

Microservices, .NET Core e Serverless

Desenvolvimento de aplicações modernas visando Alta-Disponibilidade, Auto-Escala e baixo custo

Ulili Emerson Martins Nhaga
ulili5@hotmail.com
<https://www.linkedin.com/in/ulili5/>



<ulili5/>

Ulili Emerson Martins Nhaga



Consultor Sr. & Arquiteto de Software na FCamara
ulili5@hotmail.com
<https://www.linkedin.com/in/ulili5/>

servidor assim **BigData** aplicação entender Então pode **Ulili** coisas **Asp NET** África artigo explorar trata **Microsoft** finalidade onde podemos alta disponibilidade baixo custo final **Serverless** boa parte tecnologia encontrar definição Amazon computação **nuvens** desenvolver disponibilizar Serviços **Funções** pensar precisa escalar **Eventos** nuvem pouco tempo processamento podem endpoint url serem executadas através orquestrar utilizar principais provedores algumas nome produto AWS **Azure** Functions Cloud Google OpenWhisk IBM tudo **Ai ML** bem Nesta arquitetura Normalmente 5 minutos terminar Stateless **cada requisição** evento somente HTTP execução recursos aguardando alguns Califórnia devido demanda exemplo seguir características independentemente **provedor** Auto Escala implementando preocupar escalabilidade forma requisições automaticamente Várias atender irá executar capacidade geralmente conta responsabilidade fazer focar lógica desenvolvimento código requisito negócio path precisar garantir rodando Integração demais **produtos** dentro além integrar diferentes baseado possível WebHook receber prover outras arquivo **Storage** pagamento Cenários uso desenhar simples Imagine enviar deseja link usuário alguma outro necessidade criar usuários público dependendo tipo possui site área dispositivos dados vez **https** algum Upload **imagem** redimensionar **imagens** e Token segurança acesso Estando usar consultar etc funcionalidades monitoramento disponibiliza programação Microservice linguagem escrever publicar vezes vários Framework projeto conjunto comunidade desde projetos implementar melhoria nova espaço vem **hands on** usando gerar tamanhos acessível internet cobrado acionada Smartphones resolução precisará realizar otimizar qualidade página possa onerar neste usarei HTML **passo** larga armazenar Function social technet wiki pt br contents articles aspx encontrará SAS responsável Disponibilizei versões

Desafios dos Softwares Modernos

É possível desenvolver aplicações cross-plataforma com Asp.NET?

Posso garantir ambientes idênticas para Dev, QA e Produção?

Dá para aumentar escala somente de algumas partes da aplicação?

Como lidar com auto-escala quando a demanda aumenta?

Tem como saber exatamente os erros que acontecem em produção em tempo real?

Agenda

- Microservices
 - Introdução
 - Principais características
 - Desafios do Desenvolvimento
 - Pontos de atenção
 - Deploy e Monitoramento da operação
- .NET Core
 - Introdução
 - Novidades da v2.0
 - Desenvolver microserviços
- Serverless
 - Características
 - Onde e quando usar?
 - Como é o desenvolvimento
 - Desenvolver microserviços



Projetar
Microservices
é dividir para
conquistar

.NET Core
a evolução, cross-
plataforma, alta
performance



Serverless
o futuro do cloud
já está disponível,
if this do that



Microservices

Microservices - Introdução

- Arquitetura Microservices é uma abordagem sistema distribuída para construir uma aplicação como um conjunto de pequenos serviços onde:
 - Cada serviço é autônomo e executado em seu próprio processo pode se comunicar com outros serviços usando protocolos como HTTP / HTTPS, WebSockets ou AMQP.
 - Cada microserviço implementa um domínio específico de ponto-a-ponto ou fragmento de negócios dentro de um delimitado contexto, e cada um deve ser desenvolvido de forma autônoma e ser implantável de forma independente.
 - Cada microserviço deve possuir seu modelo de dados de domínio e lógica de domínio relacionados (soberania e gerenciamento descentralizado de dados) podendo ter a base em diferentes tecnologias de armazenamento de dados (SQL, NoSQL) ou diferentes linguagens de programação.

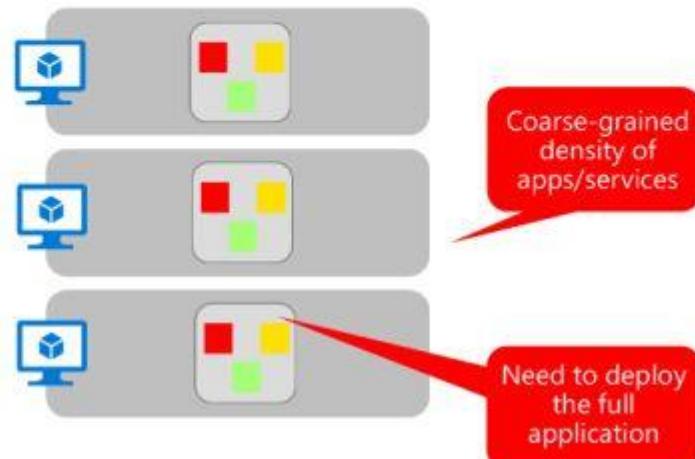
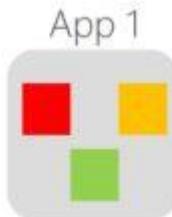
<ulili5/>

Microservices - Introdução

■ Microservices vs Monolítico (tradicional)

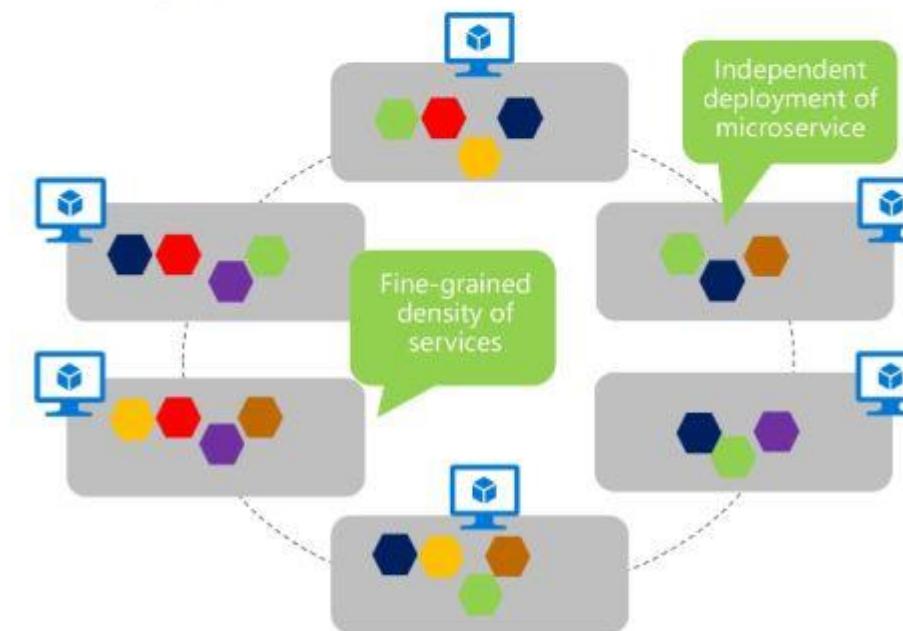
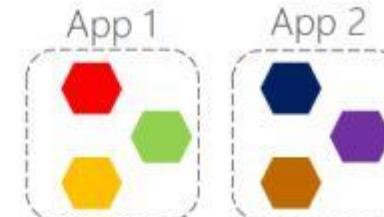
Monolithic deployment approach

- A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
- Scales by cloning the app on multiple servers/VMs



Microservices application approach

- A microservice application segregates functionality into separate smaller services.
- Scales out by **deploying each service independently** with multiple instances across servers/VMs



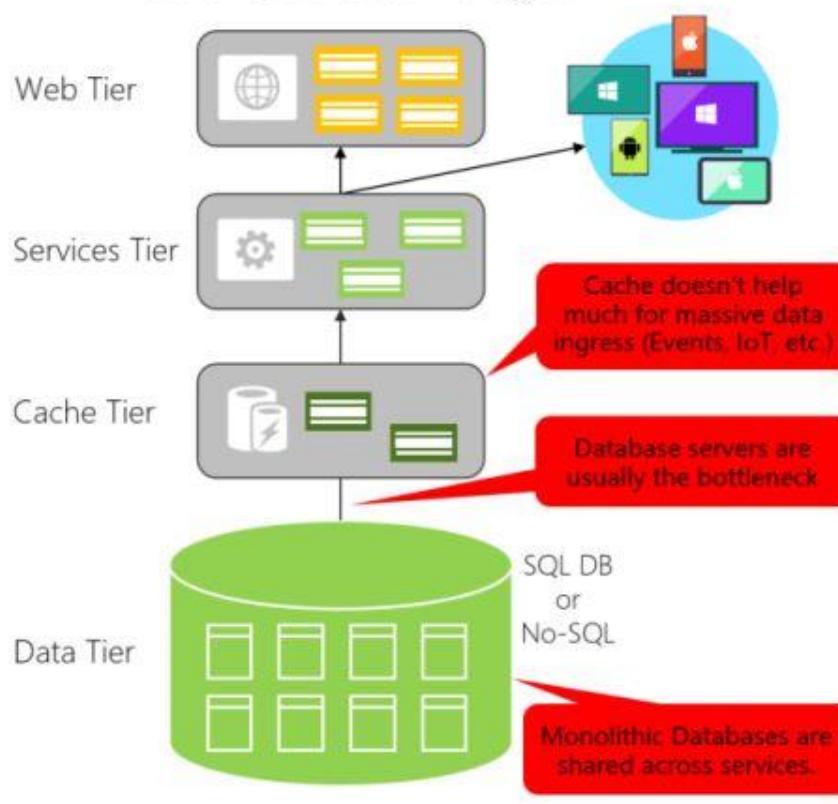
;/>

Microservices - Introdução

■ Microservices vs Monolítico (tradicional)

