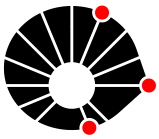

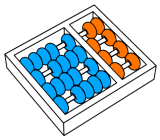

Sistemas Distribuídos - Aplicações de Micorsserviços e sua Infraestrutura

Gunter Mingato de Oliveira

Relatório Técnico - IC-PFG-20-20
Projeto Final de Graduação
2020 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Sistemas Distribuídos
Aplicações de Micorsserviços e sua Infraestrutura

Gunter Mingato de Oliveira *

Resumo

Foi feito um estudo para definir o que são microsserviços, para saber quando se aplicar, saber suas definições e seus pontos negativos. E com um ponto negativo relacionado ao tempo de comunicação entre os microsserviços foram feitas análises para poder criar microsserviços que possuam tempo de respostas adequados e segurança e autenticação de acordo com os requisitos do sistema. Algumas análises de tempo, stress e performance foram feitas para ver quais são as melhores configurações para poder atender um cliente de acordo com as suas exigências.

1 Introdução

Em tecnologia da informação as demandas por produtos vem aumentando muito nos últimos anos, bem como as exigências como ter um sistema com alta escalabilidade e disponibilidade. Antes os sistemas eram monolíticos onde toda as partes do BackEnd e FrontEnd das aplicações funcionavam em uma única máquina e em um único software, e como sistemas monolíticos não atendem às exigências, uma nova forma de desenvolver sistemas começou a surgir fazendo com que o sistema seja distribuído, ou seja, distribui todo o sistema em microsserviços: uma aplicação do sistema que possui o seu próprio BackEnd e FrontEnd, e que é possível ser executada independentemente do restante do sistema. Por outro lado, como as aplicações se comunicam através da rede de internet, surgem outros problemas: tempo de resposta de requisição entre as aplicações, e de segurança, pois os dados estarão trafegando pela internet, o que ocasiona uma maior vulnerabilidade dos dados.

Neste TCC será realizado um estudo mais detalhado de microsserviços e Middlewares para entender as melhores formas de utilizá-los, baseados nos tempos de respostas das requisições entre as aplicações (Microsserviços e middlewares). Para poder usar microsserviços em um sistema também foi necessário ter conhecimentos sobre programação, computação distribuída e redes de computadores em sistemas de gerência de recursos distribuídos. Os passos para implementação de um sistema de gerência de recursos envolvem a identificação de características e modelagem de um sistema computacional distribuído, a identificação de aplicações e suas características para serem aplicadas ao modelo, o desenvolvimento de políticas e algoritmos de alocação de recursos para otimização de critérios do Sistema e

*Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP. Trabalho de Conclusão de Curso: Engenharia de Computação

requisitos das aplicações. A partir da identificação e modelagem do sistema distribuído a ser considerado, um estudo sobre políticas de alocação de aplicações nesse tipo de sistema será desenvolvido. Em seguida, as necessidades de um sistema de gerência de recursos e/ou políticas de alocação de recursos que levem em conta a capacidade e desempenho computacional e os requisitos de aplicação modelados serão projetados e avaliados.

2 Conceitos Básicos

Microsserviço é uma pequena aplicação a qual pode ser colocada em produção, depurada, testada independentemente do resto do sistema, possuindo apenas uma função, e tendo que ser uma aplicação facilmente entendida. A função seria definida como uma função funcional ou não, ao qual pode ser a consulta (requisição de uma mensagem), e alguns processamentos no dado obtido da resposta de uma requisição.

Ter uma sistema baseado em microsserviços torna o sistema mais fácil de realizar manutenções, identificar problemas, por ter uma complexidade e organização mais simples. Outra vantagem é a possibilidade do crescimento do sistema, ou seja o sistema é escalável, tanto verticalmente como horizontalmente.Aplicações baseadas em microsserviços facilitou no desenvolvimento de cada componente do sistema, porque cada componente pode ter sua própria linguagem de programação, tanto quanto frontEnd como backEnd, e banco de dados, ou seja, uma arquitetura MVC (Model View Controller) e cada uma dessas partes também podem estar em máquinas separadas.

Por outro lado as dificuldades e o maior trabalho passou a ser na comunicação entre cada aplicação, na segurança, autenticação e no gerenciamento dos microsserviços.

