidor não gerencia ou controla a infraestrutura de nuvem subjacente, incluindo rede, servidores, sistemas operacionais, armazenamento ou até mesmo recursos de aplicativos individuais, com a possível exceção de configurações limitadas de aplicativos para usuários específicos.

Outra definição amplamente utilizada em trabalhos anteriores é a de WANG (2008), que define SaaS como um software pessoal ou empresarial que é hospedado como serviço e oferecido aos clientes através da internet. Basicamente todos os termos se convergem ao um significado. Sendo assim, é possível definir que SaaS como um software que não precisa ser instalado no cliente, o cliente paga pelo uso e não tem acesso as configurações de hardware.

Chong e Carraro (2006) dividem SaaS em pelo menos duas categorias, sendo elas:

- Serviços de linha de negócios (*line-of-business*): oferecidos a empresas e organizações de todos os portes. Os serviços de linha de negócios são muitas vezes soluções de negócios grandes e personalizáveis, destinadas a facilitar os processos de negócios, como como finanças, gestão da cadeia de suprimentos e relações com clientes. Esses serviços geralmente são vendidos aos clientes com base em assinatura. Como exemplo se tem Salesforce.com [SalesForce, 2019]
- Serviços orientados para o consumidor: sistemas oferecidos ao público em geral. Os serviços orientados para o consumidor são às vezes vendidos por assinatura, mas são frequentemente fornecidos aos consumidores sem nenhum custo, e são mantidos por venda na publicidade. O exemplo mais clássico desse tipo de serviços são os provedores de email, como Gmail, Yahoo e etc.

Esse tipo de software tem o principal objetivo facilitar o acesso aos possíveis clientes e baratear os custos de implantação. Uma vez que não há nenhuma necessidade de implantação no cliente. Chong e Carraro (2006) citam a chamada economia de cauda longa. O termo se refere ao modelo de negócio baseado em clientes de baixo retorno, onde o lucro é baseado na alta escala. Segundo os autores o modelo SaaS permite a abertura desse novo mercado, onde clientes que não estariam aptos para adquirirem uma aplicação convencional, conseguem, principalmente devido ao baixo custo e ao licenciamento diferenciado, adquirirem aplicações SaaS. A Figura 3, demostra esse novo mercado citado pelos autores.

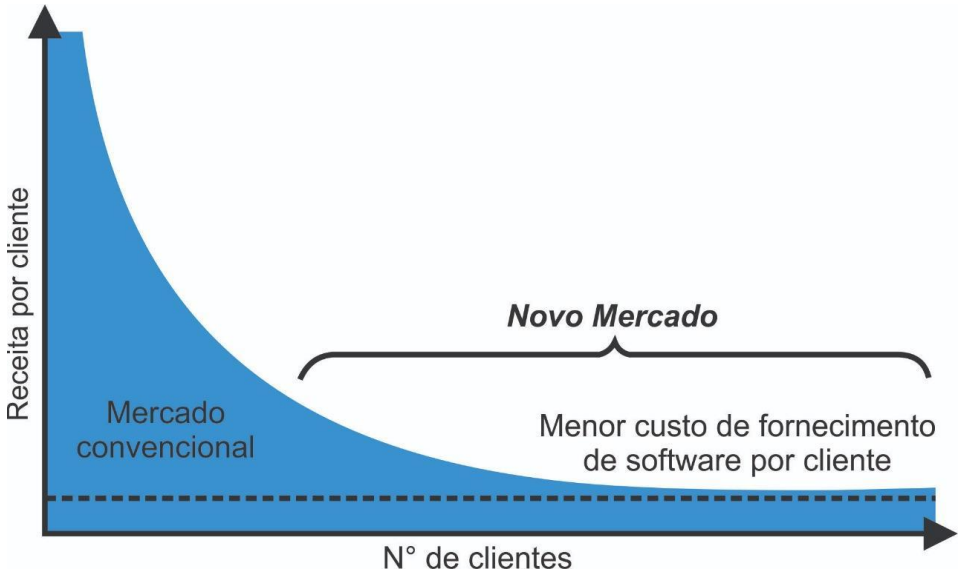


Figura 3. Novo mercado

Fonte: Chong e Carraro (2006)

Algumas outras vantagens do modelo SaaS são citadas por Velte, Velte e Elsenpeter (2010):

- Há um menor tempo para avaliar e melhorar a produtividade, quando comparado aos longos ciclos de implementação e à taxa de falhas do software corporativo;
- Existem custos menores de licenciamento de software;
- As ofertas de SaaS apresentam a maior economia de custo em relação ao software instalado localmente, eliminando a necessidade de as empresas instalarem e manterem hardware, pagarem custos de mão-de-obra e manter os aplicativos atualizados;
- O SaaS pode ser usado para evitar os ciclos de desenvolvimento personalizados para obter softwares para a organização rapidamente;
- Os fornecedores de SaaS geralmente têm auditorias de segurança muito meticulosas.

Contudo, Velte, Velte e Elsenpeter (2010) também citam a problemática no desenvolvimento desse tipo de aplicativo. Os autores citam além dos problemas técnicos, problemas culturais, como por exemplo, a cultura corporativa dominada pela inovação de engenharia e uma mentalidade de vendas de licenças. Sendo que um modelo de negócios baseado na venda de software licenciado não se transforma facilmente em um modelo de venda de assinaturas.

Já nos quesitos técnicos, o problema está ligado diretamente a arquitetura do software, devido ao trabalho árduo que é o planejamento de um modelo que seja capaz

de ser utilizado por milhares de clientes. Esse tipo de modelo de aplicação com diversos usuários de diversos clientes é chamado de *multi-tenants* ou multi-locatários (BEZEMER e ZAIDMAN, 2010). Cada cliente da aplicação é considerado um locatário e cada locatário por sua vez pode possuir mais de um usuário. Neste contexto, existe mais esse desafio nas aplicações SaaS, que é o desenvolvimento de uma estrutura correta e eficiente de uma arquitetura *multi-tenants*, seguindo os conceitos e premissas que são apresentados a seguir.

2.3 Aplicações Multi-tenants

Uma aplicação multi-tenancy ou multi-inquilino, é um sistema que permite aos clientes (inquilinos) compartilharem os mesmos recursos de hardware, oferecendo-lhes um aplicativo compartilhado e uma instância de banco de dados, enquanto possibilita que eles configurem o aplicativo para atender às suas necessidades, como se o aplicativo fosse executado em um ambiente dedicado [BEZEMER e ZAIDMAN, 2010]. Ou seja, todos os clientes utilizam o mesmo recurso, permitindo que cada cliente tenha vários usuários trabalhando ao mesmo tempo e compartilhando o mesmo recurso com usuários de outros clientes.

A ISO 17789:2014 (ISO/IEC 17789:2014) ressalta a importância do isolamento e confidencialidade na arquitetura multi-tenants:

> Multi-tenancy é a alocação de recursos físicos ou virtuais para que vários locatários com seus processamentos e dados isolados e inacessíveis uns aos outros. Em outras palavras, os usuários que pertencem a uma locação devem estar completamente inconscientes da presença de usuários de outra locação. A multilocação não afeta apenas os serviços em nuvem; Ele também afeta os recursos de negócios e administração oferecidos aos clientes de serviços de nuvem pelo provedor de serviços de nuvem. As informações sobre contas de usuários, assinaturas, uso e faturamento devem ser mantidas isoladas e visíveis apenas para os clientes que possuem as locações relacionadas. Um cuidado especial deve ser tomado em relação a recursos, como arquivos de *log*, que podem conter registros relacionados a vários inquilinos. Se um determinado cliente precisar acessar os registros de log, por exemplo, quando ocorrer um incidente, os registros de *log* deverão ser filtrados para que o cliente possa ver apenas os registros relacionados a seus arrendamentos.

Os autores Benemer e Zaidman (2010) apresentam três características principais para o modelo *multi-tenants*, sendo elas:

- Compartilhamento de recursos de hardware: a possibilidade de colocar muitos inquilinos no mesmo hardware melhora a utilização do servidor, ou seja, quanto mais próxima a taxa de uso com a taxa de recurso alocado menor será o preço médio por aplicativo, resultando assim em custos gerais mais baixos;
- Alto grau de configurabilidade; uma vez que todos os inquilinos compartilham o mesmo aplicativo, é necessário que o aplicativo seja altamente configurável para que atenda as necessidades especificadas de cada cliente. Necessidade

desde questões de layout, como necessidades referentes a legislação, devido ao mesmo aplicativo ser utilizado por clientes de países diferentes;

- Aplicativo Compartilhado e Instância do Banco de Dados: um aplicativo de um único locatário pode ter muitas instâncias em execução e elas podem ser diferentes umas das outras devido à personalização. Em aplicações multi-inquilinos, essas diferenças não existem. Isso gera a possibilidade que uma aplicação tenha um número total de instancias menor e isso refrete em custo mais baixo. Neste contexto, o banco de dado também pode ser acessado de forma única, sendo que a aplicação por ter acessos a várias bases de dados e gerar agregações de informações para melhorar a experiência do usuário final, com base nos traços de comportamento coletados.

Chong e Carraro (2006) propõem que existem três características principais em uma aplicação SaaS, sendo elas, escalabilidade, eficiência em termos de *multi-tenant* e configuração. Com base nessas três características são definidos quatro níveis de maturidade, sendo que cada nível se diferencia do anterior pela adição de umas das três características citadas. A Figura 4 demostra os quatros níveis propostos, explicados a seguir.

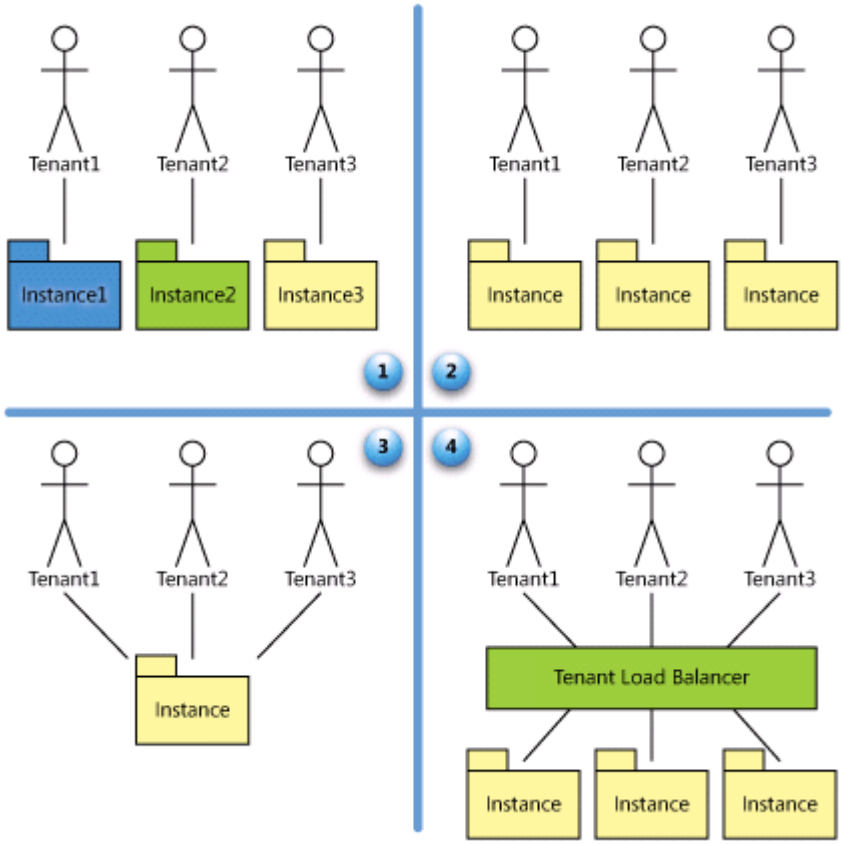


Figura 4. Modelos de maturidade

Fonte: Chong e Carraro (2006)

- Nível 1 – Customizado: o primeiro nível proposto é muito semelhando ao modelo tradicional de entrega de software, ele se baseia ao padrão ASP (*Application Service Provider*) dos anos 90. Neste modelo cada cliente possuí uma versão própria e personalizado do aplicativo que é hospedado no mesmo host. A diferença desse modelo ao tradicional, é que o a instancia do aplicativo é independente de qualquer outra, sendo assim cada cliente possuí um sistema próprio mesmo sendo executado em um host compartilhado. Este modelo traz os benefícios referentes a implantação e de desenvolvimento, como o modelo se baseia no modelo tradicional de software, a migração de uma aplicação comum para uma migração SaaS geralmente se torna mais simples e mais rápida. Contudo o modelo não é o mais eficaz em nível de manutenção e custos;
- Nível 2 – Configurável: o segundo nível assemelhasse muito ao primeiro, neste nível cada cliente tem uma instancia dedicada para si, contudo, a aplicação é comum a todos os clientes e replicada para cada um. As customizações não existem, as necessidades especiais e individuais de cada cliente, como regras de negócio, aparência e necessidades do tipo, são atendidas a nível de configuração. Enquanto o nível 1 cada cliente possuí a personalização em código, o nível 2 consegue atender aos mesmos requisitos apenas com configurações. Este modelo se sobre saí ao modelo anterior, pois traz a vantagem referente a manutenção da aplicação, uma vez que todas as aplicações possuem o mesmo código fonte, a complexidade de atualização e manutenção é reduzida drasticamente, contudo, neste nível, as mudanças arquitetônicas da aplicação já são bem mais impactantes. Para que uma aplicação consiga atender um leque de clientes apenas com configurações, a estrutura do sistema já deve estar preparada para isso, isso pode dificultar a reengenharia de aplicações originalmente não desenvolvidas para multi-tenants;
- Nível 3 – Configurável e eficiência em multi-locatários: o terceiro nível é uma evolução no segundo. Enquanto o segundo nível cada cliente possuí uma instância própria para si, neste nível todos os clientes utilizam a mesma instancia aplicativo. Para se alcançar esse nível é necessária uma evolução nos níveis de acesso da aplicação, para garantir que um cliente não interfira em outros. Esta abordagem é mais vantajosa que a anterior, pois mesmo que no nível 2 todos os clientes utilizem o mesmo código, este código estava replicado, assim gerando um uso maior de recursos de hospedagem. A desvantagem desse modelo é que a sua escalabilidade é limitada, sendo que todos os clientes ficam centralizados em apenas uma instancia, sendo que a única maneira de escala é no sentido de aumento de recursos;
- Nível 4 – Escalável, configurável e eficiência em multi-locatários: neste nível a aplicação já conta com uma componente chave, o balanceador de carga. Neste nível todos os clientes acessam o balanceador que por sua vez distribuí as requisições para um conjunto de instancias da aplicação. Todas essas instancias são réplicas umas das outras e sua quantidade pode aumentar e diminuir conforme a necessidade da demanda. Assim o limite de clientes simultâneos é ilimitado sem que a experiencia do usuário fique comprometida.

