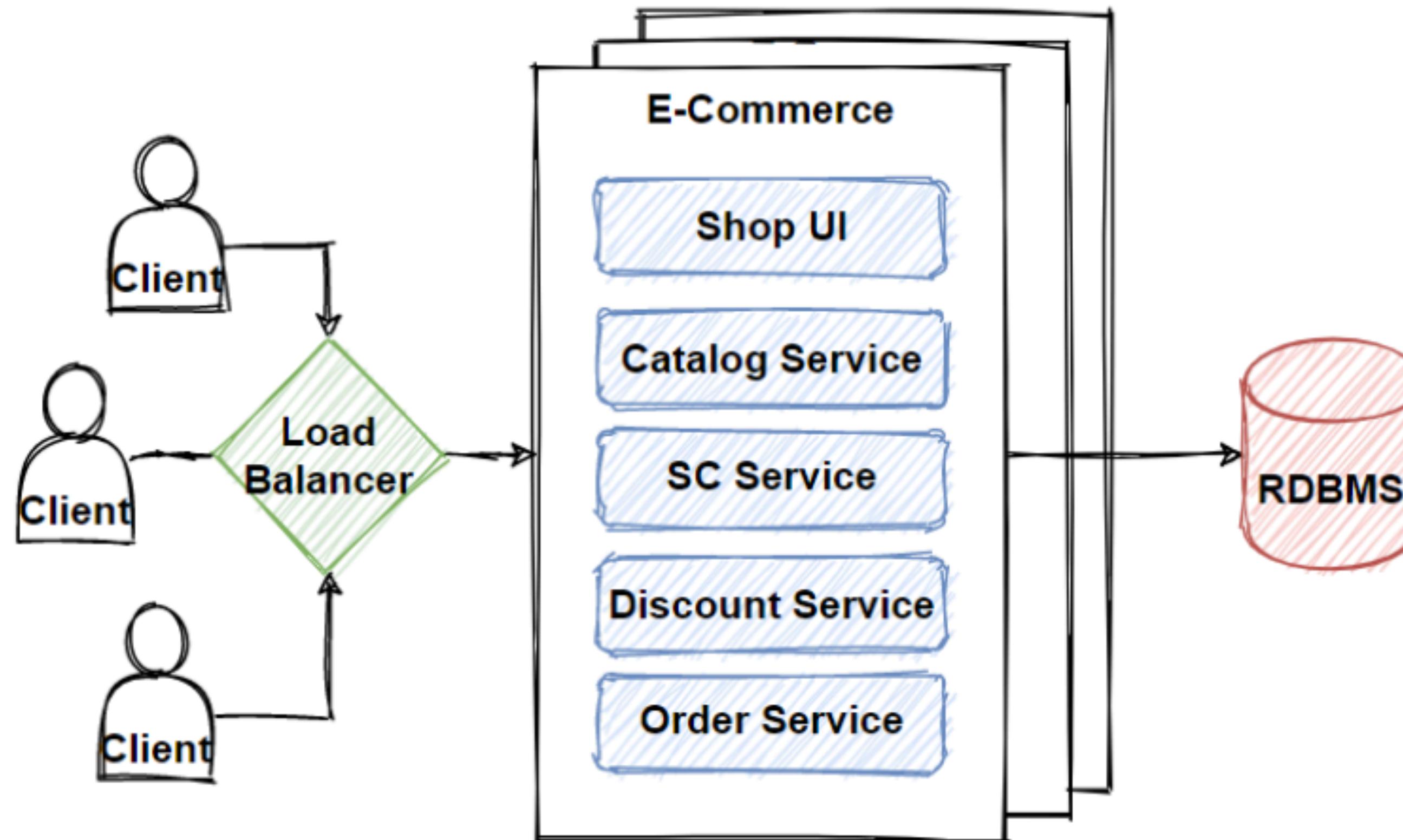


# API Gateway | Service Discovery

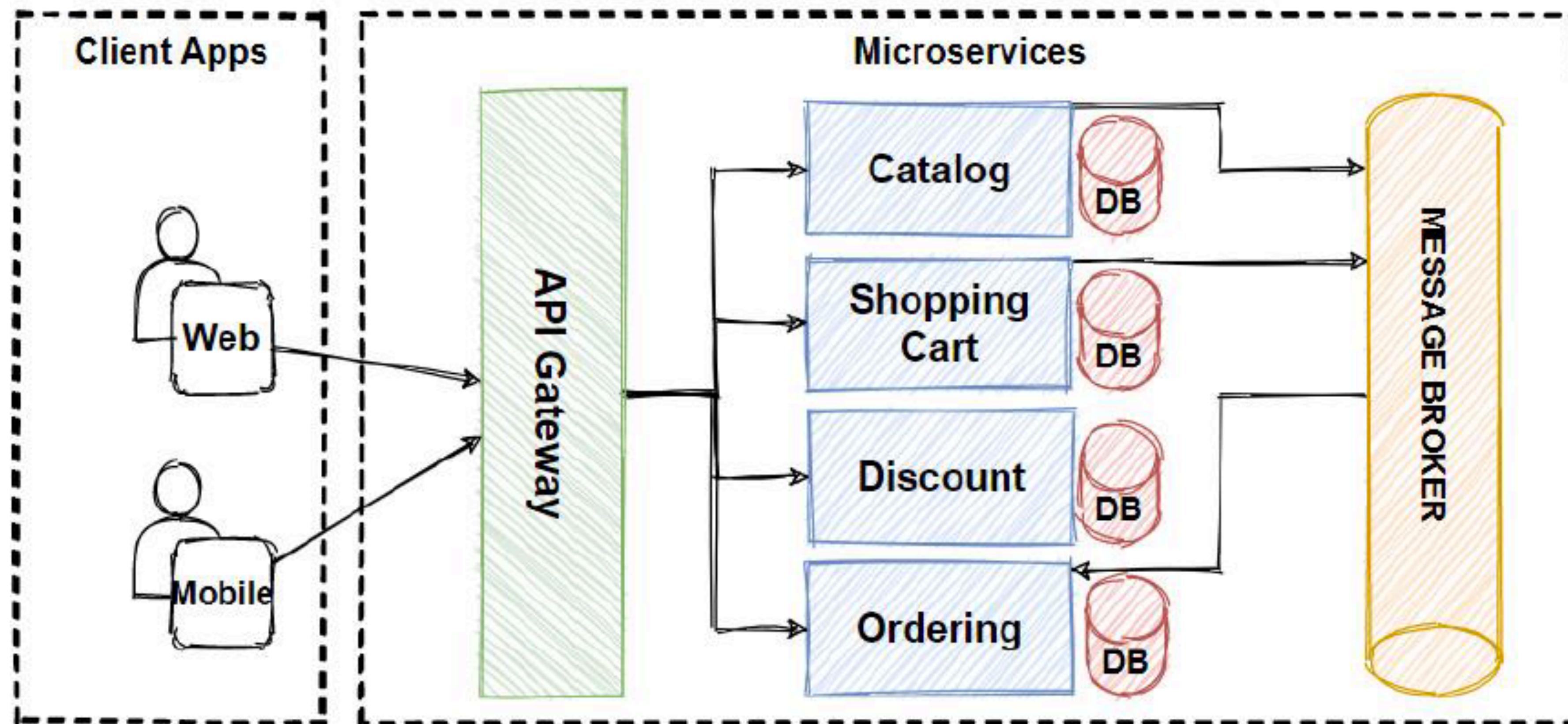
บรรยายโดย อาจารย์สัมัญ น้อยจันทร์  
คณะเทคโนโลยีสารสนเทศ  
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

# Microservices Recap



Monolithic Architecture

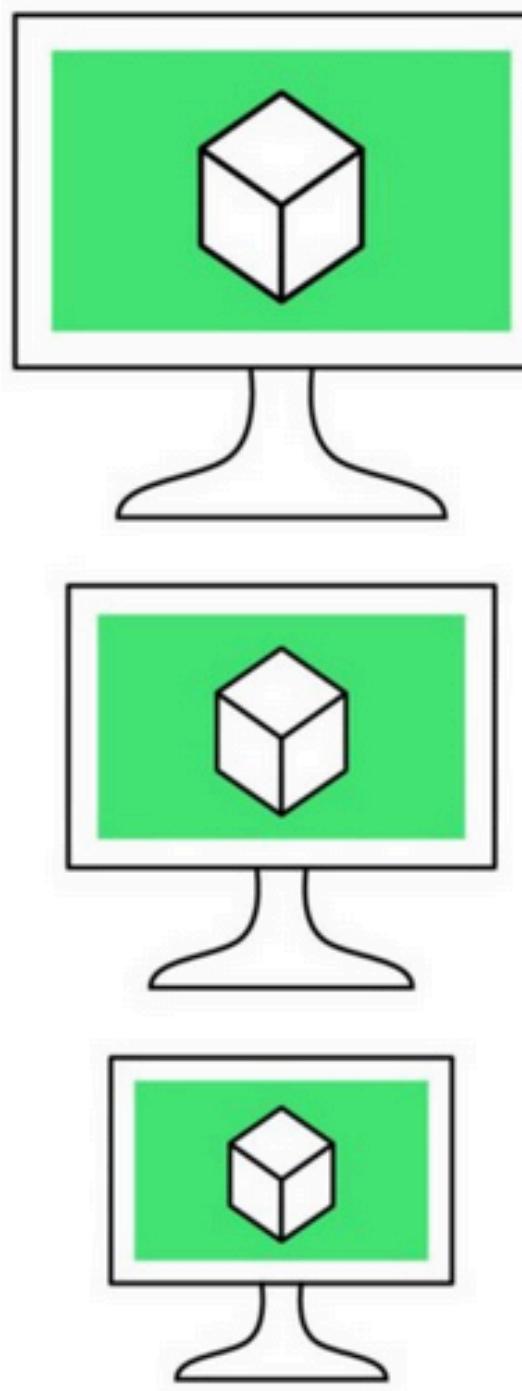
# Microservices Recap



Microservices Architecture

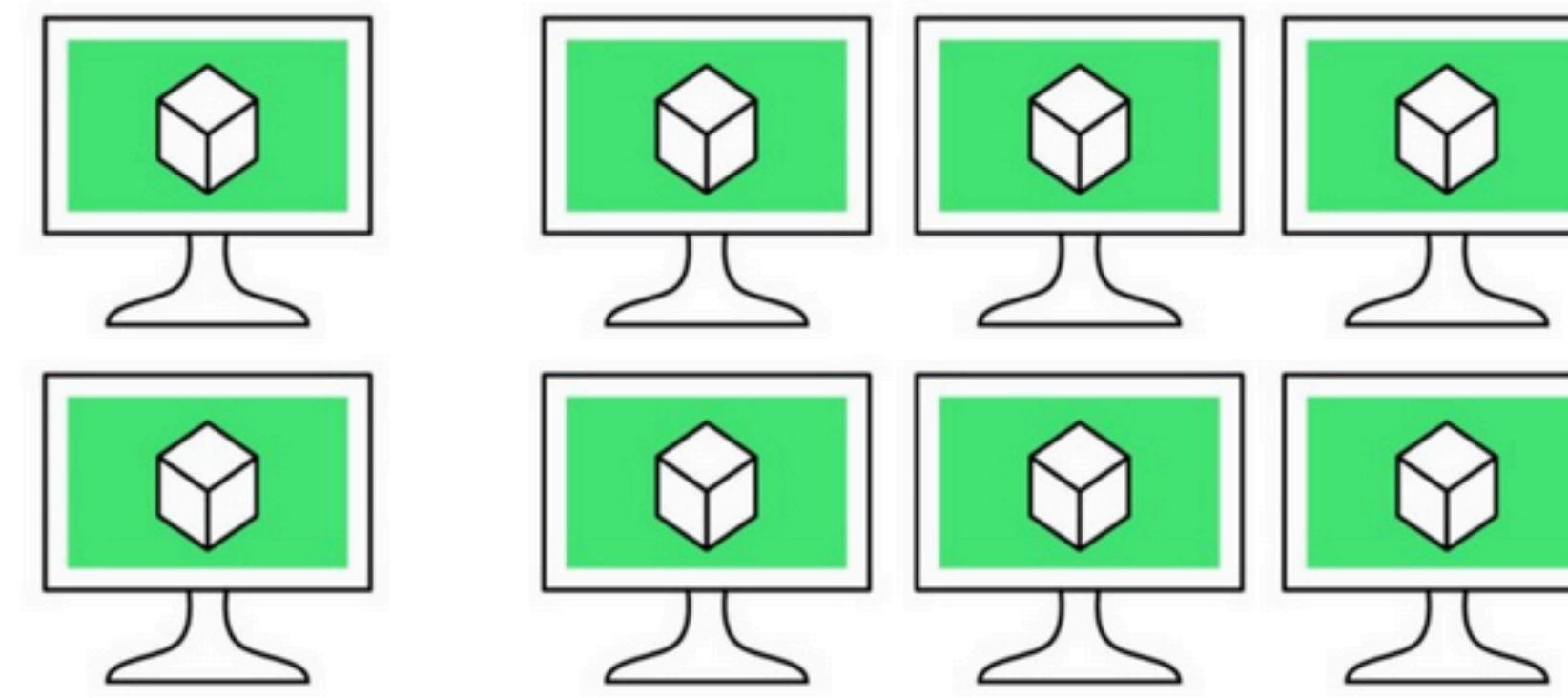
# Why Microservices?