Para poder lidar bem com essas dificuldades o sistema tem que ter uma base bem estruturada. Para lidar com dificuldade de comunicação entre as aplicações será necessário ter as interfaces dessas comunicações bem definidas. As interfaces de comunicação são conhecidas como API (Application Programming Interface), e a padronização para estas interfaces são em REST (Representation State Transfer). As principais padronizações serão de como acontecerá às requisições de GET, POST, PUT e DELETE, que são a busca, criação, alteração e exclusão de um determinado recurso, então das requisições citadas cada uma terá definida o que ela enviará e receberá no seu header, body, parameters, e como será definida a uri para cada uma delas. O protocolo usado em cada requisição é HTTP (Hypertext Transfer Protocol).

Outra forma para melhorar a comunicação entre os microsserviços, principalmente quando um microsserviço precisa fazer uma requisição para sites de terceiros e este não está nos padrões de comunicação, neste caso é usado um middleware que fará esta intermediação. Com esta funcionalidade Middlewares também auxiliarão no auxílio de migração de sistemas monolíticos para distribuídos, pois como o sistema monolítico não foi implementado nos padrões descritos e a migração sempre ocorre de forma gradativa, à medida que microsserviços vão sendo criados o sistema monolítico vai perdendo suas funcionalidades, então o middleware fará o papel de padronizar as comunicações entre as aplicações. Este processo de migração está sendo realizado no sistema que está sendo estudado neste TCC.

Para gerenciar o sistema distribuído pode ser usado um coordenador (gerencia todas as

aplicações), para poder saber o status de cada aplicação de tempo em tempo é mandado mensagens de broadcast. Neste período do TCC não foi possível implementar um coordenador, mas futuramente será implementado e para isso será feito com o auxílio do Kubernetes para fazer o monitoramento das aplicações e com o Docker para fazer a hospedagem das aplicações.

Uma das maneiras de limitar o acesso externo, e assim ter uma segurança melhor de uma sistema, é através da criação de gateways, que permite que aplicações externas enxergue apenas um grupo de endpoints, isso permite que aplicações externas não saiba que existam endpoints que servem para controle internos, assim uma aplicação externa só irá acessar um grupo de endpoints definido independentemente se a permissão da aplicação externa mudar. Também pode ser definido os endpoints que as aplicações internas poderão enxergar, isso faz com que requisições desnecessárias sejam executadas, um tipo de configuração para limitar o acesso aos endpoints de acordo com contextos pode ser visto na Figura 1. Para se criar gateways será usado o WSO2 API Manager. O API Manager também será usado para fazer a padronização das API, além disso ele possui o seu analytics que será utilizado para realizar todas as análises de chamadas das APIs.

3 Descrição do sistema

Tecnologias usadas: Spring Boot, Node, WSO2 Identity Server, Nginx, AWS (Amazon Web Service), Tomcat, MongoDB, JQuery, JavaScript, Docker, Kubernetes.

O sistema é da RedCompany, uma empresa de consultoria que traz soluções em TI, através da integração com outros sistemas.

Os projetos atualmente são altamente dependente dos sistemas da TOTVS, pois o projeto é um módulo que usa a estrutura dos bancos de dados da TOTVS e esses projetos foram desenvolvidos com tecnologias em Progres que não é muito usada no mercado, além disso esta linguagem usada no BackEnd não é Open Code (Código aberto). Por esses motivos, tem-se problema com a alta dependência da TOTVS, e nossos produtos atualmente só podem ser implantados em empresas que possuem a licença para Progress. Isso faz com que perdemos muitas oportunidades, pelo fato de ter alguns impedimentos principalmente para a implantação do sistema, e ter um sistema com difícil escalabilidade.

As aplicações são o Chat (Sistema para troca de mensagens), News (Notícias) e o UP (Assistente virtual), Consultas (Middleware para realizar consultas de cnpj, cpf, cep), WebSocket (Middleware para push/subscribe). Atualmente todas estas aplicações não tem um único login e chave de acesso para acessar todas as aplicações visto na figura 2.

A replicação dos dados para manter a consistência entre os bancos será uma tarefa simples, pois para cada aplicação sempre terá apenas um microsserviço, ou seja, não terá duplicação de uma aplicação para distribuição de cargas. Os dados que terão de ser replicados serão os os dados dos usuários e os dados de grupos (grupos de permissão de acessos para os usuários), as alterações e criações acontecerão apenas no WSO2 Identity Server, e estas alterações não serão frequentes. Após a alteração será disparada uma mensagem de broadcast para todas as aplicações do sistema para elas atualizarem os seus dados.

