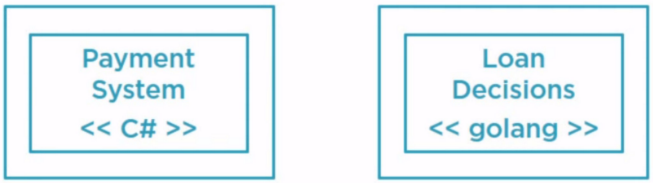
Benefícios do microsserviço

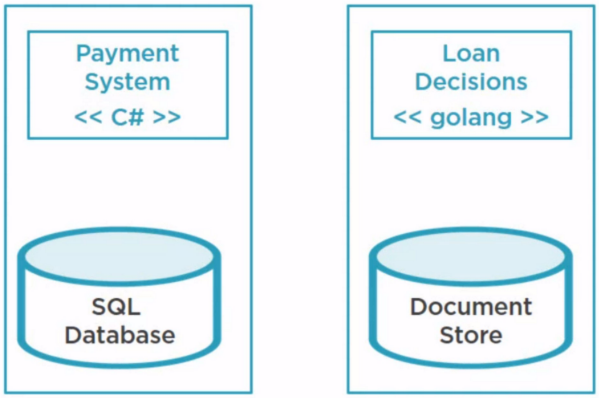
sexta-feira, 17 de setembro de 2021 10:46

Os microsserviços como um conceito arquitetônico oferecem muitos benefícios ao design geral do sistema. Esses benefícios incluem diversidade de tecnologia, resiliência do sistema, escala do sistema e facilidade de implantação. Vamos examinar isso com mais detalhes:

- **Diversidade de tecnologia:**

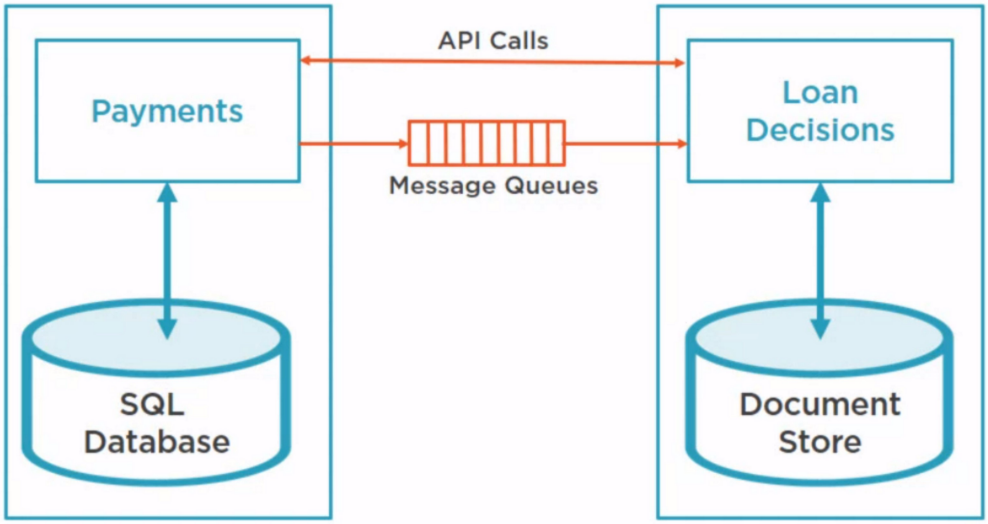


  - Quando você divide seus serviços em componentes pequenos, de baixa granularidade e autônomos, fica livre para usar facilmente diferentes tecnologias para cada serviço. Com um sistema monolítico maior, isso é muito mais difícil de fazer.
  - Isso também significa que podemos usar diferentes tipos de banco de dados para cada serviço. Por exemplo, podemos ter um serviço de pagamento que usa um armazenamento de documentos para salvar uma mensagem JSON de confirmação de pagamento para cada pagamento. Podemos ter um serviço de detalhes do cliente que usa um banco de dados de servidor SQL mais tradicional ou um serviço que gerencia o upload da foto de um cliente usando uma loja de blob.

