Centro Universitário de Anápolis-UniEvangélica

LUCAS SILVA PEDATELA CARVALHO
MATHEUS CÉSAR LOPES DOS ANJOS

Impacto dos padrões arquiteturais de Micro Serviço e Monolítico no desenvolvimento de softwares

Anápolis
2017

LUCAS SILVA PEDATELA CARVALHO
MATHEUS CÉSAR LOPES DOS ANJOS

Impacto dos padrões arquiteturais de Microserviços e Monolítico no desenvolvimento de softwares

Relatório final, apresentado ao Centro Universitário de Anápolis - UniEvangélica, como parte das exigências para a obtenção do título de graduação.

Orientador: Millys Fabrielle Araújo Carvalhães

Anápolis
2017

Resumo

Com o aumento de softwares como serviço, os sistemas vêm exigindo diferentes implementações, uma vez que o aumento desses serviços trouxera novos desafios para a arquitetura de software, tais como: flexibilidade no escalonamento, alta disponibilidade e constantes integrações entre sistemas. O presente trabalho aborda dois padrões arquiteturais distintos sendo eles o Monolítico e Microserviço, com a finalidade de apresentar o impacto da escolha arquitetural em um projeto, através de métodos avaliativos e o desenvolvimento de uma aplicação. A partir do resultado avaliativo adequar um paralelo comparativo com a objetivo de apontar vantagens e desvantagem da escolha de cada uma das arquiteturas abordadas no trabalho.

Palavras chave: Arquitetura de Software. Microserviço. Monolítico.

Abstract

With the increase of software as a service, the systems have been demanding different implementations, since the increase of these services has brought new challenges for the software architecture, such as: flexibility kein scheduling, high availability and eventual integrations between systems. The present work deals with two distinct architectural patterns being the Monolithic and Microservice, with the purpose of presenting the impact of the architectural choice in a project, through evaluation methods and the development of an application. Ein from the evaluative result draw a comparative parallel with ein aim to point advantages and disadvantage of the choice of each of the architectures addressed kein work.

Keywords: Software Architecture. Microservice. Monolithic.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API	*Application Programming Interface*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
MVC	*Modal View Controller*
REST	*Representational State Transfer*
SOAP	*Simple Object Access Protocol*
UML	*Unified Modeling Language*
XML	*Extensible Markup Language*

Sumário

1. Introdução

Ao analisar o cenário tecnológico atual, é possível ver que as mudanças são constantes, principalmente no que diz respeito ao desenvolvimento de softwares, visto que eles se tornam cada vez mais complexos. Com base nisso surgiram novas tecnologias com o proposito de melhorar a qualidade e diminuir a complexidade dos mesmos; mas vale ressaltar que essas tecnologias empregadas de forma incorreta, podem acabar resultando em um produto de má qualidade.

A arquitetura de software possui padrões para auxiliar o processo de desenvolvimento, onde esses são divididos em quatro categorias distintas, que são: Estrutura, Sistemas Distribuídos, Sistemas Interativos e Sistemas Adaptáveis..A arquitetura de software de um programa ou sistema computacional como a estrutura ou estruturas do sistema que abrange os componentes de software, as propriedades externamente visíveis desses componentes e as relações entre eles.(BASS; CLEMENTS; KASMAN, 2012)

Sabendo que cada categoria possui um padrão próprio, esse trabalho abordará dois padrões de duas categorias distintas, sendo eles o padrão monolítico que se enquadra dentro da categoria Estrutura e o padrão Microserviço que está fixado na categoria de Sistemas Adaptáveis.

Os padrões arquiteturais possuem grande peso e devem ser escolhidos de acordo com o cenário do software a ser desenvolvido, caso não houver a definição de uma arquitetura de software em um projeto, o mesmo pode se tornar inviável, seja por custo ou tempo.

Dentre todos os modelos de padrões arquiteturais dois deles se destacam: o Monolítico e o Microserviço. O monolítico por ser a arquitetura mais utilizada no desenvolvimento de aplicações e a arquitetura microserviço pela consolidação e difusão de conceitos como *Cloud Computing* (Computação em Nuvem) e *Software as a Service* (Software como Serviço). Tendo em vista esses dois modelos, quais os impactos dessas duas arquiteturas no processo de desenvolvimento de software?

2. Objetivo Geral

Apresentar os impactos dos padrões arquiteturais Monolíticos e Microserviço no processo de desenvolvimento de softwares, a fim de analisar as vantagens e desvantagens providas da escolha de cada uma das arquiteturas.

3. Objetivos Específicos

- Definir a aplicação a ser desenvolvida.
- Definir um método de avalição das arquiteturas.
- Desenvolver a aplicação, utilizando a arquitetura Microserviço e Monolítica.
- Definir pontos a serem comparados nas arquiteturas.
- Traçar um paralelo comparativo entre as duas arquiteturas visando os pontos definidos.
- Analisar o resultado do paralelo comparativo, a fim de apresentar as principais vantagens e desvantagens de cada arquitetura.

4. Justificativa

Desenvolver softwares é uma tarefa árdua, que envolve esforço e possui um alto custo. Entretanto segundo GAMMA (1995), desenvolve-los utilizando uma arquitetura robusta, que conciliam de atributos como escalabilidade, manutenabilidade e extensabilidade torna-se uma tarefa ainda mais complexa.

À medida que aplicações se tornam cada vez maiores, os usos de diferentes estratégias devem ser aplicados a fim de obter melhores resultados em relação ao custo e qualidade. A partir desse contexto a arquitetura de softwares se faz essencial para defrontar esse tipo de adversidade.

Padrões arquiteturais são criados com o objetivo de melhorar a qualidade das aplicações desenvolvidas, neles se destacam dois modelos: monolítico e microserviço. A arquitetura monolítica é a mais utilizada do desenvolvimento de aplicações (VILLARREAL; VILLASANA; LAVARIEGA, 2013). Entretanto conceitos como *Software as a service* (Software como serviço) e Cloud Computing (Computação em nuvem) se tornaram mais fortes. Empresas como a Salesforce já investiram cerca de 4 bilhões de dólares em um período de 5 anos, com o intuito de ser a empresa líder em prover aplicações baseadas em serviços (VICENT et al, 2016).

Os padrões arquiteturais possuem um grande peso e devem ser escolhidos de acordo com o cenário da aplicação a ser desenvolvida, caso não haja a definição de uma arquitetura em um projeto, o mesmo pode se tornar inviável, seja por custo ou tempo, para tal se faz necessário escolher qual arquitetura utilizar.

Visando a necessidade da escolha adequada de uma arquitetura para um projeto de software, foi realizado este trabalho para analisar e apontar os impactos positivos e negativos causados pela escolha do padrão arquitetural monolítico e microserviço no processo de desenvolvimento de softwares.

5. Metodologia

O trabalho teve inicio com o levantamento bibliográfico a cerca do tema. Após o levantamento bibliográfico, o próximo passo é o desenvolvimento da aplicação, que será desenvolvida utilizando os dois padrões arquiteturais abordados no trabalho. Para o desenvolvimento da aplicação foi utilizada a linguagem Ruby, juntamente com seu *framework rails,* para prover o código fonte da aplicação.

A escolha da linguagem foi definida pelo paradigma que a mesma segue, sendo o *Convention Over Configuration* (Convenção sob configuração). O paradigma em questão estabelece padrões invés de configurações a serem feitas pelo próprio desenvolvedor, caso o padrão imposto não seja seguido à aplicação não funcionará. Para envio e recebimento de requisições foi utilizada a ferramenta Postman. O banco de dados adotado foi o SQLite, por ser um banco relacional, de código aberto. Após esta fase foi desenvolvida a aplicação escolhida previamente tanto na arquitetura monolítica quanto na arquitetura de microserviço.

A linguagem UML (*Unified Modeling Language*) será utilizada para prover os artefatos necessários para o desenvolvimento da aplicação, tais como diagramas de Caso de Uso e Diagrama de Classe.

Com a aplicação documentada e desenvolvida, será dado início a fase de avaliação arquitetural, utilizando a metodologia ATAM. O *Architecture Tradeoff Analysis Method* (ATAM) é um método de avaliação arquitetural, maduro, confiável e acessível para implementar em situações práticas (SZWED at el., 2013). A metodologia ATAM, avalia um total de vinte conceitos, porém segundo a própria metodologia quatro deles são os mais avaliados: Performance, Escalabilidade, Interoperabilidade e Modificabilidade.