Alguns requisitos são específicos de acordo com as exigências de cada cliente, então os

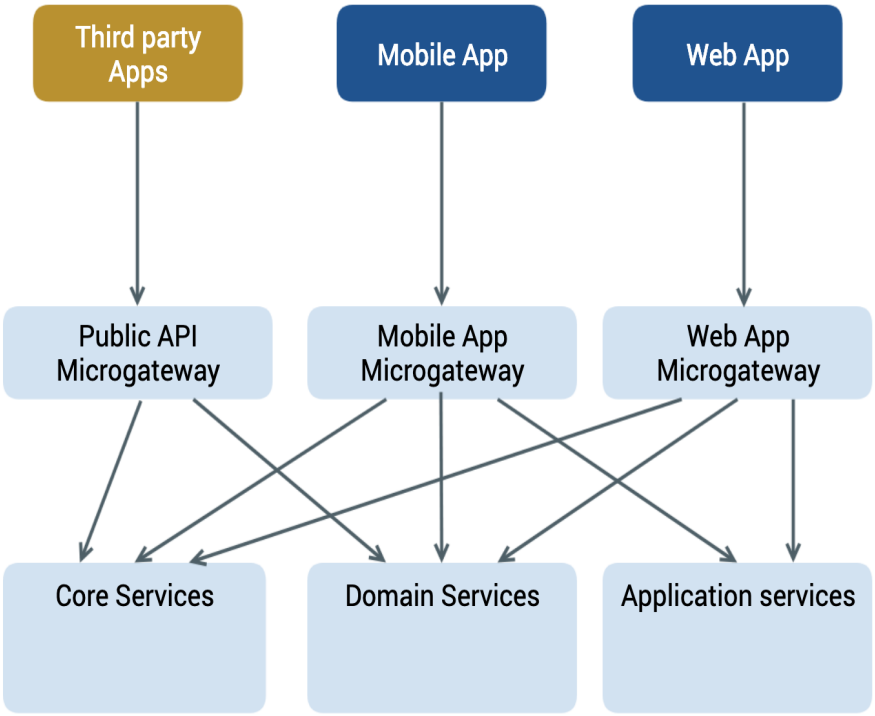

Figura 1: Exemplo de uma aplicação dos gateways [11].

requisitos que serão considerados serão requisitos comuns para todos os clientes:

- Tempo de resposta das requisições, uma quando se aplica SSO (Single Sign On) e quando não, tento o login e autorização na máquina do EndPoint.
- Possuir tempo médio de resposta de das requisições de 1000 mili segundos.
- Ter um sistema escalável, priorizando o crescimento horizontal, ou seja a criação de novos componentes, o crescimento vertical não acontecerá pelo número limitado de usuários que acessam o sistema, tendo no máximo 300 usuários acessando simultaneamente.

4 Autenticação e segurança

O Sistema está passando por uma migração que possibilitará o uso de Single Sign On (SSO), e pode-se executar os testes necessários para as análises que serão realizadas e o sistema será como na Figura 4. O SSO foi feito com uma ferramenta WSO2 Identity Server que oferece soluções às empresas a flexibilidade de implantar aplicativos e serviços no local, em nuvens privadas ou públicas ou em ambientes híbridos e migrar facilmente entre eles conforme necessário. E como todos os produtos são pré-integrados, as empresas podem se concentrar em serviços de valor agregado e chegar ao mercado mais rapidamente. A autenticação usada foi OAuth2. As aplicações já possuíam autenticação em OAuth2, então foi necessário passar o endereço de onde as aplicações iriam fazer a autenticação. Houve algumas mudanças. A arquitetura do diagrama está representado na Figura 3, onde o ClientWebApp é o FrontEnd e o ResourceServer o BackEnd das aplicações (News, Chat, Consulta, UP), o OAuth Service é o WSO2 Identity Server.

OAuth2 é um protocolo de autorização para API usada pelo Facebook e outras plataformas. O OAuth2 possui escopos para limitar o acesso de um aplicativo à conta de um usuário, o acesso aos recursos de uma aplicação são feitos através de um token, conhecido como access token que é uma autorização tipo Bearer. Para se obter o access token é necessário obtê-lo através de outro token (refresh token), e para isso também é necessário uma autenticação básica, os tokens tem tempo de expiração, geralmente o access token tem em média de 15 minutos e o refresh varia de acordo com a necessidade. Assim, caso esse token seja roubado por um hacker, o hacker não fará muitas consultas ou alterações por causa do tempo de expiração. O uso de tokens para acesso a recursos faz com que as credenciais do usuário não fiquem trafegando pela rede, assim as credenciais são usadas apenas para se obter um refresh token.. Outro benefício de usar tokens é quando uma aplicação de terceiros que precisem acessar as API do sistema, então para se obter o acesso aos tokens, a aplicação é direcionada para uma tela de login do próprio sistema, assim a aplicação de terceiro não vai ter acesso às credenciais do usuário. Além disso, quando um aplicação de terceiro obtém o token de acesso pode-se definir um escopo para limitar o acesso aos recursos do sistema.

