

A large, dark gray circular arrow graphic that curves around the central text, pointing from the top right towards the bottom right.

FullCycle



Full Cycle

Wesley Willians

Esse livro está à venda em <http://leanpub.com/fullcycle>

Essa versão foi publicada em 2021-10-15



Esse é um livro [Leanpub](#). A Leanpub dá poderes aos autores e editores a partir do processo de Publicação Lean. [Publicação Lean](#) é a ação de publicar um ebook em desenvolvimento com ferramentas leves e muitas iterações para conseguir feedbacks dos leitores, pivotar até que você tenha o livro ideal e então conseguir tração.

© 2021 Full Cycle Ltda

Conteúdo

Introdução	1
A mudança de perspectiva	1
Full Cycle Developers @Netflix	2
Devs com muitas responsabilidades	4
Times de plataforma	5
Você é Full Cycle	6
Introdução a Arquitetura de Software	7
Sustentabilidade no dia zero	7
O que é Arquitetura de Software	9
Sustentabilidade vs Arquitetura	11
Arquitetura vs Design de Software	12
O papel do Arquiteto(a) de software	14
Sistemas Monolíticos	17
Sistemas “tradicionais”	17
Restrições	18
Monolitos não são ruins	19
Deploy	20

CONTEÚDO

Necessidade de escala	21
Débitos técnicos	22

Introdução

A mudança de perspectiva

Foi em 2015 quando comecei ouvir com frequência o termo microsserviços. A cada palestra e apresentação de cases de grandes empresas, ficava mais evidente de que essa tendência iria perdurar por algum tempo.

Grandes empresas e unicórnios precisavam crescer rapidamente, gerar mais valor a cada dia, contratar mais pessoal e ter mais independência entre seus projetos. Por outro lado, quanto mais ouvia falar sobre microsserviços, também mais era evidente que utilizar tal arquitetura não era trivial. Muitas peças ainda precisavam se encaixar para que esse modelo pudesse se tornar algo mais *natural* nas organizações.

A complexidade de arquitetar, desenvolver, testar, realizar o deploy e monitorar uma única aplicação estava sendo multiplicada pelo número de microsserviços que cada empresa possuía. Apesar de muitos projetos serem pequenos, ainda assim, todos precisavam passar por essas etapas.

Com o número de sistemas crescendo exponencialmente, a área de operações dessas empresas também começou a colapsar. Profissionais que estavam acos-

tumados a receber demandas de devs para realizarem quatro deploys diários, tiveram suas rotinas alteradas para realizar quarenta ou quatrocentos.

O número de aplicações a serem monitoradas também foi se multiplicando, e conflitos entre pessoas desenvolvedoras e sysadmins se intensificaram. Estava muito claro que já estávamos em uma nova era. Uma era que não tinha mais volta.

Full Cycle Developers @Netflix

Em 17 de maio de 2018, alguns profissionais da Netflix que já possuíam anos de casa, também compartilharam suas dores e tentativas que vinham realizando desde 2012.

Naquela época, não muito diferente de outras organizações, eles possuíam papéis extremamente bem definidos para o ciclo de desenvolvimento de um software.

O número de aplicações a serem monitoradas também foi se multiplicando, e conflitos entre pessoas desenvolvedoras e sysadmins se intensificaram. Estava muito claro que já estávamos em uma nova era. Uma era que não tinha mais volta.

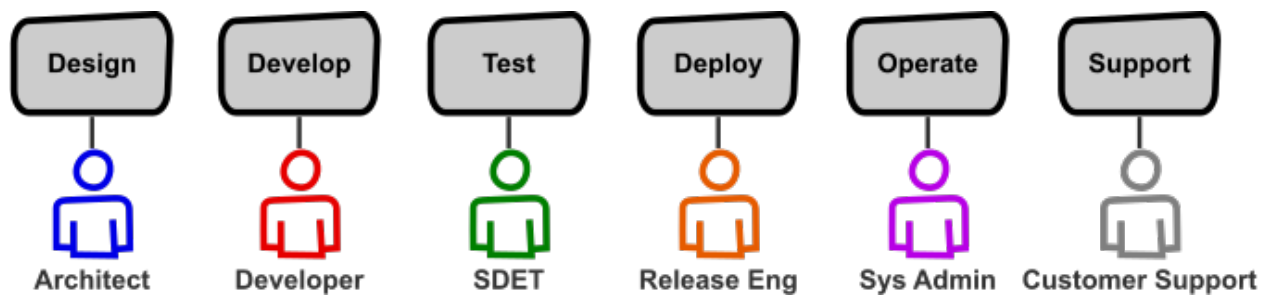


Imagem retirada do Tech Blog da Netflix.

