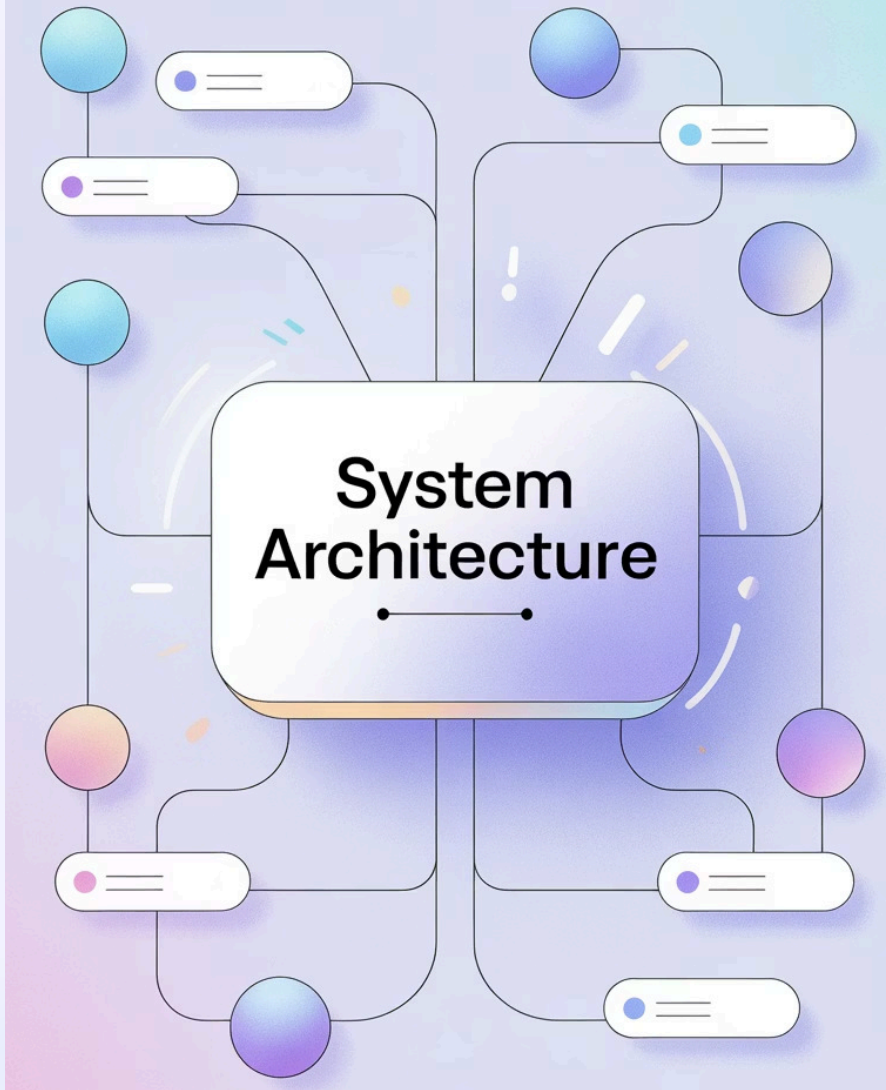
Microservices Basics: An Introduction

Microservices architecture has revolutionized how we build software. It's becoming essential knowledge for developers.

This presentation explores the fundamentals, benefits, and challenges of microservices.

S **by Saratha Natarajan**



What Are Microservices?

