**AND:**determina que se os *softgoals*descendentes forem satisfeitos os *softgoals*ascendentes serão satisfeitos.  
*∙***OR:**determina que, se algum *softgoal*descendente for satisfeito, o ascendente será  
satisfeito.

**MAKE(++):**fornece uma contribuição suficientemente positiva (*MAKE*) entre  
um *softgoal*descendente e um *softgoal*ascendente que é concebida no nível mais  
alto de satisfação. Dessa forma, ao utilizarmos *MAKE*, se o *softgoal*descendente for  
satisfeito o *softgoal*pai também será satisfeito.  
*∙***BREAK(- -):**fornece uma contribuição suficientemente negativa (*BREAK*) entre  
um *softgoal*descendente e um *softgoal*ascendente que é concebida no nível mais  
alto de negação. Portanto, ao utilizar *BREAK*, se o *softgoal*descendente for suficientemente satisfeito o *softgoal*pai será negado, ou seja não será satisfeito.  
*∙***HELP(+):**fornece uma contribuição parcialmente positiva entre um *softgoal*descendente e um *softgoal*ascendente. Dessa forma ao utilizar *HELP*, se o *softgoal*descendente for parcialmente satisfeito o *softgoal*ascendente será parcialmente satisfeito.  
*∙***HURT(-):**fornece uma contribuição parcialmente negativa entre um *softgoal*descendente e um *softgoal*ascendente. Dessa forma ao utilizar *HURT*, se o *softgoal*descendente for satisfeito o *softgoal*ascendente será parcialmente negado.  
*∙***UNKNOWN(?):**fornece uma contribuição desconhecida entre um *softgoal*descendente e um *softgoal*ascendente, podendo ser tanto positiva quanto negativa.  
*∙***EQUALS:**determina que o *softgoal*descendente só será satisfeito se o *softgoal*ascendente for satisfeito e que *softgoal*descendente será negado se o *softgoal*ascendente for negado.  
*∙***SOME:**é utilizada quando o sinal da contribuição é conhecido (positivo ou negativo), mas a extensão (parcial ou total) não é. Nesses casos, quando há alguma  
incerteza em se utilizar *HELP*ou *MAKE*deve-se utilizar o tipo de contribuição  
*SOME*+. Da mesma forma quando não há certeza em se utilizar *HURT*ou *BREAK*deve-se utilizar *SOME*-.

Para obter informações sobre os padrões, fizemos snowballing nos 15 artigos e buscamos no Google.

Referências:

1. Brown, K., & Woolf, B. (2016). Implementation patterns for microservices architectures. In *Proceedings of the 23rd Conference on Pattern Languages of Programs*. https://dl.acm.org/doi/abs/10.5555/3158161.3158170
2. Marquez, G., & Astudillo, H. (2018). Actual Use of Architectural in Microservices-Based Open Source Projects. *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, *2018*-*Decem*, 31–40. https://doi.org/10.1109/APSEC.2018.00017
3. Taibi, D., Lenarduzzi, V., & Pahl, C. (2019). *Continous Architecting with Microservices and DevOps :* (Vol. 1). Springer International Publishing. https://doi.org/10.1007/978-3-030-29193-8
4. <https://microservices.io/patterns/data/database-per-service.html>
5. Chelliah, P. R., J, H. S., Murali, A., & N, D. K. (2017). *Architectural Patterns : Uncover essential patterns in the most indispensable realm of enterprise architecture.*
6. <https://akfpartners.com/growth-blog/microservice-aggregator-pattern>
7. <https://docs.microsoft.com/en-us/azure/architecture/patterns/anti-corruption-layer>
8. Taibi, D., Lenarduzzi, V., & Pahl, C. (2018). Architectural patterns for microservices: A systematic mapping study. *CLOSER 2018 - Proceedings of the 8th International Conference on Cloud Computing and Services Science*, *2018*-*Janua*(Closer 2018), 221–232. https://doi.org/10.5220/0006798302210232
9. ~~Pacheco, V. F. (2018).~~ *~~Microservice Patterns and Best Practices~~*~~.~~
10. Lübke, D., Zimmermann, O., Pautasso, C., Zdun, U., & Stocker, M. (2019). Interface evolution patterns - Balancing compatibility and extensibility across service life cycles. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3361149.3361164
11. Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., Shen, J., & Babar, M. A. (2021). Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. *Information and Software Technology*, *131*, 106449. https://doi.org/10.1016/j.infsof.2020.106449
12. Tighilt, R., Abdellatif, M., Moha, N., Mili, H., Boussaidi, G. El, Privat, J., & Guéhéneuc, Y. G. (2020). On the Study of Microservices Antipatterns: A Catalog Proposal. *ACM International Conference Proceeding Series*, *July*. https://doi.org/10.1145/3424771.3424812
13. <https://patterns.arcitura.com/microservice-patterns/design_patterns/microservice_ambassador>
14. <https://microservices.io/patterns/communication-style/messaging.html>
15. <https://microservices.io/patterns/externalized-configuration.html>
16. Khomh, F., & Abtahizadeh, S. A. (2018). Understanding the impact of cloud patterns on performance and energy consumption. *Journal of Systems and Software*, *141*, 151–170. https://doi.org/10.1016/j.jss.2018.03.063
17. [~~https://microservices.io/patterns/data/shared-database.html~~](https://microservices.io/patterns/data/shared-database.html)
18. <https://docs.microsoft.com/en-us/azure/architecture/patterns/bulkhead>
19. <https://docs.microsoft.com/en-us/azure/architecture/patterns/competing-consumers>
20. Zimmermann, O., Stocker, M., Lübke, D., Pautasso, C., & Zdun, U. (2020). Introduction to microservice API patterns (MAP). *OpenAccess Series in Informatics*, *78*(4), 1–4. https://doi.org/10.4230/OASIcs.Microservices.2017-2019.4
21. <https://docs.microsoft.com/en-us/azure/architecture/patterns/pipes-and-filters>
22. <https://docs.microsoft.com/en-us/azure/architecture/patterns/priority-queue>
23. Balalaie, A., Heydarnoori, A., Jamshidi, P., Tamburri, D. A., & Lynn, T. (2018). Microservices migration patterns. *Software - Practice and Experience*, *48*(11), 2019–2042. https://doi.org/10.1002/spe.2608
24. <https://microservices.io/patterns/deployment/multiple-services-per-host.html>
25. <https://docs.microsoft.com/en-us/azure/architecture/patterns/strangler-fig>
26. https://microservices.io/patterns/deployment/single-service-per-host.html
27. <https://microservices.io/patterns/observability/application-logging.html>
28. <https://docs.microsoft.com/en-us/azure/architecture/patterns/federated-identity>
29. <https://docs.microsoft.com/en-us/azure/architecture/patterns/gatekeeper>
30. <https://microservices.io/patterns/data/event-driven-architecture.htmlhttps://microservices.io/patterns/data/event-driven-architecture.html>
31. <https://docs.cloudfoundry.org/devguide/deploy-apps/blue-green.html>
32. <https://martinfowler.com/bliki/CanaryRelease.html>
33. <https://microservices.io/patterns/data/cqrs.html>
34. <https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs>
35. <https://thoughtworks.github.io/pacto/patterns/cdc/>
36. <https://microservices.io/patterns/decomposition/decompose-by-business-capability.html>
37. [remoto](https://microservices.io/patterns/decomposition/decompose-by-subdomain.html)
38. Osses, F., Márquez, G., & Astudillo, H. (2018). An exploratory study of academic architectural tactics and patterns in microservices: A systematic literature review. *Avances En Ingenieria de Software a Nivel Iberoamericano, CIbSE 2018*, *February*, 71–84.
39. <https://docs.microsoft.com/en-us/azure/architecture/patterns/gateway-aggregation>
40. <https://docs.microsoft.com/en-us/azure/architecture/patterns/gateway-offloading>
41. <https://docs.microsoft.com/en-us/azure/architecture/patterns/gateway-routing>
42. <https://microservices.io/patterns/microservice-chassis.html>
43. <https://patterns.arcitura.com/microservice-patterns/design_patterns/container_sidecar>

https://dzone.com/articles/ms-chassis-pattern

*ok Adapter microservice [S16]*

*RNF principal:* Evolvability

Contexto: MSs precisam incorporar serviços existentes (p. ex., SOAP, JMS ou serviços baseados em mainframe), mas as APIs destes serviços não são consistentes com a arquitetura de MSs.

Problema: Como realizar a tradução destes serviços existentes em APIs de MSs?

Solução: Construir adaptadores (adapters) que convertem as APIs existentes para APIs que possam ser utilizadas pelos clientes dos MSs.

Este padrão é utilizado em casos em que os serviços existentes funcionam muito bem e não há necessidade de trocá-los, ou quando a mudança destes serviços pode prejudicar clientes existentes. Frequentemente é utilizado em conjunto com o padrão *Results Cache*, para reduzir o número de chamadas aos serviços legados.

Vantagens: Favorece a evolucionalidade, permitindo que sistemas sejam modernizados para MSs sem que seja necessário reimplementar todo o código legado.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Evolvability \cite{ID10} | ++ |

*Ok Aggregator [S17, S19]*

*RNF principal:* Agility

Contexto: Em um ambiente de MSs, vários serviços são responsáveis por tarefas específicas.

Problema: Como agregar o resultado de várias chamadas a diversos MSs, para servirem a uma funcionalidade de negócios maior ou mais complexa.

Solução: Implementar MSs agregadores, que são responsáveis por receber requisições, chamar outros MSs, combinar os resultados e então responder a requisição inicial [6].

Vantagens: Devido a simplicidade da API agregadora, facilita a implementação de novas funcionalidades que podem ser disponibilizadas em menor tempo.

Desvantagens: Pode aumentar a latência devido a implementação de uma nova camada intermediária.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Agility \cite{S19} | ++ |
| Performance \cite{S19} | - |

*Ok Anti-corruption layer [S20]*

*RNF principal:* Evolvability

Contexto: Em um ambiente de migração para MSs, podem existir funcionalidades legadas que não podem ser reimplementadas mas que ainda devem ser acessíveis pelos MSs, fazendo com que o novo sistema tenha que ser implementado utilizando as mesmas tecnologias legadas, dessa forma corrompendo a arquitetura de MSs.

Problema: Como fazer com que os MSs tenham acesso às funcionalidades legadas sem terem que utilizar as mesmas tecnologias do software legado?

Solução: Implementar uma camada anti-corrupção, com o objetivo de isolar a implementação legada dos MSs. Esta camada é responsável por receber as requisições dos MSs, fazer a tradução para o formato de mensagem esperado pelo sistema legado, posteriormente receber o retorno, traduzi-lo e envia-lo para os MSs.

Vantagens: Favorece a compatibilidade da nova arquitetura de MSs com sistemas legados. Permite isolar erros, auxiliando na resiliência dos MSs.

Desvantagens: A implementação da camada anti-corrupção requer a inclusão de novos serviços, podendo aumentar a latência devido a implementação dessa nova camada intermediária.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Evolvability \citeP{7} | + |
| Performance \citeP{S6} | - |

*Ok API-gateway [S2]*

*RNF principal:* Evolvability

Contexto: MSs provêm suas funções para outros serviços através de APIs. A criação de aplicações baseadas na composição de diferentes MSs requer um mecanismo de agregação *server-side*.

