



A Pragmatic Introduction to Microservices



Key Takeaway

Understand key differences between monolith and microservices, why and how to implement integration pattern with serverless, and benefits of using microservices

Agenda for Today

1. Development transformation at Amazon
2. Monolith and Microservices
3. Application Integration Patterns
4. Benefits of using microservices

A Journey.

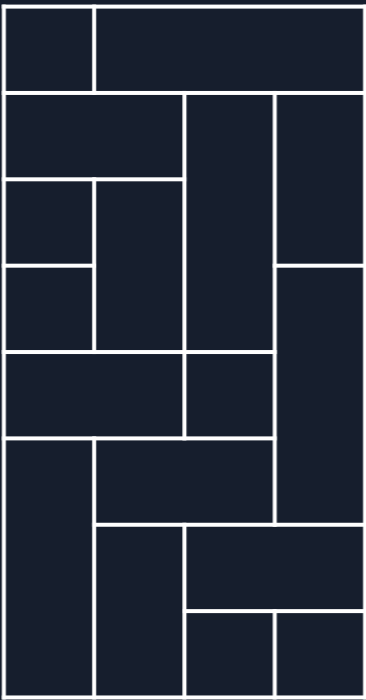
Development transformation at Amazon: 2001–2002

Lesson learned: decompose for agility

2001



2002

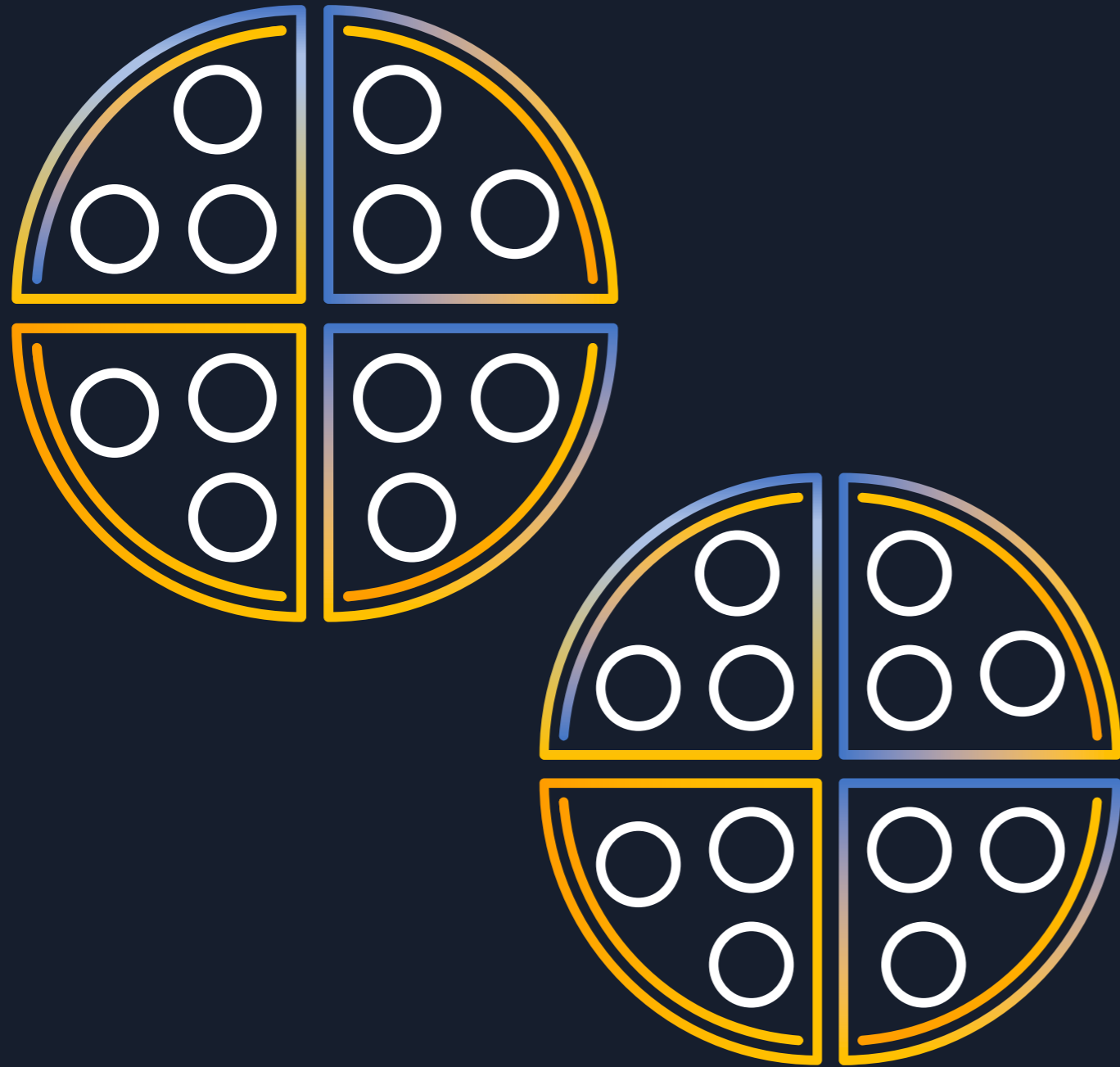


monolithic application
+ teams



microservices
+ two-pizza teams

Two-pizza teams are fast & agile



Full ownership & autonomy

You build it, you run it

DevOps – small, nimble teams

Focused innovation

2002 - API Mandate



1. *All teams will henceforth expose their data and functionality through service interfaces.*
2. *Teams must communicate with each other through these interfaces.*
3. *There will be no other form of inter-process communication allowed.*
4. *It doesn't matter what technology they use.*
5. *All service interfaces, without exception, must be designed from the ground up to be externalizable.*