A Netflix deixou de ser apenas um arquivo “war” e também foi dividida em microserviços.

Depois de muitos erros e acertos, perceberam que um dos principais pontos que sem dúvidas mudaria o jogo, seria transferir totalmente a responsabilidade de cada projeto para seus times de desenvolvimento, ou seja: agora os desenvolvedores fariam parte de todo o ciclo de desenvolvimento de suas aplicações. Da arquitetura ao deploy e monitoramento.

O grande lema se tornou: “*Operate what you build*”, ou opere o que você mesmo constrói. O raciocínio foi remover intermediários de todo processo e fazer com que a equipe de dev fique 100% responsável por seu microserviço sendo capaz de trabalhar com *feedbacks* curtos de todo processo e aprender rapidamente com isso.

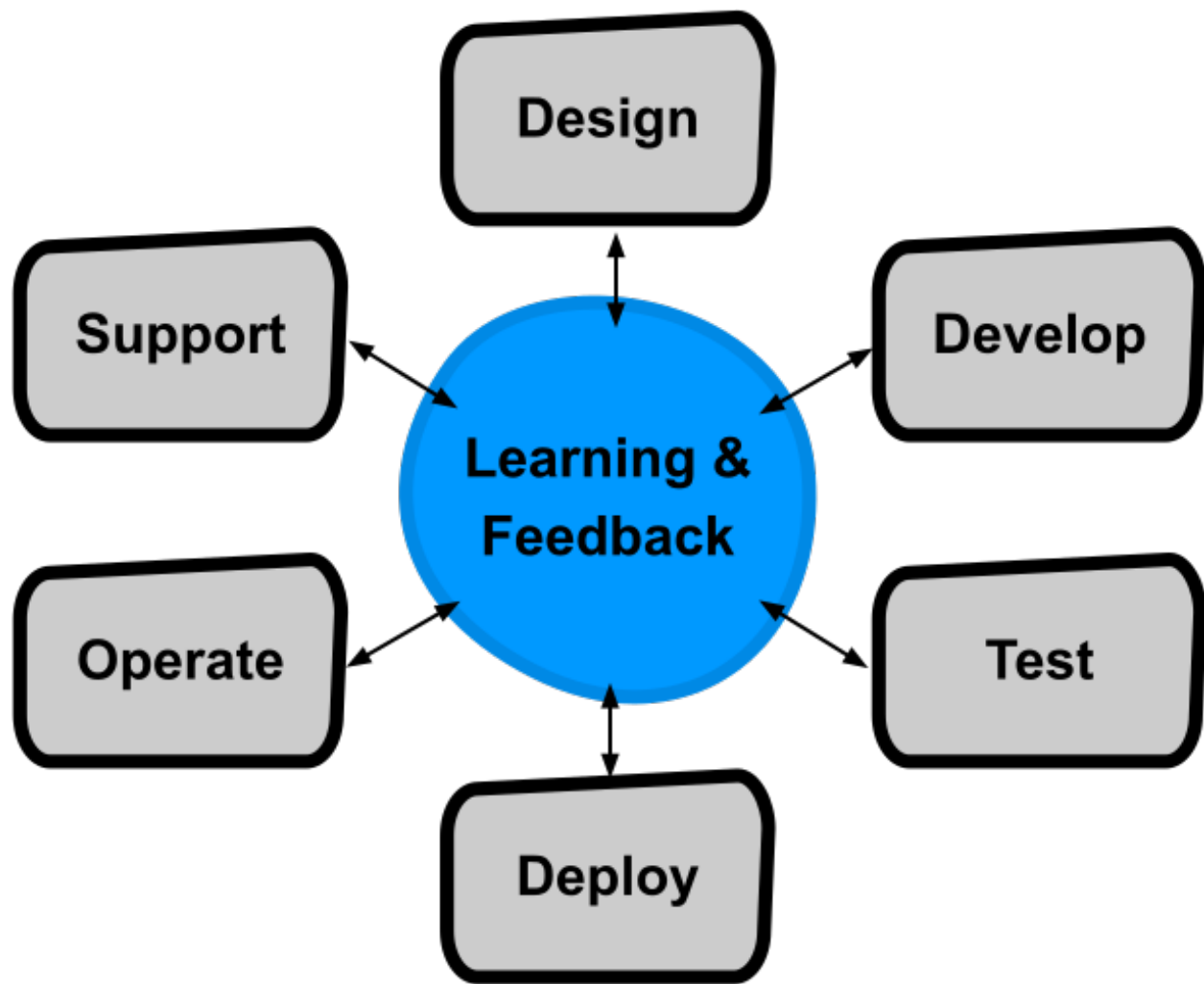


Imagem retirada do Tech Blog da Netflix.

Devs com muitas responsabilidades

Se você é uma pessoa desenvolvedora, com certeza já sabe o número imenso de responsabilidades que possui no dia a dia para entregar software com alta qualidade.

A grande questão é: colocar mais responsabilidades nas “costas” da pessoa

desenvolvedora é realmente a solução?

Foi exatamente essa pergunta que fiz para mim mesmo assim que estava na metade do artigo. Porém, ao continuar com a leitura percebi que os autores deram uma solução para minimizar esse fardo e ao mesmo empoderar devs cansados e estressados com tantos conflitos que resultavam em idas e vindas junto a área de operações para solucionar problemas em produção e para colocar suas aplicações no ar.

A solução criada estava na utilização de ferramentas desenvolvidas especificamente para que a pessoa desenvolvedora tivesse total autonomia para que realizar um deploy e entender em tempo real o comportamento de uma aplicação, de forma simples, rápida e que fundamentalmente fizesse parte do fluxo natural de trabalho.

Times de plataforma