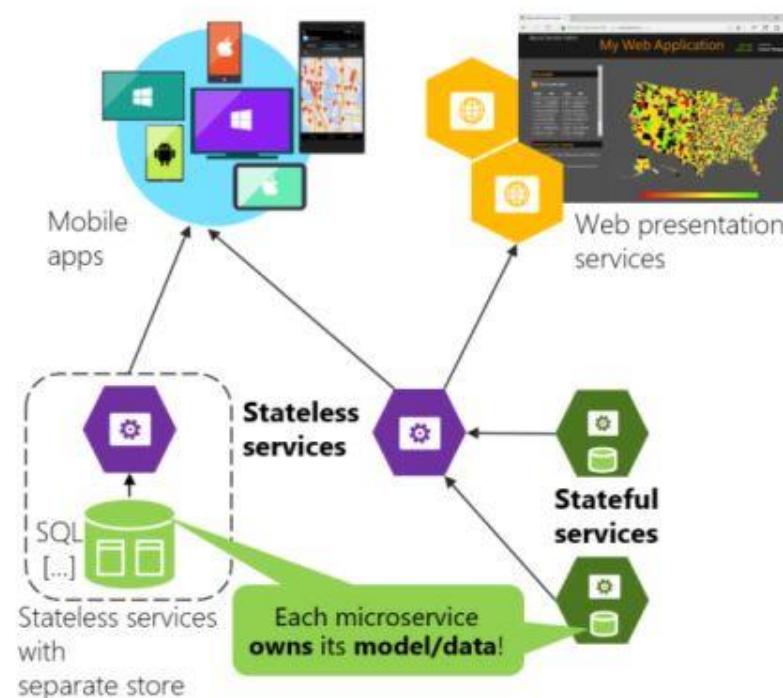
Data in Traditional approach

- Single monolithic database
- Tiers of specific technologies



Data in Microservices approach

- Graph of interconnected microservices
- State typically scoped to the microservice
- Remote Storage for cold data



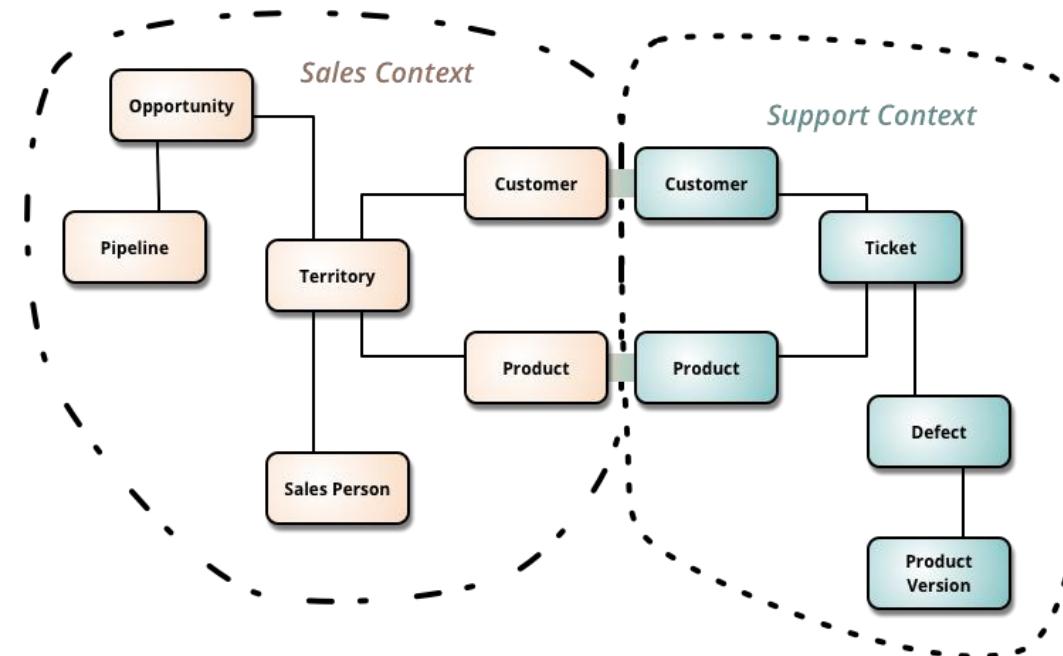
i5/

Microservices – Principais características

- Características comuns dos Microservices que devem ser levados em conta para projetar Microservices:
 - Bounded Context é um fragmento de negócio e deve ser um microserviço
 - Microservices é uma Arquitetura Lógica
 - Um serviço pode ser Stateless ou Stateful
 - API Geteway Pattern pode isolar a comunicação direta entre microserviços e cliente
 - Event-Driven Architecture com Event Bus ajudam a amenizar problemas de CAP Theorem
 - CQRS é um grande aliado para conquistar alta performance.
 - Cada serviço pode implementar sua própria arquitetura ex: DDD, CRUD, Clean, N-Layer, etc.
 - Containers, Orquestradores e Load Balance para cuidar da infraestrutura

Microservices - Bounded Context

- O conceito de Microservices deriva do Bounded Context (BC) Pattern do Domain-Driven Design (DDD).
- DDD lida com grandes modelos, dividindo-os em BC múltiplos bem delimitado, cada BC deve ter seu próprio modelo e banco de dados;
- Da mesma forma, cada Microserviço possui seus dados relacionados.
- BC geralmente tem sua própria linguagem ubiquitous para ajudar a comunicação entre desenvolvedores de software e especialistas em domínio.

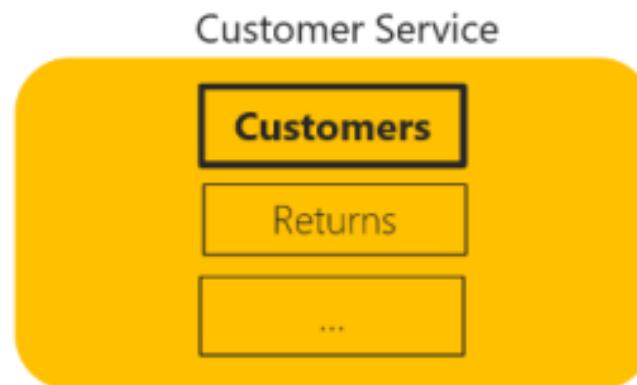
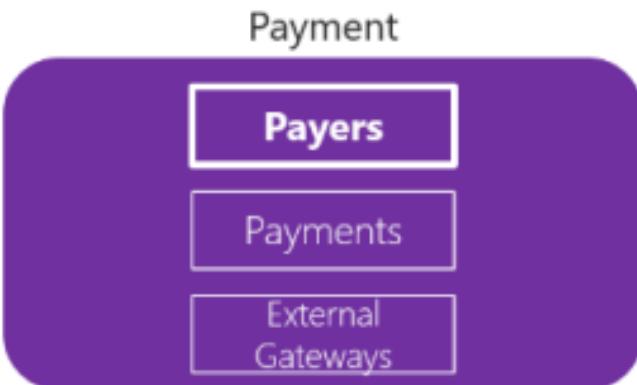
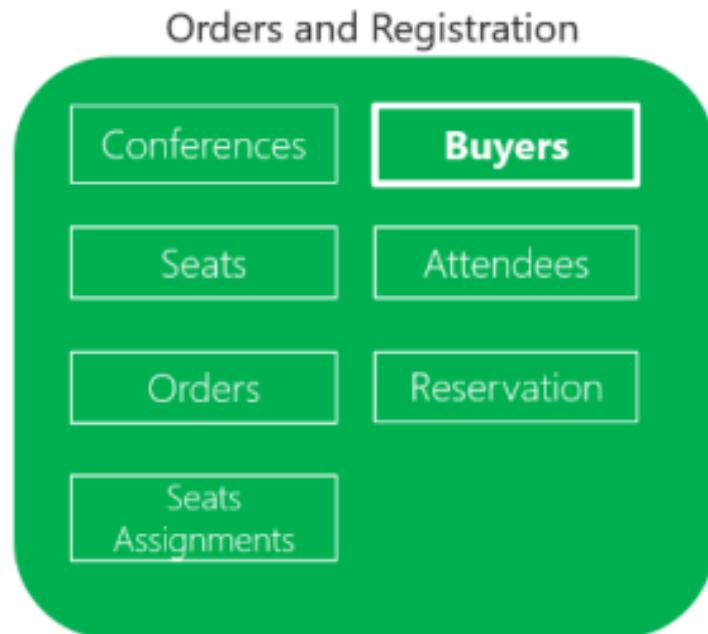


<ulili5/>

Microservices - Bounded Context

Identifying a domain model per microservice or Bounded Context

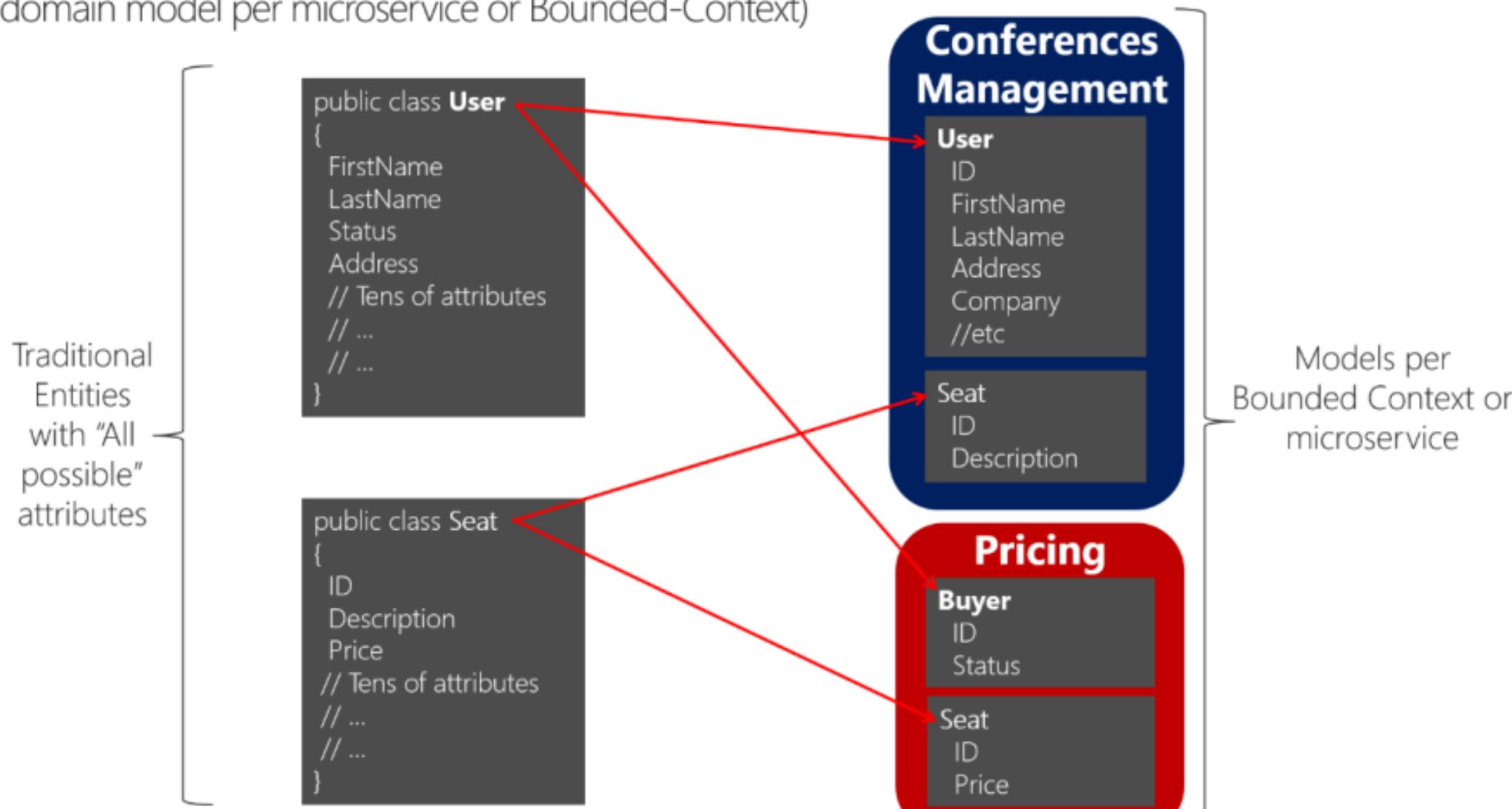
)