2.4 Banco de dados relacionais

2.4.1 Modelo relacional

Os bancos de dados em aplicações sempre foram fundamentais, pois representam o local onde os dados são persistidos para a utilização futura. Um banco de dados pode ser dito como um conjunto de dados relacionados [Elmasri e Navathe, 2005] e com um domínio específico. Para este armazenamento dos dados existem tipos de modelos de implementação, que vem evoluindo ao longo do tempo, tais como os modelos baseado em arquivo, os modelos relacionais, entre outros.

O modelo relacional hoje é o modelo mais utilizado nos sistemas corporativos atuais, principalmente devido à segurança em que o modelo se baseia e por sua facilidade de uso, em comparação a modelos anteriores [Elmasri e Navathe, 2005]. Este modelo, como o nome se refere, é baseado em relacionamentos entre coleções, ou as chamadas tabelas. O modelo se utiliza de um nível de abstração, expondo apenas os níveis lógicos e abstraindo os detalhes de armazenamento dos dados, facilitando assim a sua utilização. Os dados são organizados em tabelas de valores lógicos, que possuem colunas, que simbolizam as propriedades.

A criação de um banco de dados relacional possuí as seguintes propriedades implícitas [Moreira, Sousa e Machado, 2012]:

- O banco de dados representa alguns aspectos do mundo real, sendo chamado, às vezes, de minimundo ou de universo de discurso (UoD). As mudanças no minimundo são refletidas em um banco de dados;
- Um banco de dados é uma coleção lógica e coerente de dados com algum significado inerente. Uma organização de dados ao acaso (randômica) não pode ser corretamente interpretada como um banco de dados;
- Um banco de dados é projetado, construído e povoado por dados, atendendo a uma proposta específica. Possui um grupo de usuários definido e algumas aplicações preconcebidas, de acordo com o interesse desse grupo de usuários.

Uma aplicação moderna geralmente não acessa diretamente a base de dados, na maioria dos casos o banco de dado é construído e acessado através de um sistema de gerenciamento de base de dados (SGBD), como por exemplo o SQL Server, Oracle e MySQL. Estes sistemas possuem nativamente recursos de acesso e manutenção dos dados, simplificando ainda mais a sua utilização. Também são disponibilizadas rotinas administrativas, como backups por exemplo, deixando assim que a aplicação fim acabe apenas se preocupando com o domínio para o qual foi proposta.

2.4.2 Banco de dados em aplicações SaaS

Segundo Veras (2012) o principal motivo para a não adoção de um modelo baseado em SaaS é falta de confiança. Como a ideia principal de uma arquitetura SaaS é oferecer recursos centralizados e de menor custo, é necessário que haja um certo nível de compartilhamento de recursos que dependendo do modelo de negócio pode ser mais aceito ou não. Este problema ainda fica mais evidente quando se trata dos dados das

aplicações, pois estes são a parte mais importante da aplicação, precisando assim de um nível de isolamento adequado. Uma vez que a geração de informação geralmente é objetivo final de qualquer aplicação. Sendo assim, a criação de uma arquitetura robusta e eficiente de armazenamento de dado se torna fundamental para que uma aplicação *multi-tenants* seja aceita pelo mercado e se torne confiável.

Para a arquitetura de dados em aplicações SaaS existem diversos modelos [Veras, 2012]. Neste trabalho são abordados os 3 modelos principais, sendo elas com um banco de dados por locatário, banco de dados compartilhado entre locatários e pool compartilhado entre banco de dados.

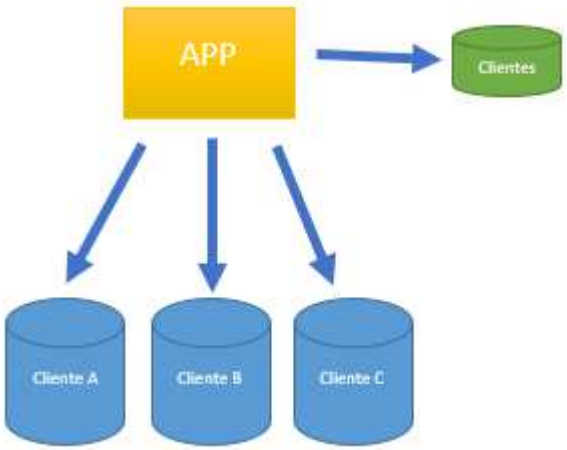


Figura 5. Banco de dados por locatário

Fonte: Elaborado pelo autor

O modelo mais clássico e de mais simples implementação é o modelo com cada locatário com uma base de dados individual. A Figura 5 mostra como este modelo é implementado. Neste modelo geralmente a aplicação é única, porém capaz de conectar com diversas bases de dados ao mesmo tempo, sendo que para fazer isso, ela se utiliza de uma base de dados auxiliar.

Nesta base de dados existe uma relação de todos clientes e os dados de conexão de cada locatário. Assim quando um usuário entra na aplicação, ela por sua vez consulta a base de clientes e se conecta com a base correspondente. Este modelo tem algumas vantagens interessantes, primeiramente, conforme comentado, a simplicidade de implementação e em segundo ponto, o alto nível de isolamento entre as bases. Como cada locatário possuí uma base distinta, não existe nenhum tipo de interferência de um locatário com outro locatário.

Em contraponto, neste modelo existem duas principais desvantagens, uma sendo o nível de manutenção e o outro é o custo. Quando existe uma base para cada locatário é necessário prever rotinas de manutenção par cada base, por exemplo, se existe mil bases é necessário a realização de mil backups. Esta lógica se vale para qualquer rotina de manutenção de base, como monitoramento, desfragmentação de índices, entre outros. Já, na questão do custo, existe um acréscimo, principalmente devido ao fato que cada base de dados possuí dados alocados, que não se referem aos dados do sistema

fim, e sim dados referentes ao funcionamento básico, como por exemplo, estrutura de tabelas, índices, procedures entre outros.

Ou seja, neste modelo pode-se afirmar que há replicação de informação entre locatários, pois os dados comuns entre as bases são replicados para cada cliente, gerando assim um acréscimo de espaço e necessidade maior de armazenamento, por consequência, uma maior necessidade de processamento.

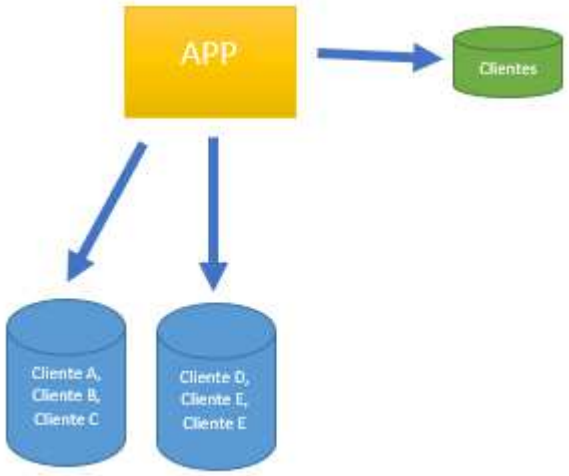


Figura 6. Banco de dados compartilhado entre locatários

Fonte: Elaborado pelo autor

O segundo modelo é o banco de dados compartilhado entre locatários. A Figura 6 mostra este modelo, que se difere do anterior, pelo fato que a mesma base de dados é dividida entre vários locatários. Sendo este o modelo geralmente com o menor custo de hardware (Veras, 2012). Parecido com o modelo anterior, a aplicação de utiliza de um banco auxiliar para fazer a conexão com a base de dados do locatário, porém como a conexão é compartilhada com mais de um locatário a aplicação recupera também algum identificador único dos locatários. Este identificador único é utilizado nas operações com o banco de dados, para que assim os dados sejam filtrados e dados de outros locatários sejam omitidos.

Para fazer este isolamento existem muitas técnicas, uma delas é a construção de *views* ou visões, que são a representação da tabela lógica com o filtro já configurado. A Figura 7 mostra um exemplo dessa abordagem.

Como pode ser visto na Figura 7, na tabela onde contém os registros é adicionado um campo que não faz parte da entidade. Este campo denominado de ‘idCliente’, serve apenas para identificar a qual cliente aquele registro pertence. Se a aplicação acessar diretamente a tabela, ela acabará listando todos os dados de todos os clientes, comportamento esse que na maioria das vezes não deve acontecer.

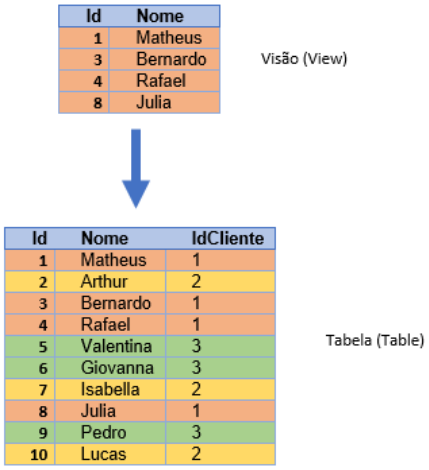


Figura 7. Tabela particionada

Fonte: Elaborado pelo autor

Por este motivo, uma opção é a utilização de uma camada superior de abstração, neste caso um *view*. Este objeto é o responsável por isolar os dados dos clientes, bem como não permitir que dados de outros clientes sejam acessados. Para fazer isso, na criação da visão já se adiciona um filtro com o valor que deve ser filtrado, este valor geralmente é acessado por uma variável de contexto ou outra técnica parecida. Uma variável de contexto é um valor de acesso global e que é imutável dentro de uma conexão.

Assim, quando uma conexão for estabelecida, esta variável é criada e a visão passa a consumi-la. Por exemplo, no caso da Figura 7, entende-se que a variável de contexto utilizada na conexão foi o "IDCliente" igual a 1, pois os dados que a visão consegue acessar são apenas os dados do cliente 1, assim caso a conexão fosse do cliente 2, a variável seria configurada para 2 e apenas os dados do cliente 2 seriam vistos. Este é apenas um modelo de utilização de uma base de dados compartilhada, existem outros modelos, como por exemplo, ao invés de particionar a tabela, se duplica a tabela, criando uma tabela para cada cliente.

O que todo o modelo de banco de dados compartilhado tem em comum é um acréscimo da complexidade da implementação da solução. Este problema ainda se mostra maior nas migrações de sistema originalmente desenvolvidos para a utilização de base de dados únicas, pois em seus projetos não são previstos acesso múltiplos em dados de vários clientes.

O modelo baseado em banco de dados compartilhado traz a economia como vantagem, esta economia vai dos custos de manutenção até os custos relativos a ajustes estruturais, pois como vários clientes acessam a mesma base, não é necessário realizar ajustes para cada cliente e sim apenas uma vez por base de dados. Conforme dito anteriormente, também o consumo de espaço físico e processamento pode ser considerado menor, uma vez que não há índice de replicações de informações estruturais da base de dados.

Já, no lado das desvantagens do modelo, existe a questão, já comentada, das aplicações terem que ser desenvolvidas, ou pelo menos adaptadas, para trabalhar com mais de um cliente por base. Esse fato pode gerar um custo extra de implementação que deve ser levado em consideração na hora do projeto, principalmente em relação ao desenvolvimento da área de segurança (Veras, 2012). Outra desvantagem deste modelo é o nível de isolamento reduzido. Como a base é compartilhada, mesmo os dados sendo isolados entre os clientes com alguma técnica como mostrado aqui, a nível de performance este modelo não conseguirá manter o isolamento. Quando cargas de trabalhos de um cliente forem mais elevadas, os outros clientes provavelmente deverão sentir em seus trabalhos.

