Mục lục thuyết trình

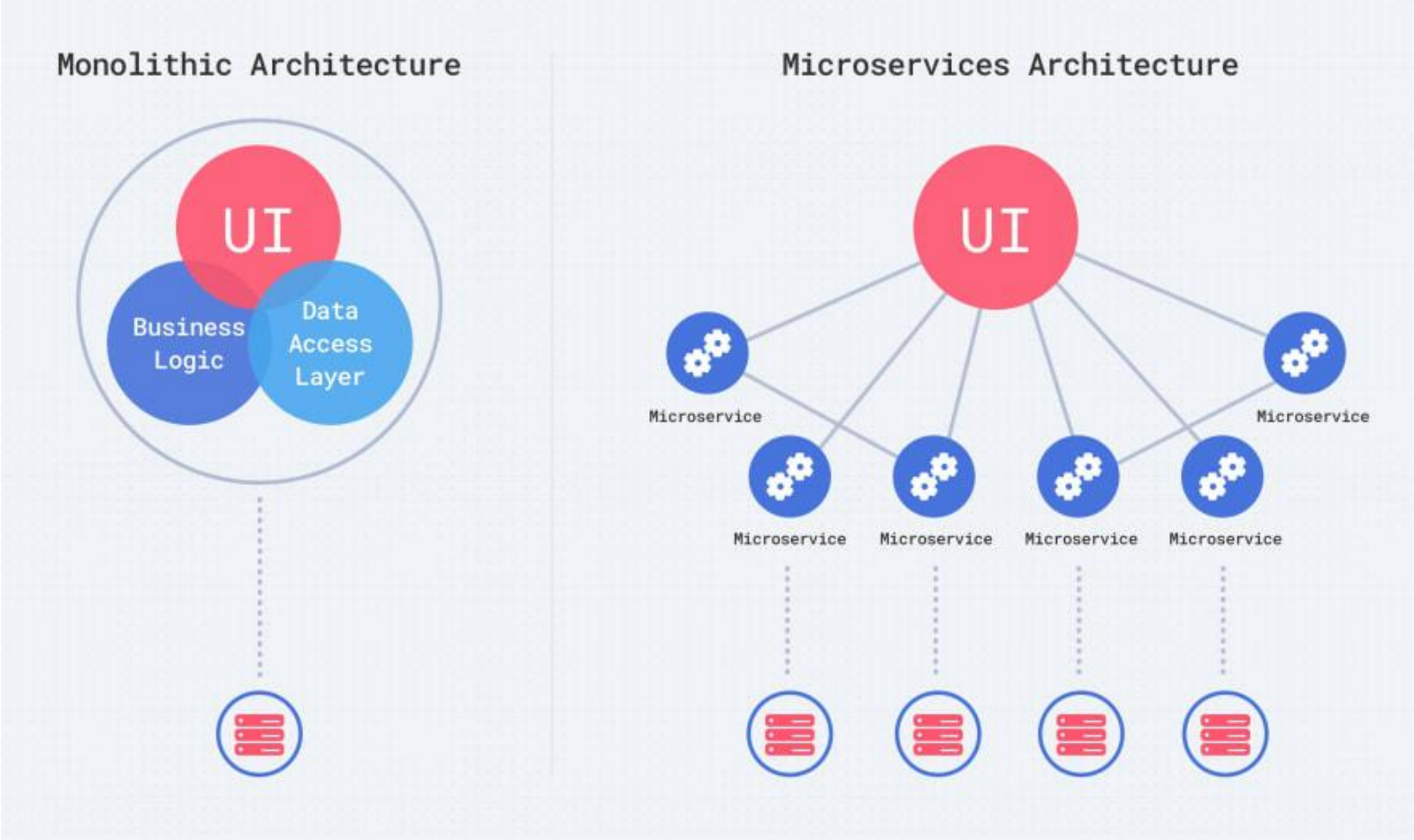
1. Introduction Microservices Architecture
2. Java Microservices with spring cloud
3. DDD Layered Architecture
4. CQRS
5. Event Sourcing
6. Demo application cqrs-es

Microservices Architecture

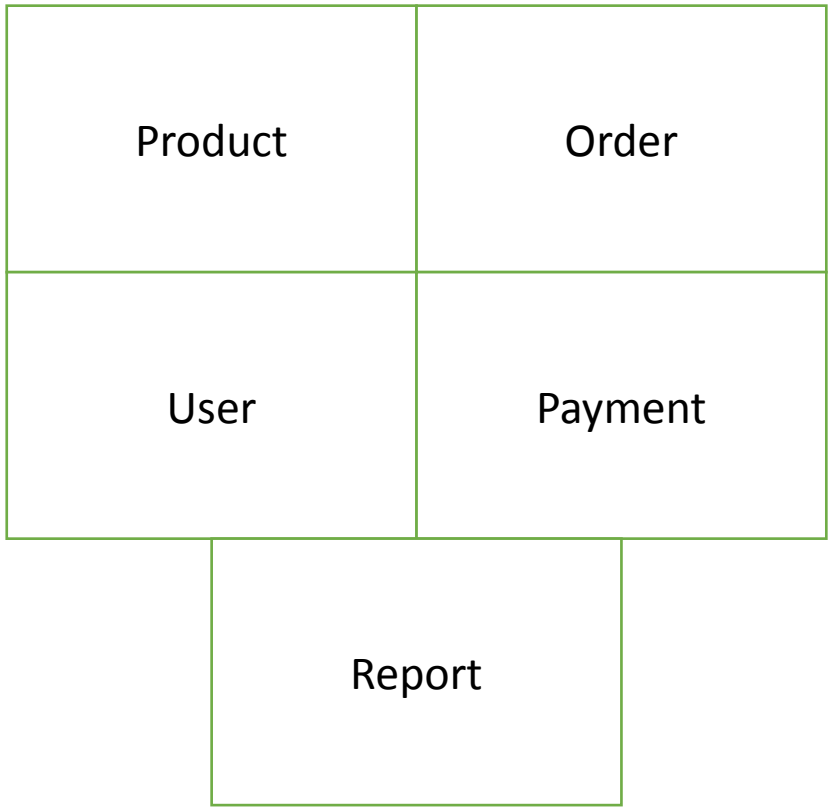
1. Introduction Microservices Architecture
2. Architecture
3. Comunicating between
4. Securing Microservice

Introduction

Micro -> Nhỏ
Service -> Dịchvụ

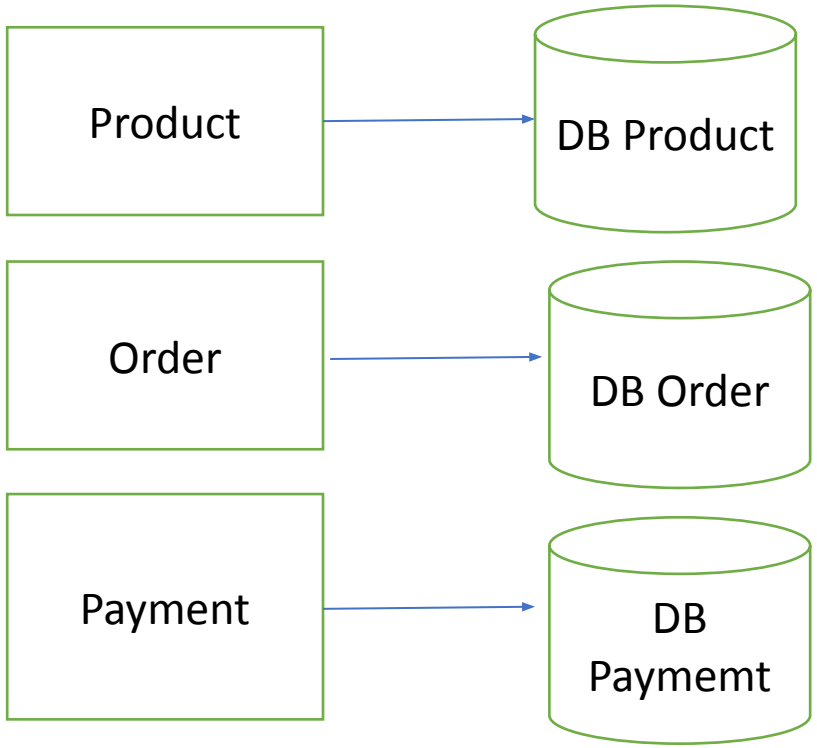


Problem Monolithic



- Scale từng module khiến code phình to
- Thêm và sửa code rất phức tạp
- Deploy khó khăn