Impact

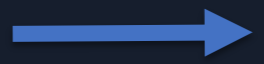
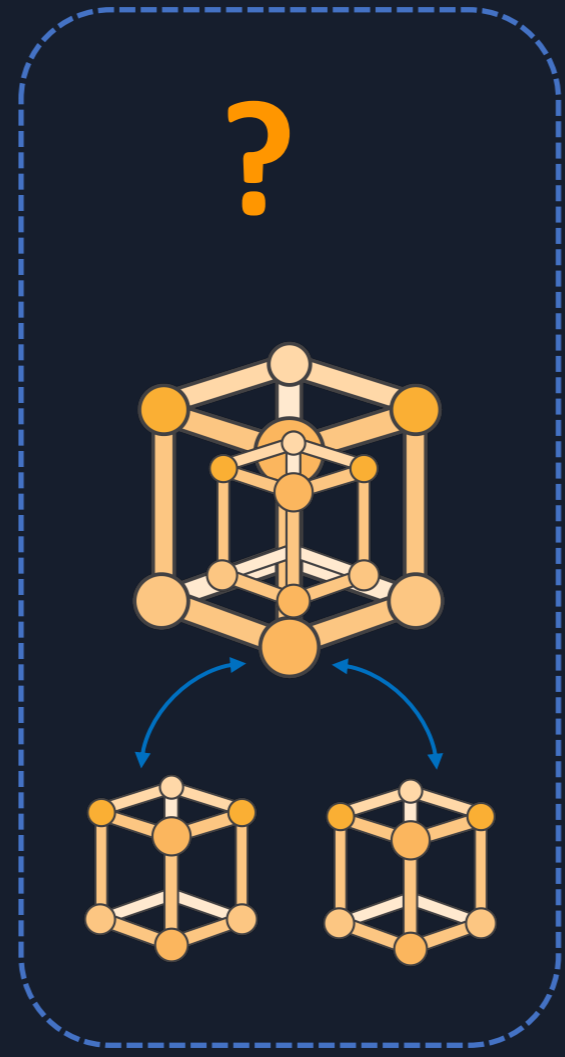
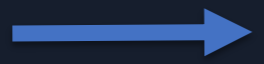
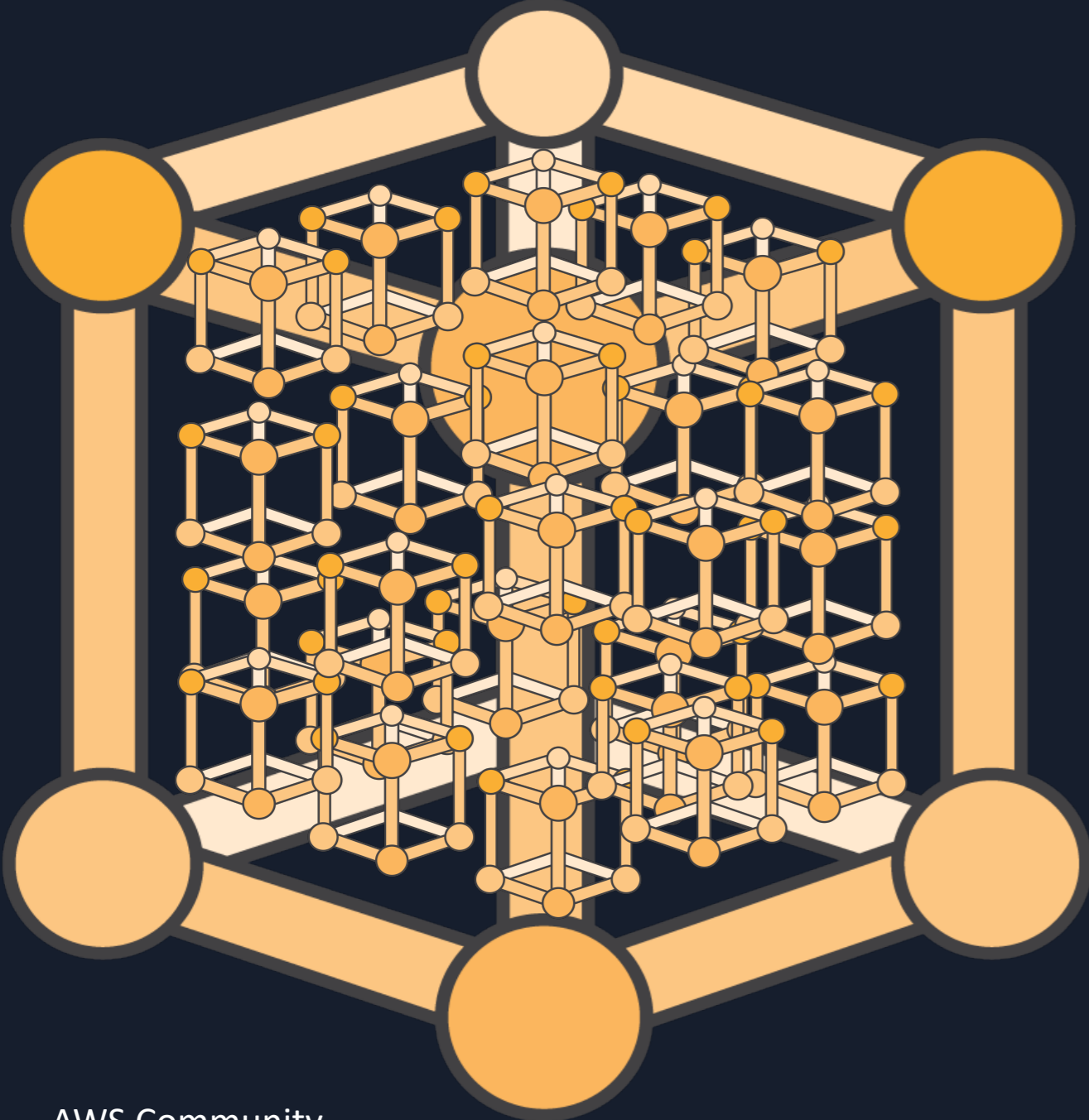