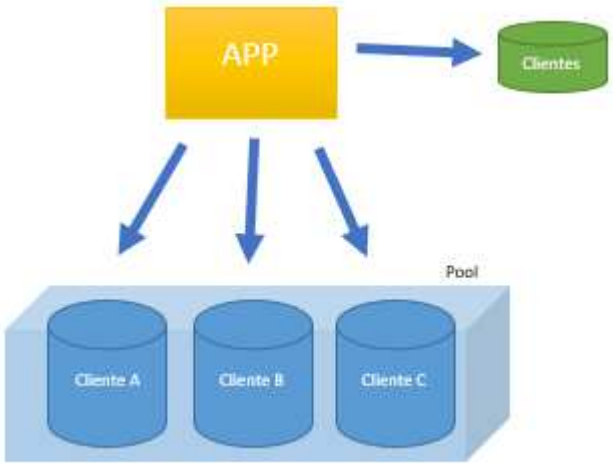


Figura 8. Pool compartilhado entre banco de dados

Fonte: Elaborado pelo autor

O terceiro modelo de arquitetura, é baseado em dois conceitos da computação em nuvem. Um deles é um conceito mais novo, considerado ainda emergente, que é *Database as a Service* (DBaaS) (ISO/IEC 17788:2014). Como o nome já sugere, a ideia desse principio é oferecer banco de dados como serviço. Esse serviço abstraí qualquer interação do cliente com a máquina hospedeira, desde de requisitos de hardware até requisitos de sistema operacional. Basicamente o cliente não faz nenhum tipo de gestão nesses requisitos, ficando assim apenas responsável pela gestão de sua base de dados.

O outro conceito por trás do terceiro modelo, é o conceito de *pool* de recursos ou também chamado de 'conjunto de recursos' (NIST, 2011). Como já mencionado anteriormente, esta característica se baseia na possibilidade de um grupo de recursos computacionais serem compartilhados com vários clientes e reatribuídos dinamicamente de acordo com a demanda do consumidor. Apresentando uma sensação de independência entre os clientes, mesmo esses clientes estando compartilhando os recursos.

Como pode ser visto na Figura 8, a ideia desse modelo é a existência de banco de dados individuais, um para cada cliente. Contudo, esses bancos de dados possam compartilhar os recursos computacionais entre eles, podendo inclusive, que haja a

divisão igual ou não dos recursos entre eles. Essa divisão poderia ser feita com limites pré-determinados ou conforme a necessidade de uso.

2.5 Trabalhos relacionados

Apesar de não ter sido possível encontrar trabalhos fortemente relacionados ao presente trabalho, é possível citar diversas publicações referentes ao tema e com abordagens parecidas com a utilizada.

Destes trabalhos, é possível destacar Matthew (2016) onde é examinado os fatores que influenciam a adoção de diferentes modelos de banco de dados *multi-tenats*, considerando os requisitos e desafios da implementação. Destaca-se no trabalho a profunda pesquisa bibliográfica e seus resultados qualitativos e quantitativos, apresentados a partir dos dados obtidos de questionários, gerando assim informações para a elaboração de medidas para a melhoria da aceitação desse tipo de banco de dados.

Já, no trabalho de Schiller (2015) é possível ver um aprofundamento no conceito de banco de dados compartilhado. Apresentando as variações do modelo, como tabelas compartilhadas, esquemas compartilhados e outros. O trabalho apresenta uma proposta do modelo com ênfase no isolamento, se baseando em uma extensão do modelo e seguindo um conceito de contexto por locatário.

Molka (2018), por sua vez, apresenta um enfoque nos bancos de dados multi-tenants em SGBDs *in-memory*. Tendo como motivação os avanços na tecnologia de armazenamento de bases de dados diretamente em memória volátil, técnica que garante maior performance. Este trabalho se utilizou de uma avaliação da ferramenta de mercado SAP-HANA (SAP-HANA, 2019) para a geração de dados. O trabalho tem como o principal objetivo fornecer suporte a provedores de banco de dados em memória para tomada de decisões, sobre alocação de carga de trabalho e alocação de recursos de uma maneira consciente e eficiente.

Por fim, se faz necessário citar o trabalho de Saleh, Fouad e Abu-Elkheir, (2014). Este trabalho propõe um esquema de banco de dados *multi-tenant* flexível, usando um conjunto de fatores e taxas para ser usado como fator de decisão do modelo a ser utilizado para cada tabela. Os modelos possíveis são: *(i)* bancos de dados separados; *(ii*) tabelas separadas; *(iii)* tabelas compartilhadas em um banco de dados.

Os autores definem dois principais tipos de clientes, os que têm altas cargas de trabalho e tem foco em qualidade de requisitos de serviços, tais como desempenho, segurança e garantia de isolamento, por outro lado, os clientes que têm baixas cargas de trabalho, estão focados em minimizar os custos e reduzir os recursos de hardware.

Para a realização da decisão do esquema a ser utilizado e com base no tipo de cliente, os autores definem algumas taxas que devem ser utilizadas para a definição da melhor opção de armazenamento. As taxas podem ser de alguns tipos, como por exemplo, taxas do sistema, taxas dos inquilinos e entre outras. As taxas do sistema são referentes a números do sistema em si. Exemplos de taxas de sistema utilizado pelos autores é o número de inquilinos e a média de tabelas de cada inquilinos. Já, para taxas dos inquilinos são utilizadas, por exemplo, o número máximo de usuários do cliente.

Por fim, os autores, apresentam um estudo de caso específico, demostrando como a proposta se comportaria em um ambiente real, baseada em um sistema de hotéis.

Como visto acima, o tema da presente pesquisa é de suma de importância para a área da computação, visto que existem diversos trabalhos sendo realizados na área. Apesar disso, tais trabalhos seguiram linhas de pesquisas distintas em suas propostas, em que todos tendem ao mesmo objetivo. Esse objetivo sempre é a melhor escolha de uma abordagem para banco de dados em aplicações *multi-tenancy*, sendo assim, é possível reafirmar a importância do trabalho de pesquisa aqui apresentado. Na seção a seguir, inicia-se a introdução do delineamento da pesquisa e nas seções subsequentes, apresenta-se o desenvolvimento da proposta apresentada.

3. Materiais e métodos

3.1. Delineamento da pesquisa

A pesquisa contém um enfoque qualitativo em nível descritivo [AZEVEDO, MACHADO e SILVA, 2011], dividido em uma revisão bibliográfica e um estudo de caso com a ferramenta *Azure SQL Database Elastic Pool*.

A revisão buscou elucidar os temas envolvidos na problemática, caracterizando-se por uma pesquisa bibliográfica, devido aos estudos ocorrem em materiais já publicados (PRODANOV, FREITAS, 2009). Os temas pesquisados foram computação em nuvem, SaaS, aplicações multi-locatários e banco de dados relacionais.

A pesquisa é do tipo aplicada, com método de estudo de caso (PRODANOV, FREITAS, 2009), onde o objetivo foi gerar dados sobre possíveis cenários e modelos utilizando a ferramenta de testes *OTLP-Bench* [OLTPBenchmark, 2019], afim de possibilitar uma análise sobre eficácia e eficiência do *Azure SQL Database Elastic Pool* no contexto detalhado na seção 4 do artigo.

3.2. Coletas e análise dos dados

A coleta de dados foi baseada em bibliografia e documentos. A coleta bibliográfica ocorreu com base em obras de divulgação e de referência, como periódicos e livros técnicos. Os documentos que foram utilizados podem ser classificados como contemporâneos e não escritos [AZEVEDO, MACHADO e SILVA, 2011], como *logs* e gráficos gerados pela ferramenta OTLP-Bench [OLTPBenchmark, 2019] de teste durante a pesquisa.

A análise de dados se deu por meio de uma análise documental [AZEVEDO, MACHADO e SILVA, 2011], com o tratamento da informação contida nos documentos acumulados ao longo da pesquisa, como dados referentes a coleta bibliográfica e com base no material produzido durante a fase de testes do projeto, e utilizando-se de procedimentos de transformação, afim de gerar informação convenientes ao tema. Sendo que a coleta bibliográfica ocorreu de dezembro de 2018 até abril de 2019, e a coleta dos documentos resultantes da fase de teste ocorreram de abril a junho de 2019.

Os dados coletados foram os arquivos “CSV” e “RES” gerados pela ferramenta *OTLP-Bench,* contendo informações sobre a carga utilizado na base de dados, arquivos “XLS” extraídos do portal *Azure,* contendo informações sobre a utilização de recursos

da plataforma e por fim, arquivos “CSV” extraídos diretamente da base de dados, contendo informações da utilização de recursos de hardware na base de dados.

Esses dados foram agrupados e formatados em planilhas separadas por modelos e cenários (os modelos e cenários são explicados na sessão 4). Após a formatação dos dados foi possível gerar gráficos e as informações que são apresentados na sessão 5.

3.2. Etapas de pesquisa

As etapas de pesquisa desenvolvidas foram o levantamento bibliográfico, definição dos cenários e modelos específicos, apresentação da ferramenta estudada, testes nos cenários e modelos propostos e a análise dos dados coletados nos testes.

O levantamento bibliográfico, apresenta um resumo sobre os temas envolvidos na problemática e este conteúdo é apresentado na seção 2. A definição dos modelos e cenários é apresentado na seção 4. Essa seção apresentada como foram realizadas as definições de ambiente e arquiteturas dos testes. As arquiteturas foram definidas como modelos e os ambientes como cenários. A apresentação da ferramenta estudada também é apresentada na seção 4, juntamente com a apresentação da ferramenta utilizada para testes. Os resultados dos testes, juntamente com a análise dos resultados é apresentado na seção 5. Sendo que essa seção apresenta os dados compilados e as informações geradas na fase de testes.

4 Estudo de caso

4.1 Apresentação do *Azure Sql Database Elastic Pool*

Hoje serviços de hospedagem são cada vez mais comuns no mercado. Por isso as grandes empresas de tecnologia do mundo vêm fazendo grandes investimentos nestas tecnologias. Uma dessas empresas foi a *Microsoft* que desenvolveu sua própria nuvem de serviços, chamado *Azure* (Azure, 2019). O portal *Azure*, hoje é uma das maiores plataformas de serviços em nuvem. Possuindo uma grande gama de serviços computacionais. Como por exemplo, máquinas virtuais, hospedagem de sites, banco de dados entre outros.

Os bancos de dados oferecidos pela plataforma são fornecidos como serviços, sendo assim o cliente final não precisa se preocupar com nenhum requisito de hardware e sistema operacional. O cliente apenas utiliza o banco de dados, com um comportamento muito parecido com um banco de dados local. Apesar da plataforma suportar uma série de SGBDs, o mais utilizado é o SQL Server. Isso principalmente devido ao fato de ser um SGBD da própria empresa. Recentemente a empresa disponibilizou uma nova funcionalidade para os serviços de banco de dados SQL Server, essa funcionalidade é o *pool* elástico. A Microsoft (*Microsoft SQL Database Elastic Pool*, 2019) apresenta essa solução como:

> Pools elásticos do Banco de Dados SQL são uma solução simples e econômica para gerenciar e dimensionar a vários bancos de dados com demandas de uso variadas e imprevisíveis. Os bancos de dados em um pool elástico estão em um único servidor do Banco de Dados SQL do

> Azure e compartilham um número definido de recursos por um preço definido. Os pools elásticos no Banco de Dados SQL do Azure permitem que desenvolvedores de SaaS otimizem o desempenho de preço para um grupo de bancos de dados dentro de um orçamento prescrito oferecendo elasticidade de desempenho para cada banco de dados.

Basicamente, a solução oferece uma maneira para que os bancos de dados utilizem recursos compartilhados. Na abordagem padrão do serviço de banco de dados, quando é realizado a configuração do banco de dados seu poder computacional é configurado e o valor pago pelo cliente é referente a essa configuração. No modelo do *pool*, ao invés de configurar o poder computacional para um banco de dados, essa configuração é realizada no *pool* e o *pool* distribui essa capacidade para seus bancos de dados. Essa distribuição, por sua vez, pode ser feita igualmente entre os bancos ou de maneira desproporcional, com bases com mais capacidades que outras. Outro ponto importante de ser mencionado, é que caso não seja feito nenhuma configuração de limites nas bases, os recursos são distribuídos entre as bases conforme a necessidade de cada uma. Por exemplo, se o *pool* possuir 10X de poder computacional, e tiver 5 bases de dados. Se apenas uma base de dados tiver sendo usada, o *pool* distribuirá 10X somente para está base. Esse tipo de distribuição é o que é chamado de elástico pela Microsoft.

A empresa apresenta essa solução como sendo a solução ideal para soluções SaaS que necessitem de um banco de dados. No decorrer dos testes será analisado o comportamento da ferramenta, gerando assim dados para afirmação se a promessa de eficiência realmente é verdadeira.