**Vertical scaling**



Increase in processing power

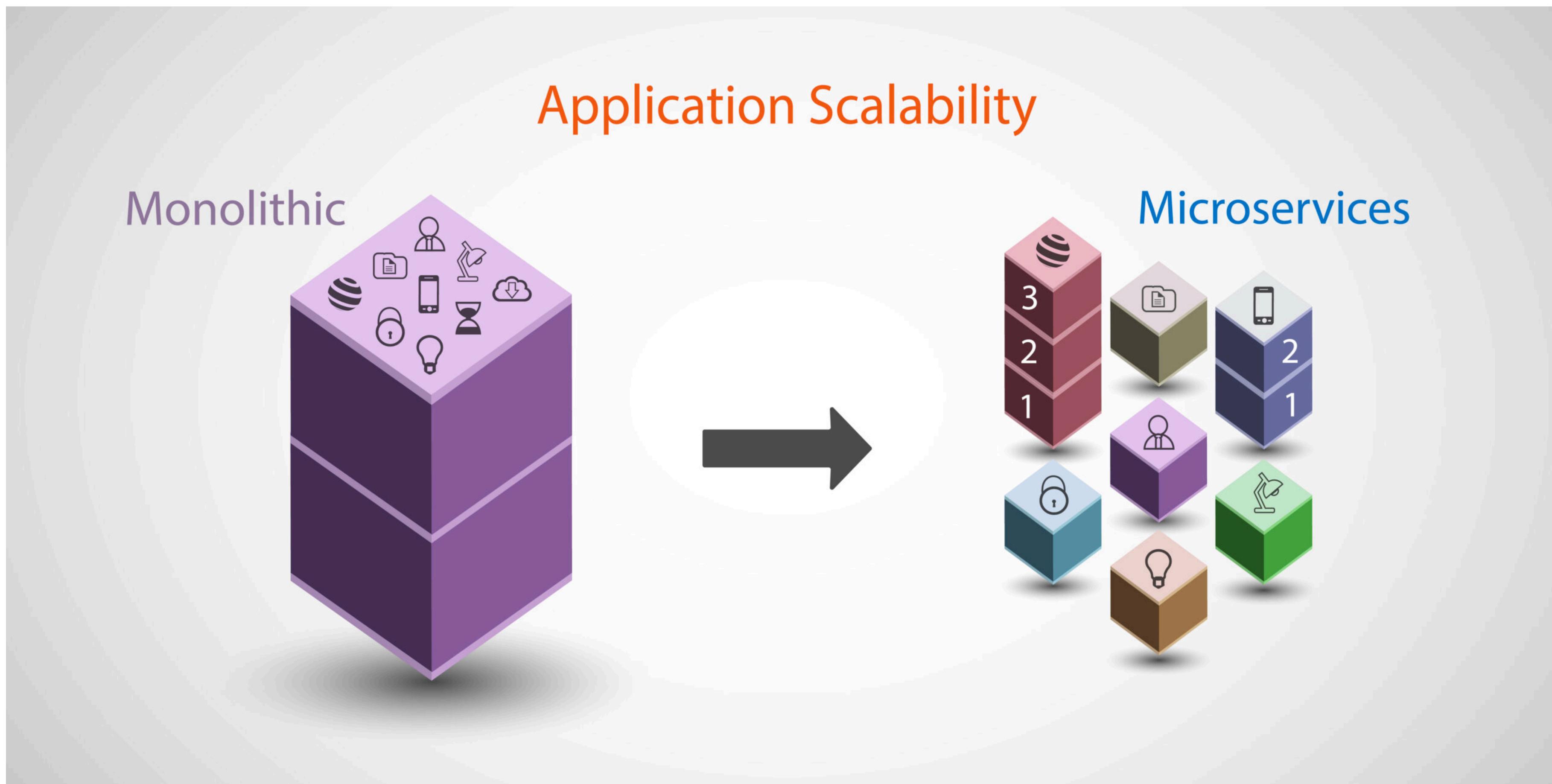
**Horizontal scaling**



Increase in number of machines

**Scalability**

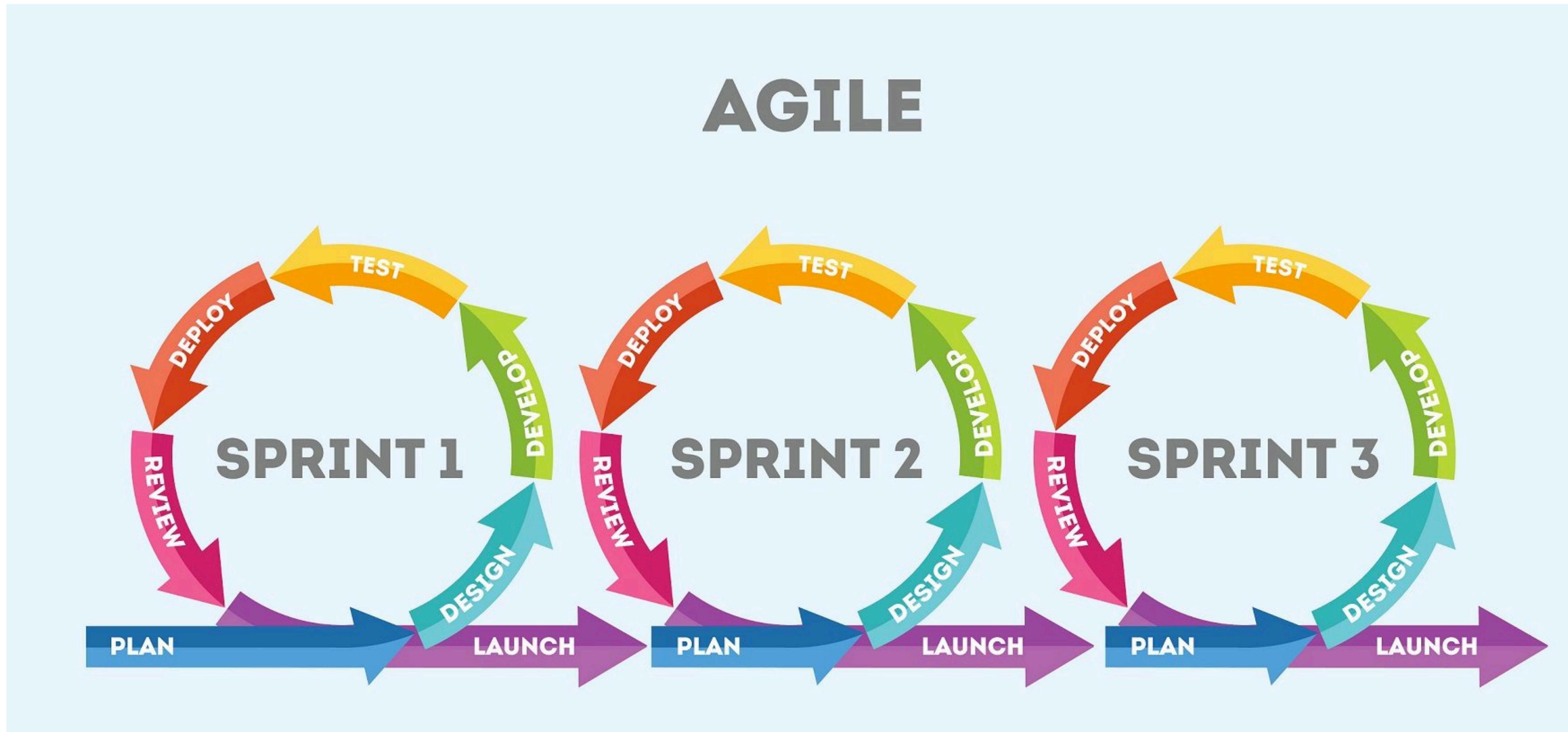
# Why Microservices?



## Scalability

Microservices architecture allows for granular scalability.

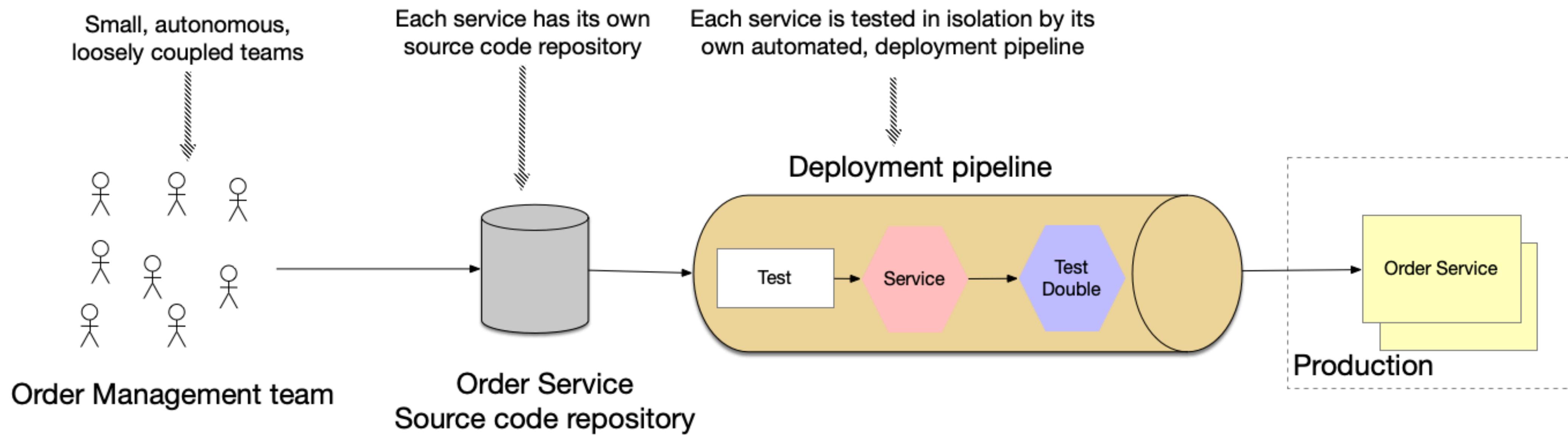
# Why Microservices?



**Faster Development and Deployment**

Microservices enable agile development practices.

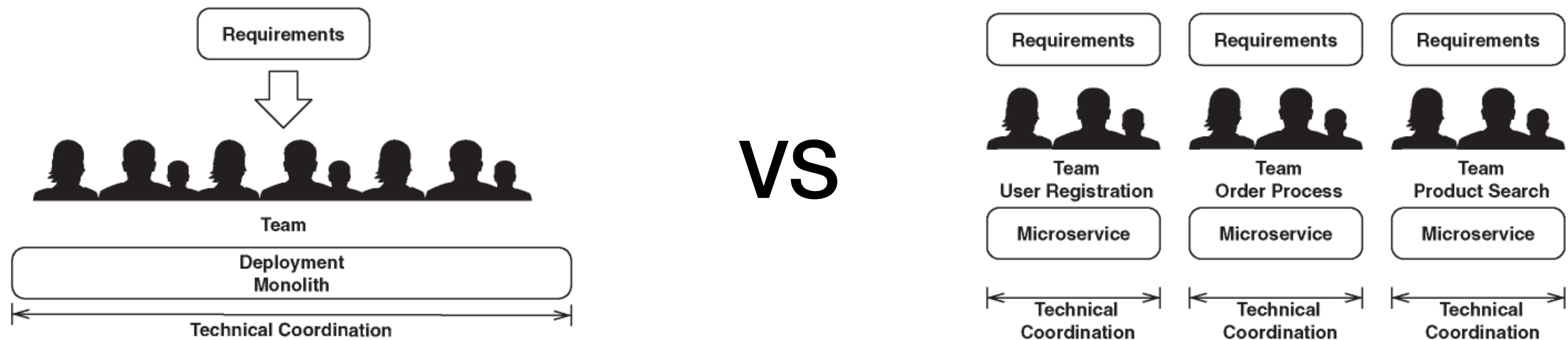
# Why Microservices?



## Faster Development and Deployment

Smaller, self-contained services are easier to develop, test, and deploy, leading to faster release cycles.

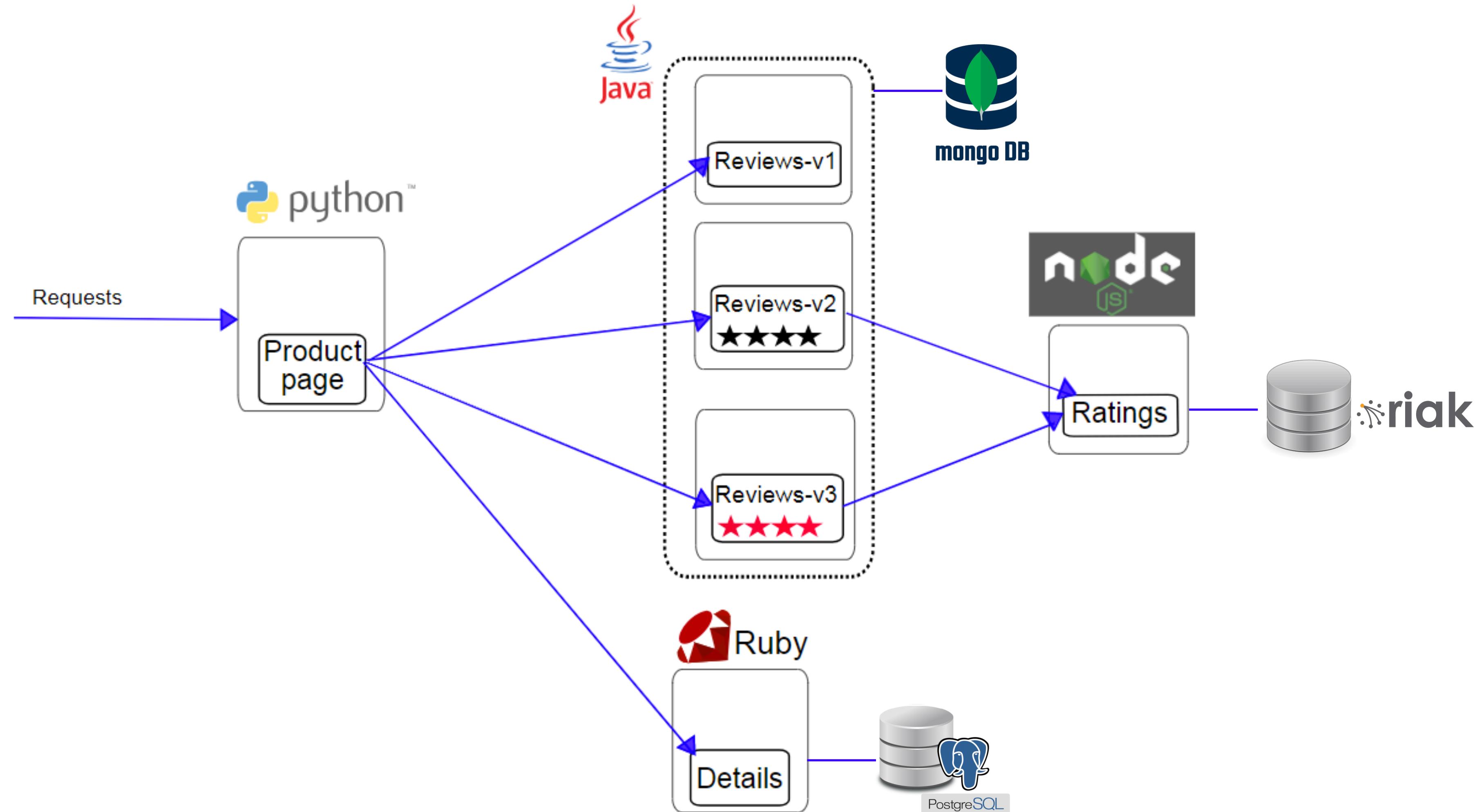
# Why Microservices?



## Faster Development and Deployment

Smaller, self-contained services are easier to develop, test, and deploy, leading to faster release cycles.