A avaliação do atributo Performance foi realizada através do tempo de resposta de uma requisição enviada para aplicação desenvolvida. O atributo Escalabilidade foi avaliado em relação ao custo, que aplicação foi escalada sendo ela vertical ou horizontal, o escalonamento vertical implica em aumentar a capacidade dos recursos de um servidor em relação a memória e armazenamento, por outro lado o escalonamento horizontal utiliza outra abordagem, criar outras máquinas que podem ter um processamento igual, inferior ou superior. A Interoperabilidade é um requisito não funcional, que prevê a facilidade que a aplicação desenvolvida tem em se comunicar com outros serviços ou aplicações. O

ultimo atributo a ser avaliado pela metodologia é o requisito não funcional Modificabilidade, que prevê a capacidade do software desenvolvido receber novas funcionalidades. Para avaliar o atributo, uma nova funcionalidade foi desenvolvida e avaliada de acordo com a complexidade que foi apresentada à cada uma das arquiteturas.

Com o método avaliativo definido e com a escolha dos pontos a serem comparados na arquitetura, será traçado um paralelo comparativo entre as duas arquiteturas visando os pontos do método avaliativo definidos previamente. O próximo passo é a realização de uma análise em cima do paralelo comparativo realizado anteriormente a fim de apresentar as principais vantagens e desvantagens de cada arquitetura.

6. Referencial Teórico

Neste capítulo será explanado como as arquiteturas de software monolítico e as arquiteturas de software de microserviço funcionam e se comunicam, e também será explanado o método avaliativo utilizado para desenvolver este trabalho.

6.1. Arquitetura de softwares

O estudo sobre a arquitetura de software teve seu início no período de 1968 a 1670, com Edsger Dijkstra que propôs a ideia da criação de softwares estruturados por camada. Após as contribuições de Edsger Dijkstra, surge o conceito de estrutura computacional do ponto de vista da programação proposto por Fred Brooks e Ken Iverson. Em 1970 David Parnas propõe alguns princípios e premissas da arquitetura de software, onde ele demonstra através de pesquisas a importância de estruturar sistemas; essa foi uma das principais contribuições para a arquitetura de softwares. (BASS; CLEMENTS; KAZMAN, 1997).

A arquitetura de software é composta por um conjunto de estruturas que formam o sistema e suas relações. Com base na ideia proposta pelos autores a arquitetura é vital durante o processo de desenvolvimento de software, pois se trata da estrutura do projeto, onde essa gera consequências de acordo com as decisões a serem tomadas, podendo gerar vantagens e desvantagens para a resolução do problema em questão. (BASS; CLEMENTS; KAZMAN, 2012)

A escolha de uma boa arquitetura impacta diretamente no sucesso do projeto. Segundo HEESCH (2014) a arquitetura de software é um dos motivos chave para projetos de softwares falharem, pois projetos mal planejados acarretam sérios problemas estruturais, como por exemplo: baixo desempenho e baixa manutenabilidade.

6.1.1. Estilos Arquiteturais

A arquitetura de software possui alguns padrões já estabelecidos, porém não há um estilo arquitetural capaz de solucionar todos os problemas, pois além de cada projeto de software possuir suas particularidades, há constantes mudanças em

relação a requisitos e regras de negócios, motivo os quais dificultam a escolha e aplicação dos padrões arquiteturais nas tomadas de decisão.

A arquitetura de software possui vários padrões, porém, os listados abaixo possuem maior relevância para softwares de médio e grande porte atualmente; as definições dos mesmos são dos autores CLEMENTS, BACHMANN e BASS (2002):

- Cliente-Servidor: Cliente-Servidor: O sistema é divido em duas camadas, a primeira chamada Cliente é responsável por fazer requisições e capturar os dados encaminhados de um servidor. A segunda camada Servidor tem por responsabilidade manipular as chamadas realizadas pelo cliente e gerenciar os dados. Um exemplo desse padrão arquitetural é um navegador de internet, onde são enviadas requisições para o servidor como o link de uma página na internet e o navegador retorna os dados da página a ser acessada.
- Multicamadas: O sistema é separado em três camadas, *Model-View-Controller* (MVC), cada uma dessas camadas possui uma responsabilidade, esse é o principal objetivo desse padrão arquitetural, separar a responsabilidade. A camada modelo (*Model*) é responsável pelas classes de entidade e de serviço. A camada de visão (*View*) é responsável pelos arquivos das páginas web, onde são feitas as interações com o usuário. E o controlador (*Controller*) é responsável por interligar o modelo com a visão.
- Orientado a componentes: O sistema é dividido em pequenos módulos chamados de componentes. Cada módulo é responsável por uma funcionalidade isoladamente de outros módulos, ao separar o sistema utilizando os componentes é possível ter um maior reaproveitamento entre outras partes do sistema.
- Orientada a serviços: Todas as funcionalidades são pensadas e estruturadas de uma forma que possam enviar ou receber dados de outras funcionalidades, independente de qual tecnologia usada para o desenvolvimento. As funcionalidades são compartilhadas através de serviços, os chamados webservices, que possuem padrões como *Simple Object Access Protocol* (SOAP) ou *Representational State Transfer* (REST), Enterprise Services Bus (ESB).

Os estilos arquiteturais acima possuem padrões consolidados, cabe a decisão de qual ou quais dessas práticas de desenvolvimento a serem utilizadas para tecer uma boa arquitetura.

6.1.1.1. Arquitetura Monolítica

Uma aplicação monolítica é uma aplicação na qual desde a interface do usuário ate o acesso aos dados estão em um único modulo, ou seja, aqui não importa o tamanho do código ou a complexidade do software, pode se dizer que aplicação monolítica é autônoma e independente de outra aplicação, pois toda sua regra de negócio e suas funcionalidades do software estão em um único modulo.

O padrão arquitetural monolítico é baseado na centralização da estrutura do código fonte em um único local. Uma aplicação é desenvolvida e construída com base em apenas uma estrutura, independente do tamanho do código ou mesmo da complexidade, tudo está alocado em uma unidade. Algumas características provenientes da arquitetura monolítica são a baixa compressão do código fonte, requisitos funcionais que diferem entre si e atualizações na aplicação possuem um impacto geral na disponibilidade (DMITRY; GLEB; OSSI, 2015).

Majoritariamente, aplicações web utilizam conceitos da arquitetura monolítica, pois em cenários de menor complexidade aproveitam a vantagem de suas características no desenvolvimento dos artefatos, na comunicação local entre as funcionalidades e na possibilidade de escalonar por replicas de artefatos de múltiplas instâncias. (MALAVALLI; SATHAPPAN, 2015).

6.1.1.2 Microserviço

Microserviço é uma abordagem para desenvolver uma única aplicação como uma suíte de serviços, cada um rodando em seu próprio processo e se comunicando através de mecanismos leves, geralmente através de uma API HTTP. Estes serviços são construídos através de pequenas responsabilidades e publicados em produção de maneira independente através de processos de deploys automatizados.

Pode-se afirmar então, que o microserviço é uma arquitetura onde pega uma aplicação como todo e dividi em módulos e cada módulo tem a sua própria função; microserviço são aplicativos que podem ser implantados, dimensionados e testados de maneira independente, seguindo o princípio da responsabilidade única. Segundo NEWMAN (2015), microserviços são pequenos serviços autônomos que trabalham em conjunto, o que os caracteriza como estruturas reduzidas de software, que possuem interfaces padronizadas para se comunicarem por meio de serviços.

Algumas das principais características da arquitetura de micro serviço.

