Domain-Driven Design (DDD) & Microservices: Patterns and Practices

Nevin Dong 董乃文
Principle Technical Evangelist
Microsoft Cooperation

Microsoft

New **patterns** and new **technologies**

Microservices

Autonomous
Bounded Context
Nomad & addressable services
Isolated
API Gateway
Decoupled
Events
Async. communication
Event Bus
Message Brokers
Service Discovery
Health Checks
Circuit Breakers
Transient Failures Handling
Resiliency
Commands
Retries with Exponential Backoff
Domain-Driven Design
Aggregates
CQRS simplified
Domain Events
Domain Entity
Mediator

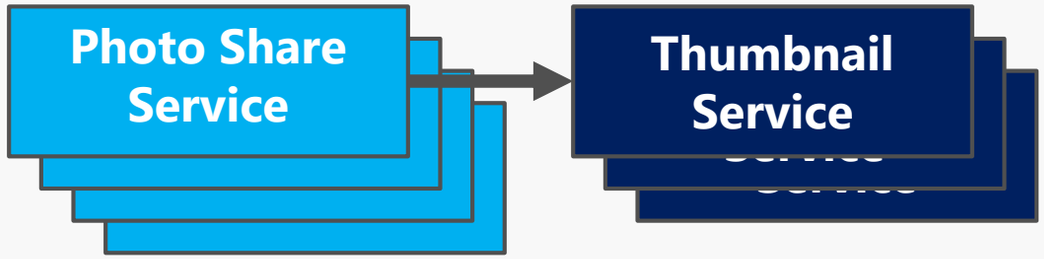
Docker Containers

Linux Containers
Docker Image
Docker Host
Windows Containers
Docker Registry
Docker Hub
RabbitMQ
Hyper-V Containers
Azure Service Bus
Azure Container Registry
NServiceBus
Orchestrators
MassTransit
Stateful Services
Brighter
Azure Service Fabric
Actors
Polly
Azure Container Service
Kubernetes
Docker Swarm
Mesos DC/OS

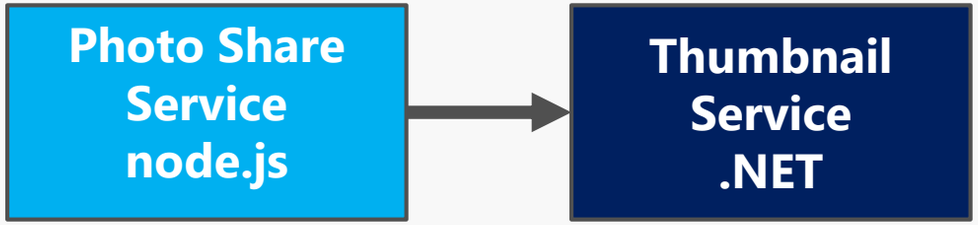
Microservices Architecture

Microservice architecture benefits

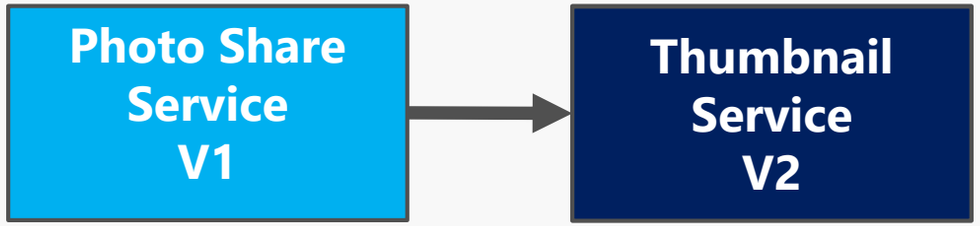
Scale Independently



Different Technology Stacks

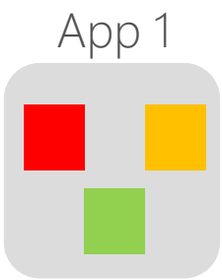


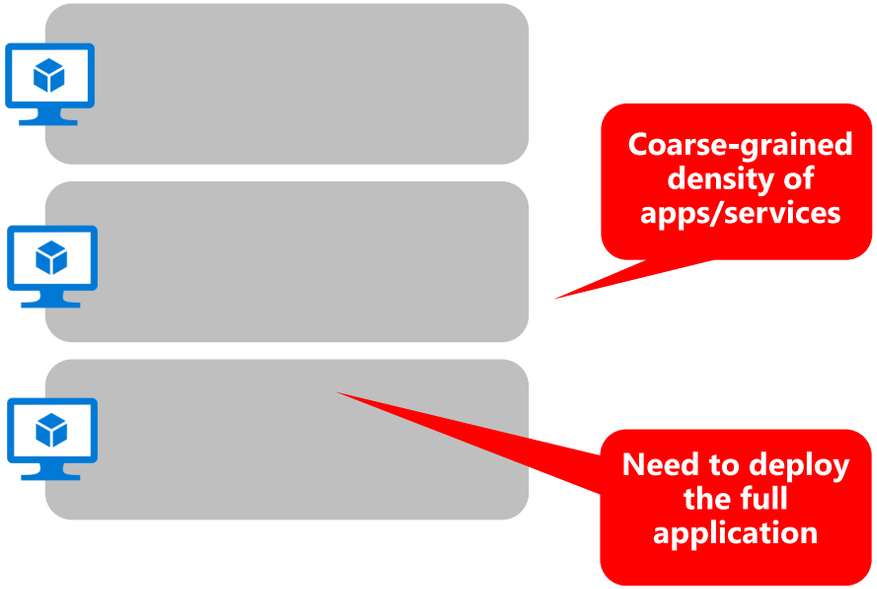
Independent Deployments



Traditional application approach

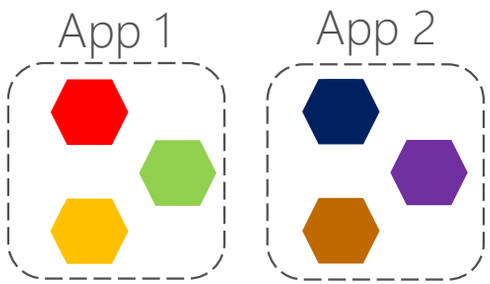
- A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
- Scales by cloning the app on multiple servers/VMs