Se você atualmente trabalha em alguma empresa que minimamente possui processos de desenvolvimento bem definidos, bem como possui o mínimo de uma cultura DevOps implementada, acredito que tudo que você leu até o momento representa a sua realidade de trabalho atual.

Grande parte das empresas já possui times de *plataforma*, que têm o objetivo de dar suporte e autonomia para as pessoas desenvolvedoras no dia a dia. Esses times criam ferramentas, padronizam pipelines exatamente para que o processo

de desenvolvimento no dia a dia seja fluido e produtivo, sem tirar o foco na entrega de valor pelos devs.

Você é Full Cycle

Se você participa do fluxo completo de uma aplicação, você é um Full Cycle Developer, todavia, como qualquer profissional de tecnologia, a evolução precisa ser constante; e esse é o objetivo desse livro: abranger os principais aspectos do mundo Full Cycle, e te tornar capaz de desenvolver aplicações de grande porte utilizando as tecnologias mais modernas do mercado onde quer que você esteja.

A partir de agora, vamos partir para uma jornada completa do *Software Development Life Cycle*. Vamos partir do mundo da arquitetura de software até o deploy e monitoramento de aplicações de grande porte.

Introdução a Arquitetura de Software

Um dos pontos fundamentais que sem dúvidas diferencia um desenvolvedor no mercado é o quão preparado ele está para desenvolver softwares sustentáveis, ou seja, aquele software que consegue ao longo do tempo gerar valor para uma organização. O software sustentável é flexível e permite grandes mudanças sem muita necessidade de reescrita.

Nesse capítulo gostaria essencialmente de fazer três “provocações” que ajudarão a você entender a real essência da Arquitetura de Software:

1. Qual a fórmula para criarmos softwares sustentáveis?
2. O que realmente é Arquitetura de Software?
3. Qual a relação entre a sustentabilidade de um software e a sua arquitetura?