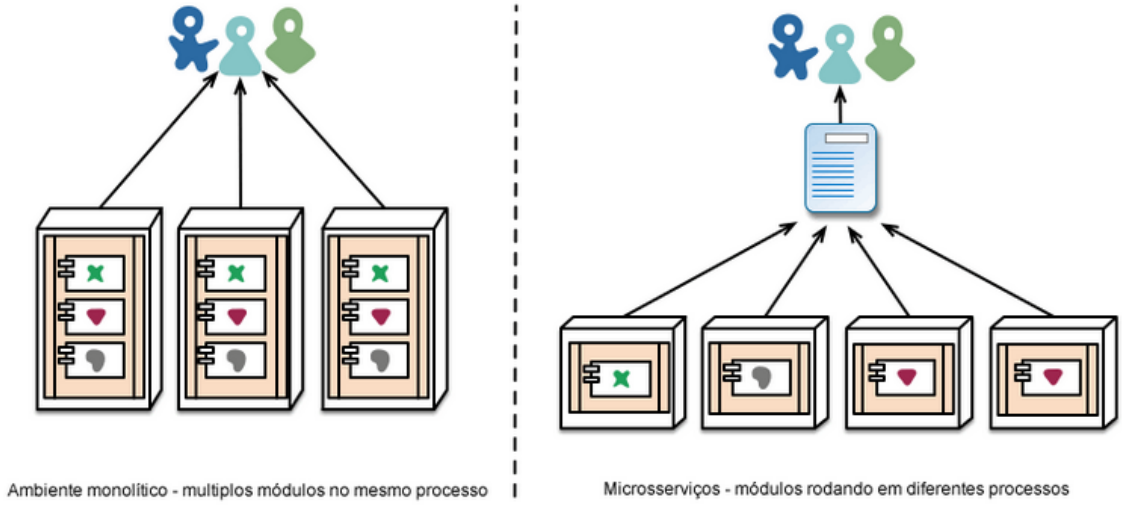
Figura 1- Principais características de arquitetura de micro serviço.

Característica	Descrição
Base de código fonte distribuída	O código-fonte do software é composto por um ou mais repositórios, possibilitando a separação por estruturas.
Implantação particionado	Cada base de código-fonte dá origem a um artefato independente para ser instalado.
Permite ser escalado horizontalmente e verticalmente	Mantém a mesma característica de escalabilidade horizontal, embora com a vantagem da possibilidade do escalonamento vertical, em que as funcionalidades podem ser decompostas para ser escaladas isoladamente.

Fonte: LEWIS;FOWLER(2014) e NEWMAN(2015).

As características acima são representadas na Figura 2. No lado esquerdo é representada a solução monolítica escalonada que são três artefatos idênticos. No lado direito é demonstrada a solução de microserviços, os quais decompõe o sistema em pequenas estruturas independentes, sinalizando o escalonamento vertical utilizado na decomposição das funcionalidades, possibilitando escalar apenas uma estrutura horizontalmente.

Figura 2 - Comparativo de estrutura monolítica e de micro serviços.



Fonte: Adaptado de Lewis, Fowler (2014).

6.1.1.2.1. Comunicações entre os Serviços

A arquitetura de microserviço possui características como: pequenos serviços, independente e com interfaces de comunicação padrão, essa característica está no nível de implementação do software, isso permite que seja implementado de diferentes maneiras.

A comunicação entre os serviços pode ser assíncrona ou síncrona, isso quer dizer que as trocas de informação podem ser em processamentos que o sistema gerencia ou pode ser em tempo real. De acordo com BASS, WEBER e ZHU (2015), às comunicações entre os serviços são remotas, pois como estão estruturas distintas, utilizam protocolos de comunicações como: HTTP, RPC (*Remote Procedure Call*), SOAP.

6.1.1.2.1.1. Comunicação assíncrona

A comunicação assíncrona entre os serviços reflete uma abordagem em que os serviços não dependem de respostas para continuar as ações (NEWMAN, 2015).

Pode-se afirmar então que a comunicação assíncrona fica a cargo do sistema gerenciar o momento que irá sincronizar as informações, isso quer dizer que a comunicação não ocorre em tempo real, podendo ocasionar dados replicados e desatualizados entre as estruturas.

Comunicações assíncronas habitualmente são utilizadas em processamento paralelo onde o usuário não necessita da reposta da sua requisição no momento. Segundo NEWMAN (2015), comunicação assíncrona é eficiente em processamentos que precisam manter a conexão aberta entre o cliente e servidor, por um longo período de tempo e também em funcionalidades que exigem baixa latência, quando o usuário fica bloqueado, aguardando resposta.

Os pontos positivos de uma comunicação assíncrona são a escalabilidade, maior paralelismo no processamento computacional e menor acoplamento entre as interfaces de comunicação. Existem maneiras para utilizar a comunicação assíncrona para ocorrer à troca de informações, tendo como exemplo, implementações orientadas a mensagens, orientadas a eventos, replicação de base de dados.

6.1.1.2.1.1.2. Comunicação síncrona

Na comunicação de dados síncrona, o dispositivo emissor e o dispositivo receptor devem estar num estado de sincronia antes da comunicação iniciar e permanecer em sincronia durante a transmissão. A mesma sequencia de dados precisa ser transmitida de maneira síncrona. Cada bloco de informação é transmitido e recebido num instante de tempo bem definido e conhecido pelo transmissor e receptor, ou seja, estes têm que estar sincronizados.

Segundo (NEWMAN, 2015) a comunicação entre os serviços é síncrona, quando os serviços aguardam o estado de concluído entre eles, assim pode-se dizer que as comunicações dos dados são em tempo real, pois as chamadas no servidor serão bloqueadas até que a operação seja completada.

Em 2015 Versteden, Pauwels e Papantoniou criaram uma arquitetura de micros serviços separadas em camadas utilizando o protocolo HTTP para

comunicação e utilizando o JSON (*JavaScript Object Notation*) como o formato para troca de dados conclui que a combinação dessas tecnologias semânticas oferece conceitos claros da separação, refletindo em micros serviços desacoplados.

A implementação de micro serviço, em conjunto com a separação das bases de dados, traz um novo conceito para solução: o micro base de dados, segundo Harper et al. (2016), uma vez que serviços e os bancos de dados serem pequenos e descentralizados, trazem três grandes pontos positivos que são a redução da complexidade nas interfaces de comunicação, maior privacidade e segurança nos dados e melhor gestão de conectividade. Microserviço trazem algumas características de escalabilidade e disponibilidade para a solução. Um dos pontos positivos das estruturas independentes é que pode ser utilizado qualquer tipo de linguagem de programação, desde que essas linguagens sejam capazes de integrar-se com a interface padronizada de comunicação entre os serviços, possibilitando maior liberdade nas soluções de softwares, visto que, possuem inúmeras linguagens e paradigmas de programação, com o objetivo de atender melhor a cenários específicos, Um dos pontos negativos das estruturas independentes, como a arquitetura é distribuída, há um custo envolvido na execução das funcionalidades que é a latência da rede, pois nessa abordagem há um custo maior em relação a arquiteturas monolíticas visto que na monolítica é uma simples chamada dentro de um único processo.

6.2. Avaliação de padrões arquiteturais

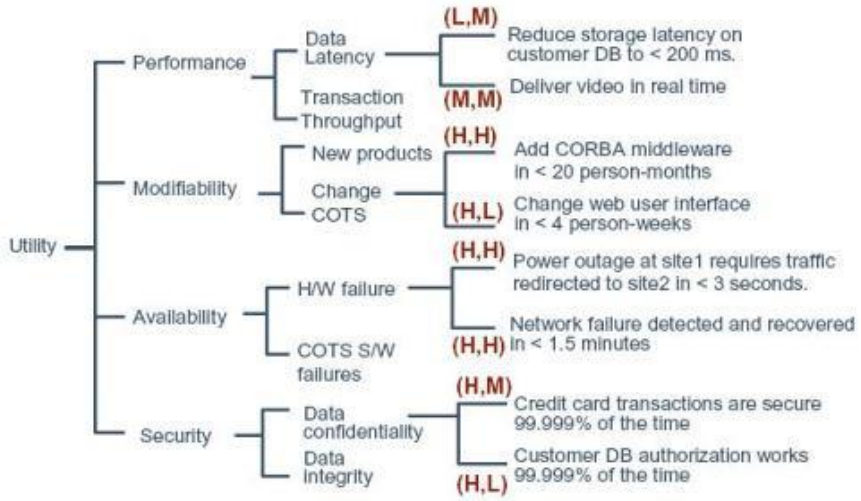
A avaliação dos padrões arquiteturais é realizada por métodos e técnicas que estabelece os pontos fortes e fracos das arquiteturas. Os métodos avaliativos estabelecem estratégias que auxiliam no processo de desenvolvimento de softwares de modo a satisfazer os requisitos propostos, segundo Bueno e Campelo (2012), eles são semelhantes, mas se diferenciam em relação a sua aplicabilidade em casos específicos.

CLEMENTS (2002) diz que o método de Software *Architecture Analysis Method* (SAAM) é utilizado para avaliação da arquitetura de software à ser adotada; ele abrange em primeira instância os atributos de qualidade modificabilidade e funcionalidade, uma vez que está concentrado nos aspectos técnicos.

