Introduction to Microservices

Agenda

- Evolution of Computing
- Introduction to Microservices
- Microservices Principles
- Advantages of Microservices
- Challenges of Microservices
- N-Layer App to Microservices
- Microservices Deployment Options
- Microservices Architecture
- Microservices Patterns

ScholarHat

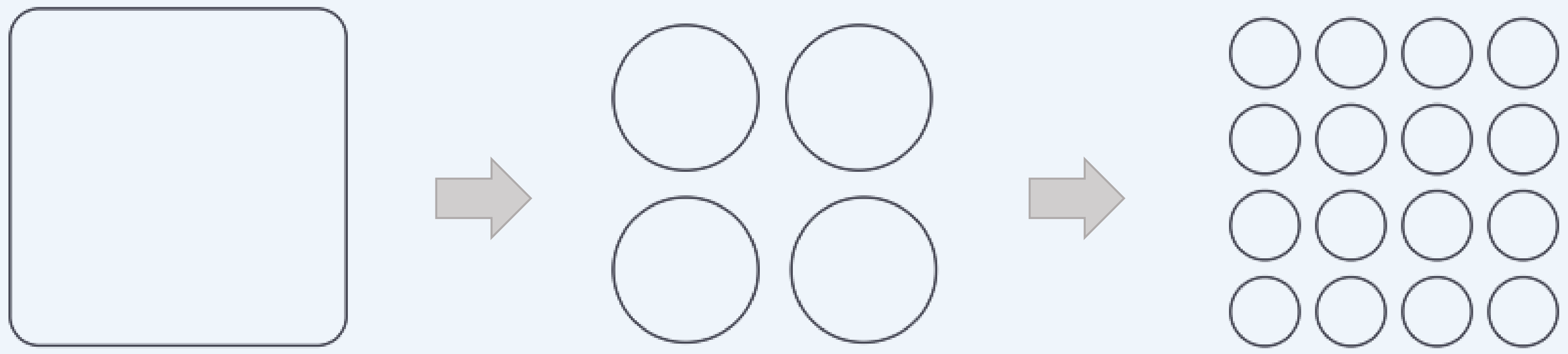
Evolution of Computing

Development Process	Application Architecture	Deployment and Packaging	Application Infrastructure
Waterfall	Monolithic	Physical Server	Datacenter
Agile	N-Tier	Virtual Servers	Hosted
DevOps	Microservices	Containers	Cloud

Introduction to Microservices

- A Microservice is a small unit that has only one responsibility or single logic which solve a specific problem.
- Microservices are small & independent services that work together to build highly automated, independent and evolving software.
- Each Microservice can be deployed independently.
- All services don't need to share the same technology stack, libraries, or frameworks.