4.2 Apresentação da ferramenta *OLTP-Bench*

A análise e ajuste de desempenho nos sistemas SGDB cada vez mais se torna algo fundamental em sistemas modernos, principalmente devido a esse tipo de sistema ser uma parte fundamental em qualquer projeto de software. Esta analise, em questão, é realizado através do uso de um *benchmark* que mede o desempenho de métricas definidas em condições de estresse. O software *OLTP-Bench* [OLTPBenchmark, 2019] é um testador extensível e de código aberto para *benchmarking* de SGDB usando um conjunto diversificado de cargas de trabalho [Difallah, 2014]. As premissas do sistema para avaliação de SGBDs são as seguintes [Difallah, 2014]:

- Escalabilidade Transacional: a capacidade de escalonar para alta *throughputs*, sem ser restringido por clientes;
- Geração Flexível de Carga de Trabalho: a capacidade de gerar cargas de trabalho para sistemas de malha aberta, fechada e semiaberta;
- Controle de taxa refinada: a capacidade de controlar a solicitação de taxas com grande precisão (pequenas oscilações de *throughput* pode dificultar a interpretação dos resultados);
- Cargas de trabalho mistas e em evolução: a capacidade de suportar cargas de trabalho mistas e alterações de taxa, composição e acessar a distribuição das

cargas de trabalho dinamicamente ao longo do tempo, para simular eventos da vida real e cargas de trabalho em evolução;

- Dados e cargas de trabalho sintéticos e reais: a necessidade de lidar com conjuntos de dados e cargas de trabalho sintéticos e reais, para evitar comprometimentos entre testes reais e facilidade de escalabilidade

Diferente da maioria dos sistemas de *benchmarking*, o *OLTP-Bench* foi projetado para conseguir ser configurável para trabalhar em diversas cargas de simulação, desde cargas especificas como cargas chamadas 'públicas', cargas essas que simulam sistemas de uso cotidiano, como a *Twitter* por exemplo. As cargas de testes padrões estão divididas em 3 classificações, sendo elas: *(i)* transacionais, que simulam transações complexas e pesadas; *(ii)* as orientadas a web, que simulam aplicativos webs, como redes sociais por exemplo, geralmente, com cargas não uniformes; e, por fim, *(iii)* as classificadas como testes de funcionalidade, cargas destinadas a testar funções especificas de cada SGBD. A Tabela 1 mostra as 3 classificações e as opções disponíveis, bem como uma breve explicação de cada opção.

Tabela 1 – Tipos de cargas

Class	Benchmark	Application Domain
Transactional	**AuctionMark**	On-line Auctions
	CH-benCHmark	Mixture of OLTP and OLAP
	SEATS	On-line Airline Ticketing
	SmallBank	Banking System
	TATP	Caller Location App
	TPC-C	Order Processing
	Voter	Talent Show Voting
Web-Oriented	**Epinions**	Social Networking
	LinkBench	Social Networking
	Twitter	Social Networking
	Wikipedia	On-line Encyclopedia
Feature Testing	**ResourceStresser**	Isolated Resource Stresser
	YCSB	Scalable Key-value Store
	JPAB	Object-Relational Mapping
	SIBench	Transactional Isolation

Fonte: Difallah (2014)

4.3 Cenários e modelos dos testes

Para realizar os testes, foram definidos 3 modelos de base de dados e 3 cenários. Os modelos têm como objetivo testar a ferramenta *Azure Elastic Pool* comparando-a com os outros modelos apresentados no item 2.4.2. Para cada modelo apresentado, no item citada, foi montado um modelo de teste. Como os testes têm como base multi-locatários, foram criadas 5 bases de dados em cada modelo, exceto no modelo de banco de dados único por locatário, denominado doravante 'Modelo A'. O modelo A não necessita de mais bases, como o modelo é baseado em uma única base por locatário, ela não sofre interferência por outros locatários, tampouco possuí performance maior ou menor dependendo o número de locatários. A modelagem de um banco de dados compartilhada entre vários locatários, denominasse neste trabalho 'Modelo B'. Para este modelo utiliza-se a técnica apresentado no item 2.4.2, onde a tabela física é dívida entre todos os locatários e os acessos a ela ocorrem com o auxílio de uma *view.* O

último modelo, o chamado ‘Modelo C’ é o modelo utilizando um *pool* de recursos, especificamente o *Azure* SQL *Elastic Pool*.

Os cenários definidos para os testes foram pensados para determinar como cada modelo se comporta em situações diferentes. O primeiro cenário utilizado foi chamado de ‘*ReadCommitted’*, este nome se refere ao nível de isolamento do banco de dados utilizado. Este é o isolamento padrão do SQL Server, considerado um isolamento médio, entre performance e controle de concorrência. Para este cenário foram utilizados 10 terminais virtuais na ferramenta OTLP-Bench. Esses terminais são o número de *Threds* que a ferramenta utiliza durante a carga de teste. Este cenário tem o objetivo de gerar uma carga alta, mas tolerável nos modelos. Para fazer um contraponto a esse cenário, foi criado um cenário muito parecido com ele, mas com uma carga de terminais menores, especificamente a metade, ou seja, 5 terminais. Este cenário possuí o mesmo nível de isolamento que o anterior e tem como objetivo mostrar como os modelos se comportam em ambientes de trabalho mais amenos.

O último cenário planejado foi denominado de ‘*Serialize’*, este nome também é devido ao nível de isolamento dele. Este nível de isolamento apesar de ser o *default* da ferramenta OTLP-Bench, não é comumente utilizado no SQL Server. Este uso é restrito devido a ser considerado um nível de isolamento ‘pessimista’. Este nível de isolamento gera muita concorrência entre as transações, uma vez que ele não permite que mais de uma transação utilize o mesmo dado ao mesmo tempo. Contudo, este isolamento é útil para verificar como cada base é afetada pelas demais. Como este nível é o que tende a segurar mais os recursos, conseguirá ser visto neste cenário como os modelos de comportam em um ambiente concorrido.

Para a execução dos testes, como mencionado anteriormente, utiliza-se a ferramenta *OTLP-Bench* [*OLTPBenchmark,* 2019]. A ferramenta foi configurada em uma máquina virtual com 1 vCPU e 2 Gb de memória RAM. Essa máquina foi criada no próprio portal do *Azure* e hospedada na mesma região dos bancos de dados. A carga escolhida para a execução dos testes foi a YCSB (*Yahoo! Cloud Serving Benchmark*), esta carga foi escolhida por ser uma carga simples estruturalmente e com métricas de resultado bem detalhadas, outro ponto favorável a essa escolha é que a carga é baseada em serviços Web, ou seja, aderente ao modelo *multi-tenants*, visto que este modelo, em sua quase totalidade, opera totalmente de forma Web. A utilização dessa carga pode ser vista também nos trabalhos de Sousa (2013) e Difallah (2014).

No item a seguir será explicado a criação dos modelos, para fim de entendimento e para uma possível replicação dos testes realizados neste trabalho. Contudo é necessário explicar um termo muito utilizado nas próximas páginas, o termo DTU (*Database Transaction Units*). O termo DTU é utilizado pelo portal *Azure* para simplificar a venda de poder computacional de banco de dados. As características físicas com CPU, memória e IO, associadas a cada medida de DTU são calibradas usando um parâmetro de comparação que simula a carga de trabalho do banco de dados real. (Microsoft ,2019). Apesar de não existir uma fórmula aberta do cálculo exato do DTU, essa unidade é amplamente utilizada e é a maneira com qual os serviços de SQL são vendidos no portal *Azure*.

4.3.1 Banco de dados por locatário – Modelo A

Para modelo A se utilizou um banco de dado com 10 DTU na camada de S0 (*Standard*). O armazenamento máximo foi definido como 10 Gb de dados. Como mencionado este modelo apenas possuí uma base de dados, sendo assim os outros modelos terão que capacidade igual a essa para cada locatário. O custo de cada DTU neste modelo, na data desse trabalho, é de R$ 5,57. Sendo o custo total dessa configuração de R$ 55,73 mensal. A imagem 9 mostra essa configuração no portal da ferramenta.

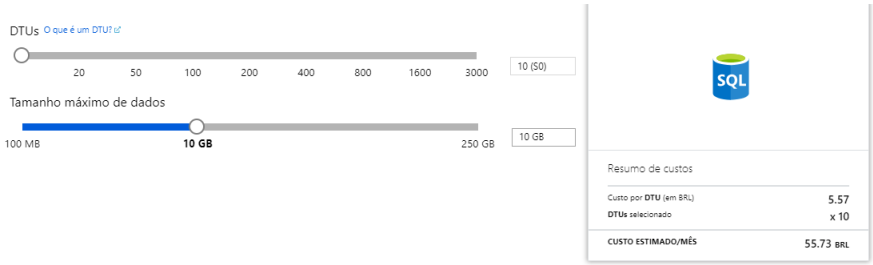


Figura 9. Configuração modelo A

Fonte: Elaborado pelo autor

4.3.2 Banco de dados compartilhado entre locatários – Modelo B

O modelo B consiste em uma única base de dados para todos os locatários. Como os testes foram idealizados com base em 5 locatários simultâneos e o Modelo A se utiliza de 10 DTU, portanto o modelo B foi configurado com 50 DTU, ou seja, como se cada cliente tivesse um poder computacional de 10 DTU. O preço de DTU desse modelo segue o mesmo do modelo anterior, resultando em um custo mensal de R$ 278,68. A Figura 10 mostra essa configuração.

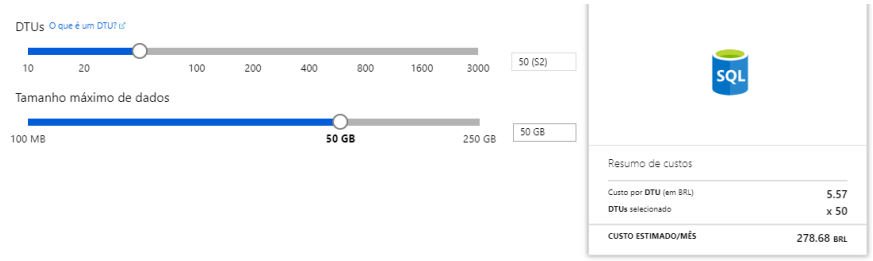


Figura 10 – Configuração modelo B

Fonte: Elaborado pelo autor

4.3.3 Pool de banco de dados compartilhado – Modelo C

O modelo C é baseado no *Azure Elastic Pool*, sendo assim foi criado um *pool* de 50 DTU e 50 Gb, respeitando a distribuição dos modelos anteriores. Contudo esse tipo de *pool* traz uma vantagem na sua criação. É possível configurar o limite que cada base de dados poderá utilizar de DTU. No modelo B não se tem isso, como a base de dados é

única, cada locatário pode utilizar o máximo da base, ou seja, 50 DTU. Neste modelo de configuração, é possível limitar para que um locatário tenha mais DTU que outro.

Para fins de testes restringimos a 10 DTU por base, para que assim consiga-se verificar exatamente se essa limitação acontece. Contudo, também se realiza testes sem esse limite, assim podendo gerar comparações entre o mesmo modelo. O custo do DTU no *pool* foi de R$ 8,35 por DTU. Apresentando um acréscimo de 49 % do valor do modelo de DTU convencional. A Figura 11 mostra a configuração do *pool* e a Figura 12 mostra a configuração dos limites entre as bases.

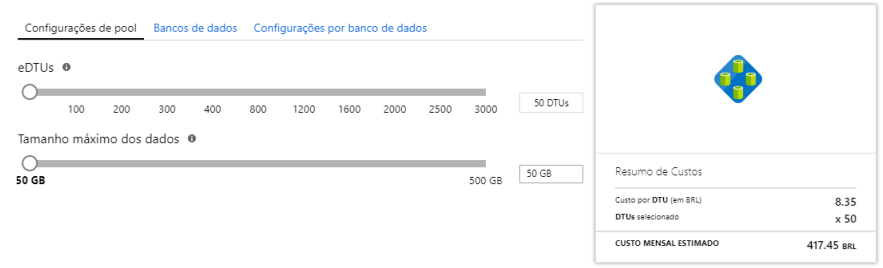


Figura 11 – Configuração modelo C – parte 1

Fonte: Elaborado pelo autor

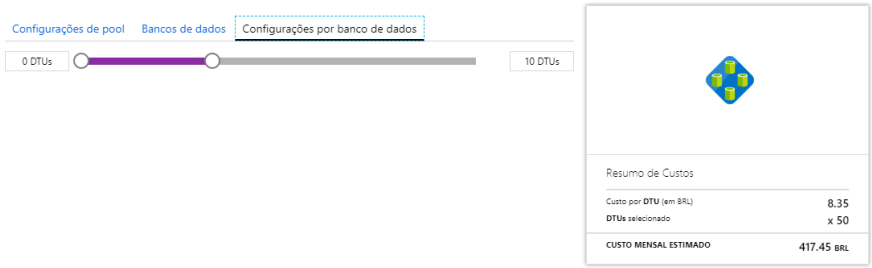


Figura 12 -Configuração modelo C – parte 2

Fonte: Elaborado pelo autor

4.3.4 Cenários

As Figuras 13, 14 e 15, demostram os cenários *ReadCommitted* 10 VM, *ReadCommitted* 5 VM e *Serialize*, respectivamente. Como é visto na Figura 13, o primeiro cenário possuí 10 terminais ou também chamados de VMs. Esses terminais simulam usuários de um cliente, portanto esse cenário simula 5 clientes com 10 usuários trabalhando ao mesmo tempo. O principal objetivo dessa simulação é tentar alcançar a maior utilização possível dos recursos com um nível de isolamento considerado mais otimista.