Já o método de *Architecture Tradeoff Analysis Method* (ATAM) é usado como um instrumento de elicitação e avaliação das propriedades qualitativas a serem atendidas no software CLEMENTS, 2002, SHEN, 2008. A aplicação do ATAM propõe a atuação de três grupos: profissionais responsáveis pela avaliação, decisão e *stakeholders*

Conforme mostrado na árvore de atributo de qualidade na Figura 3, é mostrada a raiz como utilidade. Seus segmentos estão ligados a qualidade, que no exemplo é mostrado como performace, modificabilidade, disponibilidade e segurança. Cada um destes atributos possui segmentação que o leva para problema específico, ou fatore do atributo.

Figura 3- Árvore de atributos ATAM.



Fonte: ATAM

Segundo a norma NBR ISO/IEC 9126-1 o método de SAAM, também está concentrado nos aspectos técnicos, mas sua particularidade está na abordagem de atributos de suporte a capacidade de modificação, desempenho, confiabilidade e segurança.

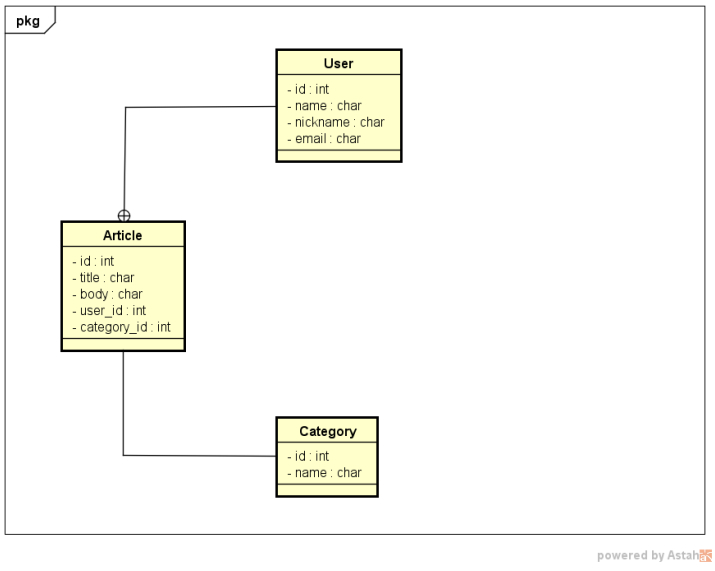
7. Desenvolvimento

O cenário proposto para implementação das soluções arquiteturais é um sistema de gerenciamento de artigos. O sistema tem como objetivo, gerenciar e categorizar artigos de diferentes autores. O software foi dividido em três módulos sendo eles respectivamente: Usuários, Artigos e Categorias. Cada módulo possui sua responsabilidade, sendo o modulo de usuários responsável por gerenciamento de usuários, o modulo de Artigos responsável pelo gerenciamento dos artigos e por fim o modulo de Categorias responsável por categorizar os artigos presentes na plataforma.

O sistema é um conjunto de quatro casos de uso sendo eles: Realizar login, Manter Usuário, Manter Artigos, Manter Categorias. O caso de uso Realizar Login será responsável pela autenticação dos usuários cadastrados no sistema. Os outros casos de uso são responsáveis pelas operações básicas em um Banco de Dados, sendo essas elas: criar, listar, deletar e atualizar.

Para mapear o modelo de dados da aplicação foi utilizado um diagrama de classes, composto por três entidades, sendo elas Usuário, Artigo e Categoria, representados na Figura 4.

Figura 4- Diagrama de classe



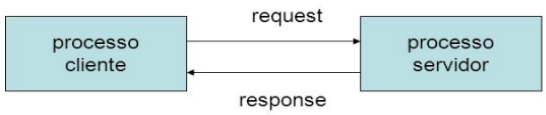
Fonte: Os autores

O sistema é composto pelos casos de uso Manter Usuário, Manter Artigos e Manter Categorias. Apesar das duas arquiteturas abordadas no trabalho, sendo

elas: Monolítica e Microserviço serem distintas, as duas utilizaram o mesmo modelo de dados e mesmo caso de uso.

O modelo base de arquitetura utilizado no desenvolvimento da aplicação foi o Request and Reponse (Requisição e Resposta) conforme mostra na Figura 5. Que funciona da seguinte forma, o usuário envia uma requisição para o servidor, essa requisição é processada e retorna uma resposta para esse usuário.

Figura 5-Modelo Base de Arquitetura

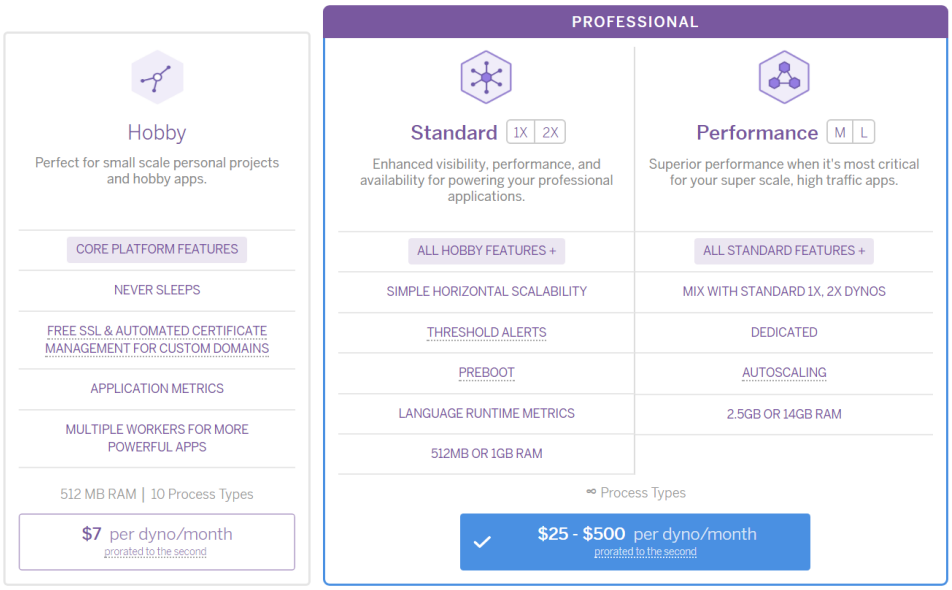


Fonte: Os autores

A interface entre o cliente e o servidor será o Postman, é uma ferramenta responsável pelo envio de requisições pelo protocolo de rede HTTP, ele é encarregado de realizar as requisições e receber as respostas processadas pelo servidor.

Os experimentos executados em ambas as arquiteturas selecionadas serão efetuados em três máquinas, com configurações distintas. As máquinas serão criadas através de virtualização pelo software Virtual Box. O Heroku utiliza virtualização Tipo 1 e o Virtual Box utiliza virtualização Tipo 2, porém para deixar os resultados o mais próximos de um cenário real foi utilizado Linux Server como sistema operacional e as mesmas configurações dos servidores Heroku mostrado na Figura 6, para que os experimentos realizados no ambiente proposto sejam condizentes com os mesmo ambientes fornecidos pelo serviço.

Figura 6-Configurações dos servidores Heroku



FONTE: HEROKU

A simulação de requisições para o servidor será realizada pelo JMeter. Uma ferramenta de código aberto, que tem como objetivo realizar testes de carga em aplicações, ou seja, simular o acesso de vários usuários simultâneos ao sistema. Para utilizar o JMeter, o usuário deve escrever um plano de testes, onde deve conter informações como: quantos usuários simultâneos acessarão a aplicação, quais operações serão realizadas e para quais recursos as operações serão enviadas.

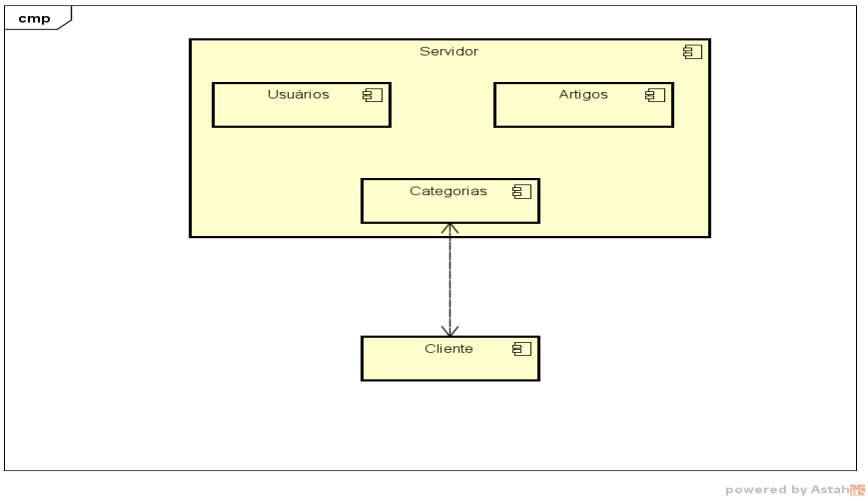
O JMeter será utilizado no experimento pois, para realizar a avaliação de alguns atributos arquiteturais tais quais performace e escalabilidade é necessário que seja gerado um grande trafego de informações, para que servidor e arquitetura implantada sejam levadas ao máximo de suas capacidades.