Monolithic vs. N-Layer vs. SOA vs. Microservices

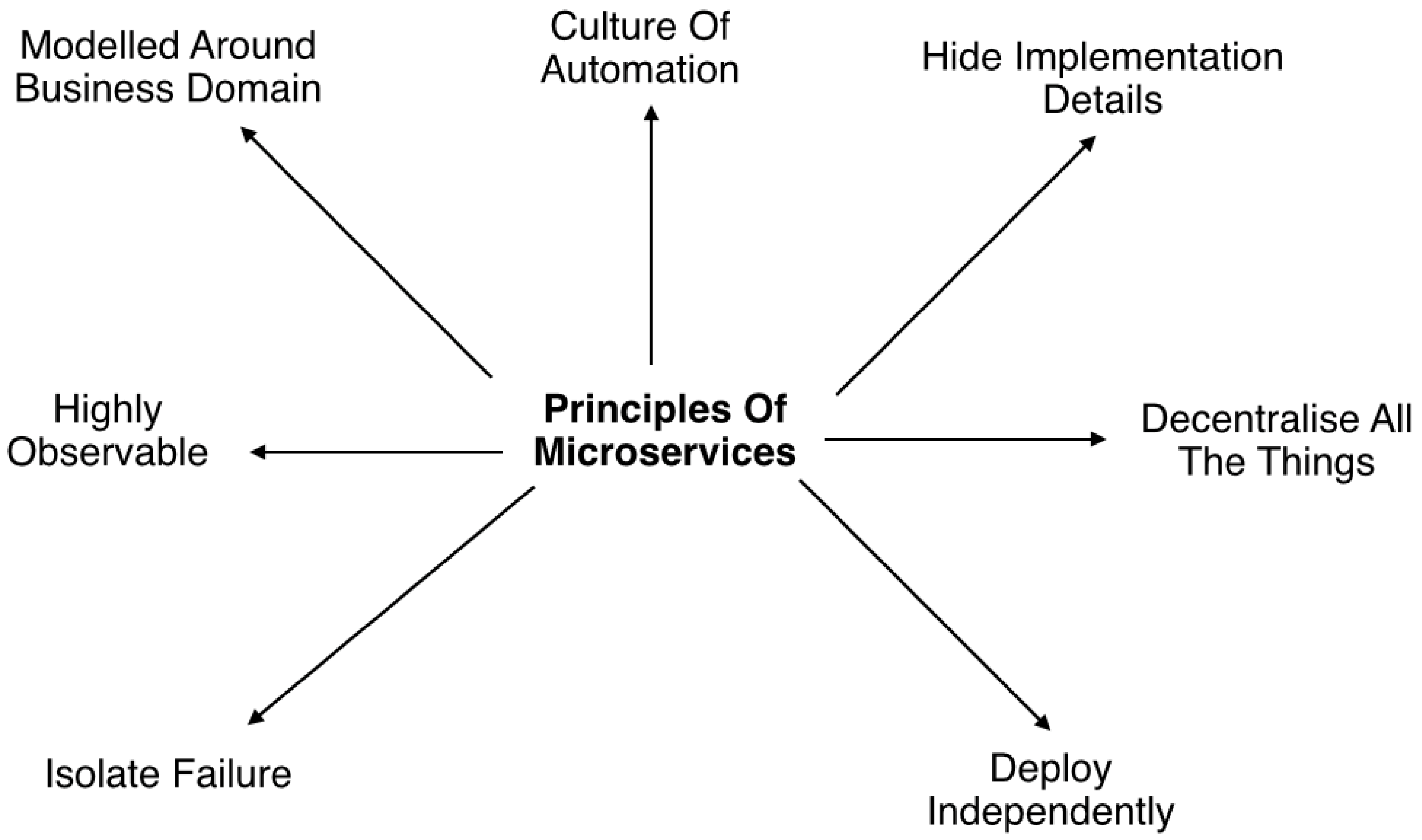


Monolithic: Single Unit

Multi Units: N-Layer/SOA

Smaller Units: Microservices

Microservices Principles


Principles Of Microservices
Modelled Around Business Domain
Culture Of Automation
Hide Implementation Details
Decentralise All The Things
Deploy Independently
Isolate Failure
Highly Observable

ScholarHat

Microservices Principles Contd..

- **Modelled around business domain :** Separate system capability into different domains and each domain will focus on one thing and its associated logic.
- **Culture of Automation :** Follow the culture of automation by designing it for continues integration and continuous delivery.
- **Hide implementation details :** Hiding the internal details reduce the coupling and helps to do changes and improvement without affecting the overall architecture.
- **Decentralization :** There is no centralized database, usually each service is designed to manage its own database.

Microservices Principles Contd..

- **Deploy Independently :** Each service can be deployed independently.
- **Failure Isolation :** The impact of a failure is less in microservice architecture compares to monolithic type as it will only affect that particular service and its associates. Other services can keep running.
- **Highly Observable :** The services should collect as much information to analyze what is happening within each of them like log events and stats.

When to use Microservices Architecture

- Large applications that require a high release velocity.
- Complex applications that need to be highly scalable.
- Applications with rich domains or many subdomains.
- An organization that consists of small development teams.

