
Gustavo José Neves da Silva

Marlon Henry Schweigert

Análise de disponibilidade utilizando Docker Swarm

Joinville

2018

UNIVERSIDADE DO ESTADO DE SANTA CATARINA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Gustavo José Neves da Silva

Marlon Henry Schweigert

ANÁLISE DE DISPONIBILIDADE UTILIZANDO DOCKER
SWARM

Trabalho de conclusão de curso submetido à Universidade do Estado de Santa Catarina
como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação

Charles Christian Miers

Orientador

Joinville, Junho de 2018

ANÁLISE DE DISPONIBILIDADE UTILIZANDO DOCKER SWARM

Gustavo José Neves da Silva

Marlon Henry Schweigert

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Ciência da Computação Integral do CCT/UDESC.

Banca Examinadora

Charles Christian Miers - Doutor (orientador)

-

-

Agradecimientos

AGRADECIMENTOS

Resumo

A crescente popularização de jogos massivos demanda por novas abordagens tecnológicas a fim de suprir as necessidades dos usuários com menor custo de recursos computacionais. Projetar essas arquiteturas, do ponto de vista da rede, é algo pertinente e impactante para o sucesso desses jogos. O objetivo deste trabalho é propor uma análise voltada a identificar abordagens para otimização dos recursos computacionais consumidos pelas arquiteturas identificadas. Esse objetivo será atingido após realizar uma pesquisa referenciada, seguida de uma análise das principais arquiteturas e, preferencialmente, a execução de simulações usando uma nuvem computacional para auxiliar na identificação de gargalos de recursos. Os resultados obtidos auxiliarão provedores de serviços *Massively Multiplayer Online Role-Playing Game* (MMORPG) a reduzir gastos de manutenção e melhorar a qualidade de tais serviços.

Palavras-chaves: Arquitetura de microserviços, Docker, Docker Swarm

Sumário

Lista de Figuras	5
Lista de Tabelas	6
Lista de Abreviaturas	7
1 Introdução	8
1.1 Fundamentação Teórica	8
1.1.1 Microserviços: Definição e Funcionamento	8
1.1.2 Containers	10
1.1.3 Docker	11
1.1.4 Docker Swarm	11
1.1.5 Arquitetura Monolítica	12
1.2 Boas práticas	13
1.3 Principais aplicações	14
1.3.1 Aplicações Web	14
1.3.2 Streaming	14
1.3.3 Jogos	14
2 Casos Comentados	17
2.1 Walmart	17
2.2 Spotify	17
2.3 Amazon	18
2.4 Guild Wars 2	18

3	Análise	19
3.1	Método de deploy	19
3.2	Arquitetura obtida	19
3.3	Análise sobre a arquitetura obtida	19
4	Conclusão	20
	Referências	21

Lista de Figuras

1.1	Microserviços podem ter diferentes tecnologias	9
1.2	Microserviços são escaláveis	10
1.3	Containers sobre sistema linux	11
1.4	Tecnologia Docker em comparação aos containers Linux (LXC).	12
1.5	Rede de Docker Swarm.	12
1.6	Arquitetura Monolítica X Arquitetura de Microserviços	13
1.7	Arquitetura Rudy, utilizada no jogo Tibia.	14
1.8	Arquitetura Salz, utilizada no jogo Albion.	15
1.9	Arquitetura Knowles, utilizada no jogo Guild Wars 2.	15

Lista de Tabelas

Lista de Abreviaturas

API *Application Programming Interface*

HTTP *Hypertext Transfer Protocol*

JSON *JavaScript Object Notation*

MMORPG *Massively Multiplayer Online Role-Playing Game*

PaaS *Platform as a Service*

REST *Representational State Transfer*

TCP *Transmission Control Protocol*

1 Introdução

1.1 Fundamentação Teórica

A fim de orientar o análise encontrada neste trabalho, se faz necessário a descrição e pesquisa por fundamentação teórica dos conceitos básicos relevantes no contexto de microserviços e implantação dessas arquiteturas. Por esse motivo, esta seção irá introduzir os conceitos de microserviços, containers, docker e implantação utilizando a técnica de docker swarm.