Sustentabilidade no dia zero

Quando um executivo contrata uma empresa para desenvolver um software é porque ele possui uma “dor” que se resolvida fará com que a companhia chegue

mais próxima de seus objetivos. Por outro lado, o retorno sobre esse investimento muitas vezes pode demorar alguns meses ou eventualmente anos. Software bem feito não é barato e obviamente o mínimo que se espera é que ele se pague com o passar do tempo. O grande ponto, é que na maioria das vezes há a necessidade de fazer com que a solução acompanhe a evolução do negócio, e é exatamente nesse momento que muitas corporações acabam perdendo diversas oportunidades, pois, infelizmente, a cada dia que passa mais difícil é fazer o software evoluir.

Quando as empresas se dão conta de que determinado software chegou em seu limite da evolução, novos projetos de “modernização tecnológica” são criados exatamente para dar conta do que o software “legado” não conseguiu.

Com certeza se você já está no mercado há alguns anos, você já viu algum caso parecido com o citado acima. Agora, a pergunta que não quer calar: O software parou de evoluir realmente porque a tecnologia evoluiu MUITO ou porque a forma com que ele foi criado tornou sua evolução cada vez mais caótica?

Sem dúvidas a tecnologia evolui. Novas linguagens, frameworks, bibliotecas são criadas todos os dias, porém, em muitos casos, a forma com que um software é criado tem um peso infinitamente superior do que a tecnologia utilizada pelo mesmo.

O software precisa nascer sustentável e continuar em produção pelo maior tempo possível. Quanto mais tempo ele fica no ar, mais retorno ele gera para empresa e isso só é possível se sua base for bem feita. Só é possível se no

dia zero cada desenvolvedor, arquiteto, product owner, entre outros, de forma intencional, pensarem como fazer com que esse software fique no ar e entregue valor por diversos anos.

Obviamente o software precisa ser bem arquitetado, porém a grande pergunta é: O que realmente é arquitetura de software?

O que é Arquitetura de Software

Apesar de não haver apenas uma resposta válida sobre o que é arquitetura de software, podemos dizer que arquitetura software é a relação entre os objetivos de negócio e suas restrições com os componentes a serem criados e suas responsabilidades visando sua evolução.

“É a organização fundamental de um sistema e seus componentes, suas relações, seu ambiente, bem como os princípios que guiam seu design e evolução.” (IEEE Standard 1471)

O processo de arquitetar um software estabelece que o que está sendo desenvolvido faça parte de um conjunto maior.

Pilares da Arquitetura de Software

Agora que temos um ponto de partida sobre o que é arquitetura de software, é importante que consigamos separar alguns de seus conceitos em pilares para

melhor entendimento.

Organização

Quando falamos em arquitetura, falamos em organizar um sistema (não o software em si) que possibilite a fácil componentização, evolução, bem como um fluxo rico para que possamos atender os objetivos de negócio gerando um produto para o cliente final.

Componentização

Você não pode pensar em arquitetura de software deixando de lado seus componentes, pois é através da junção desses componentes que nós conseguimos atingir os atributos de qualidade do sistema.

Nós devemos pensar em todo o ecossistema que existe em torno do processo que nós pretendemos fazer até o resultado final, já que esses componentes serão usados em diversos momentos do nosso trabalho.

Grande parte do trabalho no mundo da arquitetura de software envolve dominar as formas de realizar a componentização dos processos para que eles operem com eficácia, evitando o retrabalho.

Relacionamento entre sistemas

Nenhum software é uma ilha.

A integração entre softwares a cada dia é mais comum. Logo, o software e seus componentes precisam estar preparados para que eles consigam fazer parte de uma cadeia de integração entre sistemas dentro de um processo maior.

Governança

Principalmente nas grandes empresas a vasta quantidade de sistemas e integrações são cada vez maiores. Muitas pessoas veem governança como uma forma de burocratizar o processo de desenvolvimento; por outro lado, na grande maioria das vezes, o software a ser desenvolvido é apenas mais um dentre as centenas de outros que já estão em operação. Nesse ponto, ter controle, guidelines, padrão no processo de documentação, entre outros, torna-se extremamente necessário para o crescimento organizado entre todo o ecossistema da organização.

Na governança também constam os times, tecnologias, banco de dados e as integrações. São os procedimentos que são utilizados para definir toda a solução que será provida. Essa parte engloba diversas definições que os arquitetos de software e de soluções encontram em suas abordagens.