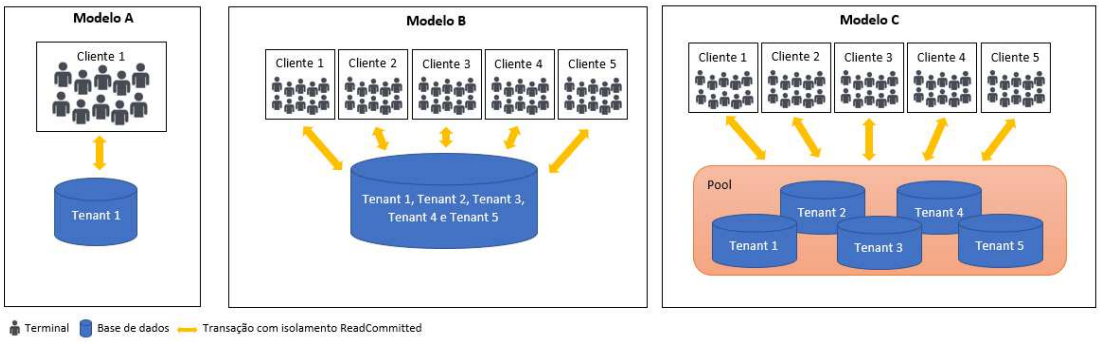


Figura 13. Cenário ReadCommitted 10 vm

Fonte: Elaborado pelo autor

A Figura 14 demonstra o segundo cenário, o denominado *ReadCommitted* 5 VM. Neste cenário os modelos são submetidos a uma carga menor, com apenas 5 terminais. A proposta do cenário é avaliar o comportamento dos modelos com uma carga mais leve e, portanto, verificar os resultados em um ambiente mais estável.

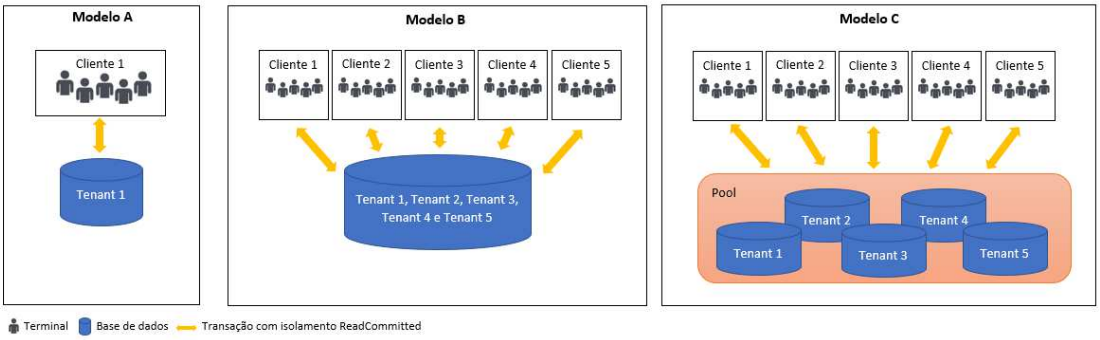


Figura 14. Cenário *ReadCommitted* 5 vm

Fonte: Elaborado pelo autor

O último cenário é mostrado na Figura 15. O cenário denominado *Serialize,* apresenta uma carga alta de 10 terminais e com o nível de isolamento mais pessimista. Por isso, pode ser considerado o cenário mais complexo e crítico. Com este tipo de cenário será possível verificar os modelos em ambiente que requerem esse nível de implementação.

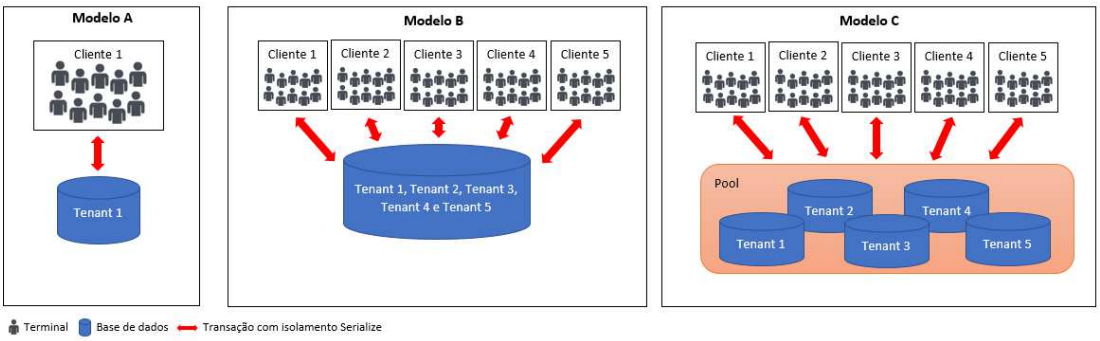


Figura 15. Cenário *Serialize*

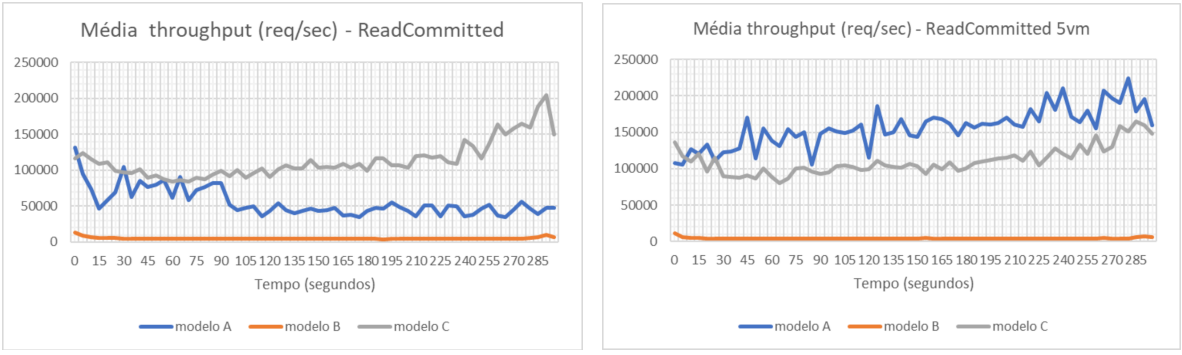
Fonte: Elaborado pelo autor

Após as definições dos modelos e cenários é possível verificar os resultados dos testes realizados. Como cada cenário tem um objetivo claro, será possível entender como os modelos se comportaram e em quais cenários cada modelo se saiu melhor. Na seção 5 são mostrados esses detalhes.

5 Apresentação e Análise de Resultados

5.1 Desempenho

Para medição do desempenho dos modelos se usará como base dados gerados pela ferramenta de testes. Os dados utilizados serão a latência média de todo o processo bem como a latência de cada tipo de operação realizada pela carga escolhida. Usa-se também o *throughput* médio de requisições por segundo. Esses dados dizem qual tipo de modelo conseguiu processar mais requisições com a menor latência, definindo assim a mais performática.



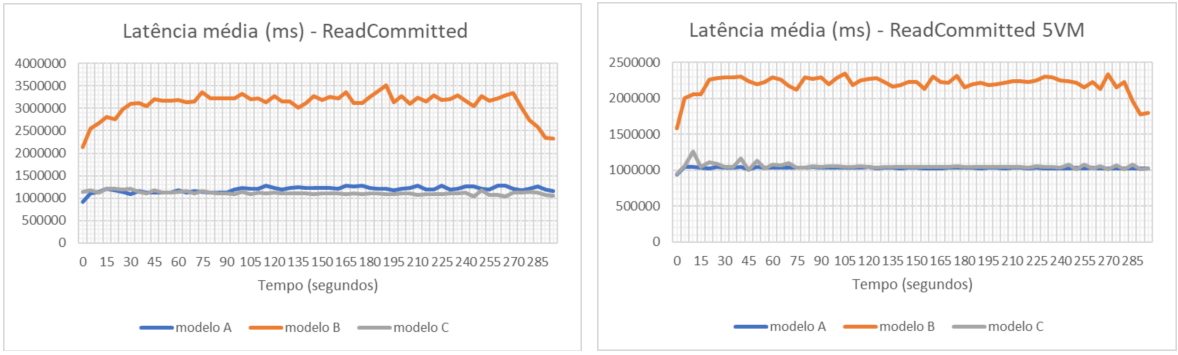
(a) (b)

Figura 16. Média throughput - ReadCommitted

Fonte: Elaborado pelo autor

Na Figura 16(a) é visto o gráfico de média de *throughput,* dos modelos nos cenários com isolamento *Readcommitted* com 10 e na Figura 16(b) com 5 terminais. Primeiramente é possível notar que o modelo B, em ambos os cenários, se mostra o menos eficiente, com uma média de requisições muito abaixo dos demais. Outro ponto que deve ser notado é que o modelo A no primeiro cenário tem uma média inicial alta, mas com o passar do tempo a média baixa até uma alcançar quase uma constante.

O mesmo modelo A no segundo cenário já tem um comportamento um pouco diferente, ao invés da média baixar com o passar do tempo, ela aumenta. Esse comportamento nos induz a afirmar que o modelo A, no primeiro cenário, deve ter alcançado algum limite de recurso e que quando a carga foi menor, no segundo cenário, o recurso respondeu melhor, gerando assim um desempenho melhor no cenário com o menor número de processos. Já, o modelo C em ambos os cenários de mostrou muito parecido, apenas com uma leve diferença no primeiro cenário.



(a) (b)

Figura 17. Latência média - *ReadCommitted*

Fonte: Elaborado pelo autor

A Figura 17 mostra os gráficos referentes a latência média em microssegundos dos modelos. Os cenários aqui apresentados também são referentes ao *ReadCommitted* com 10 terminais na Figura 17(a), e 5 terminais na Figura 17(b). Nestes gráficos os modelos A e C se mostram muitos parecidos, com uma latência do modelo C um pouco acima do modelo A no primeiro cenário. Já, no segundo cenário são praticamente iguais, salvo algumas leves diferenças no início e fim dos testes. O modelo B por usa vez mostra aqui um baixo desempenho, tendo uma sua latência muito acima dos demais modelos.

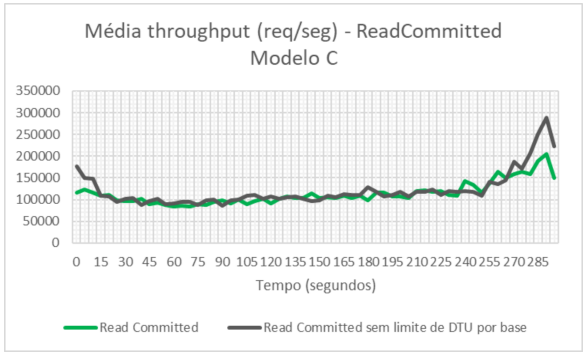
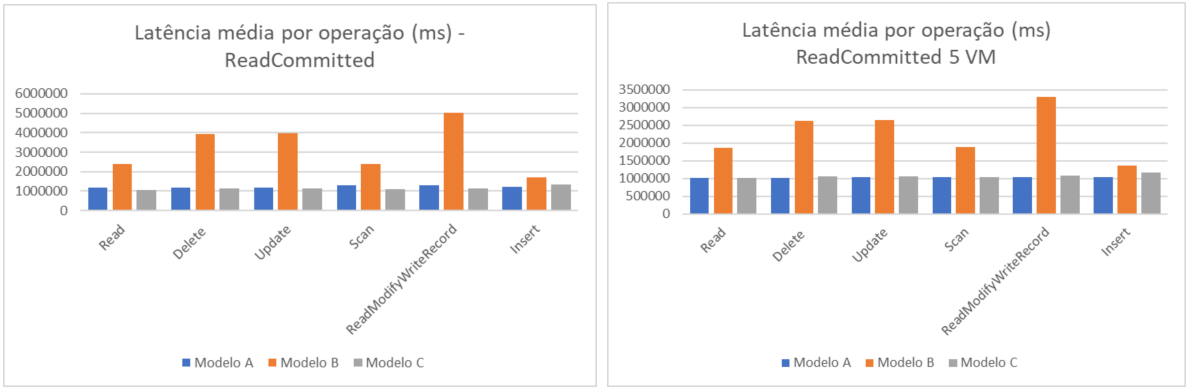


Figura 18. Média throughput- modelo C

Fonte: Elaborado pelo autor

A Figura 18, mostra o comportamento do modelo C no cenário *ReadCommitted* com 10 terminais. Conforme explicado na seção 4.3.3, é possível configurar que o *pool* restrinja o uso de DTU por base ou não. Sendo assim cria-se um cenário alternativo, derivado do cenário *ReadCommitted*. Neste cenário foi retirado o limite de DTU por base, podendo assim cada base utilizar o que estiver disponível. Conforme mostra a Figura 18 há um comportamento bem parecido em ambos os casos, apenas com uma breve diferença inicial e final. Essa diferença se deve ao fato que a carga foi disparada com 5 locatários e eles não tiveram exatamente o momento de início. Isso devido a ferramenta de teste ter que ser iniciada uma vez para cada locatário, o que gera um atraso nos inícios e fins do processo, uma vez que o sistema operacional da máquina hospedeira, não consegue iniciar os processos exatamente no mesmo momento. Na

próxima seção explica-se o de uso de recurso e abordando assim um pouco mais essa questão.