1.1.1 Microserviços: Definição e Funcionamento

Entende-se por microserviço, aplicações que executam operações menores de um macroserviço, da melhor forma possível (WILLSON, 2017; NEWMAN, 2015). O objetivo de uma arquitetura de microserviços é funcionar separadamente de forma autônoma, contendo baixo acoplamento (NEWMAN, 2015). Seu funcionamento deve ser desenhado para permitir alinhamentos de alta coesão e baixo acoplamento entre os demais microserviços existentes em um macroserviço (ACEVEDO; JORGE; PATIÑO, 2017).

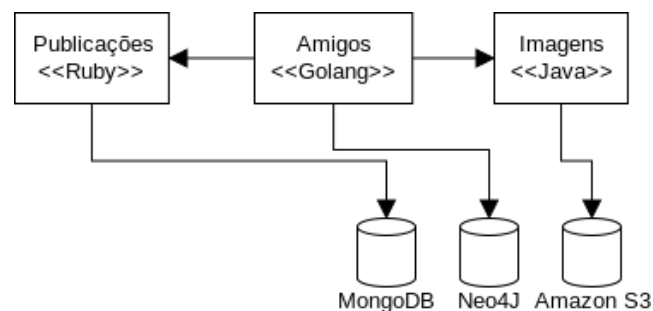
Arquiteturas de microserviços iniciam uma nova linha de desenvolvimento de aplicações preparadas para executar sobre nuvens computacionais, promovendo maior flexibilidade, escalabilidade, gerenciamento e desempenho, sendo a principal escolha de arquitetura de grandes empresas como Amazon, Netflix e LinkedIn (KHAZAEI et al., 2016; VILLAMIZAR et al., 2016). Um microserviço é definido pelas seguintes características (ACEVEDO; JORGE; PATIÑO, 2017):

- Deve possibilitar a implementação como uma peça individual do macroserviço.
- Deve funcionar individualmente.
- Cada serviço deve ter uma interface. Essa interface deve ser o suficiente para utilizar o microserviço.

- A interface deve estar disponível na rede para chamada de processamento remoto ou consulta de dados.
- O serviço pode ser utilizado por qualquer linguagem de programação e/ou plataforma.
- O serviço deve executar com as dependências mínimas.
- Ao agregar vários microsserviços, o macroserviço resultante poderá prover funcionalidades complexas.

O microsserviço deverá ser uma entidade separada. A entidade deve ser implantada como um sistema independente em um *Platform as a Service* (PaaS). Toda a comunicação entre os microsserviços de um macroserviço será executada sobre a rede, a fim de reforçar a separação entre cada serviço. As chamadas pela rede com o cliente ou entre os microsserviços será executada através de uma *Application Programming Interface* (API), permitindo a liberdade de tecnologia em que cada um será implementado (NEWMAN, 2015). Isso permite que o sistema contenha tecnologias distintas que melhor resolvam os problemas relacionados ao contexto deste microsserviço. Isso pode ser visualizado na Figura 1.1.

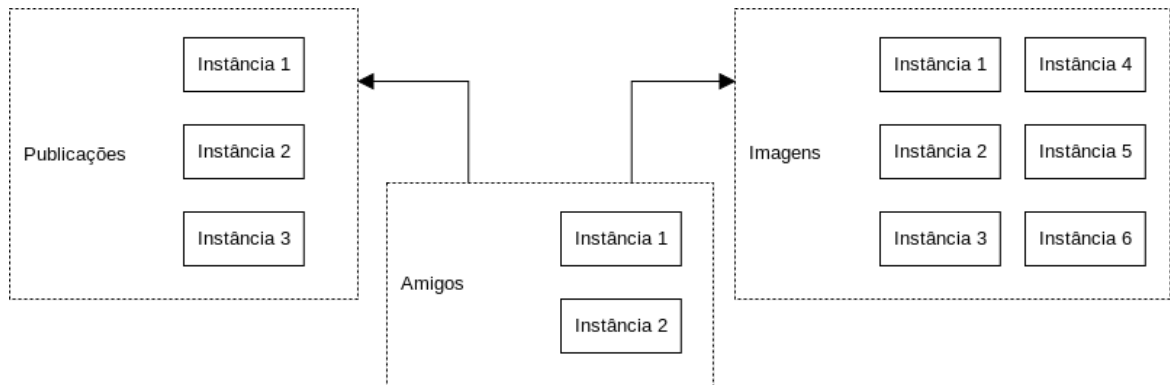
Figura 1.1: Microsserviços podem ter diferentes tecnologias



Adaptado de: (NEWMAN, 2015)

Uma arquitetura de microsserviços é escalável, como visível na Figura 1.2. Ela permite o aumento do número de microsserviços sob demanda para suprir a necessidade de escalabilidade. Este modelo computacional obtém maior desempenho, principalmente se executar sobre plataformas de computação elástica, na qual o orquestrador do macroserviço pode aumentar o número de instâncias conforme a necessidade de requisições (NADAREISHVILI et al., 2016).