Solution



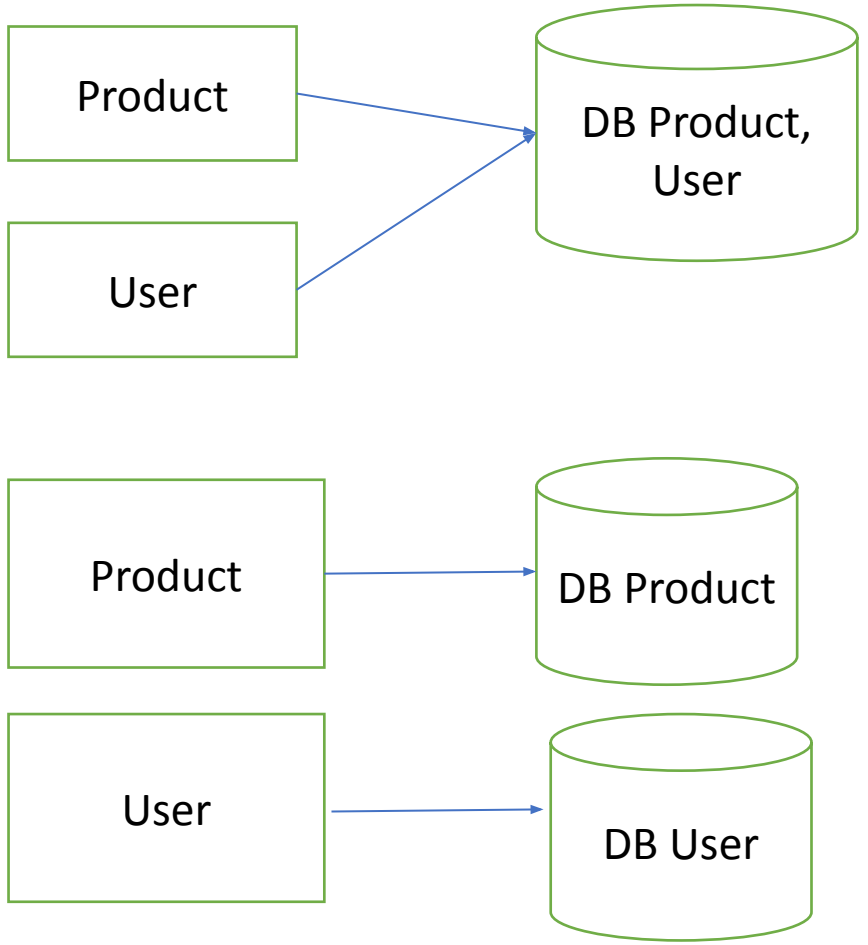
Microservice Architecture

- Ưu điểm :
- Dễ dàng scale tiết kiệm chi phí
- Dễ dàng deploy
- Tự do sử dụng các công nghệ ở mỗi service
- Khó sập hơn monolithic

- Nhược điểm:
- Khó đọc code, debug
- Quản lý khó khan
- Cần đội ngũ team chuyên nghiệp

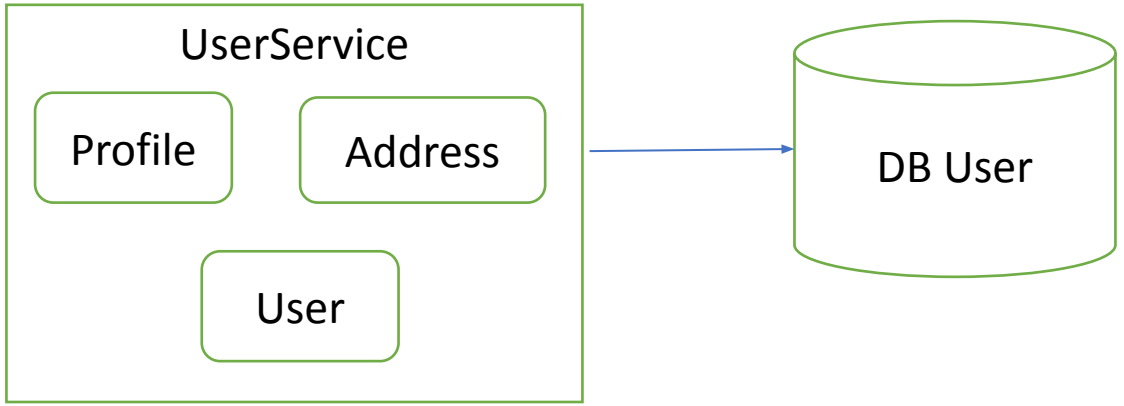
Architecting Microservices

- Mỗi microservice đều phải có cơ sở dữ liệu riêng
- Tính độc lập



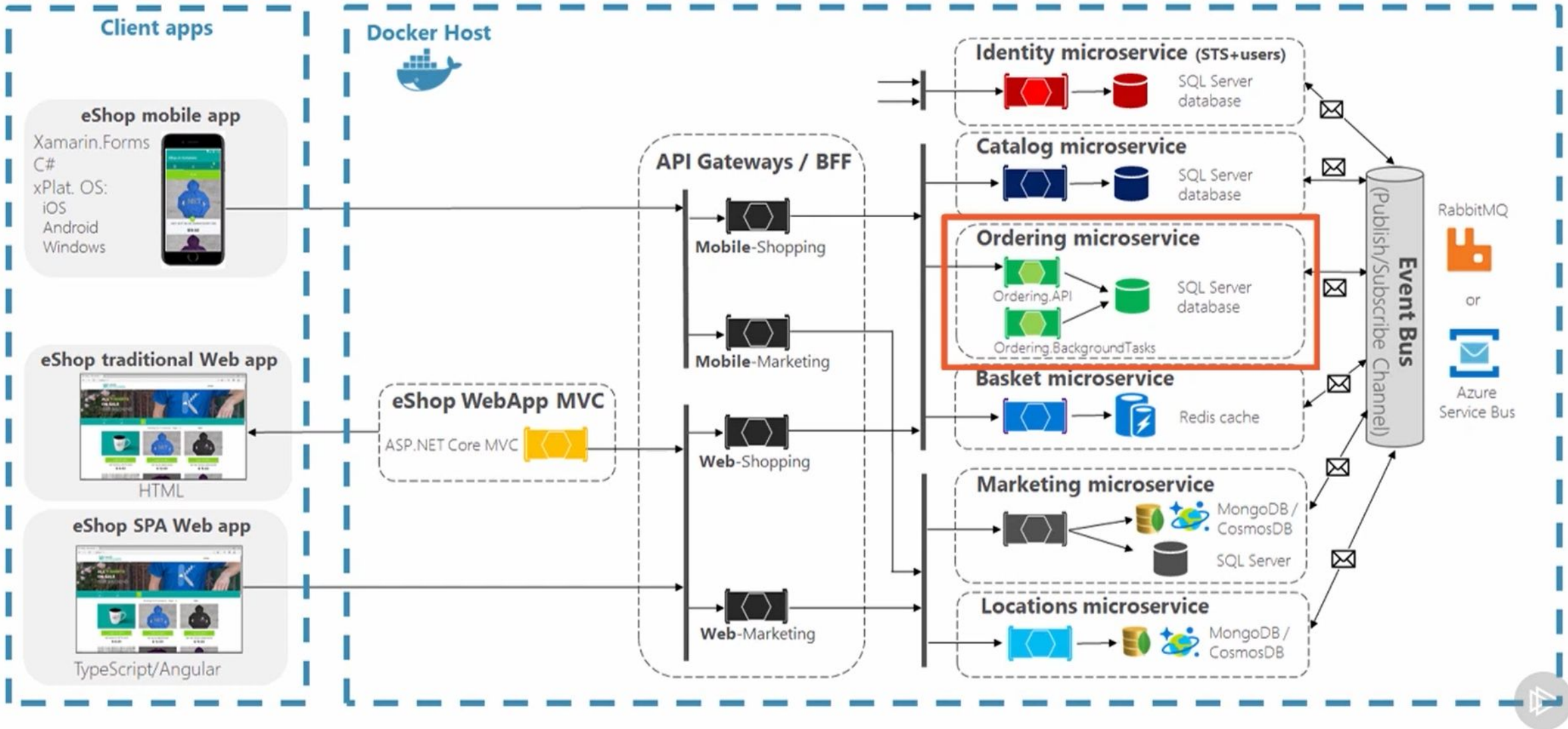
Architecting Microservices

- Xác định ranh giới Service
- Giảm thiểu việc gọi request ở nhiều nơi
- Quản lý code dễ dàng