Small & Focused

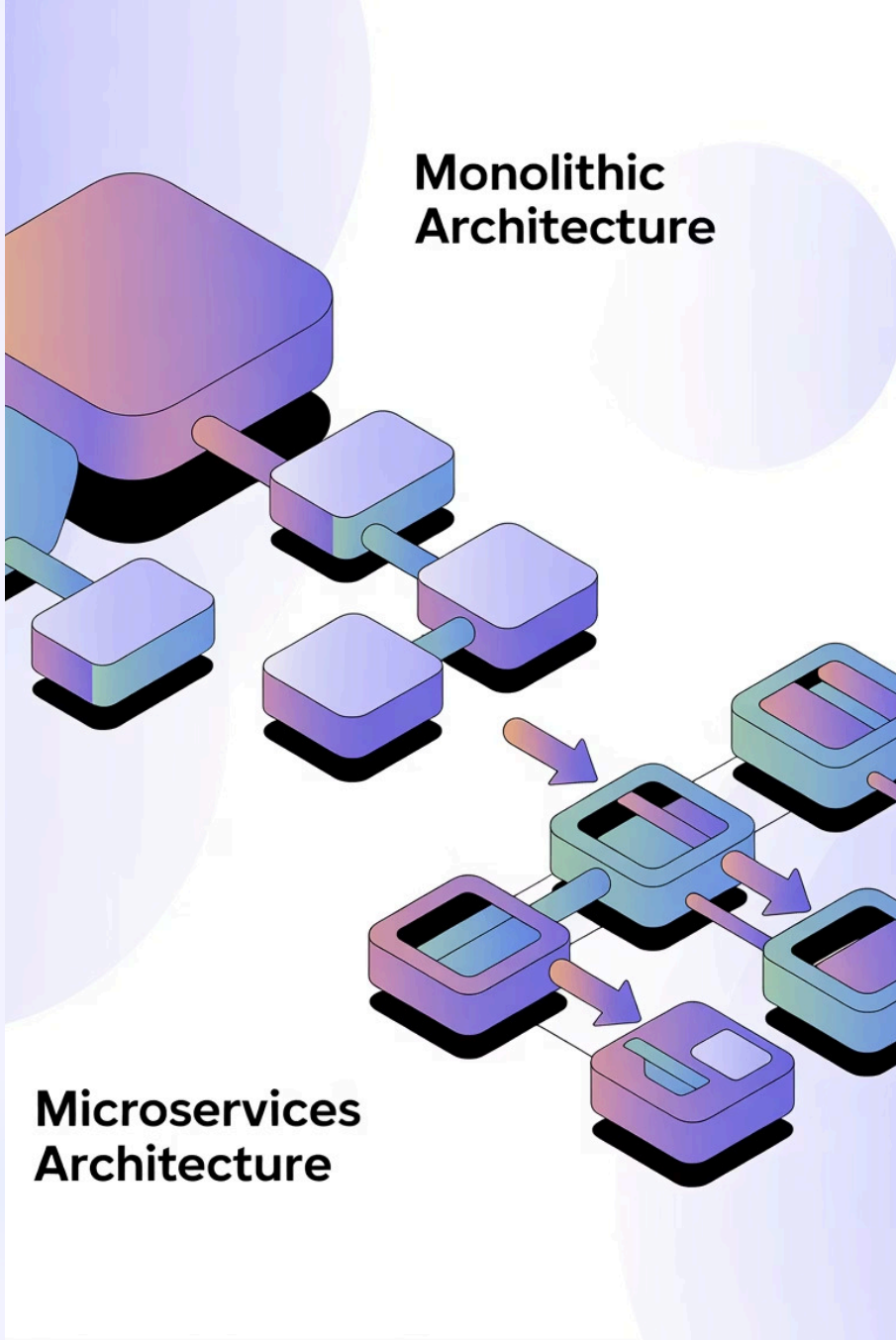
Services built around business capabilities. Each handles one specific function.

Independently Deployable

Can be updated without affecting the whole system. Teams deploy on their own schedule.

Loosely Coupled

Services communicate through well-defined APIs. Changes to one don't impact others.



History and Evolution

1

2005

Dr. Peter Rodgers coins "micro-web-services" term at Cloud Computing Conference.

2

2009-2012

Netflix begins migration from monolith to microservices architecture.