Apesar de ter mais aplicações no sistema distribuído, apenas as aplicações que vão que foram desenvolvidas por Gunter serão descritas e feitas as análises.

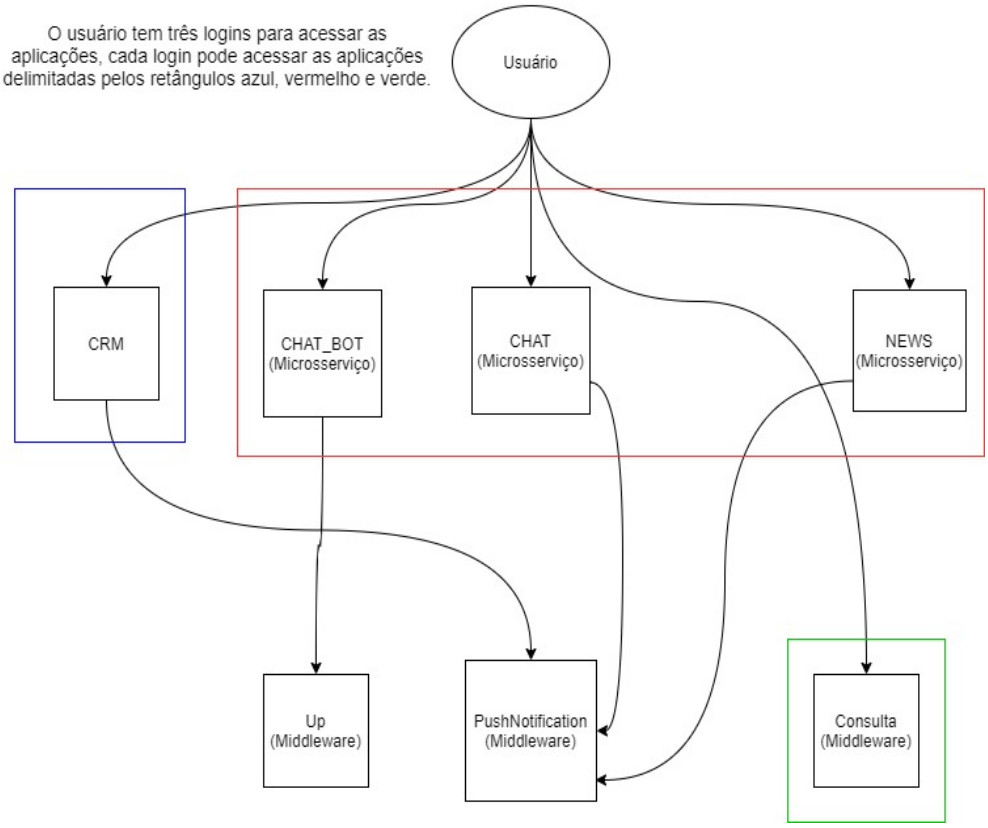

Figura 2: Esboço da Divisão atual do sistema de acordo com seus acessos. Setas representam ações e comunicações das entidades.

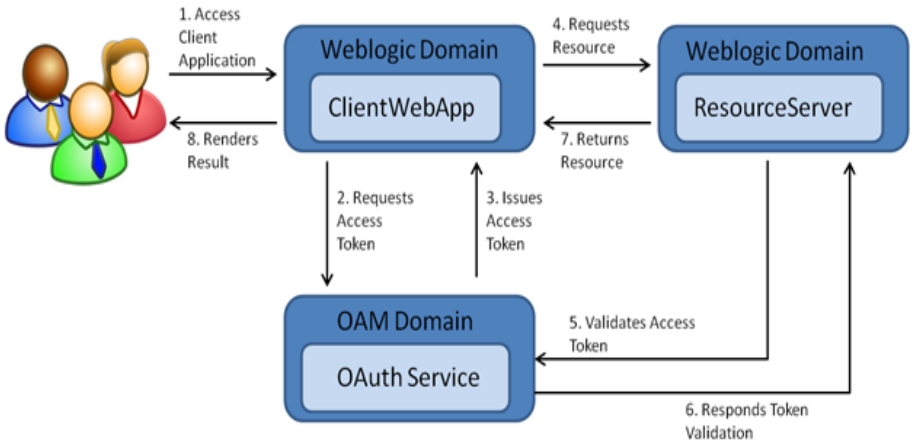

Figura 3: Arquitetura do diagrama do OAuth2 [9].