Problema: Como implementar um mecanismo de agregação de MSs *server-side*?

Solução: Implementar uma *API* *gateway*, responsável por ser o ponto de entrada que direciona as requisições para os MSs corretos, invocando múltiplos serviços, agregando resultados, transformando protocolos, cuidando da autenticação e limitando tráfego entre clientes e serviços.

Vantagens: Facilita a extensão do sistema através da customização de APIs. Permite manter a compatibilidade com sistemas legados, dessa forma favorecendo a evolução dos novos sistemas.

Desvantagens: A utilização deste padrão pode prejudicar a \textit{disponibilidade}, pois a utilização de um \textit{gateway} central de acesso para todas as requisições representa um risco, caso esse ponto apresente falha. A \textit{performance} dos serviços pode diminuir, devido a inclusão de novas operações de rede, como gerenciamento de segurança.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Portability \citeP{S2, S4} | ++ |
| Evolvability \citeP{S2, S4} | ++ |
| Availability \citeP{S2} | - |
| Performance \citeP{S2} | - |

*Ok Backend for frontend [S16]*

*RNF principal:* Evolvability

Contexto: Microsserviços que encapsulam funções no domínio de negócios não são claramente mapeados para necessidades específicas dos clientes. Por exemplo, os clientes podem ter diversos tipos de front-end, como mobile e web, cada um com necessidades específicas de acesso a funções e dados.

Problema: Como representar interfaces de serviço que sejam consistentes com a arquitetura de MSs mas que sejam adaptáveis às necessidades de cada tipo de cliente?

Solução: Construir diferentes interfaces customizadas para cada tipo de cliente, agregando somente o que é necessário para atendê-lo. Um BFF não deve conter nenhuma lógica de negócios, sendo geralmente desenvolvida pelo mesmo time que implementa a aplicação cliente. Pode utilizar o padrão *Page caches* para armazenar grandes quantidades de resultados e o padrão *Service registry*¸ para ser resiliente às mudanças de endereços de microsserviços utilizados pelos BFFs implementados.

Vantagens: Provê um endpoint menor e menos complexo, onde os clientes ficam desacoplados dos microsserviços, facilitando a troca destes serviços e diminuindo as mudanças no frontend. Previne a ocorrência de requisições concorrentes aos MSs e aumenta a autonomia do time de desenvolvimento de frontend \cite{ID6}. Fornece um grau de resiliência às aplicações, devido a possibilidade de isolamento de problemas no serviço BFF \cite{ID14}.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Compatibility \cite{ID6} | ++ |
| Maintainability \cite{ID6} | + |
| Performance \cite{ID6} | ++ |
| Modifiability \cite{ID8} | ++ |
| Evolvability \cite{ID3} | ++ |
| Resilience \cite{ID14} | + |

Ok Aggressive obsolescence *[S21]*

*RNF principal:* Evolvability

Contexto: Em um ambiente de MSs, onde é necessário manter várias versões de uma API para compatibilidade com diferentes clientes.

Problema: Como reduzir o esforço necessário para manter versões obsoletas de APIs?

Solução: Anunciar aos clientes uma data de encerramento do suporte à API.

Vantagens: Permite a evolução do sistema, estabelecendo uma data para que APIs obsoletas sejam removidas.

Desvantagens: Pode prejudicar a disponibilidade de clientes, caso algum deles ainda esteja utilizando uma API obsoleta na data em que seu suporte é encerrado.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Evolvability \cite{ID11} | ++ |
| Availability \citeP{ID11} | - |

*Ok API versioning [S21]*

*RNF principal:* Evolvability

Contexto: Uma API evolui e várias versões com melhorias são ofertadas. Em algum momento, as mudanças de uma nova versão não possuem compatibilidade com versões anteriores, prejudicando clientes que ainda utilizam estar versões.

Problema: Como um provedor de API pode indicar as capacidades atuais e a existência de possíveis incompatibilidades aos clientes?

Solução: Introduzir um indicador explícito de versão nas mensagens trocadas com os clientes. Este indicador possui números demonstrando o nível de comprometimento das alterações (p. ex. versão 2.3.1).

Vantagens: Permite que microsserviços sejam atualizados sem prejudicar clientes que ainda utilizam versões anteriores da API, os quais podem trocar para novas versões gradualmente.

Desvantagens: Pode prejudicar a \textit{agilidade} \citeP{S5} e a \textit{manutenibilidade} \citeP{S5}, devido à necessidade de desenvolvimento e manutenção de várias versões de um mesmo serviço.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Evolvability \cite{ID5} | ++ |
| Agility \citeP{S5} | - |
| Maintainability \citeP{S5} | - |

*Ok API description [S21]*

*RNF principal:* Evolvability

Contexto: Um provedor de serviços expõe uma ou mais operações em uma API, no entanto, sem documentar de que forma essas operações podem ser utilizadas.

Problema: Quais informações devem ser compartilhadas entre o provedor da API e seus clientes e de que forma?

Solução: Criar uma descrição da API que define as estruturas de mensagens de requisição e resposta, erros e conhecimentos técnicos relevantes para os clientes.

Vantagens: Comunica claramente aos clientes informações sobre operações, mensagens e erros, evitando ambiguidades de interpretação. Clientes sabem o que esperar das APIs e sobre sua evolução através do tempo.

Desvantagens: Pode aumentar os esforços de manutenção, devido a necessidade de constante atualização da descrição das APIs.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Evolvability \cite{ID11} | ++ |

Ok API Documentation and Management [11]

*RNF principal:* Maintainability

Contexto: No desenvolvimento de MSs, é necessário especificar APIs para realização de testes.

Problema: Como reduzir a complexidade de geração e gerenciamento das especificações de APIs para realização de testes?

Solução: Utilizar descrições de APIs para prover uma visão geral das funcionalidades dos microsserviços. Estas descrições são atualizadas conforme os serviços evoluem, facilitando o gerenciamento das APIs.

Vantagens: Favorece a testabilidade dos MSs, fornecendo informações para que desenvolvedores possam implementar testes, como por exemplo, testes de contrato.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Testability \cite{ID15} | + |

Ok Experimental preview *[S21]*

*RNF principal:* Evolvability

Contexto: No desenvolvimento de uma API, o provedor de serviços necessita liberar o seu uso de forma experimental, para que clientes possam iniciar a implementação de integrações. Mesmo assim, o provedor gostaria de poder realizar modificações na API de forma livre.

Problema: Como a API pode ser introduzida de forma menos arriscada e obtendo um feedback dos clientes?

Solução: Liberar acesso a API sem comprometimento com as funcionalidades oferecidas, estabilidade e longevidade, deixando isso claro para os clientes.

Vantagens: Clientes podem ter acesso prévio às APIs e influenciar no seu desenvolvimento, ao passo que provedores tem flexibilidade para responder rapidamente às mudanças indicadas.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Evolvability \cite{ID5} | ++ |

*Ok Limited lifetime guarantee [S21]*

*RNF principal:* Evolvability

Contexto: Um provedor de serviços publica uma API que é utilizada por clientes e pretende não fazer modificações que prejudiquem seus clientes, no entanto, deseja que em algum momento passa alterar sua API.

Problema: Como um provedor pode fazer com que seus clientes saibam até quando uma API será suportada?

Solução: Cada nova versão de uma API deve ser anotada com uma data limite para suporte. A partir desta data, o provedor poderá realizar alterações na API.

Vantagens: Facilita a evolução dos microsserviços, pois o suporte a versões antigas tem uma data específica para ser encerrado, favorecendo o desenvolvimento de versões desenvolvidas com tecnologias novas, sem necessidade de manter compatibilidade com APIs anteriores.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Evolvability \cite{ID5} | ++ |

*Ok Two in production [S21]*

*RNF principal:* Evolvability

Contexto: APIs evoluem e novas versões são melhoradas e oferecidas aos clientes. Em algum momento, as mudanças da nova versão não são mais compatíveis com versões anteriores, dessa forma prejudicando clientes que ainda não migraram para as versões mais novas.

Problema: Como um provedor pode gradualmente atualizar suas APIs sem prejudicar seus clientes, mas sem ter que manter um grande número de versões em produção?

Solução: Implantar e dar suporte a duas versões de uma API que provêm uma mesma funcionalidade, mas que não são compatíveis entre si. Dessa forma, clientes terão tempo para se adaptarem e migrarem para as novas versões.

Vantagens: Permite que microsserviços evoluam sem a necessidade de ficarem atrelados a versões anteriores. APIs obsoletas são gradualmente substituídas pelas novas versões.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Evolvability \cite{ID5} | ++ |

*Ok Semantic versioning [S21]*

*RNF principal:* Evolvability

Contexto: Quando se utiliza o padrão “API versioning” para versionamento das APIs, pode não ficar claro, ao observar o número da versão, o quão significantes são as alterações entre as diferentes versões.

Problema: Como os *stakeholders* podem comparar versões de APIs e detectar de forma imediata se são compatíveis?

Solução: Introduzir um esquema de versionamento de três números, x.y.z, permitindo ao provedor de uma API especificar diferentes níveis de alteração através da composição dos números identificadores, normalmente versões *major*, *minor* e *patch*.

Vantagens: Favorece a evolução das APIs, permitindo que novas versões sejam disponibilizadas e seus impactos nos clientes sejam percebidos de forma clara através dos níveis de alteração indicados no versionamento.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Evolvability \cite{ID5} | ++ |

*Ok Automated configuration [S5]*

*RNF principal:* Manageability

Contexto: Em uma arquitetura de MSs, a configuração das instâncias, serviços e hosts é feita manualmente pelos desenvolvedores.

Problema: Para cada MS, são feitas diferentes configurações para vários ambientes (desenvolvimento, teste e produção). A configuração manual leva mais tempo para ser realizada, sendo sujeita a erros. Como maximizar o tempo gasto no processo de configuração e evitar erros?

Solução: Utilização de servidores e serviços para automatizar o processo de configuração, através do uso de ferramentas de gerenciamento.

Vantagens: A automatização permite aos desenvolvedores gerenciar as configurações de forma centralizada, facilitando o compartilhamento de elementos comuns, mantendo a consistência de configuração entre diferentes serviços.

Desvantagens: A configuração manual torna mais fácil o desenvolvimento de MSs, no entanto, conforme os sistemas crescem, a automatização de configuração é necessária. Isso traz uma maior complexidade de desenvolvimento, devido a introdução de novas ferramentas e monitoramento para verificar se as configurações de cada serviço estão corretas.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Manageability \cite{ID5} | ++ |

*Ok Automating test procedure [S15]*

*RNF principal:* Maintainability

Contexto: No desenvolvimento de MSs, devem ser aplicados testes de integração entre serviços.

Problema: O processo manual de testes consome muito tempo, devido ao grande número de testes e quantidade de checagens de resultados. Como limitar a complexidade dos processos de testes?

Solução: Aplicação de procedimentos automáticos de teste, através do uso de ferramentas.

Vantagens: A utilização de ferramentas para automatização de testes promove a testabilidade dos MSs.

Desvantagens: A aplicação das ferramentas exige uma curva mínima de aprendizado, além de depender da geração de dados de monitoramento de MSs.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Testability \cite{ID15} | ++ |

*Ok CI/CD [S5]*

*RNF principal:* Agility

Contexto: A implantação independente possibilita processos de desenvolvimento e entrega contínua dos MS.