3

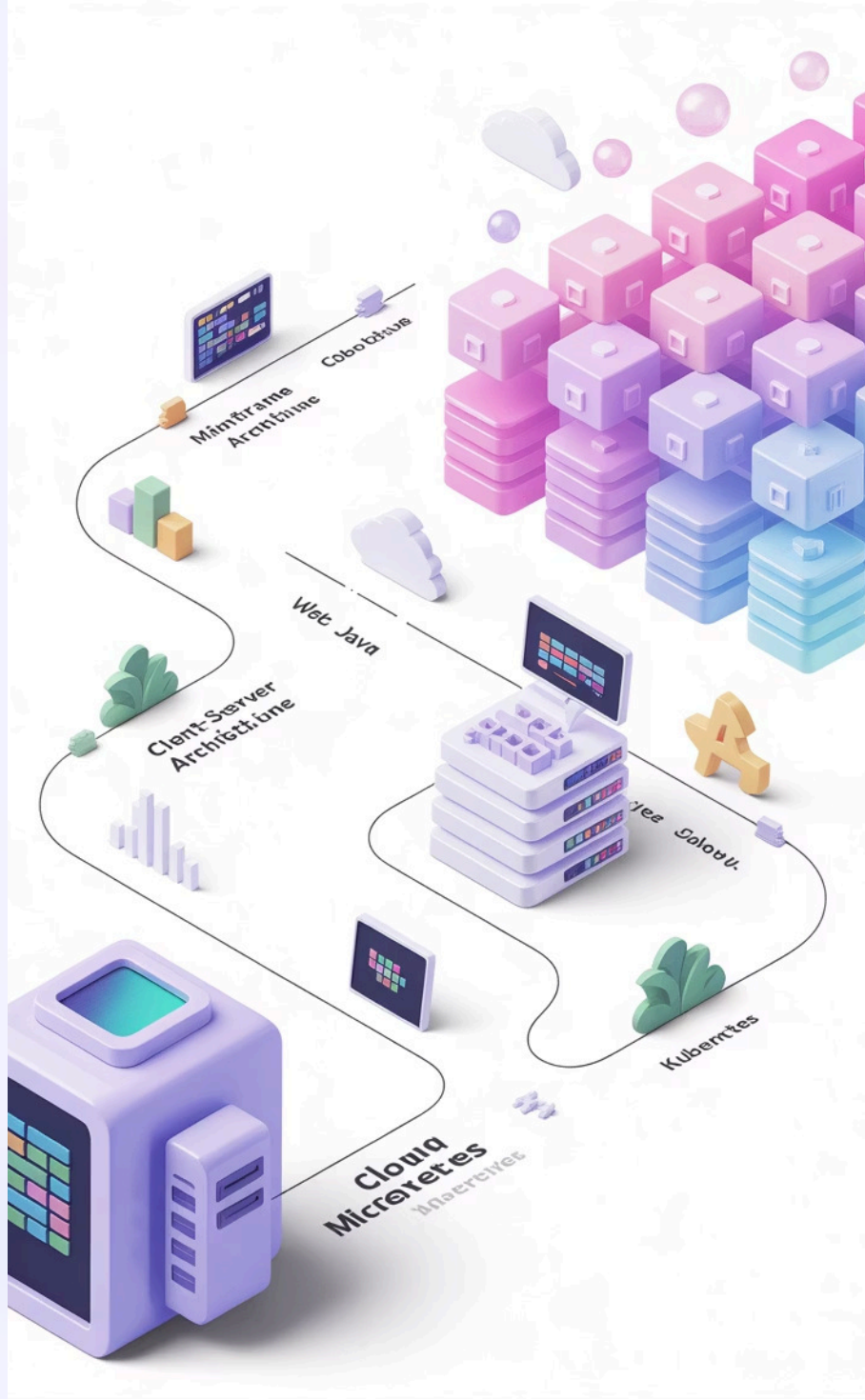
2014

James Lewis and Martin Fowler publish influential microservices article.