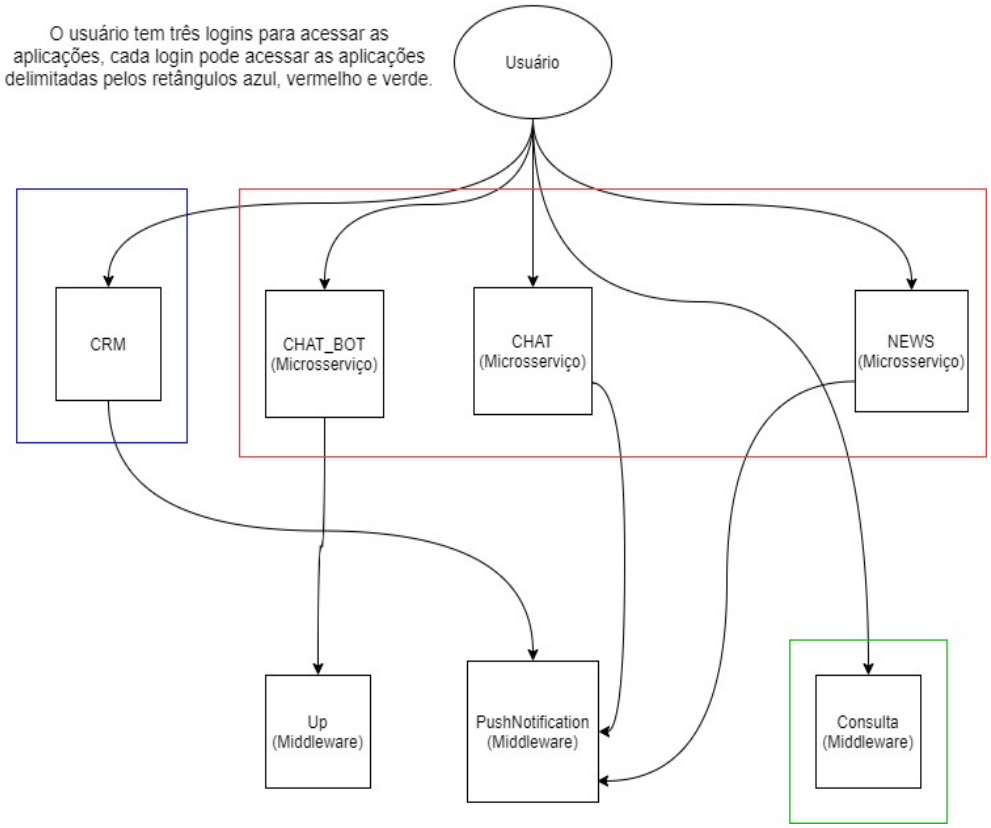

Figura 4: Esboço do Sistema com SSO (Single Sign On), e cada componente sendo um Microsserviço.

5 Aplicações

Assistente Virtual (UP) O assistente virtual, nomeado de UP, utiliza o DialogFlow: plataforma de processamento de linguagem natural que contém uma IA (Inteligência artificial) do Google para a interpretação de frases, essa ferramenta possui uma plataforma pronta para desenvolvimento de assistente virtual e algumas integrações com alguns aplicativos de Chat, como Skype, Telegram e Messenger.

– Middleware (UP)

Para a comunicação entre o ChatBot (chat do UP) com o DialogFlow foi feito um Middleware.

O Middleware foi desenvolvido com Node, usando a biblioteca Express, e esse serviço foi disponibilizado nos servidores internos. Foi usado Node pela facilidade de desenvolvimento de Middlewares e na comunicação com o DialogFlow. O fluxo de mensagens entre os componentes do UP são: o Middleware recebe requisição do ChatBot com um conteúdo de texto ou áudio, envia esse conteúdo para o DialogFlow, que envia novamente para o Middleware que interpreta a pergunta e faz a requisição para o Webmodule para obter a resposta, após obter a resposta retorna para o DialogFlow que retornará a resposta em formato de texto e áudio, assim retorna novamente a resposta para o ChatBot que irá interpretar a resposta em formato JSON e mostrará para o usuário. Este Fluxo pode ser observado na Figura 5.