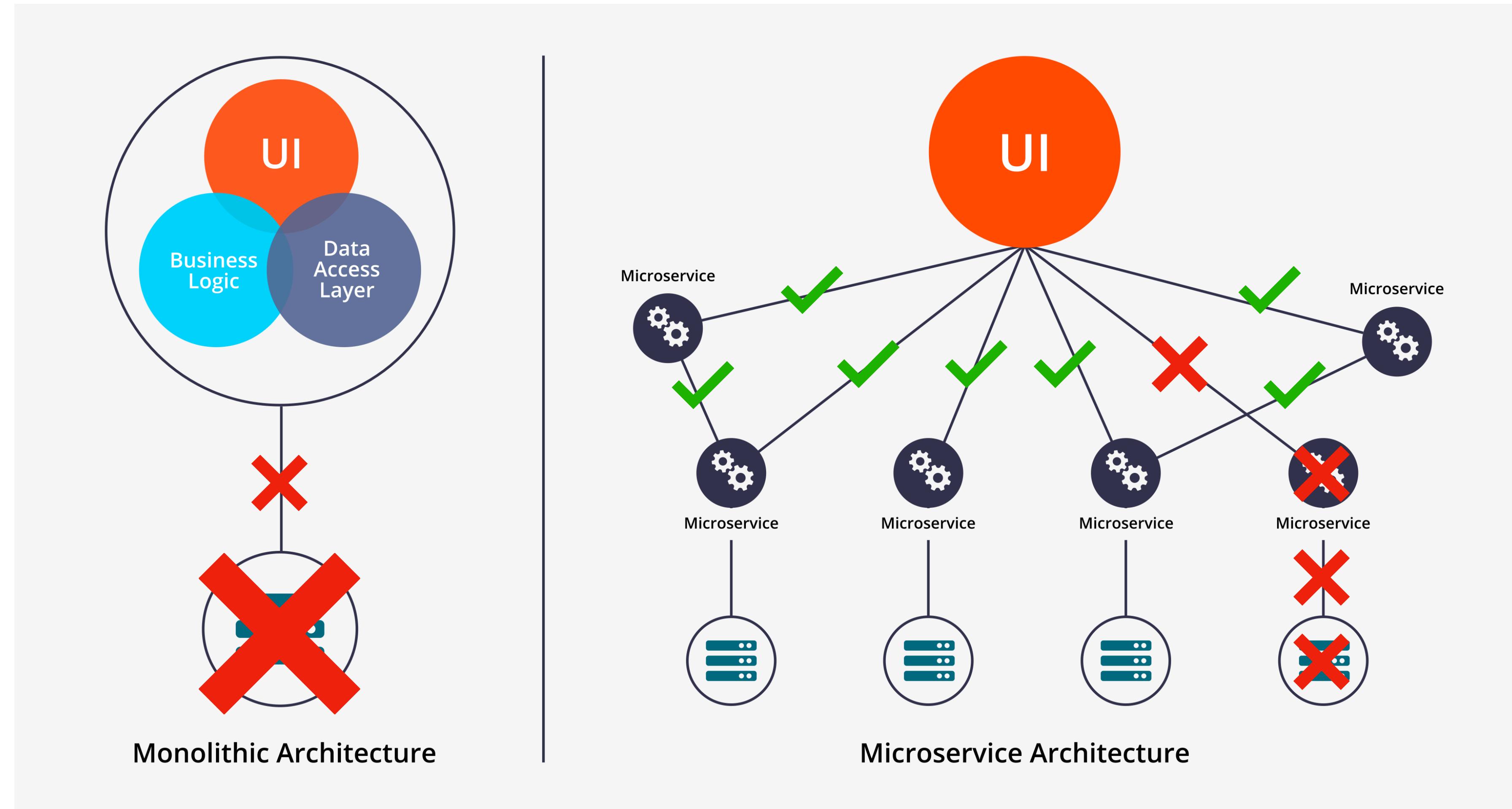
# Why Microservices?



## Technology Diversity

Microservices support the use of different technologies and programming languages for different services.

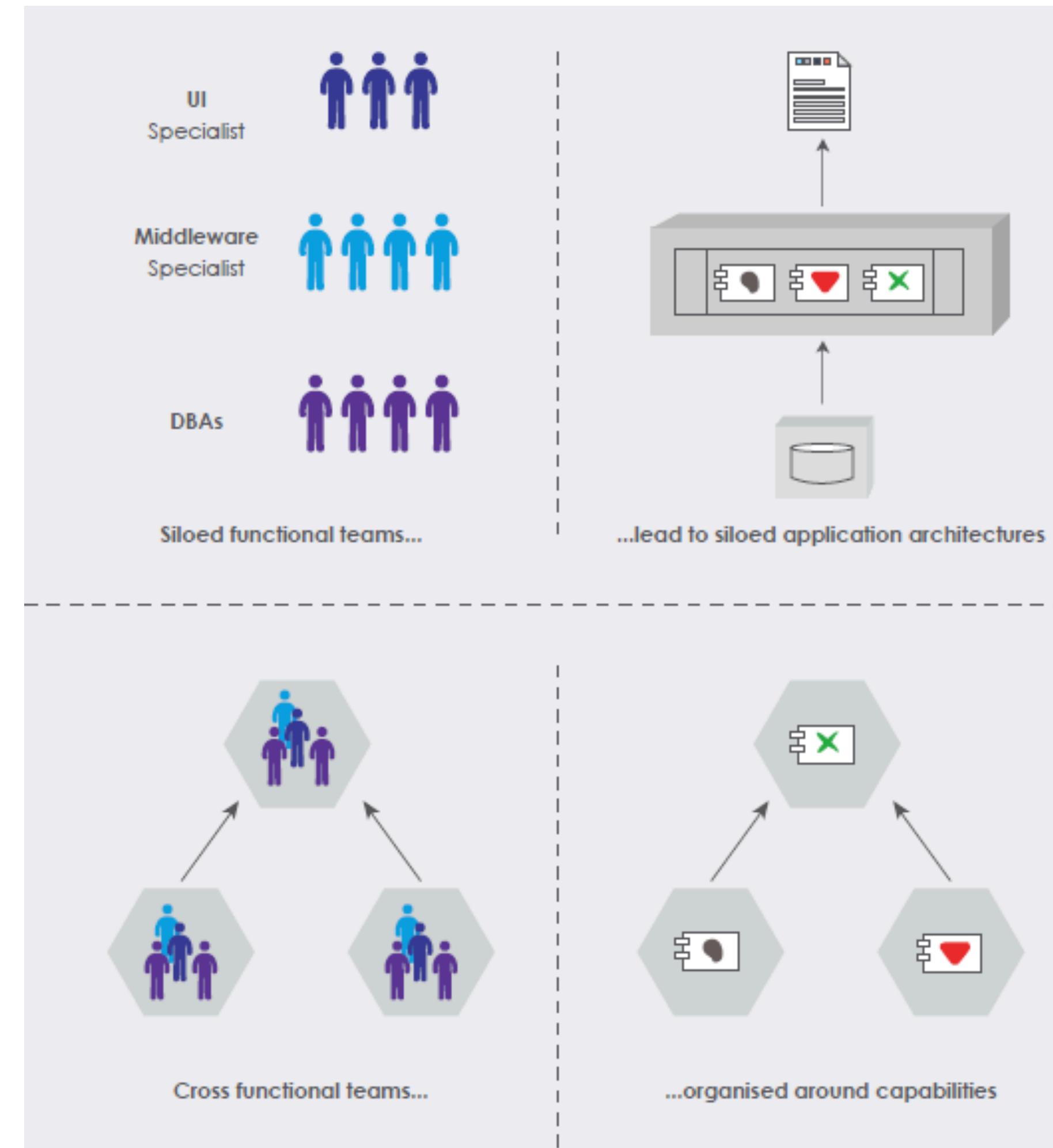
# Why Microservices?



## Fault Isolation

Each microservice operates independently. If one service fails or experiences issues, it doesn't necessarily impact the entire system.

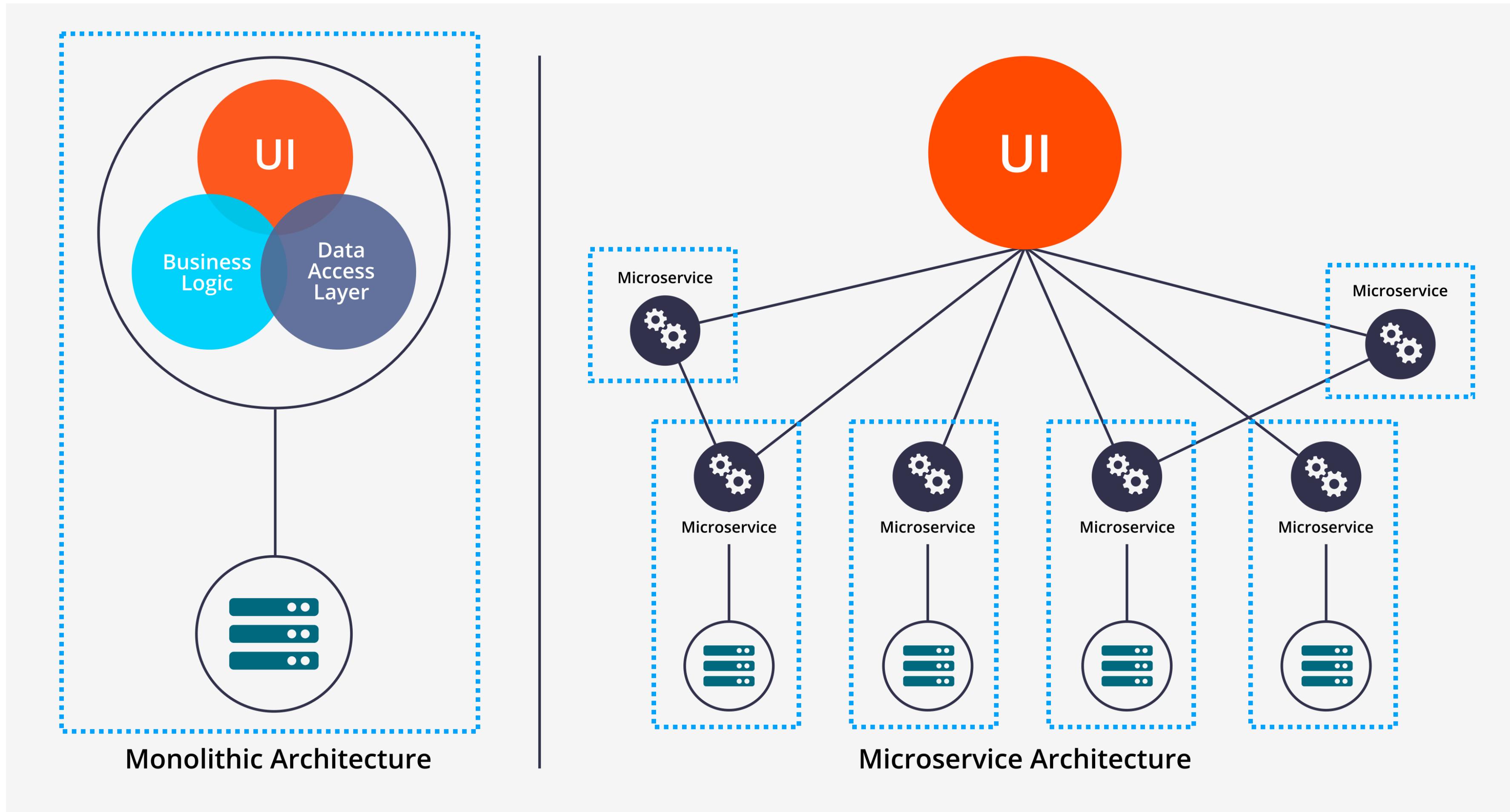
# Why Microservices?



## Improved Collaboration

Microservices architecture encourages cross-functional teams responsible for individual services.

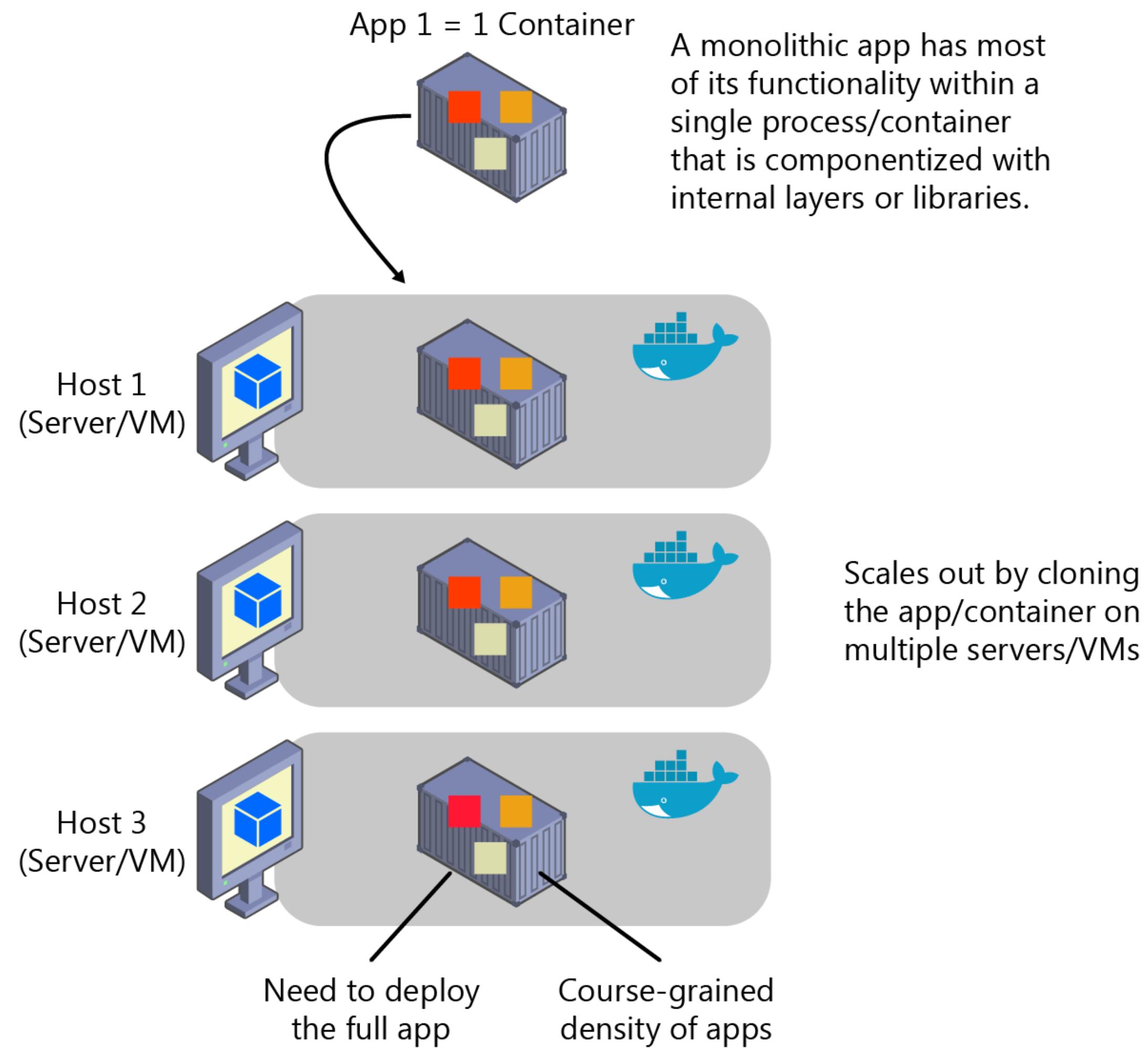
# Why Microservices?



## Ease of Maintenance

Smaller codebases are easier to maintain and update.