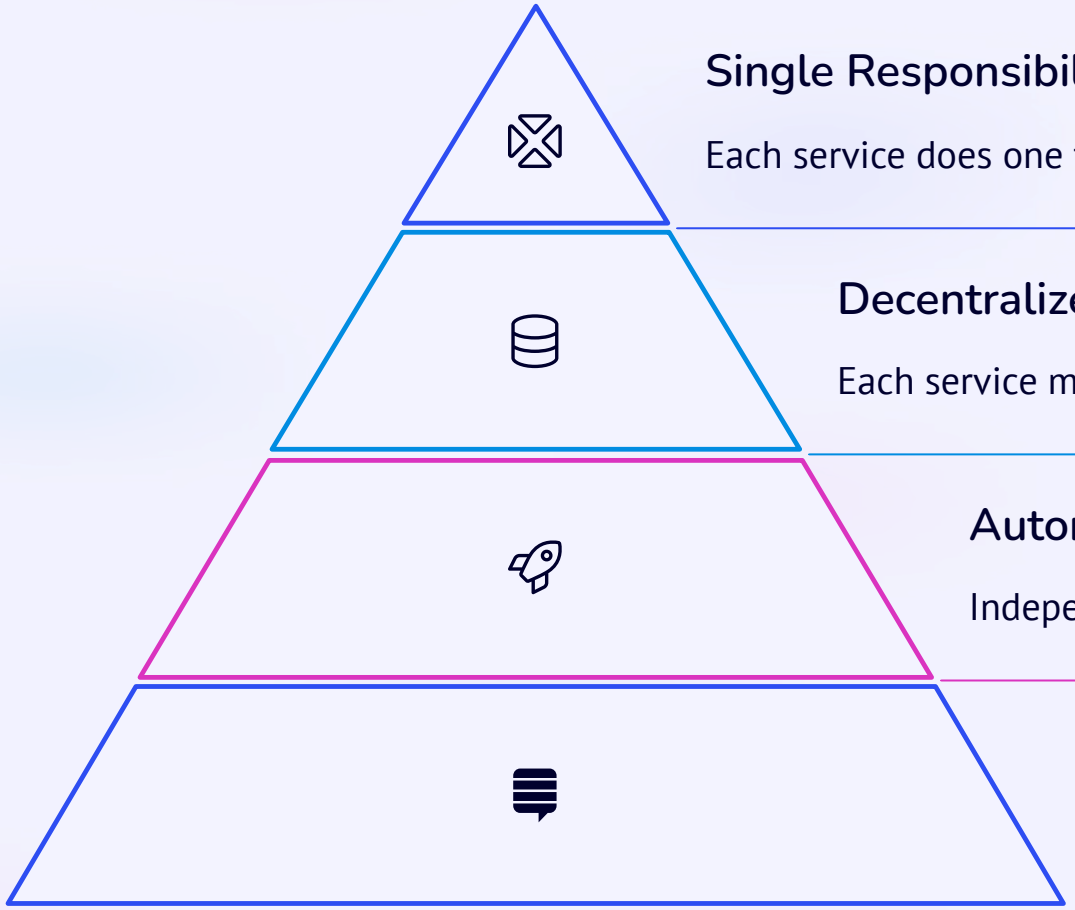
4

Post-2015

Widespread industry adoption across enterprise and startup sectors.



Core Principles of Microservices



Single Responsibility

Each service does one thing well

Decentralized Data

Each service manages its own data

Autonomous Deployment

Independent release cycles

Smart Endpoints, Dumb Pipes

Simple communication protocols

Key Benefits

Enhanced Scalability

Scale individual components based on demand. Focus resources where needed.

Improved Resilience

Failures are isolated to specific services. The system remains partially functional.

Faster Development

Smaller codebases enable quicker feature delivery. Teams work in parallel.

Technology Flexibility

Use the right tool for each job. Different languages per service.

Primary Challenges

Distributed System Complexity

Network issues, latency problems, and debugging challenges emerge.

Operational Overhead

More services mean more infrastructure. Monitoring and deployment complexity increases.

3

Data Consistency Issues

Transactions spanning multiple services are difficult. Data integrity becomes challenging.

Team Organization

Requires skilled DevOps teams. Communication patterns must evolve.

Microservices vs. Monoliths

Monolithic Architecture

- Single codebase
- One technology stack
- All-or-nothing deployment
- Simpler development
- Horizontal scaling only
- Shared database

Microservices Architecture

- Multiple codebases
- Technology diversity
- Independent deployments
- Complex orchestration
- Granular scaling
- Decentralized data

Architecture Overview

API Gateway

Single entry point for clients. Handles routing, load balancing, and authentication.

Service Discovery

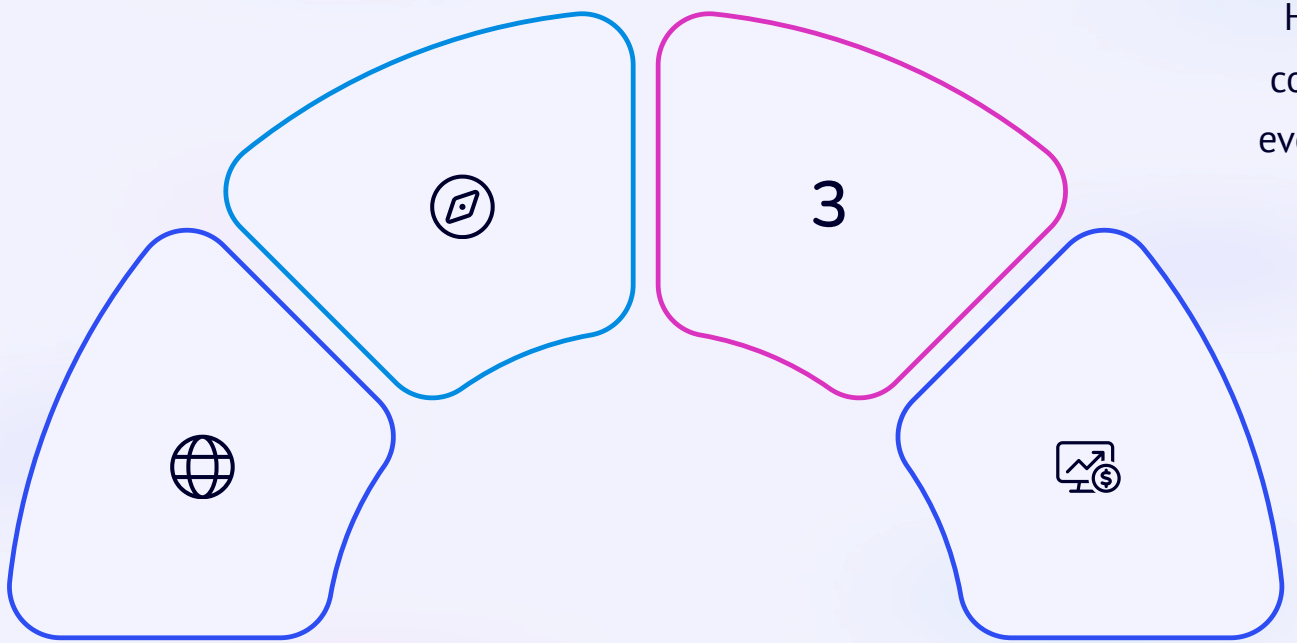
Allows services to find each other. Tracks service locations dynamically.