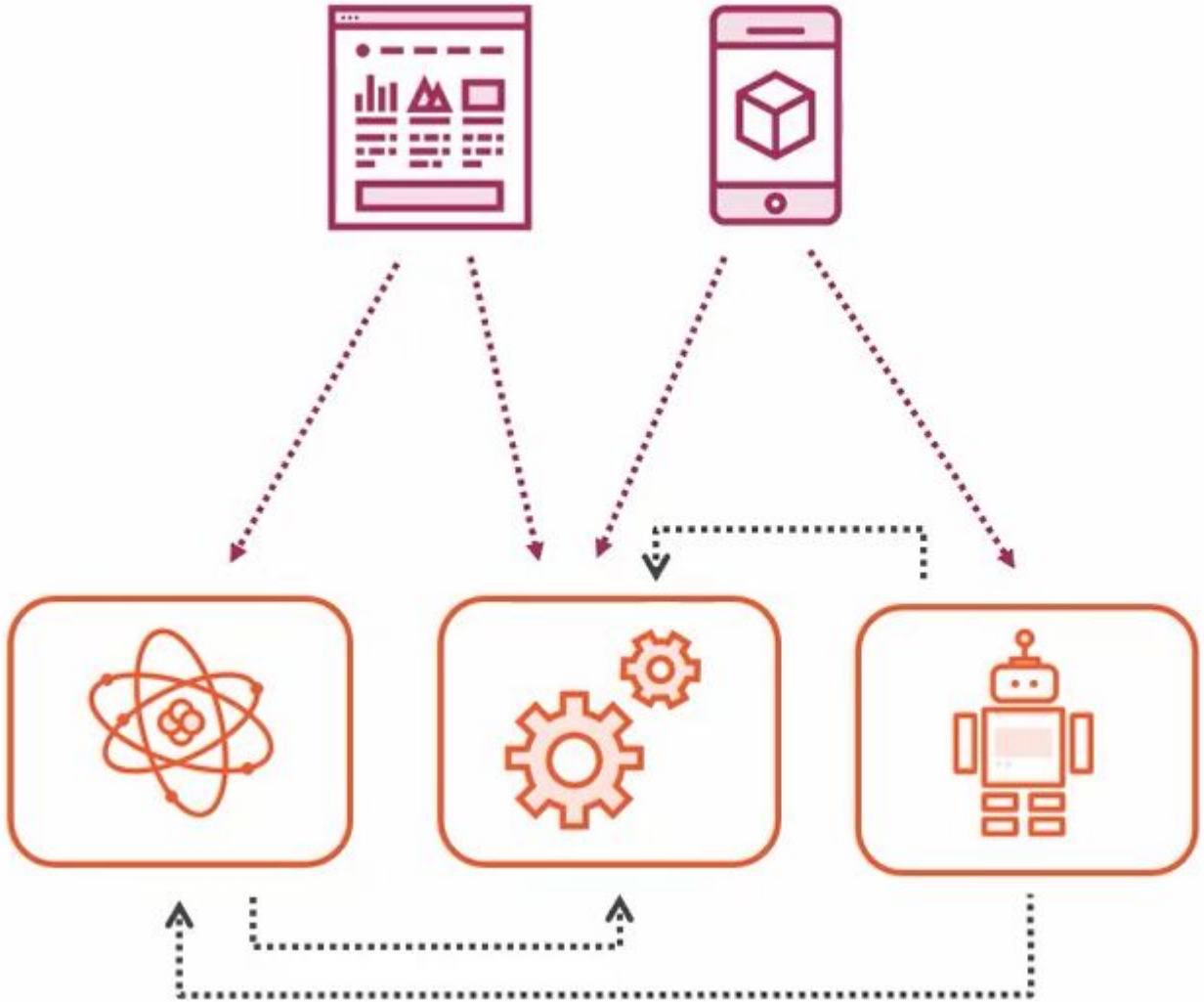


Architecting Microservices

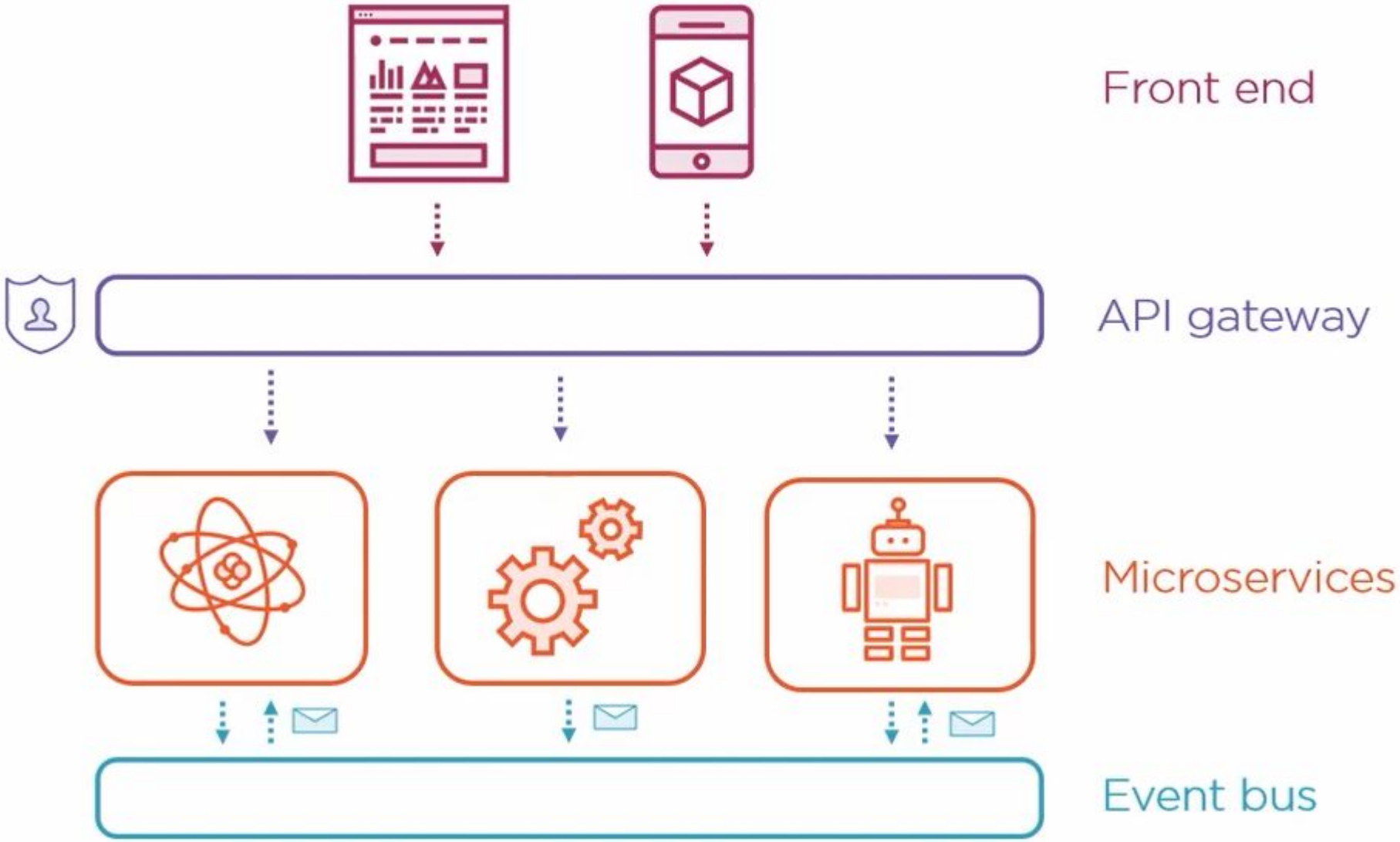
eShopOnContainers Architecture



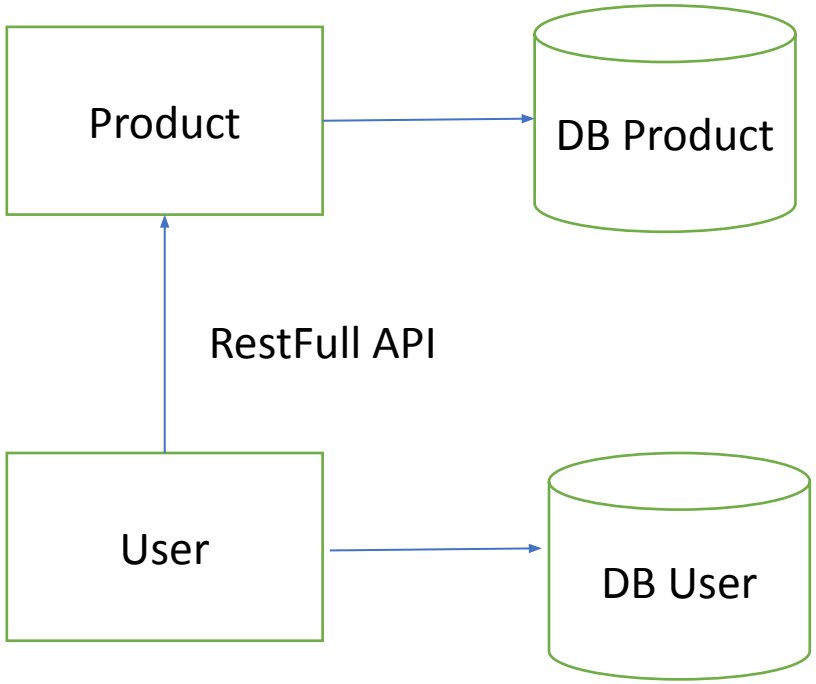
Comunication Microservices