Figura 1.2: Microserviços são escaláveis



Adaptado de: (NEWMAN, 2015)

Microserviços desenvolvidos para web utilizam arquitetura *Representational State Transfer* (REST) baseado sobre o protocolo *Hypertext Transfer Protocol* (HTTP). É uma boa prática utilizar o corpo com conteúdo da requisição e resposta no formato *JavaScript Object Notation* (JSON) nas chamadas a uma API de microserviço web (NADAREISHVILI et al., 2016).

Entretanto, necessita-se de um método para garantir o seu isolamento, a qual, em sistemas Linux, podem ser garantidos utilizando Containers. Por este motivo, se faz necessário descreve-lo.

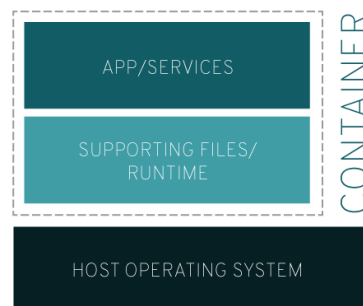
1.1.2 Containers

Os container permitem ao desenvolvedor "empacotar" sua aplicação e todas as suas dependências (ex. bibliotecas), dessa forma, lhe é garantido que a aplicação terá o mesmo comportamento independentemente do hospedeiro Linux.

Um container Linux é um conjunto de processos que executam de forma isolada do restante do sistema. Esses processos utilizam arquivos providos de uma imagem, a qual lhe garante a compatibilidade a fim de evitar problemas de versionamento e conflitos de processos. Essa separação pode ser visualizada na Figura 1.3.

Entretanto, se faz necessário uma ferramenta para gerenciamento de imagens e containers. Entra em cena a partir de 2008 o *Docker*, uma ferramenta que permite, de forma prática, a publicação de imagens no formato Dockerfile, além de contar com um gerenciador e repositório de imagens.

Figura 1.3: Containers sobre sistema linux



Fonte: (RedHat, 2018)

1.1.3 Docker

Docker é uma plataforma que nos permite "construir, embarcar e rodar uma aplicação em qualquer lugar". Ele percorreu um longo caminho em um período de tempo incrivelmente curto e atualmente é considerado uma solução padrão para um dos aspectos mais custosos do software: a implantação (MIELL; AIDAN, 2016)

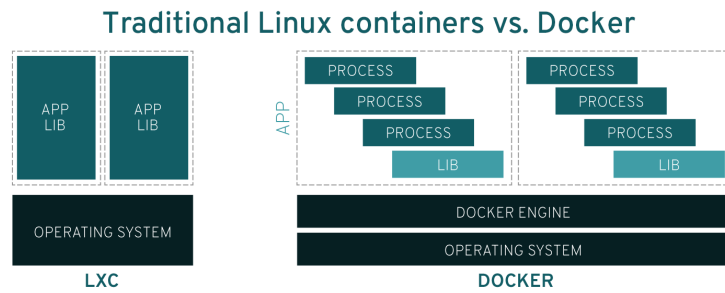
Docker é uma ferramenta desenvolvida para facilitar o processo de criação, implantação e execução de aplicações por meio do uso de containers. Pode ser pensado como um tipo de máquina virtual, porém diferentemente desta que necessita a criação de todo um sistema operacional virtual, o Docker, permite que as aplicações compartilhem o mesmo kernel Linux que o hospedeiro, reduzindo assim o tamanho da aplicação e obtendo um ganho de desempenho.

A tecnologia Docker usa o *kernel* Linux, abstraindo sistemas como *Cgroups* e *namespaces* para segregar processos a fim que eles possam ser executados de forma independente. O objetivo dos containers criados pelo Docker continua da mesma forma que os containers Linux, conhecida como *LXC*. A diferença entre Docker e *LXC* é relevante a escalabilidade, visto que containers Docker permitem múltiplos processos executando juntamente a uma biblioteca, já o padrão *LXC* permite somente um processo junto a sua biblioteca, consumindo mais recursos da máquina. Essa comparação pode ser visualizada na Figura 1.4.