>60 million deployments a year

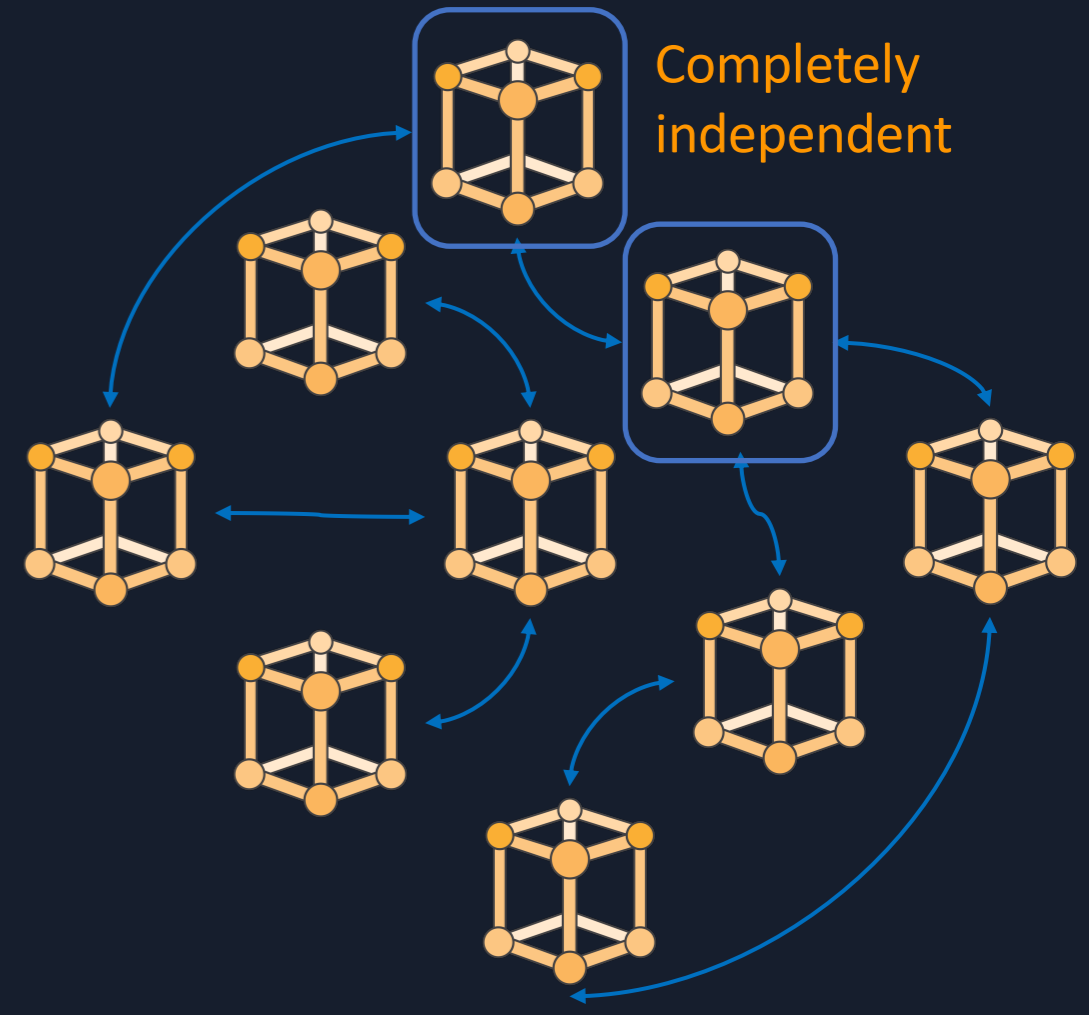
Microservices.