>

Microservices - Bounded Context

Decomposing a traditional data model into multiple domain models
(One domain model per microservice or Bounded-Context)

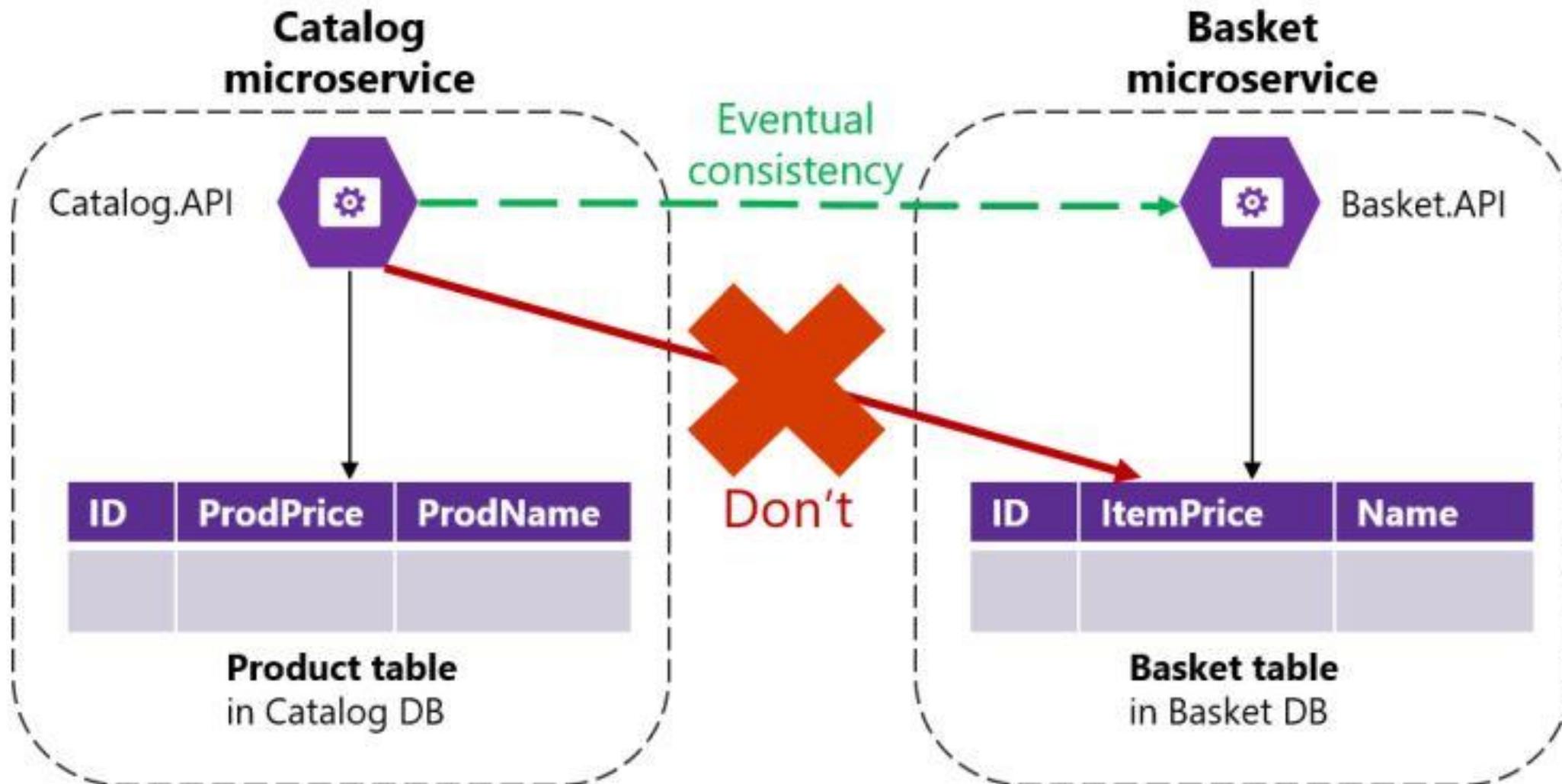


)DD).

seu

/>

Microservices - Bounded Context



Databases are private per microservice

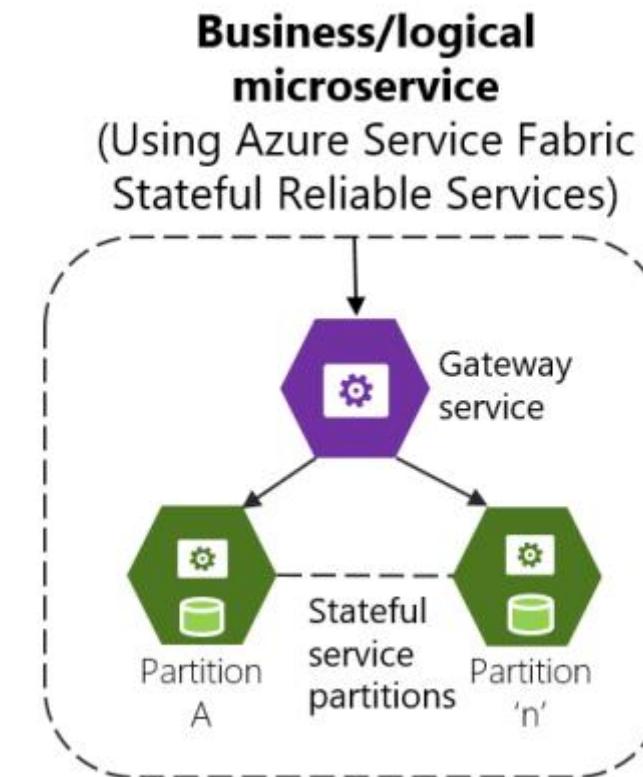
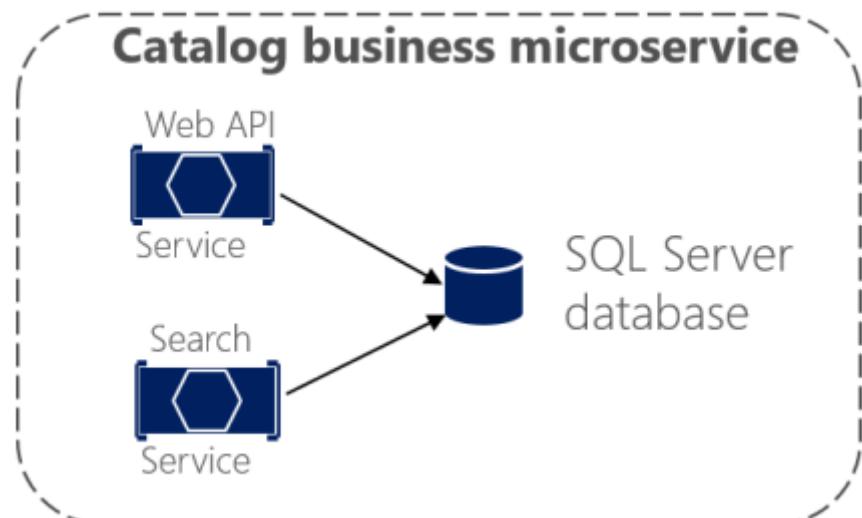
(DDD).

r seu

5/

Microservices – Arquitetura Lógica vs Física

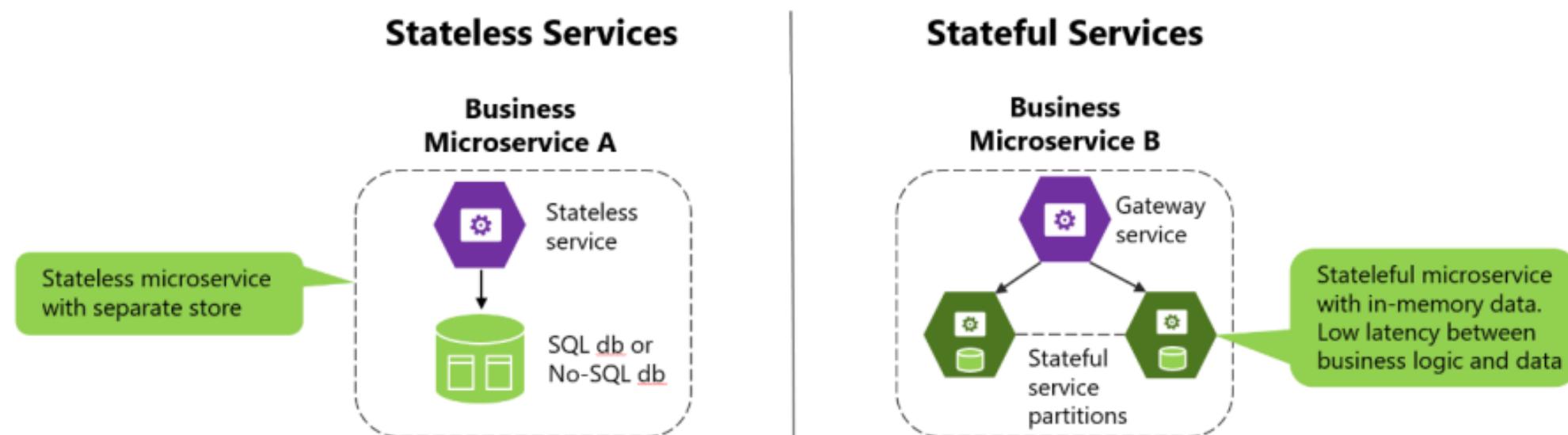
- A arquitetura lógica e os limites lógicos de um sistema não necessariamente mapeiam um-para-um para a arquitetura física ou de implantação. Pode acontecer, mas muitas vezes não.
- Embora você possa ter identificado vários BC, isso não significa que a melhor maneira de implementá-los é sempre criando um único serviço (como uma API da Web ASP.NET) ou um contêiner Docker único para cada microserviço de negócios;