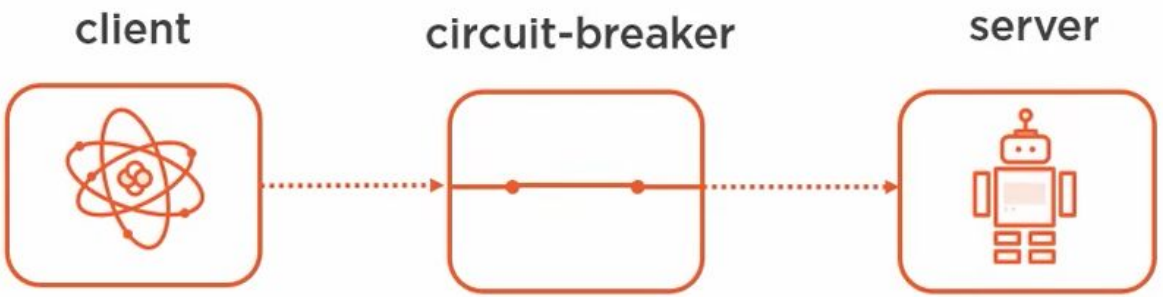


Comunication Microservices

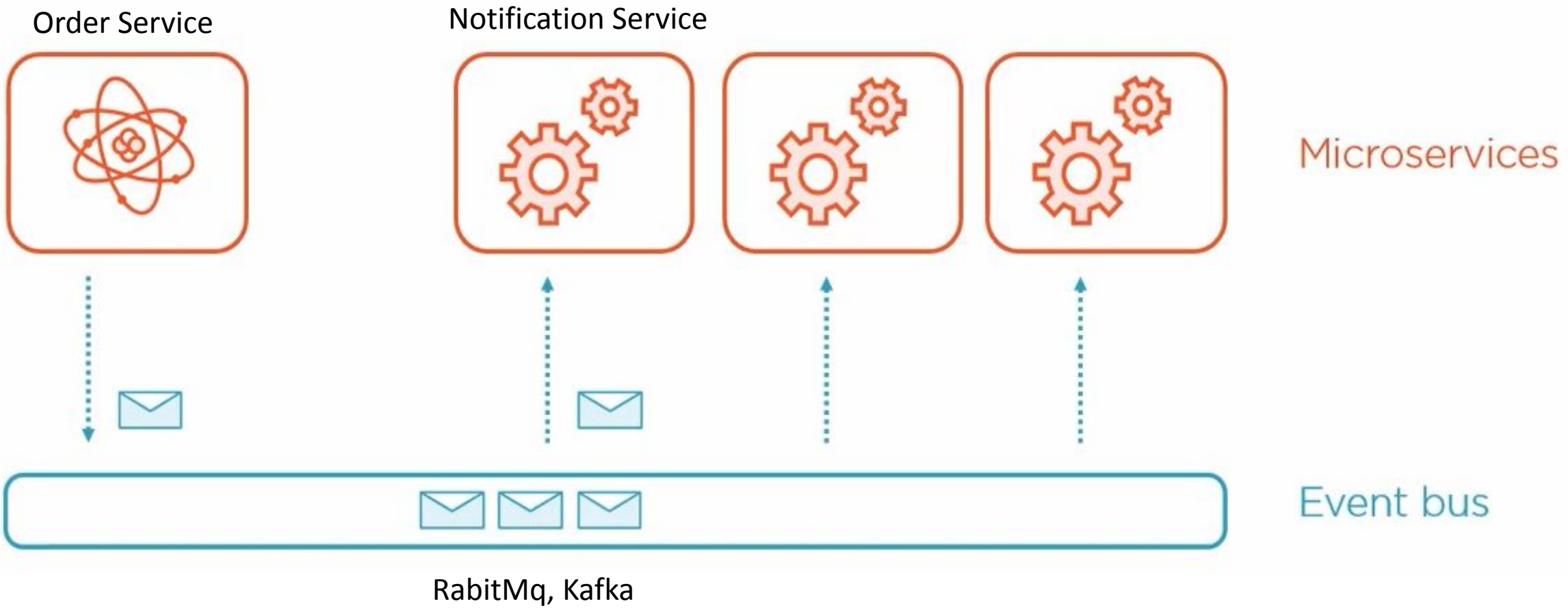


Synchronous Comunication

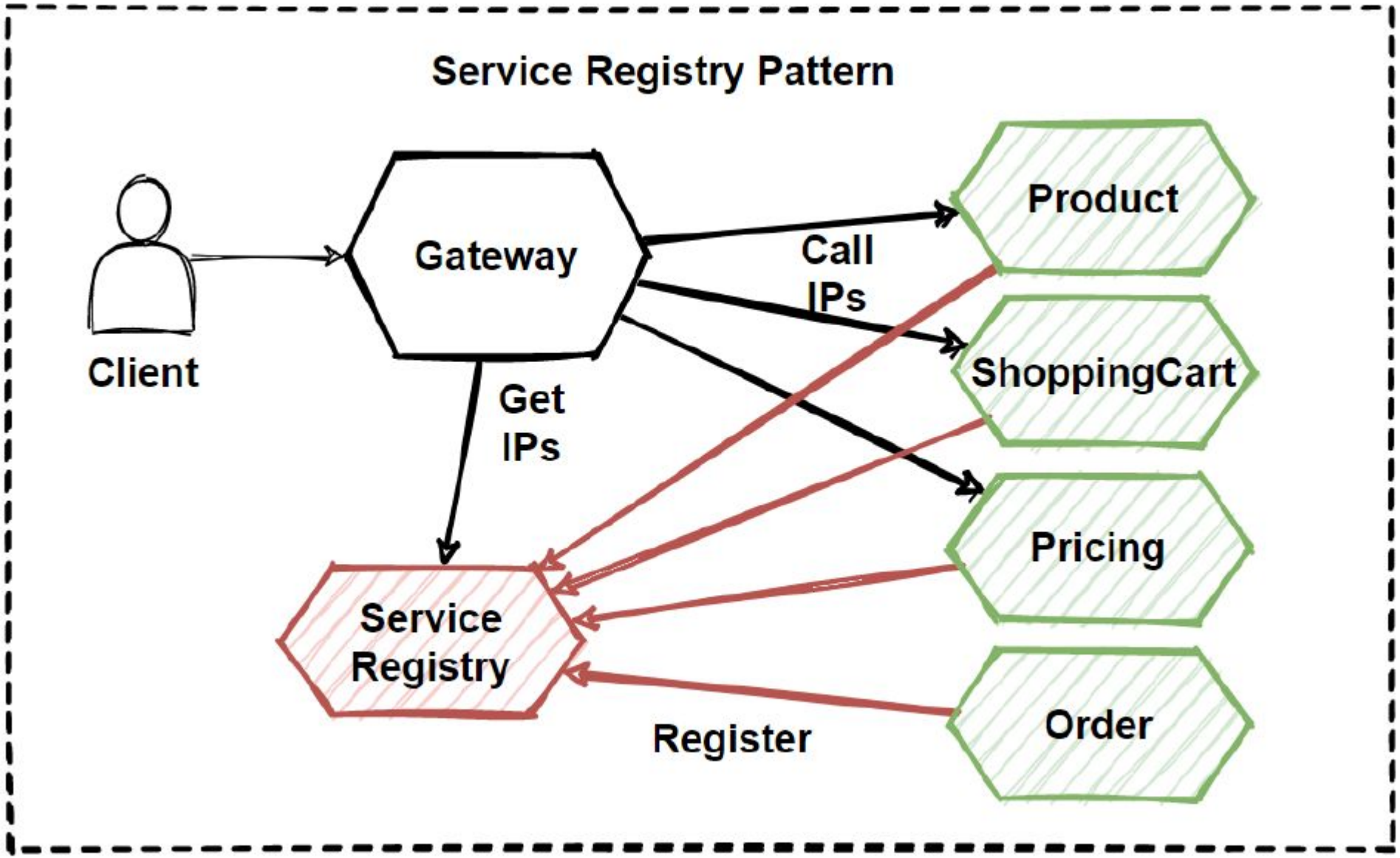




Asynchronous Comunication