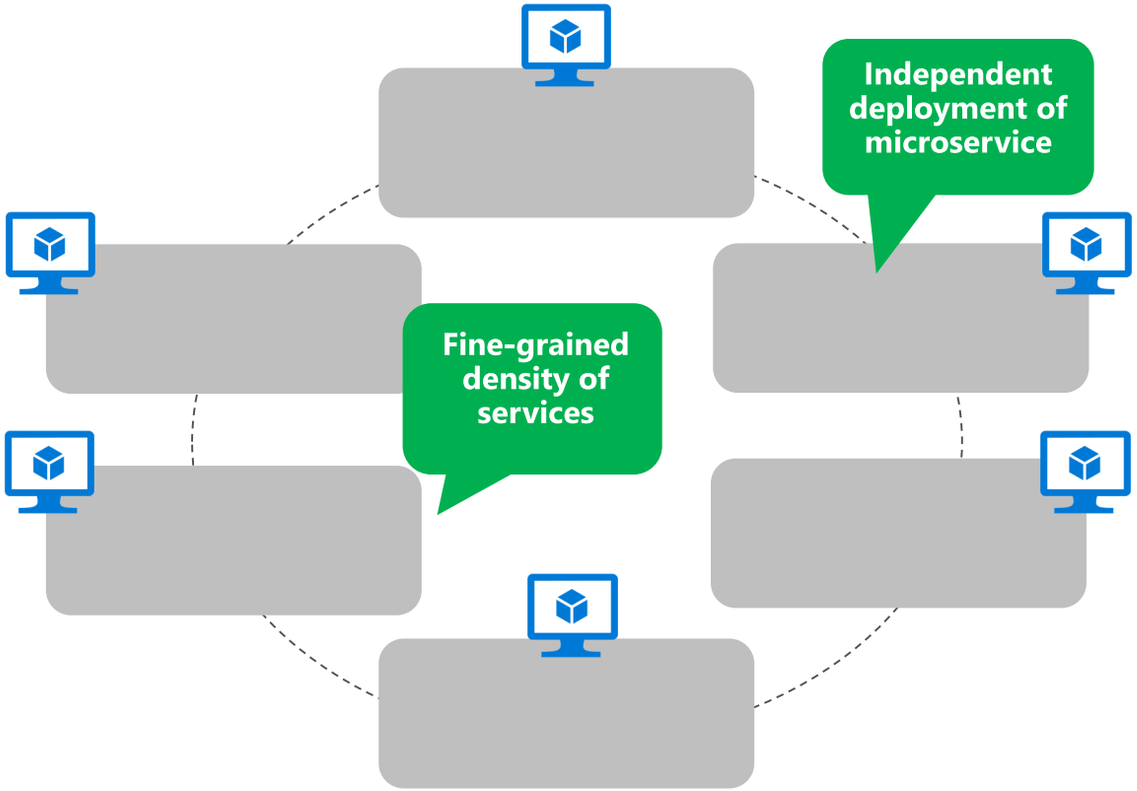




Microservices application approach

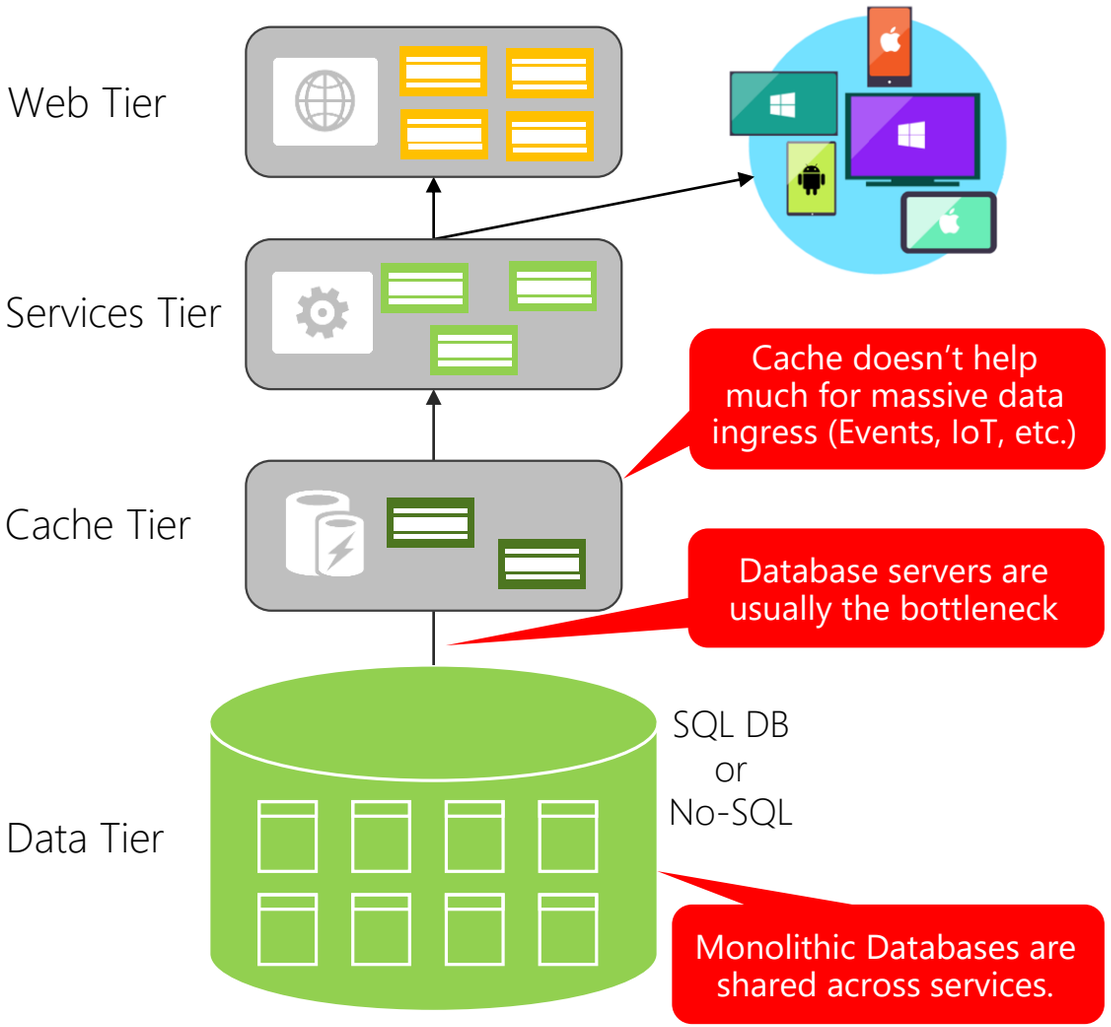
- A microservice application segregates functionality into separate smaller services.
- Scales out by **deploying each service independently** with multiple instances across servers/VMs