Data Services

Each microservice manages its own data store. Ensures independence.

Message Broker

Handles asynchronous communication. Enables event-driven architecture.



Inter-Service Communication Patterns

Synchronous (Request/Response)

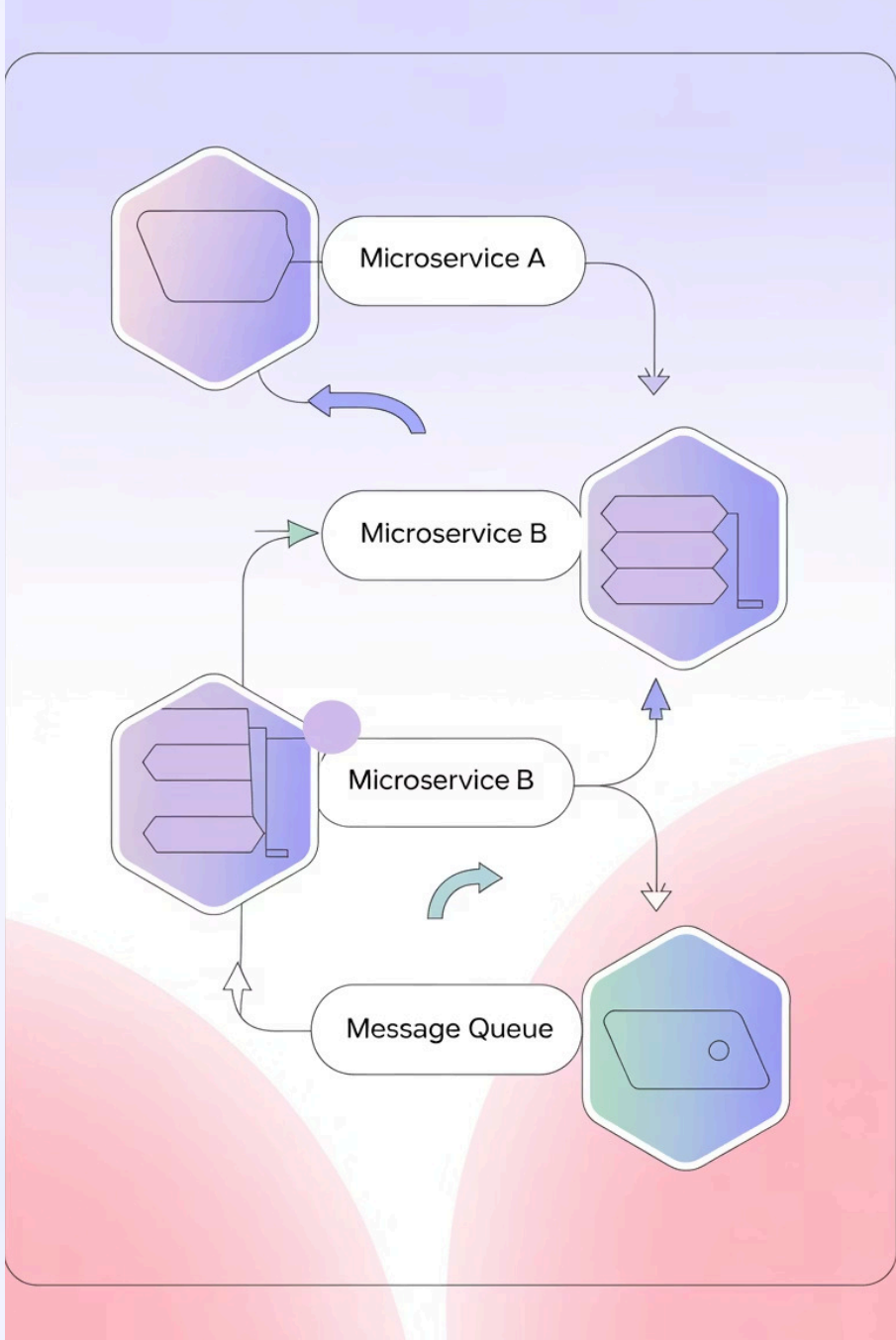
- REST APIs
- gRPC
- GraphQL
- Direct HTTP calls

Asynchronous (Event-Based)

- Apache Kafka
- RabbitMQ
- Amazon SQS
- Event-driven architecture

Service Mesh

- Istio
- Linkerd
- Consul Connect
- Traffic management



Data Management in Microservices

1

Database Per Service

Each service maintains its own data store

Saga Pattern

Manages distributed transactions across services

Event Sourcing

Stores state changes as event sequences

CQRS

Separates read and write operations

Deployment Strategies

Containerization

Package applications with dependencies using Docker. Ensure consistency across environments.