ScholarHat

Advantages of Microservices

- Language independent and framework independent.
- Support Independent development, deployment, and versioning.
- Even you can scale them in seconds without compromising the integrity of an application.
- Better fault isolation keeps other services to work even on failure.
- Zero downtime upgrades.
- Services can be from different servers or different datacenters.
- Supports CI/CD.

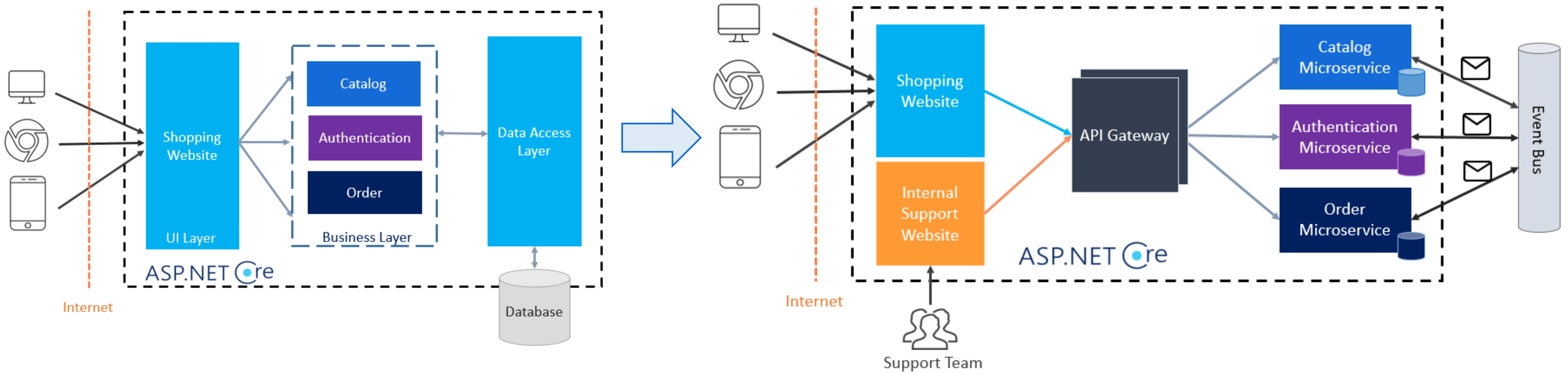
Challenges of Microservices

- **Complexity -** A microservices application is more complex as compared to the equivalent monolithic application.
- **Development and testing -** Building and testing a service that relies on other services need domains understanding and refactoring them can be difficult.
- **Lack of governance** - The decentralized approach to building microservices has advantages, but it also lead to so many problems like maintenance because of many different languages and frameworks.
- **Network congestion and latency** - The use of many small, services can result into additional latency because of interservice communication and a long chain of service dependencies. So design APIs carefully and use asynchronous communication patterns.

Challenges of Microservices Contd..