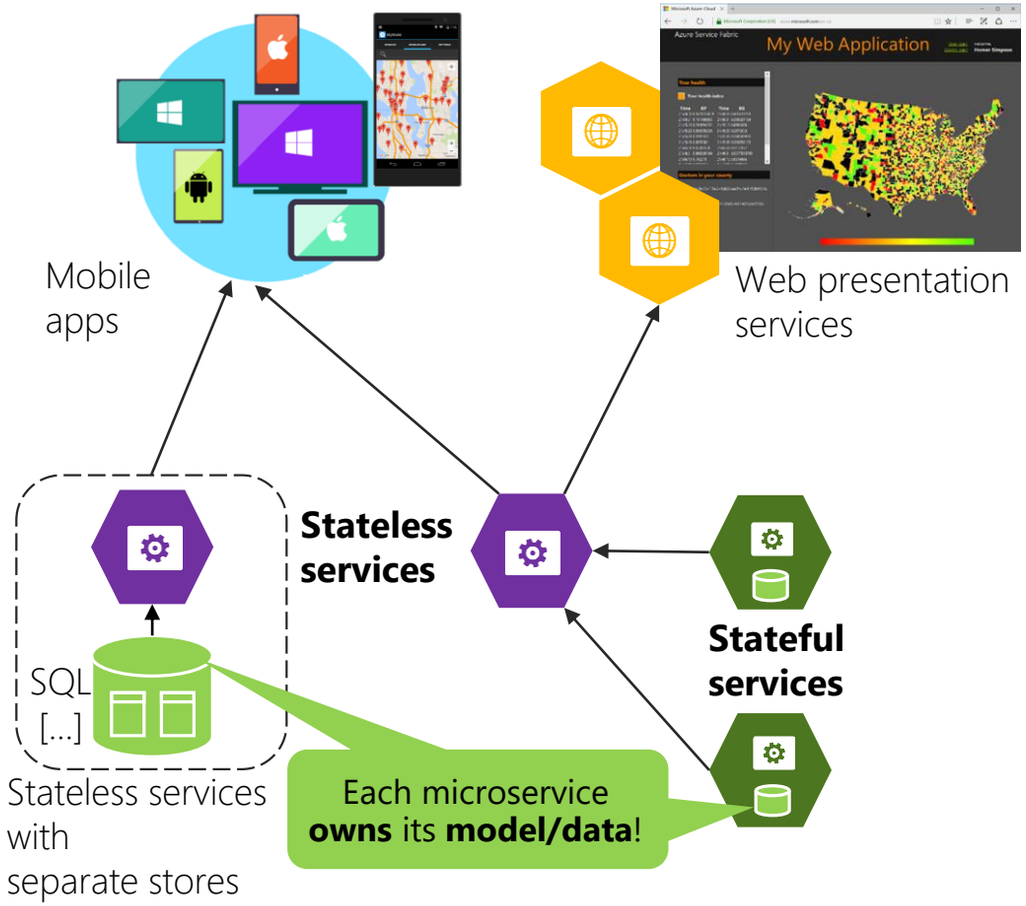
Data in Traditional approach

- Single monolithic database
- Tiers of specific technologies

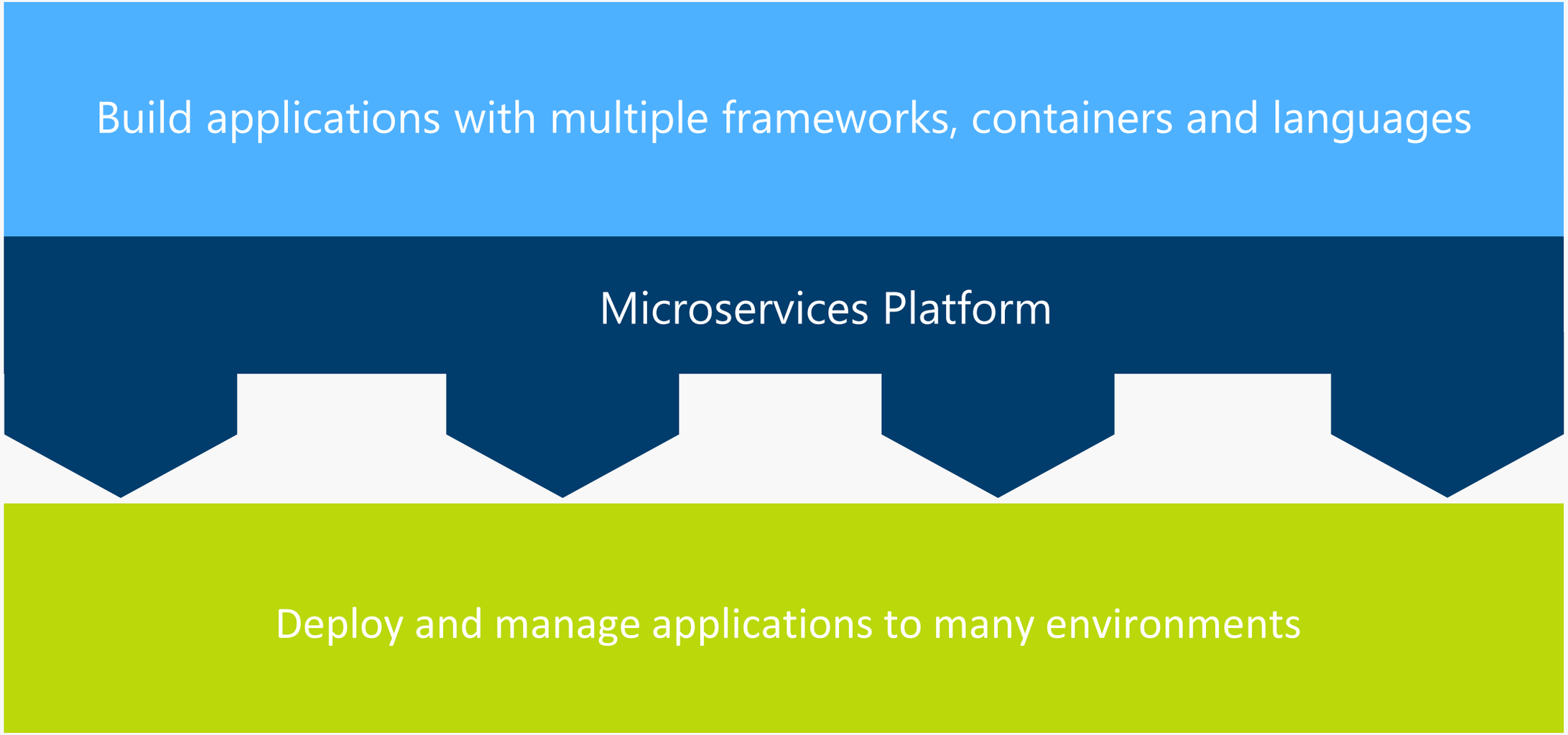


Data in Microservices approach

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



Microservices platform



Azure Service Fabric

Any OS, Any Cloud

containers and microservices

Lifecycle Management

Always On Availability

Orchestration

Programming Models

Health & Monitoring

Dev & Ops Tooling

Auto scaling

Dev Machine

Azure

On Premise Infrastructure

Other Clouds

Services Powered by Service Fabric

SQL Database

2.1 million DBs

Cosmos DB

Billions transactions/day

IoT Hub

Millions of messages

Event Hubs

60bn events/day

Microsoft runs its business on Service Fabric

Skype for Business

Skype

Cortana

Windows Intune

Intune

Microsoft Dynamics 365

Dynamics

Power BI

Designed for mission critical tier 1 workloads

30% of Azure cores run Service Fabric

Service Fabric on Azure

Microservices Platform

Highly scalable

24 X 7 High availability and failover

Windows and Linux container orchestration

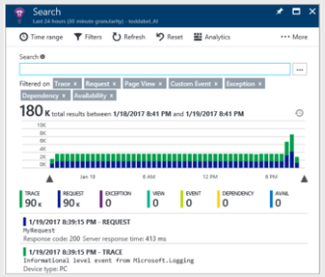
DevOps and Lifecycle management

Managed Service

Built-in auto scale

Automated platform upgrades

Built-in health and diagnostics



Integrated with AppInsights and OMS

Productive Development

Simple Programming Models for. NET, Java

Stateless and Stateful microservices

Local development identical to cloud development

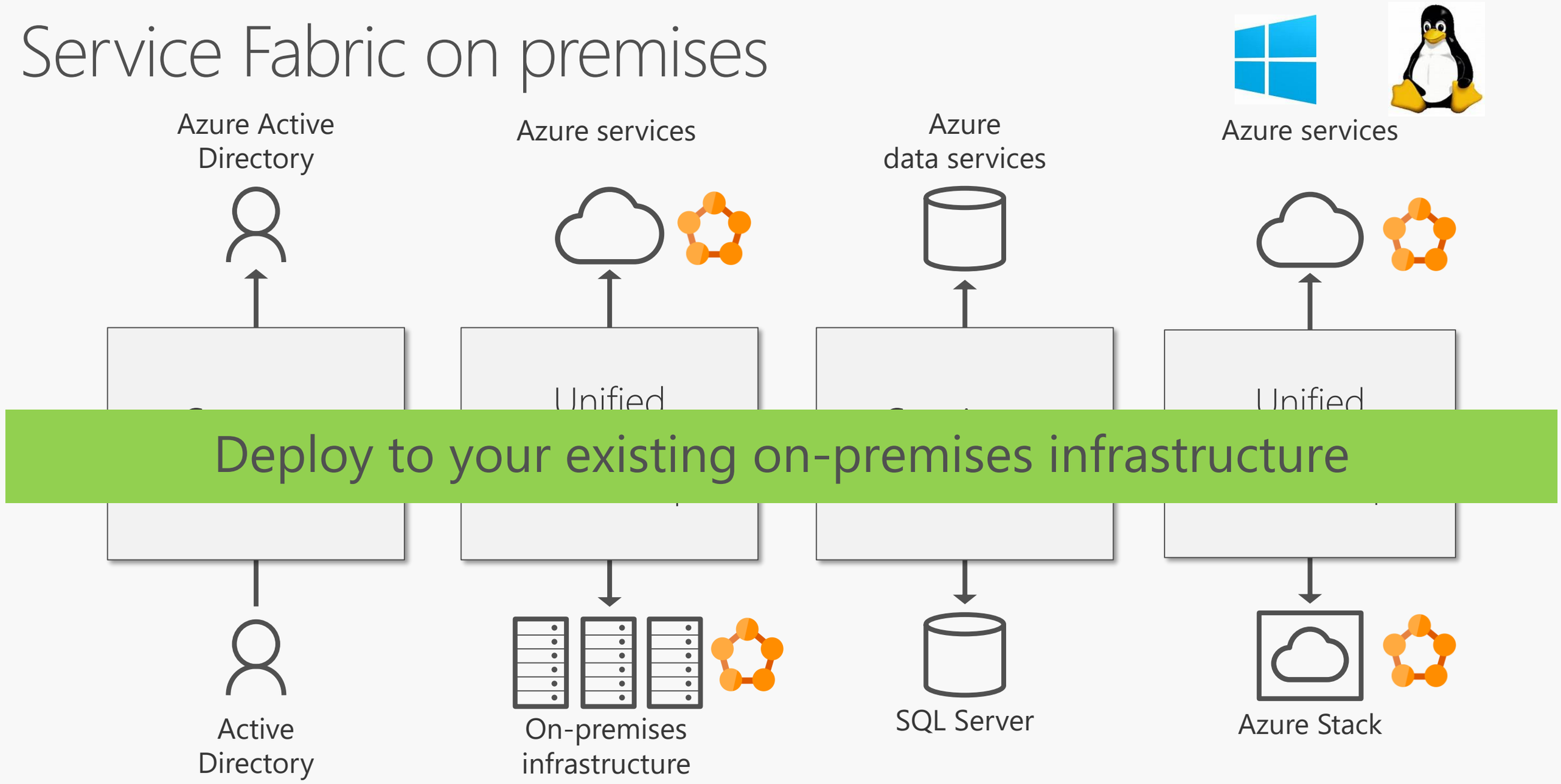
Visual Studio

Jenkins

eclipse

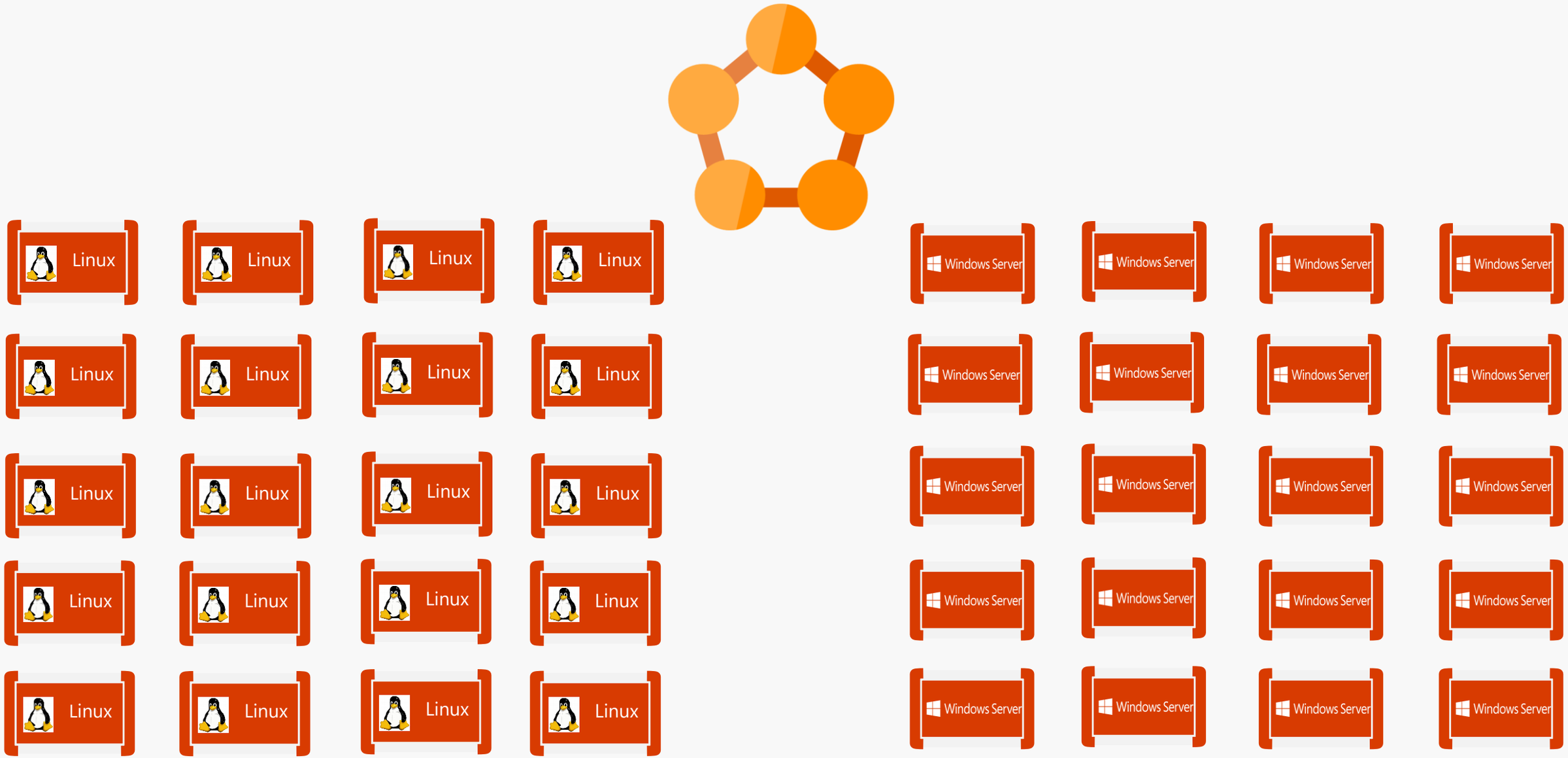
Tooling with Visual Studio, VSTS Eclipse & Jenkins