Basic Concepts and Definitions

Monolith



Microservices

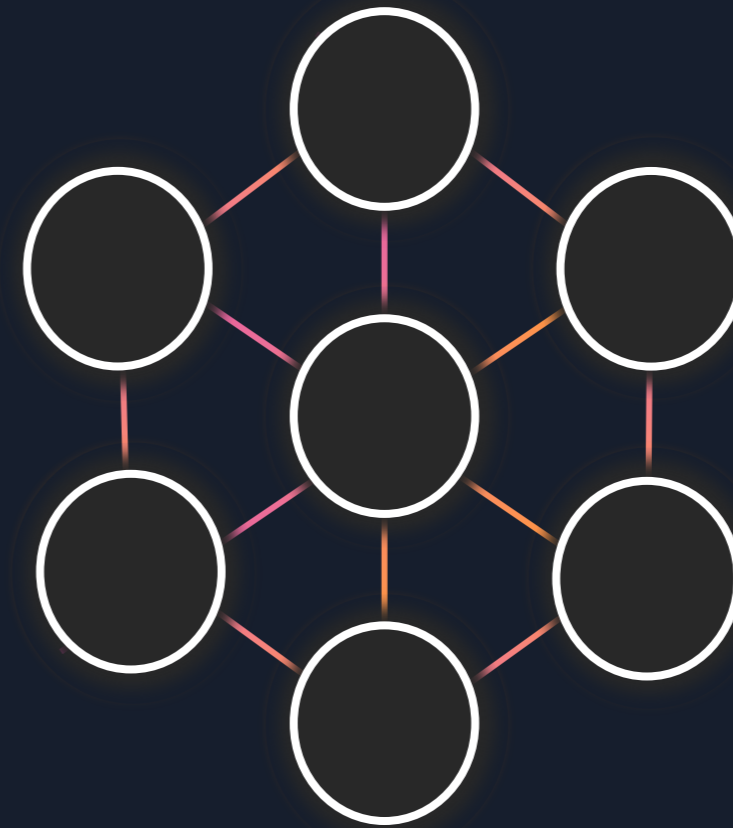


When the impact of change is small, release velocity can increase



Monolith

Does everything



Microservices

Does one thing

Traditional three-tier application architecture



Web servers
Presentation layers

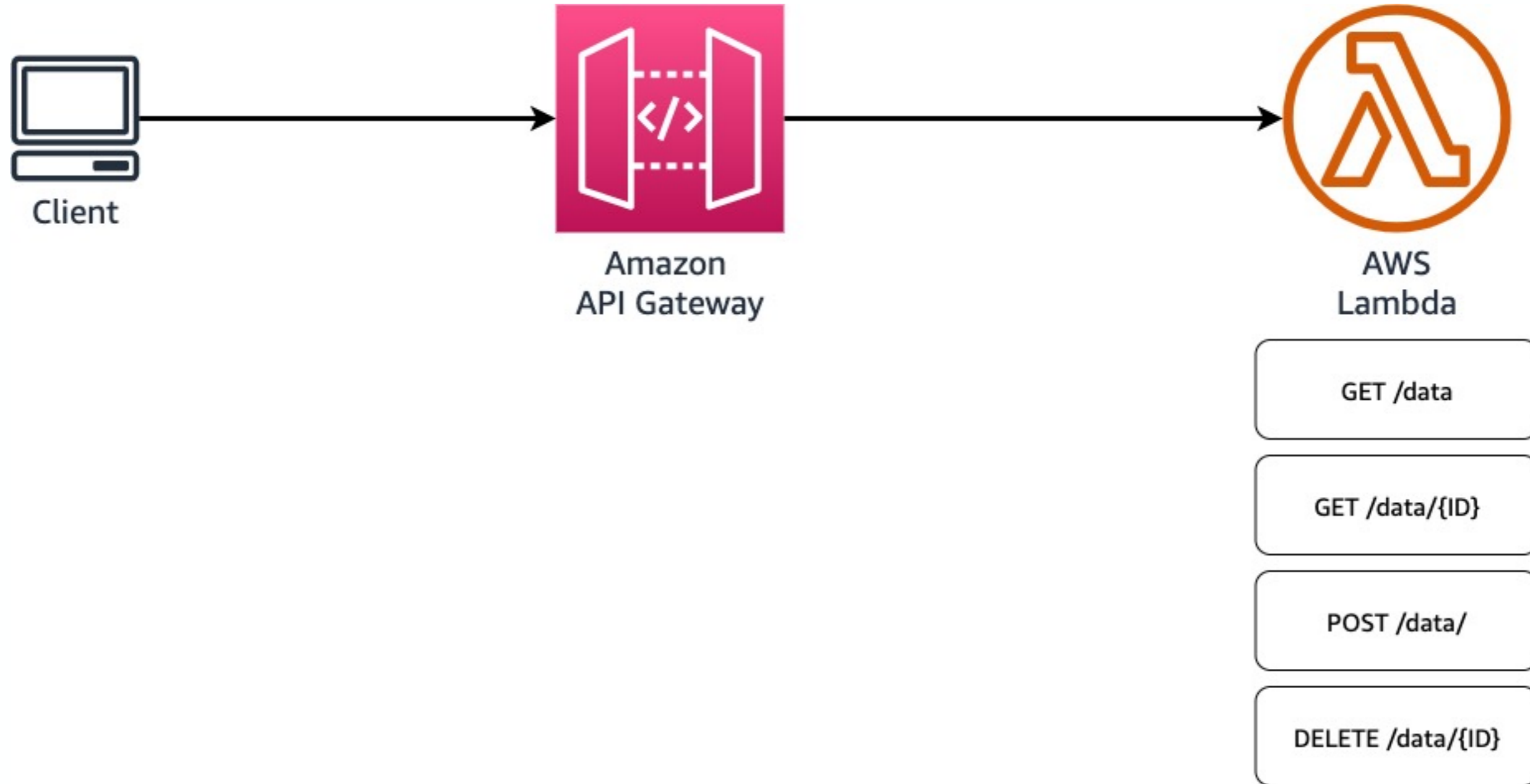


Application servers
Business logic

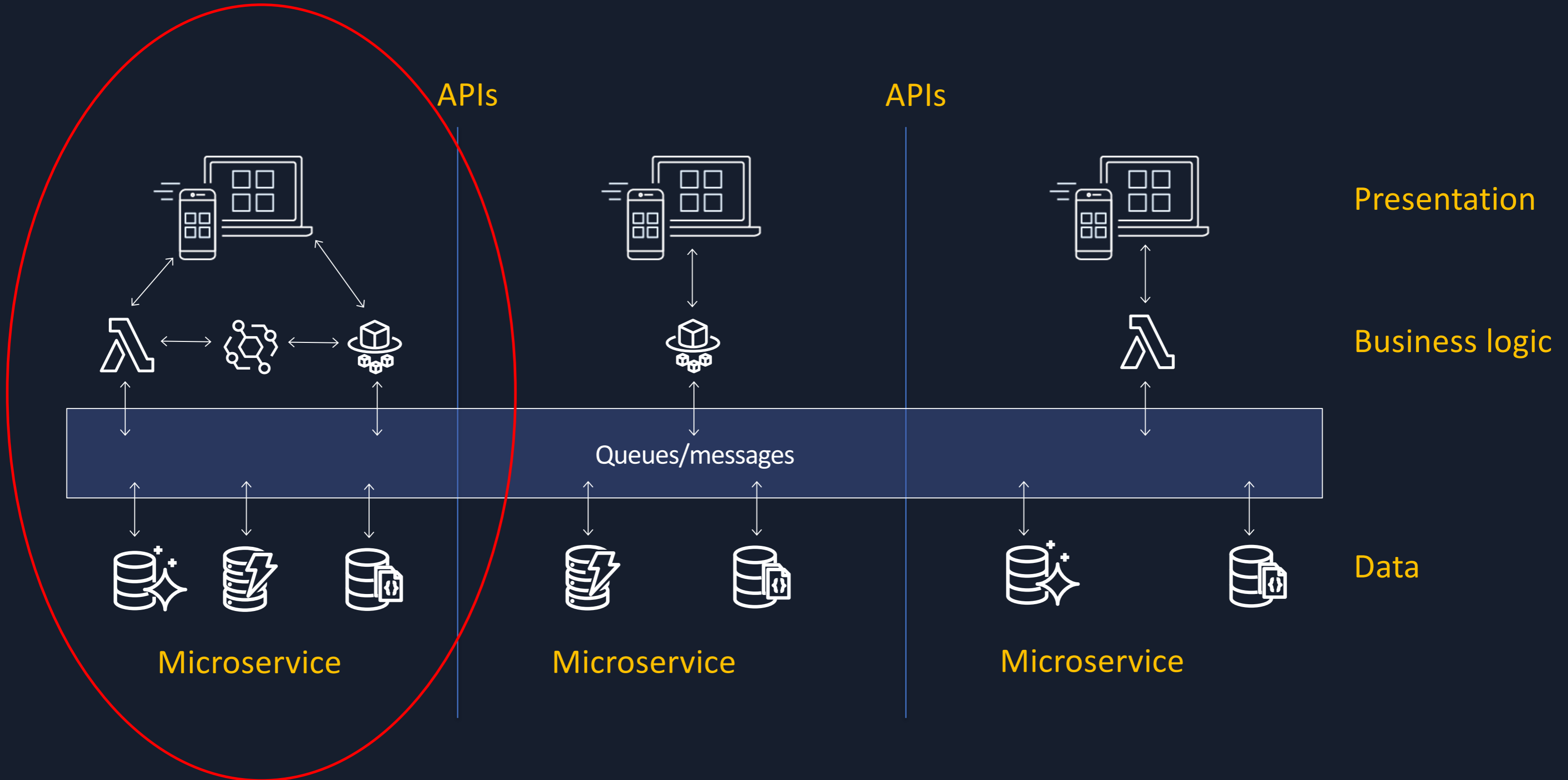


Database servers
Data layer

Use case: Monolith



A modern three-tier application architecture



A single microservice