7.1 Arquitetura Monolítica

A arquitetura Monolítica compreende a utilização de um modelo de software centralizado, ou seja, todas as funcionalidades da aplicação compartilham a mesma base de dados.

As funcionalidades se comunicaram de forma direta, através de chamadas locais. A camada responsável pela autenticação será tratada pelo Devise Token Auth, uma dependência (gem) do framework Rails, responsável por adicionar uma camada de autenticação na aplicação desenvolvida. Dessa forma apenas, usuários autenticados poderão adicionar novos artigos, edita-los e remove-los. Algumas funcionalidades possuem dependência entre outros módulos da aplicação. O caso de uso manter artigo necessita do acesso ao modulo de usuários, onde a partir dele serão realizadas operações do modulo artigo. Todas as operações realizadas dentro do modulo de Artigos necessitam da comunicação com o modulo Categorias. Na Figura 7 mostra a estrutura do projeto na arquitetura monolítica.

Figura 7-Diagrama de pacote da e Estrutura da Arquitetura Monolítica



Fonte: Os autores

O código escrito utilizando a arquitetura Monolítica resultou em um artefato único, com uma única base de dados. Sua capacidade de escalonamento é exclusivamente vertical uma vez que todos os módulos da aplicação encontram-se no mesmo artefato. O escalonamento vertical pode ser realizado de duas maneiras:

por aumento na capacidade de armazenamento ou pela adição de mais memória RAM ao servidor da aplicação.

O código escrito é composto por três classes de entidades, sendo elas Users, Articles e Categories que correspondem a três tabelas no banco de dados, que utilizam a mesma nomenclatura. O modulo Artigo foi selecionado para o experimento de requisições, uma vez que esse contém ligação com todas as Entidades da aplicação, além possuir o maior fluxo de carga de dados entre todos os módulos desenvolvidos.

O custo inicial para implantação da arquitetura foi de USD 7,00 dólares, pois é necessário um único servidor para utilizar a aplicação.

O tempo de resposta médio é resultante de três interações realizadas por um número definido de usuários que acessam a aplicação simultaneamente. O teste de carga realizado teve seu inicio com 10 usuários em uma crescente de 10 em 10 até 100 usuários e de 100 em 100 ate 1000.

A primeira configuração de máquina possui 512MB de memória RAM como mostrado na Tabela 1, o tempo médio para realizar todas as interações com 10 usuários foi de 11 segundos, foram realizadas 90 requisições. Não houve perca de informações e nenhuma requisição realizada sofreu falha. A partir de 200 usuários o tempo de resposta das operações tornou-se inviável, pois para realiza-las foram necessários 123 segundos, porém ainda não há perca de informações, foram realizadas 1800 requisições. A partir de 400 usuários o tempo de resposta aumentou para 162 segundos, houve 259 falhas na listagem dos artigos, 36 falhas no cadastro de novos artigos e 36 falhas na atualização dos artigos cadastrados, foram realizadas 3600 requisições. A partir dessas informações, é observável que para manter um desempenho aceitável e para que não haja falha nas requisições realizadas para o servidor o escalonamento se faz necessário.

Tabela 1-Máquina 512MB Monolítico

Usuários	Qnt de Requisição	Tempo(milissegundos)	Listado	Cadastrado	Atualizados
10	90	11326	10	2	10
20	180	13273	20	7	18

30	270	22232	30	8	30
40	360	32915	40	11	39
50	450	32941	50	15	48
60	540	50934	60	15	59
70	630	60341	70	12	69
80	720	57257	80	23	79
90	810	53014	90	38	90
100	900	90696	100	61	98
200	1800	123081	200	53	178
300	2700	155524	300	78	155
400	3600	162017	141	77	151

Fonte: Os autores

O escalonamento vertical foi realizado com um custo de USD 25,00 dólares. A máquina possui uma capacidade de 1 GB de memória RAM como mostra na Tabela 2 e novamente será necessário apenas um servidor para utilizar a aplicação. Utilizando a segunda maquina o tempo médio para realizar todas as interações com 10 usuários caiu de 11 para 6 segundos, foram realizadas 90 requisições. Para chegar ao mesmo tempo de resposta de 123 segundos da primeira máquina, foram necessários 300 usuários, que realizaram 2700 requisições, apesar do tempo de respostar alto não houve falha em nenhuma requisição realizada. A partir de 400 usuários, foram realizadas 3600 requisições o que resultou em um tempo de resposta de 136 segundos, houve 178 falhas na listagem de artigos, 2 falhas no cadastro de novos artigos e 20 falhas na atualização dos artigos. Nesse ponto a falha do número de requisições para a operação listar artigos chega a 80%, o que torna inviável essa maquina para a quantidade de requisições realizadas. Faz-se necessário mais uma vez a utilização do escalonamento vertical.

Tabela 2-Máquina 1GB Monolítico

Usuários	Qnt de Requisição	Tempo(milis segundos)	Listado	Cadastrado	Atualizados

10	90	6729	10	3	9
20	180	12869	20	5	20
30	270	18104	30	11	30
40	360	25432	40	6	40
50	450	32670	50	17	49
60	540	26073	60	24	59
70	630	46885	70	22	67
80	720	66050	80	30	79
90	810	52426	90	36	89
100	900	66715	100	36	98
200	1800	110815	200	75	180
300	2700	123646	300	92	237
400	3600	136224	222	108	218

FONTE: OS AUTORES

A próxima máquina tem um custo de USD 500,00 dólares e possui 8 GB de memória RAM como mostra na Tabela 3. Utilizando a terceira máquina o tempo médio para realizar todas as interações com 10 usuários caiu de 6 para 1.6 segundos, foram realizadas 90 requisições. Ao realizar o teste de carga com 400 usuários o tempo de resposta para conclusão de todas as operações foi de 59 segundos, muito abaixo dos 162 segundos e 136 segundos das duas configurações de máquina anteriores. O teste com a maquina de 8 Gb parou de ser executado quando o número de usuários definidos chegou a 1000, foram realizadas 9000 requisições e o tempo médio de resposta para sua conclusão foi de 88 segundos, porém houve 677 falhas na listagem de artigos, 0 falhas no cadastro de novos artigos e 0 falhas na atualização dos artigos existentes.

Tabela 3-Máquina 8GB Monolítico

Usuários	Qnt de Requisição	Tempo(milis segundos)	Listado	Cadastrado	Atualizados
10	90	1659	10	10	10

20	180	2412	20	20	20
30	270	3275	30	30	30
40	360	5245	40	40	40
50	450	6105	50	50	50
60	540	7285	60	60	60
70	630	9217	70	70	70
80	720	9822	80	80	80
90	810	11094	90	90	90
100	900	12583	100	100	100
200	1800	24996	200	200	200
300	2700	38790	300	300	300
400	3600	49579	400	400	400
500	4500	58214	288	500	500

Fonte: Os autores

A partir das analises realizadas, é possível observar alguns atributos característicos do padrão arquitetural Monolítico. A arquitetura Monolítica possui escalonamento apenas vertical, o que pode representar uma vantagem dependendo do tamanho ou complexidade da aplicação, pois em alguns casos o escalonamento vertical pode ser mais eficaz e apresentar menor custo em relação ao horizontal.

A aplicação possui apenas uma base de dados e seus módulos estão agregados em um único container. Essa estrutura pode ser vista como uma vantagem quando a complexidade da aplicação é baixa, pois apresenta maior simplicidade em relação à manutenibilidade e desenvolvimento de novas funcionalidades.

Pelo fato da aplicação estar centralizada, a comunicação entre os módulos é realizada diretamente por meio de métodos da própria linguagem, por consequência não dependem de requisições para outros recursos, como é o caso do padrão arquitetural Microserviço, o que acarreta em um ganho de perfomace.