Service Fabric on premises



Deploy to your existing on-premises infrastructure

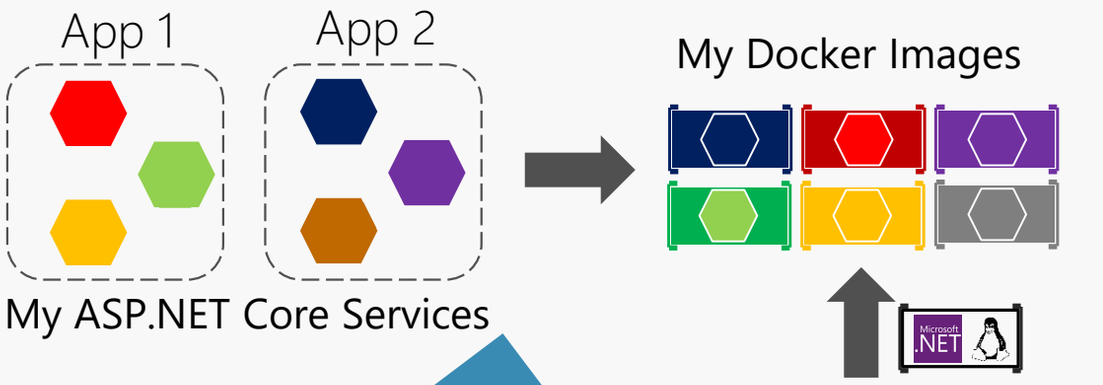
Service Fabric: Microsoft's Container Orchestrator



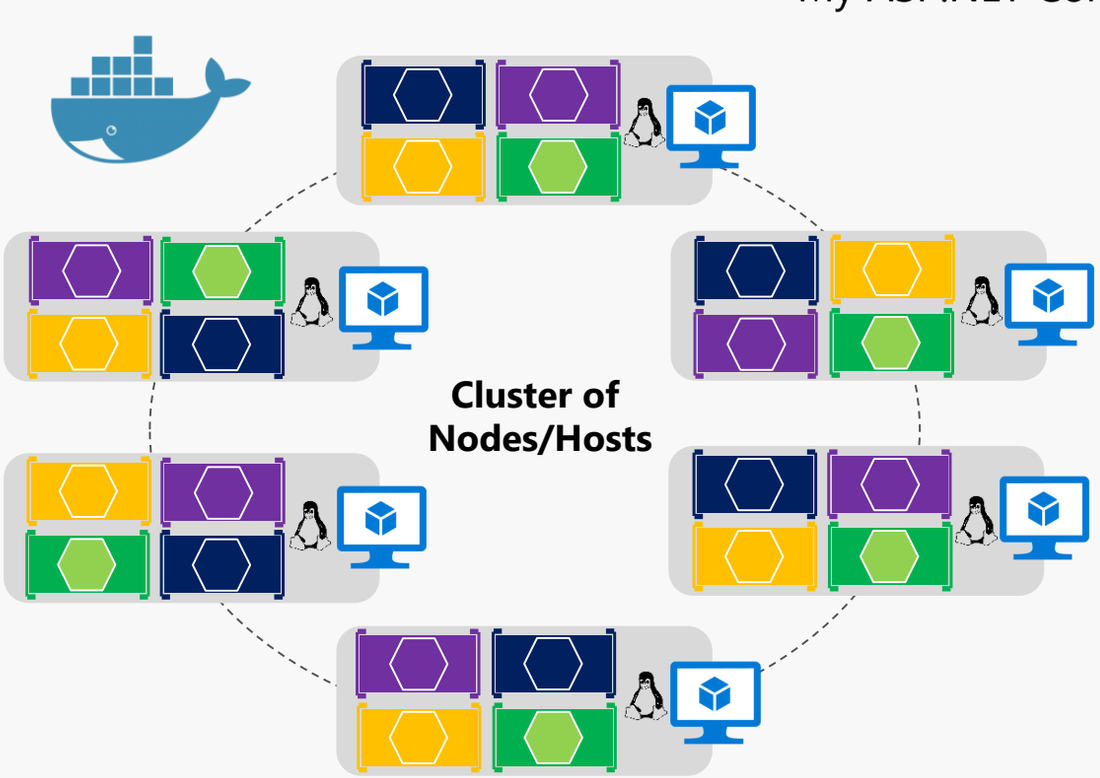
Orchestrator's Cluster managing microservices/containers

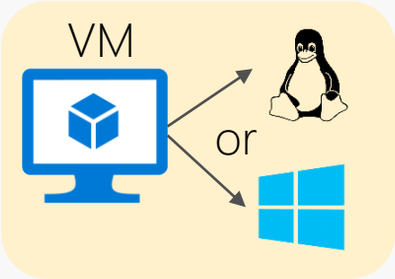
Clusters provide:

- *High scalability*
- *Automatic High Availability and resiliency*
- *High services density per host*

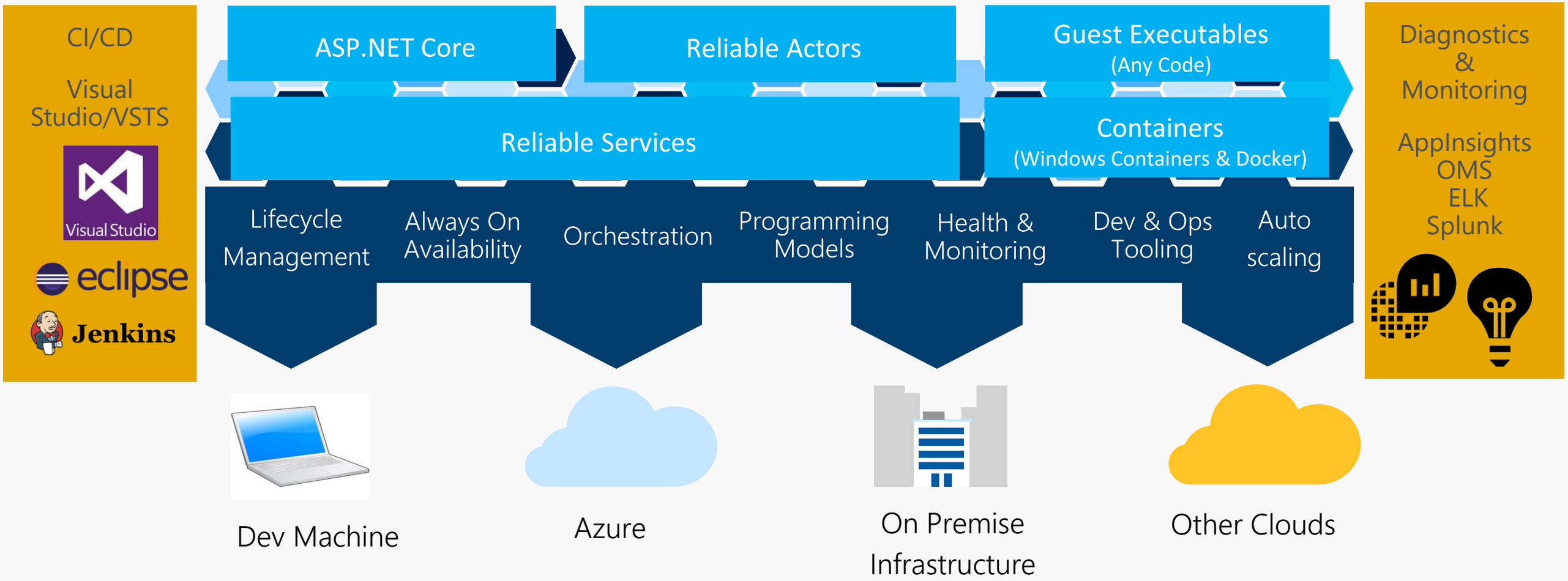


Official Docker Images
https://hub.docker.com





CI/CD, diagnostics and monitoring



Key patterns for microservices and Domain-Driven Design

Key Patterns for Microservices

1. Direct communication vs. API Gateway
2. Health checks
3. Resilient cloud applications:
   - Retries with exponential backoff plus Circuit breaker
4. Async. pub/subs communication (Event Bus)
5. Scale-out with Orchestrators