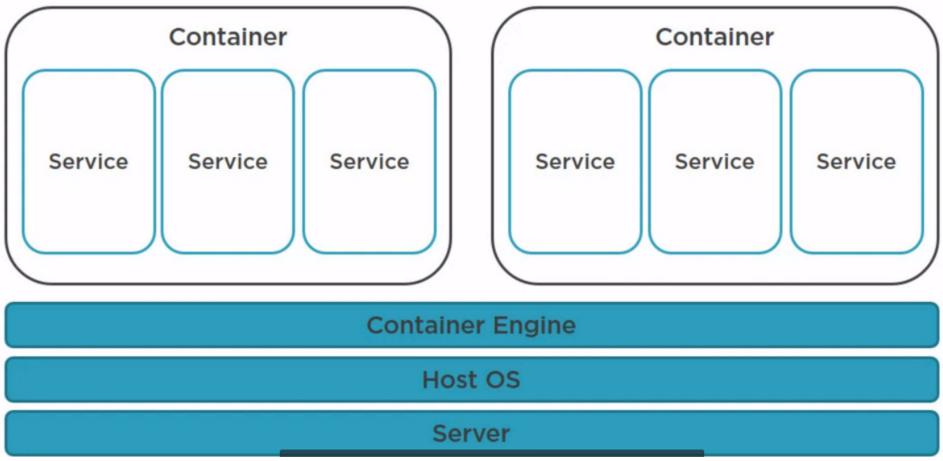


- **Resiliência:**

  - Primeiro, o que é resiliência? Existem várias definições para isso:
    i. A primeira definição afirma que os sistemas resilientes devem ser capazes de lidar com problemas e não ser protegidos contra falhas, o que em algum momento não será suficiente e resultará em quebra.
    ii. A segunda definição afirma a recuperação rápida de sistemas de forma mais explícita. Como mencionamos, você sempre pode reduzir as falhas, mas nunca estará 100% livre de falhas. Portanto, você precisa ser o melhor possível na recuperação de falhas, o que novamente anda de mãos dadas com a imagem de elasticidade de nossa primeira definição.
    iii. A terceira definição vem de um histórico da cadeia de suprimentos e é mais sobre como manter um sistema funcionando. Adaptado a um contexto mais geral, o sistema deve ser capaz de lidar com mudanças e interrupções menores ou maiores.



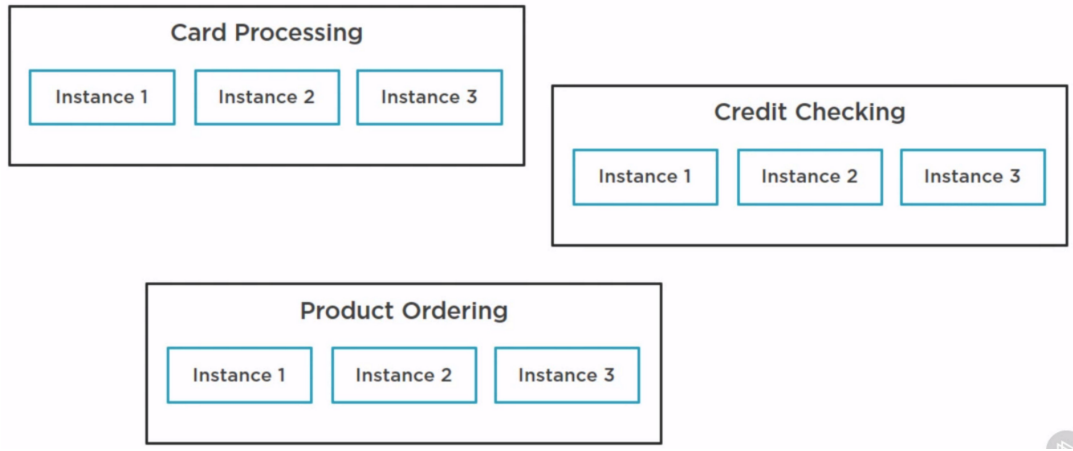
- Resiliência em arquiteturas de microsserviço:
  - Em arquiteturas de microsserviço, os aplicativos são divididos em serviços distribuídos simples e altamente desacoplados que se comunicam entre si por meio de mecanismos de comunicação leves, como filas de mensagens e APIs HTTP.
  - Os microsserviços são altamente dissociados e desenvolvidos para falhas, portanto, são capazes de lidar com falhas e interrupções.
  - O isolamento de serviços simples e pequenos os torna independentes uns dos outros, de modo que quando um falha, os outros não param de funcionar.
  - Eles foram criados explicitamente para esperar falhas no próprio serviço, bem como falhas em outros serviços. Por meio dessa dissociação, realmente obtemos muita resiliência em nossos aplicativos. Eles obtêm a capacidade de lidar com mudanças e interrupções, mantendo a agilidade e velocidade no processo de desenvolvimento. No entanto, as partes de elasticidade e recuperação da resiliência ainda precisam de mais do que apenas mudar para arquiteturas de microsserviço em desenvolvimento para serem cumpridas. E a resiliência com implantações?