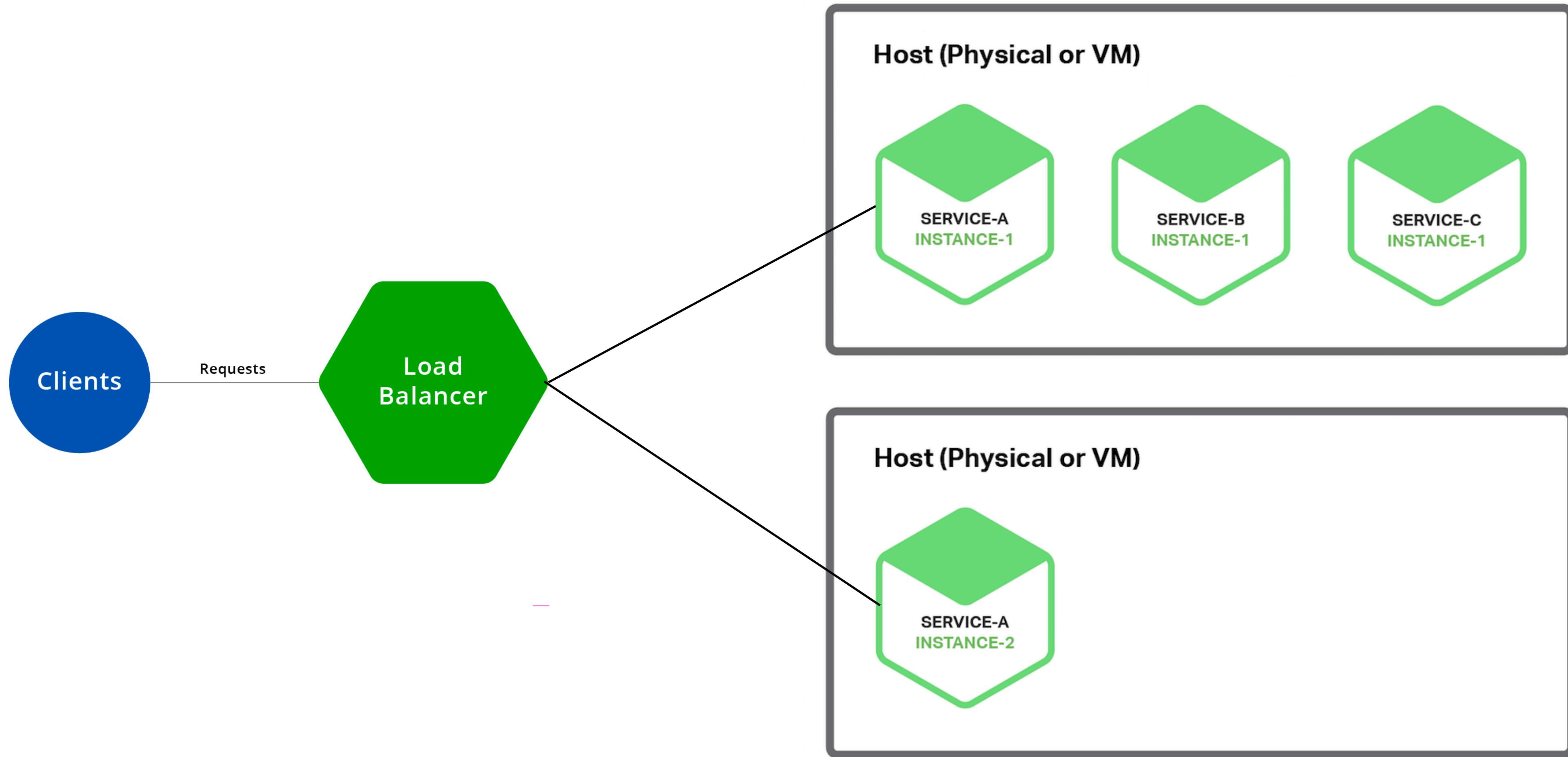
# Why Microservices?



## Resilience and Availability

Redundancy and failover strategies are implemented for the entire monolithic app.

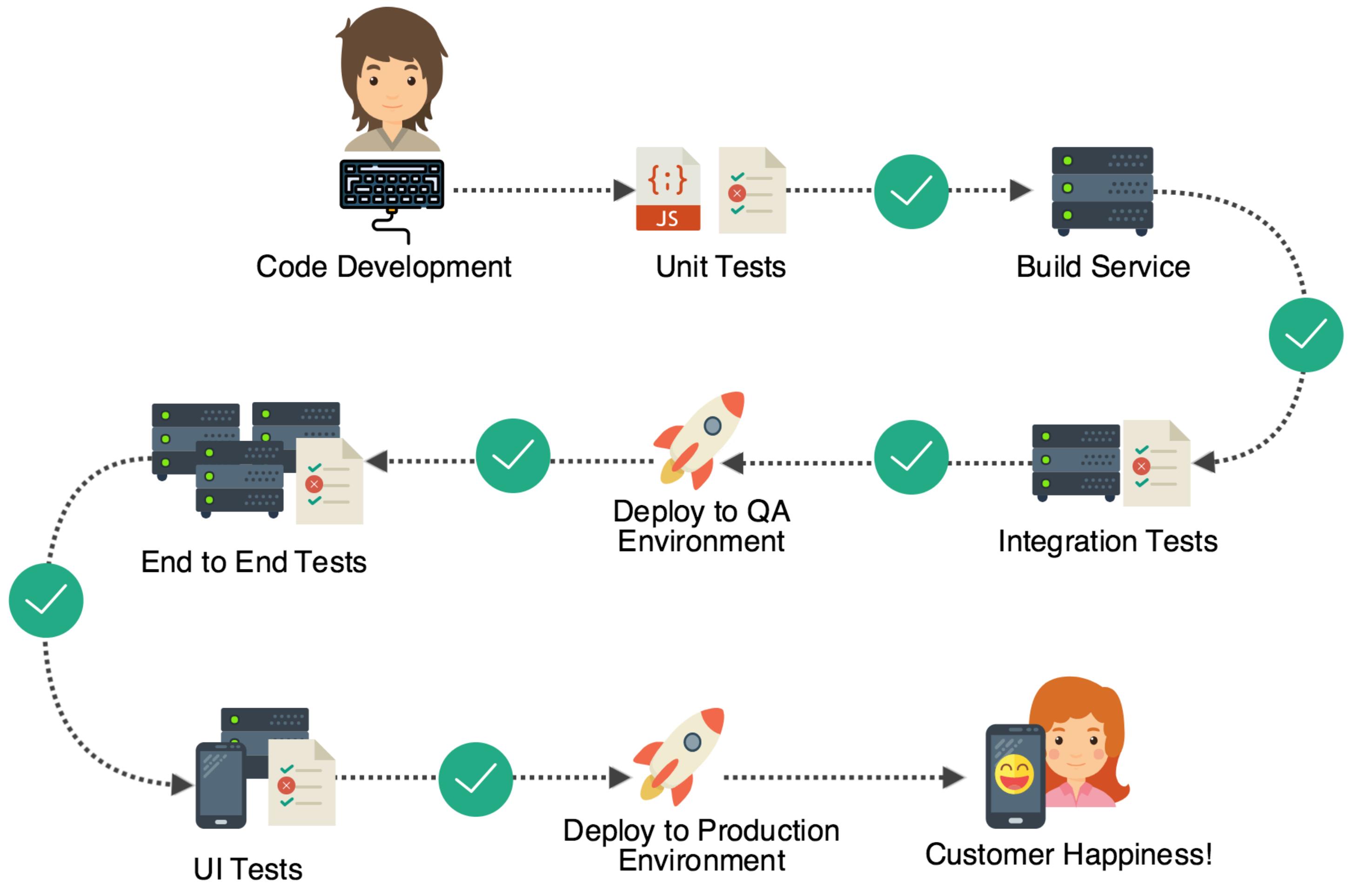
# Why Microservices?



## Resilience and Availability

Redundancy and failover strategies can be implemented at the microservice level.

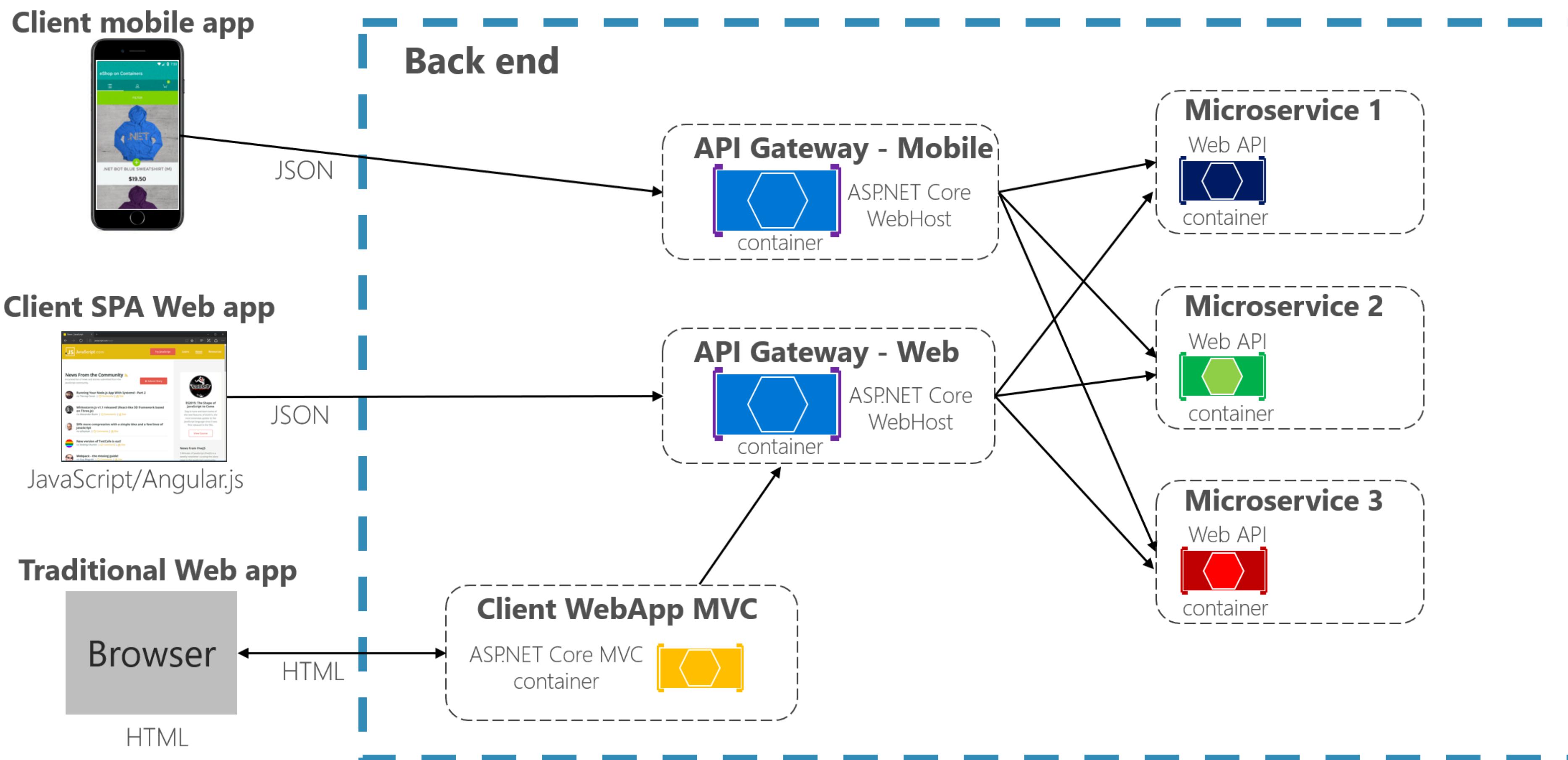
# Why Microservices?



## Enhanced Testing

Smaller services are easier to test comprehensively.

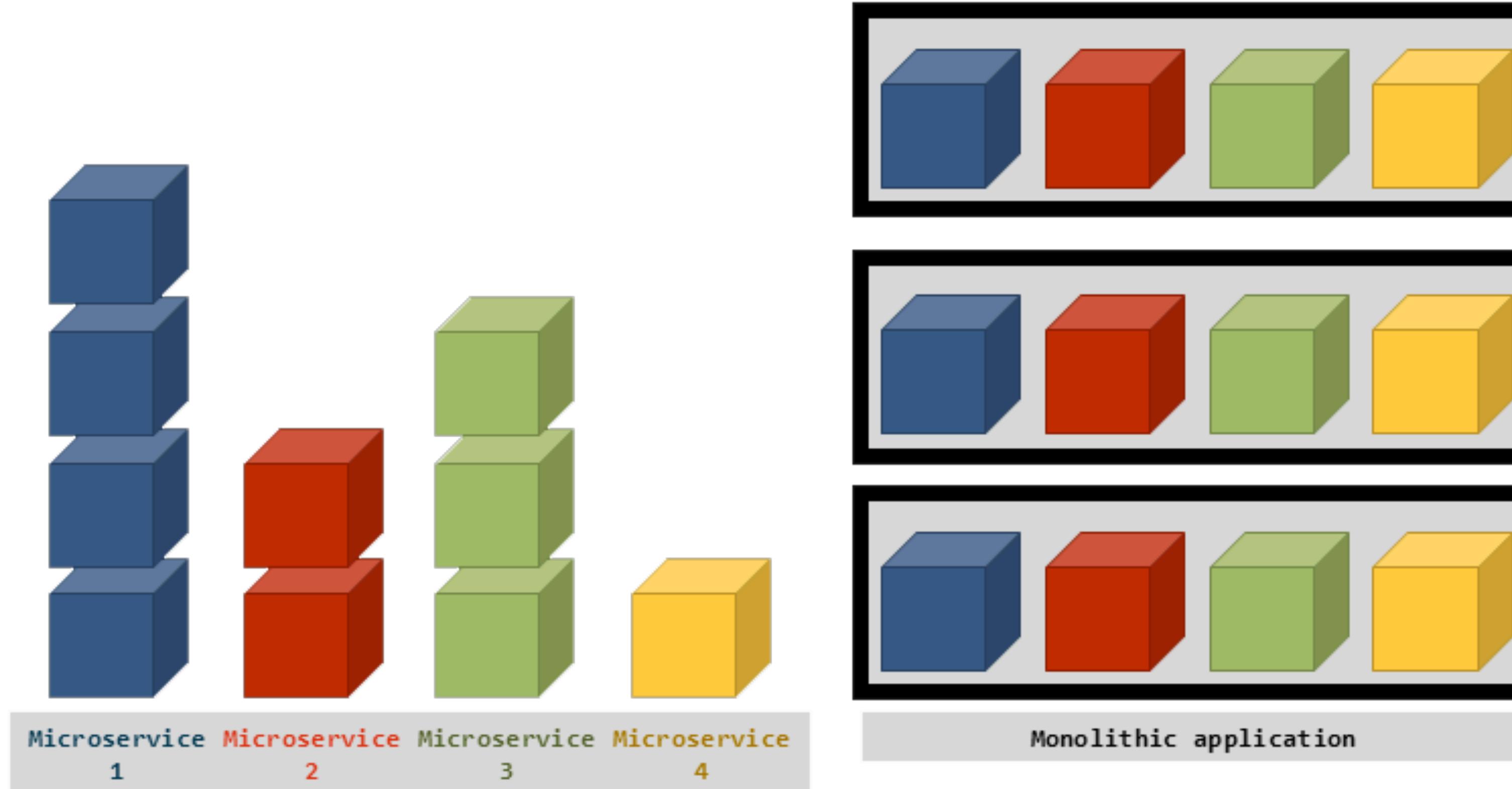
# Why Microservices?



## Improved User Experience

Microservices can be designed to serve specific user-facing features or functionalities.