<ulili5/>

Microservices – Stateless ou Stateful

- **Stateless** possui os bancos de dados externos, usando bancos relacionais como o SQL Server, ou NoSQL como o MongoDB ou o Azure Document DB.
- **Stateful** possui os dados embarcados dentro do microservice. Estes dados podem existir não apenas no mesmo servidor, mas dentro do processo microservice, na memória e persistentes em discos e replicados para outros nós. (obs: se for implementar com Docker use Microsoft Orleans ou Akka.NET)



<ulili5/>

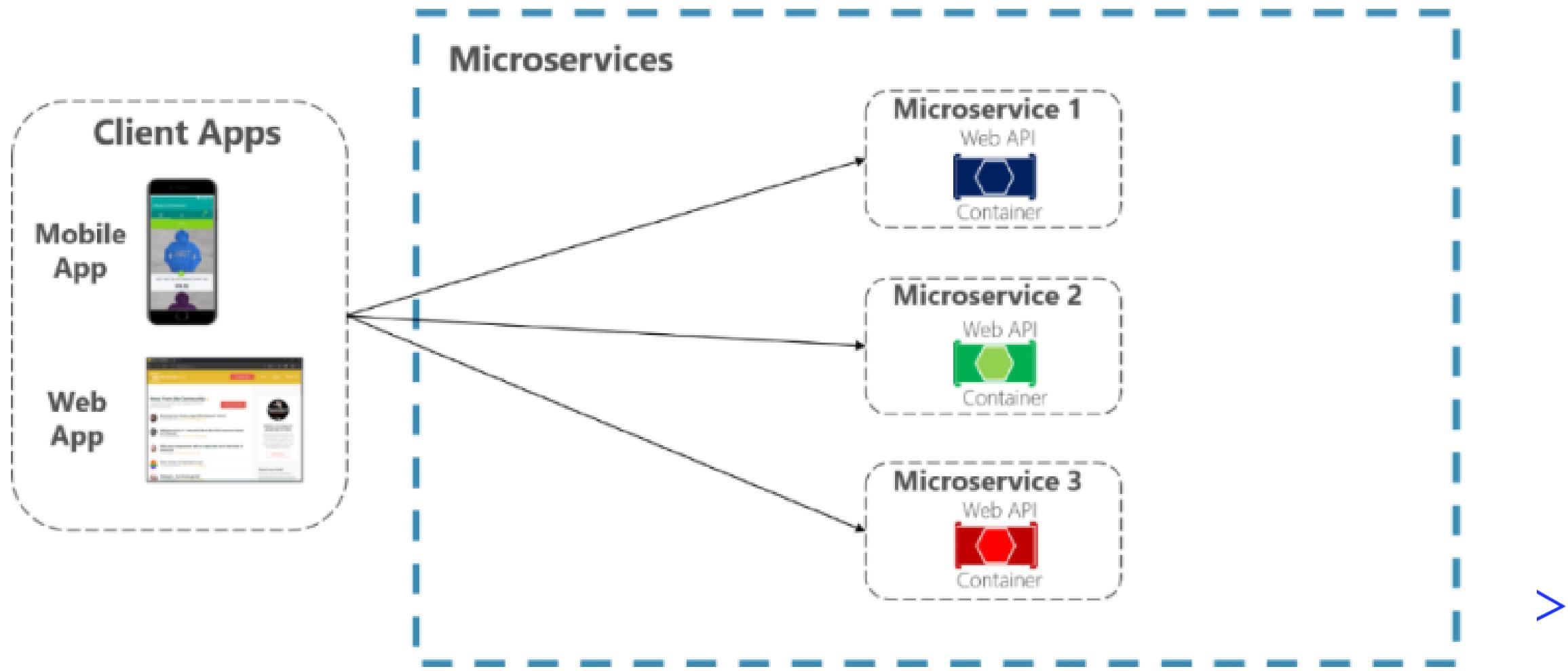
Microservices – API Geteway Pattern

- Este é um serviço que fornece um único ponto de entrada para certos grupos de microservices.
- É semelhante ao Facade pattern do orientado a objetos, mas neste caso, faz parte de um sistema distribuído, onde ele abstrai endpoint para N microserviços.
- Funciona como Register, quando um serviço for instanciado ela se identifica e registra a sua rota interna que será usada para direcionar as requisições externas

Microservices – API Geteway Pattern

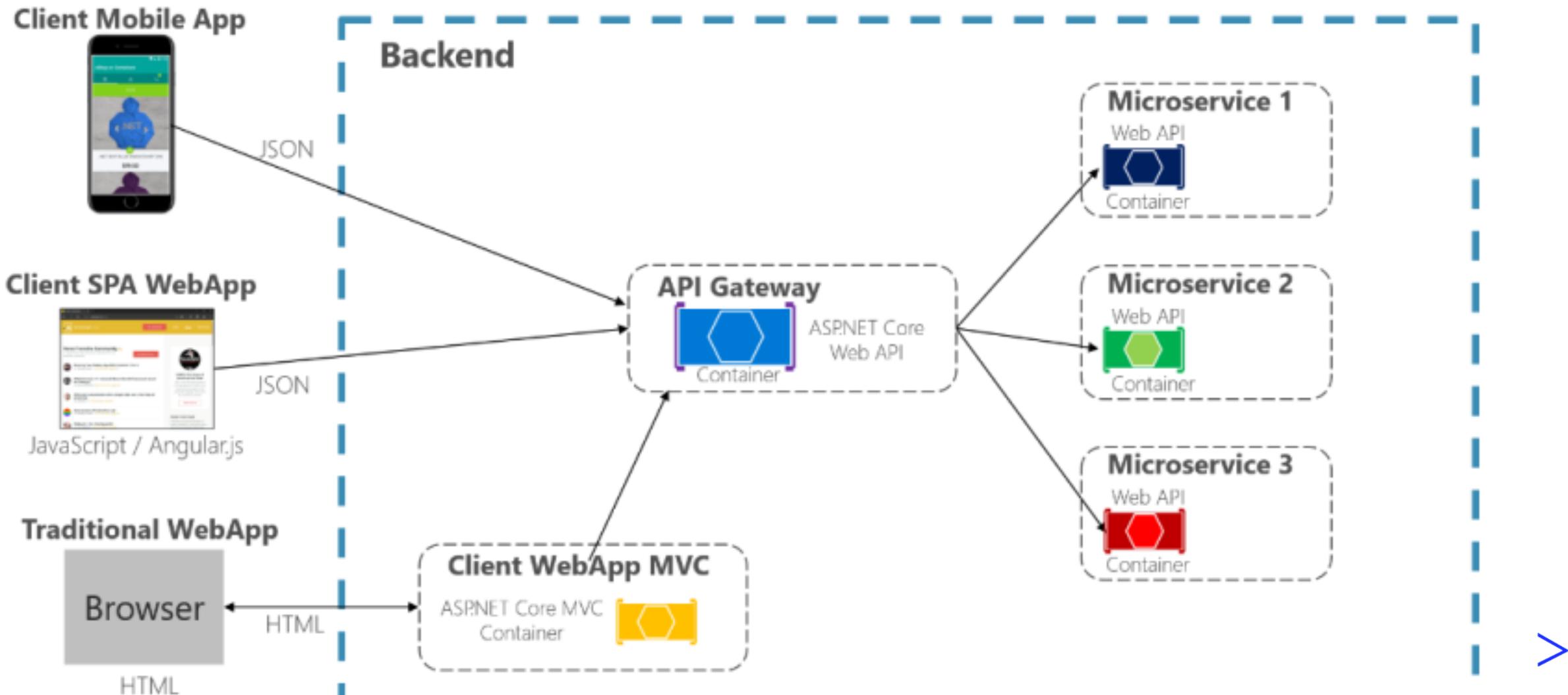
Direct Client-To-Microservice communication