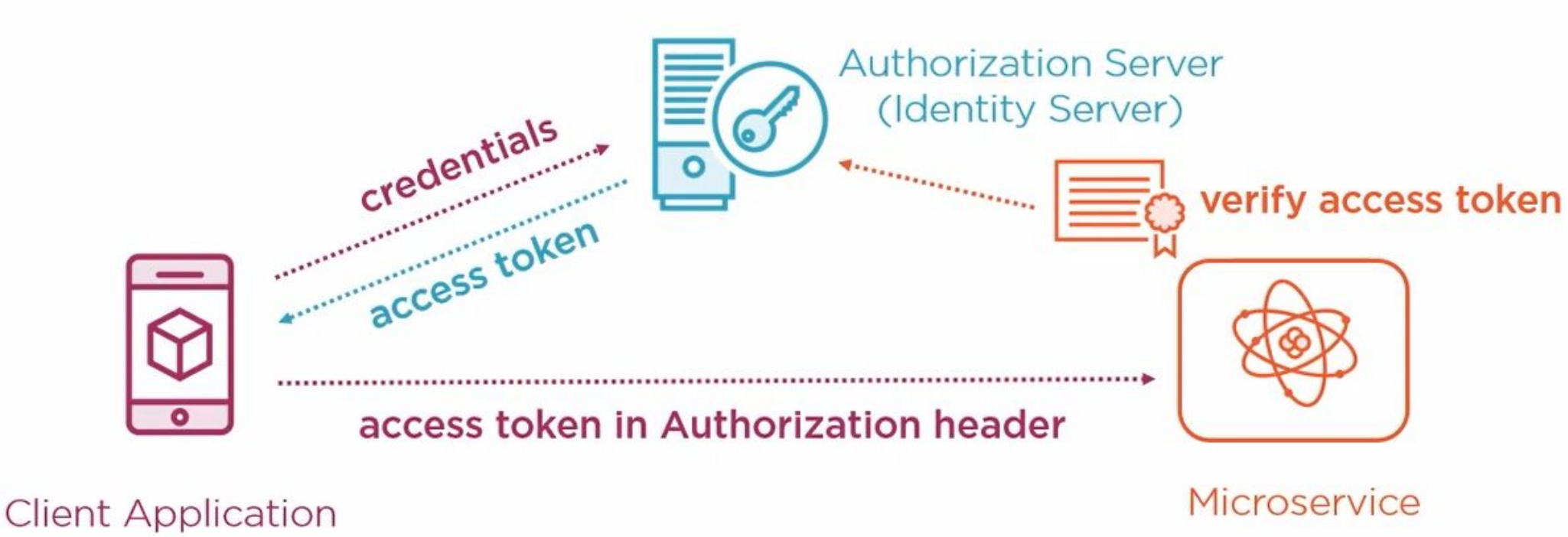


Service Discovery

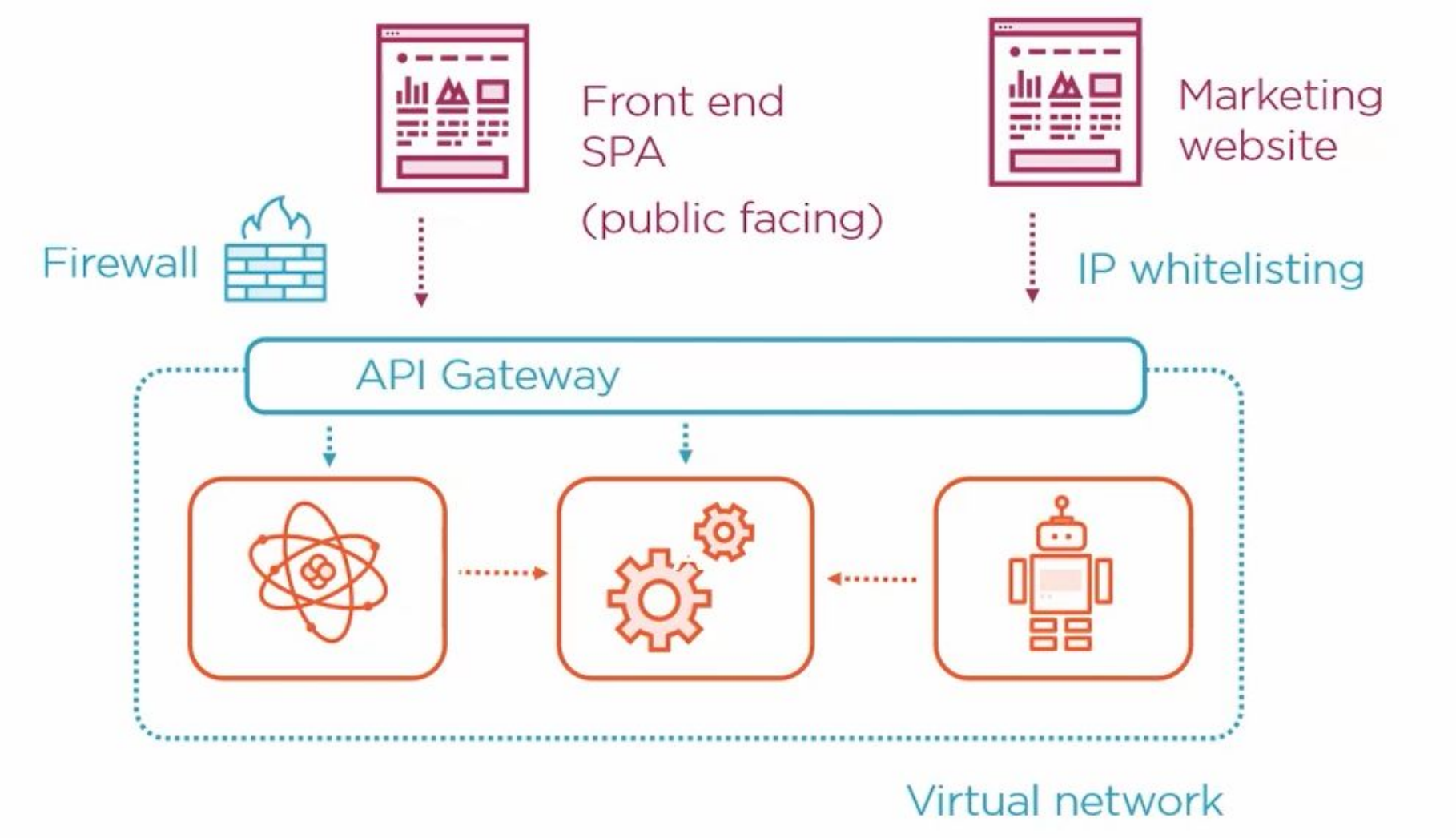


Securing Microservice

- Mã hóa dữ liệu bằng Giao thức TLS protocol (Transport Layer Security)
- Authentication
- Authorization