– News e Chat

News é uma aplicação de notícias, tendo a parte administrativa, pois apenas alguns usuários poderão criar, alterar e excluir uma notícia, e a parte de timeline onde todos usuários poderão ler as notícias publicadas.

Chat é uma aplicação de comunicação entre os usuários através de troca de mensagens.

– WebSocket

WebSocket é um Middleware Push/Subscribe ao qual todas as aplicações FrontEnd se subscrevem em um contexto e assim o BackEnd manda requisições para o FrontEnd para mandar uma notificação ou manter o FrontEnd atualizado.

– Consultas

Consultas é um Middleware que realiza consultas de CNPJ, CPF e CEP. As consultas após serem realizadas são guardadas no banco de dados local, e caso precise atualizar o usuário deverá informar.

6 Testes e Análises

As análises serão divididas em quatro cenários: dois cenários terão a configuração da figura 2, e outros dois o da Figura 4. Para cada Cenário teremos um teste que simulará uma

situação diária e real, ou seja, onde teremos um número limitado de 200 usuários usando simultaneamente o sistema e cada usuário estará fazendo no máximo duas requisições por vez; Outros testes são testar Performance e stress da aplicação WSO2 Identity Server, e outro para testar a capacidade das aplicações, considerando 500 usuários simultâneos (por ser um número alto talvez não consiga simular exatamente os 500 usuários por limitações da máquina que está realizando os testes) com um período de 5 segundos.

As análises serão feitas nos EndPoints considerados os mais usados no sistema, algumas aplicações serão desconsideradas também pelo fato de terem rodando em tecnologias iguais e em servidores com as mesmas capacidades. Para os testes foi usado a ferramenta JMeter Apache. O JMeter obedece a uma estrutura de escopos, ou seja, cada teste tem que estar dentro de um escopo, por exemplo, o teste de requisições HTTP tem que estar dentro de um grupo de usuários, que basicamente especifica quantas requisições serão executadas e um determinado número de período. Dentro dos grupos de usuários teremos uma árvore de resultados para verificar se as requisições tiveram sucesso. Para poder analisar o desempenho foi usado o Summary Report e o Graph Results.

– Infraestrutura computacional

A máquina que realizou os testes (Testador) com o JMeter, está separada fisicamente das máquinas onde estão hospedadas as aplicações. Então para se ter uma boa conexão com o mínimo tempo de atraso possível o testador está conectado diretamente na rede, através de um cabo de rede. O testador é uma máquina com sistema operacional Windows 10, com 16 GB de memória RAM, com um processador Intel(R) Core(TM) i7 com 2 Núcleos e 4 Processadores Lógicos.

Duas máquinas hospedarão as aplicações:

- Servidor local: a máquina está na mesma rede que o testador. Esta máquina própria para ser um servidor, com 12 GB de memória RAM, com um processador Intel(R) Xeon(R), 3.30GHz, 3292 Mhz, 2 Núcleos, 4 Processadores Lógicos.
- Servidor AWS: servidor da Amazon Web Services localizado no Oeste dos EUA (Oregon), com uma CPU, e 2 GB de memória. Esta instância foi criada apenas para realizar testes, como os testes feitos no servidor aws será apenas para análise de tempo, não será necessário um máquina muito robusta.

– Cenário 1: Testes da aplicação no Servidor local

- Teste 1: Teste de tempo das requisições.

No teste 1 foi visto que o sistema obteve um bom desempenho tendo a sua média de resposta de 430 ms como visto na figura 6. Também que a quantidade de transferência de dados praticamente não teve influência sobre as médias. Por outro lado um erro encontrado mostrado na figura 6 se fosse em um sistema real não aconteceria, pois o erro ocorreu pelo acesso a dois recursos simultaneamente, e no dia a dia para este caso é um caso muito raro de acontecer, e portanto é um erro que pode-se desprezar.

- Teste 2: Teste de Performance e Stress.

Neste caso o sistema não possui um bom desempenho, afetando muito os usuários pois um tempo médio de 4 segundos para cada requisição, pode ser considerado um sistema com mau funcionamento, visto na Figura 6. Para fazer o servidor ter um melhor desempenho as funções de enviar mensagem e buscar mensagem, devem ser melhor implementadas, e como visto no teste 1 as médias das requisições são ótimas, assim pode concluir que um dos fatores que podem ter comprometido o desempenho do servidor está na sua capacidade de processamento, e assim uma das maneiras de corrigir isso seria criar um outro servidor para balanceamento de carga e/ou aumento da capacidade de processamento do servidor.