- Isolates application dependencies
- Provides consistent environments

Orchestration

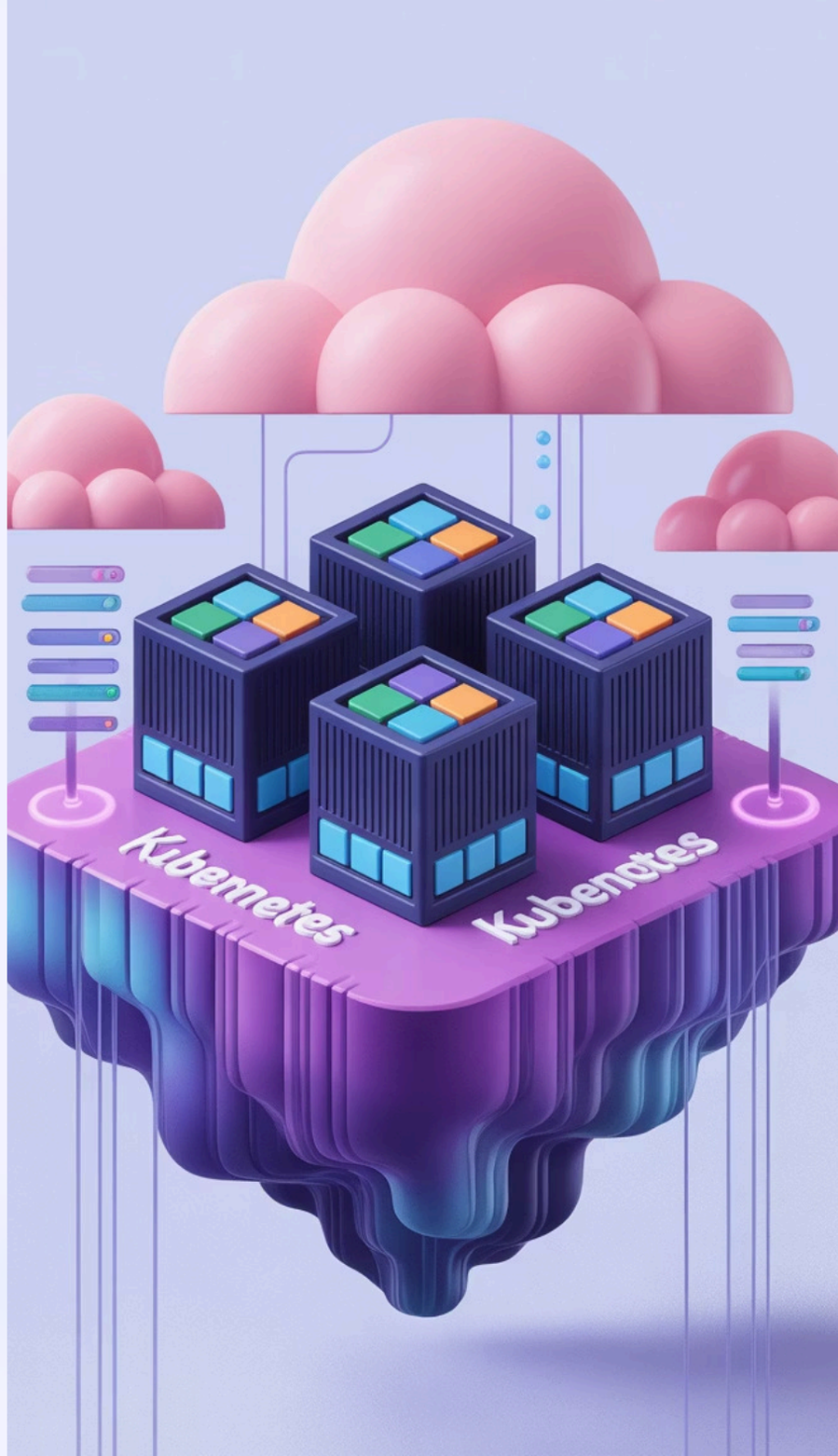
Manage containers with Kubernetes. Automate deployment, scaling, and operations.