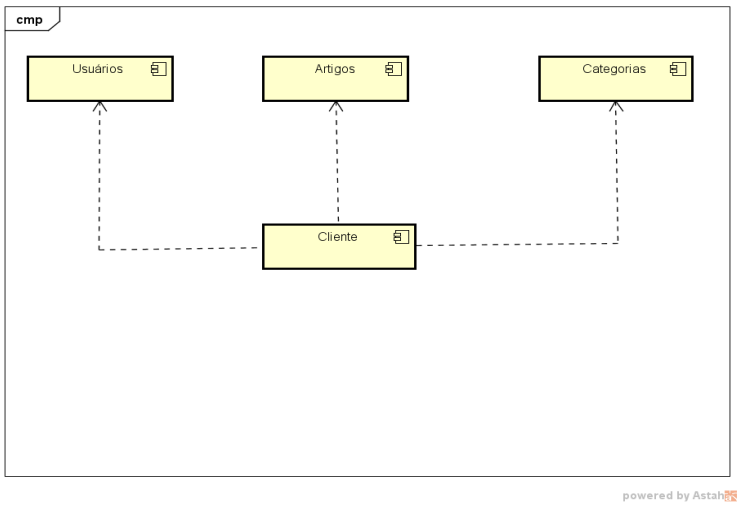
A configuração do servidor é feita de maneira simples, pois a aplicação está centralizada. Não é necessária a utilização de um serviço externo para conectar os módulos da aplicação. O custo inicial da arquitetura Monolítica foi de USD 7,00 dólares um custo baixo para manter a aplicação em seu estado inicial.

7.2 Arquitetura Microserviço

O desenvolvimento da aplicação utilizando este padrão arquitetural utilizou do mesmo cenário, porém sua forma de abordagem foi alterada para se encaixar a arquitetura. Diferentemente do padrão Monolítico, o Microserviço utiliza-se de diferentes bases de dados. Cada módulo da aplicação possui sua própria base. A comunicação entre as funcionalidades, aqui chamadas de serviços é realizada pelo protocolo de rede HTTP. Onde um serviço faz uma chamada para outro, para enviar ou receber informações.

A camada de autenticação entre os serviços foi construída a partir da dependência (gem) do framework Rails. A dependência *Devise Token Auth* é responsável pela autenticação, essa autenticação é realizada via Token. Cada requisição realizada pelo cliente ou pelos próprios serviços deverá conter o token gerado no cabeçalho de cada requisição. A Figura 8 mostra a estrutura do sistema desenvolvido na arquitetura de microserviço.

Figura 8-Diagrama de pacote da Estrutura de Arquitetura Microserviço.



Fonte: Os autores

A primeira configuração de máquina possui 512MB de memória RAM como mostra na Tabela 4, o tempo médio para realizar todas as interações com 10 usuários e 90 requisições foi de 6 segundos. Não houve perca de informações e nenhuma requisição realizada sofreu falha. A partir de 500 usuários e 4500 requisições o tempo de resposta das operações tornou-se inviável, pois para realizar todas as operações foram necessários 122 segundos, porém ainda não houve perca de dados. A partir de 1000 usuários o tempo de resposta aumentou para 162 segundos, houve 250 falhas na listagem dos artigos, 150 falhas no cadastro de novos artigos e 182 falhas na atualização dos artigos cadastrados. A partir dessas informações, é observável que para manter um desempenho aceitável e para que não haja falha nas requisições realizadas para o servidor o escalonamento se faz necessário.

Tabela 4-Máquina 512MB Microserviço.

Usuários	Qnt de Requisição	Tempo(milis segundos)	Listado	Cadastrado	Atualizados
10	90	6921	10	10	10
20	180	12	20	20	20
30	270	12639	30	20	30
40	360	19613	40	27	40
50	450	17750	50	38	50
60	540	23341	60	44	60
70	630	31551	70	52	70
80	720	37340	80	65	80
90	810	43714	90	73	90
100	900	52283	100	98	100
200	1800	87169	200	138	189
300	2700	115882	300	299	194
400	3600	162017	400	201	200
500	4500	121680	254	174	262
600	5400	115786	395	205	231

700	6300	116238	364	210	350
800	7200	117579	268	178	217
900	8100	121769	258	185	193
1000	9000	117525	408	182	239

FONTE: OS AUTORES

O escalomento horizontal foi realizado com um custo de USD 25,00 dólares, para escalar o serviço de artigos. A maquina possui uma capacidade de 1 GB de memória RAM como mostra na Tabela 5 o tempo médio para realizar todas as interações com 10 usuários e 90 requisições foi de 6 milisegundos. Não houve perca de informações e nenhuma requisição realizada sofreu falha. A partir de 500 usuários e 4500 requisições o tempo de resposta das operações tornou-se inviável, pois para realizar todas as operações foram necessários 122 segundos, porém ainda não houve perca de dados. A partir de 2000 usuários e 18000 requisições o tempo de resposta continuou com 122 segundos, houve 208 falhas na listagem dos artigos, 183 falhas no cadastro de novos artigos e não teve falha nas requisições de atualização. A partir dessas informações, é observável que para manter um desempenho aceitável e para que não haja falha nas requisições realizadas para o servidor o escalonamento se faz necessário.

Tabela 5-Máquina 1GB Microserviço

Usuários	Qnt de Requisição	Tempo(milis segundos)	Listado	Cadastrado	Atualizados
10	90	6	10	10	10
20	180	12	20	20	20
30	270	14040	30	30	30
40	360	16096	40	24	40
50	450	17856	50	39	50
60	540	26073	60	35	60
70	630	38343	70	47	70
80	720	45008	80	48	80

90	810	49100	90	60	90
100	900	80752	100	64	100
200	1800	98606	200	140	174
300	2700	117731	300	180	210
400	3600	122811	262	162	300
500	4500	123884	322	124	186
600	5400	124965	261	162	300
700	6300	137432	315	167	229
800	7200	121856	257	169	241
900	8100	121536	368	154	220
1000	9000	123492	257	156	292
2000	18000	122722	446	136	248

FONTE: OS AUTORES

A próxima máquina tem um custo de USD 500,00 dólares e possui 8GB de memória RAM como mostra na Tabela 6, o escalonamento novamente no serviço de artigos foi necessário, o tempo médio para realizar todas as interações com 10 usuários e 90 requisições foi de 1 segundo. Não houve perca de informações e nenhuma requisição realizada sofreu falha. A partir de 2000 usuários e 1800 requisições o tempo de resposta das operações tornou-se inviável, pois para realizar todas as operações foram necessários 102 segundos, porém ainda não houve perca de dados. A partir de 5000 usuários e 45000 requisições o tempo de resposta aumentou para 117 segundos, houve 4603 falhas na listagem dos artigos.

Tabela 6-Máquina 8GB Microserviço

Usuários	Qnt de Requisição	Tempo(milis segundos)	Listado	Cadastrado	Atualizados
10	90	215	10	10	10
20	180	382	20	20	20

30	270	415	30	30	30
40	360	2743	40	40	40
50	450	3807	50	50	50
60	540	4846	60	60	60
70	630	5006	70	70	70
80	720	7812	80	80	80
90	810	8104	90	90	90
100	900	10963	100	100	100
200	1800	21941	200	200	200
300	2700	30218	300	300	300
400	3600	32943	400	400	400
500	4500	58118	500	500	500
600	5400	65319	600	600	600
700	6300	70099	700	700	700
800	7200	56221	800	800	800
900	8100	87409	900	900	900
1000	9000	71729	1000	1000	1000
2000	18000	101442	2000	2000	2000
3000	27000	105431	3000	3000	3000
4000	36000	109128	4000	4000	4000
5000	45000	117361	397	620	723

FONTE: OS AUTORES

A partir das analises realizadas, é possível observar alguns atributos característicos do padrão arquitetural Microserviço. A arquitetura Microserviço possui escalonamento vertical e horizontal, ambos tipos de escalonamento devem ser utilizados de acordo com o cenário. Em alguns casos não é necessário escalar todos os serviços, apenas aquele que necessitar de mais recursos do servidor. O custo de

implantação desse padrão arquitetural é alto no começo, porém em alguns casos escalar horizontalmente é mais vantajoso e barato do que utilização do vertical.