Problema: Como automatizar o gerenciamento dos processos de teste e implantação?

Solução: Utilizar ferramentas de integração e implantação para automatizar os processos de teste e implantação.

Vantagens: A integração do desenvolvimento e operação (DevOps), associado à entrega contínua, permite diminuir o tempo entre *releases* do sistema, mantendo a qualidade do software. Falhas no sistema podem ser identificadas de forma mais fácil. Testes e implantação se tornam mais eficientes devido à automação.

Desvantagens: Existe um aumento de esforço, tempo e custo necessário para adoção das ferramentas de CI/CD, além de que pode haver uma resistência organizacional.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Maintainability \citeP{S1, S6, S5} | + |
| Deployability \cite{ID5} | ++ |
| Manageability \citeP{S5} | ++ |
| Continous integration \citeP{S5} | ++ |
| Time to market reduction \citeP{S5} | + |
| Testability \citeP{S5} | ++ |
| Agility |  |

*Ok Centralized load balancing [S15]*

*RNF principal:* Performance

Contexto: Várias instâncias de MSs estão sendo executadas de forma independente. As requisições devem ser distribuídas entre essas instâncias de forma a maximizar a utilização dos serviços.

Problema: Como fazer a distribuição das requisições entre as várias instâncias para que nenhum deles fique significativamente sobrecarregado comparado aos outros?

Solução: Utilizar um servidor central para controle da carga de requisições enviadas aos MSs. A distribuição das requisições é baseada pela aplicação de algoritmos, como, por exemplo, Round-Robin. O servidor também pode verificar se algum serviço não está respondendo, reencaminhando as requisições para os que estão funcionando.

Vantagens: A utilização de balanceamento traz um aumento de performance, pois as requisições podem ser encaminhadas em paralelo para vários serviços, minimizando a sobrecarga e maximizando o tempo de resposta.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Performance \citeP{S15} | + |
| Availability \citeP{S1, S6} | ++ |

*Ok Distributed load balancing [S15]*

*RNF principal:* Performance

Contexto: Várias instâncias de MSs estão sendo executadas de forma independente. As requisições devem ser distribuídas entre essas instâncias de forma a maximizar a utilização dos serviços.

Problema: Como fazer a distribuição das requisições entre as várias instâncias para que nenhum deles fique significativamente sobrecarregado comparado aos outros?

Solução: Nesta solução, o balanceamento é feito pelo próprio cliente que necessita enviar uma requisição aos serviços. Este cliente deve manter uma lista de serviços, através da utilização de um serviço de descoberta de serviços. Dessa forma, o cliente sabe quem são os serviços disponíveis. Através da aplicação de algoritmos de balanceamento, o cliente pode decidir para qual instância uma requisição deve ser enviada.

Vantagens: A utilização de balanceamento traz um aumento de performance, pois as requisições podem ser encaminhadas em paralelo para vários serviços, minimizando a sobrecarga e maximizando o tempo de resposta.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Performance \citeP{S15} | + |
| Availability \citeP{S1, S6} | ++ |

*Ok Ambassador [S22]*

*RNF principal:* Performance

Contexto: MSs precisam acessar componentes compartilhados que realizam tarefas comuns, como monitoramento, log e auditoria.

Problema: Os componentes compartilhados não podem ser copiados de forma redundante para o ambiente do MS, pois são desenvolvidos de forma independente. Por outro lado, o acesso remoto a eles pode ser ineficiente.

Solução: Um container especial é criado para armazenar componentes utilitários, que farão a comunicação entre o MS e os componentes compartilhados, os quais são executados em instâncias remotas.

Vantagens: Melhora a performance no acesso aos componentes remotos, através da utilização dos componentes compartilhados que são executados no mesmo host que o serviço. Pode auxiliar na manutenibilidade, criando uma interface entre MSs e componentes compartilhados, os quais são mantidos por outras equipes, além de permitir isolar falhas, ajudando na resiliência dos MSs.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Maintainability \citeP{S6} | ++ |
| Resilience \citeP{S14} | ++ |
| Performance \cite{13} | + |

*Ok Hybrid [3]*

*RNF principal:* Evolvability

Contexto: Em um ambiente de MSs, é necessário haver comunicação entre os MSs.

Problema: Como implementar a comunicação entre MSs?

Solução: Implementar uma combinação entre os padrões *Service Registry* e *API-Gateway*, substituindo este último por um barramento para comunicação entre MSs. Clientes podem se comunicar com o barramento, o qual atua como um registrador de serviços, direcionando as requisições para os MSs.

Vantagens: Facilita a migração de sistemas em SOA, os quais já utilizam barramento para comunicação entre serviços.

Desvantagens: Um barramento único para comunicação pode afetar a disponibilidade de todo sistema, caso ocorra uma falha neste ponto de comunicação.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Evolvability \citeP{S2, S4} | + |
| Availability \citeP{S4} | - |

*Ok Asynchronous messaging [S23]*

*RNF principal:* Availability

Contexto: MSs precisam lidar com requisições recebidas de seus clientes e de outros serviços.

Problema: Como fazer a comunicação entre serviços em uma arquitetura de MSs?

Solução: Utilizar troca de mensagens assíncrona para comunicação entre serviços, através de canais de mensagem (*message brokers*).

Vantagens: Baixo acoplamento dos serviços em tempo de execução e aumento da disponibilidade dos serviços, devido aos *message brokers* fazerem buffer de mensagens até que o recebedor possa consumi-las.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Loose coupling \cite{14} | ++ |
| Availability \cite{14} | + |

*Ok Externalized configuration [S24]*

*RNF principal:* Portability

Contexto: Aplicações normalmente utilizam vários serviços externos, como processamento de pagamentos, e-mails, mensagens etc.

Problema: Como fazer com que serviços sejam executados em diferentes ambientes (p. ex., desenvolvimento, teste, produção) sem modificação?

Solução: Externalizar todas as configurações da aplicação, incluindo credenciais de conexão a banco de dados e endereços de rede. Na inicialização de um serviço, as configurações são lidas de uma fonte externa, como variáveis de ambiente ou arquivos de configuração.

Vantagens: A aplicação pode rodar em múltiplos ambientes sem modificações ou recompilações. A segurança é melhorada, pois o acesso às configurações fica externo aos serviços e pode ser restringido.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Security \citeP{S6, S7} | ++ |
| Portability \cite{15} | ++ |

*Ok Inconsistency handler [S15]*

*RNF principal:* Availability

Contexto: Para alcançar a disponibilidade, instâncias de um mesmo MS são executadas de forma distribuída. Estes serviços interagem em paralelo com um repositório de dados, utilizando a consistência eventual em detrimento da instantânea, onde uma atualização de dados é feita em somente uma das réplicas. Após a transação ter sido finalizada com sucesso, a alteração deve ser sincronizada com as demais. Durante o período em que a alteração não foi sincronizada, pode haver uma inconsistência eventual, onde um cliente que fizer uma consulta por aquele dado pode obter um resultado ainda não atualizado.

Problema: Como manter a disponibilidade e minimizar possíveis problemas de inconsistência dos dados?

Solução: Utilizar ferramentas que realizem a sincronização dos dados através do gerenciamento das mudanças de estado realizadas nos MSs. Os eventos são capturados em logs e então replicados aos demais MSs.

Vantagens: Permite manter a disponibilidade dos MSs através da distributividade, tratando os possíveis problemas de consistência dos dados replicados entre as instâncias.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Availability \citeP{S15} | ++ |

*Ok Database cluster [2, 3]*

*RNF principal:* Scalability

Contexto: MSs precisam persistir dados em um banco de dados.

Problema: Como construir uma arquitetura de banco de dados em MSs?

Solução: Armazenar dados em um cluster de banco de dados, que pode ser acessado por todos os MSs.

Vantagens: Aumenta a escalabilidade do sistema, permitindo que os bancos de dados sejam alocados para equipamentos dedicados. Este padrão é indicado para implementações com grande quantidade de tráfego de dados. Para manter a consistência dos dados, cada MS deve ter acesso a somente um subconjunto de tabelas pertinentes ao seu domínio de negócio /cite{ID4}.

Desvantagens: Pode levar a maiores dificuldades na gerenciabilidade \citeP{S2} e aumentar a complexidade de desenvolvimento \citeP{S2} inerente a uma arquitetura de \textit{cluster}. A disponibilidade \citeP{S2} pode ser afetada devido ao risco de falhas causado pela inclusão de um outro componente distribuído.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Performance \cite{ID4} | ++ |
| Scalability \cite{ID1, ID4} | ++ |
| Manageability \cite{ID2} | - |
| Agility \cite{ID2} | - |
| Availability \cite{ID2} | - |

*Ok Database per service [S18]*

*RNF principal:* Scalability

Contexto: MSs precisam persistir dados em um banco de dados.

Problema: Como construir uma arquitetura de banco de dados em MSs?

Solução: Manter um banco de dados privado para cada MS e acessível somente através de sua API. As transações de cada serviço devem envolver somente seu próprio banco de dados.

Vantagens: Um banco de dados pode ser facilmente escalado em um cluster em um segundo momento, caso necessário. Times de desenvolvimento podem trabalhar independentemente em cada serviço, alterando esquemas de bancos de dados sem afetar outros times. O acesso aos dados e esquemas por parte de outros MSs é impedida, melhorando a segurança e consistência dos dados /cite{ID4}. MSs podem utilizar tecnologias de persistência distintas /cite{ID5}.

Desvantagens: A \textit{performance} \citeP{S5} pode diminuir em situações em que é necessário implementar consultas que envolvem múltiplos MSs. A garantia da consistência dos dados entre as diferentes bases de dados também fica mais difícil, podendo afetar a \textit{confiabilidade} \citeP{S5}. A gerência de várias bases de dados também pode prejudicar a \textit{gerenciabilidade} \citeP{S5}.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Scalability \cite{ID4} | ++ |
| Independence \cite{ID4, ID5} | ++ |
| Security \cite{ID5} | + |
| Modularity \cite{ID4} | ++ |
| Maintainability \cite{ID5} | ++ |
| Evolvability \cite{ID5} | ++ |
| Changeability \cite{ID5} | ++ |
| Performance \cite{ID5} | - |
| Reliability \cite{ID5} | some- |
| Manageability \cite{ID2} | some- |

*Ok Local database proxy [S25]*

*RNF principal:* Scalability

Contexto: MSs precisam persistir dados em um banco de dados.

Problema: Como construir uma arquitetura de banco de dados em MSs?

Solução: Utilizar replicação entre as bases de dados através de um esquema *master/slave* e um proxy para direcionar as requisições. Operações de escrita são gerenciadas pelo master e replicadas para os nós slave, enquanto que as leituras são processadas pelos slaves. Cada MS deve utilizar um proxy local, o qual distribui a carga de trabalho entre master e slaves.

Vantagens: A utilização do esquema master/slave permite incluir e remover nós, provendo elasticidade em tempo de execução. A utilização de cache local e proxy para redirecionamento conforme a carga, melhora a performance dos serviços.

Desvantagens: Possui limitações quando é necessário escalar para operações de escrita.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Performance \cite{16} | ++ |
| Scalability \cite{16} | + |

Ok Local sharding-based router [S25]

*RNF principal:* Scalability

Contexto: MSs precisam persistir dados em um banco de dados.

Problema: Como construir uma arquitetura de banco de dados em MSs?