1.1.4 Docker Swarm

É uma ferramenta nativa do Docker que permite criar clusters de containers, chamado swarms, o que possibilitam a escalabilidade de recursos de acordo com a demanda(carga) (TURN-

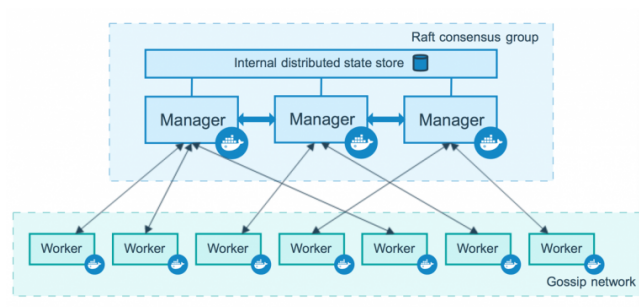
Figura 1.4: Tecnologia Docker em comparação aos containers Linux (LXC).



Fonte: (RedHat, 2018)

BULL, 2017) O que possibilita que diversos hospedeiros de Docker estejam inseridos no mesmo pool de recursos, facilitando assim a implantação de containers, uma vez que o Swarm disponibiliza uma API de integração que abstrai grande parte das atividades necessárias a administração dos containers e promove um tipo de tolerância a falhas

Figura 1.5: Rede de Docker Swarm.



Fonte: (RedHat, 2018)

O seu principal objetivo é resolver problemas de gerência de microsserviços, a qual antes eram resolvidos somente com *Kubernetes*, uma ferramenta criada pela Google em 2015. Com Docker Swarm, pode-se ter um nó líder que gerenciará a rede, e nós trabalhadores. Um exemplo de rede Docker Swarm pode ser visualizado na Figura 1.5.

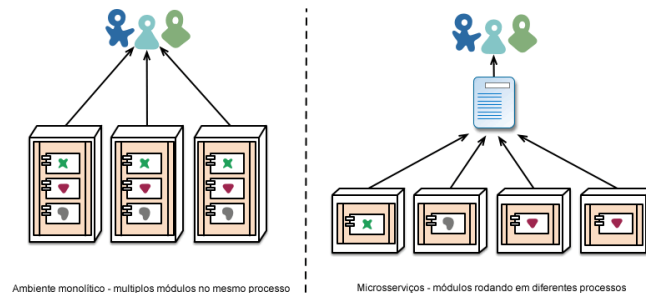
1.1.5 Arquitetura Monolítica

Arquitetura Monolítica, é uma arquitetura de desenvolvimento, na qual tipicamente, apesar da complexidade dos sistemas ser quebrada ao se utilizar módulos, esses são projetados para a criação de um único executável, o qual possui todos os seus módulos executados em uma mesma máquina. Com o passar do tempo, o sistema cresce e tende a tornar-se cada vez mais complexo, o que gera diversos problemas em sua manutenção, por exemplo temos, a escalabilidade do sistema, que exige que o mesmo seja replicado inteiramente,

mesmo que apenas uma parte desse seja necessária na nova instância, aumentando assim os custos.

Com a necessidade de escalabilidade, as arquiteturas de microsserviços obtiveram sucesso em grandes projetos comerciais como LinkedIn, Google e Youtube.

Figura 1.6: Arquitetura Monolítica X Arquitetura de Microsserviços



1.2 Boas práticas

- Não armazenar dados em containers, uma vez que esses podem ser parados, destruídos ou mesmo substituídos, se necessário armazenar dados deve ser feito em um volume, com o cuidado de evitar que dois ou mais containers escrevam dados em um mesmo volume, o que poderia causar o corrompimento dos dados.
- Não criar imagens grandes, já que essas possuirão uma distribuição complexa, uma imagem deve possuir apenas as bibliotecas e arquivos necessários para a execução da aplicação ou processo.
- Não executar mais de um processo/aplicação em um único container, pois tal comportamento acarretará em aumento da complexidade do gerenciamento e no número de logs.
- Não dos endereços IP dos containers, o endereço IP do container pode se alterar quando o mesmo é iniciado e parado. Caso seja necessária a comunicação entre a aplicação ou microsserviço e um outro container, é recomendado o uso de variáveis de ambiente para transmitir o hostname e porta corretos de um container para outro.

1.3 Principais aplicações

1.3.1 Aplicações Web

Microserviços desenvolvidos para web utilizam arquitetura REST baseado sobre o protocolo HTTP. É uma boa prática utilizar o corpo com conteúdo da requisição e resposta no formato JSON nas chamadas a uma API de microserviço web (NADAREISHVILI et al., 2016).

