UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ANA BEATRIZ SOUZA HAMAD

ADAPTAÇÃO OPERACIONAL DE UMA APLICAÇÃO ARQUITETADA EM MICROSSERVIÇOS E OS IMPACTOS EM TEMPO DE RESPOSTA

CAMPINA GRANDE - PB

2024

ANA BEATRIZ SOUZA HAMAD

ADAPTAÇÃO OPERACIONAL DE UMA APLICAÇÃO ARQUITETADA EM MICROSSERVIÇOS E OS IMPACTOS EM TEMPO DE RESPOSTA

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador : Reinaldo Cézar de Moraes Gomes

CAMPINA GRANDE - PB

2024

ANA BEATRIZ SOUZA HAMAD

ADAPTAÇÃO OPERACIONAL DE UMA APLICAÇÃO ARQUITETADA EM MICROSSERVIÇOS E OS IMPACTOS EM TEMPO DE RESPOSTA

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Reinaldo Cézar de Moraes Gomes

Orientador – UASC/CEEI/UFCG

Andrey Elísio Monteiro Brito

Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 17 de Maio de 2024.

CAMPINA GRANDE - PB

RESUMO

A hospedagem de uma aplicação em um ambiente centralizado pode oferecer vantagens em termos de simplificação de arquitetura, baixo tempo de resposta em condições normais, controle da infraestrutura, mitigação de complexidades operacionais e gerenciamento, porém quando se trata de um ambiente de produção, tal solução pode não ser viável a médio e longo prazo por ser um ambiente limitado, instável, com poucas janelas de otimização e expansão do serviço.

Uma das soluções possíveis de evolução em termos de infraestrutura é a distribuição do sistema em cloud pública, no caso desse estudo em nós AWS, com microsserviços orquestrados com Kubernetes, de forma a aproveitar as garantias operacionais de ambas as plataformas para mitigar boa parte do leque de falhas que um ambiente centralizado oferece, a custo de relativa perda de desempenho em relação a tempo de resposta.

Esse estudo de caso tem como objetivo elencar os principais problemas que existem hoje na implementação do Sênior Saúde Móvel, uma plataforma de prontuário digital arquitetada em microsserviços e hospedado de forma centralizada no data center do laboratório NUTES-UEPB. Ao final pretende-se adaptar e implementar a mesma aplicação em infraestrutura distribuída, estudar o tradeoff relativo ao tempo de resposta, e por fim sumarizar soluções, melhorias e trabalhos futuros referentes à avaliação dos resultados observados.

Podemos concluir que apesar da diferença de desempenho de tempo de resposta, as garantias providas pela solução distribuída, o potencial de expansão, manutenção do sistema, escalabilidade e o nível aceitável de tempo de resposta com prospectos de otimização justificam positivamente a adoção da adaptação para um futuro ambiente de produção.

OPERATIONAL ADAPTATION OF AN APPLICATION ARCHITECTED IN MICROSERVICES AND THE IMPACTS ON RESPONSE TIME

ABSTRACT

Hosting an application in a centralized environment can offer advantages in terms of architectural simplification, low response time under normal conditions, infrastructure control, mitigation of operational complexities and management, but when it comes to a production environment, such a solution It may not be viable in the medium and long term as it is a limited, unstable environment, with few opportunities for optimization and expansion of the service.

One of the possible solutions for evolution in terms of infrastructure is the distribution of the system in a public cloud, in the case of this study on AWS nodes, with microservices orchestrated with Kubernetes, in order to take advantage of the operational guarantees of both platforms to mitigate a large part of the range of failures that a centralized environment offers, at the cost of relative loss of performance in relation to response time.

This case study aims to list the main problems that exist today in the implementation of Sênior Saúde Móvel, a digital medical record platform architected in microservices and hosted centrally in the data center of NUTES-UEPB laboratory. In the end, we intend to adapt and implement the same application in a distributed infrastructure, study the tradeoff regarding response time, and finally summarize solutions, improvements and future work regarding the evaluation of the observed results.

We can conclude that despite the difference in response time performance, the guarantees provided by the distributed solution, the potential for expansion, system maintenance, scalability and the acceptable level of response time with optimization prospects positively justify the adoption of adaptation to a future production environment.

Adaptação operacional de uma aplicação arquitetada em microsserviços e os impactos em tempo de resposta

Ana Beatriz S. Hamad
ana.hamad@ccc.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

Reinaldo Gomes
Orientador
reinaldo@computacao.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

RESUMO

A hospedagem de uma aplicação em um ambiente centralizado pode oferecer vantagens em termos de simplificação de arquitetura, baixo tempo de resposta em condições normais, controle da infraestrutura, mitigação de complexidades operacionais e gerenciamento, porém quando se trata de um ambiente de produção, tal solução pode não ser viável à médio e longo prazo por ser um ambiente limitado, instável, com poucas janelas de otimização e expansão do serviço.

Uma das soluções possíveis de evolução em termos de infraestrutura é a distribuição do sistema em cloud pública, no caso desse estudo em nós AWS, com microsserviços orquestrados com Kubernetes, de forma a aproveitar as garantias operacionais de ambas as plataformas para mitigar boa parte do leque de falhas que um ambiente centralizado oferece, a custo de relativa perda de desempenho em relação a tempo de resposta.

Esse estudo de caso tem como objetivo elencar os principais problemas que existem hoje na implementação do Sênior Saúde Móvel, uma plataforma de prontuário digital arquitetada em microsserviços e hospedado de forma centralizada no data center do laboratório NUTES-UEPB. Ao final pretende-se adaptar e implementar a mesma aplicação em infraestrutura distribuída, estudar o tradeoff relativo ao tempo de resposta, e por fim sumarizar soluções, melhorias e trabalhos futuros referentes à avaliação dos resultados observados.