Solução: Dividir os dados em múltiplos bancos de dados, separando-os em grupos funcionais independentes, de forma a evitar operações *join* com outros bancos de dados. As requisições são processadas por um redirecionador local, que determina qual banco de dados é o mais adequado.

Vantagens: O sistema pode ser escalado através da inclusão de novos nós. O agrupamento funcional permite a escalabilidade tanto para operações de leitura quanto de escrita. A utilização de proxy para redirecionamento conforme a carga, associado ao balanceamento entre os fragmentos, melhora a performance dos serviços.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Performance \cite{16} | ++ |
| Scalability \cite{16} | ++ |

*Ok Shared database server [8]*

*RNF principal:* Evolvability

Contexto: MSs precisam persistir dados em um banco de dados.

Problema: Como construir uma arquitetura de banco de dados em MSs?

Solução: Armazenar dados em um único banco de dados, que pode ser acessado por todos os MSs.

Vantagens: Útil em situações de migração de sistema monolítico para MS, permitindo que o esquema de dados seja reutilizado na nova arquitetura sem grandes modificações.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Evolvability \citeP{S2, S4} | ++ |
| Independence \cite{17} | -- |

*Ok Server-side discovery [8]*

*RNF principal:* Availability

Contexto: Múltiplas instâncias de um mesmo MS são executadas em diferentes containers. A comunicação entre eles precisa ser definida de forma dinâmica e eficiente.

Problema: Como implementar a descoberta de serviços entre os MSs?

Solução: Implementar um serviço balanceador, responsável por receber as requisições dos clientes, e um serviço registrador, que faz a tarefa de dinamicamente transformar endereços DNS em IPs. O balanceador, ao receber uma requisição, solicita ao serviço registrador os endereços das instâncias disponíveis. O balanceador decide qual instância será utilizada para atender à requisição, devolvendo o endereço para o cliente, que se comunicará diretamente com o serviço.

Vantagens: Permite maior resiliência, através da detecção de problemas e reinício de serviços e facilita a migração, pois serviços legados podem ser substituídos simplesmente trocando-se o endereço no serviço de registros.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Performance \citeP{S2} | ++ |
| Availability \citeP{S2} | + |
| Resilience \citeP{S2} | + |
| Evolvability \citeP{S2} | + |

*Ok Client-side discovery [8]*

*RNF principal:* Availability

Contexto: Múltiplas instâncias de um mesmo MS são executadas em diferentes containers. A comunicação entre eles precisa ser definida de forma dinâmica e eficiente.

Problema: Como implementar a descoberta de serviços entre os MSs?

Solução: Implementar um serviço registrador, que faz a tarefa de dinamicamente transformar endereços DNS em IPs. Para se comunicar com um serviço, o cliente primeiro faz a solicitação a um serviço registrador para descobrir o endereço do serviço. Posteriormente, o cliente faz a requisição diretamente ao serviço desejado, podendo utilizar algoritmos de balanceamento para decidir qual serviço receberá a requisição.

Vantagens: Facilita a comunicação direta entre cliente e serviço. Permite maior resiliência, através da detecção de problemas e reinício de serviços e facilita a migração, pois serviços legados podem ser substituídos simplesmente trocando-se o endereço no serviço de registros.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Performance \citeP{S2} | ++ |
| Availability \citeP{S2} | + |
| Resilience \citeP{S2} | + |
| Evolvability \citeP{S2} | + |

*Ok Manual registry [S15]*

*RNF principal:* Availability

Contexto: MSs podem ter várias instâncias implantadas e a quantidade dessas instâncias pode mudar com o passar do tempo. A localização dos serviços em execução deve ser armazenada em um serviço central para que outros MSs possam consultar.

Problema: Como implementar o registro dos serviços em execução em um serviço central?

Solução: Implementar o registro dos serviços de forma manual, onde um usuário inclui as informações no serviço central.

Vantagens: Auxilia na disponibilidade dos serviços, pois, caso o usuário detecte uma falha em um MS, pode removê-lo do serviço de registro para que os clientes não o utilizem. No entanto, como isso é feito manualmente, pode demorar para que falhas sejam detectadas. Por outro lado, o registro manual facilita que MSs de outros sistemas sejam incluídos no serviço de registros conforme a necessidade dos usuários.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Availability \citeP{S15} | some+ |

*Ok Self-registry [S15]*

*RNF principal:* Availability

Contexto: MSs podem ter várias instâncias implantadas e a quantidade dessas instâncias pode mudar com o passar do tempo. A localização dos serviços em execução deve ser armazenada em um serviço central para que outros MSs possam consultar.

Problema: Como implementar o registro dos serviços em execução em um serviço central?

Solução: Cada MS pode se registrar ou se remover do registro no momento em que é inicializado ou desligado.

Vantagens: Devido ao registro automático e monitoramento dos serviços, a detecção de falhas ocorre com rapidez, evitando que MSs inválidos sejam acessados pelos clientes.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Availability \citeP{S15} | ++ |

*Ok Third-party registry [S15]*

*RNF principal:* Availability

Contexto: MSs podem ter várias instâncias implantadas e a quantidade dessas instâncias pode mudar com o passar do tempo. A localização dos serviços em execução deve ser armazenada em um serviço central para que outros MSs possam consultar.

Problema: Como implementar o registro dos serviços em execução em um serviço central?

Solução: Cada MS pode se registrar ou se remover do registro no momento em que é inicializado ou desligado. Essa tarefa é implementada através de componentes de terceiros, os quais são implantados na arquitetura de MSs.

Vantagens: Devido ao registro automático e monitoramento dos serviços, a detecção de falhas ocorre com rapidez, evitando que MSs inválidos sejam acessados pelos clientes.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Availability \citeP{S15} | ++ |

*Ok Bulkhead [S26]*

*RNF principal:* Availability

Contexto: Uma aplicação é composta por múltiplos MSs, cada um atendendo a um ou mais clientes. Uma carga excessiva ou uma falha em um serviço irá impactar todos os consumidores deste serviço.

Problema: Como evitar que a sobrecarga ou falha de um serviço impacte todos os clientes deste serviço?

Solução: Particionar as instâncias dos MSs em diferentes grupos, baseado na carga de consumo e requisitos de disponibilidade. Caso uma falha ocorra, ela pode ser isolada, causando impacto somente nos clientes daquele grupo de serviços. Dessa forma, é possível sustentar a funcionalidade do serviço para alguns consumidores que não estejam alocados àquele grupo.

Vantagens: Melhora a disponibilidade dos serviços, permitindo isolar falhas, mantendo o serviço em funcionamento nas demais instâncias alocadas em outros grupos.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Availability \citeP{S6, S14} | ++ |

*Ok Competing consumers [S27]*

*RNF principal:* Performance

Contexto: MSs devem lidar com um grande número de requisições. Uma forma comum para implementar o tratamento destas requisições é utilizar um sistema de mensagens, onde as requisições são enviadas por este canal de comunicação até os serviços consumidores, que farão o processamento da requisição e devolverão o resultado.

Problema: Como gerenciar as requisições recebidas de forma que cada requisição seja enviada para um serviço consumidor específico e de forma balanceada, evitando a sobrecarga dos consumidores?

Solução: Utilizar uma fila de mensagens para implementar um canal de comunicação entre a aplicação e as instâncias do serviço consumidor. As requisições são inseridas na fila e processadas pelos consumidores.

Vantagens: A confiabilidade é melhorada pois, caso ocorra uma falha de um serviço consumidor, a requisição retorna para a fila e pode ser processada por outra instância que esteja em funcionamento. Os serviços consumidores podem ser escalados dinamicamente conforme a carga de processamento requerida.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Availability \cite{19} | ++ |
| Performance \cite{S6. S7} | + |

*Ok Embedded entity [S12]*

*RNF principal:* Performance

Contexto: A informação requerida por um cliente de um serviço contém dados estruturados, cujos elementos se relacionam uns com os outros. Por exemplo, dados cadastrais de um consumidor podem estar relacionados com números de telefone, endereços, outros dados agregados etc. O cliente pode requerer que todos estes dados sejam retornados.

Problema: Como evitar que múltiplas requisições sejam enviadas à API para obter os diversos dados relacionados?

Solução: Fazer com que todas as informações relevantes sejam embutidas no retorno da requisição feita a API. Por exemplo, caso a API retorne os telefones de um consumidor, deve-se retornar também seus dados cadastrais.

Vantagens: Para retornar um dado e suas relações, menos requisições à API são necessárias, trazendo um ganho de performance, já que os dados estarão disponíveis mais rapidamente. No entanto, caso a quantidade de dados seja muito grande, pode demorar mais tempo para que sejam transferidos do serviço para o cliente.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Performance \citeP{S12} | + |

*Ok Linked information holder [S12]*

*RNF principal:* Performance

Contexto: A informação requerida por um cliente de um serviço contém dados estruturados, cujos elementos se relacionam uns com os outros. Por exemplo, dados cadastrais de um consumidor podem estar relacionados com números de telefone, endereços, outros dados agregados etc. O cliente pode requerer que todos estes dados sejam retornados.

Problema: Como evitar a transferência de grandes mensagens contendo diversos dados que nem sempre são requeridos pelo cliente?

Solução: Embutir, no retorno da requisição feita à API, links para todas as informações relacionadas. Por exemplo, caso a API retorne os telefones de um consumidor, deve-se retornar apenas um link para um endpoint que contenha os dados cadastrais. Dessa forma, caso o cliente do serviço necessite destes dados, é possível acessá-los.

Vantagens: A mensagem de retorno enviada pelo serviço é menor e utiliza menos recursos de comunicação, no entanto, são necessárias mais trocas de mensagens caso o cliente necessite de outros dados relacionados.

Desvantagens: Existe um esforço e custo maior de desenvolvimento, devido a necessidade de se implementar endpoints para retorno das informações relacionadas.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Performance \citeP{S12} | + |
| Cost reduction \cite{S12} | - |
| Agility \cite{S12} | - |

*Ok Page cache [S16]*

*RNF principal:* Performance

Contexto: Um sistema em MSs utiliza o padrão *backend for frontend* para construção de interfaces customizadas para clientes (p. ex., web ou mobile).

Problema: Um MS que representa uma entidade de negócios pode retornar informação em excesso, a qual não pode ser facilmente exibida, principalmente em aplicações mobile.

Solução: Utilizar o padrão *page cache* em conjunto com o padrão *backend for frontend*. Uma interface específica deve ser disponibilizada ao cliente, para que ele possa solicitar um conjunto limitado de uma coleção muito maior de dados. Os dados ficam armazenados em cache no serviço do BFF e podem ser solicitados pelo cliente, que vai consumindo-os conforme necessário.

Vantagens: Existe um ganho de performance do ponto de vista do cliente, pois menos tráfego é transportado pela rede. O cliente não precisa esperar o retorno da consulta do grande volume de dados, pois eles já estarão em cache no serviço do BFF. Além disso, menos dados chegam ao cliente e podem ser mais rapidamente exibidos.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Performance \citeP{S1, S6, S7} | + |

*Ok Pipes and filters [S28]*

*RNF principal:* Performance

Contexto: Uma aplicação deve executar várias tarefas de complexidades diferentes sobre uma informação de seu domínio. Uma abordagem seria implementar esse processamento em um módulo monolítico, no entanto, reduziria as oportunidades de refatoramento de código, otimização e reuso. Além disso, algumas tarefas são diferentes das outras, requerendo maior ou menor capacidade computacional.