Securing Microservice



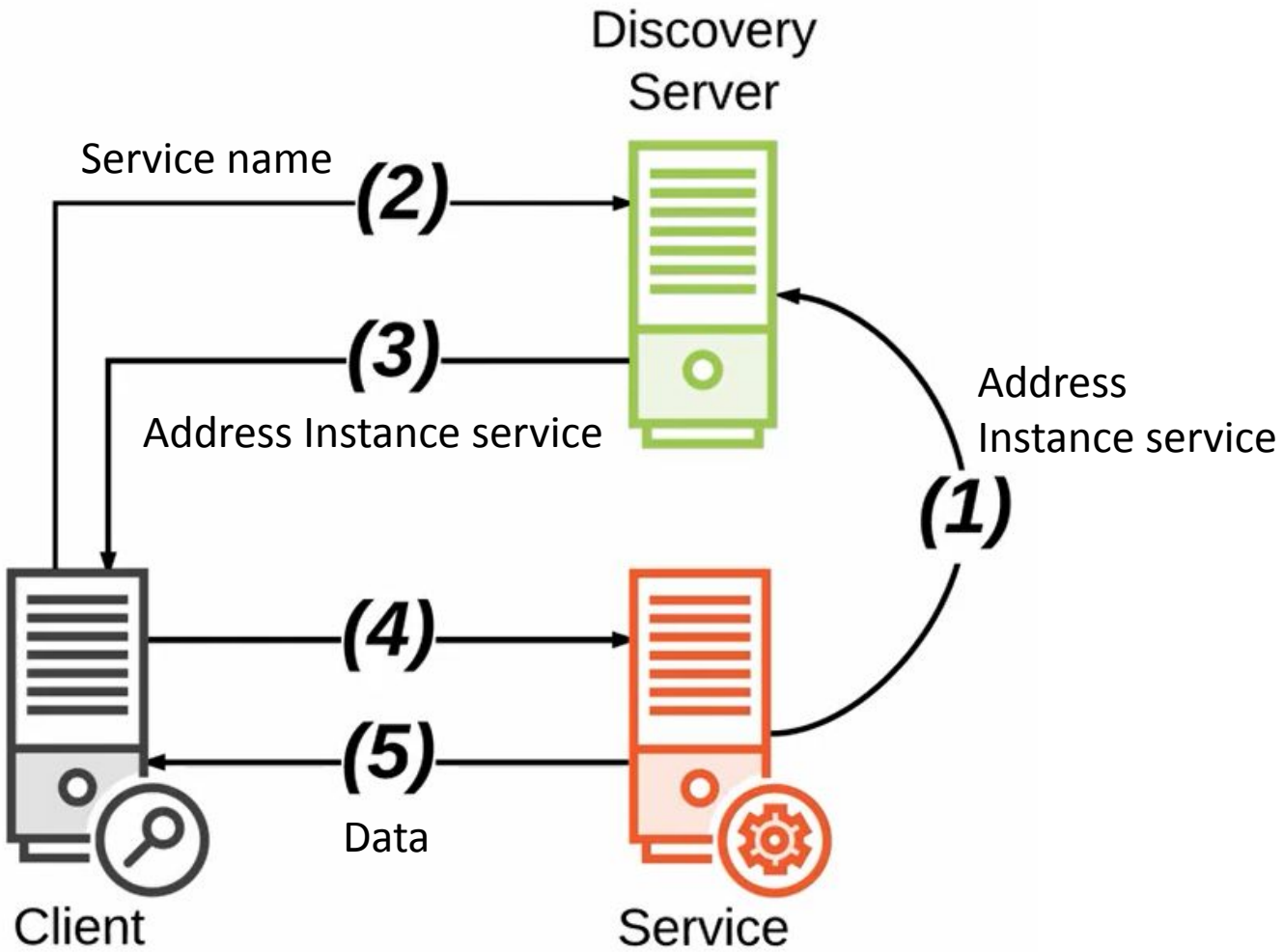
Spring cloud with Microservice

1. Service discovery
2. Distributed Configuration
3. Load balancing
4. Intelligent routing
5. Circuit Breaker

Spring cloud with Microservice

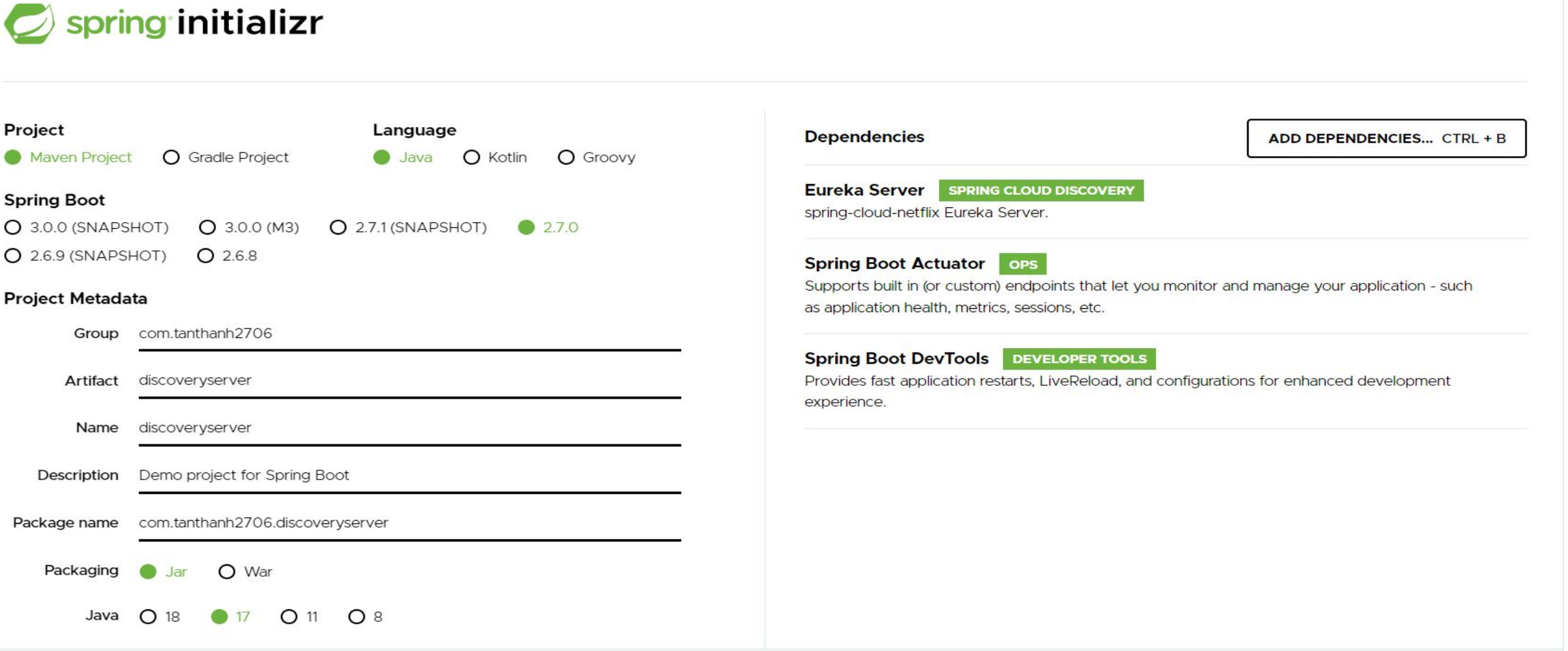
- Spring Cloud là một công nghệ phần mềm sử dụng để phát triển các ứng dụng phân tán.

Service discovery



Discovery server

- Tạo discovery server với spring cloud Eureka



Discovery server

DiscoverserverApplication.java

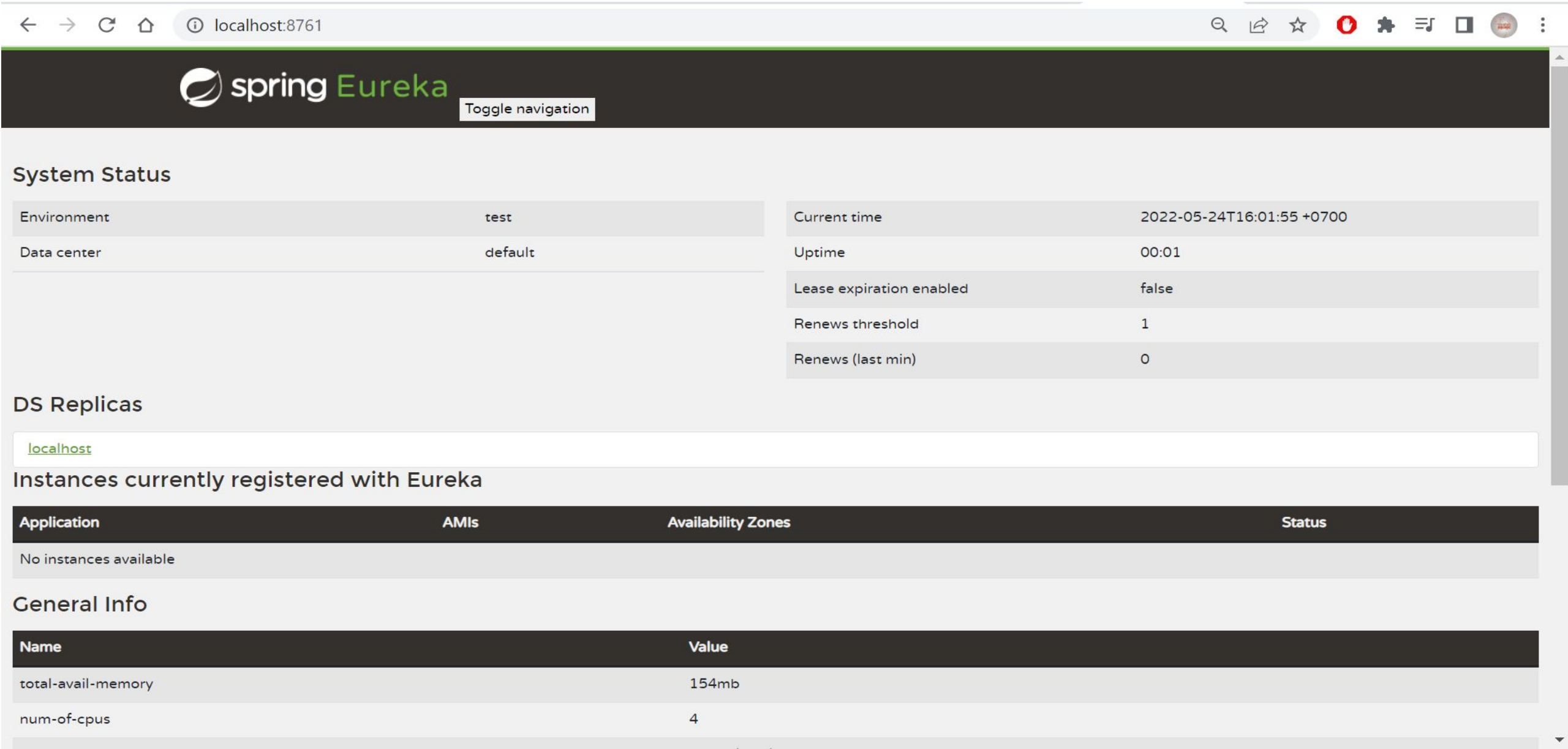
```
1  package com.tanthanh.discoverserver;
2
3  import org.springframework.boot.SpringApplication;
6
7  @EnableEurekaServer
8  @SpringBootApplication
9  public class DiscoverserverApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(DiscoverserverApplication.class, args);
13     }
14
15 }
16
```

