Prova de conceito

## Título

Uso do padrão “Branch por abstração” para decomposição de arquitetura monolítica em microsserviços.

## Objetivo

Realizar a decomposição de parte de nosso sistema monolítico em microsserviços usando o padrão de referência “Branch por abstração” sem que os serviços sejam interrompidos durante a migração de arquitetura.

## Escopo

O protótipo criado nessa prova de conceito irá usar o padrão de decomposição com o objetivo migrar o modulo de vendas do sistema monolítico para microsserviços, onde estão inclusas as funções de pedidos, clientes e produtos.

## Requisitos

##### Funcionais:

* Realizar pedidos;
* Cadastrar clientes;
* Cadastrar produtos;

##### Não funcionais:

* Os serviços devem permanecer em funcionamento durante a migração de arquitetura.
* Os contratos com os clientes não podem ser alterados.
* O desempenho deve ser simular ou melhor.

## Metodologia

* Foi utilizado metodologia **ágil** para desenvolvimento da prova de conceito com sprints semanais e incrementais.
* Para a execução as seguintes ferramentas foram usadas:
  + **Trello** para gerenciamento das tarefas;
  + **Github** para versionamento de código fonte;
  + .**Net Framework 4.8**, **Framework .Net 8** e **Visual Studio** para desenvolvimento.
  + **Swagger** e **Postman** para realização de testes e documentação.

## Cronograma

As tarefas foram distribuídas em 3 sprints de duas semanas cada totalizando 45 dias. As atividades ficaram divididas da seguinte forma:

* **Sprint 01:** Planejamento e desenvolvimento do protótipo;
* **Sprint 02:** Desenvolvimento do protótipo, execução de testes e análise dos resultados;
* **Sprint 03:** Documentação da prova de conceito;

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Figura 1 - Sprints catalogadas no Trello.

## Recursos

* **Humanos:** 1 desenvolvedor backend;
* **Tecnológicos:** Ferramenta para desenvolvimento e versionamento, acesso ao código fonte monolítico.
* **Financeiro:** Horas de trabalho.

## Critérios de sucesso

Os critérios abaixo balizam o sucesso da prova de conceito:

* As funcionalidades devem estar 100% migradas para arquitetura de microsserviços;
* Os serviços devem estar em pleno funcionamento (pedidos, clientes e produtos);
* Os contratos com os usuários devem ser mantidos;
* Deve ocorrer pouca ou nenhuma instabilidade/interferência durante a migração dos serviços;

## Critérios x resultados

Após a implementação do padrão de decomposição “Branch por abstração” os resultados esperados foram obtidos. A seguir demonstro cada um deles:

##### Funcionalidades migradas para arquitetura de microsserviços:

Inicialmente tínhamos um sistema monolítico onde os serviços de pedidos, clientes e produtos funcionavam juntos:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Figura 2 - Sistema monolítico

Após a migração temos cada um dos serviços independentes:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Figura 3 - Microsserviço de clientes

Interface gráfica do usuário, Aplicativo, Email

Descrição gerada automaticamente

Figura 4 - Microsserviço de pedidos

Interface gráfica do usuário, Aplicativo, Email

Descrição gerada automaticamente

Figura 5 - Microsserviço de produtos

##### Serviços em pleno funcionamento (pedidos, clientes e produtos):

Durante e após a migração os serviços se mantiveram em funcionamento sem interrupções. A seguir temos os testes de funcionamento:  
  
**Teste 01**: Criando um cliente com o método [POST] - </api/cliente> apontando para o serviço monolítico:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**Teste 02**: Obtendo a lista de todos os clientes com o método [GET] - [/api/cliente](file:///C:\api\cliente) apontando para o serviço monolítico:

Tela de computador com jogo

Descrição gerada automaticamente

**Teste 03:** Criando um cliente com o método [POST] - </api/cliente> apontando para o microsserviço de clientes:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**Teste 04:** Obtendo a lista de todos os clientes com o método [GET] - [/api/cliente](file:///C:\api\cliente) apontando para o microsserviço de clientes:

Interface gráfica do usuário

Descrição gerada automaticamente

Neste último teste é possível verificar que ao realizar a consulta é retornado dois clientes sendo que o primeiro foi cadastrado pelo serviço monolítico e o segundo pelo microsserviço de clientes.