Problema: Como implementar esse tipo de processamento de forma a maximizar as oportunidades de refatoramento de código, otimização e reuso?

Solução: Decompor as atividades de processamento em componentes (ou filtros), cada um executando uma única tarefa. Através da padronização do formato de dados que cada filtro recebe e envia, eles podem ser combinados em uma sequência de processamento (pipeline).

Vantagens: Evita a duplicação de código e facilita a inclusão e remoção de componentes. Filtros podem ser escalados e executados em paralelo, dando mais poder de processamento para componentes que mais necessitam, melhorando a performance. Caso um filtro falhe, o processamento pode ser reagendado ou então outra instância do filtro pode executar o processamento, aumentando a disponibilidade do sistema.

Desvantagens: Sua implementação pode trazer mais complexidade, principalmente se os filtros estiverem distribuídos em diferentes servidores.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Maintainability \citeP{S6, S7} | + |
| Availability \citeP{S6, S7} | ++ |
| Agility \cite{21} | - |
| Performance \cite{21} | + |

*OK Priority queue [S29]*

*RNF principal:* Performance

Contexto: Aplicações podem delegar tarefas específicas para outros serviços. Para isso, são utilizadas filas de mensagens que devem ser processadas em segundo plano. Normalmente, a ordem em que essas mensagens são recebidas não é importante, mas em outros casos é necessário priorizar requisições específicas.

Problema: Como priorizar requisições em uma fila de mensagens?

Solução: Utilizar uma fila de mensagens com atribuição de prioridade, para que a ordenação das mensagens a serem processadas seja feita pela ordem de prioridade e não pela ordem de chegada.

Vantagens: Permite priorizar disponibilidade e performance para clientes específicos, permitindo oferecer níveis diferentes de serviços conforme a necessidade.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Performance \cite{22} | + |
| Availability \cite{22} | + |

*OK Reactive auto-scaling [S15]*

*RNF principal:* Scalability

Contexto: Serviços precisam ser escalados de forma horizontal, ou seja, aumentar o número de instâncias executando um mesmo serviço.

Problema: Como escalar horizontalmente os serviços?

Solução: Utilizar uma abordagem reativa de escala, ou seja, monitorar métricas e, através de limiares, decidir se um MS deve ter o número de instâncias aumentado ou diminuído. Por exemplo, pode-se monitorar os valores de utilização de CPU, latência e carga de rede ou utilizar métricas de filas de mensagens. Caso um valor extrapole um limiar superior, o próprio sistema de gerenciamento deve aumentar as instâncias, ou, caso ultrapasse um limiar inferior, é feita a diminuição.

Vantagens: Permite que o sistema seja escalado de forma automática.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Scalability \cite{S15} | ++ |

*OK Proactive auto-scaling [S15]*

*RNF principal:* Scalability

Contexto: Serviços precisam ser escalados de forma horizontal, ou seja, é necessário aumentar o número de instâncias executando um mesmo serviço.

Problema: Como escalar horizontalmente os serviços?

Solução: Utilizar uma abordagem proativa de escala, escalando o sistema baseado em algum processo de previsão de carga, como, por exemplo, o histórico de requisições para cada MS ou pela conversão das requisições em quantidade de CPU e memória utilizada.

Vantagens: Permite que o sistema seja escalado de forma automática, no entanto, é necessário que o sistema esteja rodando a algum tempo para que existam dados históricos para realizar a previsão.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Scalability \cite{S15} | ++ |

*Ok Results cache [S16]*

*RNF principal:* Performance

Contexto: Fazer chamadas a serviços remotos pode ser custoso, devido a latência e tempo de processamento nos serviços.

Problema: Como melhorar a performance quando é necessário fazer repetidas chamadas aos serviços?

Solução: Utilizar um cache local de resultados das chamadas, diminuindo a necessidade de chamar várias vezes um serviço remoto. Para que os dados não se tornem obsoletos, deve-se usar um tempo de cache baixo.

Vantagens: Melhora a performance do serviço que faz a chamada a outro serviço remoto, diminuindo o número de requisições, permitindo que as operações sejam feitas com os dados armazenados localmente.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Performance \cite{S1, S6, S7} | + |

*OK Scalable store [S16]*

*RNF principal:* Scalability

Contexto: Em um sistema baseado em MSs, é necessário persistir estados para representar as interações anteriores e atuais dos usuários.

Problema: Como representar a persistência de estados em uma aplicação de MSs?

Solução: Persistir os estados em uma área de armazenamento escalável, disponível e compartilhada por qualquer número de aplicações. Pode-se utilizar, por exemplo, bancos de dados relacionais ou “NoSQL”. Os padrões *results cache* e *pages cache* geralmente utilizam o padrão *Scalable store*.

Vantagens: Permite a escalabilidade da área de armazenamento compartilhada, desta forma aumentando a disponibilidade dos serviços.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Scalability \cite{S1} | ++ |
| Availability \cite{S1} | ++ |

*Ok Edge server [S30]*

*RNF principal:* Evolvability

Contexto: Um sistema legado foi decomposto em MSs.

Problema: Como esconder dos sistemas clientes a complexidade da estrutura interna dos MSs? Como o status e uso dos serviços pode ser monitorado?

Solução: Através da inclusão de uma nova camada ao sistema, atuando como uma porta de entrada para os serviços. Esta camada é responsável por fazer um roteamento dinâmico das requisições recebidas aos MSs disponíveis. Dessa forma, mudanças internas na estrutura dos serviços não afetam os clientes, sendo possível também interceptar e monitorar todo o tráfego. Para evitar que a nova camada seja um ponto único de falha, deve ser utilizado mecanismos de balanceamento de carga.

Vantagens: Permite que a estrutura dos serviços seja alterada sem causar tanto impacto nos clientes. Por ser um ponto único de entrada, facilita a monitorabilidade dos MSs.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Evolvability \cite{23} | ++ |
| Monitorability \cite{23} | ++ |

*Ok Multiple service per host [S31]*

*RNF principal:* Deployability

Contexto: Em um sistema em MSs, cada serviço é implantado como um conjunto de instâncias de serviço, objetivando aumentar o desempenho e disponibilidade.

Problema: Como empacotar e implantar os serviços?

Solução: Executar múltiplas instâncias de serviços em um mesmo servidor.

Vantagens: Facilita a escalabilidade devido a possibilidade de implantar várias instâncias em um mesmo servidor. Melhor utilização dos recursos disponíveis em cada servidor.

Desvantagens: Dificulta o monitoramento e rastreabilidade dos serviços, devido a ter várias instâncias rodando no mesmo servidor e utilizando os mesmos recursos.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Scalability \cite{S2, S4} | ++ |
| Monitorability \cite{24} | - |

*Ok Single service per host [S33]*

*RNF principal:* Deployability

Contexto: Em um sistema em MSs, cada serviço é implantado como um conjunto de instâncias de serviço, objetivando aumentar o desempenho e disponibilidade.

Problema: Como empacotar e implantar os serviços?

Solução: Executar cada instância de serviço em servidores separados.

Vantagens: Permite isolar completamente os serviços, reduzindo a possibilidade de conflito de recursos. Facilita o monitoramento dos recursos utilizados pelos serviços devido a cada instância executar um único serviço.

Desvantagens: Reduz a eficiência na utilização de recursos, já que é necessário mais hosts, um para cada serviço. A escalabilidade pode ser prejudicada, pois, para escalar um serviço, é necessário escalar o host inteiro, utilizando mais recursos.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Independency \cite{S2, S4, S5} | ++ |
| Monitorability \cite{24} | + |
| Resources efficiency \cite{24} | - |
| Scalability \cite{S2} | - |
| Deployability \cite{S2} | ++ |

*Ok Containerization [S1]*

*RNF principal:* Deployability

Contexto: Em um sistema em MSs, cada serviço é implantado como um conjunto de instâncias de serviço.

Problema: Como simplificar a implantação dos serviços?

Solução: Utilizar containers para implantação, os quais incluem tudo que é necessário para um MS ser executado, como bibliotecas e dados.

Vantagens: Apresenta uma melhor performance na comunicação com a máquina hospedeira. Permite melhor aproveitamento dos recursos disponíveis, reduzindo custos de infraestrutura. A escalabilidade das instâncias é facilitada devido a característica mais leve dos containers. O isolamento de cada instância favorece a segurança e a independência dos serviços.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Scalability \cite{S1} | + |
| Independency \cite{S9} | ++ |
| Reliability \cite{S6} | ++ |
| Performance \cite{S15} | + |
| Security \cite{S10} | ++ |
| Availability \cite{S1} | ++ |
| Deployability \cite{S1} | ++ |

*OK Service-side Circuit Breaker [S15]*

*RNF principal:* Availability

Contexto: Em caso de falha, MSs devem ficar indisponíveis para novas requisições.

Problema: Como evitar que MSs continuem recebendo requisições em caso de falha?

Solução: Implementar um proxy para interceptar requisições entre serviços e evitar que um serviço receba requisições caso esteja com problemas. No caso do padrão service-side, este serviço roda no mesmo servidor que o MSs, interceptando todas as requisições que chegam ao serviço e bloqueando-as caso o MS esteja indisponível.

Vantagens: MSs podem trabalhar em conjunto, causando dependências entre eles. Este padrão minimiza efeitos em cascata causados pela falha de um dos serviços na cadeia de dependência, auxiliando na disponibilidade dos MSs.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Availability \cite{S15} | ++ |

*Ok Client-side Circuit Breaker [S15]*

*RNF principal:* Availability

Contexto: Em caso de falha, MSs devem ficar indisponíveis para novas requisições.

Problema: Como evitar que MSs continuem recebendo requisições em caso de falha?

Solução: Implementar um proxy para interceptar requisições entre serviços e evitar que um serviço receba requisições caso esteja com problemas. No caso do padrão client-side, este serviço roda no mesmo servidor que o cliente dos demais MSs, interceptando todas as requisições e bloqueando as que são enviadas para serviços indisponíveis.

Vantagens: MSs podem trabalhar em conjunto, causando dependências entre eles. Este padrão minimiza efeitos em cascata causados pela falha de um dos serviços na cadeia de dependência, auxiliando na disponibilidade dos MSs.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Availability \cite{S15} | ++ |

*Ok Proxy Circuit Breaker [S15]*

*RNF principal:* Availability

Contexto: Em caso de falha, MSs devem ficar indisponíveis para novas requisições.

Problema: Como evitar que MSs continuem recebendo requisições em caso de falha?

Solução: Implementar um proxy para interceptar requisições entre serviços e evitar que um serviço receba requisições caso esteja com problemas. No caso do padrão proxy, este serviço é executado como um serviço separado, que fica entre o cliente e o MS, interceptando todas as requisições e bloqueando as que são enviadas para serviços indisponíveis.

Vantagens: MSs podem trabalhar em conjunto, causando dependências entre eles. Este padrão minimiza efeitos em cascata causados pela falha de um dos serviços na cadeia de dependência, auxiliando na disponibilidade dos MSs.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Availability \cite{S15} | ++ |

*Ok Contributor Coupling [S15]*

*RNF principal:* Modularity

Contexto: Na migração de um sistema monolítico para MS, o sistema deve ser decomposto em pequenos serviços, buscando-se a correta granularidade dos serviços, objetivando uma alta coesão e baixo acoplamento.