Podemos concluir que apesar da diferença de desempenho de tempo de resposta, as garantias providas pela solução distribuída, o potencial de expansão, manutenção do sistema, escalabilidade e o nível aceitável de tempo de resposta com prospectos de otimização justificam positivamente a adoção da adaptação para um futuro ambiente de produção.

1 INTRODUÇÃO

A arquitetura em microsserviços é um modelo de distribuição a nível de software que se caracteriza pela composição de vários módulos de serviços independentes que se comunicam entre si e formam uma aplicação completa. Essa composição é transparente para o usuário [7], e tem como principais objetivos a flexibilização do desenvolvimento, replicação, expansão das funcionalidades e manutenção dos serviços.

Essa arquitetura não está atrelada necessariamente à infraestrutura, sendo flexível o suficiente para ser implantada em servidores distribuídos mantendo sua transparência. Adaptar uma aplicação para executar em infraestrutura distribuída vem embarcado de complexidades que são mitigados pela instanciação centralizada, como por exemplo a comunicação entre os serviços, desafios de consistência, latência e gerenciamento. Tais complexidades por outro lado podem ser extremamente benéficas a médio e longo prazo, considerando que dependendo da estratégia de distribuição, são implementadas garantias de escalabilidade, automatização de rotinas de recuperação, tolerância à falhas e expansão da infraestrutura visando a saúde e o crescimento da aplicação.

O caso de estudo se refere ao Sênior Saúde Móvel (SSM), uma plataforma de prontuário digital arquitetado em microsserviços, originalmente hospedado centralizado em um servidor, apresentando algumas deficiências na implementação e fortes dependências operacionais, que contribuem especialmente na disponibilidade do serviço.

A ausência de um histórico de monitoramento e de rotinas de recuperação na implementação do SSM, limita as naturezas de avaliação dos impactos da distribuição em termos da infraestrutura, o mesmo serve para a implementação migrada em cloud pública, como ainda está em processo de desenvolvimento e não há histórico de dados reais do sistema, este trabalho se deterá a avaliar em termos de tempo de resposta da aplicação em ambas as soluções, e o paralelo entre os prós e os contras das disposições.

Existem problemas sólidos e inerentes ao sistema centralizado que serão aqui elencados, porém não avaliados, dada sua natureza e a ausência de um cenário equivalente com a solução em cloud pública, no caso a AWS. Apesar da não avaliação, os problemas são deveras relevantes e fazem parte ativamente do dia a dia da manutenção do sistema, bem como justificam os esforços da migração e baseiam os trabalhos futuros de aprimoramento do sistema.

1.1 Descrição do Estudo de Caso

O Sênior Saúde Móvel é uma plataforma de prontuário digital combinado com o monitoramento de pacientes em tempo real, através de coleta de dados via smartwatch Fitbit. Como o nome já indica, a estrutura original da aplicação é voltada para o acompanhamento de saúde de idosos, porém sua arquitetura permite a adaptação para outras finalidades, como monitoramento de pacientes em um hospital, e a mais recente adaptação para acompanhamento da saúde de motoristas de caminhão.

A plataforma é arquitetada em microsserviços conteinerizados em Docker e executados com Docker Compose, sendo um dos

TCC 2023.2, UFCG, 2024
© 2024 Ciência da Computação, Universidade Federal de Campina Grande.

serviços uma integração com API Fitbit, que hospeda os dados coletados dos relógios em seus servidores, a aplicação consome os dados e alimenta seu próprio banco de dados para disponibilização e sincronização das informações dos pacientes. Dispõe bancos de dados Percona MongoDB, RabbitMQ como barramento de mensagens entre os serviços e Redis para cache e fila de requisições para o servidor Fitbit.

A plataforma está hospedada em uma VM no data center do NUTES-UEPB, operando de forma centralizada, com banco de dados local com políticas de backup semanal, e a comunicação entre os serviços é feita pelo localhost.

2 PROBLEMAS

Os problemas que cercam o Sênior Saúde Móvel (SSM) não são apenas do serviço e sua construção, mas especialmente das limitações da sua infraestrutura que impactam diretamente o serviço. As considerações iniciais para o estudo da adoção da cloud pública para hospedar o SSM decorre da ausência de garantias essenciais vistas para uma plataforma da sua natureza. Apesar de a aplicação dispor de uma implementação distribuída em microsserviços sólidos, independentes e transparentes, o benefício da distribuição passa a ser muito mais um facilitador do processo de desenvolvimento do que uma vantagem operacional no cenário da atual implementação, com limitações de recursos e sem garantias de disponibilidade, confiabilidade, redundância e escalabilidade.

Ao considerar uma arquitetura distribuída para uma aplicação, são analisadas as necessidades de distribuição frente ao desenvolvimento, a manutenibilidade dos serviços e a disposição operacional do sistema, para assegurar o funcionamento satisfatório do serviço com o mínimo de intercorrências tanto frente às demandas internas da aplicação quanto à experiência do usuário com o sistema.

Centralizar a aplicação em um único host pode ser vantajoso ao passo que mitiga complexidades de comunicação e otimização do tráfego de mensagens entre os microsserviços, plena consistência dos dados e maior controle sobre a infraestrutura, simplificando a implementação de monitoramento e segurança. Porém, à médio e longo prazo se torna laboriosa a manutenção e o crescimento da aplicação, pois o meio se torna limitado em termos de recursos físicos, além do risco da contração de gargalo de rede devido ao crescimento da aplicação e maior fluxo de dados transitando, e o risco de indisponibilidade em caso de desastre não planejado no hospedeiro, que pode comprometer o serviço por tempo indeterminado.

2.1 Problemas Inerentes do Ambiente Centralizado

2.1.1 Escalabilidade

Pode-se dizer que a escalabilidade é o problema central quando se trata de infraestrutura centralizada. O fato de que há claras limitações especialmente físicas de para onde o sistema irá expandir em caso de crescimento da demanda do serviço, a concentração de dados e o gargalo de rede em atender todas as requisições dos usuários, faz com que seja um modelo impraticável de oferta de serviços, salvo aplicações internas da empresa, com demanda previsível, número de usuários limitados e infraestrutura dedicada para tal.