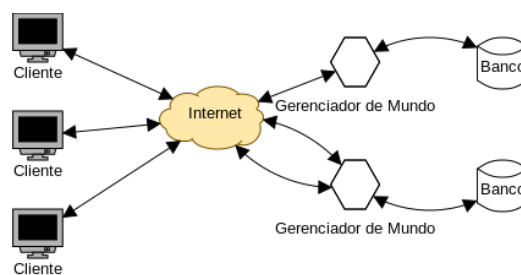
1.3.2 Streaming

Devido as suas propriedades os microserviços, possibilitam que operações de streaming (cloud-based) possuam maior elasticidade e escalabilidade do que uma implementação padrão (monolítica), além de possibilitar a redução de custos de operação. (MOULDING, 2017)

1.3.3 Jogos

Alguns exemplos de arquitetura de microserviços para jogos MMORPG são as arquiteturas apresentadas por Rudy (Figura 1.7), Salz (Figura 1.8) e a arquitetura escrita por Knowles (Figura 1.9).

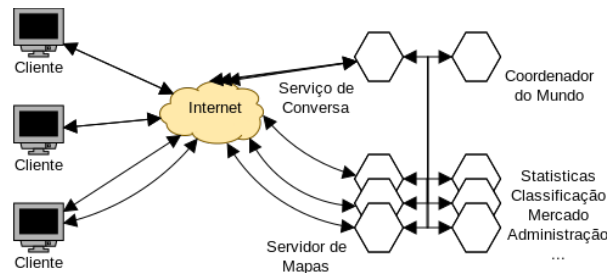
Figura 1.7: Arquitetura Rudy, utilizada no jogo Tibia.



Adaptado de: (RUDDY, 2011)

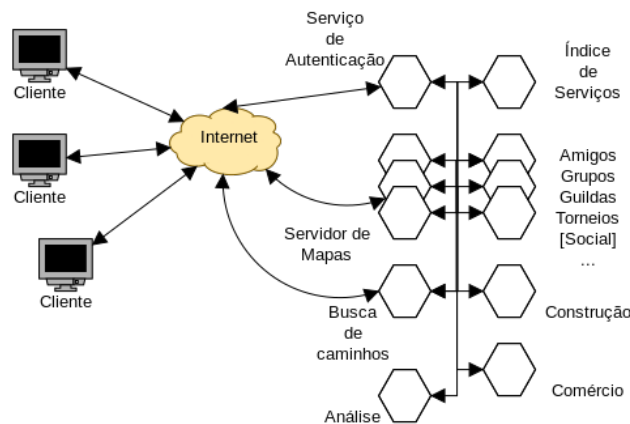
A arquitetura Rudy (Figura 1.7) é formada por um sistema cliente-servidor monolítico, na qual cada microserviço individual gerencia um mundo mútuo dos demais gerenciadores de mundo (RUDDY, 2011). Essa arquitetura dificulta a escalabilidade, modificações e manutenção (ACEVEDO; JORGE; PATIÑO, 2017), além de segregar a comunidade de jogadores em servidores menores (RUDDY, 2011). Inicialmente essa arquitetura foi pensada para ser um sistema Cliente-Servidor monolítico. A arquitetura Rudy é uma

Figura 1.8: Arquitetura Salz, utilizada no jogo Albion.



Adaptado de: (SALZ, 2016)

Figura 1.9: Arquitetura Knowles, utilizada no jogo Guild Wars 2.



Adaptado de: (WILLSON, 2017)

arquitetura de microsserviços adaptada de um serviço cliente-servidor (RUDDY, 2011). O jogo Tibia¹, operante sobre essa arquitetura, possui 68 mundos oficiais (Sendo 2 servidores de teste) (RUDDY, 2011), com capacidade para 1.050 clientes em cada servidor, na qual encontra-se restringido pelo gerenciador de mundo.

A arquitetura Salz (Figura 1.8) é formada por diversos microsserviços (SALZ, 2016). O principal objetivo dessa arquitetura é modularizar o serviço visando melhorar a escalabilidade. Ela é atualmente utilizada no jogo Albion Online². A arquitetura é planejada para funcionar conforme a seguinte especificação(SALZ, 2016):

- O mundo é distribuído sobre os vários servidores de mapas. Cada microsserviço gerencia uma região do mundo, denominado *chunk*.
- Jogadores mudam a conexão com os microsserviços quando estão posicionados na borda de um *chunk*.
- A autorização de acesso aos microsserviços é obtido pelo banco de dados.