AWS Lambda — Event-driven function

Event Source

Function

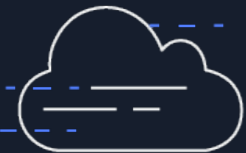
Services / Other



Changes in data state



Requests to endpoints



Changes in resource state



Node.js

Python

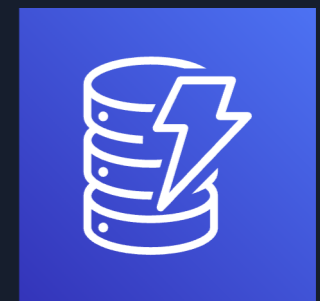
Java

C#

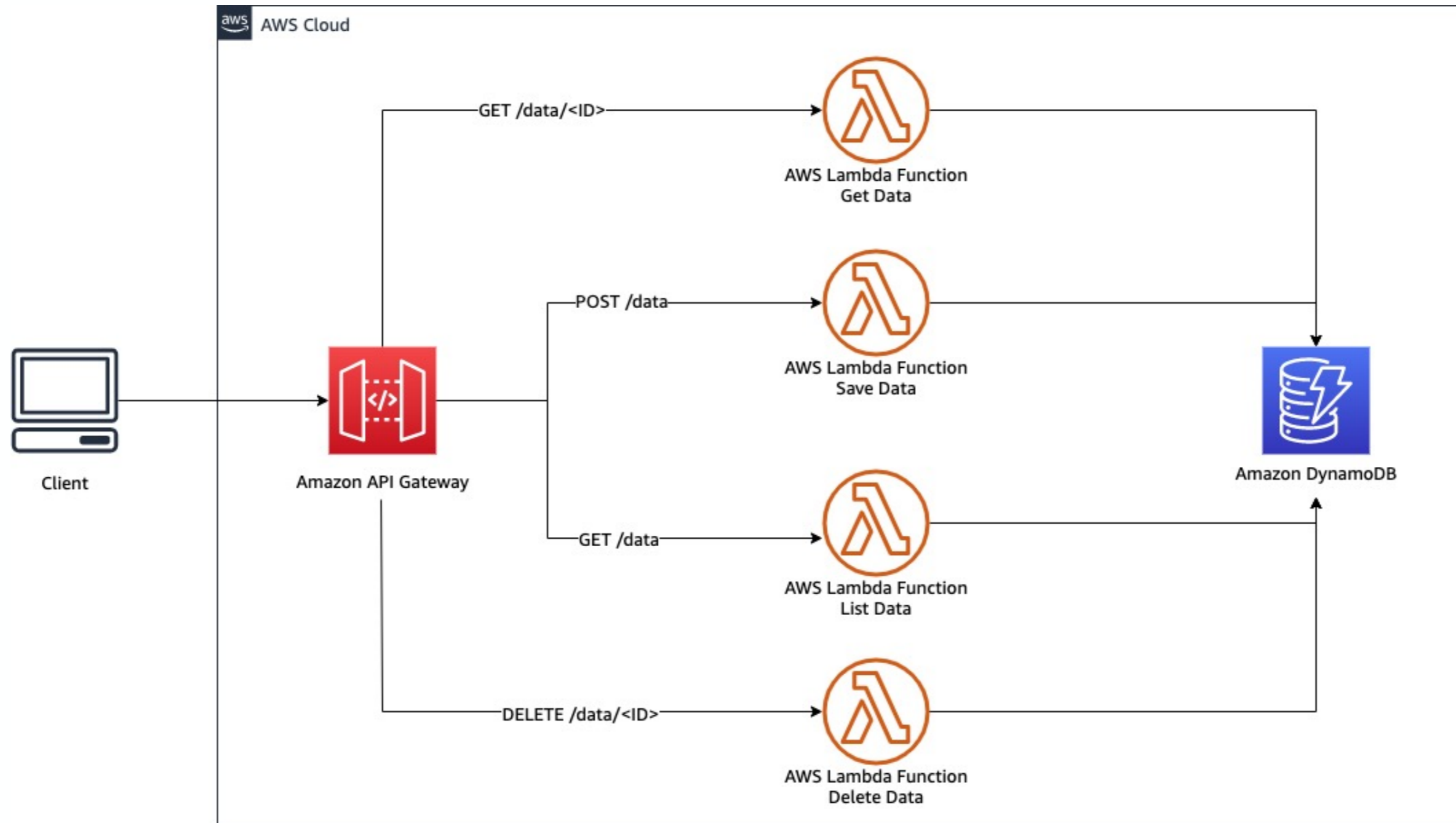
Go

Ruby

Bring Your Own



Use case: Serverless API Diagram

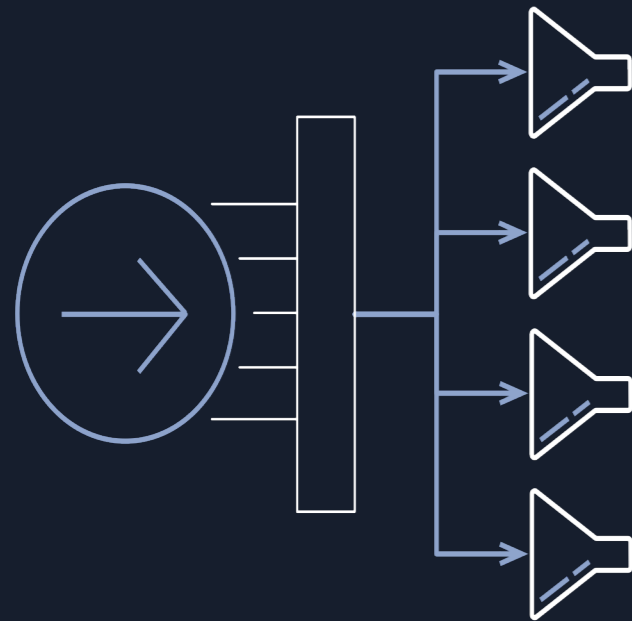


Application Integration

Consider how you integrate applications and modular services

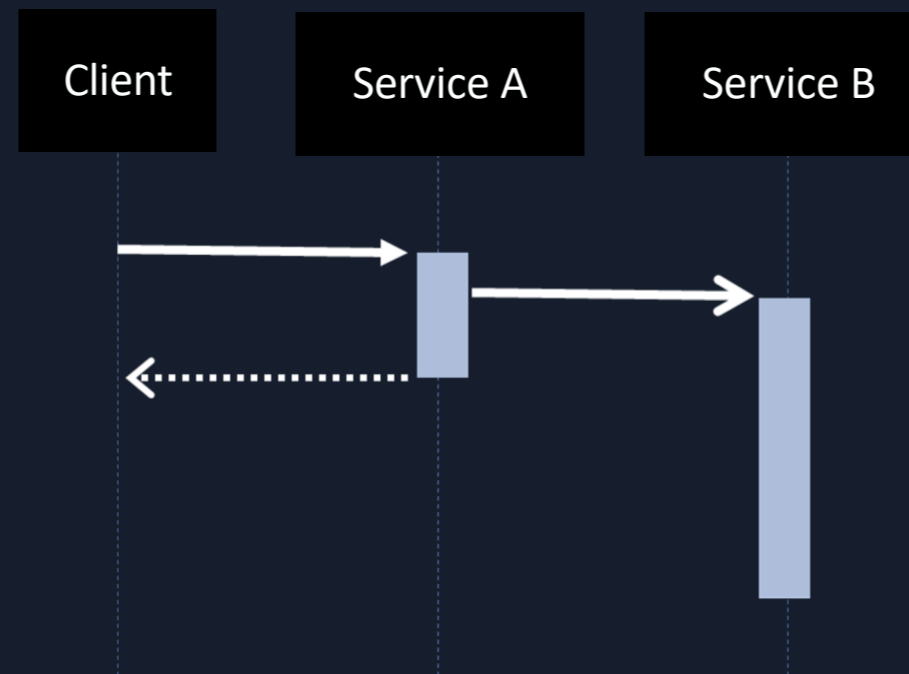
	Synchronous – API based	Asynchronous – event driven
Inter- / intraservice	Common for communication between apps	Common for communication within apps
Scalability	Requires tools to manage point-to-point connections	Is nearly infinitely scalable
Cost	Provisioning for peak use leads to low CPU utilization	Scales to 0 (pay-for-use cost benefits)
Latency	Can be very low	Is higher in theory, but latency requirements are rarely as low as expected (e.g., consider P50, P99)
Agility	Is easy to get started Is hard to use point to point in large scale	Decoupled systems increase agility dramatically

