Trade-offs da Implementação de Arquiteturas de Microsserviços Utilizando o Modelo de Atores

Wander Douglas Andrade de Souza
Orientador: Alfredo Goldman
Co-orientadores: João Francisco Lino Daniel,
Renato Cordeiro Ferreira e Thatiane de Oliveira Rosa

26 de Maio de 2021

1 Introdução

Cada vez mais usuários procuram por aplicações com alta responsividade e disponibilidade. Além disso, as equipes de desenvolvimento têm se tornado cada vez maiores, o que exige uma nova maneira de organizá-las. Para atender a essas demandas, tem sido recorrente adotar o estilo arquitetural de microsserviços. Uma das principais características desse estilo é o baixo acoplamento entre os serviços, o que permite maior independência e escalabilidade do sistema.

De modo a aproveitar o máximo dessas características, uma das abordagens recomendadas é a reativa. Nessa abordagem, a comunicação entre serviços é assíncrona, ou seja, um serviço que necessita se comunicar com outros emite uma mensagem, e a responsabilidade de reagir ou não a ela pertence a cada serviço.

Um dos arcabouços mais utilizados para implementar arquiteturas reativas é o Akka, uma plataforma que implementa o modelo de atores, um modelo conceitual para lidar com concorrência que regras gerais sobre como os componentes de um sistema devem se comportar e interagir entre si.

Apesar dos benefícios descritos anteriormente, a escolha de uma plataforma reativa para implementar sistemas reativos necessita ser cautelosa, pois esse paradigma traz uma mudança na forma de pensar a solução para sistemas de grande escala. Considerando esse contexto, o objetivo deste projeto é analisar as vantagens e desvantagens na implementação de sistemas reativos utilizando programação reativa.

Este projeto está estruturado da seguinte forma: A Seção 2 detalha os conceitos técnicos que são importantes para a compreensão da proposta. A Seção detalha a proposta, apresentando as etapas planejadas para alcançar o objetivo descrito. A Seção 4 apresenta o cronograma de execução deste projeto.

2 Fundamentação Teórica

2.1 Arquitetura de software

Arquitetura de software é "o conjunto de estruturas necessárias para a compreensão de um sistema, que abrange elementos que compõem ele, a relação entre eles e suas propriedades" (BASS; CLEMENTS; KAZMAN, 2013).

No contexto de arquitetura de software, um conceito muito importante é o de estilo arquitetural. Um estilo arquitetural define uma família de sistemas em um padrão de organização de estrutura. Em outras palavras, um estilo arquitetural define componentes e conectores que podem ser utilizados em uma instância de um membro desse estilo, com um conjunto de regras com relação a como eles podem ser combinados (RICHARDSON, 2018).

Existe uma variada gama de estilos arquiteturais, este projeto explora especificamente o estilo arquitetural baseado em microsserviços, sendo abordado na próxima seção.

2.1.1 Estilo aruitetural de microsserviços

Atualmente, os sistemas vivem um contexto no qual são exigidas algumas características-chave esperadas de um sistema e seus serviços: confiabilidade, escalabilidade e disponibilidade. Além disso, com o aumento da complexidade dos sistemas, houve um aumento do número de equipes para atender as demandas técnicas dos sistemas. Esse acréscimo traz desafios de organizar grandes equipes de desenvolvimento de software de maneira a otimizar o tempo de entrega de novas funcionalidades e oferecer manutenção aos sistemas.

Para atender a tais requisitos, o estilo arquitetural de microsserviços tem sido adotado (RICHARDSON, 2018). Um dos fatores que favorece a adoção desse estilo arquitetural nesse contexto é que, conforme afirma Fowler (2014), com microsserviços, o sistema é dividido em pequenas partes distribuídas e autônomas – os microsserviços – focadas em um Bounded Context.

Após apresentar os conceitos básicos de arquitetura de software e microsserviços, a próxima seção explora alguns conceitos relacionados à reatividade e concorrência.

2.2 Reatividade e Concorrência

Esta seção define e diferencia os conceitos para sistemas reativos e programação reativa. Ao final, explica do modelo de atores, utilizado pelo qual é muito relevante para o contexto deste projeto.

2.2.1 Sistemas Reativos

Segundo Merson (2020), adotar a orientação a eventos como meio de comunicação entre os serviços aumenta a escalabilidade e a taxa de transferência dos sistemas. Conforme o mesmo autor, serviços que enviam mensagens não necessitam ficar ociosos esperando por uma resposta, e uma mesma mensagem/evento

pode ser consumido por diversos outros serviços, se publicadas em um sistema de mensageria.