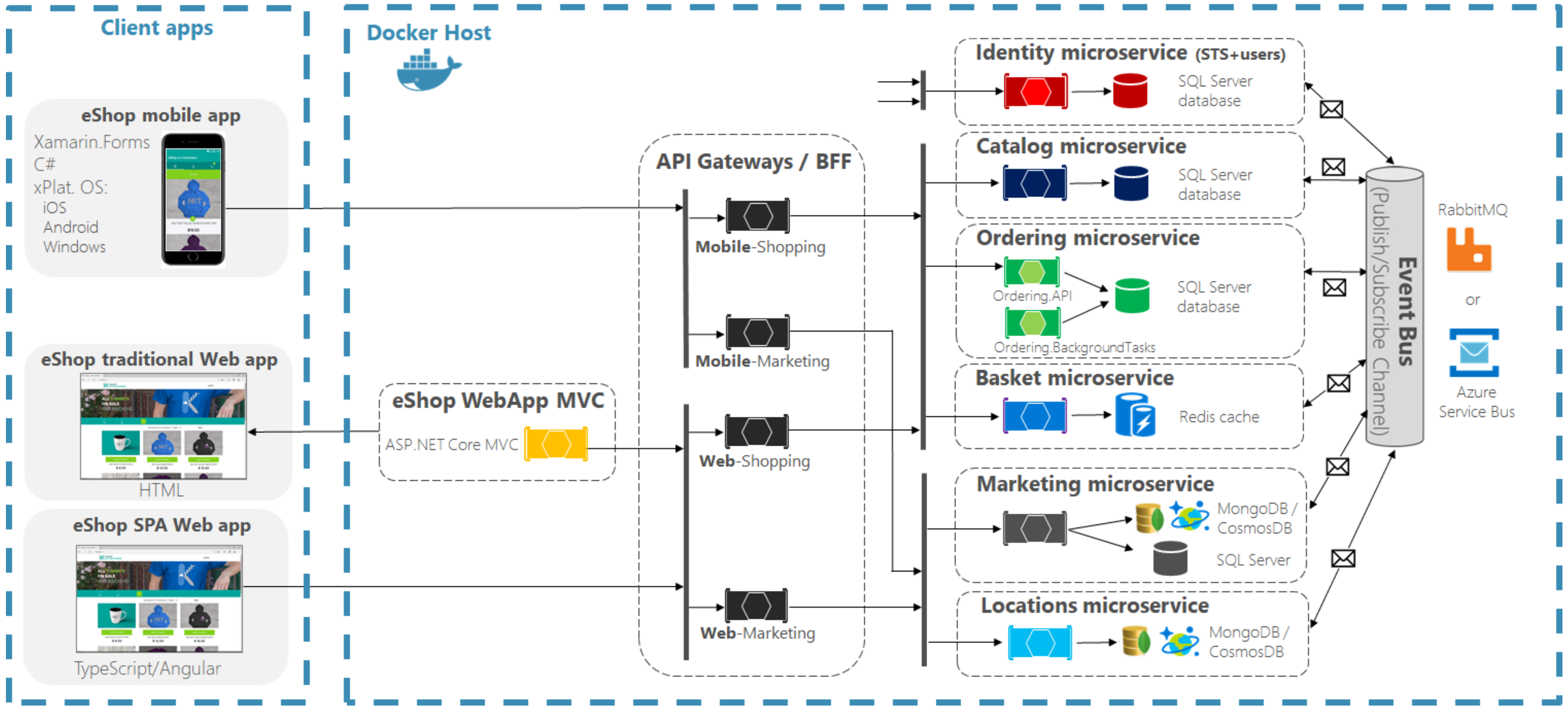
- **Data Integrity** - Each microservice is responsible for its own data persistence. As a result, data consistency can be a challenge.
- **Management** - To be successful with microservices requires a mature DevOps culture. Correlated logging across services can be challenging for a single user operation.
- **Versioning** - Be careful while updating a service. It must not break services that depend on it. So without careful design, you might have problems with backward or forward compatibility.
- **Skillset** - Microservices are highly distributed systems. So need a skilled and experience team to implement it.