Event-driven architectures drive reliability and scalability



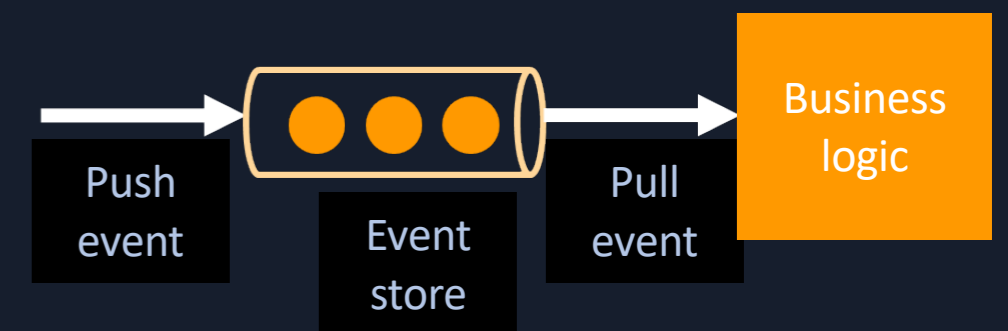
Event routers

Abstract producers and consumers from each other



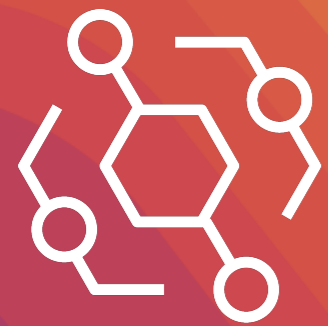
Asynchronous events

Improve responsiveness and reduce dependencies



Event stores

Buffer messages until services are available to process

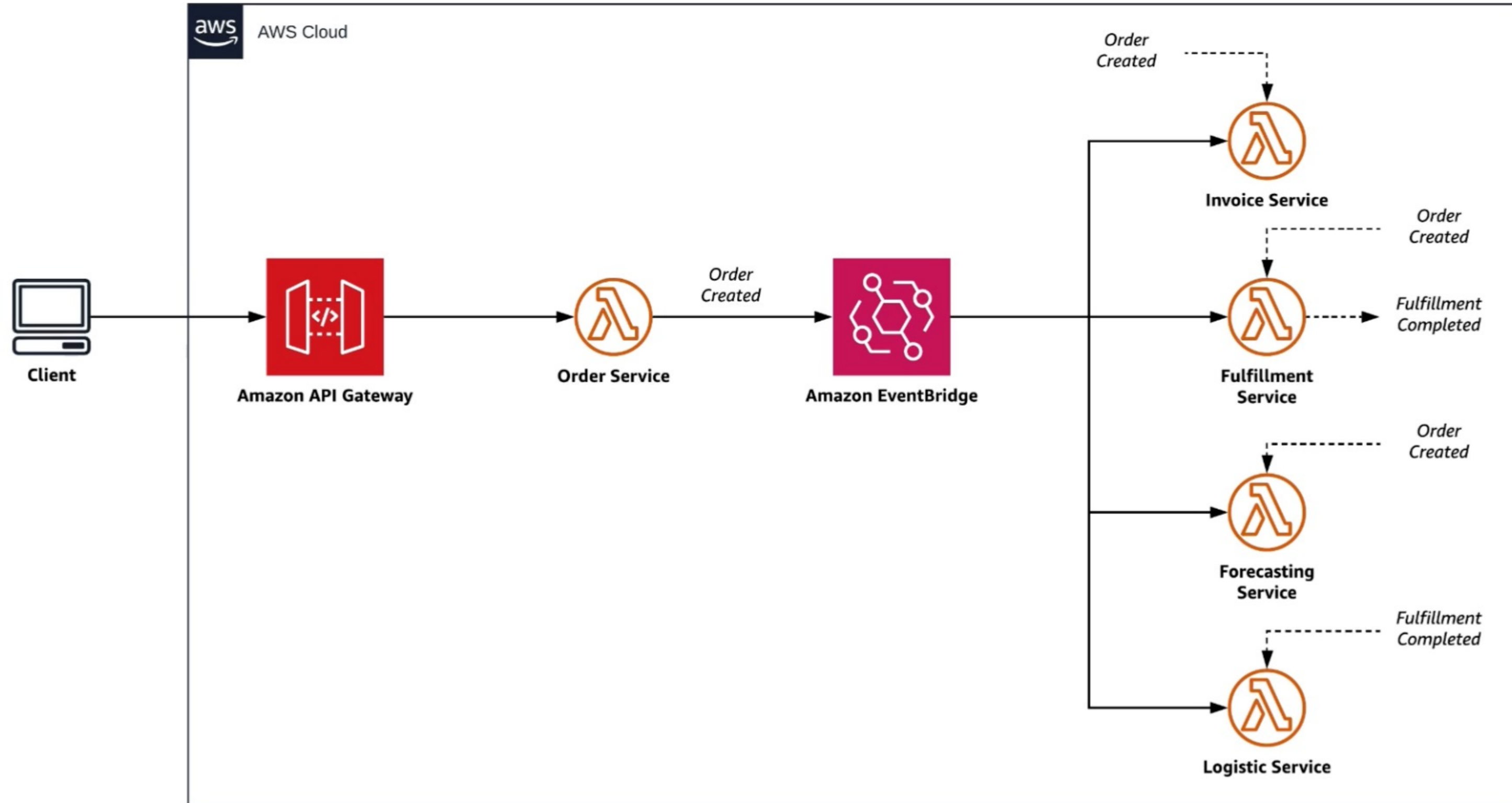


Amazon EventBridge

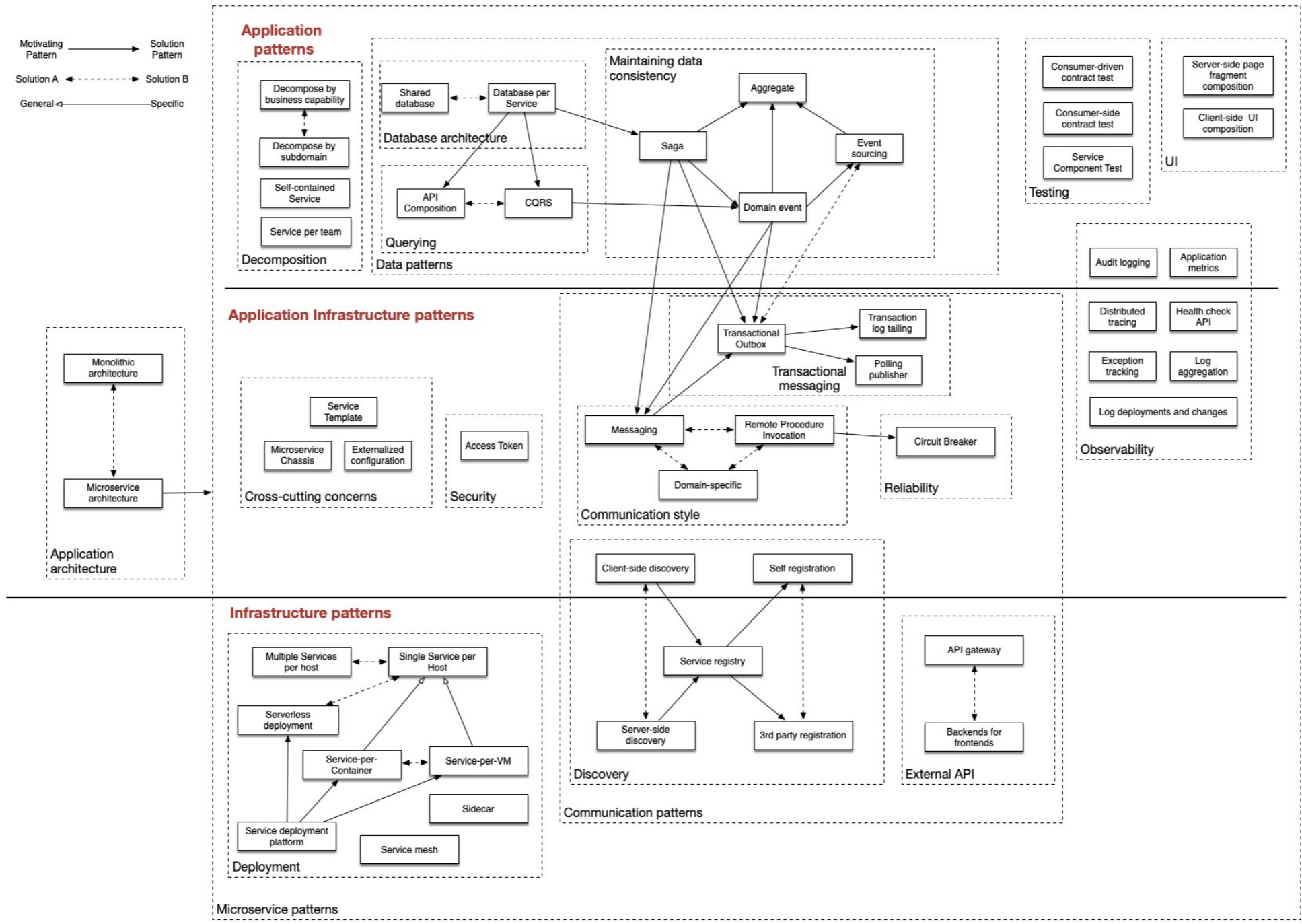
A serverless event bus service for AWS services, your own applications, and SaaS providers.