Sustentabilidade vs Arquitetura

Começamos esse capítulo falando da real necessidade de termos softwares sustentáveis ao longo do tempo para que eles possam gerar valor para a

organização tendo assim possibilidade de se pagar e ainda contribuir de forma direta ou indireta para o crescimento negócio.

Sem dúvidas, uma das formas para fazer com que isso ocorra é trabalhar expressivamente no processo arquitetural. O software que é gerado sem visão de futuro está fadado a existir apenas no curto prazo, logo, há uma grande probabilidade de que a iniciativa que deveria melhorar os resultados da organização possa acabar tendo efeito reverso.

Trabalhar com arquitetura de software de forma intencional pode definir o sucesso ou o fracasso do projeto.

Lembre-se: o sucesso de um projeto de software não está na primeira entrega, mas sim nas evoluções subsequentes com o passar dos anos.

Arquitetura vs Design de Software

É muito comum ver profissionais substituírem a palavra arquitetura por design de software. A grande pergunta é: Arquitetura e design de software representam a mesma coisa?

A resposta é não! A arquitetura de software trata de conseguirmos alinhar os objetivos do negócio ao projeto a ser entregue. Decisões estritamente arquiteturais podem impactar desde a escolha entre Cloud Providers até mesmo a divisão de times e a quantidade de sistemas que uma organização precisará.

Decisões que impactam diretamente em quais componentes e *vendors* que um projeto utilizará são decisões arquiteturais.

Exemplo: Para que os logs de um sistema possam ser centralizados e facilmente recuperados, a Elastic Stack será utilizada.

Podemos perceber que tal decisão afetará a aplicação como um todo, além da possível contratação de mais infraestrutura para que a Elastic Stack seja instalada, ou mesmo, uma eventual contratação de um serviço gerenciado na Elastic Cloud.

Decisões como essa podem afetar todos os sistemas de uma organização, seu orçamento, o tempo em que um possível bug pode ser corrigido, o forma com que cada time trabalhará no dia a dia com observabilidade, além do conhecimento básico na stack que será requerida por cada desenvolvedor e eventuais treinamentos que os mesmos deverão receber para operar tais ferramentas.

Por outro lado, quando tomamos decisões de quais patterns GoF (Gang of Four) o projeto utilizará; SOLID, DRY, Clean Code, a quantidade de camadas de uma aplicação, estamos nos referindo fundamentalmente ao design do software.

O design de software possui seu escopo voltado as boas práticas no processo de desenvolvimento. Ele define diretamente padrões que normalmente são refletidos em linha de código, garantindo sua fácil manutenção, extensibilidade e evolução.

Muitas vezes decisões de design impactam diretamente em decisões arquite-

turais, como no exemplo citado sobre a Elastic Stack. Se uma aplicação foi desenhada com o intuito de centralizar seus logs e utilizará a Elastic Stack, tal decisão afetará o negócio, uma vez que tal ferramenta auxiliará no processo de recuperação de erros do software afetando assim sua disponibilidade.

O papel do Arquiteto(a) de software

Atualmente com a grande tendência de que pessoas desenvolvedoras sejam full cycle e de que as equipes sejam cada vez menores e com responsabilidades claras no processo de criação e manutenção de seus microsserviços, muitas organizações acabam questionando a necessidade da existência de um arquiteto ou arquiteta de software.

Apesar de ser cada vez mais natural de que as equipes tenham autonomia para tomar decisões de design e arquitetura, é muito evidente de a grande pressão do dia a dia faça que com o software tome caminhos arquiteturais diferentes do que se havia planejado para garantir sua sustentabilidade, bem como na garantia do atendimento dos atributos de negócio; por conta disso, um arquiteto ou arquiteta de software deve se fazer presente nos projetos.

Perfil profissional

Um arquiteto ou arquiteta normalmente possui uma grande bagagem como pessoa desenvolvedora e comumente é considerada *sênior*. Apesar de ter um

grande conhecimento técnico, esse tipo de profissional tem facilidade em “furar a bolha da *techniquês*” para entender melhor o negócio e o papel da solução tecnológica.