No estudo de caso do Sênior Saúde Móvel (SSM), uma aplicação em produção que em condições normais dispõe de certa previsibilidade quanto a quantidade de usuários mas não necessariamente de acessos. Por um lado pode ser vantajoso, em termos de custo, executar em um host dedicado que atende à esta demanda, porém no caso de expansão das funcionalidades, número de pacientes e/ou novas métricas sobre os pacientes, haverá sofrimento por parte da infraestrutura para lidar com a alta do volume de processamentos, maior latência devido ao tráfego de mensagens entre os componentes da aplicação e piora na entrega do serviço. Como o host referido se trata de uma máquina virtual em um servidor compartilhado com outras máquinas virtuais, o trabalho de expansão dos recursos físicos pode ser laborioso, e no pior caso, comprometer os dados da aplicação.

Uma das maneiras de escalar o serviço é replicando a instância e distribuindo as requisições entre elas, que desafoga o gargalo das requisições, porém vem com o problema da sincronização dos dados entre as instâncias, que mais uma vez não é garantido na implementação original. Soluções de escalabilidade são essencialmente distribuídos, uma aplicação construída sobre infraestrutura e ferramentais centralizados não são escaláveis sem esforço de sincronização e consistência entre as partes.

2.1.2 Disponibilidade

Hospedar sistemas em servidores próprios há um risco atrelado da manutenção dos mesmos ativos e acessíveis, e que isso nem sempre está nas mãos do administrador do data center, como em caso de quedas de energia e da rede ser atrelada à uma organização superior que delimita o tráfego externo e as regras de acesso de onde o serviço está hospedado. Em caso de desastre, o sistema ficará incontestavelmente fora do ar, o que é um problema grave no estudo de caso aqui apresentado.

O monitoramento de pacientes internados em hospitais por exemplo, requer disponibilidade dos dados 24/7, e em caso de desastre que pode levar tempo indeterminado para ser solucionado, pode trazer sérios problemas não só para reparar o sistema quanto no relacionamento com o cliente.

No estudo de caso deste trabalho, o SSM é hospedado em um data center de pequeno porte que faz parte do Núcleo de Tecnologias Estratégicas em Saúde (NUTES), laboratório que faz parte do Centro de Ciências Biológicas e Saúde da Universidade Estadual da Paraíba. O data center atualmente não dispõe de gerador de energia elétrica dedicado e sua rede tem acesso atrelado à organização da universidade. Os desastres dessa natureza são totalmente imprevisíveis e incapacitantes quando ocorrem.

2.1.3 Tolerância a Falhas

Dado todos os panoramas anteriores, pode-se aferir que a tolerância a falhas é mínima da forma que o sistema centralizado está concebido. O marco zero do estudo da tolerância a falhas para uma aplicação distribuída é assegurar o pleno funcionamento da mesma em um ambiente centralizado, mapeando os cenários de falha interna do serviço para então implementar os mecanismos de tolerância para um cenário distribuído.

Em termos operacionais, a tolerância a falhas deve ser implementada de forma transparente à aplicação, com mecanismos

de autoscaling e self-healing, ou seja, deve-se implementar meios em que dada uma falha operacional ocorra, ele próprio atue para manter o sistema disponível.

No caso do sistema centralizado do SSM, o autoscaling é inexistente por motivos já discutidos em tópicos anteriores, mas mais importante ainda, o self-healing também não se aplica. O deploy da aplicação é executado em Docker Compose, uma ferramenta de conteinerização que permite a execução de mais de um contêiner ao mesmo tempo, uma forma prática de instanciar uma aplicação, mas que não é indicada para aplicações em produção pois não garante liveness nem self-healing, ou seja, caso um dos contêineres falhe, não há mecanismos de recuperação automática do mesmo, comprometendo a aplicação até que seja resolvido manualmente.

2.2 Problemas Gerais

2.2.1 Tempo de Resposta

O tempo de resposta se refere ao tempo que o servidor leva para atender a uma requisição do usuário, que pode ser influenciado pela infraestrutura hospedeira e pela arquitetura da implementação. Ao distribuir uma aplicação em microsserviços, considera-se que a comunicação e o processamento de operações serão realizadas por múltiplas entidades coordenadas, e quanto mais distribuída, especialmente em termos de infraestrutura, maior o impacto na latência e tempo de resposta da aplicação.

Em ambientes centralizados, a influência do tempo de resposta parte do tráfego do volume de requisições, limitações de recursos do host relativos aos processos, políticas de acesso e capacidade da rede, porém se beneficia da localidade geográfica para mitigar impactos visíveis de latência.

Em um sistema hospedado em cloud pública sofre com maior janela de tempo de resposta, uma vez que em via de regra não há controle e conhecimento onde estão localizados os recursos computacionais que executam a aplicação, que podem estar próximos e demandando menos esforços de comunicação, quanto podem estar distantes, influenciando mais notavelmente, e essa métrica não é estática, caso não seja o cenário de aluguel de hosts dedicados, haverá migrações de recursos dentro das próprias zonas de disponibilidade [1] de forma totalmente transparente.

A latência no ambiente de cloud pública também é inevitável, no caso do objeto de estudo do SSM, por razões de redução de custos durante o desenvolvimento da arquitetura base da infraestrutura, é temporariamente hospedado na zona AWS de Ohio, nos Estados Unidos, o impacto tanto em tempo de resposta quanto em latência é visível, porém dentro do considerado aceitável em termos de qualidade. Vale enfatizar que no momento não há dados reais de pacientes hospedados nessa infraestrutura.

Alguns meios de minimizar esses impactos são arquitetar a aplicação de forma a reduzir o tráfego interno, tempo de transmissão e uso de recursos para responder a uma requisição, além de alocação adequada de recursos dos nós para o tipo de processamento que o mesmo hospeda, que é possível de ser contratado em cloud pública, porém limitado em cenário centralizado.