Discovery server

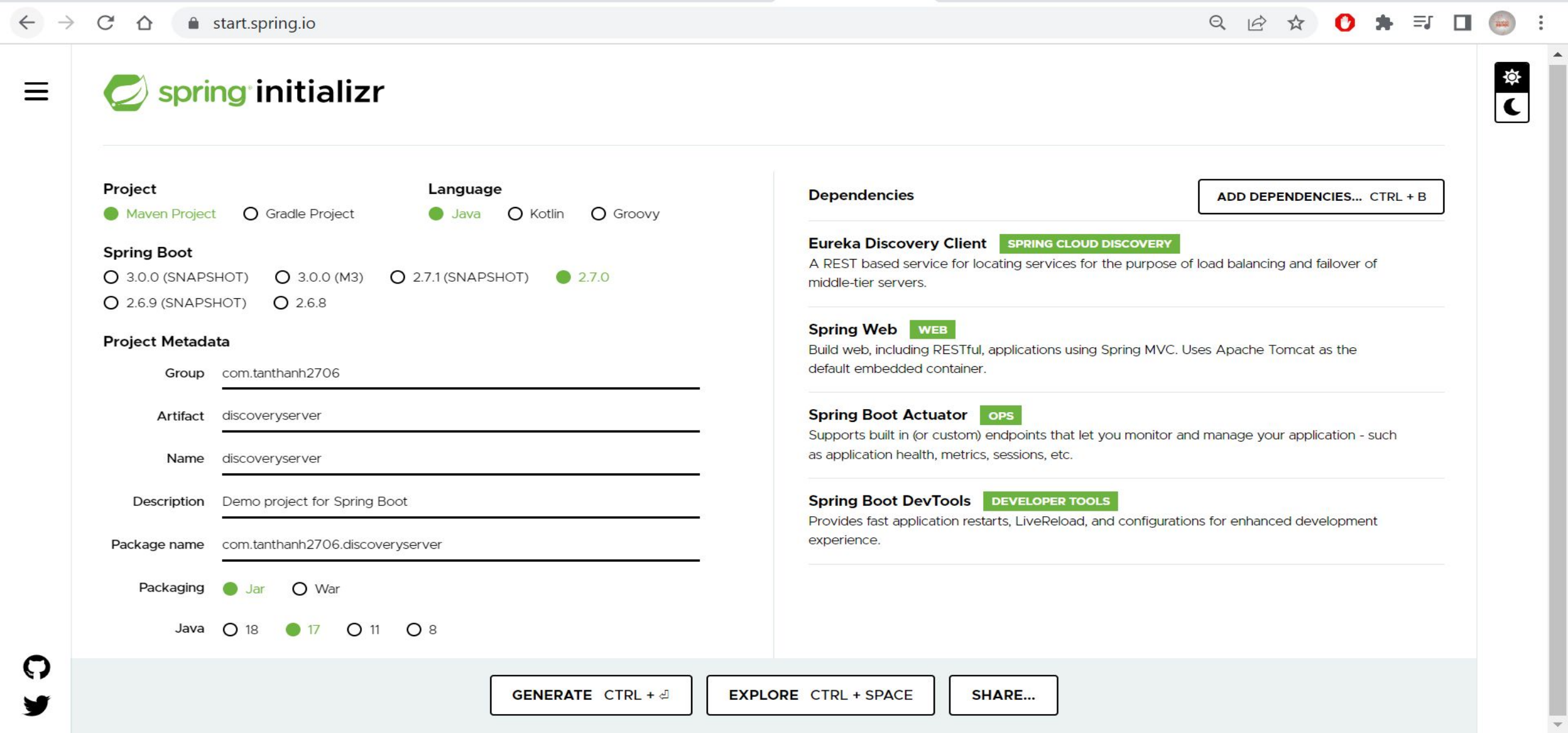
application.properties

```
1 spring.application.name=discovery-server
2 eureka.client.register-with-eureka=false
3 eureka.client.fetch-registry=false
4 server.port=8761
5
```

Discovery server



Application Services



Application Services

ServiceApplication.java

```java
package com.tanthanh.service;

import org.springframework.boot.SpringApplication;

@EnableDiscoveryClient
@SpringBootApplication
public class ServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(ServiceApplication.class, args);
    }

}
```

application.properties

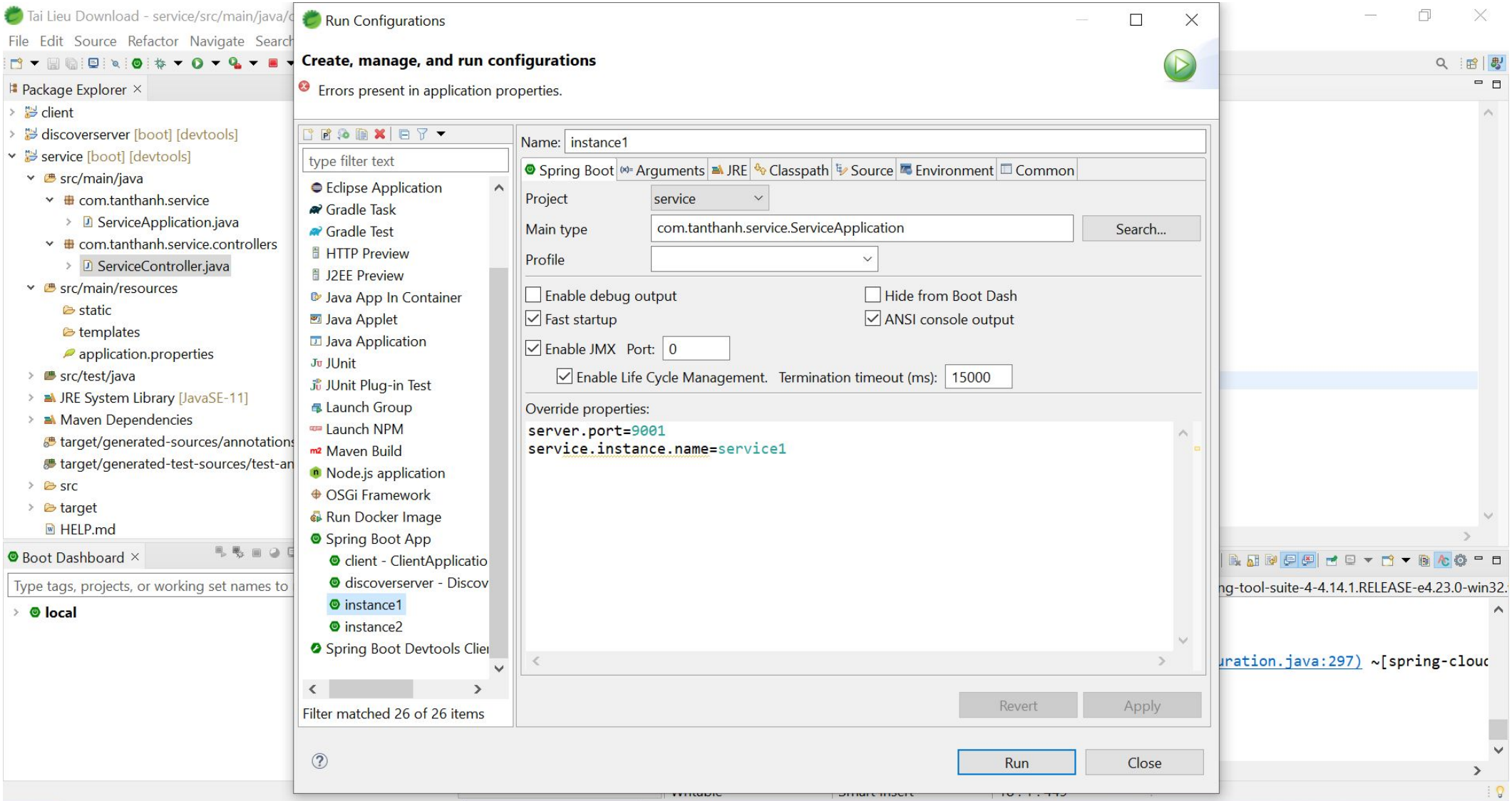
```
spring.application.name=service
eureka.client.service-url.defaultZone=http://localhost:8761/eureka

```

Application Services

ServiceController.java

```java
package com.tanthanh.service.controllers;

import org.springframework.beans.factory.annotation.Value;

@RestController
public class ServiceController {

    @Value("${service.instance.name}")
    private String instance;

    @RequestMapping("/")
    public String sayHello() {
        return "hello from "+instance;
    }
}

```

Application Services



Application Services

spring Eureka

Toggle navigation

System Status

Environment	test
Data center	default

Current time	2022-05-24T16:14:49 +0700
Uptime	00:00
Lease expiration enabled	false
Renews threshold	5
Renews (last min)	0

DS Replicas

localhost

Instances currently registered with Eureka

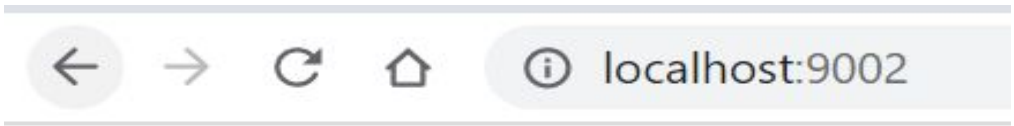
Application	AMIs	Availability Zones	Status
SERVICE	n/a (2)	(2)	UP (2) - LVN01000155.corp.capgemini.com:service:9002 , LVN01000155.corp.capgemini.com:service:9001

General Info

Name	Value
total-avail-memory	112mb
num-of-cpus	4

Application Services

hello from service1



hello from service2

Application Client

application.properties

```
1 spring.application.name=client
2 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
3 eureka.client.register-with-eureka=false
```