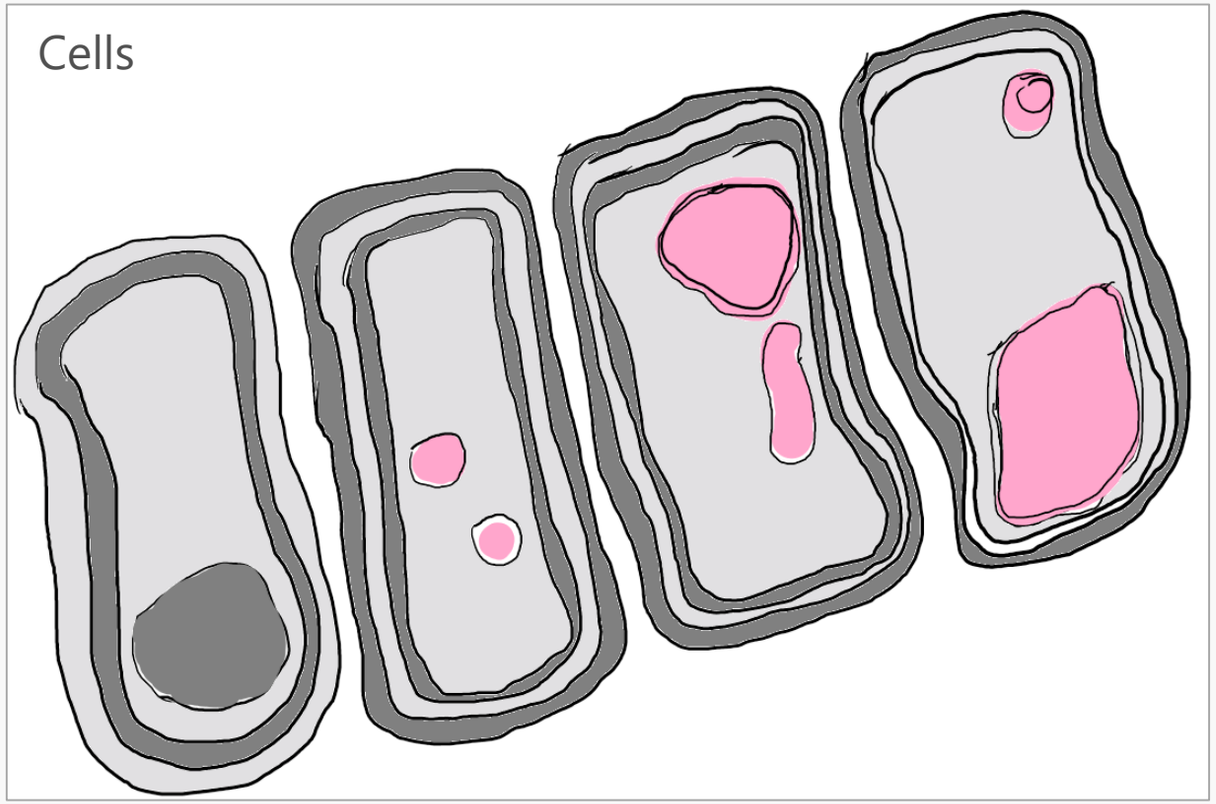
Domain-Driven Design (DDD) Patterns

Bounded Context == Business Microservice boundary

Use in your Core-Domain microservices, task oriented with lots of business rules & transactions

1. Simplified CQRS when using DDD in a microservice
2. Rich Domain Model vs. Anemic Domain Model
3. Domain Entity
4. Aggregates
5. Value Object
6. Domain Events (within a single microservice)

The Bounded Context pattern

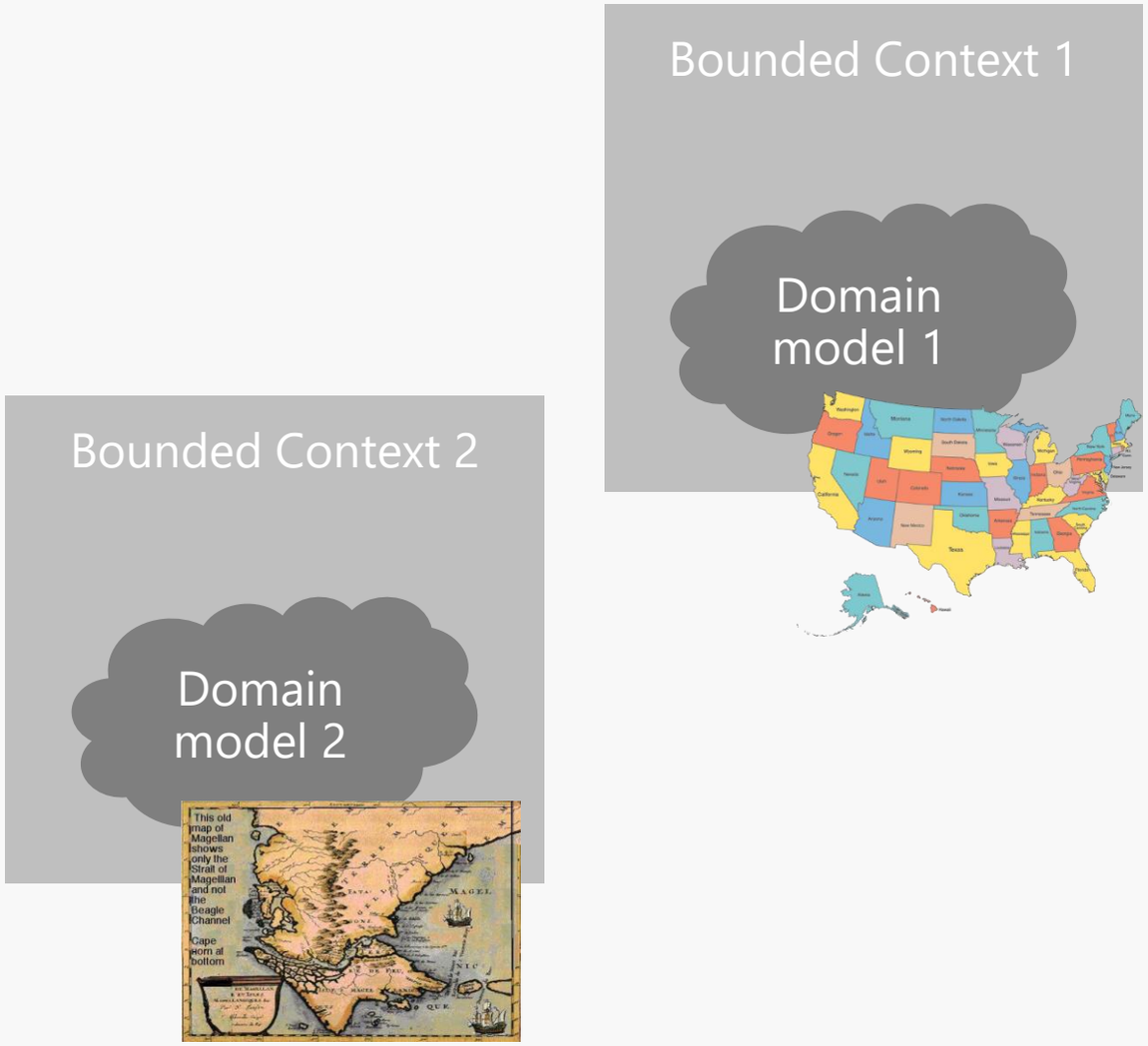


Independent
Autonomous
Loosely coupled composition

"Cells can exist because their membranes define what is in and out and determine what can pass"
[Eric Evans]

Bounded Context pattern in Domain-Driven Design

A domain model applies within a *Bounded Context*



In a typical enterprise system, there are multiple Bounded Contexts

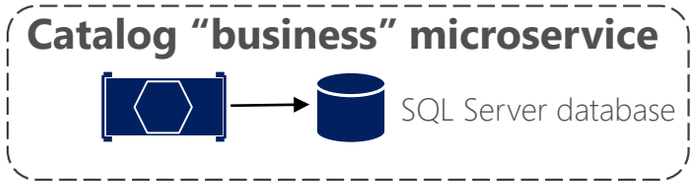
Thus, multiple domain models

Not one big domain model across the entire system!