2.2.2 Desempenho

Uma vez que toda a aplicação está concentrada em um só nó, a depender do volume de requisições simultâneas, pode gerar um gargalo e sobrecarga de recursos, comprometendo o sistema. Sem mecanismos de replicação e distribuição de requisições, o limite de processamento e resposta decresce inevitavelmente. Na aplicação SSM, o deployment em Docker Compose não dispõe de mecanismos de autoscaling horizontal, diferentemente do Kubernetes [2], que provê API específica para esta finalidade, impraticável para ambiente centralizado.

O autoscaling horizontal regula automaticamente o número de réplicas de um objeto a partir de métricas pré estabelecidas de desempenho desejáveis, afim de minimizar os impactos de uma carga imprevista acontecer e comprometer a aplicação. Métricas de uso de recursos, tráfego de rede ou métricas customizadas externas ao Kubernetes podem ser declaradas e combinadas à implementação dos objetos, e gerenciadas automaticamente pelo controller.

Mesmo com autoscaling horizontal, a sobrecarga do host que agrega os pods também pode ocorrer, necessitando assim de políticas de autoscaling vertical. O autoscaling vertical é a replicação de um nó de forma integral no cluster Kubernetes, que atingido um dado limiar de recurso, é instanciado uma réplica do nó em questão. No momento, autoscaling de nenhuma natureza está implementado na solução da AWS, visto que o cluster ainda se encontra em fase de desenvolvimento, porém está previsto em próximos passos.

3 OBJETIVOS

O estudo têm como objetivo elencar os principais problemas existentes na implantação da plataforma Sênior Saúde Móvel, afim de embasar trabalhos futuros de evolução do serviço. Como meio de adaptação, foi implementada uma versão da plataforma em cloud AWS, distribuída em um cluster Kubernetes com control plane único e múltiplos workers, escalando serviços e banco de dados em nós distintos, como implementação base para futuras melhorias e complexidades.

Como validação da estrutura base, serão executadas avaliações da plataforma [9] do SSM em termos de tempo de resposta [5] nos dois cenários de infraestrutura distintos, um ambiente de desenvolvimento que segue a estrutura do sistema centralizado, e outra versão distribuída na AWS. Nesse cenário, o tempo de resposta e latência provê meios de avaliar o impacto da adaptação da distribuição dos serviços em múltiplos nós [6]. A consistência e a média dos tempos de resposta dentro dos níveis de qualidade são as características consideradas favoráveis à adaptação, mesmo frente à discrepância do cenário centralizado, que apesar de tudo indicar que terá melhores resultados numéricos, se trata de um cenário impraticável em produção. Vale salientar que o ambiente centralizado de desenvolvimento segue os mesmos padrões e implementação do ambiente de produção hoje vigente.

4 A ARQUITETURA

4.1 Ambiente Centralizado

Figura 1: Arquitetura dos microsserviços no ambiente centralizado

4.2 Ambiente Distribuído em Cloud Pública - AWS

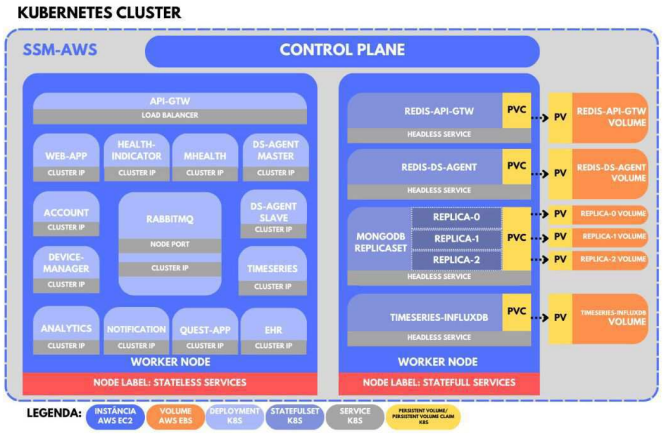


Figura 2: Arquitetura dos microsserviços no ambiente clusterizado na AWS

5 EXPERIMENTAÇÃO

A avaliação de tempo de resposta consistirá de quatro fases: 10 threads de 10 requisições cada, 10 threads de 100 requisições, 100 threads de 10 requisições, e por fim 100 threads de 100 requisições. Os testes serão conduzidos pela ferramenta JMeter [3], e serão avaliados métricas de tempo de resposta em decorrer do tempo, percentil de tempo de resposta, média de tempo de resposta em relação ao número de threads ativas, transações por segundo, e latência em tempo decorrido [8]. O experimento será aplicado em dois cenários: ambiente centralizado da aplicação, ambiente clusterizado na AWS sem autoscaling horizontal de pods, mantendo justo o comparativo em termos da quantidade de recursos para ambas as arquiteturas da aplicação.

Para os testes foi utilizado a plataforma Apache JMeter, que provê testes de carga, estresse e análise de resultados a partir dos resultados das requisições [4].

6 RESULTADOS

6.1 Sumário Geral

A seguir, apresenta-se a tabela de resultados gerados a partir da plataforma JMeter. Um dos destaques a ser discutidos trata-se do percentil do tempo de resposta às requisições, que trata-se de uma métrica que ordena de forma crescente os resultados de tempo de resposta, e seu resultado equivale ao valor máximo para o intervalo de x% de amostras, por exemplo, o resultado do 90 Percentil é o valor máximo de tempo de resposta de um intervalo dos primeiros 90% das requisições. As demais métricas são autoexplicativas.