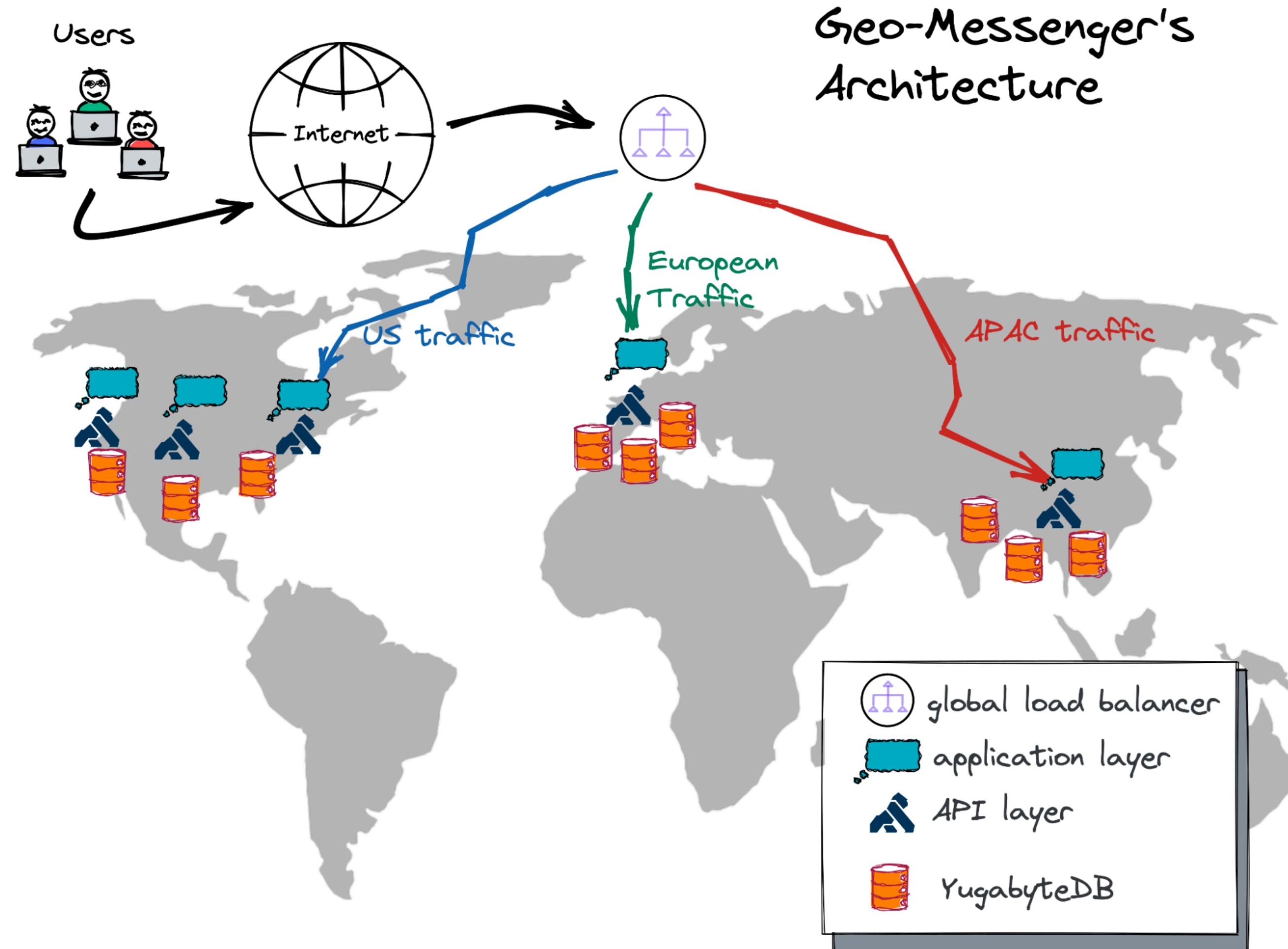
# Why Microservices?



## Elasticity

Auto-scaling can be implemented at the microservice level.

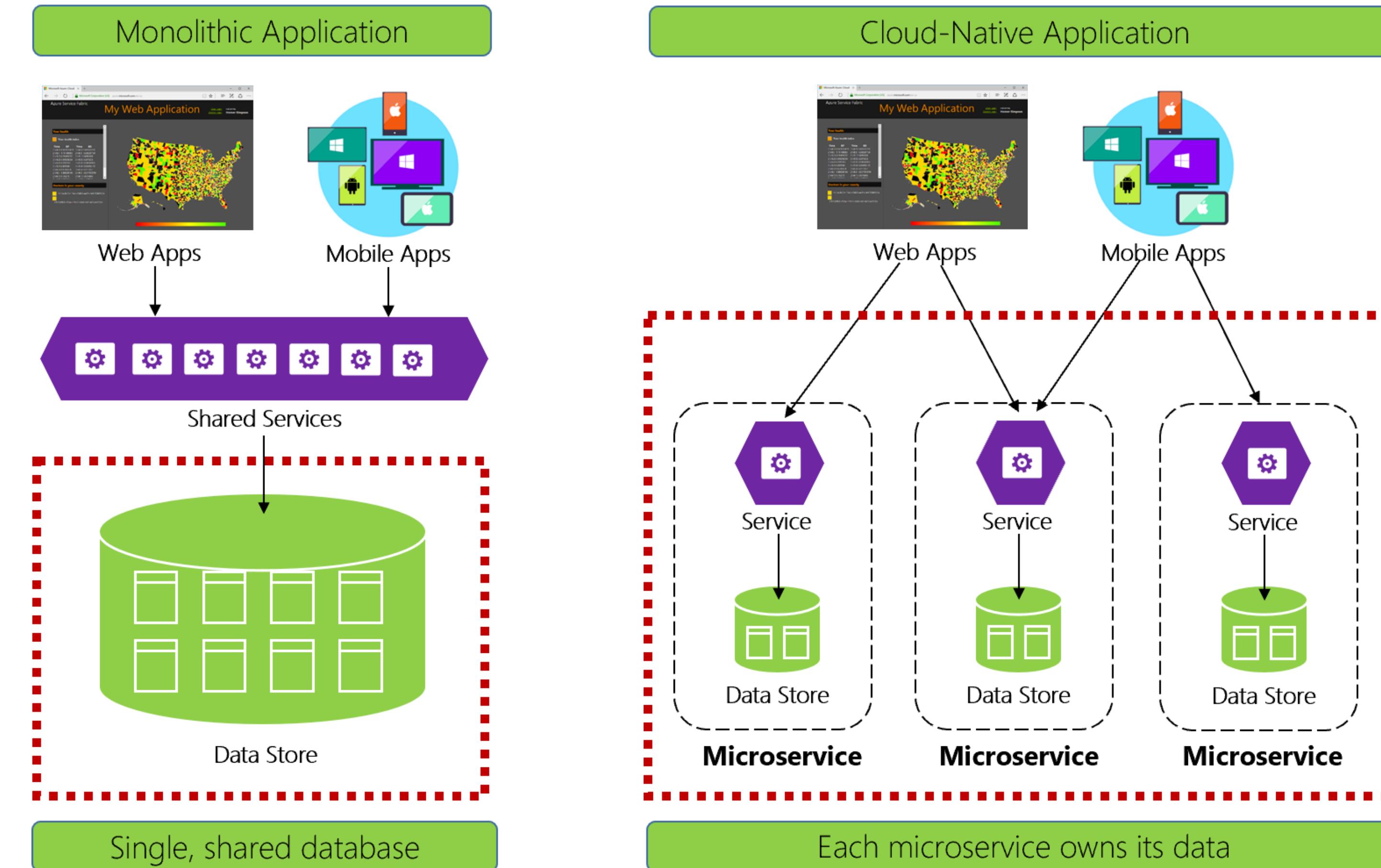
# Why Microservices?



## Global Reach

Microservices can be deployed in multiple geographic locations.

# Why Microservices?



## Decentralization

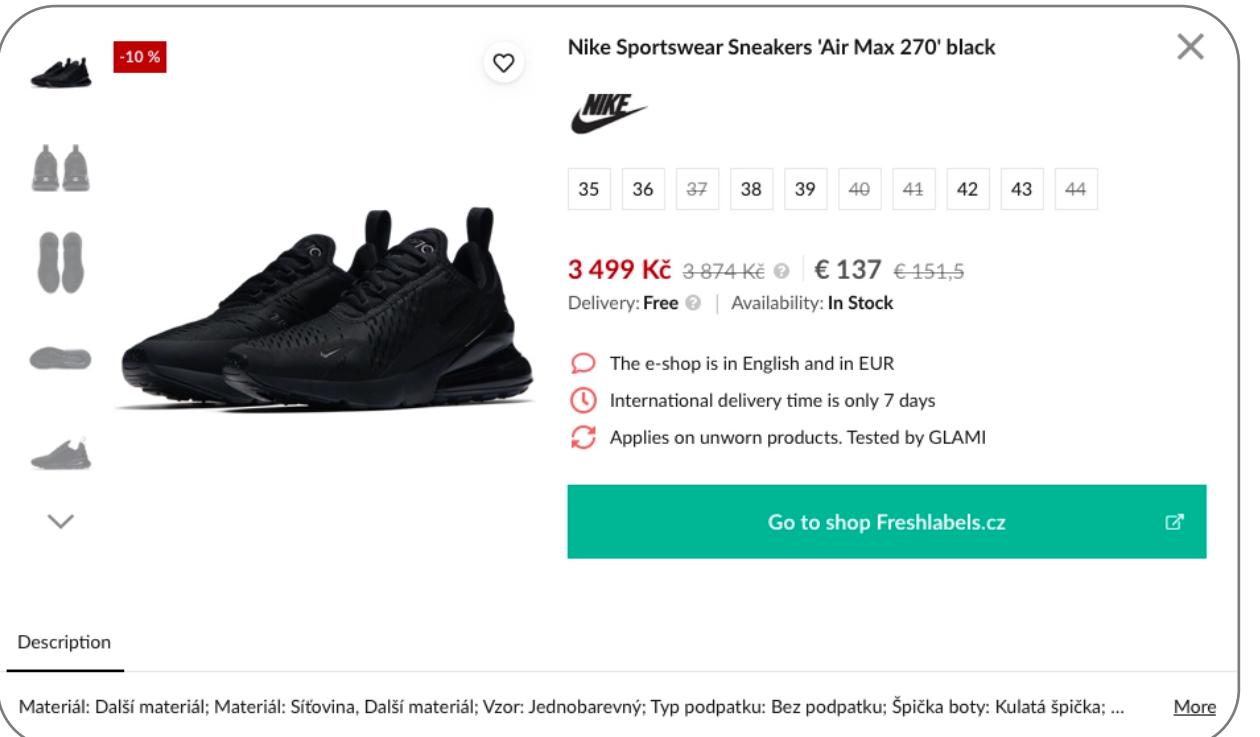
Microservices reduce central points of failure.

# API Gateway



# API Gateway

Web App (client) ต้องการเรียก web service เพื่อนำข้อมูลสินค้าไปแสดง



**Server-side Web App**

Online store ใช้ microservices  
จึงมี services เกี่ยวกับข้อมูลสินค้าอยู่ 3 ตัวแยกกัน

**Product Info service**

**Recommendation service**

**Review service**

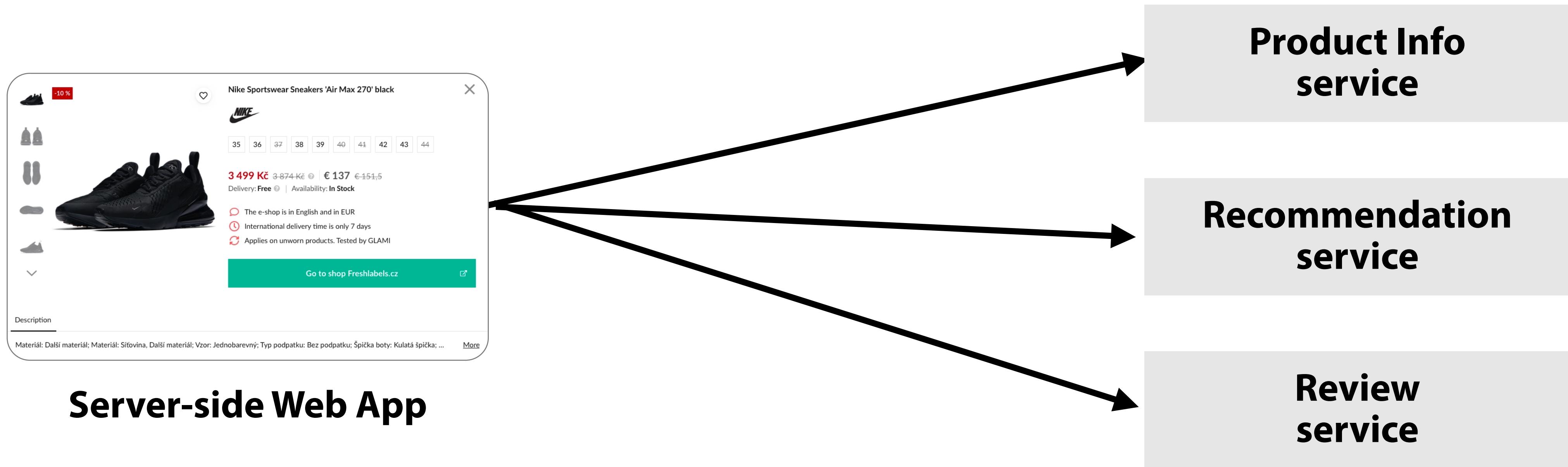
**Online Store Services**

mobile app จะต้องเรียก web service อย่างไร?

# API Gateway

Web App (client) ต้องการเรียก web service เพื่อนำข้อมูลสินค้าไปแสดง

Online store ใช้ microservices จึงมี services เกี่ยวกับข้อมูลสินค้าอยู่ 3 ตัวแยกกัน

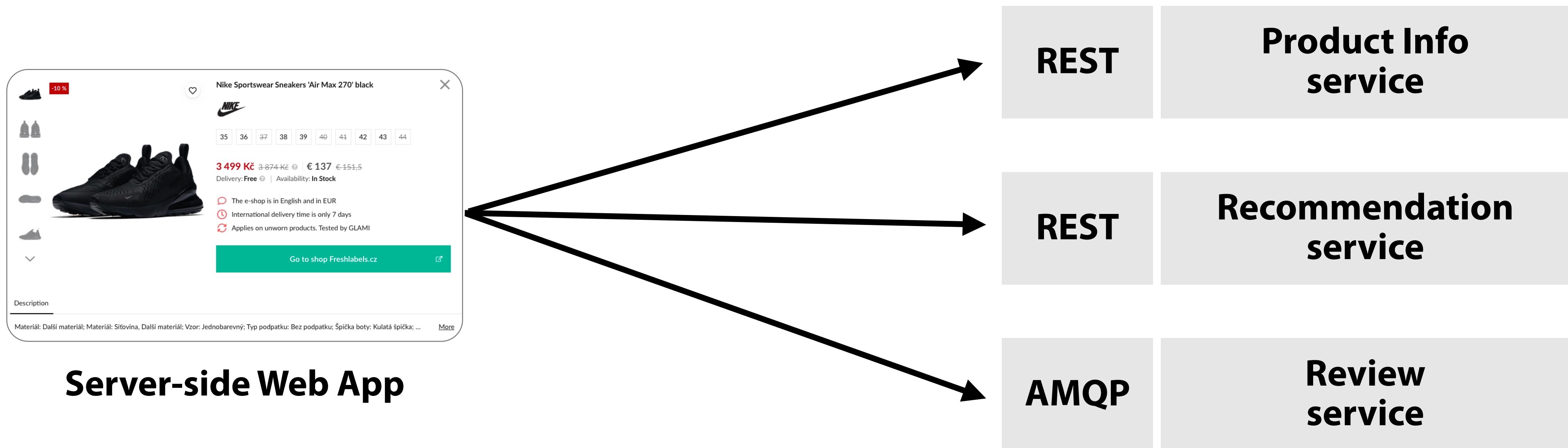


ก็เรียกแยกทีละตัวเลย ดีไหม?