**Teste 05:** Obtendo cliente específico pelo método [GET] - [/api/cliente/{codigo}](/api/cliente/%7bcodigo%7d) em ambos os serviços:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Figura 6 - Obtendo cliente pelo sistema monolitico

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Figura 7 - Obtendo o cliente pelo microsserviço de clientes

**Teste 06:** Atualizando cliente com o método [PUT] - [/api/cliente](file:///C:\api\cliente) apontando para o microsserviço de clientes e consultando atualização pelo serviço monolítico:  
Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Figura 8 - Atualizando cliente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Figura 9 - Cliente atualizado

**Teste 07:** Apagando dados de um dos clientes pelo método [DELETE] - [/api/cliente](file:///C:\api\cliente) apontando para o microsserviço de cliente:

Tela de computador com jogo

Descrição gerada automaticamente

Teste 08: Consultando novamente a lista de clientes para verificar se a exclusão funcionou, usando o método [GET] - [/api/cliente](file:///C:\api\cliente) do sistema monolitico:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Ao fim de todos os testes é possível perceber que apesar dos ambientes distintos o comportamento dos contratos é o mesmo, conforme esperado pelo uso do padrão.  
Dessa forma as aplicações consumidoras não sentem o impacto da mudança.

##### Os contratos com os usuários devem ser mantidos:

Todos os contratos foram mantidos, assegurando que nenhuma modificação fosse percebida pelos usuários:

* Contratos do sistema em arquitetura monolítica x microsserviços respectivamente:
  + Clientes:

Padrão do plano de fundo

Descrição gerada automaticamente

Aplicativo

Descrição gerada automaticamente com confiança média

* + Pedido:

Padrão do plano de fundo

Descrição gerada automaticamente

Aplicativo, Padrão do plano de fundo

Descrição gerada automaticamente

* + Produto:

Padrão do plano de fundo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Para tornar esse processo possível foi necessário criar uma interface comum entre os serviços como garantia da manutenção do padrão:

Texto

Descrição gerada automaticamente

Figura 10 - Interface que garante o contrato com os clientes.

Texto

Descrição gerada automaticamente

Figura 11 - Exemplo de implementação da interface no sistema monolitico.

Texto

Descrição gerada automaticamente

Figura 12 - Exemplo de implementação da interface no sistema em microsserviço.

##### Deve ocorrer pouca ou nenhuma instabilidade/interferência durante a migração dos serviços:

Nenhuma instabilidade ou interferência dos serviços foi relatada pelos usuários da aplicação durante e posteriormente a implementação da migração dos serviços de pedidos, clientes e produtos.

## Conclusão

Após todo o processo de análise, prototipação, testes e documentação desta POC, a conclusão a que se chega é que o padrão de decomposição “branch por abstração” cumpriu muito bem a necessidade imposta de realizar a migração de arquitetura de artefatos pré-existentes em nosso sistema monolítico para a arquitetura de microsserviços, sem que ficasse perceptível aos usuários as mudanças em andamento.

É claro que a migração não é simples pois tem percalços no caminho. Para que o resultado fosse atingido foi necessário criamos, por exemplo, um projeto de compartilhamento de recursos em .Net Standard 2.0, que provê a compatibilidade entre parte do software em .Net Framework 4.8 e .Net 8. Fora isso foi necessário realizar modificações no sistema legado para garantir a eficácia do padrão, tornando os controllers das APIs compatíveis com a interface que garante o contrato com os usuários.

Além da migração inicial muito a de se pensar no futuro. Questões como banco de dados compartilhado entre os serviços ainda precisam ser levantadas e estudadas, algo que não é objetivo deste documento.

A conclusão a que se chega é que o padrão pode ser adotado pela equipe de desenvolvedores e arquitetos como base para a migração de serviços futuros, pois atende a necessidade da equipe e norteia o desenvolvimento do produto com base nas diretrizes impostas pela empresa.

## Referencias

Newman, Sam. *Migrando sistemas monolíticos para microsserviços: Padrões evolutivos para transformar seu sistema monolítico*. 1. ed. São Paulo: Novatec Editora Ltda, 2020.