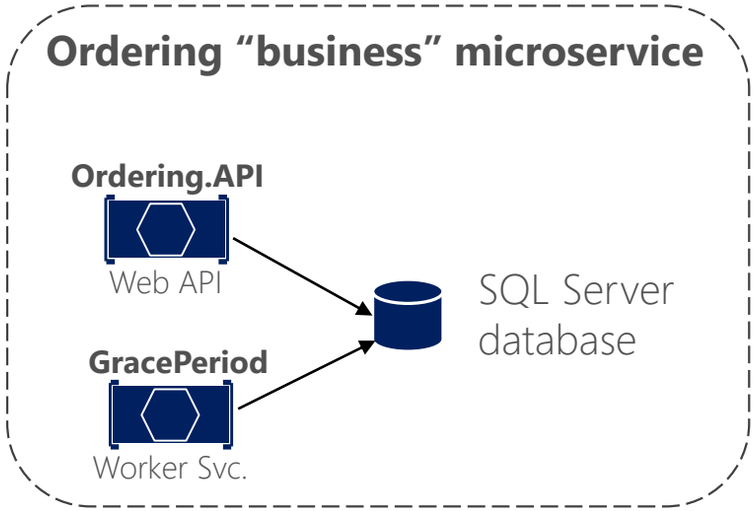
Bounded Context == “Business Microservice” boundary

Business/Logical Microservices (Bounded Contexts)

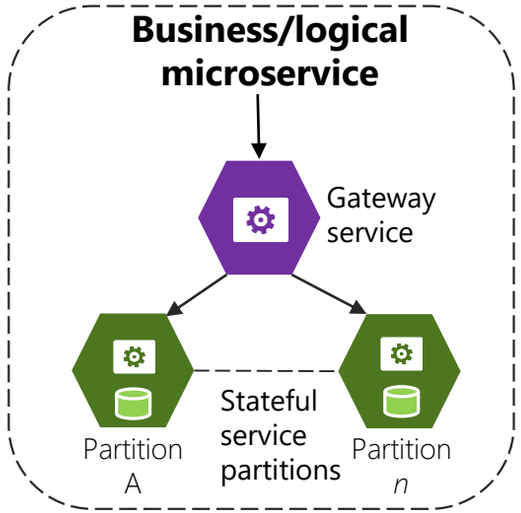
Example 1



Example 2



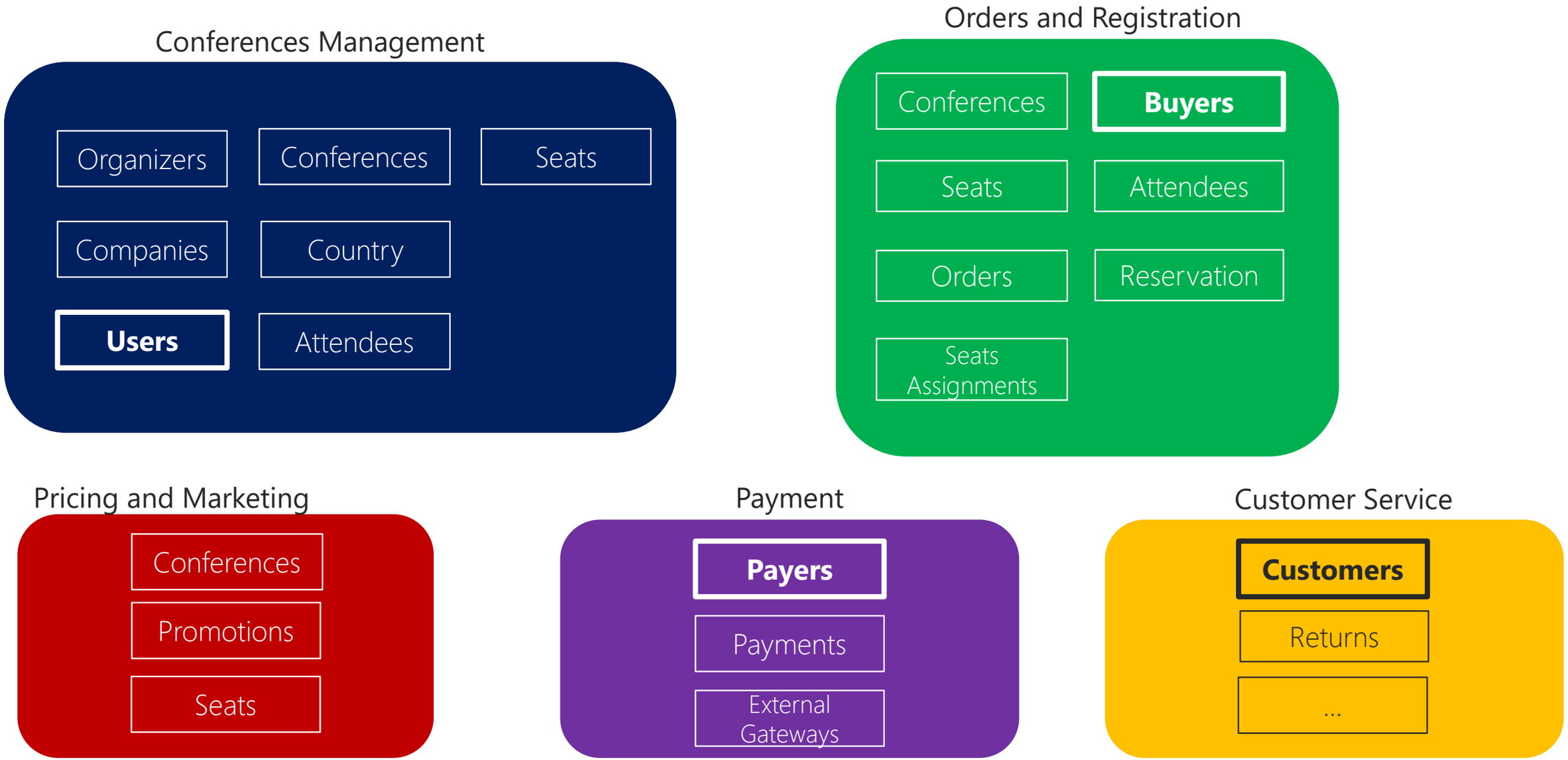
Example 3



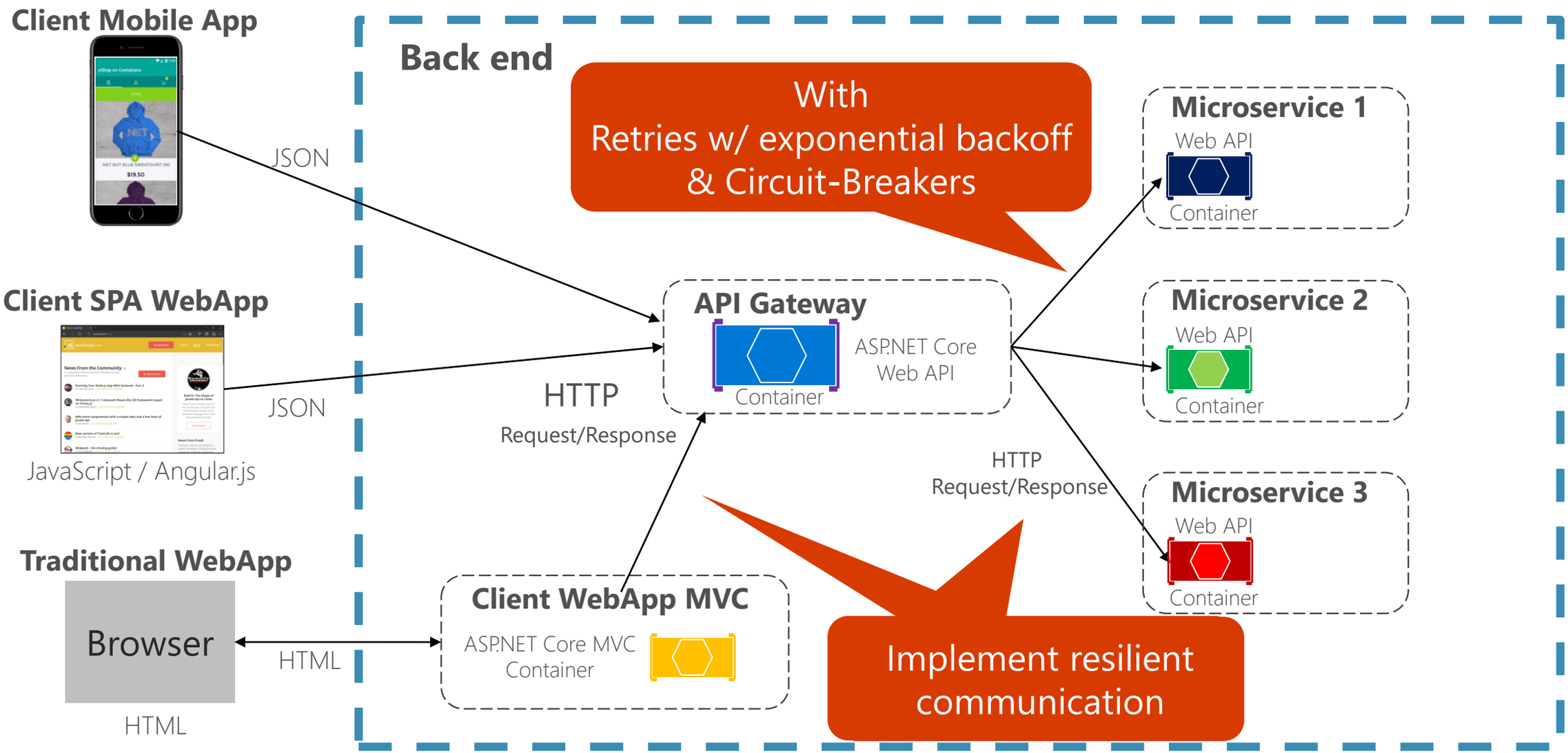
(Using Azure Service Fabric Stateful Reliable Services)

- The Logical Architecture can be different to the Physical/Deployment Architecture
- A Bounded Context can be implemented by 1 or more services (i.e. ASP.NET Web API)