Architecture



Microservices – API Geteway Pattern

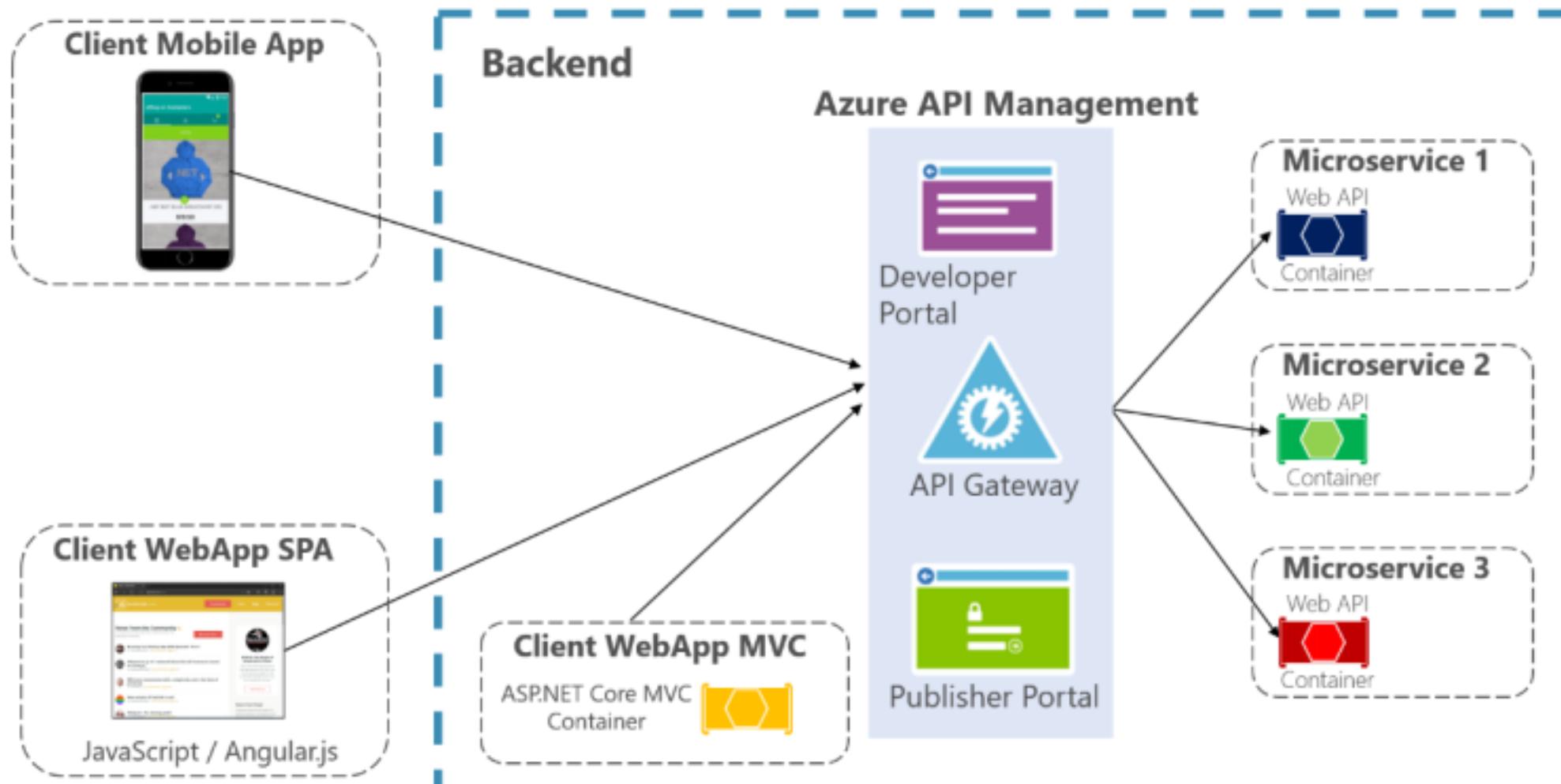
Using the **API Gateway Service**



Microservices – API Geteway Pattern

API Gateway with Azure API Management

Architecture

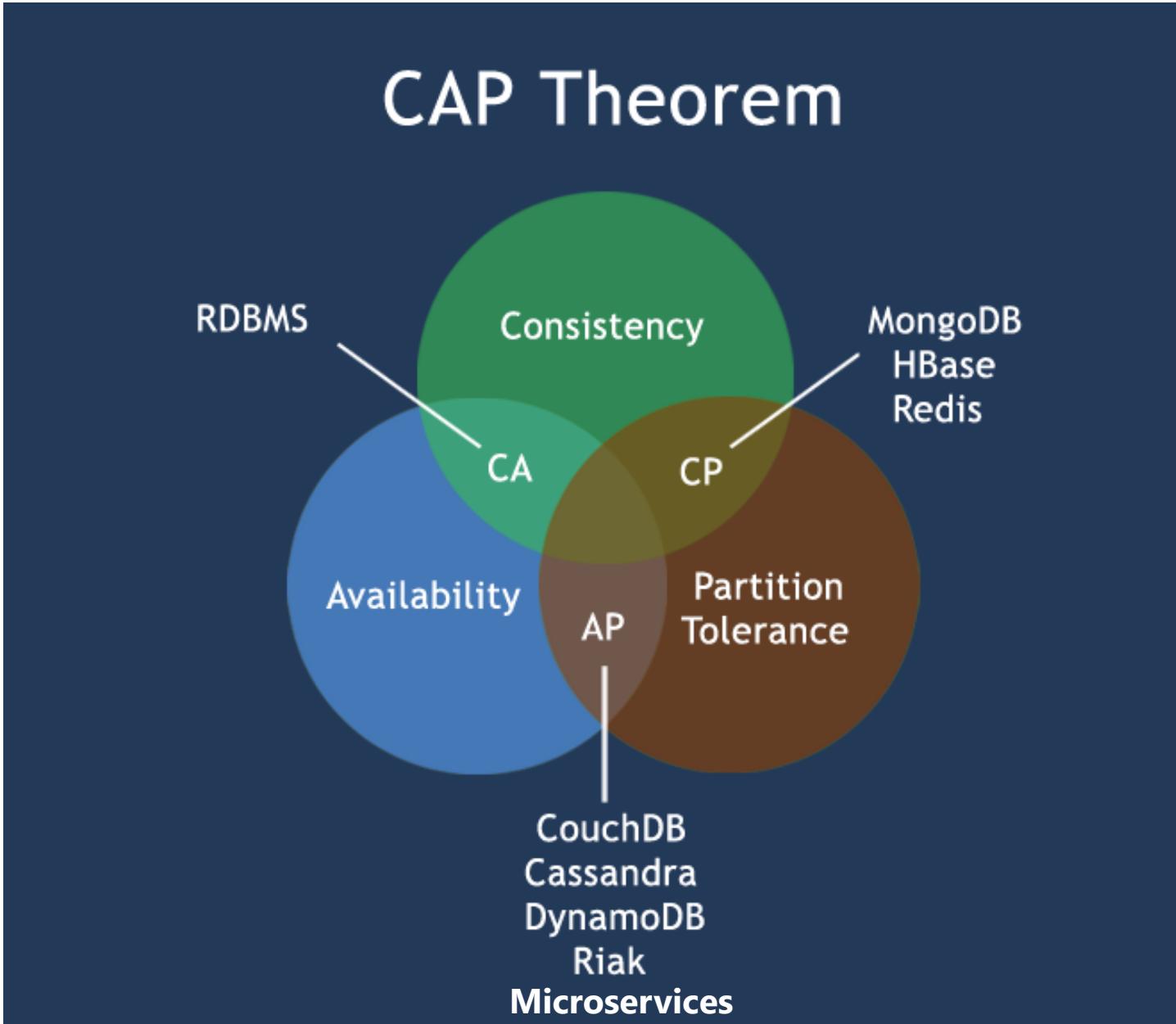


Microservices – Event-Driven Architecture

- Para lidar com eventual consistência, que é um gap do CAP Theorem a transação orientado à evento é um grande aliado do Microservice
- Um microservice publica um evento de integração quando algo acontece dentro de seu domínio e outro microservice precisa estar ciente disso.
- Os receptores podem atualizar suas próprias entidades de domínio, o que pode fazer com que mais eventos de integração sejam publicados
- Este mecanismo de publicação/assinatura de eventos geralmente é executado usando uma implementação de um Event Bus.
- Um ponto importante é que você pode querer se comunicar com vários microservices que estão inscritos no mesmo evento. Para fazer isso, você pode usar mensagens de publicação / assinatura com base em comunicação de evento

<ulili5/>

Microservices – Event-Driven Architecture

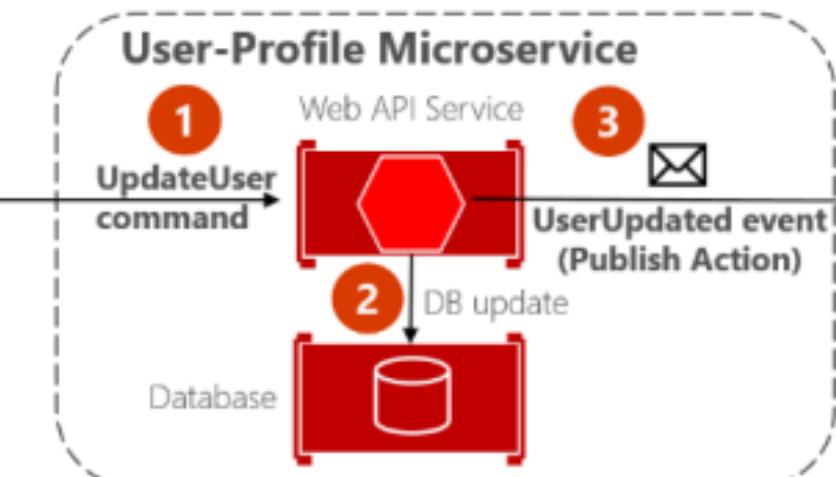


Microservices – Event-Driven Architecture

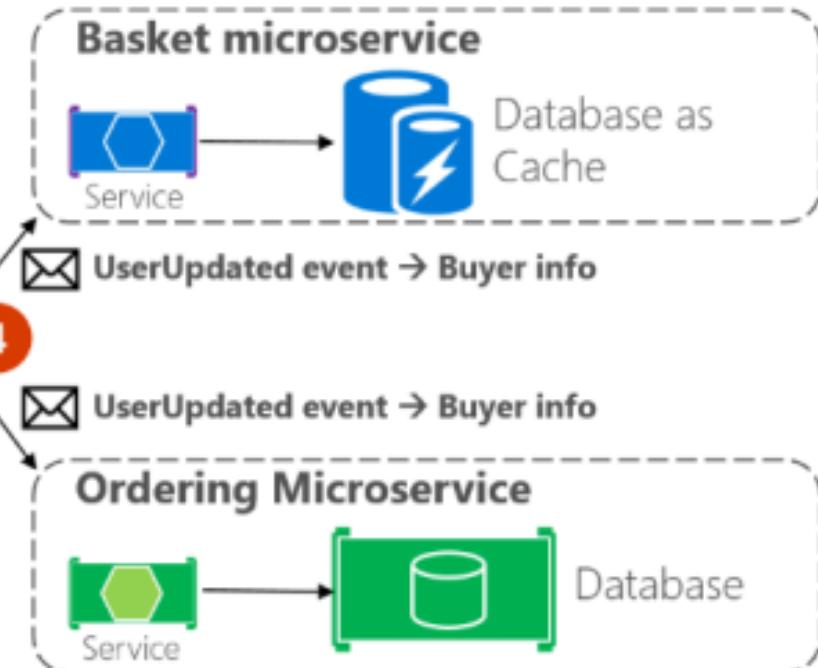
Asynchronous event-driven communication

Multiple receivers

Back end



Event Bus
(Publish/subscribe channel)



Eventual consistency across microservices based on event-driven async communication

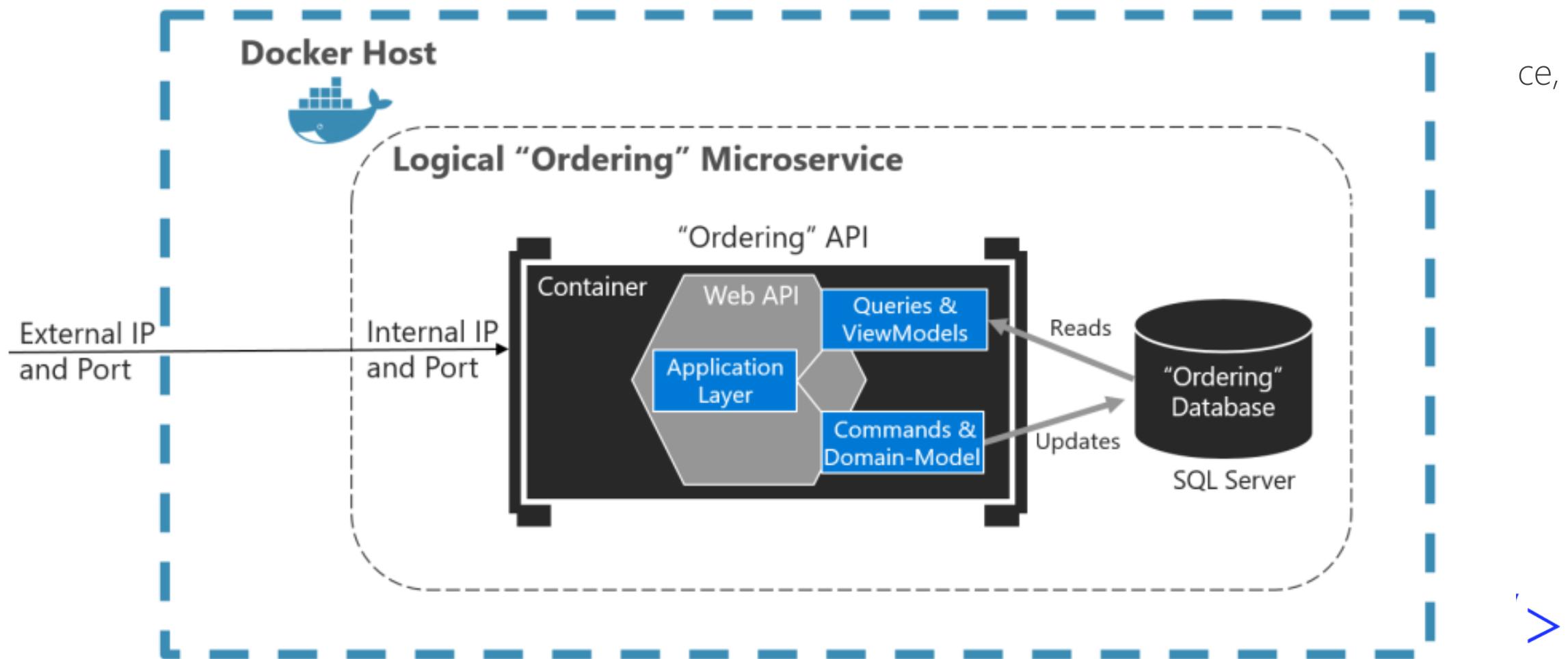
Microservices – CQRS (Command and Query Responsibility Segregation)