(a) (b)

Figura 19. Latência média por operação - *ReadCommitted*

Fonte: Elaborado pelo autor

Na Figura 19 se vê a latência média separada por operação. Os cenários apresentados são o *ReadCommitted* com 10 terminais mostrado na Figura 19(a) e 5 terminais na Figura 19(b). É possível notar nessa comparação que todos os modelos têm uma latência menor no segundo cenário e seguem os mesmos níveis de diferença entre os modelos.

O ponto mais importante que é possível ver aqui, é em relação a operação *insert.* Nota-se claramente que em ambos os cenários a operação de inserção é a que tem a menor diferença entre os modelos. Até mesmo o modelo B que até agora vem se mostrando com o desempenho menor, tem um resultado muito próximo dos demais modelos. Isso faz com que se possa suspeitar que o desempenho do modelo B é afetado pelas consultas na base. Essa suspeita também pode ser embasada no fato que todos os clientes estão utilizando a mesma tabela do banco de dados.

Sendo assim além de uma concorrência maior, existe uma tabela maior que as dos demais modelos. Assim, pode-se afirmar que o desempenho do modelo B pode ser melhorado com uma otimização em sua estrutura. Uma hipótese de melhoria seria a criação de um índice de cobertura com a chave primaria sendo identificar do *tenant,* otimizando assim as consultas referentes a cada locatário. Como o objetivo do trabalho não é especificamente o melhoramento do modelo B e nem tampouco favorecer um ou outro modelo aqui apresentado, considerando o teste dessa hipótese num trabalho futuro.

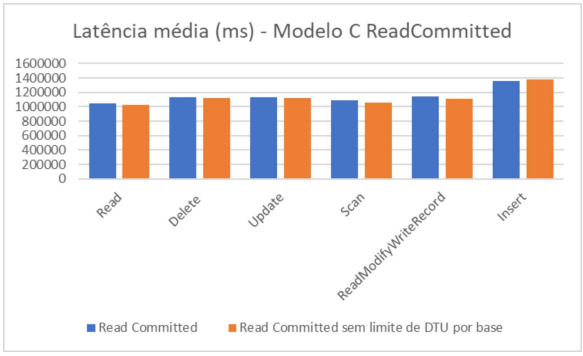


Figura 20. Latência média por operação – Modelo C

Fonte: Elaborado pelo autor

O gráfico presente na Figura 20 é comparativo do modelo C, com a limitação de DTU por base e sem a limitação. Conforme foi observado antes, nos dois casos existe uma média muito próxima, sem a necessidade muita análise.

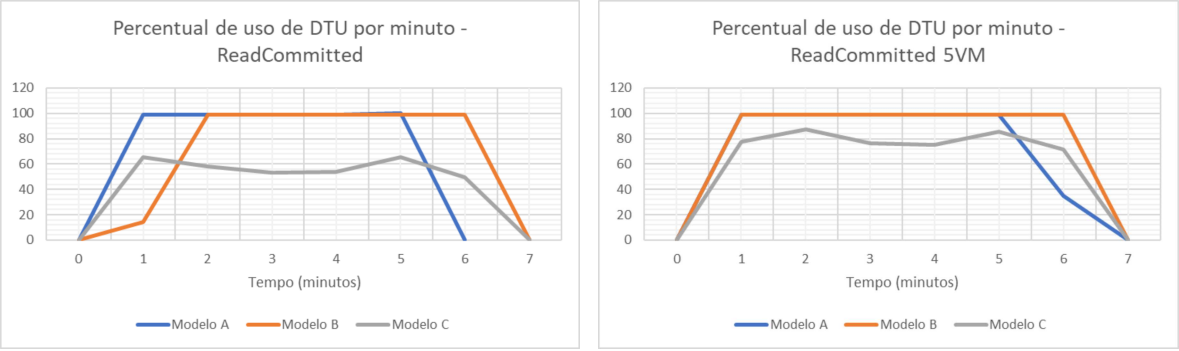
Como visto aqui, o modelo B mostrou o pior desempenho em todos os gráficos. Por sua, vez o modelo A apresentou um comportamento melhor com cargas menores, diferente do modelo C que mostrou comportamento parecido com duas cargas diferentes. Concluindo assim que o modelo C foi igual e até melhor que o modelo A.

5.2 Uso de recurso

Conforme visto na sessão 2.1., o conceito de *cloud*, está ligado diretamente entre os recursos de hardware disponíveis e a utilização desses recursos, sendo um modelo ideal que a disponibilidade seja muito próxima da utilização, gerando assim menor desperdício de poder computacional.

Para a verificação dos usos de recurso de cada modelo usa-se como base os dados gerados pelo portal do *Azure*, dados esses referentes a DTUs consumidas, e dados gravados na próxima base de dados. Esses dados foram extraídos da tabela *sys.dm_db_resource_stats*. Essa tabela armazena informações estatísticas de uso dos recursos, sendo assim possível levantar o uso de CPU e IO em cada momento de execução dos testes.

Para dados originários do portal *Azure* existe como granularidade mínima o minuto, sendo assim as coletas sempre ocorrem no intervalo mínimo de 1 minuto. Assim nos gráficos a seguir, os DTUs são em relação ao tempo em minutos, especificamente sete minutos, sendo cinco minutos de execução, mais um antes do início e um minuto no fim. Já, no caso das métricas colhidas a partir *dm_db_resource_stats*, o intervalo mínimo é de 15 segundos entre as coletas. Para essas métricas utiliza-se 300 segundos referentes ao período de execução e mais 45 segundos, divido entre o início e o fim da execução.



(a) (b)

Figura 21. Percentual médio de uso de DTU - *ReadCommitted*

Fonte: Elaborado pelo autor

Na Figura 21 apresenta-se o percentual de DTU no período, esse percentual se trata da média máximo em cada minuto. A Figura 21(a) representa o cenário *ReadCommitted* com 10 terminais e a Figura 21(b) com 5 terminais. É possível notar que os modelos A e B nos dois cenários estão no limite quase durante todo o processo, já o modelo C tem um comportamento diferente. No primeiro cenário se tem o modelo ficando em um faixa de 60% de uso de DTU, isso nos sugere que neste caso a performance foi limitada por algum outro fator, não sendo necessariamente um limite de poder computacional. Já, no segundo cenário se tem o modelo C alcançando 80% durante quase todo o processo.

Na Figura 22 é possível analisar o comportamento do modelo C no ambiente com limitação e sem limitação de DTU. Conforme já visto, a média na configuração padrão ficou em de 60%, com valores menores no início e no fim do processo. Já quando tirado a limitação de DTUs a média sobe mais nestes momentos, percebesse ali picos de 80%, tanto no início como no fim do processo. Este fenômeno pode ser explicado, pois, conforme já mencionado, todas as cargas de todas as bases não começam e terminam ao mesmo, sendo assim observasse, um momento que nem todas as bases estão necessitando de recursos. Esse fato, juntamente com a ausência do limite de DTU entre bases, faz que a média no início e fim aumente. Este comportamento pode ser visto mais claramente na Figura 23.

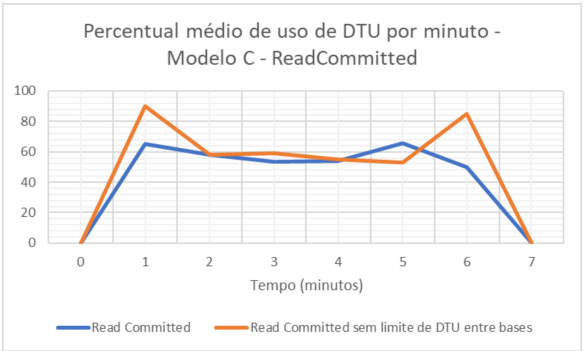


Figura 22. Percentual médio de uso de DTU – Modelo C

Fonte: Elaborado pelo autor

A Figura 23 mostra a distribuição do DTUs entre as bases. Essa distribuição é referente ao uso máximo de DTU em cada minuto de cada base. Por isso, conforme pode ser visto há um ponto onde os 100% são ultrapassados. Essa discrepância é gerada devido as bases terem máximas em períodos diferentes dentro do mesmo minuto, gerando assim uma soma superior ao 100%. Contudo, o mais importante de ser visto aqui é que no início e no fim do processo as bases utilizam mais que 10 DTUs, como pode visto na base ysbc1, onde no minuto 1 ela ultrapassa os 25 DTUs. Isso mostra que a configuração de limitação de DTUs realmente funciona conforme prometido pelo fornecedor.

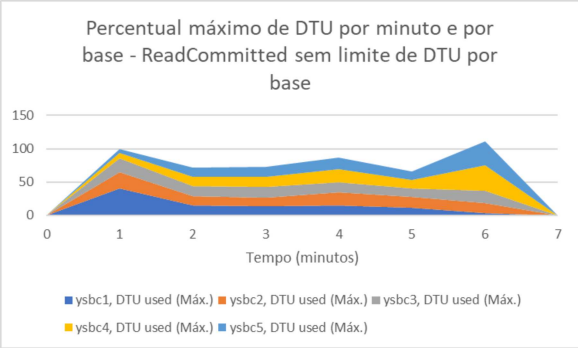
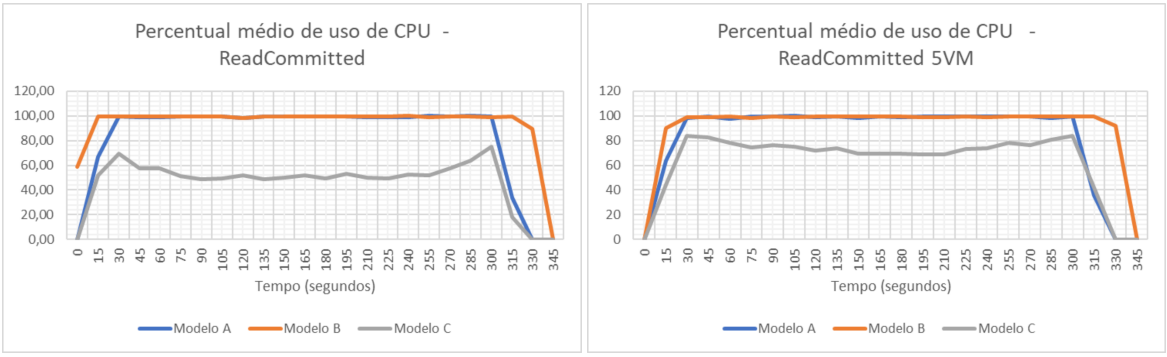


Figura 23. Percentual máximo de DTU por base

Fonte: Elaborado pelo autor

A Figura 24, mostra o percentual médio de uso de CPU no cenário *ReadCommitted*, sendo a Figura 24(a) com 10 terminais e a Figura 24(b) com 5 terminais. Aqui se pode ver que todos os modelos seguem, quase, o mesmo padrão de uso de CPU e DTU, apresentado anteriormente. Se vê aqui que o modelo A e B utilizam 100% de CPU, enquanto o modelo C fica na zona de 60% no primeiro cenário e 80% no segundo cenário.

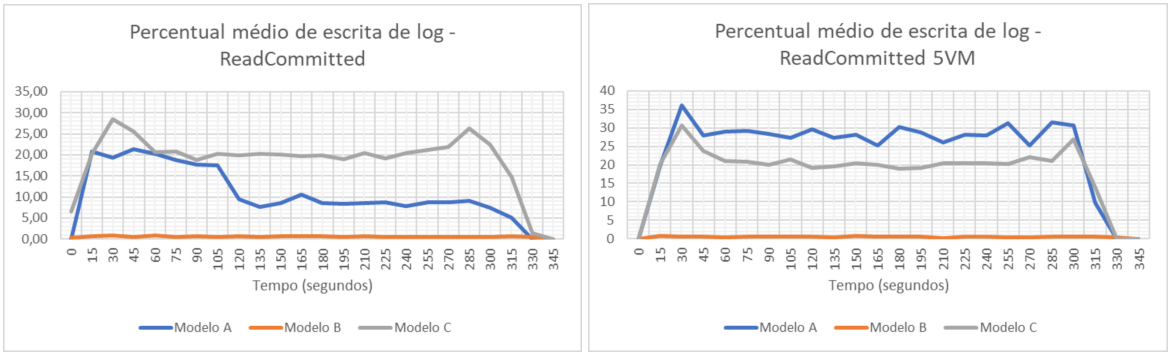


(a) (b)

Figura 24. Percentual médio de uso de CPU - *ReadCommitted*

Fonte: Elaborado pelo autor

A Figura 25 mostra o percentual médio de escrita de *log*, com o cenário de 10 terminais na Figura 25(a) e com 5 terminais na Figura 25(b). Nesta Figura se vê como o modelo C se mostra muito parecido nos 2 cenários com uma faixa variando de 20% a 25%. Por usa vez, o modelo A mostra, no primeiro cenário, uma tendência de queda durante o processo, iniciando com 20% e depois alcançando uma média de 10%. Já, no segundo cenário o modelo A aparece com uma tendência mais alta, ficando na média dos 30%. O modelo C por sua vez, se mostra o menor em ambos os cenários, ficando com médias abaixo de 5%. Esse comportamento, provavelmente é refletido da sua baixa eficiência de *throughput,* apresentado anteriormente.