Sumário de Resultados Agregados - SSM Centralizado				
Métricas	10thr. 10reqs.	10thr. 100reqs.	100thr. 10reqs.	100thr. 100reqs.
Média Tempo de Resposta	44ms	35ms	99ms	60ms
Mediana Tempo de Resposta	26ms	24ms	48ms	50ms
90 Percentil	83ms	62ms	314ms	87ms
95 Percentil	85ms	64ms	529ms	107ms
99 Percentil	143ms	89ms	575ms	421ms
Mínimo Tempo de Resposta	16ms	16ms	16ms	15ms
Máximo Tempo de Resposta	151ms	198ms	612ms	776ms
Vazão Média	39,8 req/s	46,2 req/s	179,0 req/s	297,2 req/s

Tabela 1: Resultados referentes à execução do teste de desempenho no Sênior Saúde Móvel em ambiente centralizado

Sumário de Resultados Agregados - SSM AWS				
Métricas	10thr. 10reqs.	10thr. 100reqs.	100thr. 10reqs.	100thr. 100reqs.
Média Tempo de Resposta	209ms	176ms	318ms	258ms
Mediana Tempo de Resposta	171ms	170ms	274ms	233ms
90 Percentil	190ms	180ms	591ms	336ms
95 Percentil	649ms	187ms	788ms	373ms
99 Percentil	656ms	418ms	835ms	987ms
Mínimo Tempo de Resposta	125ms	123ms	127ms	126ms
Máximo Tempo de Resposta	658ms	1101ms	1066ms	1995ms
Throughput Médio	39,1 req/s	46,2 req/s	186,9 req/s	299,4 req/s

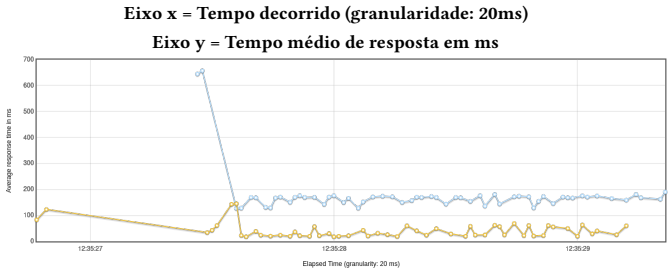
Tabela 2: Resultados referentes à execução do teste de desempenho no Sênior Saúde Móvel em ambiente clusterizado na AWS

6.2 Tempo Médio de Resposta x Tempo Decorrido

Os gráficos a seguir representam as observações do comportamento do tempo de resposta no decorrer do tempo de execução das requisições pela plataforma Apache JMeter, tal como a distribuição dos tempos de resposta às requisições, que tem como objetivo aqui ilustrar a distribuição das requisições com tempos anômalos e inferir a relevância de tais tempos no cenário observado.

6.2.1 10 Threads de 10 Requisições

Diante dos dados observados nos gráficos abaixo e os dados do sumário anterior, podemos concluir que apesar de a amostra da AWS apresentar resultados anômalos mais expressivos no início da execução do experimento, tais resultados representam menos de 5% da amostra. A média de tempo de resposta por sua vez está dentro dos parâmetros normais de qualidade e constância no decorrer do tempo, o esperado para o tamanho da amostragem.



SSM-Centralizado SSM-AWS

Figura 3: Tempo médio de resposta em relação ao tempo decorrido (10 threads, 10 requisições)

SSM-Centralizado SSM-AWS

Figura 4: Distribuição do tempo de resposta (10 threads, 10 requisições)

6.2.2 10 Threads de 100 Requisições

Em um cenário onde há poucos usuários e maior volume de requisições, então, é possível observar que o número de requisições anômalas é menor ainda, menos de 1% das amostras, e apresentando o melhor resultado em termos de constância dentre todos os cenários observados, sugerindo equivalência de desempenho entre as soluções, respeitando as proporções e realidades das infraestruturas.

SSM-Centralizado SSM-AWS

Figura 5: Tempo médio de resposta em relação ao tempo decorrido (10 threads, 100 requisições)

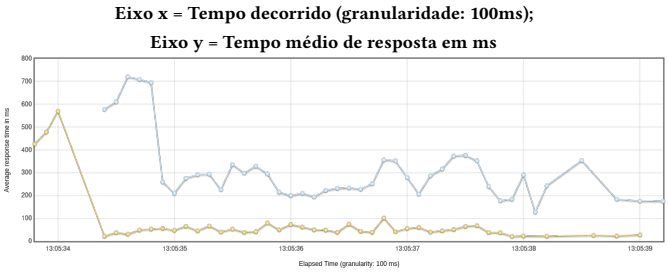
SSM-Centralizado SSM-AWS

Figura 6: Distribuição do tempo de resposta (10 threads, 100 requisições)

6.2.3 100 Threads de 10 Requisições

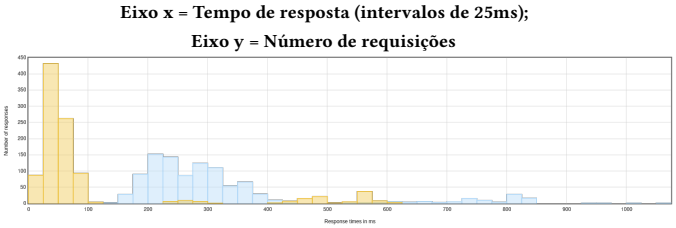
Neste cenário, que representa um grande volume de usuários fazendo um pequeno volume de requisições, podemos observar uma maior inconstância no tempo de resposta ao decorrer do tempo, elucidando a importância do autoscaling horizontal para manutenção da qualidade da entrega do serviço para uma carga considerável de usuários. Como dito em seções anteriores, o autoscaling horizontal

não está presente na implementação testada, e o resultado apresentado nesse cenário de experimentação é a base da justificativa de trabalhos futuros relacionados a isso. Por mais que os tempos médios sejam relativamente aceitáveis para tempo de resposta de uma aplicação web, os percentis apresentam os piores resultados dentre todos os cenários, reforçando a necessidade de otimização da implementação.