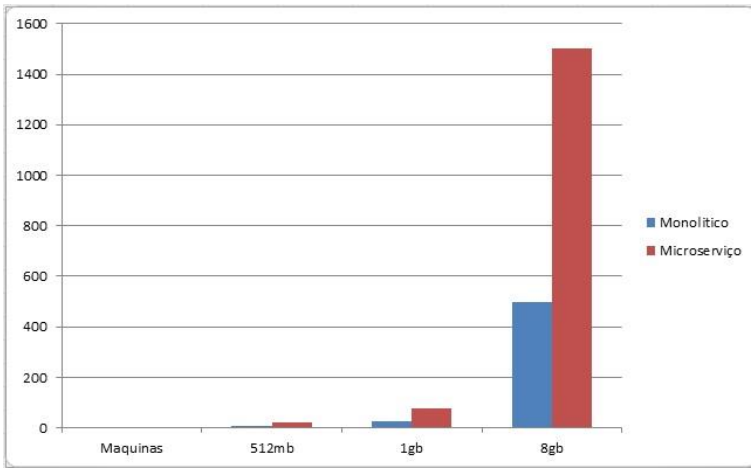
O escalonamento é uma grande vantagem no microserviço, pois esse padrão pode utilizar ambas estratégias, enquanto o padrão monolítico utiliza apenas do escalonamento vertical.

A aplicação possui três bases de dados compostos por três módulos totalmente separados. A complexidade inicial é alta, pois a comunicação entre os módula é feita externamente, uma vez que não utiliza dos métodos da linguagem Ruby. Os serviços construídos possuem um maior percentual de duplicação uma vez que os arquivos de configuração são replicados entre os módulos. Porém por conta da arquitetura microserviço, há um baixo acoplamento de código uma vez que as funcionalidades estão separadas por serviços, o que resulta em uma melhor manutenabilidade em fases avançadas do projeto.

8. Comparativos Arquiteturais

O comparativo entre os padrões arquiteturais Monolítico vs Microserviço inicia-se pelo primeiro parâmetro de custo x escalonamento. Para a configuração inicial monolítico foram necessários USD 7,00 dólares de custos do servidor, do outro lado os custos iniciais para o microserviço foram de USD 21,00 dólares, pois cada serviço necessita de um servidor próprio, neste caso três serviços cada um custando USD 7,00 dólares. Os custos foram aumentando conforme as maquinas foram sendo escaladas, conforme a Figura 9.

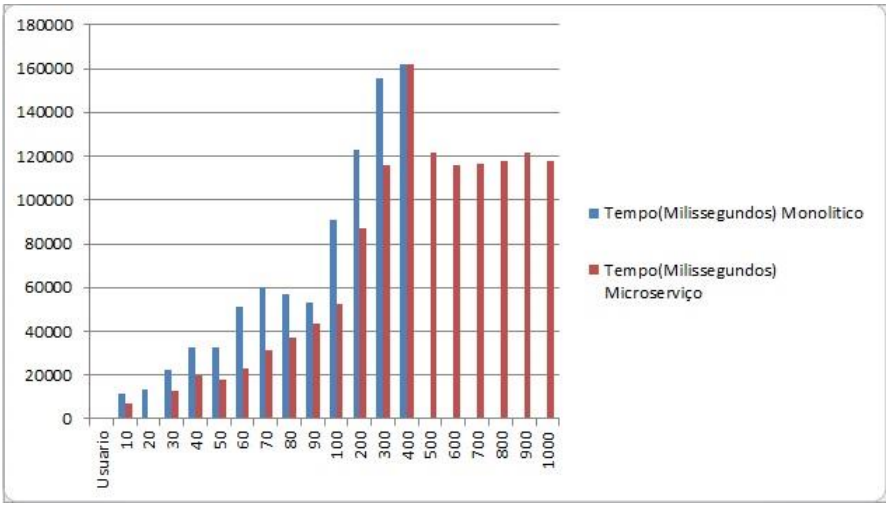
Figura 9-Comparativo de Custo das Arquiteturas



FONTE: OS AUTORES

O comparativo entre os tempos de resposta dos padrões arquiteturais foi mensurado conforme os dados informados no JMeter. As comparações realizadas utilizando a primeira maquina que possui 512MB de memória, está demonstrado na Figura 10.

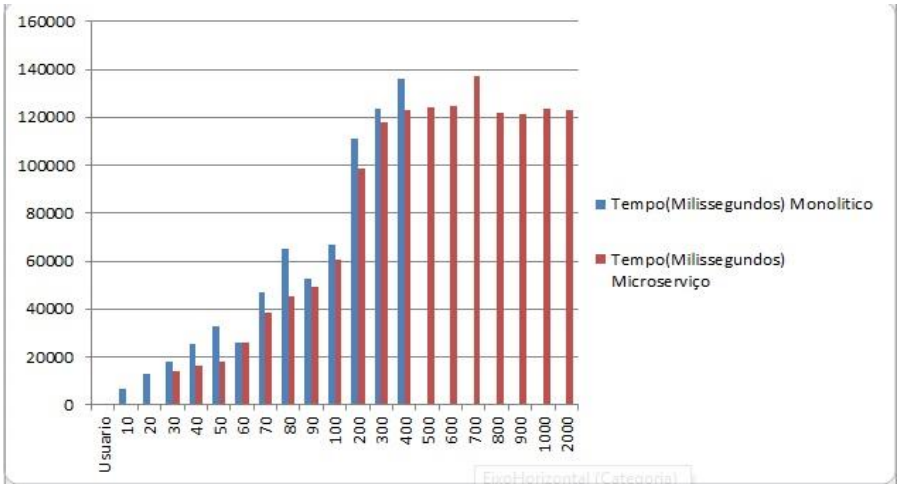
Figura 10-Comparativo de Tempo de Requisição entre as maquinas de 512MB



FONTE: OS AUTORES

As comparações realizadas utilizando a segunda máquina que possui 1GB de memória, está demonstrado na Figura 11.

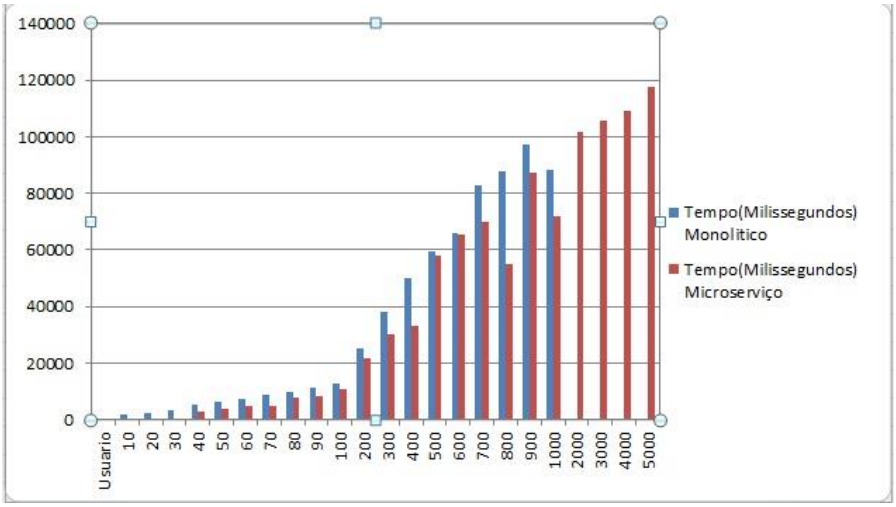
Figura 11-Comparativo de Tempo de Requisição entre as maquinas de 1GB



FONTE: OS AUTORES

As comparações realizadas utilizando a terceira maquina que possui 8GB de memória, está demonstrado na figura Figura 12.

Figura 12-Comparativo de Tempo de Requisição entre as maquinas de 8GB



FONTE: OS AUTORES

Os padrões arquiteturais foram comparados conforme os atributos abordados neste trabalho, suas características foram avaliadas através de notas

classificadas de 0 a 5 conforme mostra na Tabela 7, onde 0 não atende o conceito e 5 o atende completamente.

Tabela 7-Comparativo dos Padrões Arquiteturais

Atributo	Monolítico	Microserviço
Escalabilidade	2	5
Interoperabilidade	2	5
Perfomace	3	4
Modificabilidade	4	3
Total	**11**	**17**

FONTE: OS AUTORES

A arquitetura monolítica chegou a pontuação de 11 pontos enquanto a microserviço um total de 17 pontos. Com base na tabela comparativa, a arquitetura microserviço utiliza melhor os atributos avaliados na pesquisa, sendo uma arquitetura valida para ser utilizada no cenário proposto.