Entender a estrutura da organização, suas prioridades, disponibilidade financeira e do conhecimento do perfil profissional das equipes envolvidas nos projetos, o arquiteto(a) precisa ter a habilidade de balancear o que é melhor para a companhia naquele exato contexto.

Muitas vezes esse tipo de profissional pode ser extremamente criticado(a) pelas decisões tomadas, principalmente pela opção da não utilização das tecnologias mais modernas em determinados momentos, ou mesmo por decisões impopulares que se fazem necessárias.

É cada vez mais usual vermos arquiteto(a)s tendo a função de intermediar o processo de comunicação e conhecimento dos projetos entre diversos times e *stakeholders* diminuindo assim o ruído entre as partes para então viabilizar a implementação de decisões arquiteturais.

Um arquiteto(a) por time

Em muitos times e organizações não vemos formalmente o cargo ou título de um arquiteto(a). Em contrapartida, por conta de experiências anteriores, pessoas desenvolvedoras atribuídas em cargos liderança técnica, muitas vezes, acabam tomando decisões arquiteturais fazendo assim, informalmente, o papel

de arquiteto(a).

Área de arquitetura nas organizações

Algumas empresas, principalmente as que possuem muitas equipes e projetos em andamento, preferem criar um departamento específico de arquitetura de software. Os profissionais dessa área ficam responsáveis por revisar, aprovar e auxiliar as equipes e projetos a tomarem as melhores decisões alinhadas ao contexto atual da organização.

Além disso, a área de arquitetura monitora, gerencia os processos de mudança e garante a governança reforçando assim um padrão de qualidade em todos os projetos.

Apesar de muitos entenderem que áreas como essa podem burocratizar o processo de desenvolvimento e retirar a liberdade dos devs, é evidente que quando há uma grande quantidade de equipes, sistemas e tecnologias envolvidas é imperativo que haja controle para garantir a sustentação de todo ecossistema a médio e longo prazo.

Sistemas Monolíticos

Sistemas “tradicionais”

Nada melhor de que exemplos concretos para facilitar entendimento quando falamos em desenvolvimento de sistemas, logo, para começarmos esse capítulo, imagine um sistema de gerenciamento de produtos onde tais produtos podem ser categorizados e disponibilizados para venda. Também tal sistema permite que o usuário(a) possua uma área de busca, bem como o recurso de checkout para realizar a compra.

Com isso em mente, já é possível termos ideia de outras áreas que poderiam existir dentro desse sistema, como um catálogo para exibição dos produtos, evoluindo assim para uma loja virtual.

Podemos utilizar plataformas para desenvolver tal loja como um *Magento* por exemplo.

Essa é uma abordagem super comum principalmente devido a evolução de ferramentas, CMSs e frameworks. Todos os recursos fazem parte de uma mesma aplicação.

Esse é o clássico exemplo de um sistema monolítico, onde todas as operações acontecem em sua própria estrutura utilizando normalmente uma única linguagem de programação. Nele, são incorporadas todas as entidades, regras de negócio, disponibilização de APIs e integrações. Obviamente tudo está fortemente acoplado em um único sistema que normalmente possui mais de uma responsabilidade.

Restrições

Pela própria natureza dos sistemas monolíticos, normalmente eles utilizam uma única linguagem de programação. Por exemplo, se nós temos um sistema monolítico feito em Ruby, é improvável tenhamos alguma coisa escrita em PHP ou Python.

Em muitos casos essa possível “limitação” de utilizar uma única tecnologia dentro de um sistema pode fazer com que a organização não tire proveito de tecnologias, que para determinados casos de uso, sejam mais eficientes trazendo assim mais valor para o negócio como um todo.

Além disso, temos que concordar que ao colocarmos, cem, duzentas, ou mesmo mil pessoas para trabalhar na mesma base de código pode ser em determinadas situações algo caótico.

Monolitos não são ruins

Apesar das restrições citadas acima, trabalhar com sistemas monolíticos não é nenhum demérito ou atestado de obsolescência.

Sistemas monolíticos na maioria das vezes sem dúvidas é a melhor opção para grande parte das empresas. Afinal de contas, nem toda empresa possui 6000 devs como Mercado Livre.