# API Gateway

Web App (client) ต้องการเรียก web service เพื่อนำข้อมูลสินค้าไปแสดง

Online store ใช้ microservices  
จึงมี services เกี่ยวกับข้อมูลสินค้าอยู่ 3 ตัวแยกกัน



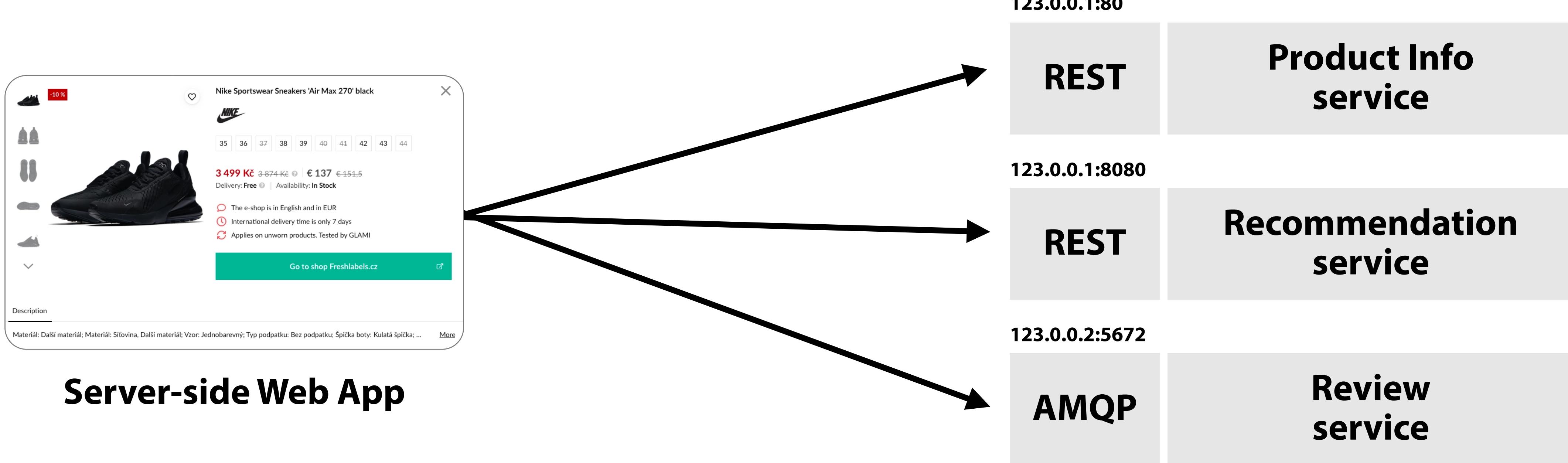
## Online Store Services

แล้วถ้าหากว่าแต่ละ service ใช้ protocol ต่างกันล่ะ?  
ลำพัง REST คงไม่เป็นไร เพราะไดร์ ๆ ก็ใช้กัน แต่ถ้าเป็นตัวอื่น ๆ ก็อาจจะลำบากหน่อยนะ

# API Gateway

Web App (client) ต้องการเรียก web service เพื่อนำข้อมูลสินค้าไปแสดง

Online store ใช้ microservices  
จึงมี services เกี่ยวกับข้อมูลสินค้าอยู่ 3 ตัวแยกกัน



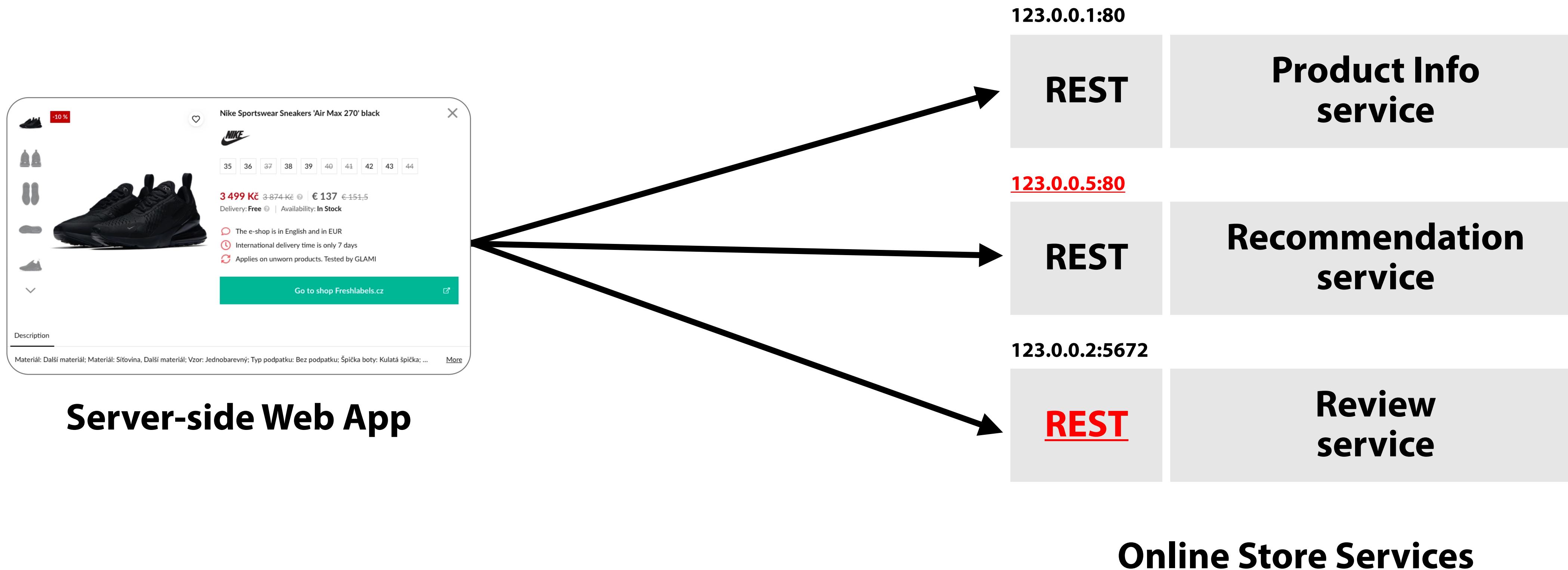
**Server-side Web App**

แล้วถ้าหากว่าแต่ละ service อยู่บน server คนละตัว ใช้ IP address และเลข port ไม่เหมือนกันล่ะ?  
Mobile app ต้องจำ IP address และเลข port ของแต่ละ service เอาไว้

# API Gateway

Web App (client) ต้องการเรียก web service เพื่อนำข้อมูลสินค้าไปแสดง

Online store ใช้ microservices  
จึงมี services เกี่ยวกับข้อมูลสินค้าอยู่ 3 ตัวแยกกัน

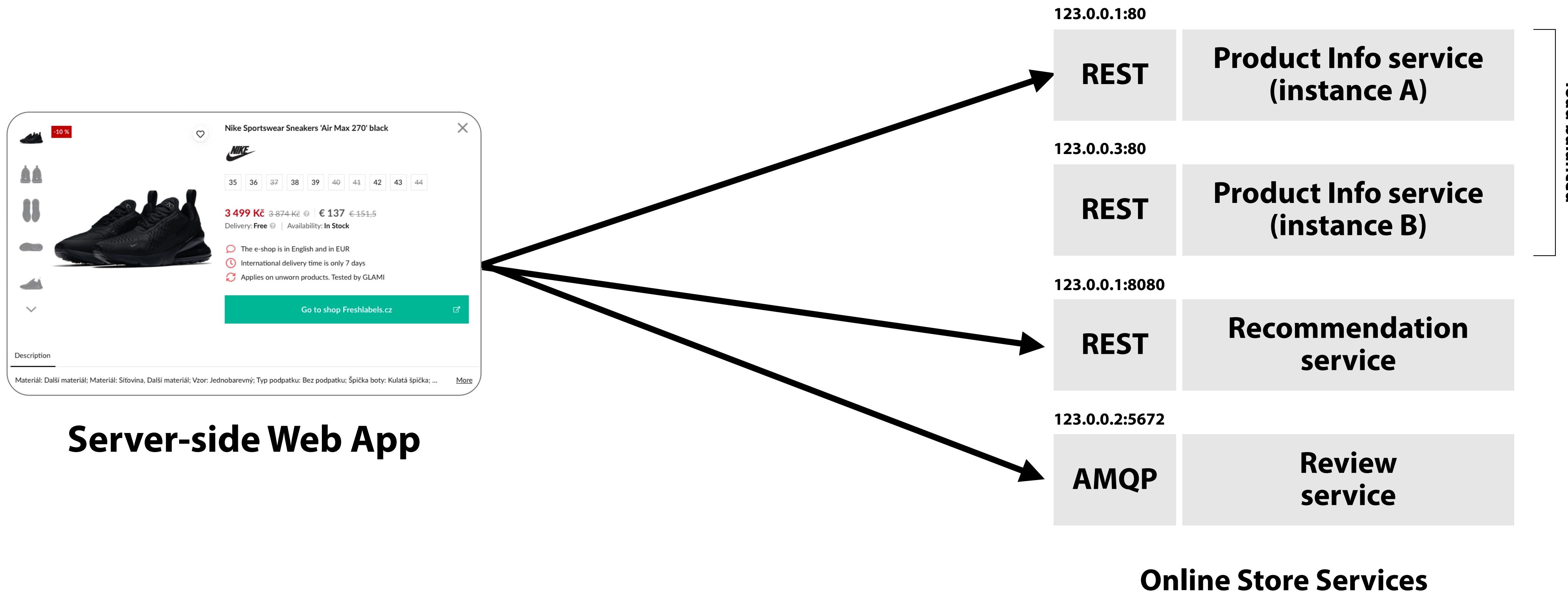


แล้ววันเดี๋นเดี๋นถ้าเกิดเราเปลี่ยน IP address หรือเลข port หรือ protocol ล่ะ?  
แล้วต้องมาตามแก้ใน mobile app อีก แล้วอัพเดตขึ้น app store ใหม่

# API Gateway

Web App (client) ต้องการเรียก web service เพื่อนำข้อมูลสินค้าไปแสดง

Online store ใช้ microservices  
จึงมี services เกี่ยวกับข้อมูลสินค้าอยู่ 3 (+1) ตัวแยกกัน

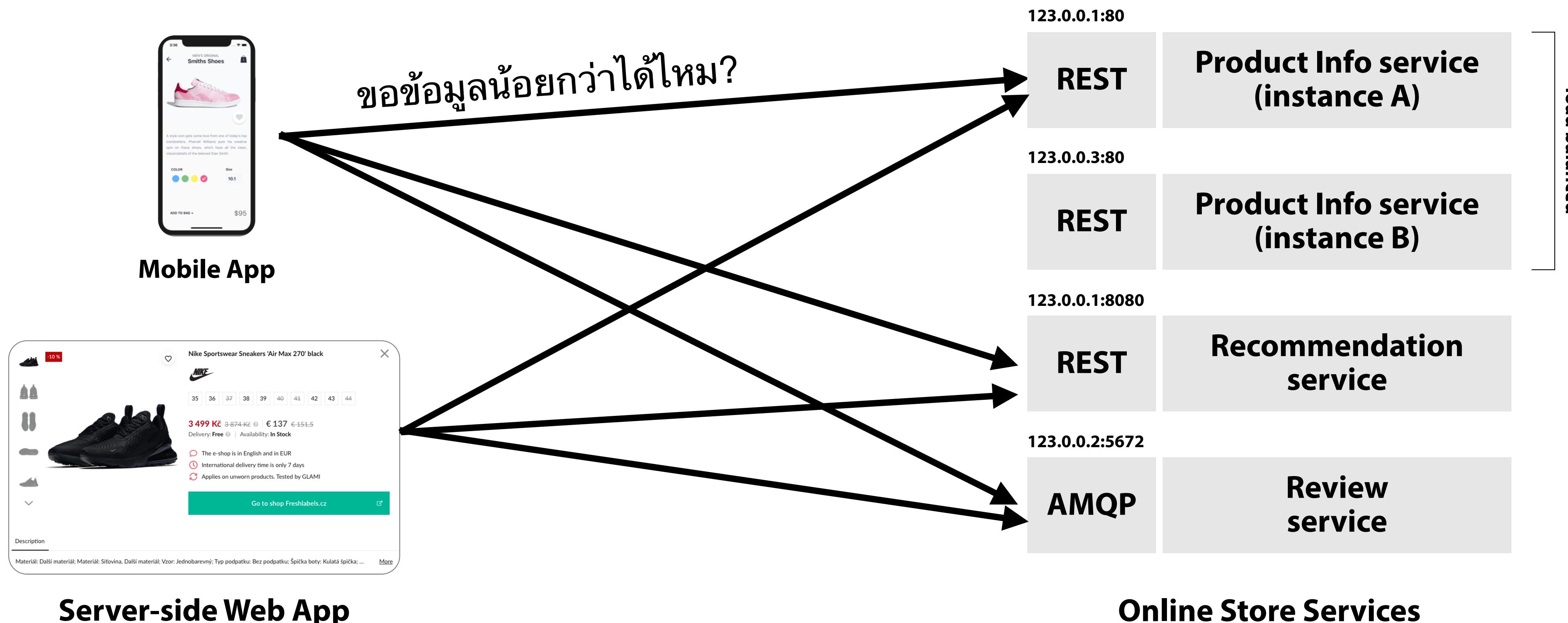


ยังไม่พอ เขายังไปทำ load balancing เพิ่มด้วย ทำให้ service เดียว มีมากกว่า 1 instance  
Mobile app ก็ไม่รู้ด้วย ก็เรียกที่ instance อันเดิมอันเดียวต่อไป

# API Gateway

client ต้องการเรียก web service เพื่อนำข้อมูลสินค้าไปแสดง

Online store ใช้ microservices  
จึงมี services เกี่ยวกับข้อมูลสินค้าอยู่ 3 (+1) ตัวแยกกัน



ต่อมา มีทำ mobile app ขึ้นมาด้วย ก็เป็น client เหมือนกัน  
แต่เนื้ตมันไม่ค่อยแรง อยากจะขอแสดงข้อมูลน้อยกว่า web app ได้ไหมล่ะ?

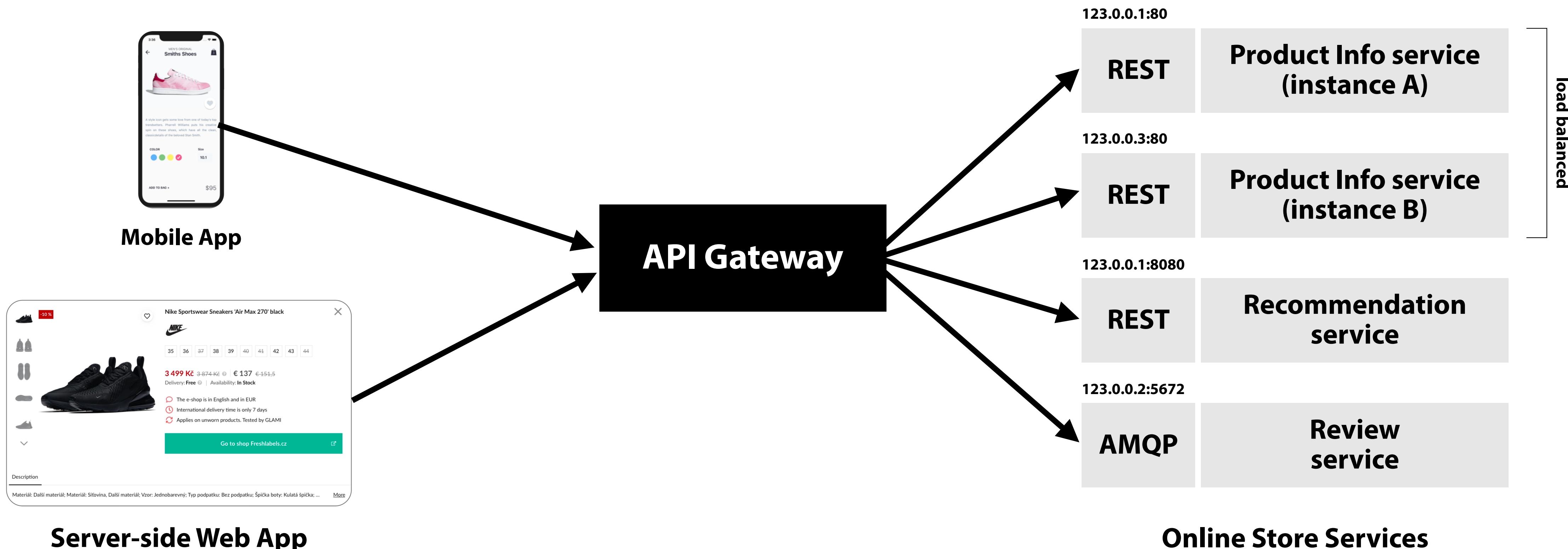
A young Black man with short hair, wearing a purple hoodie, is looking directly at the camera with a neutral expression. He is holding a small, round, red button with a black smiley face graphic in his hands, which are held out towards the viewer. The background is plain white.