N-Layer App to Microservices



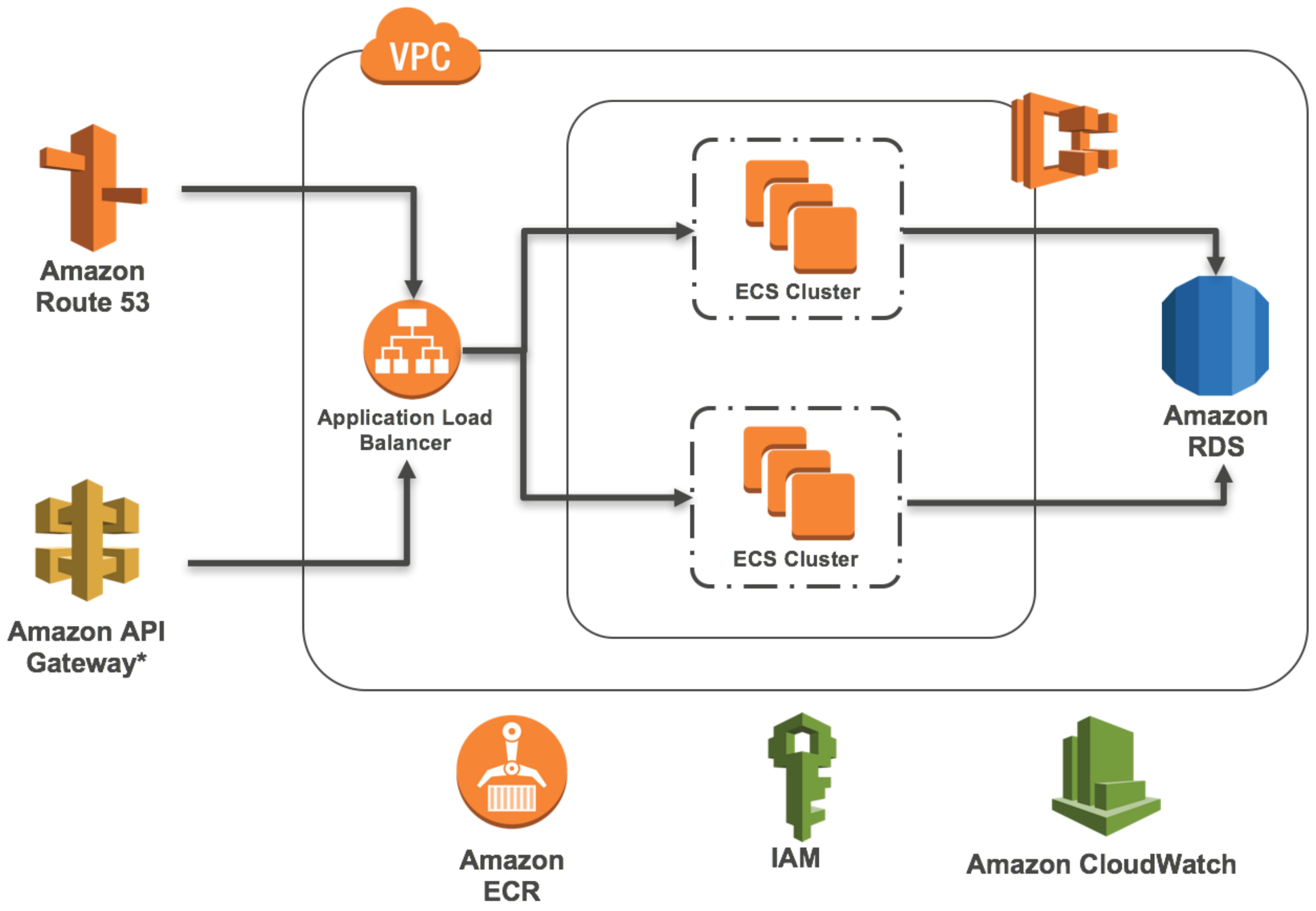
eShopOnContainers reference application

(Development environment architecture)