SSM-Centralizado SSM-AWS

Figura 7: Tempo médio de resposta em relação ao tempo decorrido (100 threads, 10 requisições)

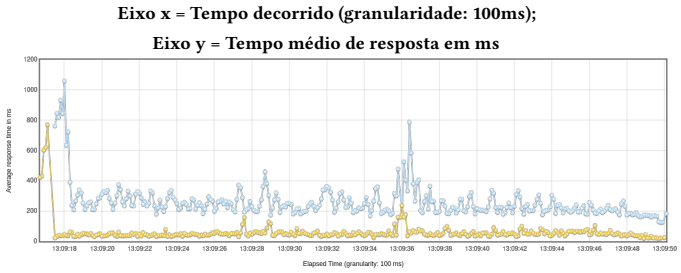


SSM-Centralizado SSM-AWS

Figura 8: Distribuição do tempo de resposta (100 threads, 10 requisições)

6.2.4 100 Threads de 100 Requisições

Como cenário mais substancial, com muitos usuários e muitas requisições por cada um, podemos observar um fenômeno similar ao segundo cenário, um comportamento similar entre ambas as soluções, com picos de tempo de resposta no mesmo intervalo de tempo, além de certa inconstância na solução na AWS, decorrente do citado anteriormente em relação ao autoscaling horizontal dos serviços e otimização. O resultado porém ainda é mais satisfatório que o experimento anterior, apresentando percentis satisfatórios e anomalias em menos de 5% da amostra e média de tempo de resposta também melhor, apesar da maior demanda.



SSM-Centralizado SSM-AWS

Figura 9: Tempo médio de resposta em relação ao tempo decorrido (100 threads, 100 requisições)

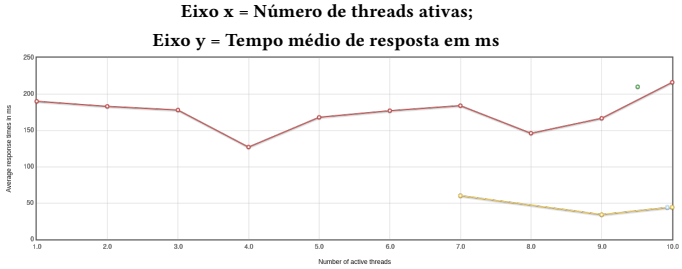
SSM-Centralizado SSM-AWS

Figura 10: Distribuição do tempo de resposta (100 threads, 100 requisições)

6.3 Threads Ativas x Tempo de Resposta

Os gráficos a seguir apresentam o comportamento da média do tempo de resposta em relação à quantidade de threads ativas em cada experimento. Podemos concluir que o número de threads ativas é o fator de influência principal na qualidade do tempo de resposta da plataforma.

6.3.1 10 Threads de 10 Requisições



SSM-Centralizado SSM-AWS

Figura 11: Tempo de Resposta em relação a threads ativas (10 threads, 10 requisições)

6.3.2 10 Threads de 100 Requisições

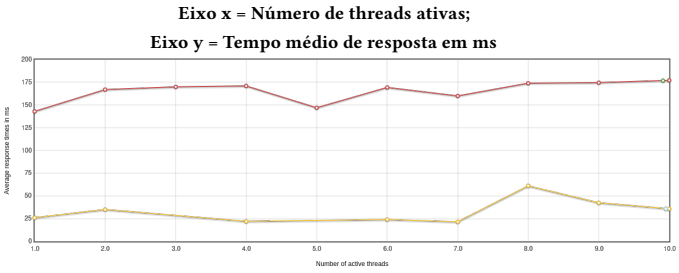


Figura 12: Tempo de Resposta em relação a threads ativas (10 threads, 100 requisições)

6.3.3 100 Threads de 10 Requisições

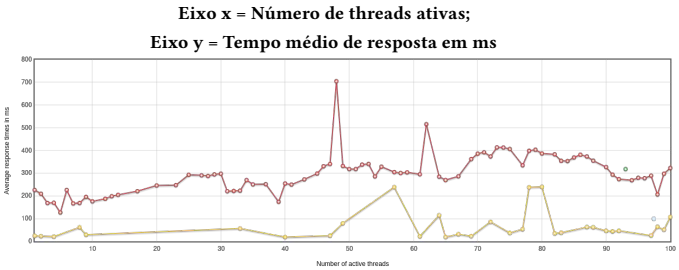


Figura 13: Tempo de Resposta em relação a threads ativas (100 threads, 10 requisições)

6.3.4 100 Threads de 100 Requisições

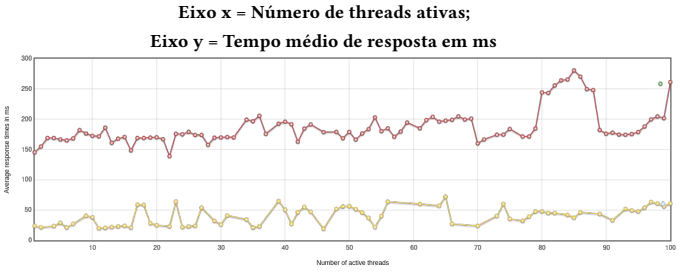


Figura 14: Tempo de Resposta em relação a threads ativas (100 threads, 100 requisições)

6.4 Transações por Segundo

Neste quesito, podemos observar nos gráficos a seguir um comportamento similar em ambas as implementações, reforçando a equivalência dos serviços.