Problema: Como obter uma alta coesão e baixo acoplamento na decomposição de sistemas monolíticos para MSs?

Solução: Fazer a divisão dos serviços baseando-se em aspectos dos times de desenvolvimento. Por exemplo, módulos do sistema monolítico que giram em torno das mesmas capacidades de negócio e que são desenvolvidos por um mesmo grupo de pessoas poderiam ser colocados em um mesmo MS, dessa forma diminuindo a necessidade de comunicação externa e maximizando a comunicação interna e a coesão dentro do time de desenvolvimento.

Vantagens: A correta decomposição dos MSs melhora a escalabilidade do sistema, a independência dos serviços e a sua implantação, dando mais agilidade ao processo de desenvolvimento. A manutenibilidade é melhorada devido à coesão dos serviços e ao baixo acoplamento.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Scalability \cite{S15} | ++ |
| Independency \cite{S15} | ++ |
| Deployability \cite{S15} | ++ |
| Maintainability \cite{S15} | + |
| Modularity \cite{S15} | ++ |

*OK Logical coupling [S15]*

*RNF principal:* Modularity

Contexto: Na migração de um sistema monolítico para MS, o sistema deve ser decomposto em pequenos serviços, buscando-se a correta granularidade dos serviços, objetivando uma alta coesão e baixo acoplamento.

Problema: Como obter uma alta coesão e baixo acoplamento na decomposição de sistemas monolíticos para MSs?

Solução: Fazer a divisão dos serviços baseando-se nos princípios da responsabilidade única dos MSs, ou seja, elementos de software que são alterados pela mesma razão devem ser mantidos unidos. Dessa forma, os limites de cada serviço são determinados pelas suas responsabilidades ou pelas suas mudanças de comportamento.

Vantagens: A correta decomposição dos MSs melhora a escalabilidade do sistema, a independência dos serviços e a sua implantação, dando mais agilidade ao processo de desenvolvimento. A manutenibilidade é melhorada devido à coesão dos serviços e ao baixo acoplamento.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Scalability \cite{S15} | ++ |
| Independency \cite{S15} | ++ |
| Deployability \cite{S15} | ++ |
| Maintainability \cite{S15} | + |
| Modularity \cite{S15} | ++ |

*Ok Semantic coupling [S15]*

*RNF principal:* Modularity

Contexto: Na migração de um sistema monolítico para MS, o sistema deve ser decomposto em pequenos serviços, buscando-se a correta granularidade dos serviços, objetivando uma alta coesão e baixo acoplamento.

Problema: Como obter uma alta coesão e baixo acoplamento na decomposição de sistemas monolíticos para MSs?

Solução: Fazer a divisão dos serviços baseando-se nas noções de limites de contexto originárias do DDD (Domain-driven Design). Os limites de cada MS são determinados através do exame de conteúdo e semântica do código fonte do sistema.

Vantagens: A correta decomposição dos MSs melhora a escalabilidade do sistema, a independência dos serviços e a sua implantação, dando mais agilidade ao processo de desenvolvimento. A manutenibilidade é melhorada devido à coesão dos serviços e ao baixo acoplamento.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Scalability \cite{S15} | ++ |
| Independency \cite{S15} | ++ |
| Deployability \cite{S15} | ++ |
| Maintainability \cite{S15} | + |
| Modularity \cite{S15} | ++ |

*Ok Strangler [S32]*

*RNF principal:* Evolvability

Contexto: À medida que os sistemas envelhecem, as tecnologias utilizadas para seu desenvolvimento e sua arquitetura tendem a se tornar obsoletas. Conforme novas funcionalidades são adicionadas, sua complexidade aumenta, tornando a manutenção ou inclusão de novas funcionalidades bastante difícil. Buscando modernizar seus sistemas, organizações buscam substituí-los, utilizando a arquitetura de MSs. No entanto, a completa substituição do sistema legado é uma tarefa bastante complexa.

Problema: Como fazer a substituição de um sistema legado por um novo sistema desenvolvido em MS?

Solução: Fazer uma substituição incremental de funcionalidades específicas do sistema legado para MSs. Criar uma interface que intercepta requisições vindas do sistema legado e encaminha para a aplicação legada ou para os novos serviços. Os clientes podem continuar utilizando a interface, e a migração pode ocorrer de forma transparente, até que se substitua todo o sistema antigo.

Vantagens: Permite a evolucionalidade do sistema legado, que pode ser migrado para uma nova arquitetura, além de manter a compatibilidade com o código antigo até que tenha sido migrado.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Evolvability \cite{S3} | ++ |
| Compatibility \cite{S3} | ++ |

*Ok Aggregation processing of monitored data [S15]*

*RNF principal:* Monitorability

Contexto: Diferentes tipos de dados sobre os MSs foram coletados através do monitoramento dos serviços.

Problema: Como processar e analisar os dados de monitoramento coletados?

Solução: Armazenar os dados coletados em formato agregado, reduzindo, dessa forma, o espaço de armazenamento necessário.

Vantagens: Com a redução do espaço de armazenamento, os dados agregados podem ser armazenados por um longo prazo.

Desvantagens: Informações mais detalhadas são perdidas ao armazenar em formato agregado.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Monitorability \cite{S15} | ++ |

*Ok None-aggregation processing of monitored data [S15]*

*RNF principal:* Monitorability

Contexto: Diferentes tipos de dados sobre os MSs foram coletados através do monitoramento dos serviços.

Problema: Como processar e analisar os dados de monitoramento coletados?

Solução: Armazenar os dados coletados como foram coletados, ou seja, de forma detalhada.

Vantagens: Tendo os dados de log no formato em que foram coletados, é possível fazer análises mais detalhadas.

Desvantagens: O formato detalhado aumento muito a necessidade de espaço de armazenamento, impossibilitando a guarda por longos períodos.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Monitorability \cite{S15} | ++ |

*Ok Application metrics [S5]*

*RNF principal:* Performance

Contexto: Sistemas desenvolvidos em MSs são frequentemente sujeitos a contratos de garantia de nível de serviço. Dessa forma, é crucial o monitoramento de sua performance.

Problema: Como fazer o monitoramento da performance dos MSs?

Solução: Utilizar ferramentas de monitoramento que permitam coletar dados e estatísticas sobre performance e falhas de cada MS pertencente a um sistema.

Vantagens: O monitoramento de todos os aspectos dos MSs possibilita um entendimento completo sobre o comportamento de uma aplicação, provendo dados que permitem melhorar a performance e qualidade dos serviços.

Desvantagens: A utilização de ferramentas e infraestrutura de monitoramento pode requerer o refatoramento dos serviços e a unificação de logs de serviços heterogêneos, levando a um aumento na complexidade de desenvolvimento.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Performance \cite{S5} | + |
| Agility | - |

*Ok Centralized monitor [S15]*

*RNF principal:* Availability

Contexto: Em um ambiente de MSs, é necessário monitorar os serviços para detectar ou antecipar falhas, objetivando tomar ações para que o sistema se recupere.

Problema: Como fazer o monitoramento dos serviços?

Solução: Utilização de um serviço central, o qual é responsável por coletar e atualizar informações sobre a disponibilidade dos demais serviços e atualizar o status de sua disponibilidade no registrador de serviços (service registry).

Vantagens: Através da checagem de falhas, é possível evitar que requisições sejam encaminhadas para o serviço até que ele se recupere da falha.

Desvantagens: O ponto central de monitoramento pode se tornar um ponto único de falhas.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Availability \cite{S15} | + |

*Ok Symmetric monitor [S15]*

*RNF principal:* Availability

Contexto: Em um ambiente de MSs, é necessário monitorar os serviços para detectar ou antecipar falhas, objetivando tomar ações para que o sistema se recupere.

Problema: Como fazer o monitoramento dos serviços?

Solução: Fazer com que cada serviço seja monitorado por outros serviços vizinhos a ele.

Vantagens: Através da checagem de falhas, é possível evitar que requisições sejam encaminhadas para o serviço até que ele se recupere da falha.

Desvantagens: Esta abordagem pode gerar inconsistências.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Availability \cite{S15} | + |

*Ok Arbitral monitor [S15]*

*RNF principal:* Availability

Contexto: Em um ambiente de MSs, é necessário monitorar os serviços para detectar ou antecipar falhas, objetivando tomar ações para que o sistema se recupere.

Problema: Como fazer o monitoramento dos serviços?

Solução: Um grupo de serviços independentes é responsável por detectar falhas nos serviços. Caso algum nó detecte uma falha, ela deve ser confirmada pela maioria dos demais nós pertencentes ao grupo.

Vantagens: Através da checagem de falhas, é possível evitar que requisições sejam encaminhadas para o serviço até que ele se recupere da falha.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Availability \cite{S15} | ++ |

*OK Centralized storage of monitored data [S15]*

*RNF principal:* Monitorability

Contexto: Em um sistema desenvolvido em MSs, são coletados dados obtidos através de monitoramento dos serviços.

Problema: Como armazenar os dados coletados através do monitoramento dos MSs?

Solução: Armazenar os dados coletados de todos os MSs em um componente central de armazenamento.

Vantagens: Possibilita a análise dos logs em um lugar centralizado, diminuindo a necessidade de administração.

Desvantagens: Um único ponto central de armazenamento pode se tornar um ponto único de falha.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Monitorability \cite{S15} | ++ |

*Ok Decentralized storage of monitored data [S15]*

*RNF principal:* Monitorability

Contexto: Em um sistema desenvolvido em MSs, são coletados dados obtidos através de monitoramento dos serviços.

Problema: Como armazenar os dados coletados através do monitoramento dos MSs?

Solução: Armazenar os dados coletados de forma descentralizada, localmente, em componentes localizados em cada host, plataforma ou serviço.

Vantagens: Permite uma melhor escalabilidade da solução de monitoramento, pois cada novo serviço provê um novo local de armazenamento.

Desvantagens: A indisponibilidade de um local de armazenamento pode causar a perda de dados de monitoramento. A existência de muitos locais de armazenagem requer mais esforço de administração.

Desvantagens: Um único ponto central de armazenamento pode se tornar um ponto único de falha.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Monitorability \cite{S15} | ++ |

*Ok Distributed logging [S5]*

*RNF principal:* Monitorability

Contexto: Em um sistema desenvolvido em MSs, são coletados dados obtidos através de monitoramento dos serviços.

Problema: Como armazenar os dados coletados através do monitoramento dos MSs?

Solução: Cada serviço deve armazenar logs em um mesmo servidor externo, responsável por agregar estes dados.

Vantagens: Permite ter um repositório único de logs, força os MSs a usarem o mesmo formato de log, simplifica o monitoramento e o processo de análise.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Monitorability \cite{S5} | ++ |

*OK Local monitor [S15]*

*RNF principal:* Security

Contexto: A natureza distribuída dos MSs resulta em uma complexa rede de interação entre os serviços, abrindo brechas para ataques de segurança contra as aplicações.

Problema: Como detectar ataques de segurança contra MSs?

Solução: Executar um segundo serviço ao lado de cada MS, responsável por monitorar e detectar comportamentos anormais ou ataques em diferentes níveis do serviço.