API Gateway

# API Gateway

client ต้องการเรียก web service เพื่อนำข้อมูลสินค้าไปแสดง

Online store ใช้ microservices  
จึงมี services เกี่ยวกับข้อมูลสินค้าอยู่ 3 (+1) ตัวแยกกัน

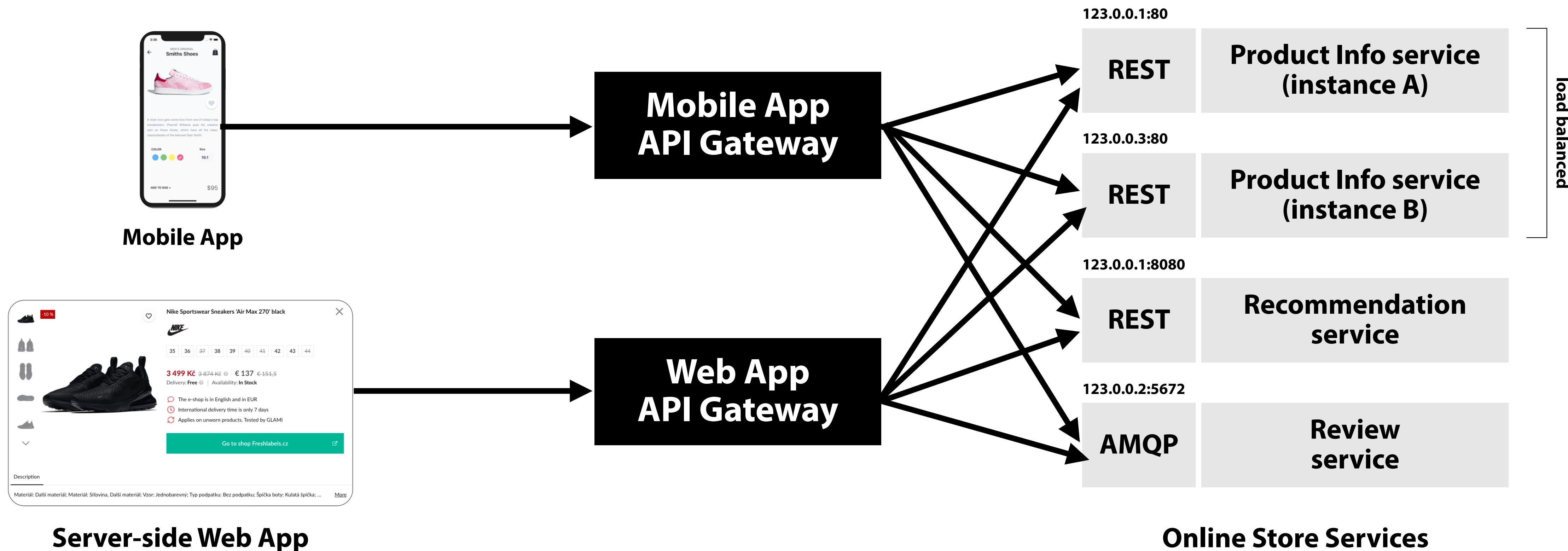


API Gateway ทำให้เกิด single entry point อย่างได้อะไรเรียกว่าที่ API Gateway ได้เลย ถ้ามีอะไรมาเปลี่ยนแปลงที่ฟัง web service คนที่จำเป็นต้องปรับเปลี่ยนก็มีแค่ API Gateway คนเดียว client ไม่ต้องเปลี่ยนตาม

# Backends for Frontends

client ต้องการเรียก web service เพื่อนำข้อมูลสินค้าไปแสดง

Online store ใช้ microservices จึงมี services เกี่ยวกับข้อมูลสินค้าอยู่ 3 (+1) ตัวแยกกัน



Backends for Frontends เป็นรูปแบบที่มี API Gateway มากกว่า 1 ตัว แยกสำหรับ client แต่ละประเภท

# ข้อดีของ API Gateway

- ทำให้ client ไม่จำเป็นต้องรู้รายละเอียดของ microservices ว่ามีการจัดโครงสร้างภายในอย่างไร
- ทำให้ client ไม่ต้องจดจำที่อยู่ (IP address และ port) ของแต่ละ service
- สามารถทำ API ให้เหมาะสมกับ client แต่ละประเภทได้
- ทำให้จำนวน request ที่ client ต้องเรียก web service ลดน้อยลง
- ลดความยุ่งยากผึ้ง client นำไปให้ API Gateway จัดการแทน
- API Gateway สามารถเปิด API ให้ client เรียกด้วย protocol ที่เป็น standard และเหมาะสมกับการสื่อสารผ่านเว็บ (web-friendly) มากกว่า เพื่อเรียกต่อไปยัง service ซึ่งใช้ protocol อะไรก็ได้ภายใน

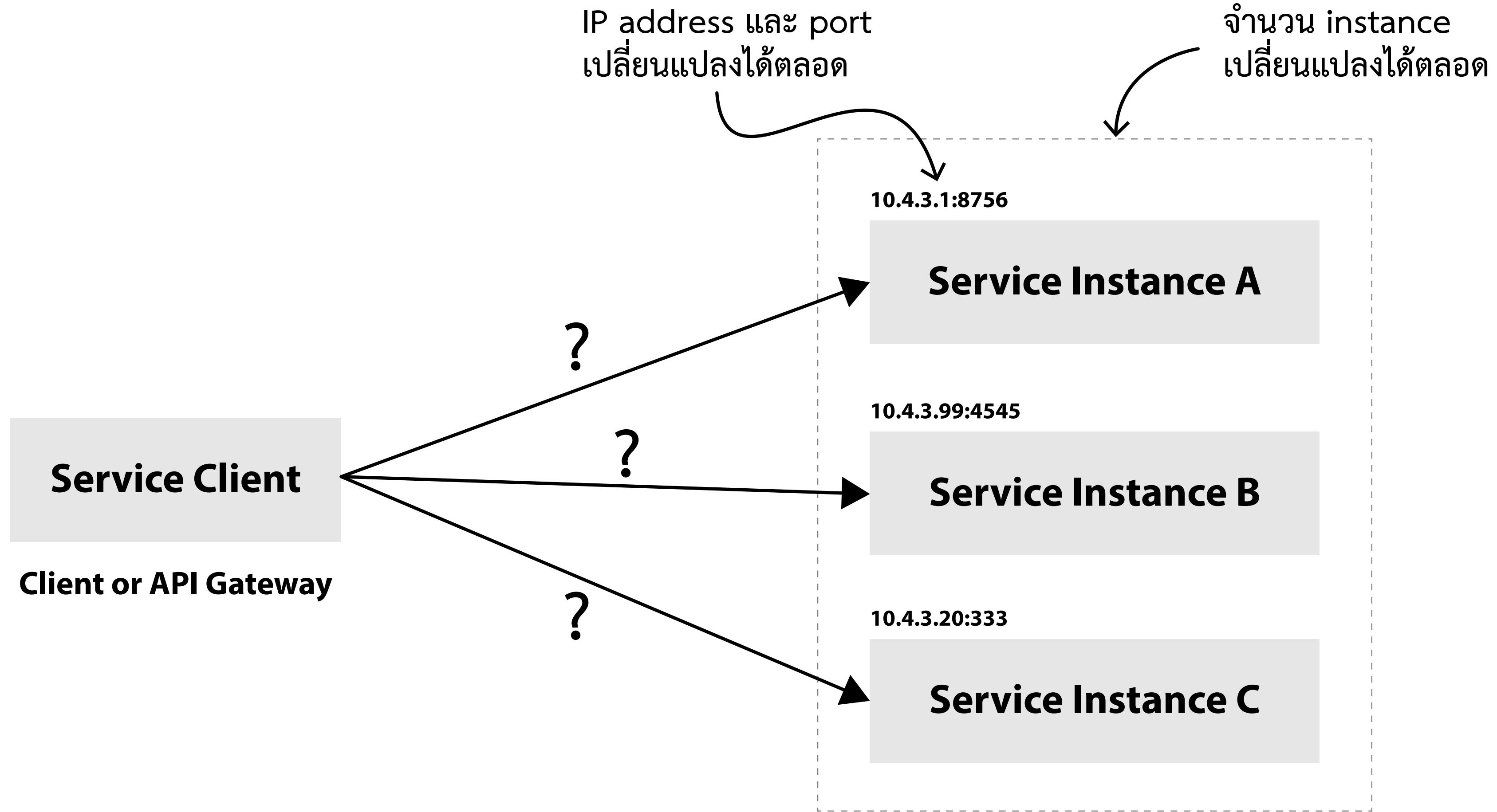
# ข้อเสียของ API Gateway

- เพิ่มความซับซ้อนในการพัฒนา เพราะ API Gateway ก็เป็นส่วนหนึ่งที่สามารถเปลี่ยนแปลงได้เรื่อย ๆ และต้องมีการพัฒนาและดูแล
- response time เพิ่มมากขึ้น เนื่องจากจำเป็นต้องเรียกไปที่ API Gateway ก่อน แล้ว API Gateway ก็จะไปเรียก service ต่ออีกที (แต่ด้วยประสิทธิภาพของเทคโนโลยีปัจจุบันก็ทำให้ข้อเสียข้อนี้ลดน้อยลงเรื่อย ๆ)

# Service Discovery



# Service Discovery

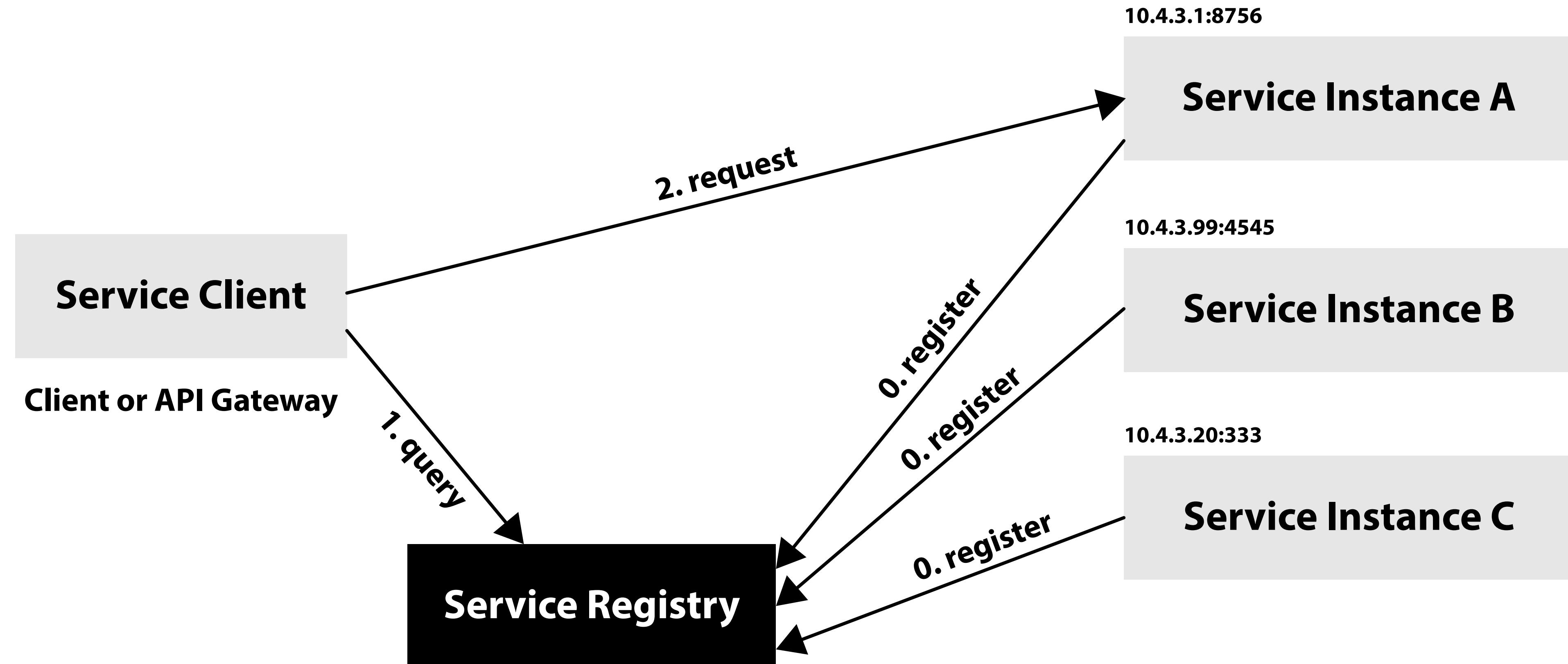


ถ้า service มีการเปลี่ยนแปลง เช่น จำนวน instance เปลี่ยน หรือ IP address และเลข port เปลี่ยน client หรือ API Gateway หรือ service อื่น ๆ จะรู้ได้อย่างไร

A medium shot of a young Black man with short hair, wearing a purple hoodie. He is looking directly at the camera with a neutral expression. His hands are held out in front of him, palms facing up, holding a small, round, red button. The button has a black graphic of three dots arranged in a triangle pattern. The background is plain white.