- Handles container scheduling
- Manages service discovery

Continuous Delivery

Implement robust CI/CD pipelines. Enable frequent, reliable releases.

- Automates testing and deployment
- Enables canary deployments



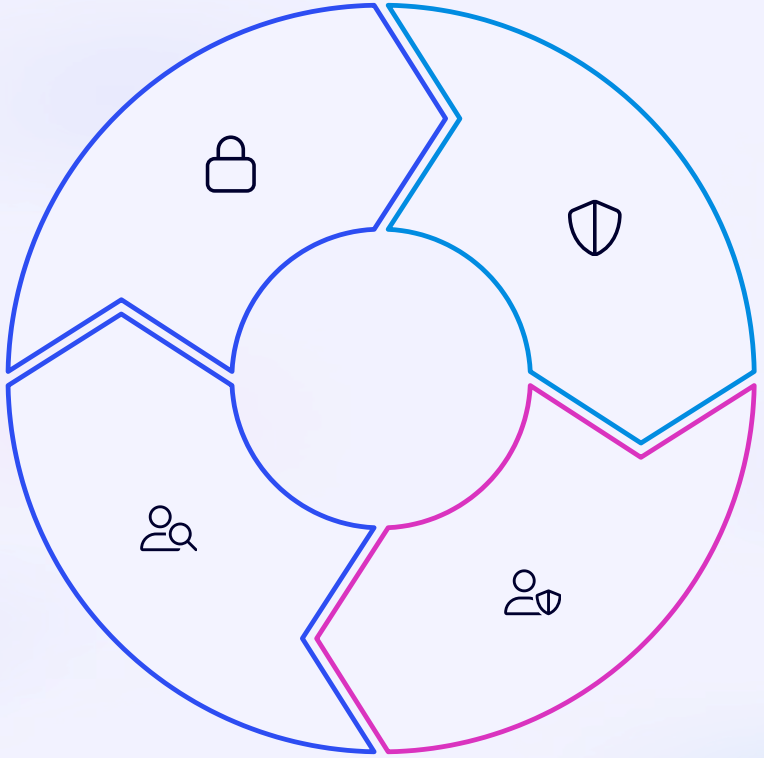
Security Considerations

Authentication & Authorization

Secure service-to-service communication with mTLS or OAuth tokens

Security Monitoring

Detect anomalies with continuous scanning and logging analysis



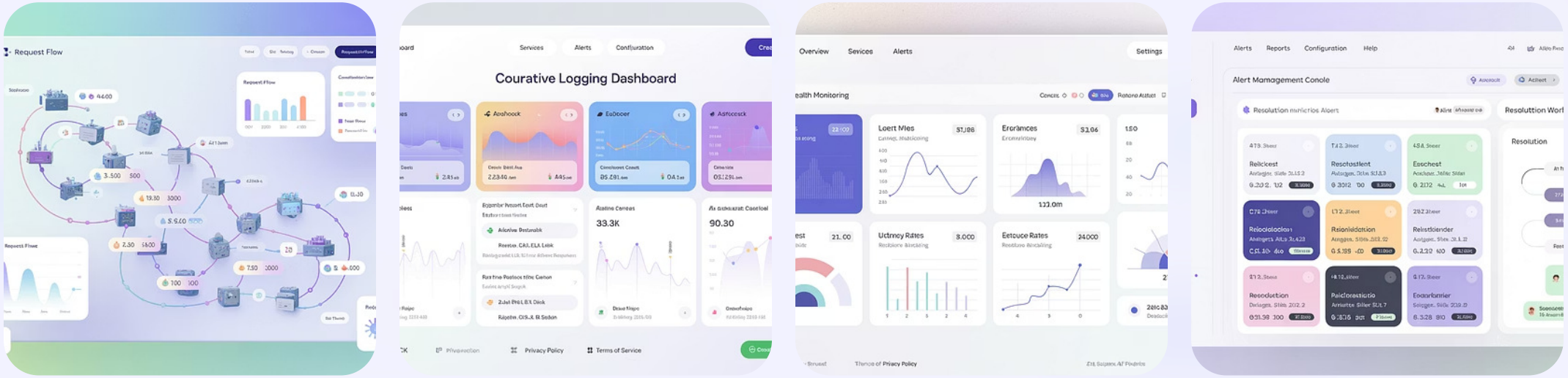
API Gateway Security

Implement rate limiting, input validation, and threat protection

Secrets Management

Securely store and distribute credentials using vault solutions

Monitoring and Observability



Comprehensive observability requires distributed tracing, centralized logging, metrics monitoring, and alert management systems.

Tools like Prometheus, Grafana, Jaeger, and the ELK stack form the observability backbone.