Como o testador é uma máquina com certas limitações, não é possível mandar requisições suficientes para quebrar o servidor, ou seja, a máquina tem um limite. Por exemplo, foi feito um teste para tentar fazer 5000 requisições simultâneas, mas o programa JMeter parou de funcionar.

– Cenário 2: Teste com aplicação no Servidor AWS

O tempo médio das requisições, visto na figura 6, tiveram um bom tempo, pois de acordo com os requisitos do sistema 1,5 segundos é considerado um tempo bom, com uma diferença para o Cenário 1 de aproximadamente 900 ms. Assim pode-se cortar gastos de hospedagem de servidores colocando-os no Oeste dos EUA, pois os servidores da AWS em São Paulo são mais caros.

– Cenário 3: Aplicação no servidor local, com a autorização sendo feita no WSO2 Identity Server, e hospedado no servidor local.

Os tempos de resposta foram bons, de acordo com o requisito do sistema, com um tempo de resposta de 890 ms, visto na figura 6. Apesar de ter apenas uma função, a aplicação News, o tempo de requisição foi maior comparado com o Cenário 1 Teste 1, pois a autenticação não foi feita mais na própria aplicação. Com os resultados dos testes do Cenário 2 temos um média de tempo se considerarmos a aplicação em Oregon (EUA) de aproximadamente 900 ms + 900 ms = 1800 ms, no caso o tempo já pode ser um problema para algumas aplicações.

– Cenário 4: Teste de Performance e Stress, da aplicação de Autenticação com o WSO2 Identity Server hospedada no servidor local.

Os resultados estão em um tempo razoavelmente bons, pois estão um pouco acima da média esperada de 1000ms, mas caso fossem alocados em um servidor da AWS no Oeste dos EUA os tempos podem ser bem ruins, mas alguns tempos podem ser reduzidos através do aumento de capacidade de processamento, pois como os testes foram feitos em máquinas locais a maior parte do tempo da requisição foi por causa de processamento da requisição dos usuários.

Análises Gerais

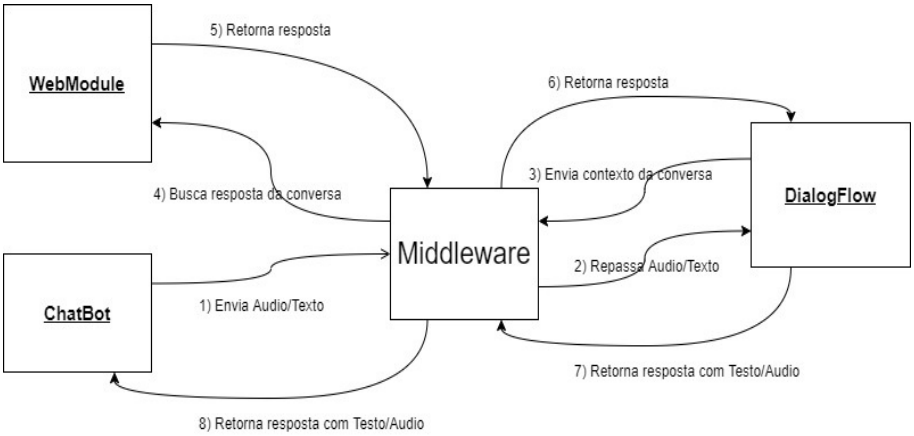

Figura 5: Diagrama de fluxo do funcionamento do Assistente Virtual.