Service Registry

# Client-Side Service Discovery



client หรือ API gateway ถาม service registry ว่า service ที่ต้องการนั้นอยู่ที่ IP address และ port อะไร แล้วจึง request ไปที่ IP address และ port นั้น

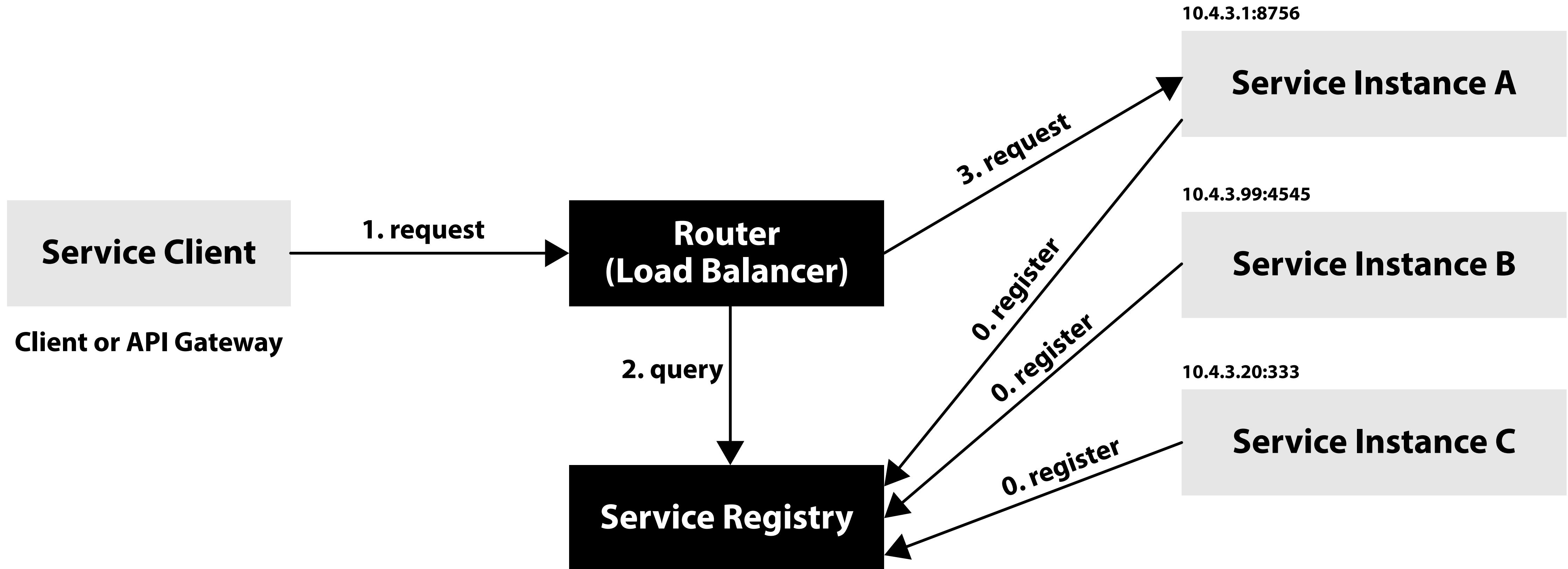
# ข้อดีของ Client-Side Service Discovery

- network hop น้อยกว่า server-side service discovery

# ข้อเสียของ Client-Side Service Discovery

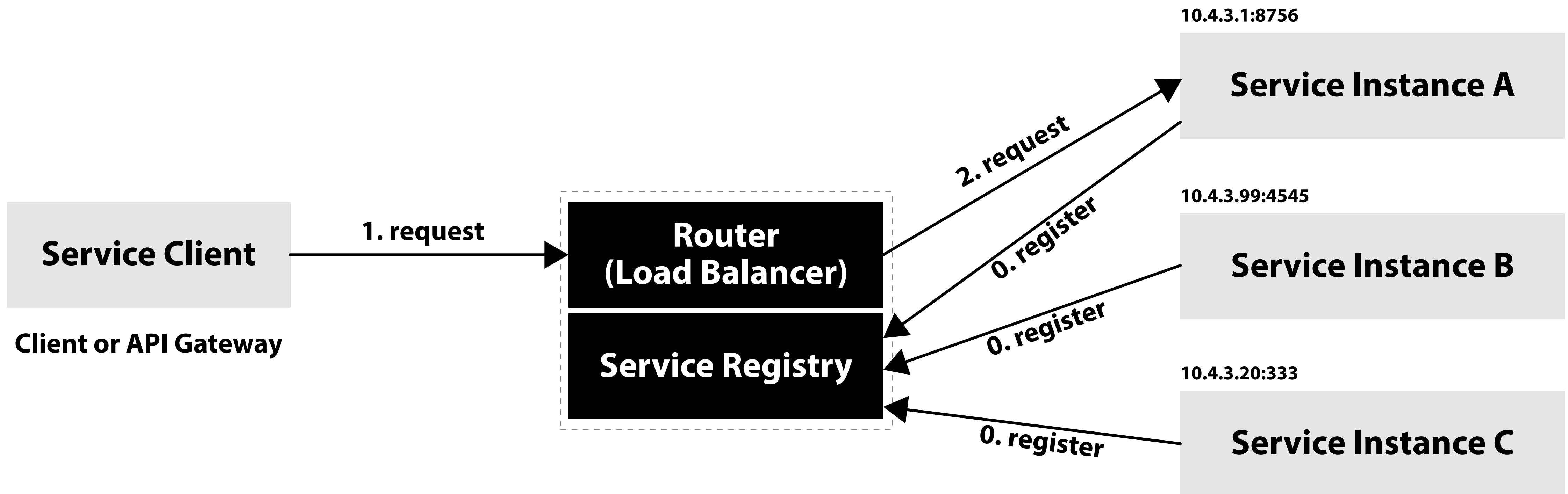
- เกิด coupling ระหว่าง client กับ service registry

# Server-Side Service Discovery



client หรือ API gateway ส่ง request ไปที่ router (load balancer)  
จากนั้น router จะไปถาม service registry ว่า service ที่ต้องการอยู่ที่ IP address และ port อะไร  
router ก็จะส่ง request ต่อไปที่ IP address และ port นั้น

# Server-Side Service Discovery



หรือ router อาจจะมี service registry อัญในตัวเลยก็ได้

# ข้อดีของ Server-Side Service Discovery

- code ฝั่ง client จะไม่ยุ่งยากเมื่อเทียบกับ client-side service discovery เพราะในกรณีนี้ client แค่ส่ง request มาที่ router เฉย ๆ ไม่ต้องห่วงว่า service อยู่ที่ไหน
- cloud environment บางตัวก็มีฟังก์ชันนี้มาให้อยู่แล้ว เช่น AWS Elastic Load Balancer

# ข้อเสียของ Server-Side Service Discovery

- ถ้า cloud environment ไม่ได้มี router มาให้อยู่แล้ว เราต้องสร้างหรือจัดหามาติดตั้ง และก็ต้องดูเรื่องการทำ replication เพื่อรับรองรับ traffic load ด้วย
- ถ้า router ไม่ได้เป็น TCP-based ก็จะต้องรองรับ protocol ที่จำเป็นด้วย เช่น HTTP, gRPC, Thrift เป็นต้น
- network hop จะมากกว่าเมื่อเทียบกับ client-side service discovery

# Service Registry

คือ database ที่เก็บข้อมูลเกี่ยวกับ service แต่ละตัวในระบบว่ามี instance อะไรบ้าง อยู่ที่ IP address และ port อะไร

เมื่อ service แต่ละตัวถูกเปิดทำงาน (startup) ก็จะมา register กับ service registry เพื่อบอกว่าตัวเองพร้อมรับ request และเมื่อถูกปิดการทำงาน (shutdown) ก็จะมา unregister กับ service registry เพื่อบอกว่าตัวเองไม่พร้อมรับ request แล้ว

service registry อาจจะมีการเรียก health check API ที่อยู่บน instance ของ service แต่ละตัวเป็นระยะ ๆ เพื่อตรวจดูว่า service ยังพร้อมรับ request อยู่หรือไม่

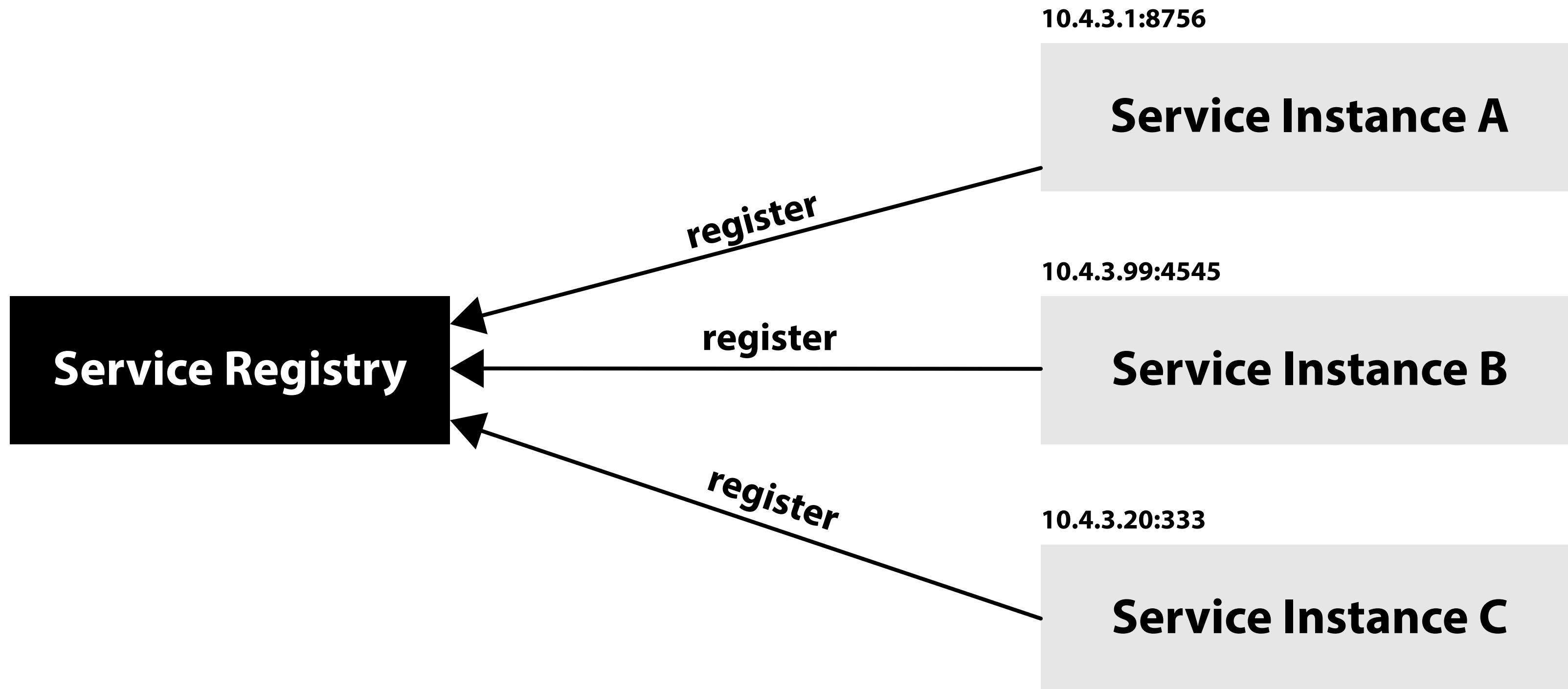
# ข้อดีของ Service Registry

- client และ/หรือ router สามารถค้นหาตำแหน่งของ service ได้

# ข้อเสียของ Service Registry

- ถ้า infrastructure ไม่ได้มี service registry มาให้อยู่แล้ว เราต้องสร้างหรือจัดทำติดตั้ง และก็ต้องดูแลเพื่อรับ traffic load ด้วย
- ถ้า Service registry ล่ม client จะเข้าถึง service ต่าง ๆ ไม่ได้ และถึงแม้ว่าถ้ามีการทำ caching ข้อมูลของ service registry เอาไว้ด้วย ถ้า service registry ล่ม client ก็ยังคงเข้าถึง service ต่าง ๆ ได้อยู่ แต่ข้อมูลอาจไม่เป็นปัจจุบันแล้ว (out of date)

# Self Registration



service จะทำการ register ตัวเองกับ service registry ตอน startup  
และก็จะ renew registration เป็นระยะ ๆ เพื่อบอกว่าตัวเองยังพร้อมรับ request อญ  
และตอน shutdown ก็จะ unregister ตัวเองออกจาก service registry

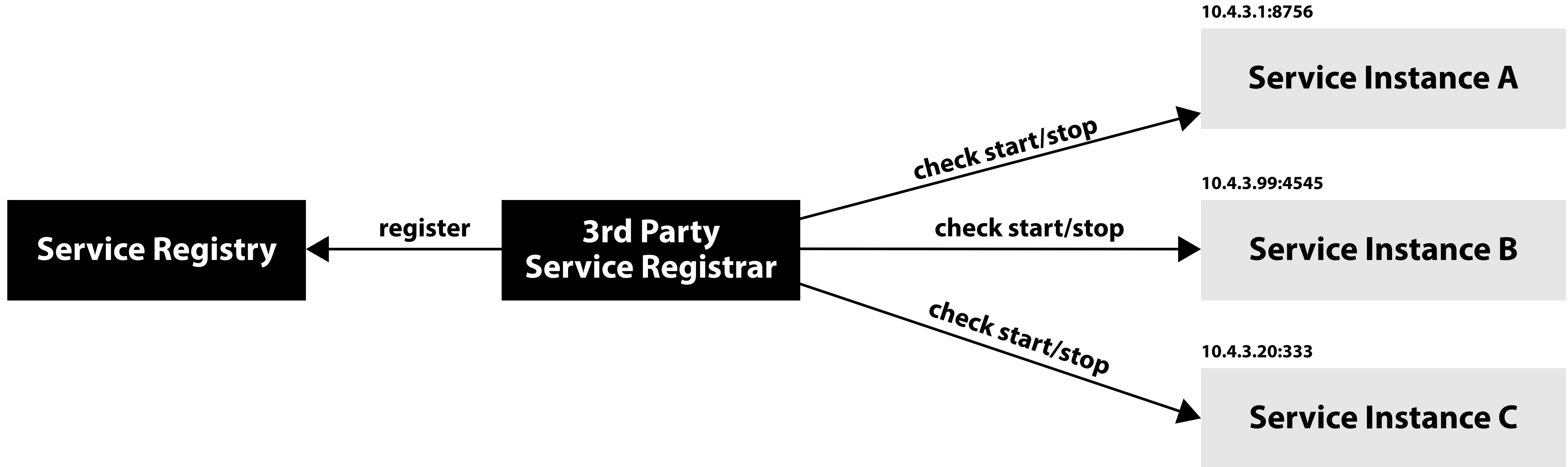
# ข้อดีของ Self Registration

- เนื่องจากว่า service instance เป็นคน register ตัวเองกับทาง service registry หมายความว่าเราสามารถเก็บข้อมูลสถานะอื่น ๆ เพิ่มเติมที่มากกว่าแค่ UP กับ DOWN ได้ เช่น STARTING, AVAILABLE เป็นต้น

# ข้อเสียของ Self Registration

- เกิด coupling ขึ้นระหว่าง service กับ service registry
- ในการเขียนโค้ดทำ service registration ก็จะต้องใช้ภาษาเดียวกันกับที่ใช้เขียน service
- service ที่ยัง run อยู่แต่ไม่สามารถรับ request ได้มักจะไม่ไป unregister ตัวเองออกจาก service registry

# 3rd Party Registration



3rd service registrar จะเช็คสถานะ start/stop ของ service  
แล้วไป register / unregister กับ service registry ให้อีกที

# ข้อดีของ 3rd Party Registration

- โค้ดฝั่ง service ก็จะน้อยลง เพราะไม่ต้องทำเรื่อง service registration เองแล้ว
- service registrar สามารถทำ health check ได้เรื่อย ๆ เพื่อตรวจดูว่า service instance ยังทำงานอยู่ไหม

# ข้อเสียของ 3rd Party Registration

- service registrar อาจจะรู้ว่า service ยัง run อยู่หรือไม่ แต่อาจจะไม่รู้ว่า service พร้อมรับ request หรือไม่ (แต่ service registrar บางตัว เช่น Netflix Prana ก็สามารถทำ health check เพื่อตรวจสอบสถานะของ service ได้)
- ถ้า infrastructure ไม่ได้มี service registrar มาให้ เราต้องสร้างหรือจัดหามาติดตั้ง และดูแลเอง
- ถ้าใช้ 3rd party registration ทำ service registration แล้ว service registrar จะเป็นส่วนสำคัญที่เราต้องมั่นใจว่ามันจะสามารถทำงานได้ตลอด