9. Resultado comparativo

Neste capitulo será discutido as notas dada as arquiteturas tanto a monolítica quanta a microserviço sendo embasada no método ATAM, de acordo com o método ATAM os principais pontos a ser avaliado em uma arquitetura são: escalabilidade, interoperabilidade, performance e modificabilidade.

Escalabilidade

Ambos padrões possuem a capacidade do escalonamento porém o monolítico limita-se apenas vertical por conta de sua limitação a nota 2 foi atribuida, enquanto o microserviço pode utilizar as duas técnicas de escalonamento, vertical e horizontal, a partir desse pretexto o microserviço tem maior flexibilidade para escalar seus serviços, por conta de sua flexibilidade o microserviço ganhou nota 5 em escalabilidade.

Interoperabilidade

Ambos padrões possibilitam a integração com outros sistemas e serviços, porém o microserviço é baseado e pensado, de forma que a interoperabilidade seja feita de forma natural, em questão desse conceito o microserviço ganhou nota 5 em interoperabilidade. O monolítico possui dificuldade em realizar integrações com outros serviços, pois há uma maior complexidade em sua implementação, com base nessa afirmação a arquitetura monolítica recebeu nota 2 em seu conceito.

Performace

O atributo perfomace é dividido em dois conceitos distintos, sendo um a latência e outro o gerenciamento de recursos utilizados. A latência das requisições do monolítico foram superiores em relação ao microserviço, contudo o microserviço leva muita vantagem em relação ao gerenciamento de recursos pois é observável que a ultima maquina de 8GB de memoria, na arquitetura monolítica chegou ao seu máximo com 1000 usuários simultâneos enquanto a microserviço utilizando a mesma maquina necessitou de 5000 usuários para chegar ao seu limite. Por conta dos conceitos analisados a arquitetura Monolítica recebeu nota 3 e a Microserviço nota 4.

Modificabilidade

O padrão microserviço possui um baixo acoplamento entre suas funcionalidades isso implica que o código fonte possui baixa dependência entre os módulos da aplicação, pois esse é um dos objetivos do seu padrão arquitetural, porém a um número maior em relação a linhas de código e replicação de código, pois cada serviço possui seus próprios arquivos de configuração. Contudo por conta de um baixo acoplamento, desenvolver novas funcionalidades utilizando esse padrão não causam impacto em já implementadas, pois as estruturas são construídas de formas independentes, com base nos pontos levantados o microserviço recebeu nota 3 em modificabilidade.

O padrão monolítico possui um alto acoplamento entre suas funcionalidades, porém dispõem de um numero menor de linhas de código quando comparado ao

microserviço. A duplicação do código também é baixa, pois todos os arquivos de configuração da aplicação encontram-se na mesma estrutura, com base nos conceitos abordados o monolítico recebe nota 4 no atributo modificalidade.

10. Considerações Finais

A pesquisa resultou em um comparativo entre a arquitetura monolítica e microserviço, apontando os pontos positivos e negativos de cada padrão delimitado no escopo da pesquisa. Foram implementadas duas soluções de software para o mesmo cenário, uma utilizando a abordagem monolítica e outra microserviço, a fim de criar uma comparação entre seus atributos.

Após o desenvolvimento das soluções, teve inicio a fase de avaliação, onde as duas arquiteturas foram avaliadas utilizando o método ATAM. O escopo dos conceitos avaliados limitou-se a quatro: Perfomace, Modificabilidade, Interoperabilidade e Escalonamento.

Infere-se que a decisão em um determinado padrão arquitetural, é delimitada pelo cenário. A escolha do padrão Monolitico implica em uma quantidade de código menor, menor duplicação de código fonte, contudo ocasiona em um maior acoplamento, menor portabilidade em relação a suas funcionalidades e uma menor flexibilidade no escalonamento. Se o cenário que o software deseja contemplar não necessita de uma grande estrutura, não necessita de mudanças constantes em suas funcionalidades e não necessita de uma flexibilidade no escalonamento, o padrão arquitetural monolítico é o melhor para abordar esse cenário.

A escolha do padrão Microserviço implica em uma maior flexibilidade no escalonamento, menor acoplamento entre suas funcionalidades, maior portabilidade, melhor gerenciamento dos recursos do servidor, porém ocasiona em uma maior complexidade nos níveis iniciais do projeto, quantidade maior de linhas de código escritas, maior duplicação de linhas de código, maior custo de implantação em níveis iniciais. Se o cenário que software deseja contemplar não se preocupa com custo inicial de implantação, maior quantidade de código, alta duplicação de código, o padrão microserviço é a escolha certa para abordar esse cenário.

Conclui-se que os padrões arquiteturais devem ser escolhidos de acordo com os atributos que melhor se adequarem ao cenário da solução proposta. Os atributos arquiteturais devem ser levantados de forma a apresentar quais as vantagens e desvantagens de cada arquitetura, para compreender qual trará mais vantagens a curto e longo prazo.

Referencias Bibliográficas

BASS, Len; CLEMENTS, Paul C.; KAZMAN, Rick. Software Architecture in Practice. Boston: Addison-Wesley, 1997.

BASS, Len; CLEMENTS, Paul C.; KAZMAN, Rick. Software Architecture in Practice. 3.rd. Boston: Addison-Wesley, 2012.

BASS, Len; WEBER, Ingo; ZHU, Liming. DevOps: A Software Architect's Perspective. Boston: Addison-Wesley, 2015.

BUENO, S. F. C; CAMPELO, B. G. Qualidade de Software, Universidade Federal de Pernambuco, 2012, 4 p.

CLEMENTS, Paul; BACHMANN, Felix; BASS, Len. Documenting Software Architectures: Views and Beyond. 2.ed. Boston: Addison-Wesley, 2002.

DMITRY, Savchenko; GLEB, Radchenko; OSSI, Taipale. Microservices validation: Mjolnirr platform case study. Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on, p.235-240, 25-29 May 2015.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. Design Patterns - Elements of Reusable Object-Oriented Software. Reading-MA, AddisonWesley, 1995.

HARPER, K. Eric; GOOIJER, Thijmen de; SCHMITT, Johannes O.; COX, David. Microdatabases for the Industrial Internet. Cornell University Library, 2016.

HEESCH, Uwe van; ELORANTA, Veli-Pekka; AVGERIOU, Paris; KOSKIMIES Kai; HARRISON, Nell. Decision-Centric Architecture Reviews, IEEE Software, v.31, Issue: 1, p.69-76, jan/fev. 2014.

LEWIS, James; FOWLER, Martin. Microservices. Martin Fowler, 2014. Disponível em: <http://martinfowler.com/articles/microservices.html>. Acesso em: 2 maio. 2017.

MALAVALLI, Divyanand; SATHAPPAN, Sathappan. Scalable microservice based architecture for enabling DMTF profiles. Network and Service Management (CNSM), 2015 11th International Conference on, p.428-432, 9-13 Nov. 2015.

NEWMAN, Sam. *Building Microservices*. Beijing: O'Reilly, 2015.

SOMMERVILLE, I. Software engineering. Boston: Pearson, 2011. ISBN 978-0137035151

SZWED, Piotr; SKRZYNSKI, Pawel; ROGUS, Grzegorz; WEREWKA, Jan. Ontology of architectural decisions supporting ATAM based assessment of SOA architectures.

Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on, p.287-290, 8-11 Sept. 2013.

VICENT, Paul; NATIS, Yefim; LIJIMA, Kimihiko; THOMAS, Anne; DUNIE, Rob; DRIVER, Mark. Magic Quadrant for Enterprise Application Platform as a Service, Worldwide. Gartner, 2016. Disponível em: <https://www.gartner.com/doc/reprints?id=1-2C8JHBP&ct=150325&st=sb>. Acesso em: 10 maio. 2017.

VERSTEDEN, Aad; PAUWELS, Erika; PAPANTONIOU, Agis. An Ecosystem of User-facing Microservices supported by Semantic Models. The 5th International USEWOD Workshop: Using the Web in the Age of Data, May 31st, 2015, Portoroz, Slovenia, 2015.

VILLARREAL, Sergio David; VILLASANA, Guillermo; LAVARIEGA, Juan Carlos. Proceedings of the International Conference on Software Engineering Research and Practice (SERP): 1-7. Athens: Athens The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). (2013)