- Removes friction of writing “point-to-point” integrations
- Works across dozens of AWS and SaaS applications
- Provides simple programming model
- Fully managed; pay-as-you-go

Use case: Event-Driven Architecture



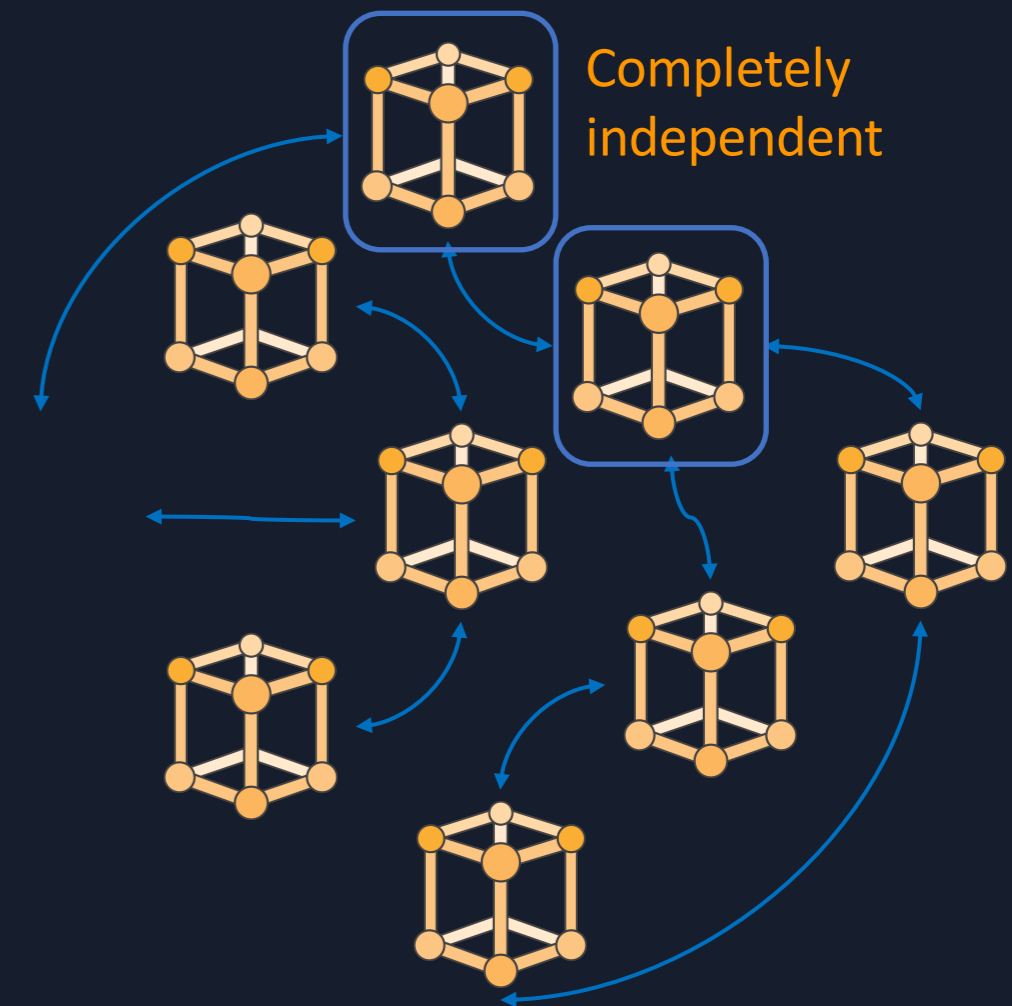
Learn more patterns



Advantages and Why

Small independent Teams

- **Requires knowledge** of a small, well-understood domain
- **Empowers** small, independent teams to move at their own pace, shortening the cycle times
- **Fully-responsible** including development, staging, and production



Quality

- **Reduced** complexity
- **Smaller deployments** & failure isolation, and allows graceful failure handling
- **Fully responsible** teams



Innovation

- **Vertical alignment** with the business owners
- **Fast-feedback** loops & low cost of failure
- **Freedom** to choose and replace the technology stack individually (per domain)



Scalability

- **Properly decoupled** services can be scaled horizontally and independently from each other
- **Leverage** the optimal data persistence solution per service
- **Appropriate** and optimal technologies for a specific service





Thank you

Download the PPT
Code Repo + Demo



<https://bit.ly/aws-community-asean-content>