Vantagens: Permite monitorar eventos de rede e evitar ataques contra a segurança dos MSs.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Security \cite{S15} | + |
| Monitorability \cite{S15} | ++ |

*Ok External monitor [S15]*

*RNF principal:* Security

Contexto: A natureza distribuída dos MSs resulta em uma complexa rede de interação entre os serviços, abrindo brechas para ataques de segurança contra as aplicações.

Problema: Como detectar ataques de segurança contra MSs?

Solução: O monitoramento e detecção de ataques é feito por um servidor externo aos MSs.

Vantagens: Permite realizar um monitoramento mais completo, com avaliações e ações sendo tomadas como resultado do estado geral do sistema.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Security \cite{S15} | ++ |
| Monitorability \cite{S15} | ++ |

*Ok Health check endpoint [S5]*

*RNF principal:* Availability

Contexto: MSs podem ser implantados em qualquer lugar e podem estar indisponíveis por uma quantidade de tempo específica ou devido a um determinado contexto.

Problema: Como verificar se um MS está sendo executado e respondendo a requisições?

Solução: Adicionar pontos de checagem de MSs, responsáveis por verificar periodicamente o status e a habilidade dos serviços em responder requisições.

Vantagens: MSs que estão indisponíveis podem ser detectados e os clientes podem ser avisados para que não enviem requisições a estes serviços.

Desvantagens: A quantidade de comunicação entre os serviços aumenta devido a constante checagem de status, podendo afetar a performance. A quantidade de código fonte também aumenta, gerando um esforço maior de manutenção dos serviços.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Availability \cite{S1, S5} | ++ |
| Performance \cite{S1, S5} | - |
| Maintainability \cite{12} | - |

*OK Instrumentation [S15]*

*RNF principal:* Monitorability

Contexto: Em um sistema de MSs, é necessário gerar e coletar dados de log em nível de host, plataforma e serviço.

Problema: Como gerar e coletar dados estáticos e de execução sobra cada MS?

Solução: Utilizar a instrumentação para coletar dados. A instrumentação pode ser de host, de plataforma e de serviço. Em todos os casos, é necessário a instalação de um agente que fica responsável por coletar informações.

Vantagens: Possibilita a coleta de dados em vários níveis no domínio de serviços (host, plataforma e serviço), permitindo que sejam armazenados posteriormente.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Monitorability \cite{S15} | ++ |

*Ok Logging [S15]*

*RNF principal:* Monitorability

Contexto: Em um sistema de MSs, é necessário gerar e coletar dados de log sobre requisições e respostas enviadas aos clientes dos serviços.

Problema: Como gerar e coletar dados sobre requisições de entrada e saída dos MS?

Solução: Gravar em log informações sensíveis sobre as requisições que entram e saem dos MSs. Cada entrada no log deve representar uma requisição específica, contendo data e hora da ocorrência, tempo de resposta, código de resposta, ID da instância do MS, URL do MS e método chamado. Estes dados são periodicamente agregados por ferramentas de log.

Vantagens: A coleta dos dados de requisições dos MSs permite analisar as informações que estão trafegando, o tempo de resposta, além de rastrear qual serviço foi responsável por determinado processamento.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Monitorability \cite{S15} | ++ |
| Traceability \cite{S15} | ++ |

*Ok Distributed tracing [S15]*

*RNF principal:* Monitorability

Contexto: Em um sistema de MSs, é necessário gerar e coletar dados de log sobre interações entre os serviços.

Problema: Como gerar e coletar dados sobre interações entre os MS?

Solução: Gravar em log informações sobre as chamadas que serviços fazem entre si e entre clientes, bem como a quantidades de requisições feitas entre eles.

Vantagens: A coleta dos dados de requisições dos MSs permite determinar quem iniciou uma sequência de chamadas a um MS e dessa forma poder analisar a raiz de um problema. Os dados podem ser combinados com outras formas de log para melhorar o processo de análise.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Monitorability \cite{S15} | ++ |
| Traceability \cite{S15} | ++ |

*Ok Log aggregator [S34]*

*RNF principal:* Monitorability

Contexto: MSs em execução geram informações sobre erros ocorridos, avisos, informações e outros dados, os quais são gravados em log em um formato padronizado.

Problema: Como entender o comportamento de uma aplicação e resolver problemas?

Solução: Utilizar um serviço de log centralizado que agrega dados de log de cada instância de MSs.

Vantagens: Permite que usuários façam procuras nos dados de log e realizem análises, sendo possível configurar alertas quando determinadas mensagens aparecem.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Monitorability \cite{S1, S9} | ++ |
| Traceability \cite{S9} | + |

*Ok Client certificate [S15]*

*RNF principal:* Security

Contexto: Em um sistema é necessário proteger dados e informações para que somente entidades autorizadas tenham acesso. A natureza distribuída dos MSs faz com que a interação entre serviços se torne bastante complexa. Pessoas mal-intencionadas podem se aproveitar desta complexidade para realizar ataques contra as aplicações na tentativa de obter acesso.

Problema: Como proteger os canais de comunicação entre os serviços?

Solução: Utilizar certificados digitais para autenticar os hosts participantes de uma troca de mensagens e criptografar as mensagens trocadas, protegendo, dessa forma, o canal de comunicação entre eles.

Vantagens: A utilização de certificados digitais traz segurança na troca de mensagens entre os MSs, evitando que entidades externas tenham acesso ao seu conteúdo.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Security \cite{15} | + |

*Ok Key exchanged-based communication authentication [S15]*

*RNF principal:* Security

Contexto: Em um sistema é necessário proteger dados e informações para que somente entidades autorizadas tenham acesso. A natureza distribuída dos MSs faz com que a interação entre serviços se torne bastante complexa. Pessoas mal-intencionadas podem se aproveitar desta complexidade para realizar ataques contra as aplicações na tentativa de obter acesso.

Problema: Como proteger os canais de comunicação entre os serviços?

Solução: Utilizar a troca de chaves para autenticar os hosts participantes de um processo de uma comunicação entre dois serviços, utilizando-as para criptografar as mensagens. Apenas os participantes da comunicação conhecem a chave e podem descriptografar as mensagens trocadas.

Vantagens: A utilização de chaves de criptografia traz segurança na troca de mensagens entre os MSs, evitando que entidades externas tenham acesso ao seu conteúdo.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Security \cite{15} | ++ |

*Ok Federated identity [S15]*

*RNF principal:* Security

Contexto: Em um sistema é necessário proteger dados e informações para que somente entidades autorizadas tenham acesso. A natureza distribuída dos MSs faz com que a interação entre serviços se torne bastante complexa. Pessoas mal-intencionadas podem se aproveitar desta complexidade para realizar ataques contra as aplicações na tentativa de obter acesso.

Problema: Como proteger os canais de comunicação entre os serviços e permitir que somente pessoas autorizadas tenham acesso aos serviços?

Solução: Delegar a autenticação dos usuários para serviços de terceiros. Um serviço pode requerer informações de autenticação a uma entidade federada externa, a qual retorna um token com dados sobre o usuário. Este token pode ser utilizado em conjunto com certificados digitais para criptografia do canal de comunicação e autenticação dos usuários envolvidos na comunicação. O serviço que recebe o token pode consultar a entidade federada externa para validação dos dados e autenticação.

Vantagens: A utilização de tokens em conjunto com certificados digitais traz segurança na troca de mensagens entre os MSs, evitando que entidades externas tenham acesso ao seu conteúdo. As entidades federadas externas facilitam a implementação da segurança, evitando a necessidade de gerenciamento de uma entidade federada local.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Security \cite{15} | + |
| Manageability \cite{[S35]} | + |

*Ok Gatekeeper [S36]*

*RNF principal:* Security

Contexto: Aplicações expõe suas funcionalidades para os clientes através da aceitação e processamento de requisições e normalmente incluem em seu código fonte funções para fazer autenticação e validação, processamento de requisições, acesso a armazenamento e outros serviços.

Problema: Um usuário malicioso pode comprometer o sistema e ganhar acesso ao ambiente que hospeda a aplicação, aos mecanismos de segurança, serviços e dados.

Solução: Para minimizar o risco de usuários ganharem acesso a informações sensíveis, deve-se desacoplar hosts ou tarefas que expõe interfaces públicas do código que processa requisições e acessa áreas de armazenamento. Isso pode ser conseguido através do uso de façades ou tarefas dedicadas que interagem com o cliente e manipulam as requisições.

Vantagens: Permite minimizar o risco de ataques contra as aplicações, atuando como um firewall de uma rede, examinando requisições e fazendo decisões sobre permitir ou não requisições sejam aceitas e processadas.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Security \cite{6, 7} | + |

*OK Correlation ID [S16]*

*RNF principal:* Monitorability

Contexto: Uma aplicação em MSs é construída usando *backend for frontends* e múltiplos MSs e *adapters*, com caminhos complexos de chamadas entre os serviços

Problema: Como fazer para depurar chamadas complexas entre MSs quando não se sabe em qual MS um problema está localizado?

Solução: Utilizar Ids de correlação, os quais geralmente são números passados em cada requisição a um serviço e encaminhados em requisições sucessivas. Quando algum serviço faz log de alguma ocorrência, este ID é logado, tornando possível identificar qual requisição causou o log e quais serviços foram executados na cadeia de atendimento da requisição.

Vantagens: Com a utilização dos Ids de correlação em associação com Log aggregator para unir os logs de todos os serviços, é possível fazer uma busca pelo ID. Através do timestamp das execuções pode-se construir o grafo de execuções, analisar chamadas e parâmetros, dessa forma facilitando o rastreamento de problemas.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Traceability \cite{1} | + |

*Ok Event-driven [S37]*

*RNF principal:* Reliability

Contexto: Em um sistema em MSs, cada serviço possui seu próprio banco de dados. No entanto, algumas transações envolvem mais de um serviço e é necessário garantir a consistência entre os dados armazenados por cada serviço.

Problema: Como manter a consistência de dados entre serviços?

Solução: Implementar uma solução baseada em eventos, onde cada serviço publica um evento sempre que um dado é atualizado. Outros serviços podem se inscrever neste evento. Quando um evento for recebido, os serviços inscritos serão notificados para também atualizarem seus dados.

Vantagens: Permite manter a consistência entre os dados através de múltiplos serviços sem necessidade de utilizar transações distribuídas.

Desvantagens: Sua implementação é mais complexa.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Reliability \cite{30} | + |

*Ok Blue green deployment [S38]*

*RNF principal:* Availability

Contexto: Em um sistema em MSs, é necessário implantar novos serviços ou versões de serviços existentes.

Problema: Como eliminar o tempo de indisponibilidade de um sistema ao implantar um serviço?

Solução: Utilizar o padrão blue green, que consiste em ter dois ambientes idênticos de produção, onde, a qualquer momento, qualquer um dos ambientes pode estar em produção. Ao realizar uma nova implantação, deve-se fazê-lo em apenas um dos ambientes, blue ou green. Após a implantação, devem ser feitos testes para garantir que tudo está funcionando, para então alterar as configurações de roteamento e encaminhar as requisições para o servidor que possui os novos serviços. Se algo inesperado acontecer, pode-se voltar a utilizar a servidor anterior, onde não foram feitas alterações nos serviços.

Vantagens: Elimina o tempo de indisponibilidade dos serviços.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Availability \cite{31} | + |

*Ok Canary release [S39]*

*RNF principal:* Availability

Contexto: Em um sistema em MSs, é necessário implantar novos serviços ou versões de serviços existentes.