(a) (b)

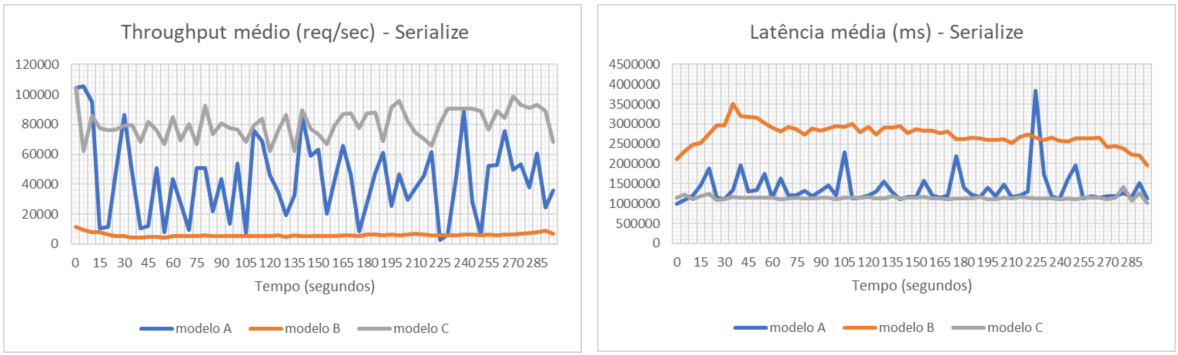
Figura 25. Percentual médio de escrita de log - *ReadCommitted*

Fonte: Elaborado pelo autor

Com os dados aqui apresentados, é possível concluir que os modelos A e B se mostraram parecidos no uso de DTU, apenas mostrando diferenças quando analisado a escrita de *logs*, mas o comportamento se justifica pelo baixo desempenho do modelo B. Por sua vez, o modelo C se mostrou melhor em todos os gráficos, mostrando que gerencia melhor a utilização dos recursos. Sendo possível afirmar que o modelo C foi o melhor na avaliação de uso de recursos.

5.3 Isolamento

Conforme explicado anteriormente, foi criado o cenário chamado *Serialize.* Este cenário utiliza o nível de isolamento padrão da ferramenta de teste. Este tipo de isolamento tende a ser bem menos performático, pois as requisições são enfileiradas para garantir total integridade entre as operações. Sendo assim um cenário perfeito para verificar o nível de isolamento de cada modelo, isolamento esse, que é essencial para banco de dados, conforme visto na sessão 2.4.2.



(a) (b)

Figura 26. *Throughput* e latência - *Serialize*

Fonte: Elaborado pelo autor

Mostrasse na Figura 26(a) o *throughput* e na Figura 26(b) a latência média, dos cenários, nos 3 modelos. É possível notar que o modelo B se mostra o pior, seguindo assim a mesma tendência dos outros cenários. Um ponto bem interesse que se deve ver aqui, é em relação a instabilidade do modelo A. Em ambos os cenários, o modelo mostra um comportamento atípico, onde as métricas oscilam de uma forma muito brusca. Ficando aqui um outro ponto de estudo futuro, afim de determinar porque há essa variação tão grande neste modelo. Analisando o modelo C se pode ver uma variação bem menor dos valores e em ambas métricas ele se mostra o melhor, sendo o menos afetado por este nível de isolamento até agora.

Analisando a Figura 27 observasse a latência média por operação. Aqui existe um comportamento bem parecido com os outros cenários apresentados até agora, contudo, com uma diferença na operação de inserção. Nos cenários anteriores é visto uma operação de inserção que era menos alterada entre os modelos, mas agora, neste cenário é possível notar que o modelo B também é afetado. Isso deve-se ao nível de isolamento *serialize,* literalmente serializar as transações que afetam dados. Como no modelo B uma mesma tabela é utilizada para todos os clientes, as inserções de todos os clientes concorrem um com os outros, gerando assim uma fila maior que os dos outros modelos.

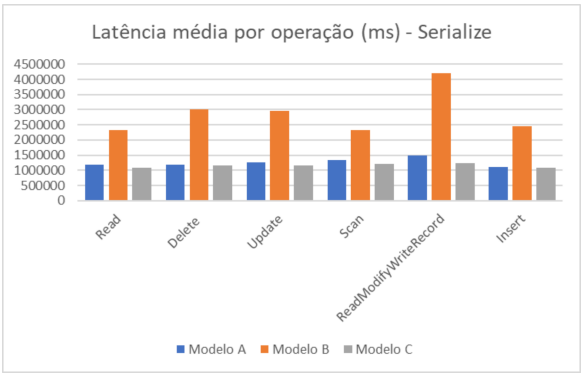
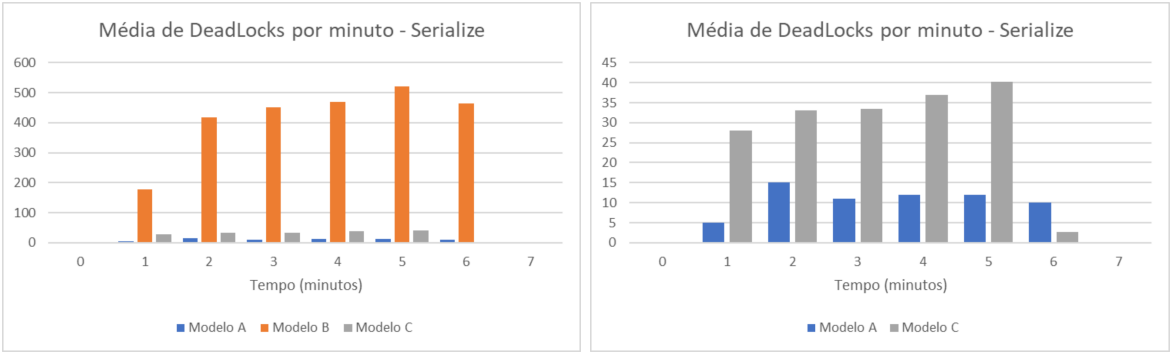


Figura 27. Latência média por operação - *Serialize*

Fonte: Elaborado pelo autor

Quando se fala em performasse e isolamento em banco de dados é fundamental mencionar a questão dos *DeadLocks.* Em uma tradução livre, o termo seria traduzido como 'morte por bloqueio'. É exatamente isso que acontece, quando uma transação tranca outra e essa outra acabada trancando a primeira, as duas transações entram em uma situação onde nenhuma dela conseguirá sair. Por isso, o próprio SGBD acaba finalizando uma transação, gerando o chamado *deadlock.* Por esse motivo, a Figura 28 mostram um detalhamento dos *deadlocks* ocorridos nos cenários *serialize.* Como os outros cenários possuem um nível de isolamento mais otimista, não existe a ocorrência desse tipo de problema.



(a) (b)

Figura 28. Média de *deadlocks*

Fonte: Elaborado pelo autor

A Figura 28(a) mostra a média de *deadlocks* que ocorreram durante o processo em cada modelo. Notasse ali, que o modelo B é o mais afetado disparadamente. Isso ocorre por motivos já comentados desse nível de isolamento. Na Figura 28(b), é apresentado um detalhe comparativo apenas entre o modelo A e C, sendo possível analisar de uma melhor forma quais dos dois modelos se saiu melhor. É possível ver que o modelo A se mostra bem melhor que o modelo C em quantidade de *deadlocks,* com menos da metade da quantidade do modelo C. Com esse dado, é possível afirmar que o modelo C, é melhor que o modelo B, porém pior que o modelo A. Sendo o modelo A o que se mostrou menos propício a ocorrência de *deadlocks* neste cenário.

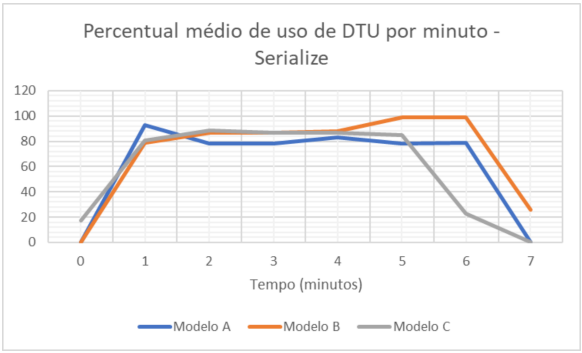
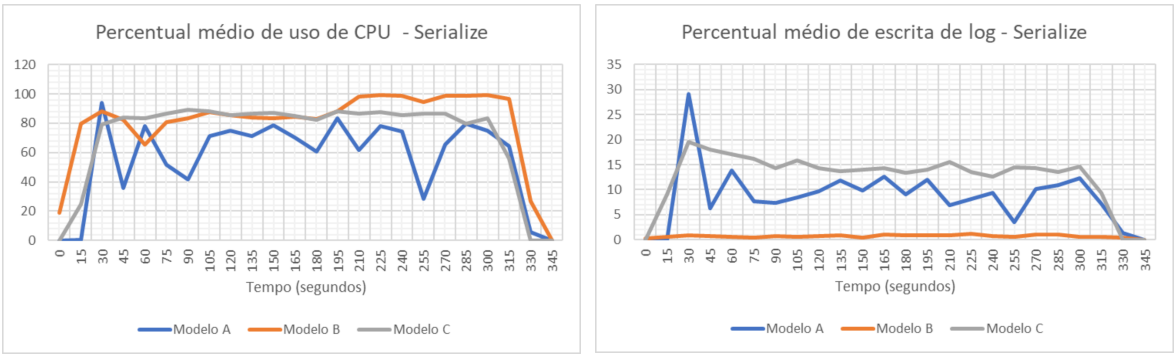


Figura 29. Percentual médio de uso de DTU – *Serialize*

Fonte: Elaborado pelo autor

Em relação ao consumo de recurso neste cenário, é possível que na Figura 29, os 3 modelos utilizaram níveis bem parecidos de DTU durante os testes, ou seja, neste cenário os modelos se comportaram um pouco diferente dos cenários anteriores. Sendo que os 3 modelos ficaram, na maior parte do tempo, na faixa de 80-100% de uso de DTU.



(a) (b)

Figura 30. Percentual médio de uso de CPU- *Serialize*

Fonte: Elaborado pelo autor

Na Figura 30(a) é apresentado o uso de CPU e na Figura 30(b) a escrita de *logs* dos modelos. É possível notar que a utilização de CPU do modelo A apresenta uma variação alta, bem parecida com a variação vista no *throughput* e latência desse modelo. Já, o modelo B e C tem um uso de CPU bem parecidos. Quando se analisa o percentual médio de escrita em *log* dos modelos é possível notar que o modelo B possui uma média muito inferior aos demais modelos. A principal hipótese desse comportamento se deve ao fato de o modelo ter apresentando o pior resultado em desempenho, gerando uma quantidade menor de IO de escrita, consequentemente, o menor percentual. Já, o modelo A também apresenta variações no decorrer no processo, bem parecido com o que ocorreu no uso de CPU.

É possível concluir aqui, que o modelo que mais garante isolamento entre os *tenants* é o modelo A. O modelo C mesmo sendo muito superior ao modelo B, não consegue garantir o mesmo nível de isolamento do modelo A.

5.4 Custo

O custo é uma métrica essencial para qualquer avaliação e conforme visto na sessão 2.2, o modelo de arquitetura SaaS tem como o objetivo de negócio explorar novos mercados. Mercados que as arquiteturas convencionais não alcançam devido aos custos mais elevados. Por esse motivo, um modelo com custo menor tende a ser o mais indicado para a utilização no modelo SaaS.

Conforme já apresentado anteriormente, todas as bases foram criadas no portal *Azure,* utilizando a mesma região geográfica. Para o modelo A se utilizou uma base única com capacidade 10 DTU nível S0. No modelo B se utilizou uma base única

particionada em 5 locatários, com 50 DTUs nível S0. Para o modelo C, foi utilizado um *pool* para 5 bases separadas com 50 DTUs nível S1. A diferença de nível dos modelos se deve ao fato que a configuração inicial do *pool* deve ser de no mínimo S1 para bases com mais de 5 DTUs. Como era o objetivo do trabalho apresentar um cenário mínimo e realista, se optou pelas configurações mínimas de cada modelo, não favorecendo assim nenhum dos 3 modelos. O quadro 2 Apresenta o resumo dos custos de DTU para cada modelo.

Quadro 2. Custo de DTU por modelo.

Modelo	Custo por DTU (R$/Mês)
Modelo A	5,57
Modelo B	5,57
Modelo C	8,35

Fonte: Elaborado pelo autor

Importante ressaltar sobre os custos, é que apenas foram levantados os custos diretos. Custos indiretos como backup, tempo implantação e entre outros, não foram aqui apresentados, por não fazerem parte do custo essencial e que poderiam variar conforme a aplicação, que assim, impossibilitam uma comparação justa entre os modelos.