- CQRS é um padrão arquitetura que separa os modelos de leitura e gravação de dados
- Pode ser considerado um padrão baseado em comandos e eventos mais, opcionalmente, em mensagens assíncronas
- Em muitos casos, o CQRS é usado nos cenários mais avançados com necessidade de alta performance, podendo ter um banco de dados físico diferente para leituras (consultas) do que para escritas (atualizações)

Microservices – CQRS (Command and Query Responsibility Segregation)

Simplified CQRS and DDD microservice

High level design



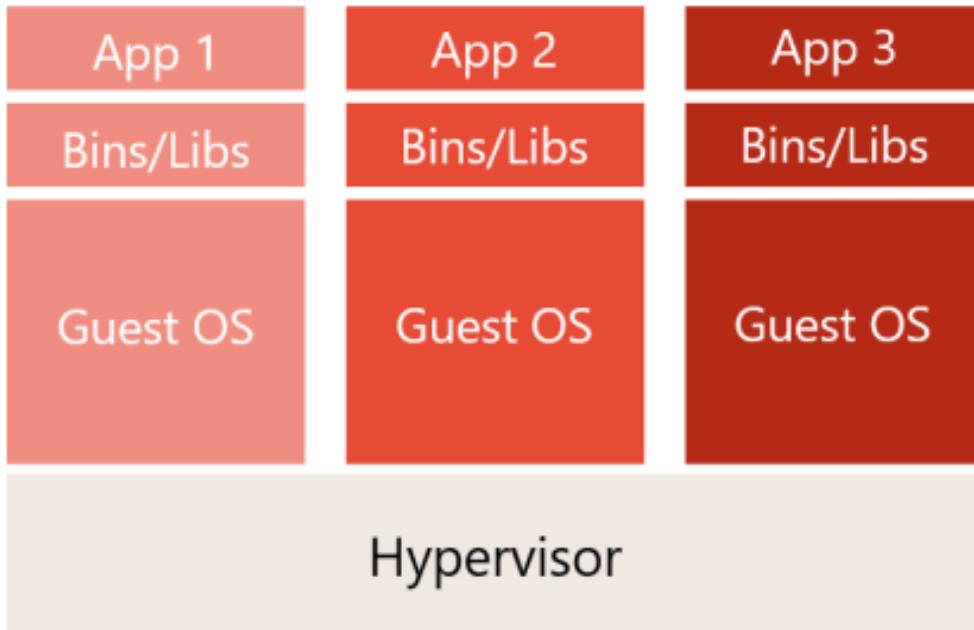
Microservices – Containers

- Containerização é uma abordagem para o desenvolvimento de software em que um aplicativo ou serviço, suas dependências e sua configuração (abstraída como arquivos de manifesto de implantação) são empacotados juntos como uma imagem de contêiner.
- O aplicativo contêinerizado pode ser testado como uma unidade e implantado como uma instância de imagem de contêiner para o sistema operacional host (SO).
- Os container também isolam aplicativos uns dos outros em um sistema operacional compartilhado.
- Os aplicativos containerizados são executados em cima de um host de contêiner que, por sua vez, é executado no sistema operacional (Linux ou Windows)
- Os containers têm consomem memória significantemente menor do que as imagens da máquina virtual (VM)

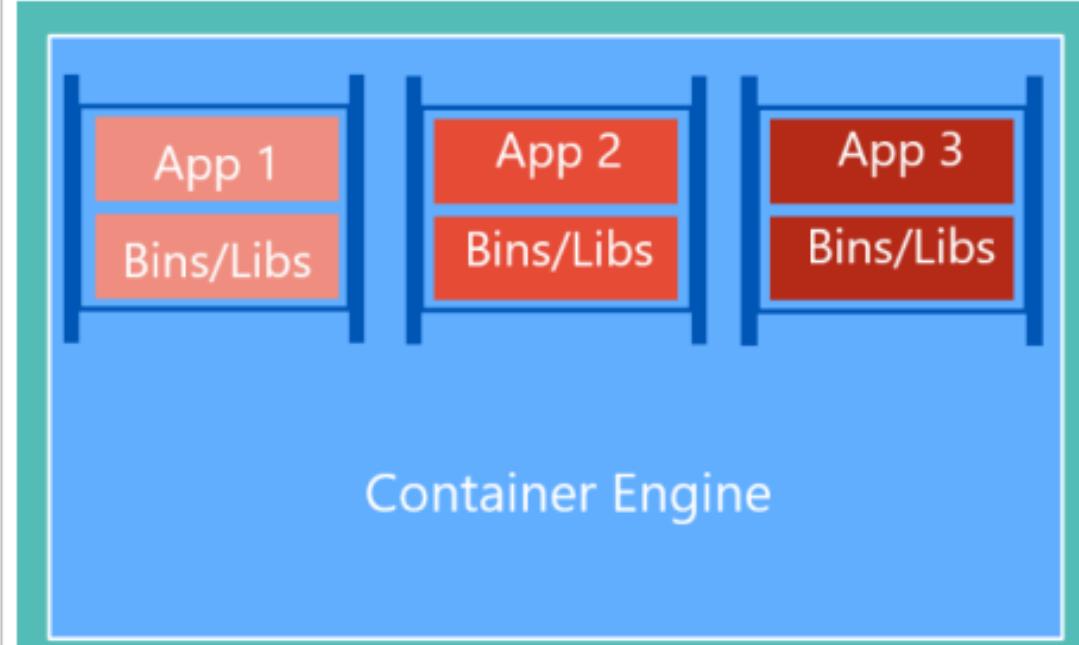
<ulili5/>

Microservices – Containers

Virtual Machines



Docker Containers



Operating System

Host Operating System

Infrastructure



Infrastructure

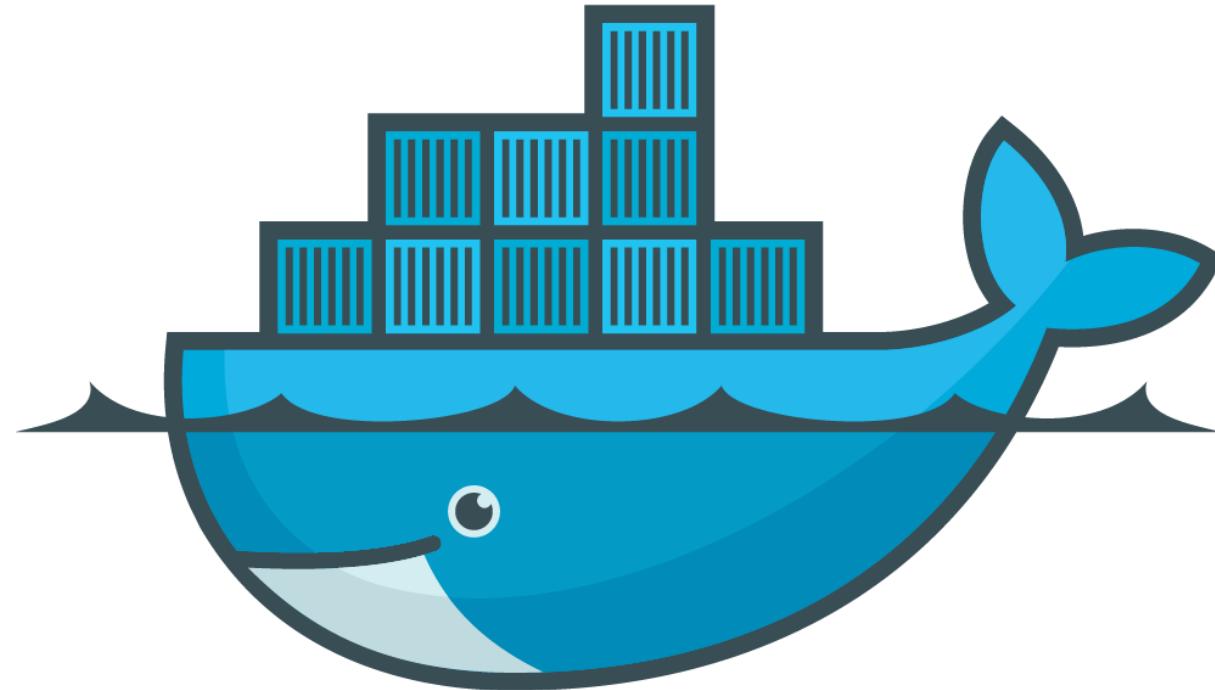


Microservices – O que é Docker?