Para Bonér e Klang (2016), um sistema reativo é um estilo arquitetural que permite que múltiplas aplicações individuais se unam como uma única unidade, reagindo ao redor, enquanto se mantêm cientes sobre cada um dos elementos internos. Para isso, devem atender as seguintes características definidas no Manifesto Reativo, proposto por Bonér, Farley, Kuhn e Thompson (2014):

- "**Responsivos:** O sistema responde em um tempo razoável, se possível.[...] Sistemas responsivos visam prover tempos de resposta curtos e consistentes, estabelecendo margens de tolerância confiáveis que garantem uma qualidade de serviço consistente [...].
- **Resilientes:** O sistema continua responsivo em caso de falha [...]. Falhas são contidas dentro de cada componente, isolando-os uns dos outros, portanto, garantindo que partes do sistema podem falhar e se recuperar sem comprometer o sistema como um todo [...].
- **Elásticos:** O sistema continua responsivo mesmo sob variações de carga de demanda. Sistemas Reativos podem reagir a mudanças na taxa de solicitações por meio do aumento ou diminuição dos recursos alocados para lidar com essas entradas [...].
- **Orientados a Mensagens:** Sistemas Reativos baseiam-se em transferência de mensagens assíncronas para estabelecer fronteiras entre os componentes e garantir baixo acoplamento, isolamento, transparência na localização [...]. "

2.2.2 Programação reativa

Programação reativa é um subconjunto da programação assíncrona e um paradigma de programação, no qual a disponibilidade de novas informações orienta a lógica de computação interna dos componentes de uma aplicação. Esse paradigma propõe decompor um problema em etapas menores, em que cada uma pode ser executada de maneira assíncrona e sem bloqueios, e então serem combinadas para produzir um fluxo de trabalho (BONÉR; KLANG, 2016). Com isso, tal paradigma auxilia a lidar com um dos maiores obstáculos para a escalabilidade de aplicações, que é a contenção (BONÉR, 2017).

2.2.3 Modelo de atores

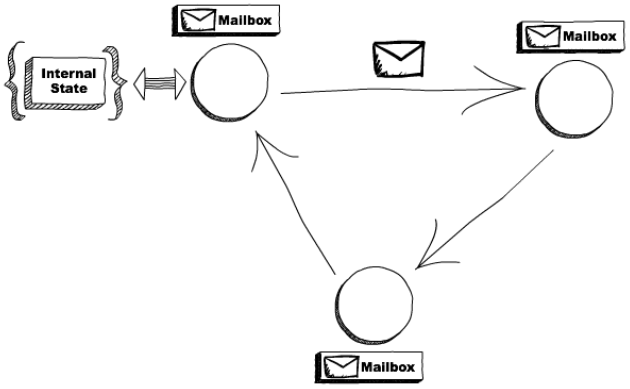
O modelo de atores é um modelo conceitual criado para lidar com a concorrência. Ele define algumas regras gerais sobre como os componentes de um sistema devem se comportar e interagir entre si.

O principal elemento desse modelo é o "ator". Ele é uma unidade primitiva de computação que recebe uma mensagem com alguma informação e aplica uma lógica de negócio. Além disso, eles não compartilham estado entre si.

A comunicação entre atores se dá por troca de mensagens. Quando um ator recebe uma mensagem, ele pode fazer uma das seguintes ações: Criar mais atores; Enviar outras mensagens para outros atores; Determinar o que fazer com a próxima mensagem.

Apesar de múltiplos atores poderem ser executados simultaneamente, cada ator só processa as mensagens direcionadas a ele de maneira sequencial. As mensagens são enviadas de maneira assíncrona, que precisam ser armazenadas enquanto o ator está processando alguma outra mensagem que veio antes. Essas mensagens são armazenadas no *mailbox*. A figura 1 ilustra o esquema geral de interação em um modelo de atores.

Figura 1: Funcionamento interno de um modelo de atores



Fonte: https://www.brianstorti.com/the-actor-model/

3 Proposta

Para Boner e Klang (2016), a programação reativa utilizada em contextos de comunicação síncrona faz com que se perca parte da resiliência, pois uma falha no fluxo de processamento interno faz com que o cliente da requisição tenha de lidar com a ela. Segundo os autores, os mecanismos de autocura presente nos sistemas reativos são condizentes para resolver este problema, pois o sistema se encarrega do tratamento da falha, sem ser necessário notificar o cliente.

Neste trabalho, serão analisadas as vantagens e desvantagens na implementação de sistemas reativos utilizando programação reativa em comparação com a sistemas que utilizam mecanismos síncronos para comunicação interna.

Para a análise, os experimentos planejados envolvem implementar utilizando programação reativa uma arquitetura de microsserviços reativa e comparar com uma implementação não reativa, e a partir dos dados obtidos destes experimentos, realizar uma série de comparações em dimensões ainda em discussão

4 Cronograma

Atividadade	**Mai**	**Jun**	**Jul**	**Agosto**	**Set**	**Out**	**Nov**	**Dez**
Estudo do Akka	x	x	x					
Planejamento e execução de experimentos			x	x	x	x		
Escrita da monografia						x	x	x

Referências

BASS, Len, CLEMENTS, Paul, KAZMAN, Rick. 2013. Software Architecture in Practice. Third Edition. Addison-Wesley, 2013.

RICHARDSON, Chris, Microservices Patterns, vol. 104, no. 25. 2018.

FOWLER, Martin, BoundedContext, 2014, disponível em: https://martinfowler.com/bliki/BoundedContext.html

MERSON, Principles for Microservice Design: Think IDEALS, Rather than SOLID, 2020, disponível em: https://www.infoq.com/articles/microservices-design-ideals/

BONÉR KLANG, Reactive programming vs. Reactive systems 2016, disponível em: https://www.oreilly.com/radar/reactive-programming-vs-reactive-systems/

BONÉR FARLEY KUHN THOMPSON, Reactive Manifesto, 2014, disponível em: https://www.reactivemanifesto.org/pt-BR

BONÉR, Jonas, Reactive Microsystems. 2017.