5.5 Resumo dos resultados

Conforme apresentado nos resultados dos itens anteriores, é possível avaliar o comportamento da ferramenta *Azure SQL Elastic Pool* em um ambiente multi-locários. Para fazer isso comparou-se o modelo C com o modelo B e A. O modelo A é considerado a base de referência, uma vez que este é o modelo clássico de banco de dados. O modelo B por sua vez, é uma abordagem concorrente do modelo C, pois ambos tentam de alguma maneira compartilhar recursos com mais clientes.

Sendo assim, se considera positivo o modelo C quando seus níveis se comparam com A e são melhores que B. Primeiramente em relação a performance, é visto um comportamento bom com níveis de *throughput* próximos e até mais altos que o modelo A e com latências praticamente iguais à A. No uso de recursos foi visto que o aproveitamento de DTU foi melhor que o modelo A, mostrando níveis menores de uso de CPU e escrita de logs. Para os testes de isolamento, os resultados não foram positivos. É possível ver, principalmente nos números de *deadlocks*, que o modelo C apesar de melhor que o modelo B, é pior que o modelo A, apresentando uma possível interferência entre as bases. Em relação ao custo, o modelo C se mostra mais caro, principalmente, no que tange à utilização do *pool*, que exigi níveis da camada de preços diferentes dos outros modelos. Assim é considerado que a ferramenta tem um custo maior, uma vez que não consegue cumprir o mesmo rendimento de A com o mesmo preço.

O Quadro 3 apresenta um resumo comparativo dos resultados obtidos, comparando os modelos B, C e A.

Quadro 3. Comparativos dos modelos.

Métricas	Modelo B	Modelo C
Desempenho	<A ∧ <C	≅ A ∧ >B
Uso de recursos	≅ A	>A ∧ >B
Isolamento	<<A	< A ∧ >B
Custos	<A ∧ <C	>A ∧ >B

Fonte: Elaborado pelo autor

Após a análise dos resultados e os estudos apresentados durante o desenvolvimento da pesquisa é possível responder à questão base da proposta do trabalho. A ferramenta *Azure* SQL *Database Elastic Pool,* consegue entregar o melhor custo benefício em bancos de dados em uma aplicação *multi-tenancy*? Pelas evidências apresentadas durante a pesquisa, pode ser dito que sim. Apesar do modelo ser o mais caro entre os aqui estudados, seus resultados são satisfatórios e justificam o investimento. Mesmo o modelo B sendo o mais barato, o estudo mostrou que o isolamento neste modelo é o pior entre os modelos estudados. Portanto, conforme a ISO 17789:2014 (ISO/IEC 17789:2014) que ressalta a importância do isolamento, o modelo B se mostra ineficiente para a utilização em aplicações *multi-tenancy.* Portanto, é possível afirmar que o modelo C se mostrou, sim, uma opção válida para a utilização em aplicações com esse tipo de arquitetura.

6 Considerações finais

Através do desenvolvimento do trabalho foi possível avaliar e quantificar métricas sobre a ferramenta analisada neste estudo de caso. Foram vistos os aspectos positivos e negativos de sua utilização. Levantou-se aqui uma comparação equilibrada entre as demais abordagens, a fim de possibilitar que trabalhos futuros utilizem os dados aqui gerados como referência, ajudando que novos experimentos sejam realizados e validados.

Em relação as áreas estudo de trabalho, apresentou-se os modelos acadêmicos e aplicou-se em uma ferramenta comercial, tornando assim o trabalho útil para ambos os mundos. No mundo acadêmico como experimento de validação ou de exemplo de uso e na indústria, como uma referência comparativa, para auxílio na escolha de uma opção de arquitetura e ferramenta para banco de dados em aplicações multi-locatários.

Em relação aos objetivos apresentados no início do trabalho, todos eles foram atingidos. Foram analisados os conceitos referentes a computação em nuvem, bem como os aspectos de SaaS. Foram analisados os fundamentos sobre os bancos de dados para aplicações SaaS e apresentado modelos de utilização. Após a análise do referencial teórico, ocorreu a criação dos modelos e cenários de testes previstos. Finalizando os objetivos, com a execução dos testes e o demonstrativo dos resultados encontrados em cada métrica.

Como o tema abordado no trabalho trata-se de um tema emergente, observou-se dificuldades nos testes, devido a ferramenta *OLTP-Bench* não estar preparada para

testes em ambientes compartilhados e principalmente por não possuir suporta a bases de dados com tabelas compartilhadas. Esse problema obrigou ajustes na ferramenta para a implementação do suporte a essa arquitetura, gerando assim tarefas adicionais no decorrer do trabalho. Justamente devido a esses tipos de problemas, que o trabalho se limitou apenas a testar os modelos principais de banco de dados em aplicações multi-locatários, limitando-se também, em apenas uma ferramenta de mercado.

Para trabalhos futuros, conforme sugerido ao longo do texto, sugere-se o estudo sobre dois fenômenos observados no desenvolvimento da pesquisa. O primeiro é sobre o modelo B, que foi mostrado na Figura 19(a). O modelo neste cenário mostrou que a operação que menos teve impacto na sua performance foi a operação de inserção. Gerando assim a hipótese que se as outras operações fossem otimizadas, com o uso de um índice, por exemplo, consequentemente a performance geral do modelo deverá melhorar. Possibilitando, que o modelo se torne mais eficiente e, portanto, passível de uma nova comparação com os outros modelos. Para a melhoria desse modelo, sugere-se a utilização de técnicas de *tunning* na base dados, afim de otimizar a operação de consulta.

O segundo fenômeno é em relação ao modelo A no cenário *Serialize*. Conforme visto na Figura 19(a) e 19(b), o modelo mostrou uma grande variação de *throughput e* latência durante o processo de teste no cenário *Serialize.* Para a compreensão desse comportamento é necessário a criação de um teste mais longo e específico nesse modelo, para que seja possível identificar o que leva a essa variação tão brusca e qual é o impacto disso em uma aplicação real.

Além das sugestões mencionadas acima, também se propõe como trabalho futuro a proposta de realizar, os testes aqui apresentados, em outros cenários e com outras ferramentas comerciais, como por exemplo a utilização de *Docker* (*Docker*, 2019) e *Kubernetes* (*Kubernetes, 2019*) ou até mesmo testes com ferramentas *NoSQL*, como a *Dynamodb* (*Dynamodb, 2019*). Gerando assim mais evidências e dados comparativos, melhorando ainda mais a possibilidade de escolha de uma ferramenta ideal para o uso de banco de dados em aplicações SaaS.

Referências

Amazon (2019), “Portal Amazon AWS”, 2019, disponível em <https://aws.amazon.com/pt/>, Acesso em 14 de junho de 2019.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I. e Zaharia, M. (2009), “Above the clouds: A berkeley view of cloud computing”. Technical report, EECS Department, University of California, Berkeley.

Aymerich, F. M. (2008), “An aproach to a cloud computing network”. In: Proceedings of the 1st International Conference on the Applications of Digital Information and Web Technologies (ICADIWT). Washington, DC (US): IEEE Computer Society, p. 113- 118.

Azevedo, D., Machado, L. e Silva L. V. (2011), “Métodos e Procedimentos de Pesquisa - Do projeto ao relatório final”. Editora Unisinos, São Leopoldo.

Azure (2019), “Portal Azure”, 2019, disponível em <https://Azure.microsoft.com/pt-br/>, Acesso em 14 de junho de 2019.

Bezemer, C. e Zaidman A., (2010), “Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare?” in ACM International Conference Proceeding Series. 88-92.

Cloud Security Alliance (CSA), (2017), Security Guidance for Critical Areas of Focus in Cloud Computing - Version 4.0. Cloud Security Alliance, 2017.

Chong F. e Carraro G. (2006), “Architecture Strategies for Catching the Long Tail”. Microsoft Corporation. 2006. Disponível em <http://cistratech.com/whitepapers/MS_longtailsaas.pdf>. Acesso em: 17 de abril 2019.

Difallah, D. E., Pavlo A., Curino C. e Cudre-Mauroux, P. (2014), “OLTP-Bench: An extensible testbed for benchmarking relational databases” In VLDB.

Docker (2019), “Enterprise Container Platform for High-Velocity Innovation”, 2019, disponível em < https://www.docker.com/>, Acesso em 13 de julho de 2019.

Dynamodb (2019), “Amazon DynamoDB”, 2019, disponível em <https://aws.amazon.com/pt/dynamodb/>, Acesso em 13 de julho de 2019.

Elmasri, R. e Navathe, S. B. (2005), “Sistemas de banco de dados”. 6ª ed. São Paulo; Editora Pearson.

International Data Corporation - IDC, (2019), “Worldwide Public Cloud Services Spending Forecast to Reach $210 Billion This Year, According to IDC”, 2019, disponível em < https://www.idc.com/getdoc.jsp?containerId=prUS44891519&utm_medium=rss_feed&utm_source=Alert&utm_campaign=rss_syndication>, Acesso em: 24 de abril de 2019.

ISO/IEC 17788:2014: Information technology – Cloud computing – Overview and vocabulary. Geneva, Switzerland: International Organization for Standardization.

ISO/IEC 17789:2014: Information technology — Cloud computing — Reference architecture. Geneva, Switzerland: International Organization for Standardization.

Kubernetes, (2019), “Orquestração de contêiner pronto para produção”, 2019, disponível em <https://kubernetes.io/pt/>, Acesso em 13 de julho de 2019.

Kim, W. (2009), “Cloud computing: today and tomorrow”. Journal of Object Technology. Zurich: ETH Zurich, v. 8, n. 1, p. 65-72, Jan-Fev/2009

Matthew, O. (2016). “Establishing a Standard Scientific Guideline for The Evaluation And Adoption Of Multi-Tenant Database.”, 219 f. Tese de Doutorado, Universidade de Wolverhampton.

Microsoft (2018), “Os pools elásticos ajudam você a gerenciar e dimensionar vários bancos de dados SQL do Azure”, 2018, disponível em <

https://docs.microsoft.com/pt-br/Azure/sql-database/sql-database-elastic-pool>, Acesso em 11 de dezembro de 2018.

Microsoft (2019), “Camadas de serviço no modelo de compra baseado em DTU”, 2019, disponível em <https://docs.microsoft.com/pt-br/Azure/sql-database/sql-database-service-tiers-dtu>, Acesso em 13 de junho de 2019.

Microsoft SQL Database Elastic Pool (2019), “Os pools elásticos ajudam você a gerenciar e dimensionar vários bancos de dados SQL do Azure”, 2019, disponível em <https://docs.microsoft.com/pt-br/azure/sql-database/sql-database-elastic-pool >, Acesso em 16 de junho de 2019.

Moreira, L. O., Sousa, F. R. C. e Machado, J. C. (2012), “Analisando o desempenho de banco de dados multi-inquilino em nuvem”., In SBBD ’12, p 161–168.

Molka, K. (2018). “Performance and Cost Optimization of Multi-Tenant In-Memory Database Clusters”, 195 f. Tese de Doutorado, Universidade de Londres.

National Institute Of Standards And Technology (NIST). “The NIST definition of cloud computing”. Gaithersburg, MD (US): NIST, 2011.

Schiller, O. (2015). “Supporting Multi-Tenancy in Relational Database Management Systems for OLTP-style Software as a Service Applications.”, 199 f, Tese de Doutorado, Universidade de Stuttgart.

OLTPBenchmark. OLTPBenchmark, (2019). http://www.oltpbenchmark.com.

Prodanov, C. C. e Freitas, E. C. (2009), “Metodologia do Trabalho Científico – Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico”. Editora Feevale, Novo Hamburgo.

SAP-HANA (2019), “Plataforma de dados in-memory”, 2019, disponível em <https://www.sap.com/brazil/products/hana.html> Acesso em 13 de julho de 2019.

Saleh, A. I., Fouad, M. A., Abu-Elkheir, M. (2014), “A Hybrid Multi-Tenant Database Schema for MultiLevel Quality of Service” in International Journal of Advanced Computer Science and Applications.

SalesForce (2019), “Sales Force”, 2019, disponível em <https://www.salesforce.com/br/> Acesso em 17 de junho de 2019.

Sousa, F. R. C. (2013), “RepliC: replicação elástica de banco de dados multi-inquilino em nuvem com qualidade de Serviço”, 116 f. Tese de Doutorado, Universidade Federal do Ceará.

Tsai W T, Bai X Y e Huang Y. (2014), “Software-as-a-service (SaaS): perspectives and challenges”. Sci China Inf Sci, 57.

Wang, L.; Von Laszewski, G., Kunze, M., Tao, J. (2008), “Cloud computing: a perspective study”, New Generation Computing.

Veras, M. (2012), “Cloud Computing: nova Arquitetura da TI”, 1ª ed. Rio de Janeiro: Brasport.

Velte, A. T., Velte T. J e Elsenpeter R. (2010), “Cloud computing: a practical approach”. New York: McGraw-Hill.