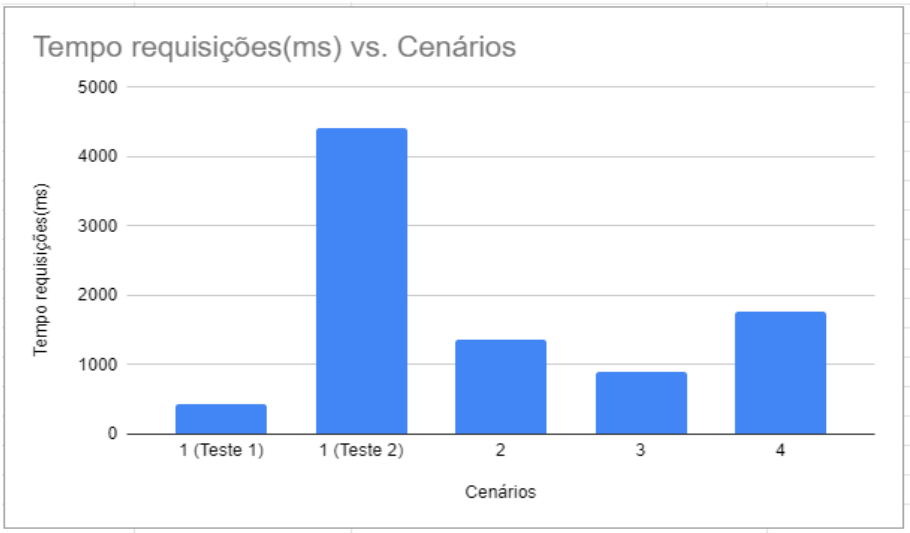

Figura 6: Gráfico do testes tempo das requisições de acordo com os cenários.

Os resultados dos testes de maneira geral foram bons, pois de acordo com o requisito de ter um tempo de resposta menor que 1000 ms, e alguns que não foram tão bons também podem ser corrigidos adicionando outros servidores para distribuição de carga (crescimento horizontal), ou aumentar o capacidade de processamento (crescimento vertical). Assim pode-se fazer um sistema baseado nas especificações do cliente, por exemplo, caso o cliente deseje que os dados de seus usuários não fiquem na nuvem (Servidores da Amazon), mas aceite que algumas aplicações como o news, que não possui dados que são mais públicos, fique na nuvem por questões de ter uma disponibilidade alta, e principalmente pelos custos, essa distribuição das aplicações poderiam ser realizadas com o auxílio do Docker e do Kubernetes, que ajudam no deploy e gerenciamento das aplicações. Caso um cliente exija que um dos requisitos do sistema seja o mais rápido possível, e não considere ter custos baixos, as aplicações podem ser hospedadas nos Servidores da AWS em São Paulo.

Assim o sistema será bem escalável e fácil manutenção, alteração, isso permite atender melhor às exigências dos clientes de acordo com os seus requisitos.

7 Conclusão

Neste período de Trabalho de Conclusão de Curso as mudanças que foram realizadas nas aplicações da empresa RedCompany foram de acordo com os estudos sobre microsserviços, e conhecimentos sobre sistemas distribuídos, esses conhecimentos ajudaram a fazer um sistema mais estruturado que possibilita ter escalabilidade, os estudos também ajudaram a tirar dúvidas e ter um definição clara sobre esses tipos de sistemas e como separar cada componente, pois inicialmente ao começar a desenvolver as aplicações alguns conceitos não estavam muito claros. As análises dos tempos de requisições, preocupações com a segurança e coordenação, são alguns ônus dos sistemas distribuídos, mas as vantagens como escalabilidade, ganho em facilidade de manutenção para cada aplicação fazem compensar esses ônus, e com as análises foi visto que o apesar do tempo das requisições este é um fator que não afetará os usuários de forma negativa desde que o tempo esteja dentro do tempo determinado pelos requisitos do sistema.

Referências

[1] Microservices Architecture Enables DevOps - Migration to a Cloud-Native Architecture - Armin Balalaie and Abbas Heydarnoori, Sharif University of Technology

Pooyan Jamshidi, Imperial College London

[2] Open Issues in Scheduling Microservices in the Cloud

Maria Fazio and Antonio Celesti, University of Messina - Rajiv Ranjan and Chang Liu, Newcastle University - Lydia Chen, IBM Research - Massimo Villari, University of Messina

[3] Microservices - The Journey So Far and Challenges Ahead

Pooyan Jamshidi, Carnegie Mellon University - Claus Pahl, Free University of Bozen-Bolzano - Nabor C. Mendonça, University of Fortaleza - James Lewis, ThoughtWorks - Stefan Tilkov, INNOQ

[4] Microservices: Architecting for Continuous Delivery and DevOps - 2018 IEEE International Conference on Software Architecture

Lianping Chen - Lianping Chen Limited - Dublin, Ireland

[5] The Design and Architecture of Microservices

Alan Sill, Texas Tech University,

[6] Using Microservices for Legacy Software Modernization

Holger Knoche and Wilhelm Hasselbring, Kiel University

[7] Microservices

Johannes Thönes - Editor: Robert Blumen, Symphony Commerce

[8] WSO2 Identity Server: https://wso2.com/

[9] OAuth2: https://oauth.net/2/

https://docs.oracle.com/cd/E82085$_0$1/160023/JOS

[10] Empresa RedCompany: https://www.redcompany.com.br/

[11] Figura 1: https://igia.github.io/docs/architecture