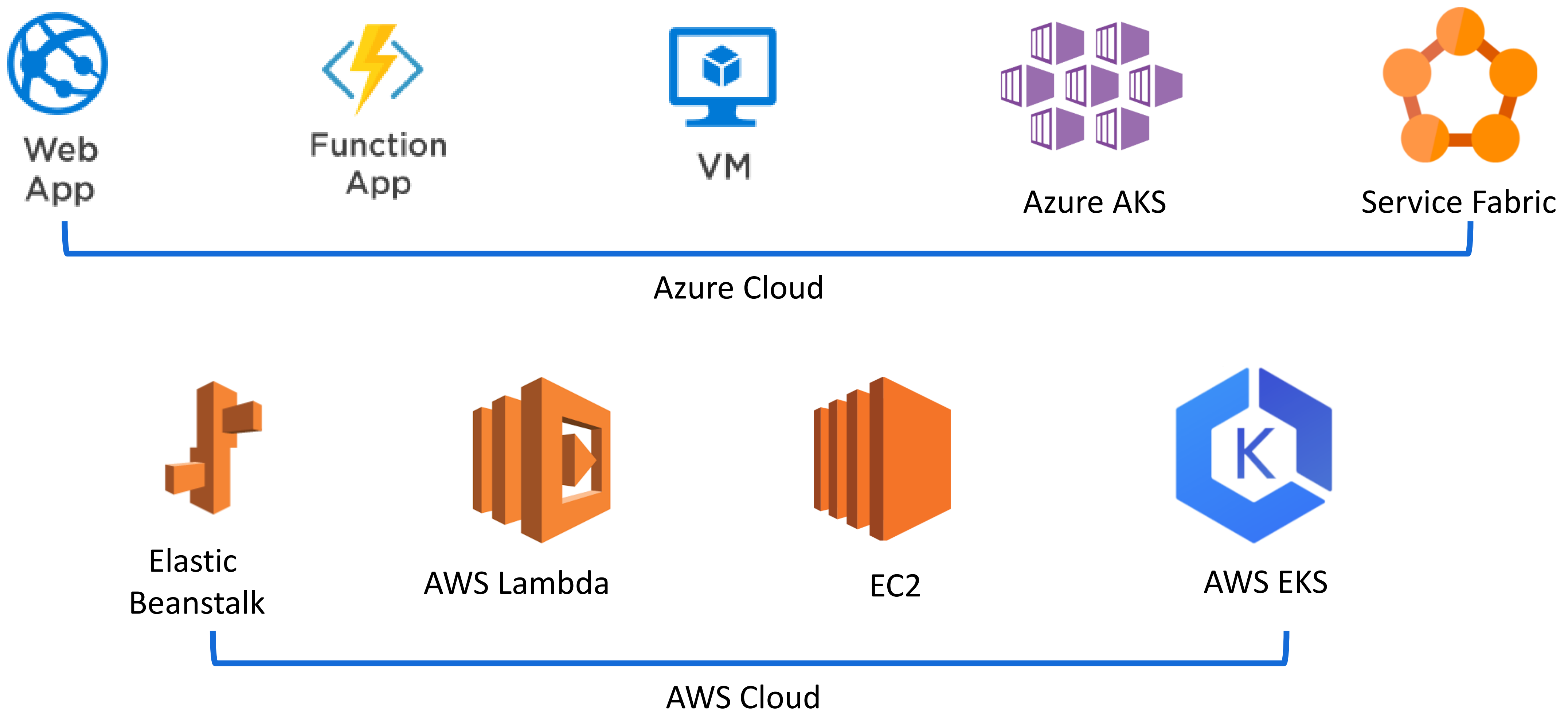


Reference: https://github.com/dotnet-architecture/eShopOnContainers/tree/master

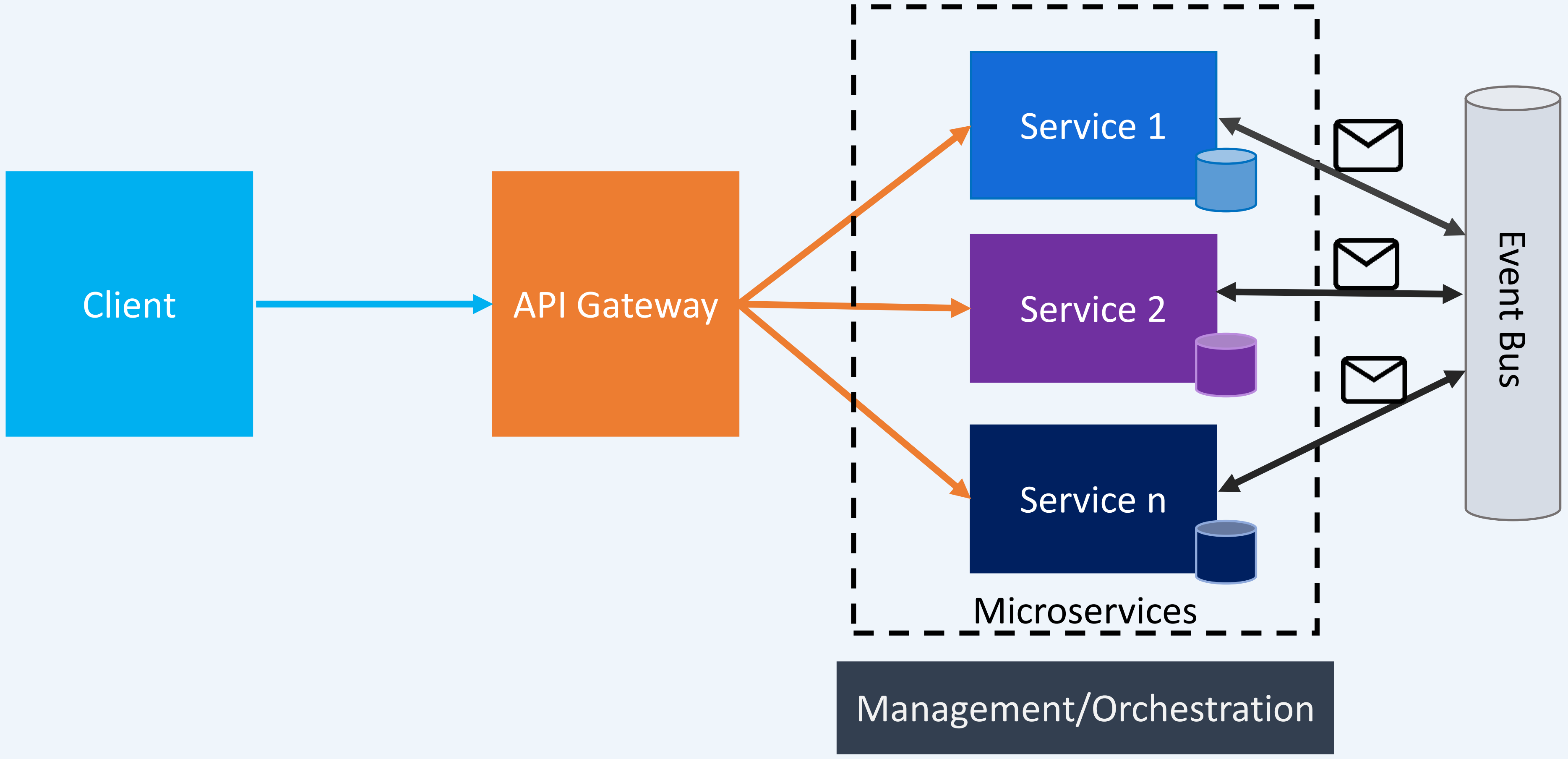
Microservices Architecture on AWS



Microservices Deployment Options



Microservices Architecture



Microservices Orchestration

Docker (Docker Swarm)

Kubernetes

Unmanaged Cluster

Azure AKS

AWS EKS

Managed Cluster

Greenfield vs. Brownfield

- **Greenfield Development** happens when you start a brand new project with clean slate development. No legacy code or no old development to maintain. Starting project from scratch with no restrictions on what you're doing (other than business rules).
- **Brownfield Development** happens when business decide to develop/improve upon an existing application infrastructure. As an upgrade is implemented into an existing solution.

Microservices Patterns

Decomposition Patterns

- Decompose by Business Capability
- Decompose by Subdomain
- Strangler Pattern

Integration Patterns

- API Gateway Pattern
- Aggregator Pattern
- Client-Side UI Composition Pattern

Database Patterns

- Database per Service
- Shared Database
- CQRS Pattern
- Saga Pattern

Communication Patterns

- Request/Response Pattern
- Messaging Pattern
- Event Driven Pattern

Cross-Cutting Concern Patterns

- Externalized Configuration
- Service Discovery Pattern
- Circuit Breaker Pattern

Observability Patterns

- Log Aggregation
- Performance Metrics
- Distributed Tracing
- Health Check

Deployment Patterns

- Multiple Service Instances per Host
- Service Instance per Host
- Serverless Deployment
- Service Deployment Platform