6.4.1 10 Threads de 10 Requisições

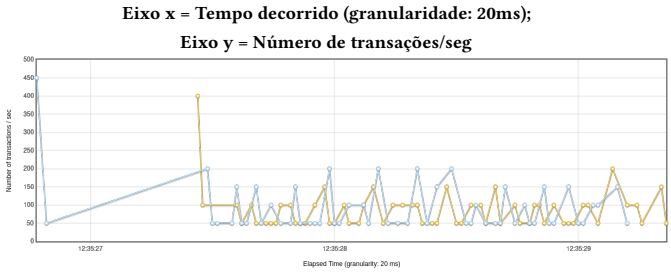


Figura 15: Transações por segundo (10 threads, 10 requisições)

6.4.2 10 Threads de 100 Requisições

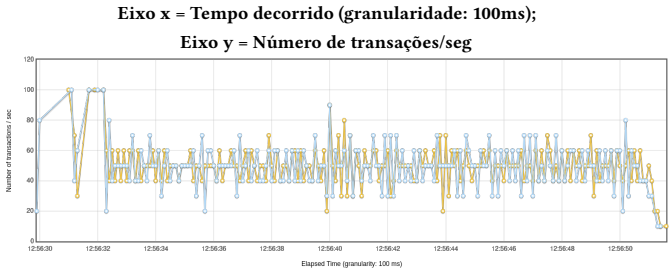


Figura 16: Transações por segundo (10 threads, 100 requisições)

6.4.3 100 Threads de 10 Requisições

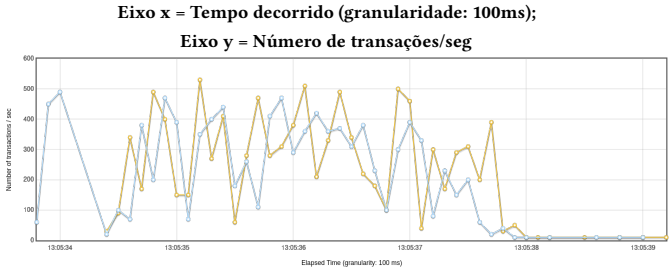
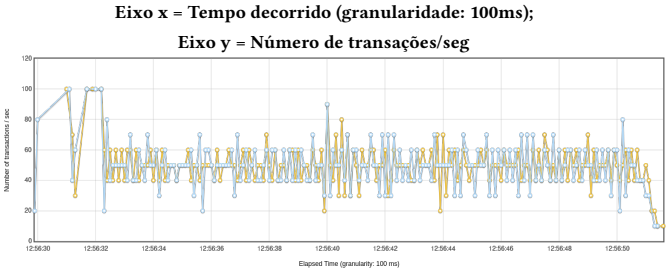


Figura 17: Transações por segundo (100 threads, 10 requisições)

6.4.4 100 Threads de 100 Requisições



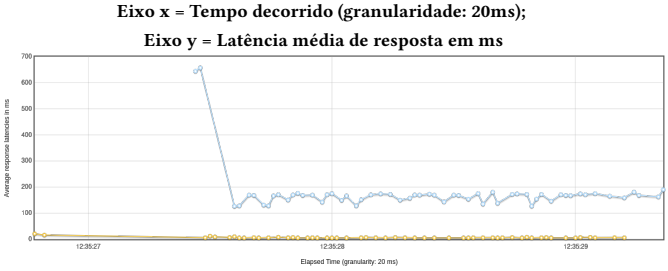
SSM-Centralizado SSM-AWS

Figura 18: Transações por segundo (100 threads, 100 requisições)

6.5 Latência x Tempo Decorrido

Como descrito na sessão de problemas, o impacto na latência média de resposta seria incontestavelmente presente na solução clusterizada na AWS, especialmente por razões geográficas como já discutido. Porém, pode-se observar com relevante clareza a inconstância da latência também relacionada ao número de usuários simultâneos, similar à justificativa apresentada para o tempo médio de resposta e a janela de otimização que existe a ser implementado.

6.5.1 10 Threads de 10 Requisições



SSM-Centralizado SSM-AWS

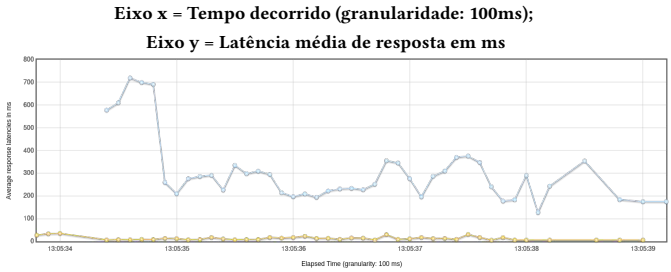
Figura 19: Latência de resposta por tempo decorrido (10 threads, 10 requisições)

6.5.2 10 Threads de 100 Requisições

SSM-Centralizado SSM-AWS

Figura 20: Latência por tempo decorrido (10 threads, 100 requisições)

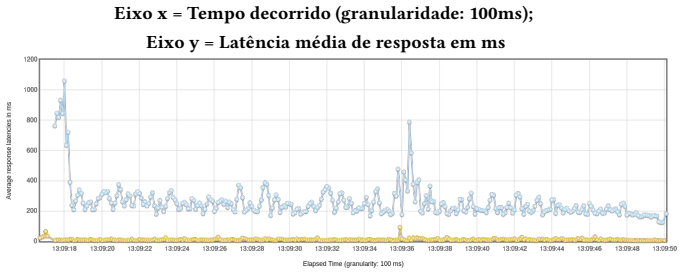
6.5.3 100 Threads de 10 Requisições



SSM-Centralizado SSM-AWS

Figura 21: Latência por tempo decorrido (100 threads, 10 requisições)

6.5.4 100 Threads de 100 Requisições



SSM-Centralizado SSM-AWS

Figura 22: Latência por tempo decorrido (100 threads, 100 requisições)

7 CONCLUSÕES

Diante do exposto, pode-se concluir que a natureza de problemas hoje existentes na implementação centralizada do Sênior Saúde Móvel não justifica o ganho em desempenho, quando se é possível emplacar um serviço de qualidade dentro das proporções em ambiente distribuído. Os prospectos de expansão, escalabilidade e manutenção do serviço provido na AWS e clusterizado com Kubernetes, mitiga problemas e abre portas para a evolução contínua da plataforma, além dos percalços enfrentados pela infraestrutura física vigente.

Os resultados satisfazem, apesar da solução clusterizada ainda estar em fase de desenvolvimento. Os testes aqui apresentados provêm um sumário de melhorias e caminhos a serem analisados nos próximos passos, como apontado na sessão 3.

