
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	8
1.1.2 Containers	9
1.1.3 Docker	9
1.1.4 Docker Swarm	9
1.2 Histórico	10
1.2.1 Arquitetura Monolítica	10
1.2.2 Arquitetura de Microserviços	10
1.3 Funcionamento	11
1.4 Boas práticas	11
1.5 Principais aplicações	12
1.5.1 Aplicações Web	12
1.5.2 Streaming	12
1.5.3 Jogos	12
2 Casos Comentados	13
2.1 Walmart	13
2.2 Spotify	13

2.3	Amazon	13
2.4	Guild Wars 2	13
3	Análise	14
3.1	Método de deploy	14
3.2	Arquitetura obtida	14
3.3	Análise sobre a arquitetura obtida	14
4	Conclusão	15
	Referências	16

Lista de Figuras

1.1	Microserviços podem ter diferentes tecnologias	10
1.2	Microserviços são escaláveis	11

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*

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 microsserviços e implantação dessas arquiteturas. Por esse motivo, esta seção irá introduzir os conceitos de microsserviços, containers, docker e implantação utilizando a técnica de docker swarm.

1.1.1 Microserviços

Entende-se por microsserviço aplicações que executam operações menores de um macroserviço, da melhor forma possível (WILLSON, 2017; NEWMAN, 2015).

Um microsserviço 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 microsserviço.
- 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.

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.

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 (citar Docker in Practice)

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.

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) (citar The Docker Book) 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

1.2 Histórico

1.2.1 Arquitetura Monolítica

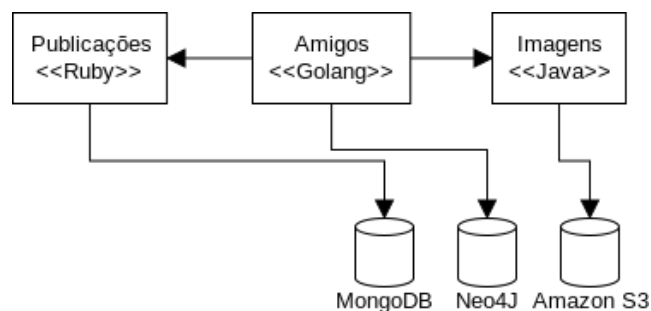
(Pensar) Arquitetura Monolítica, é uma arquitetura de desenvolvimento, na qual tipicamente , apesar da complexidade dos sistemas ser quebrada ao se utilizar módulos, esses são projetados para a criação de um único executável, o qual possui todos os seus módulos executados em um mesma máquina. Com o passar 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.

1.2.2 Arquitetura de Microserviços

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).

Figura 1.1: Microserviços podem ter diferentes tecnologias

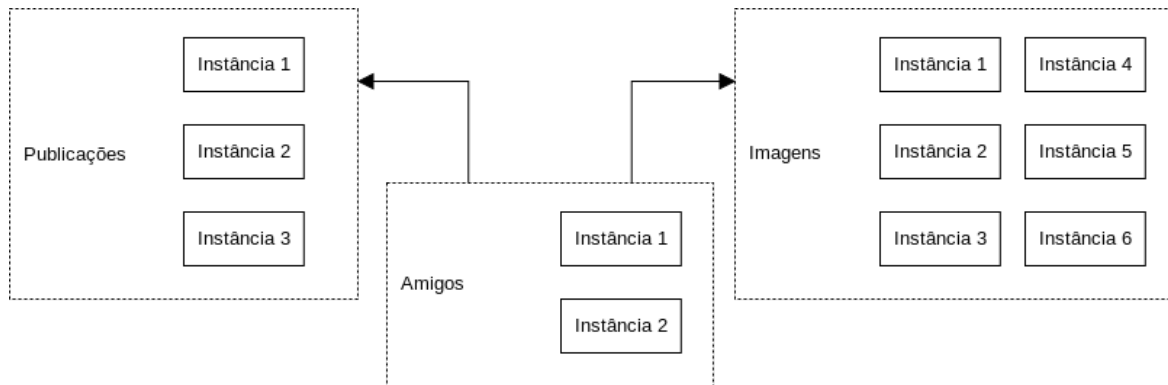


Adaptado de: (NEWMAN, 2015)

O microserviço deverá ser uma entidade separada. Ela 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).

Figura 1.2: Microsserviços são escaláveis



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).

1.3 Funcionamento

1.4 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 alterado quando o mesmo é iniciado e parado. Caso seja necessária a comunicação entre a aplicação ou microserviço e um outro container, é recomendado o uso de variáveis de ambiente para transmitir o hostname e porta corretos de um container para outro.

1.5 Principais aplicações

1.5.1 Aplicações Web

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).

1.5.2 Streaming

1.5.3 Jogos

2 Casos Comentados

2.1 Walmart

2.2 Spotify

2.3 Amazon

2.4 Guild Wars 2

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.
- 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.
- 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>>.
- 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.
- 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>.