Problema: Como reduzir os riscos de implantar uma nova versão de um serviço em um ambiente de produção?

Solução: Implantar o serviço em um subconjunto da infraestrutura e disponibilizá-lo para um grupo restrito de usuários.

Vantagens: Esta abordagem permite implantar gradualmente um serviço para poucos usuários e ir aumentando conforme se ganha confiança na performance do serviço. Caso algo errado ocorra, pode-se rotear os usuários para as versões anteriores rapidamente.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Availability \cite{32} | + |

*Ok Command and Query Responsibility Separation [S40, S41]*

*RNF principal:* Performance

Contexto: Em um sistema em MSs, cada serviço possui seu próprio banco de dados. Dessa forma, fazer consultas que envolvam dados de múltiplos serviços se torna mais complicado.

Problema: Como implementar consultas que retornem dados de múltiplos serviços?

Solução: Separar as operações de escrita e atualização de dados. São utilizados comandos para atualizar dados e queries para ler dados. A realização de consultas deve ser feita em uma visão que possui uma réplica dos dados de todos os serviços. Esta réplica é somente para leitura e é atualizada por um outro serviço, que se inscreve para receber notificações a cada alteração de dados feita pelos serviços.

Vantagens: Possibilita um escalonamento independente das cargas de leitura e escrita. A performance pode ser melhorada devido ao uso de esquemas de dados otimizados individualmente para leitura e escrita. A segurança é melhorada devido a facilidade de controlar entidades que fazem escrita nos dados. A separação de operações permite uma melhor modularização e manutenibilidade do sistema.

Desvantagens: O padrão CQRS é complexo para ser implementado.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Maintainability \cite{S6} | + |
| Performance \cite{S6} | + |
| Security \cite{S6} | + |
| Modularity \cite{S6} | + |
| Agility \cite{33, 34} | - |

*Ok Consumer-driven contracts [S42]*

*RNF principal:* Evolvability

Contexto: MSs precisam ser alterados para atender novos requisitos de seus clientes. Tais alterações levam a criação de novas interfaces específicas para cada tipo de cliente.

Problema: Como minimizar os impactos causados por alterações de interface e como saber de antemão quais clientes serão afetados por elas?

Solução: Implementar contratos entre serviços e seus clientes. Cada contrato possui uma especificação das funções que o serviço deve fornecer ao cliente. O serviço deve conhecer todos os contratos existentes. Dessa forma, quando uma alteração é feita no serviço, é possível saber quais clientes serão impactados, ou se convém desenvolver uma nova interface específica, para que clientes existentes não sejam impactados.

Vantagens: Favorece a evolução dos MSs, pois permite rastrear os impactos das alterações feitas nos serviços, minimizando seus efeitos sobre os clientes dos serviços. Permite fazer testes nos serviços sem a necessidade de participação dos clientes.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Evolvability \cite{S3} | ++ |
| Maintainability \cite{S3} | + |

*Ok Decompose by business capabilities [S43]*

*RNF principal: Modularity*

Contexto: Na migração de um sistema monolítico para MS, o sistema deve ser decomposto em pequenos serviços, buscando-se a correta granularidade dos serviços, objetivando uma alta coesão e baixo acoplamento.

Problema: Como obter uma alta coesão e baixo acoplamento na decomposição de sistemas monolíticos para MSs?

Solução: Definir os serviços de acordo com as capacidades de negócio. Uma capacidade de negócio é algo que organização faz e que gera valor, como, por exemplo, gerenciamento de compras ou gerenciamento de clientes. Cada capacidade de negócio corresponde a um serviço.

Vantagens: Os serviços tendem a ficaram estáveis, pois as capacidades de negócio também são estáveis. Times de desenvolvimento organizados ao redor dos valores de negócio ao invés de características técnicas. A manutenibilidade é melhorada devido à coesão dos serviços e ao baixo acoplamento.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Scalability \cite{S15} | ++ |
| Independency \cite{S15} | ++ |
| Deployability \cite{S15} | ++ |
| Maintainability \cite{S15} | ++ |
| Modularity \cite{S15} | ++ |

*Ok Decompose by subdomain [S44]*

*RNF principal: Modularity*

Contexto: Na migração de um sistema monolítico para MS, o sistema deve ser decomposto em pequenos serviços, buscando-se a correta granularidade dos serviços, objetivando uma alta coesão e baixo acoplamento.

Problema: Como obter uma alta coesão e baixo acoplamento na decomposição de sistemas monolíticos para MSs?

Solução: Definir os serviços de forma que correspondam aos subdomínios identificados na análise feita através do DDD (Domain-Driven Design). No DDD, o domínio é o negócio da organização e consiste de múltiplos subdomínios, cada qual correspondendo a diferentes partes do negócio.

Vantagens: Os serviços tendem a ficaram estáveis, pois os subdomínios também são estáveis. Times de desenvolvimento organizados ao redor dos valores de negócio ao invés de características técnicas. A manutenibilidade é melhorada devido à coesão dos serviços e ao baixo acoplamento.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Scalability \cite{S15} | ++ |
| Independency \cite{S15} | ++ |
| Deployability \cite{S15} | ++ |
| Maintainability \cite{S15} | + |
| Modularity \cite{S15} | ++ |

*Ok Deploy into a Cluster and Orchestrate Containers [S10]*

*RNF principal: Manageability*

Contexto: Um sistema em MSs utiliza CI (Continuous Integration).

Problema: Como implantar instâncias de serviços em um cluster? Como orquestrar a implantação de todos os serviços com o menor esforço possível?

Solução: Utilizar um sistema para gerenciamento do cluster, onde é possível implantar containers de serviços sob demanda e gerenciar falhas e reinicialização de nós caso necessário.

Vantagens: O gerenciamento de falhas melhora a confiabilidade dos MSs. A utilização de uma ferramenta traz mais agilidade na implantação.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Deployability \cite{S10} | + |
| Reliability \cite{S6, S7, S10} | + |
| Agility \cite{S10} | + |
| Manageability \cite{38} | + |

*Ok Change Code Dependency to Service Call [S10]*

*RNF principal: Independence*

Contexto: Em um sistema em MSs, existe um componente que está atuando como uma dependência para outros serviços ou componentes;

Problema: Quando é apropriado alterar a dependência a nível de código para a dependência a nível de serviço?

Solução: Deve-se tentar manter o código dos serviços o mais separado possível. Quando códigos-fonte são compartilhados, existe uma chance maior de falha nos componentes dependentes.

Vantagens: A mudança para dependência de serviço torna os MSs mais independentes, melhorando a manutenibilidade do sistema.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Independence \cite{S10} | + |
| Modularity \cite{S10} | + |
| Maintainability | + |

*OK Gateway aggregation [S45] (é igual ao padrão Aggregator)*

*RNF principal: Performance*

Contexto: Em um sistema de MSs, um cliente precisa fazer várias chamadas a vários serviços backend, dispendendo recursos a cada requisição.

Problema: Como minimizar o número de requisições que o cliente precisa realizar para executar uma tarefa?

Solução: Utilizar um gateway para reduzir o número de requisições entre os serviços e o cliente. O gateway recebe as requisições do cliente e encaminha para os vários serviços backend. Ao receber o retorno de todos os serviços, o gateway agrega os resultados e devolve para o cliente.

Vantagens: Melhora a performance em situações em que existe muita latência, como redes celulares.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Performance \cite{39} | + |

*OK Gateway offloading [S46]*

*RNF principal: Maintainability*

Contexto: Em um sistema de MSs, funcionalidades são utilizadas por múltiplos serviços e requerem configuração, gerenciamento e manutenção. Serviços que são distribuídos com várias aplicações aumentam a carga de gerenciamento. Alterações na funcionalidade compartilhada precisam ser implantadas em todos os serviços que compartilham a função.

Problema: Como minimizar os problemas causados pelo compartilhamento de funcionalidades?

Solução: Mover as funcionalidades compartilhadas para um gateway, liberando assim os serviços, que podem focar apenas nas suas tarefas de negócio.

Vantagens: Melhora a manutenibilidade, pois desacopla funcionalidades e permite que times especializados possam desenvolvê-las.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Maintainability \cite{S6} | + |

*Ok Gateway routing [S47]*

*RNF principal: Evolvability*

Contexto: Em um sistema de MSs, um cliente precisa consumir múltiplos serviços, os quais possuem endereços de rede diferentes e que devem ser gerenciados. Caso a API de um MS seja alterada, todos os clientes que a utilizam deverão ser alterados.

Problema: Como diminuir a necessidade de gerenciamento de vários endereços por parte dos clientes e minimizar impactos que mudanças em APIs podem causar nos clientes?

Solução: Inserir um gateway na frente de um conjunto de MSs. Este gateway é responsável por receber as requisições dos clientes, encaminhá-las para os serviços e então devolver aos clientes o resultado.

Vantagens: O cliente, ao invés de gerenciar vários endereços de serviços, deverá gerenciar apenas o endereço do gateway. Impactos nos clientes, causados por alterações na API dos serviços, podem ser minimizados, pois o gateway pode manter uma API simplificada, enquanto o serviço pode evoluir para interfaces mais complexas do mesmo serviço.

Desvantagens: O gateway pode inserir um ponto único de falha, além de poder se tornar um gargalo na comunicação do cliente com os serviços.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Evolvability \cite{41} | + |
| Maintainability \cite{41} | + |
| Resilience \cite{S14} | ++ |
| Availability \cite{41} | some- |
| Performance \cite{41} | - |

*Ok Microservice chassis [S48, S49]*

*RNF principal: Reusability*

Contexto: No desenvolvimento de um sistema em MSs, comumente é necessário gastar um tempo significativo escrevendo lógica para construção e testes dos serviços. Além disso, é necessário desenvolver soluções para segurança, configuração externalizada, log e checagem de falhas. Essas tarefas muitas vezes se repetem para cada serviço.

Problema: Como diminuir a necessidade de execução de tarefas repetidas no desenvolvimento de um sistema em MSs?

Solução: Criar um framework que pode servir como base para o desenvolvimento dos MSs. Este framework deve conter todas as implementações que são comuns a todos os MSs do sistema.

Vantagens: Permite uma maior agilidade no desenvolvimento dos serviços.

Desvantagens: Deve haver um framework para cada linguagem utilizada na programação dos MSs, o que pode ser um obstáculo na adoção de novas tecnologias.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Agility \cite{42} | + |
| Technology flexibility \cite{42} | some- |
| *Reusability \cite{44}* | + |
| *Maintainability \cite{44}* | + |

*Ok Sidecar [S50]*

*RNF principal: Performance*

Contexto: Um MS precisa ter acesso a componentes e serviços que proveem tarefas comuns, como monitoramento, log e auditoria. A cópia destes componentes em cada MS se torna redundante, ao mesmo tempo que o acesso remoto a eles se torna ineficiente.

Problema: Como fazer com que MSs interajam eficientemente com componentes e serviços utilitários?

Solução: Componentes e serviços utilitários devem ser implantados no mesmo host que o MS, mas dentro de um container a parte. Este container é chamado de sidecar.

Vantagens: Melhora a performance dos MSs, pois o acesso aos componentes utilitários é feito no mesmo host.

|  |  |
| --- | --- |
| RNF impactado | Nível de impacto |
| Performance \cite{43} | ++ |