¹Tibia: <http://www.tibia.com>

²Albion Online: <https://albiononline.com>

- O coordenador do mundo é responsável por tudo que seja de escopo global (*e. g.*, Grupos, chat global, guildas, *etc.*).

A arquitetura Knowles (Figura 1.9) é distribuída em diversos microsserviços, assim como a arquitetura Salz (Figura 1.8). A diferença em comparação a arquitetura Salz (Figura 1.8) está na decomposição da arquitetura para outros microsserviços e a conexão direta entre esses microsserviços e o cliente. O principal objetivo dessa arquitetura é facilitar a manutenção e desempenho de reinicialização do macroserviço (WILLSON, 2017). Guild Wars 2³ é um jogo que executa sobre a arquitetura Knowles. Ele é popularmente conhecido por ter seus serviços sempre ativos, visto que a arquitetura possibilita desativar pequenos pedaços do serviço para manutenções básicas e a sua reinicialização é rápida para manutenções críticas.

³Guild Wars 2: <https://www.guildwars2.com>

2 Casos Comentados

2.1 Walmart

Por volta do ano de 2012, o Walmart decidiu migrar sua arquitetura monolítica para a arquitetura de microsserviços, uma vez a arquitetura anterior falhou 2 anos seguidos na entrega dos serviços durante o black friday. A arquitetura de microsserviços foi escolhida devido a sua escalabilidade de acordo com a demanda e também buscando atingir o mais próximo possível a meta de 100% de disponibilidade de seus serviços. Os custos operacionais foram reduzidos significativamente, uma vez que com essa nova arquitetura, a empresa pode migrar de servidores com hardware específico para servidores com hardware "comum"(AMD64). Foi atingida uma economia de 40% do poder computacional e de 20-50% do custos em geral (HÁMORI, 2016). Estima se que por volta de 3.000 engenheiros utilizando essa arquitetura são responsáveis por 30.000 alterações por mês nos sistemas do Walmart (VIZARD, 2015)

2.2 Spotify

O Spotify possui mais de 75 milhões de usuários por mês, em que cada uma dessas sessões duração em média 23 minutos, ou seja, escalabilidade e disponibilidade são imperativas. Além desses fatores a empresa possui mais de 90 equipes de desenvolvimento, 600 desenvolvedores e 5 escritórios de desenvolvimento em 2 continentes construindo o mesmo produto, portanto era necessário reduzir ao máximo as dependências existentes entre os componentes do software (HÁMORI, 2016). A solução encontrada foi a arquitetura de microsserviços com equipes de desenvolvimento full-stack, que são compostas por desenvolvedores back-end, desenvolvedores front-end, testers, designer de interfaces de usuário e dono do produto. Suas missões não sobrepoem as missões de outras equipes, já que são equipes autônomas.

2.3 Amazon

Por volta do ano de 2001, a Amazon, possuía uma grande arquitetura monolítica (III, 2015). Porém com o passar do tempo, os projetos foram amadurecendo, mais desenvolvedores foram adicionados e a base de código foi crescendo e a complexidade da arquitetura foi crescendo, o que ocasionou no aumento de duração do ciclo de desenvolvimento dos softwares. Mas a complexidade ainda maior era a de prever a demanda devido ao tráfego de acesso flutuante, a Amazon perdia dinheiro e a maioria da capacidade de processamento era desperdiçada em momentos de baixo tráfego. Ao realizar a migração para o Amazon Web Services(AWS) permitiu em conjunto com o uso da arquitetura de micros-serviços implementar uma maior escalabilidade, além de permitir o deployment contínuo do código, atualmente os engenheiros realizam o deploy do código a cada 11.7 segundos (MAUERSBERGER, 2017).

2.4 Guild Wars 2

Guild Wars 2 foi publicado em 2012, possuindo um vasto ambiente a qual jogadores podem interagir entre si. Entretanto, a sua arquitetura de microsserviços possui 80 tipos de microsserviços diferentes, entre serviços baseados em *Web* e protocolos proprietários sobre *Transmission Control Protocol* (TCP). Nesse sentido, a sua grande dificuldade é o gerenciamento de reinicialização dos microsserviços, a qual precisam ser coordenados por um orquestrador a fim de permitir o roteamento correto das requisições dos jogadores. A solução encontrada para este problema foi um orquestrador a qual coloque serviços ao hardware, permissões, roteamento de requisições e isolamento, visto que esta arquitetura de microsserviços não utiliza virtualização e não tem os benefícios das técnicas de Docker Swarm.