- Docker virou sinônimo de container.
- O Docker é um projeto de código aberto para automatizar a implantação de aplicativos como containers portáteis e auto-suficientes que podem ser executados na nuvem ou no local.
- A Docker também é uma empresa que promove e desenvolve essa tecnologia, trabalhando em colaboração com fornecedores de nuvem, Linux e Windows, incluindo Microsoft

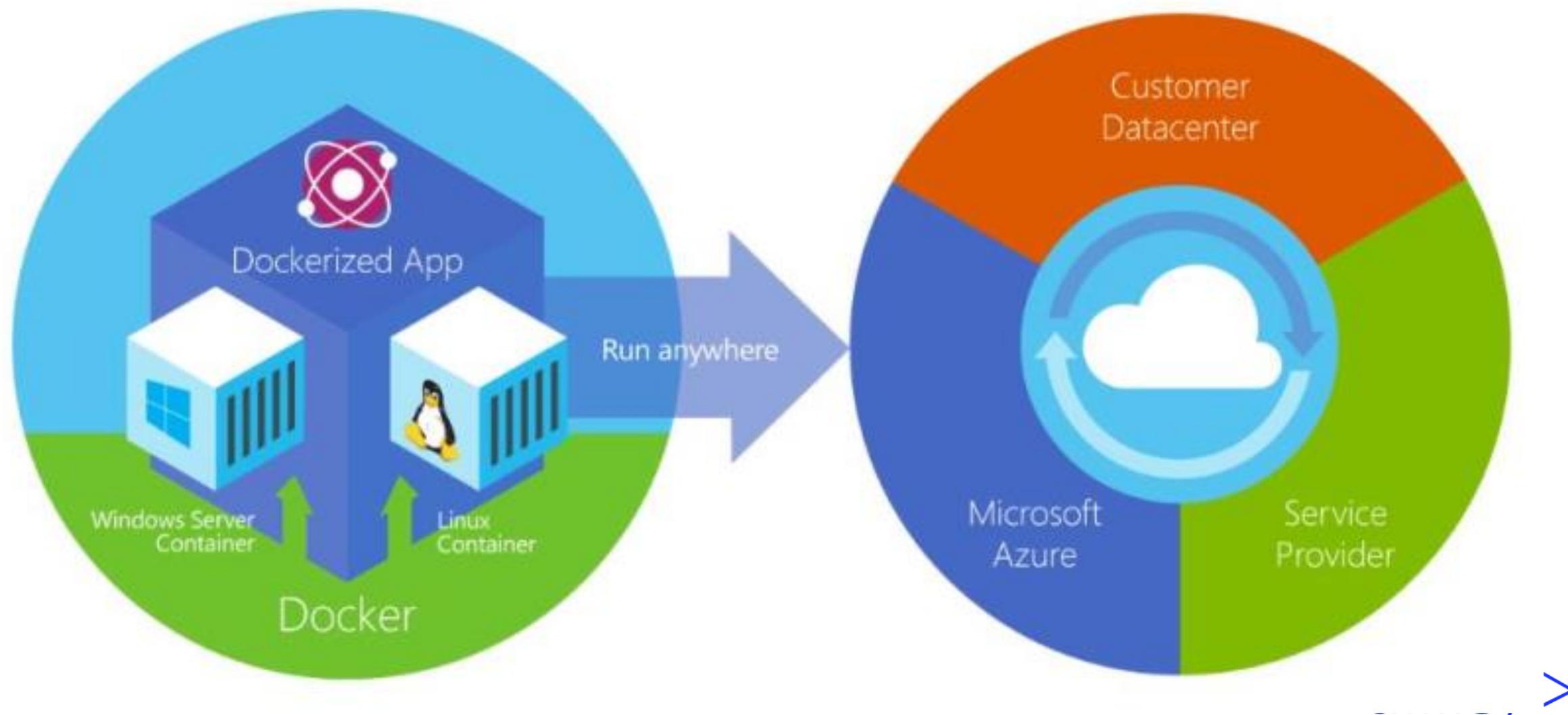
<ulili5/>

Microservices – O que é Docker?



docker

Microservices – O que é Docker?



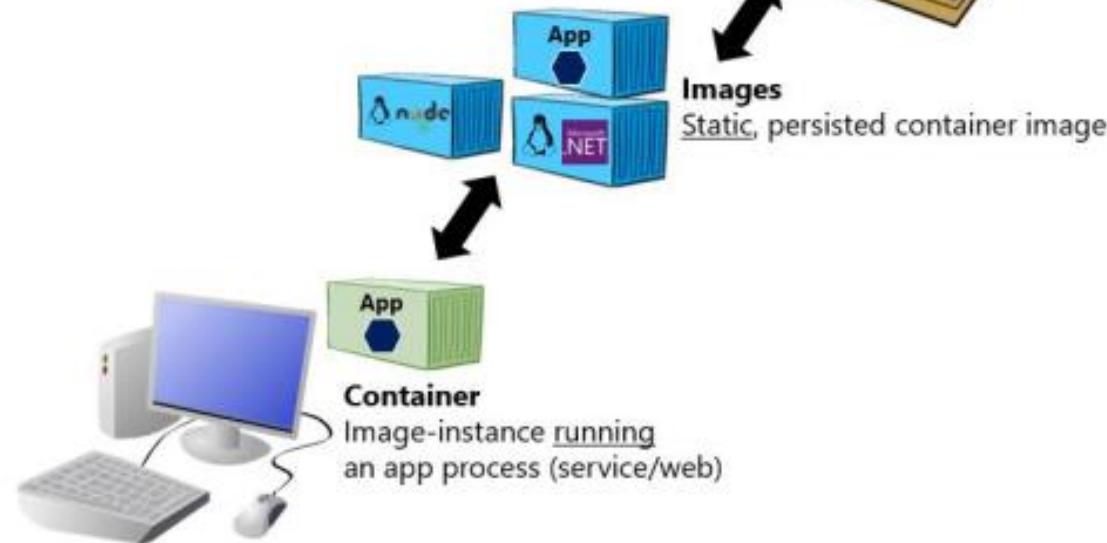
Microservices – Principais termos do Docker?

- Container image
- Container
- Tag
- Dockerfile
- Repository (repo)
- Registry
- Docker Hub
- Docker Trusted Registry (DTR)
- Docker Community Edition (CE)
- Docker Enterprise Edition (EE)
- Compose
- Cluster

<ulili5/>

Microservices – Principais termos do Docker?

Basic taxonomy in Docker



Hosted Docker Registry

Docker Trusted Registry on-prem.

Docker Hub Registry

Docker Trusted Registry on-cloud

Azure Container Registry

AWS Container Registry

Google Container Registry

Quay Registry

Other Cloud

On-premises

('n' private organizations)

Public Cloud

(specific vendors)



Microservices – Orquestradores

- Uma ferramenta que simplifica o gerenciamento de clusters e hosts Docker.
- Permitem que você gerencie suas imagens, contêineres e hosts através de uma interface de linha de comando (CLI) ou uma interface gráfica.
- Você pode gerenciar rede de contêineres, configurações, balanceamento de carga, descoberta de serviço, alta disponibilidade, configuração do host Docker e muito mais.
- Um orquestrador é responsável por executar, distribuir, escalar e recuperar tasks em uma coleção de nós.
- Os principais Orquestradores são **Mesosphere DC / OS**, **Kubernetes**, **Docker Swarm** e **Azure Service Fabric**

Microservices – Orquestradores



Microservice Platform
(Orchestrators/Clusters)

<ulili5/>

Microservices não é uma bala de prata!!

- Assim como tudo na área de tecnologia, não existe uma solução que se aplica a todos os problemas!!! Microservices também tem os seus pontos fracos e cenários onde não é aconselhado.
 - Como todo aspecto funcional é um serviço individual, então, em um grande projeto, existem muitos serviços. A monitoração desses serviços aumenta a sobrecarga.
 - Quando há uma falha no serviço, rastreá-lo pode ser um trabalho árduo (recomendável usar Application Insights).
 - Chamadas de serviço um para o outro, então o rastreamento do caminho e a depuração também podem ser difíceis (recomendável usar Visual Studio Docker tools ou VS 2017).
 - Cada serviço gera um registro, portanto, se não haver monitoração central do log. Isso pode ser algo doloroso, e precisamos de um sistema de gerenciamento de registro muito bom para isso (recomendável usar Application Insights).
 - Com microservices, cada serviço se comunica através de API / chamadas remotas, que têm mais sobrecarga do que com as chamadas de comunicação interprocesso do software monolithic.

<ulili5/>

Microservices – cenários de uso

- Aplicações que demandam Alta-Disponibilidade
- Serviços escaláveis
- Analise e processamento de dados não estáticos
- Aplicações interativas baseadas em sessão como Games por ex
- Análise de dados e fluxos de trabalho
- Coleta de dados, processamento e IoT
- Aplicações de Chat

<ulili5/>

Microservices – cenários não recomendáveis

- CRUD simples
- Aplicações corporativas com poucos acessos
- Aplicações de relatórios ex BI
- SPA com poucos acessos
- Portais intranet
- Renderização de imagens e vídeos (* devido a limitações de placa de vídeo).

Até aqui tranquilo?

- Ficou claro o que é Microservices?
- Com relação as suas Características OK..?
- Ficou claro como é possível lidar com do sistema distribuído?
- Será que devo sair daqui pensando e mudar tudo?
- Com relação a onde aplicar clareou?

Então vamos pros demos, mas antes vamos conhecer o .NET Core >>>

.NET Core

ASP.NET Core 2.0

.NET Core - Introdução

- O .NET Core e o ASP.NET Core oferecem várias vantagens comparados ao desenvolvimento .NET tradicional. Você deve usar o .NET Core para seus aplicações levando em conta o seguintes fatores:
 - Cross-platform
 - Usar Microservices
 - Docker containers
 - Alta performance e escalabilidade

*Desenvolvimento tradicional com .NET podem suportar esses requisitos, mas o ASP.NET Core e o .NET Core foram otimizados para oferecer suporte melhorado para os cenários acima