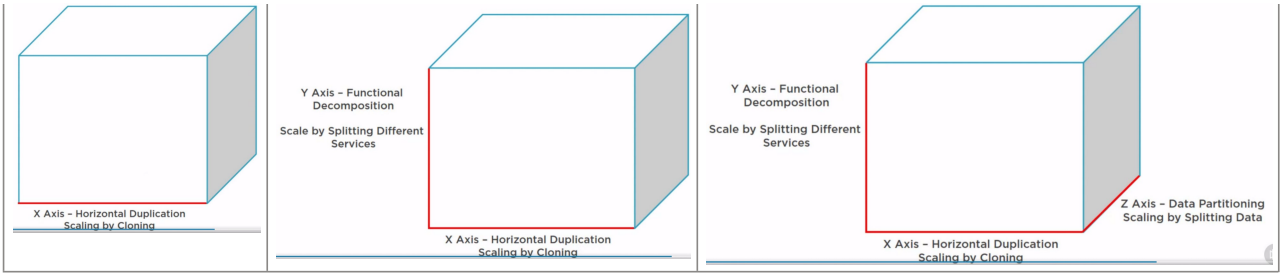
- Resiliênciana implantação de microsserviço:
  - A elasticidade e a recuperação, que em parte estão embutidas nos próprios serviços, são mais uma tarefa para o gerenciamento de implantação. É aqui que os contêineres e as infraestruturas de cluster entram em ação. Com a conteinerização, obtemos uma tecnologia simples para empacotar nossos microsserviços em unidades implantáveis persistentes. Como esses contêineres podem ser executados em vários ambientes, os desenvolvedores não precisarão cuidar das diferenças de desenvolvimento, teste e produção. Porém, agora vem a parte difícil. Esses contêineres precisam ser

implantados, gerenciados e escalados na produção e lembre-se, queremos ter resiliência, elasticidade e recuperação.
    - Queremos que isso seja executado em infraestruturas de cluster que abstraem o hardware subjacente e nos tragam redundância, escalabilidade, elasticamente para cima e para baixo horizontalmente.
    - O escalonamento e a redundância nos ajudam a não quebrar completamente em caso de falha. No entanto, para sermos realmente resilientes, precisamos ser capazes de lidar com as falhas e recuperar o estado original pretendido do aplicativo. Quando partes do aplicativo são desativadas, elas precisam ser ativadas novamente. Agora vamos dar uma olhada no dimensionamento.

- **Dimensionamento:**



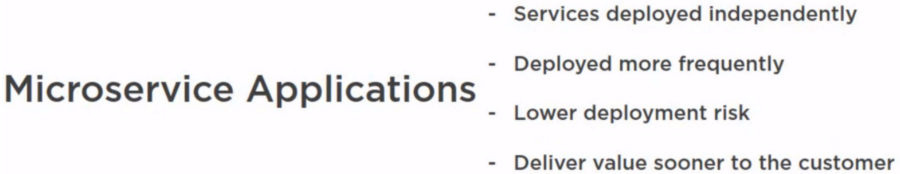
    - Com grandes serviços monolíticos, temos que dimensionar tudo juntos. Quando pequenas partes de nosso sistema geral têm desempenho restrito, mas se esse comportamento está preso em um aplicativo monolítico gigante, temos que lidar com o dimensionamento de tudo como uma peça. Com serviços menores, podemos apenas dimensionar os serviços que precisam de dimensionamento, o que nos permite executar outras partes do sistema em hardware menor e menos potente.
    - Como você pode ver na imagem acima, cada uma das caixas maiores é um microsserviço. Esses microsserviços têm várias instâncias em execução para atender à demanda. Se, de repente, obtivermos um grande aumento na demanda, queremos adicionar mais instâncias. Isso pode ser feito automaticamente se você estiver usando um sistema de infraestrutura que ofereça suporte a dimensionamento elástico, como o Azure da Microsoft ou Amazon Web Services ou você pode iniciar manualmente instâncias adicionais, mas o ponto é que deve ser fácil adicionar ou diminuir o número de instâncias em execução sem interromper a execução do sistema como um todo. Conforme você adiciona instâncias adicionais de um serviço, espera que as mensagens entrem nesse serviço para serem distribuídas entre cada instância. Por exemplo, se estivermos processando pagamentos com cartão e tivermos cinco instâncias de nosso serviço de pagamento com cartão em execução e nosso sistema enviar 10 mensagens de pagamento, esperaríamos que cada uma dessas cinco instâncias recebesse duas cada uma das mensagens de pagamento, se estiverem sendo distribuídas uniformemente entre todos eles.
    - Tipos de dimensionamento:



- Eixo X: consiste em executar várias cópias de um aplicativo por trás de um balanceador de carga. Se houver 10 instâncias, cada instância lida com uma parte igual da carga. Esta é uma abordagem simples e comumente usada para dimensionar um aplicativo. Uma desvantagem dessa abordagem é que, como cada cópia acessa potencialmente todos os dados, os caches requerem mais memória para serem eficazes. Outro problema com essa abordagem é que ela não aborda os problemas de desenvolvimento e complexidade de aplicativos crescentes.
- Eixo Y (microsserviço): divide o aplicativo em vários serviços diferentes. Cada serviço é responsável por uma ou mais funções intimamente relacionadas. Existem algumas maneiras diferentes de decompor um aplicativo em serviços. Uma abordagem é usar a decomposição baseada em verbo para encontrar serviços que implementam um único caso de uso, como um check-out online. A outra opção é decompor o aplicativo por substantivo e criar serviços responsáveis por todas as operações relacionadas a uma entidade específica, como gerenciamento de clientes ou pagamentos com cartão. Um aplicativo pode usar uma combinação de decomposição baseada em verbo e baseada em substantivo.
- Eixo Z: cada servidor executa uma cópia idêntica do código. Nesse aspecto, é bastante semelhante à escala do eixo X. A grande diferença é que cada servidor é responsável por apenas um subconjunto dos dados. Algum componente do sistema é responsável por rotear cada solicitação ao servidor apropriado. Um critério de rotina comumente usado é um atributo da solicitação, como a chave primária da entidade que está sendo acessada. Outro critério de roteamento comum é o tipo de cliente. Por exemplo, um aplicativo pode fornecer aos clientes pagantes um acordo de nível de serviço mais alto do que os clientes livres, alcançando sua solicitação para um conjunto diferente de servidores com mais capacidade. As divisões do eixo Z são comumente usadas para dimensionar bancos de dados. Os dados são particionados ou fragmentados em conjuntos de servidores com base em um atributo de cada registro.

- **Facilidade de implantação:**

  - Uma única mudança em um enorme aplicativo monolítico requer que todo o aplicativo seja implantado para liberar a mudança. Isso poderia ser uma implantação de grande impacto e alto risco para sua organização. Na prática, implantações de grande impacto e alto risco acabam acontecendo com pouca frequência devido ao medo e risco compreensíveis.



  - Com microsserviços, podemos fazer uma alteração em um único serviço e implantá-lo

independentemente do resto do sistema. Isso nos permite implementar o nosso código com mais rapidez. Se ocorrer um problema, ele pode ser isolado rapidamente para um serviço individual, facilitando a reversão rápida. Isso também significa que podemos fornecer novas funcionalidades aos clientes mais rapidamente.

- A melhor maneira de implantar um microsserviço é usando um pipeline de compilação automatizado, onde uma vez que um desenvolvedor verifica seu código, você segue um processo definido como confirmar as alterações, compilar o código, executar qualquer análise de código estático, executar qualquer unidade e testes de integração, empacotamento a compilar no servidor de compilação e, finalmente, implantar em um ambiente. Depois que isso acontecer, você pode promover esse pacote de compilação para o seu ambiente de teste para mais testes. Quando todos estiverem satisfeitos com esse pacote, ele poderá ser implantado em produção. O tempo de retorno para isso deve ser idealmente muito rápido para que novos recursos ou correções possam ser implantados na produção muito rapidamente. Usar um pipeline de implantação automatizado como este, juntamente com os benefícios arquitetônicos dos próprios microsserviços, oferece um sistema muito flexível e fácil de mudar.