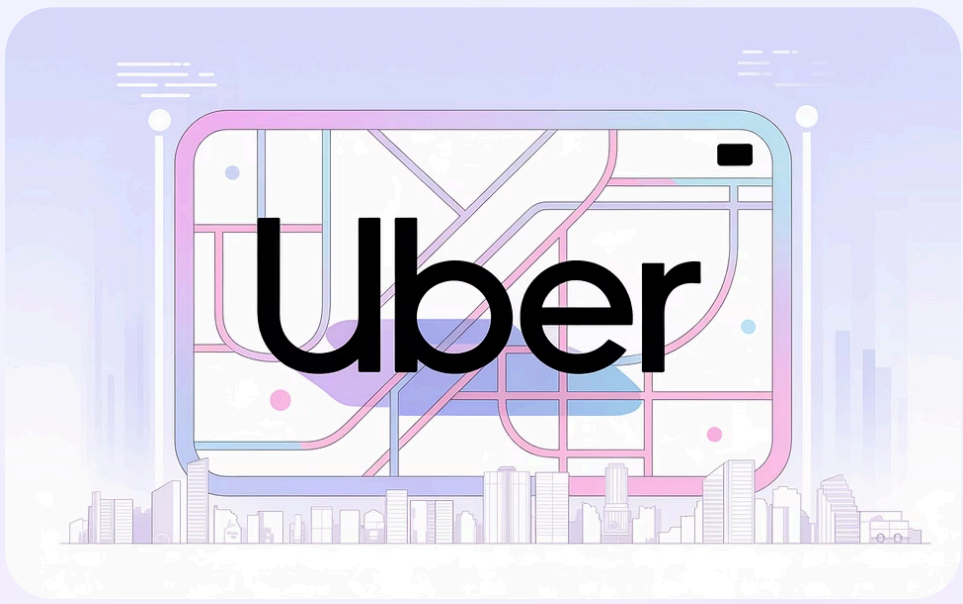
Real-World Success Stories

Netflix

Migration to microservices enabled 1B+ streaming hours monthly. Improved resilience and deployment speed.

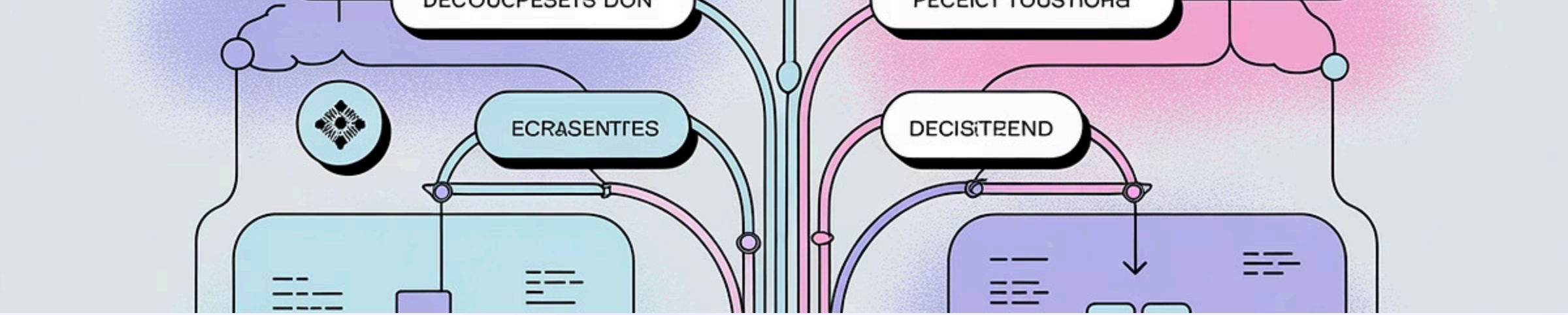
Amazon

Two-pizza team approach with service-oriented architecture. Fueled explosive growth and innovation.



Uber

Powers real-time matching of riders and drivers. Handles millions of concurrent requests globally.



When to Use (or Avoid) Microservices

Good Fit	Poor Fit
Large, complex applications	Simple applications with few domains
Organizations with multiple teams	Small teams with limited resources
Systems requiring different scaling needs	Applications with tight data consistency requirements
Applications needing technology diversity	Projects with tight deadlines and no DevOps expertise

Conclusion and Key Takeaways

1

Start Small

Begin with a monolith. Extract services gradually as boundaries become clear.

2

Invest in Automation

CI/CD pipelines and monitoring are essential. Not optional.

3

Domain Boundaries

Service boundaries based on business domains work best.

4

Team Structure

Organize teams around services. Conway's Law is real.