.NET Core – Aplicações modernas

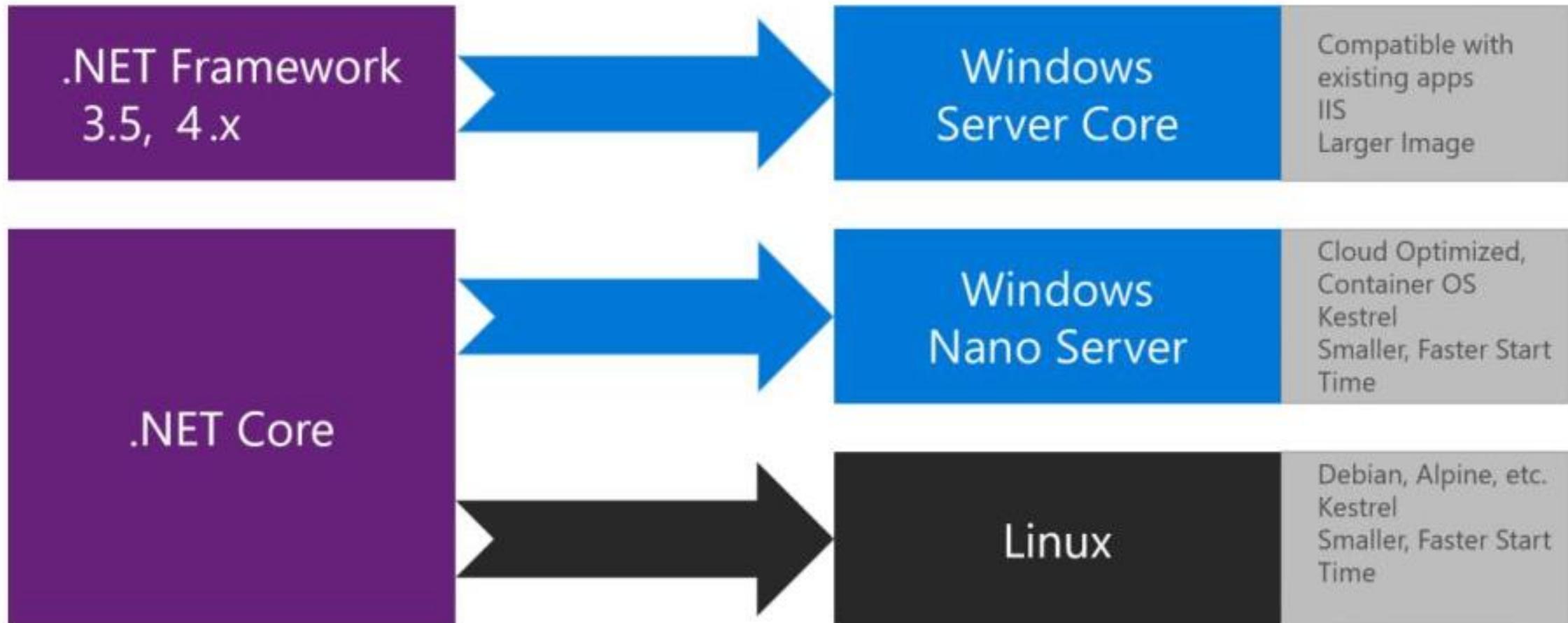
- As aplicações web modernas têm maiores expectativas dos usuários e maiores demandas como nunca antes.
- Espera-se que as aplicações Web de hoje estejam disponíveis 24 horas por dia, a partir de qualquer lugar do mundo, e sejam responsivos de praticamente qualquer dispositivo ou tamanho de tela.
- As aplicações Web devem ser seguros, flexíveis e escaláveis para atender às demandas em demanda.
- Cada vez mais, cenários complexos devem ser tratados por experiências de usuários avançadas construídas no cliente usando o JavaScript e se comunicando de forma eficiente através de APIs da web.

ASP.NET Core is optimized for modern web applications and cloud-based hosting scenarios. Its modular design enables applications to depend on only those features they actually use, improving application security and performance while reducing hosting resource requirements.

5/>

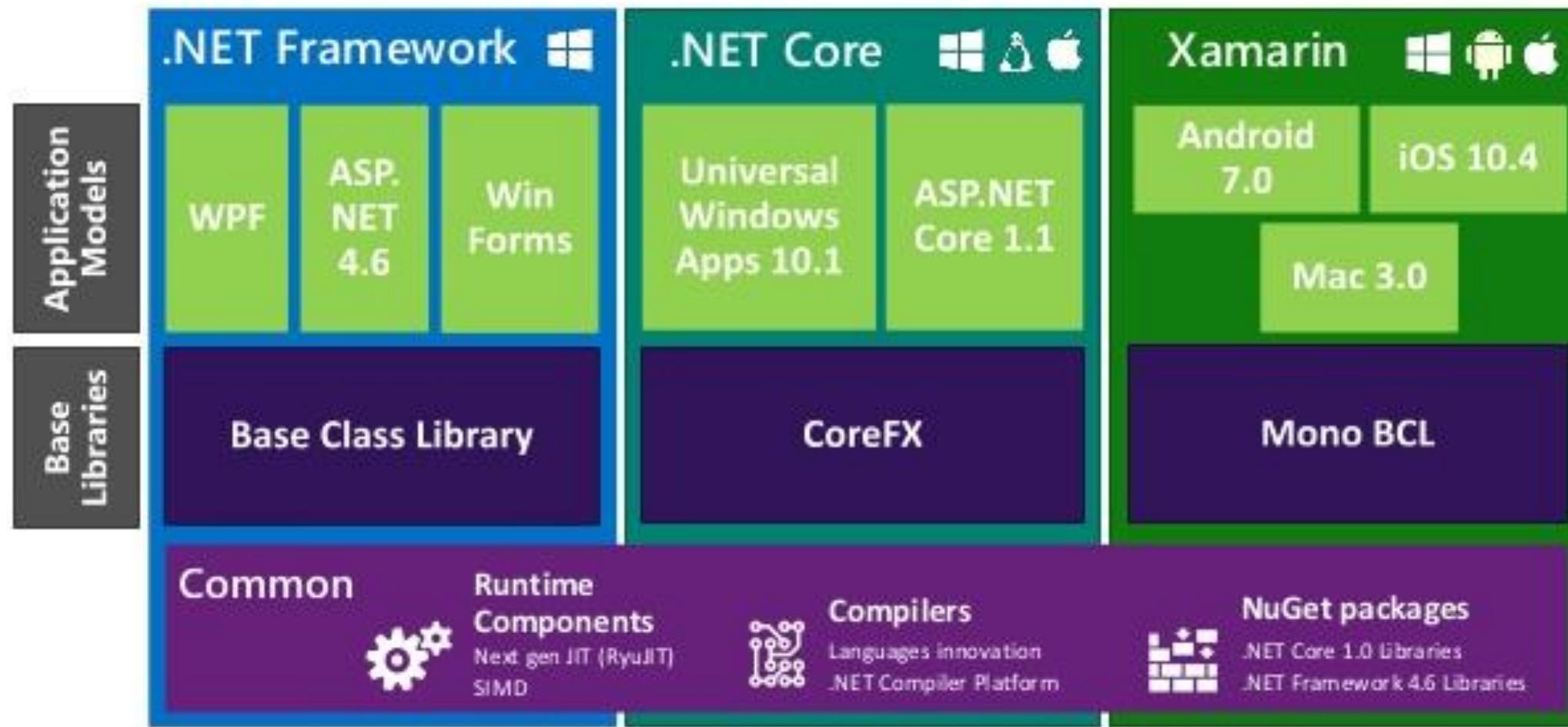
.NET Core – Cross-Plataforma

What OS to target with .NET containers



.NET Core – .NET Core vs .NET Framework

The big picture of .NET Platforms



.NET Core – .NET Standard

.NET Standard 2.0 Released!

Has much bigger API surface

Extended to cover intersection between .NET Framework and Xamarin

Also makes .NET Core 2.0 bigger as it implements .NET Standard 2.0

+20K

More APIs than
.NET Standard 1.x

Can reference .NET Framework libraries

Compatibility shim allows referencing existing .NET Framework binaries

No recompile required – also covers existing NuGet packages

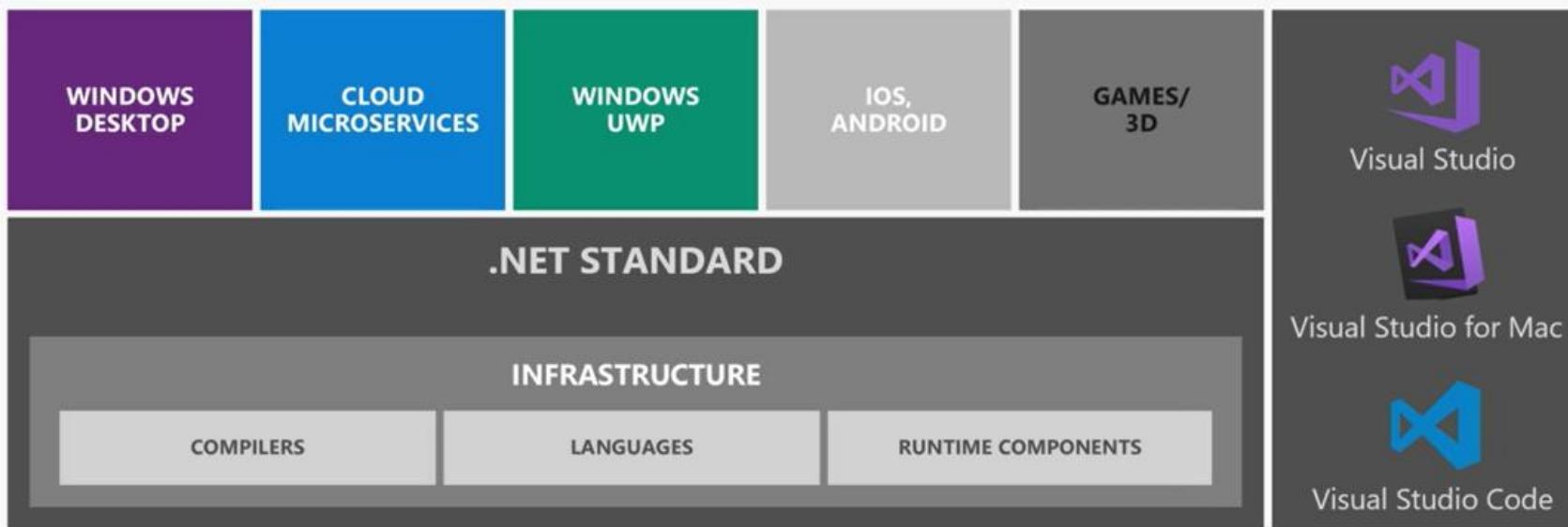
Limited to libraries that only use APIs that are available for .NET Standard

~70%

of NuGet packages
are API compatible

.NET Core – .NET Core vs .NET Framework

.NET Standard



.NET Standard allows sharing code, binaries, and skills between .NET client, server, and all flavors

.NET Standard provides a specification for any platform to implement
All .NET runtimes provided by Microsoft implement the standard

DEMO



Serverless

Serverless

"Server o quê?" "Quer dizer sem servidor?"
"Como assim aplicação sem servidor?" "Calma
aí, quero entender esse *trem!*". Então vamos
lá, traduzindo ao pé da letra você pode
entender que sejam essas coisas, mas eu lhe
diria que é quase isso!!! Ficou confuso?

<ulili5/>

Serverless – definição

- Serverless é a computação nas nuvens que lhe permite desenvolver e disponibilizar Aplicações, Serviços ou Funções sem que você tenha que pensar no servidor, com Serverless você não precisa provisionar, escalar e gerenciar nenhum servidor.

- Amazon AWS

<ulili5/>

Serveless – produtos

- AWS Lamda (Amazon)
- Azure Functions (Microsoft)
- Cloud Functionsbeta (Google)
- OpenWhisk (IBM).

<ulili5/>

DEMO

Valeu pela oportunidade!

Obrigado!!!

Ulili E. M. Nhaga

<https://www.linkedin.com/in/ulili5>