3 Análise

3.1 Método de deploy

3.2 Arquitetura obtida

3.3 Análise sobre a arquitetura obtida

4 Conclusão

Conclusão

Referências

- ACEVEDO, C. A. J.; JORGE, J. P. G. y; PATIÑO, I. R. Methodology to transform a monolithic software into a microservice architecture. In: *2017 6th International Conference on Software Process Improvement (CIMPS)*. Zacatecas, Mexico: IEEE, 2017. p. 1–6.
- BENEVIDES, R. *10 things to avoid in docker containers*. 2016. [Online; accessed 25. May 2018]. Disponível em: <<https://developers.redhat.com/blog/2016/02/24/10-things-to-avoid-in-docker-containers/>>.
- HÁMORI, F. *How Enterprises Benefit From Microservices Architectures*. 2016. Disponível em: <<https://blog.risingstack.com/how-enterprises-benefit-from-microservices-architectures/>>.
- III, S. M. F. *What Led Amazon to its Own Microservices Architecture*. 2015. Disponível em: <<https://thenewstack.io/led-amazon-microservices-architecture/>>.
- KHAZAEI, H. et al. Efficiency analysis of provisioning microservices. In: *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. Luxembourg, Austria: IEEE, 2016. p. 261–268.
- MAUERSBERGER, L. *Why Netflix, Amazon, and Apple Care About Microservices*. 2017. Disponível em: <<https://blog.leanix.net/en/why-netflix-amazon-and-apple-care-about-microservices>>.
- MIELL, I.; AIDAN, S. H. *Docker in Practice*. Manning Publications, 2016. ISBN 978-1617292729. Disponível em: <<https://www.amazon.com.br/Docker-Practice-Ian-Miell/dp/1617292729>>.
- MOULDING, J. *How microservices take cloud-based TV operations to another level*. 2017. [Online; accessed 25. May 2018]. Disponível em: <<http://www.v-net.tv/2017/09/07/how-microservices-take-cloud-based-tv-operations-to-another-level/>>.
- NADAREISHVILI, I. et al. *Microservice Architecture: Aligning Principles, Practices, and Culture*. O'Reilly Media, 2016. ISBN 978-149195625-0. Disponível em: <<https://www.amazon.com/Microservice-Architecture-Aligning-Principles-Practices/dp/1491956259>>.
- NEWMAN, S. *Building Microservices*. O'Reilly Media, 2015. ISBN 978-149195035-7. Disponível em: <<https://www.amazon.com.br/Building-Microservices-Sam-Newman/dp/1491950358>>.
- RedHat. *O que é um container Linux?* 2018. [Online; accessed 24. May 2018]. Disponível em: <<https://www.redhat.com/pt-br/topics/containers/whats-a-linux-container>>.
- RUDDY, M. *Inside Tibia, The Technical Infrastructure of an MMORPG*. 2011. Disponível em: <http://twvideo01.ubm-us.net/o1/vault/gdceurope2011/slides-/Matthias_Rudy_ProgrammingTrack_InsideTibiaArchitecture.pdf>.

SALZ, D. *Albion Online - A Cross-Platform MMO (Unite Europe 2016, Amsterdam)*. 2016. Disponível em: <<https://www.slideshare.net/davidsalz54/albion-online-a-crossplatform-mmo-unite-europe-2016-amsterdam>>.

TURNBULL, J. *The Docker Book*. James Turnbull, 2017. ISBN 0. Disponível em: <<https://dockerbook.com/>>.

VILLAMIZAR, M. et al. Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures. In: *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. Cartagena, Colombia: IEEE, 2016. p. 179–182.

VIZARD, M. *Walmart Embraces Microservices to Get More Agile*. 2015. Disponível em: <<http://www.baselinemag.com/enterprise-apps/walmart-embraces-microservices-to-get-more-agile.html>>.

WILLSON, S. C. *Guild Wars Microservices and 24/7 Uptime*. 2017. Disponível em: <http://twvideo01.ubm-us.net/o1/vault/gdc2017/Presentations/Clarke-Willson_Guild Wars 2 microservices.pdf>.