Identifying a Domain Model per Microservice/BoundedContext

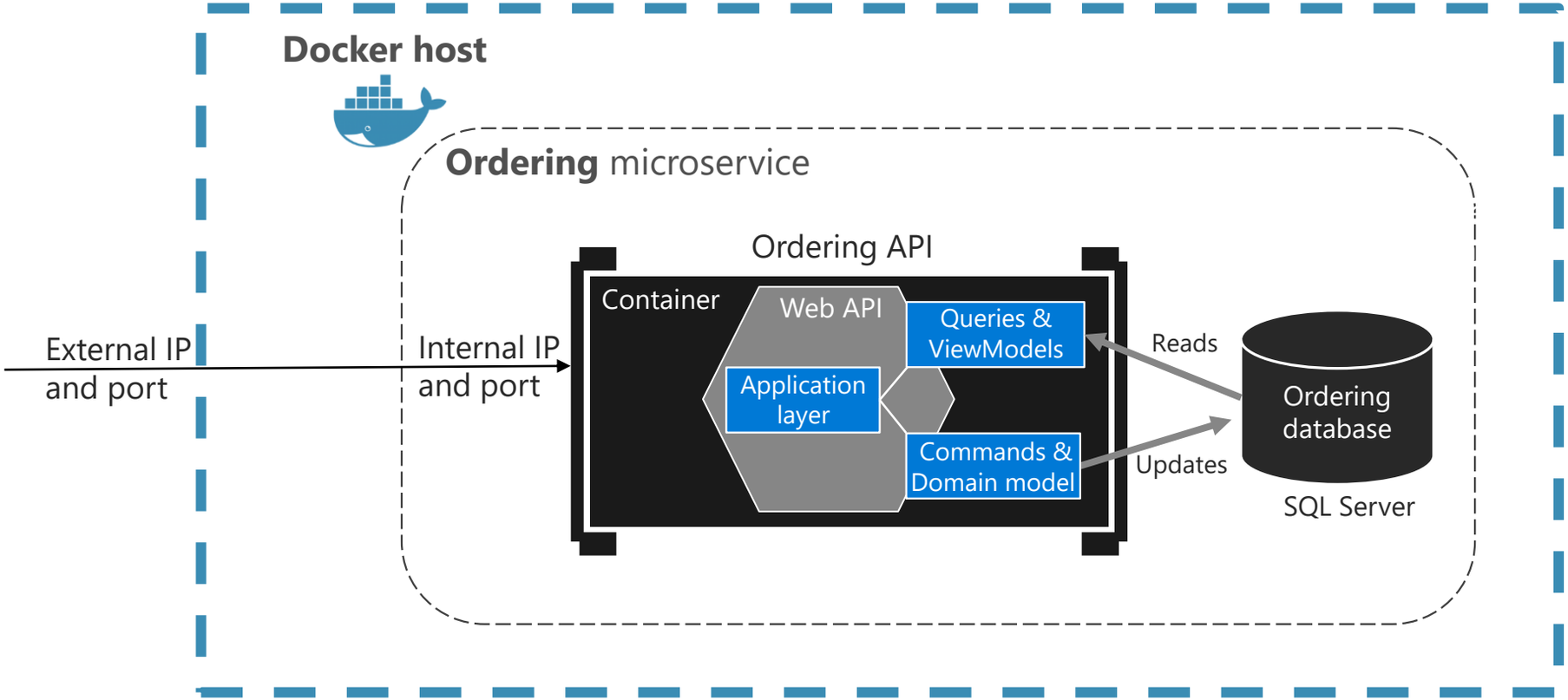


Building **resilient** cloud applications



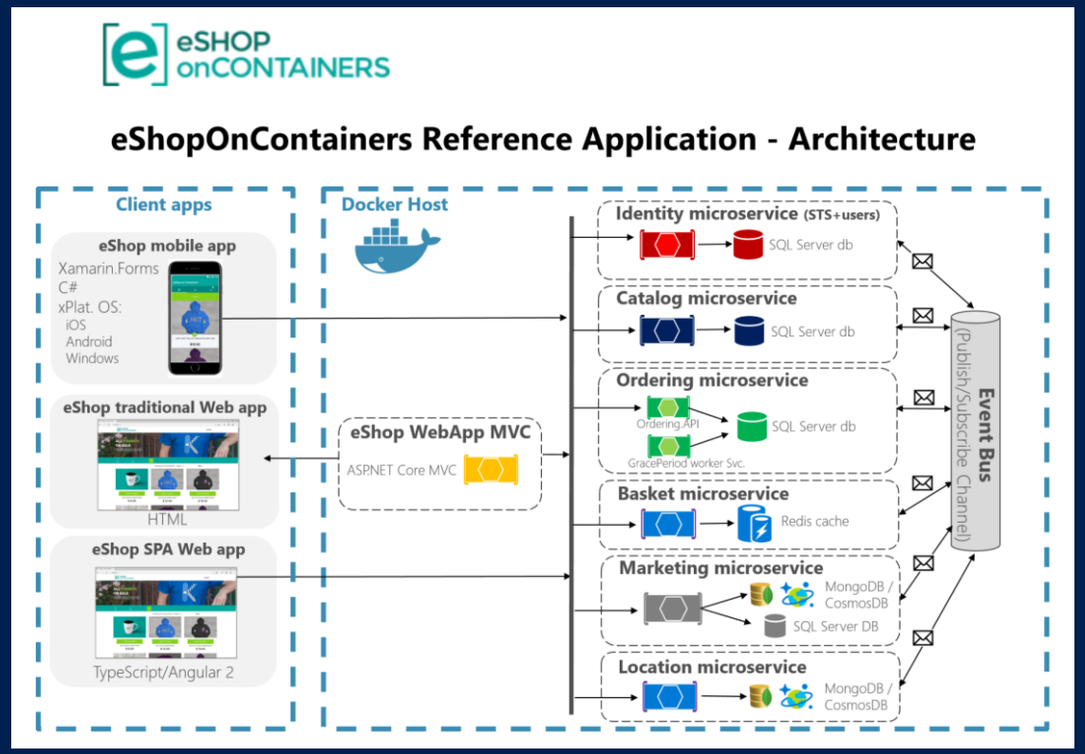
Simplified CQRS and DDD Microservice

High-Level Design

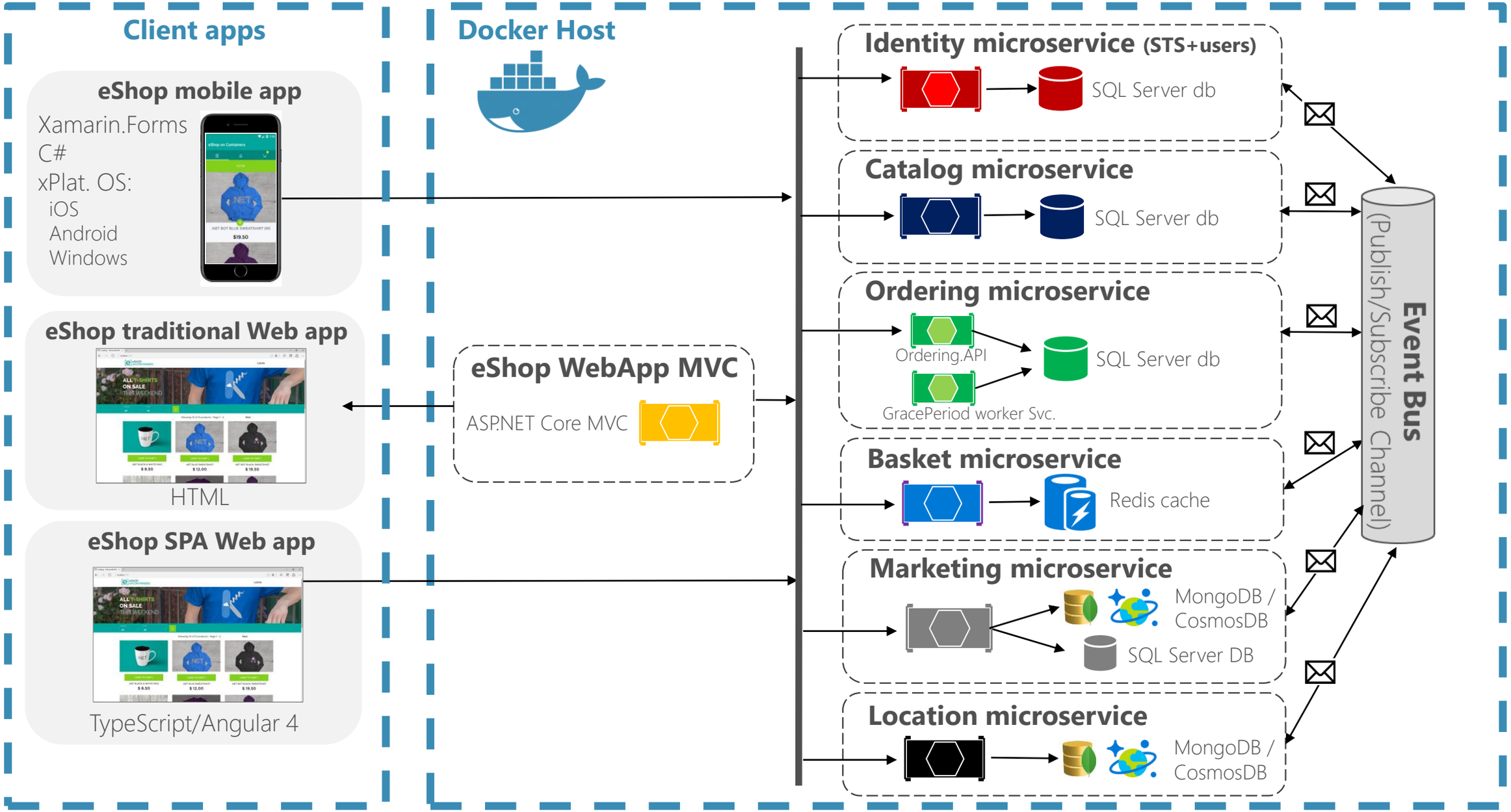


Practices: eShopOnContainer

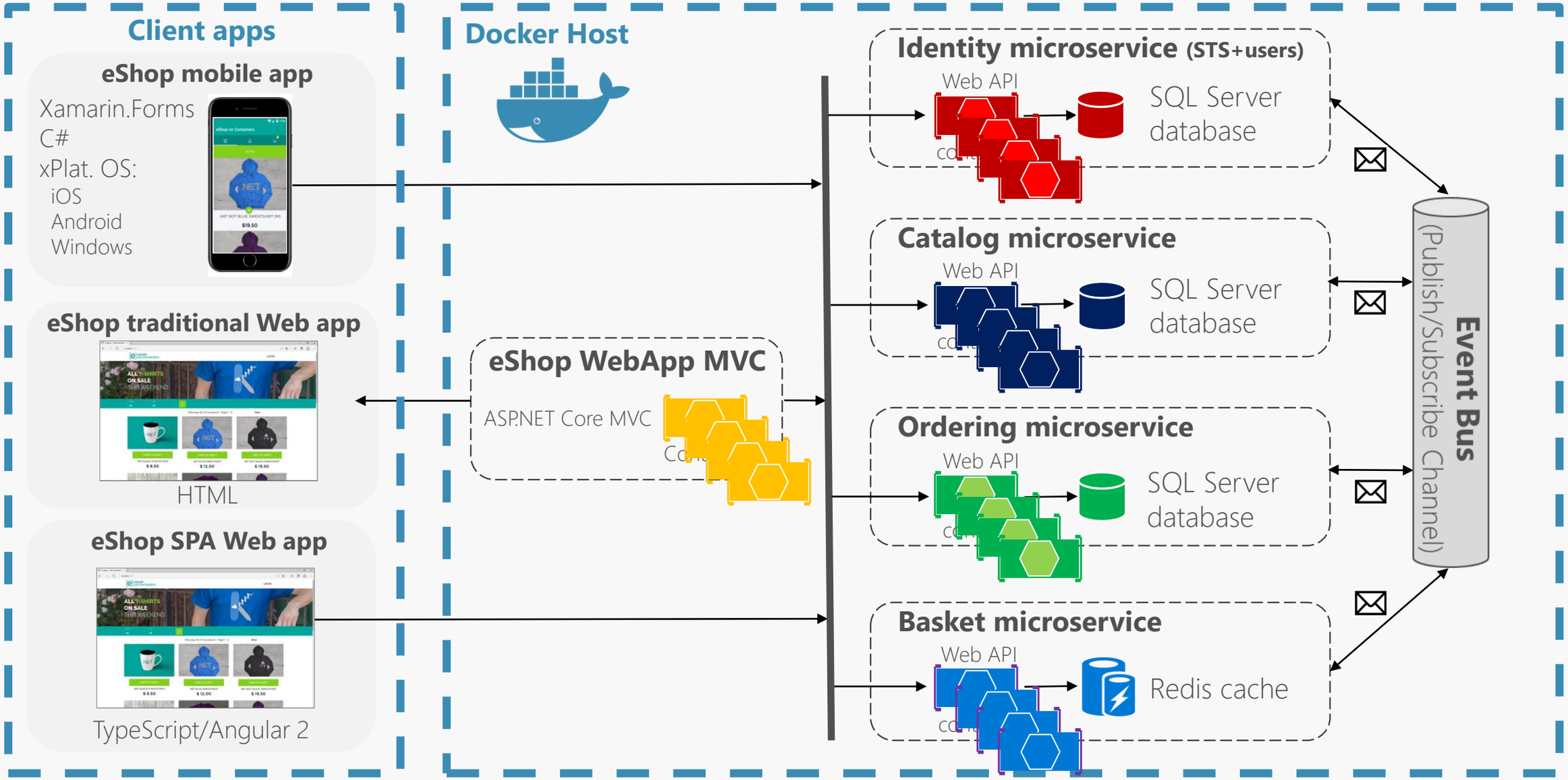
eShopOnContainers Microservices and Docker Containers
End-to-end solution

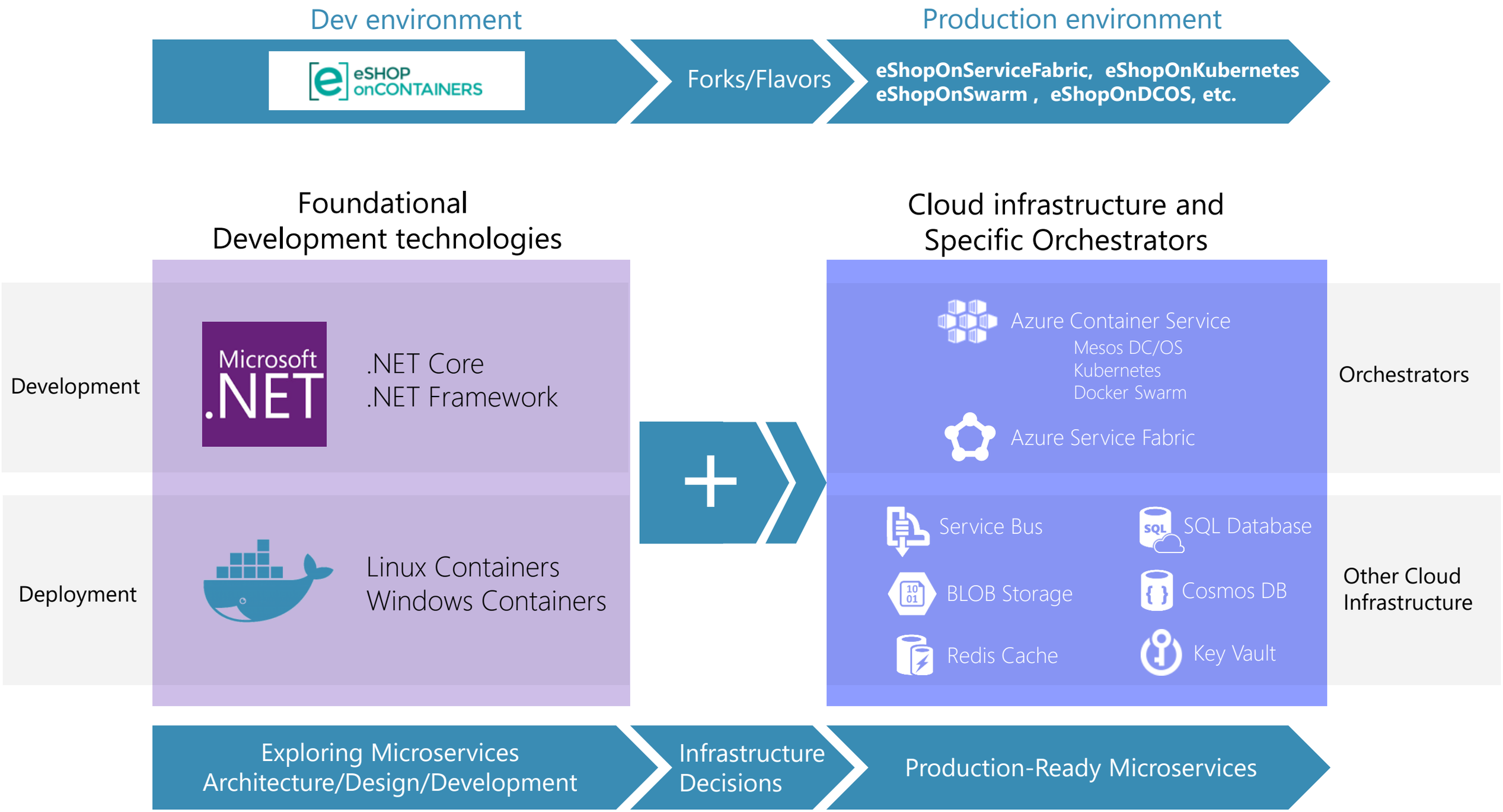


eShopOnContainers Reference Application - Architecture



Scaling out eShopOncontainers




Dev environment
eSHOP onCONTAINERS
Forks/Flavors
Production environment
eShopOnServiceFabric, eShopOnKubernetes
eShopOnSwarm , eShopOnDCOS, etc.
Foundational
Development technologies
Development
Microsoft .NET
.NET Core
.NET Framework
Deployment
Linux Containers
Windows Containers
Cloud infrastructure and
Specific Orchestrators
Azure Container Service
Mesos DC/OS
Kubernetes
Docker Swarm
Azure Service Fabric
Orchestrators
Service Bus
SQL Database
BLOB Storage
Cosmos DB
Redis Cache
Key Vault
Other Cloud
Infrastructure
Exploring Microservices
Architecture/Design/Development
Infrastructure
Decisions
Production-Ready Microservices

Domain-Driven Design (DDD) & Microservices: Patterns and Practices

THANK YOU

Nevin Dong 董乃文
Principle Technical Evangelist
Microsoft Cooperation

Microsoft