Trabalhar com sistemas monolíticos reduz a complexidade e aumenta a eficiência principalmente em situações em que as próprias regras de negócio da empresa estão em constante mudança. Startups, empresas fazendo validações de modelos de negócio, bem como corporações que não fazem o uso “pesado” de tecnologia como principal ponto de sustentação, são grandes candidatas a trabalharem com sistemas monolíticos.

Via de regra, “todo” sistema deve começar com um monolito.

Martin Fowler em seu artigo [MonolithFirst](https://martinfowler.com/bliki/MonolithFirst.html)¹ faz duas grandes observações logo no início:

1. “Quase todas as histórias de sucesso de sistemas utilizando microsserviços começaram com um monolito que ficou grande e depois foi quebrado em partes.”

¹<https://martinfowler.com/bliki/MonolithFirst.html>

2. “Todos os casos que ouvi de sistemas que começaram diretamente utilizando microsserviços tiveram grandes problemas”

Para que uma companhia inicie o processo de desenvolvimento de seus sistemas utilizando uma abordagem diferente da monolítica, sem dúvidas os profissionais ali inseridos devem ter muitas cicatrizes com experiências de sucesso anteriores, caso contrário, há uma grande probabilidade do projeto fracassar.

Deploy

Sistemas monolíticos possuem uma característica singular quando precisam ir ao ar. Como tudo encontra-se dentro da mesma estrutura, por mais simples que seja determinada mudança, o deploy de 100% da aplicação precisa ser realizado.

É evidente que quando todas as áreas de uma empresa estão concentradas em um único sistema, o risco de interrupção de todas as áreas aumenta a cada deploy realizado. Em contrapartida, complexidades de comunicação entre diversos sistemas, a necessidade de criação de uma grande quantidade de pipelines de entrega, bem como todos os aspectos comuns em entregar e gerir projetos em produção são reduzidos drasticamente.

Necessidade de escala

Quando trabalhamos com grandes sistemas, invariavelmente teremos a necessidade de escala.

Segundo a definição da [Gartner](#)²:

“Escalabilidade é a medida da capacidade de um sistema de aumentar ou diminuir o desempenho e o custo em resposta às mudanças nas demandas de seus aplicativos e processamento.”

Nesse ponto, é evidente de que quando temos a necessidade de escalar um sistema monolítico, não há a possibilidade de escalarmos apenas as funcionalidades que naquele momento estão exigindo mais recursos computacionais, ou seja, todo sistema precisa escalar.

Por exemplo, se um ecommerce receber uma grande quantidade de acessos em seu catálogo de produtos e por consequência precisar de mais recursos computacionais para segurar a carga, todas as outras funcionalidades, que muitas vezes não precisariam ser escaladas, terão de escalar, pois tudo faz parte de um único conjunto.

Em determinados casos essa necessidade pontual de escala por um recurso pode fazer com que os custos com infraestrutura sejam elevados, pois claramente há uma ineficiência técnica embutida no processo.

²<https://www.gartner.com/en/information-technology/glossary/scalability>

Em contrapartida, em determinadas situações, não é porque um sistema precisa ser escalado por inteiro que seus custos serão mais elevados quando trabalhamos com outro tipo de arquitetura, como a de microsserviços por exemplo. Temos que ter em mente que quando trabalhamos com arquiteturas distribuídas, há também outros custos inerentes a essa modalidade.

Débitos técnicos

É evidente que qualquer tipo de aplicação, independente da arquitetura a ser utilizada em algum momento terá débitos técnicos, ou seja, pequenas melhorias, refatorações, implementações que deveriam ser realizadas e que foram “deixadas para depois”.

A medida em que os débitos técnicos se acumulam ao longo do tempo, a instabilidade do sistema como um todo acaba aumentando, gerando assim uma queda considerável na produtividade da manutenção e implementação de novos recursos.

Quando trabalhamos com sistemas monolíticos, isto acaba sendo potencializado, uma vez que há um alto acoplamento em todos os componentes da aplicação.

Por outro lado, isso não significa que pequenos sistemas não possuem ou não possuirão débitos técnicos. Todavia, como a base de código é limitada, tais débitos não terão tanta influência na solução como um todo.