Por mais que haja discrepância no tempo de resposta médio das implementações, os tempos observados são satisfatórios em ambos os ambientes em termos qualitativos de experiência do usuário.

8 TRABALHOS FUTUROS

Implementação de autoscaling horizontal para os microsserviços da aplicação, implementação de autoscaling vertical para os nodes do

cluster Kubernetes, ou seja, das máquinas virtuais AWS, implementação de políticas de backup e gerenciamento de lifecycle dos nodes, utilizar flavors otimizados para o tipo de serviço instanciado em cada worker node ao invés de flavors genéricos, implementação de pipeline CI/CD para fluxo de instanciação e atualização dos serviços, implementação de automação de configurações e troubleshooting da aplicação, e por fim aplicação de zero trust no ambiente AWS.

9 AGRADECIMENTOS

Nessa longa jornada que representa mais de 1/3 da minha vida (até este momento) dentro da UFCG, vivi das mais diversas histórias e também vi a história ser escrita em Computação, colecionei amigos (isso inclui os professores, sim!) que mudaram a minha vida, amigos que mudei a vida, e o orgulho que carrego na minha história é que estou graduando em Computação, mas eu aprendi muito mais que uma profissão, eu encontrei meu lugar no mundo e todos os envolvidos têm parte nisso, e não têm palavras suficientes no dicionário para descrever a minha gratidão por tudo e todos.

Quero agradecer a minha mãe por ter acreditado no meu potencial, por ter apostado na educação minha e da minha irmã e confiado em nós para correr atrás de um futuro diferente, esse diploma tem meu nome, mas tenho orgulho de dizer que é dela. Foram anos demais, muita paciência e fé envolvida, mas enfim a minha teimosia deu frutos. Agradecer também a minha irmã, que nunca duvidou de mim e sempre me deu apoio quando o desespero me batia.

Quero agradecer a minha psicóloga, dra. Karla Silveira Ribeiro, por todos os anos de acompanhamento, e por todo esse tempo ter esfregado na minha cara todo o meu potencial, nunca ter me permitido desistir, por ter me ensinado a vencer de mim mesma, e sem isso eu não teria chegado tão longe.

Agradecer também ao meu amado Gabriel, uma das primeiras pessoas que aprendi a admirar logo quando entrei em Computação, que me ensinou não só todo tipo de conhecimentos aleatórios de computação, como também xadrez, história geral, inglês, alemão, ler a tabela do brasileirão, distinguir pneus de Fórmula 1, comprar vinhos de qualidade por até 7,00 euros, e o melhor de todos, que é o significado de felicidade. Sem ele me fazendo rir enquanto eu chorava de desespero, sem ele brigando para eu estudar, sem ele incentivando o melhor de mim, eu não teria chegado a escrever esse TCC.

Agradecer ao meu querido orientador, prof. Reinaldo, que me orienta na vida desde antes mesmo da minha matrícula em Computação, que é o mais próximo de um pai com quem pude conviver todos esses anos, e com quem tive oportunidade de tomar lições acadêmicas e lições de vida em mesma proporção.

Agradecer a todos os professores fantásticos, tenho orgulho de dizer que Computação @ UFCG é Seleção. Um abraço especial em Fubica por todos os formulários de matrícula bem sucedidos, o senhor é o maior que nós temos.

E por fim, uma breve lista do Faustão dos meus amigos que me acompanharam nessa longa jornada de 8 anos em Computação: obrigada Kaio Kassiano, Matteus Silva, Fábio Silva e Iury Gregory por serem minha inspiração profissional, obrigada a Marcus Tenório por toda a orientação para encontrar meu caminho em Computação, obrigada a Larissa Braz e Pedro Yóssis por serem meu espelho (quando eu crescer quero ser como vocês!), obrigada a todos os meus meninos do Guardians por terem me incentivado a saber mais pra poder ensinar corretamente, obrigada a Edson, Rodrigo e Eduardo por todo auxílio e aprendizado, obrigada ao prof. Paulo Barbosa por me permitir conceber esse estudo e dar o primeiro passo para o futuro com esse documento, e obrigada a todos os meus muitos amigos que abrilhantaram a minha jornada nessa universidade. Todos aqui citados moram eternamente no meu coração.

Obrigada!

REFERÊNCIAS

[1] [n. d.]. AWS Documentation. https://docs.aws.amazon.com/
[2] 2020. Kubernetes Concepts Documentation. Retrieved June 22, 2020 from https://kubernetes.io/docs/concepts/
[3] Apache Community Code. [n. d.]. Apache JMeter User's Manual. https://jmeter.apache.org/usermanual/index.html
[4] Nisha Jha and Rashmi Popli. 2017. Comparative Analysis of Web Applications using JMeter. *International Journal of Advanced Research in Computer Science* 8, 3 (April 2017), 774–777. https://www.ijarcs.info/index.php/Ijarcs/article/view/3095/3078 ISSN No. 0976-5697.
[5] Ritika Kumari. 2024. Response Time Testing – What it is How to Measure it? Retrieved April 4, 2024 from https://testsigma.com/blog/response-time-testing/
[6] Junhui Fu Kunhua Zhu and Yancui Li. 2010. Research the performance testing and performance improvement strategy in web application. *International Conference on Education Technology and Computer (ICETC)* (June 2010), 328–332. https://doi.org/10.1109/ICETC.2010.5529374
[7] Andrew S. Tanenbaum Maarten van Steen. 2020. *Distributed Systems* (3rd. ed.). Martin van Steen, Pearson Education, Inc.
[8] Prince Sinha. 2022. Monitoring Application Response Times. Retrieved February 24, 2022 from https://scoutapm.com/blog/application-response-time-monitoring
[9] Indeed Editorial Team. 2023. 5 Types of Response Time Metrics and How To Measure It. Retrieved January 4, 2023 from https://www.indeed.com/career-advice/career-development/response-time-testing