Application Client

ClientController.java

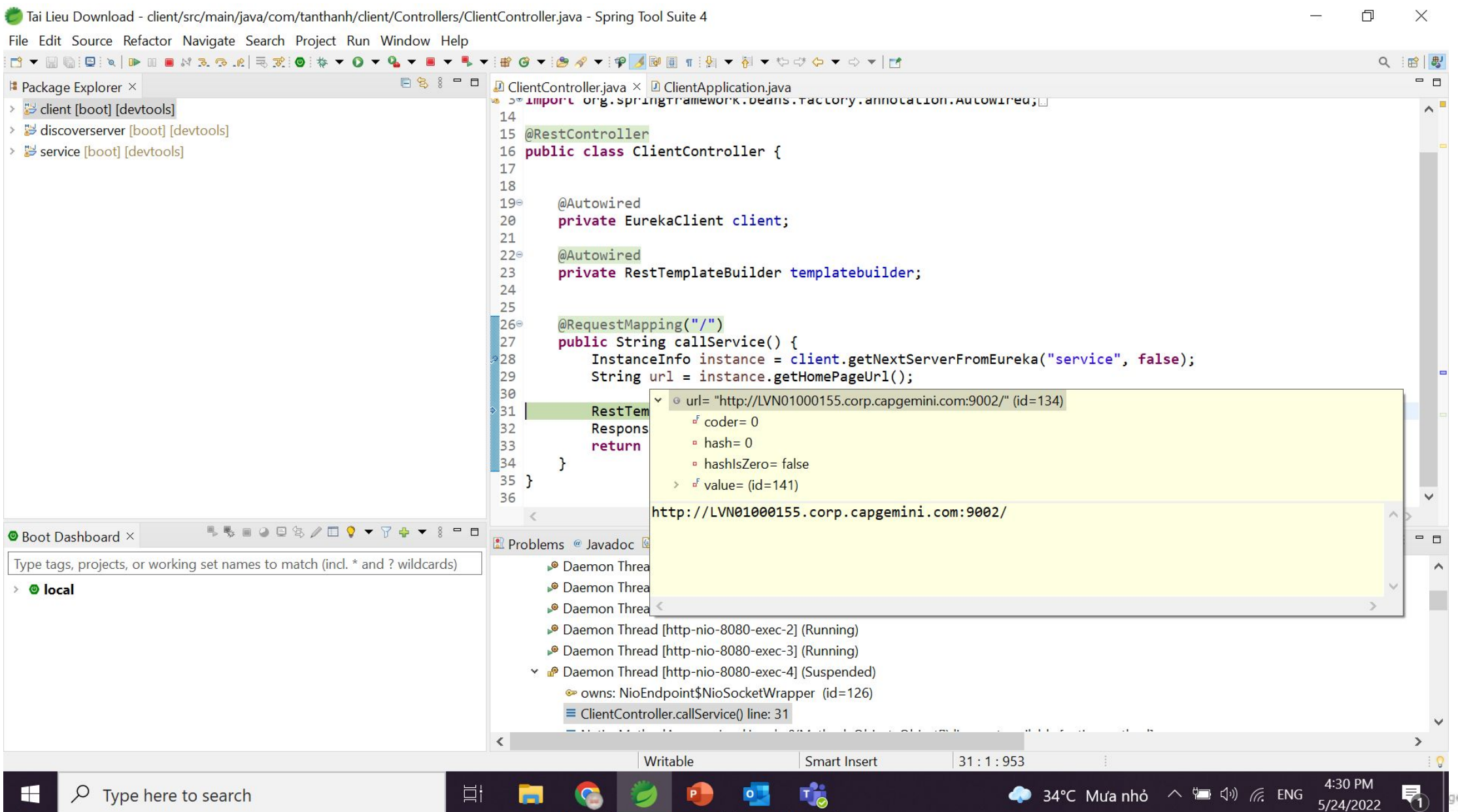
```
import org.springframework.beans.factory.annotation.Autowired;

@RestController
public class ClientController {


    @Autowired
    private EurekaClient client;

    @Autowired
    private RestTemplateBuilder templatebuilder;


    @RequestMapping("/")
    public String callService() {
        InstanceInfo instance = client.getNextServerFromEureka("service", false);
        String url = instance.getHomePageUrl();

        RestTemplate restTemplate = templatebuilder.build();
        ResponseEntity<String> response = restTemplate.exchange(url, HttpMethod.GET, null, String.class);
        return response.getBody();
    }
}

```

Application Client

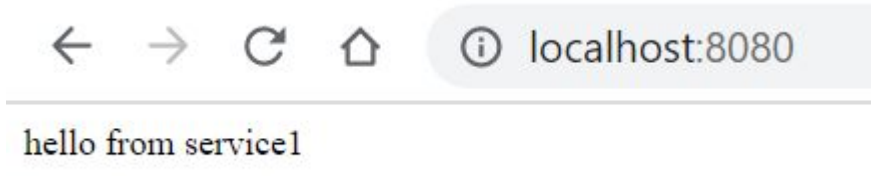
ClientController.java

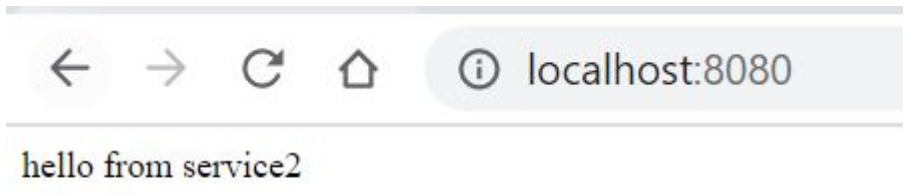
```
import org.springframework.beans.factory.annotation.Autowired;

@RestController
public class ClientController {


    @Autowired
    private EurekaClient client;

    @Autowired
    private RestTemplateBuilder templatebuilder;


    @RequestMapping("/")
    public String callService() {
        InstanceInfo instance = client.getNextServerFromEureka("service", false);
        String url = instance.getHomePageUrl();

        RestTemplate restTemplate = templatebuilder.build();
        ResponseEntity<String> response = restTemplate.exchange(url, HttpMethod.GET, null, String.class);
        return response.getBody();
    }
}

```

Application Client

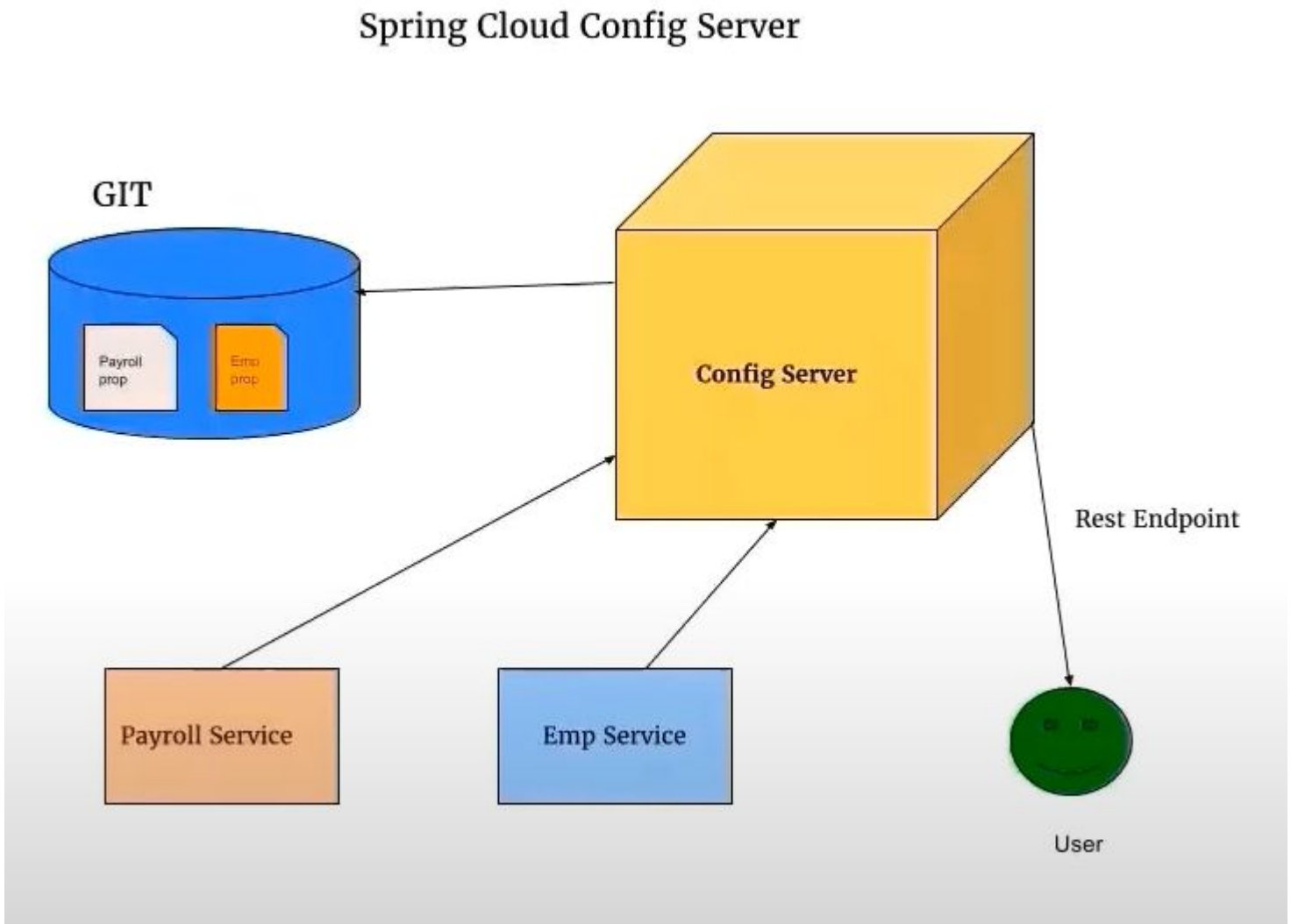


Application Client

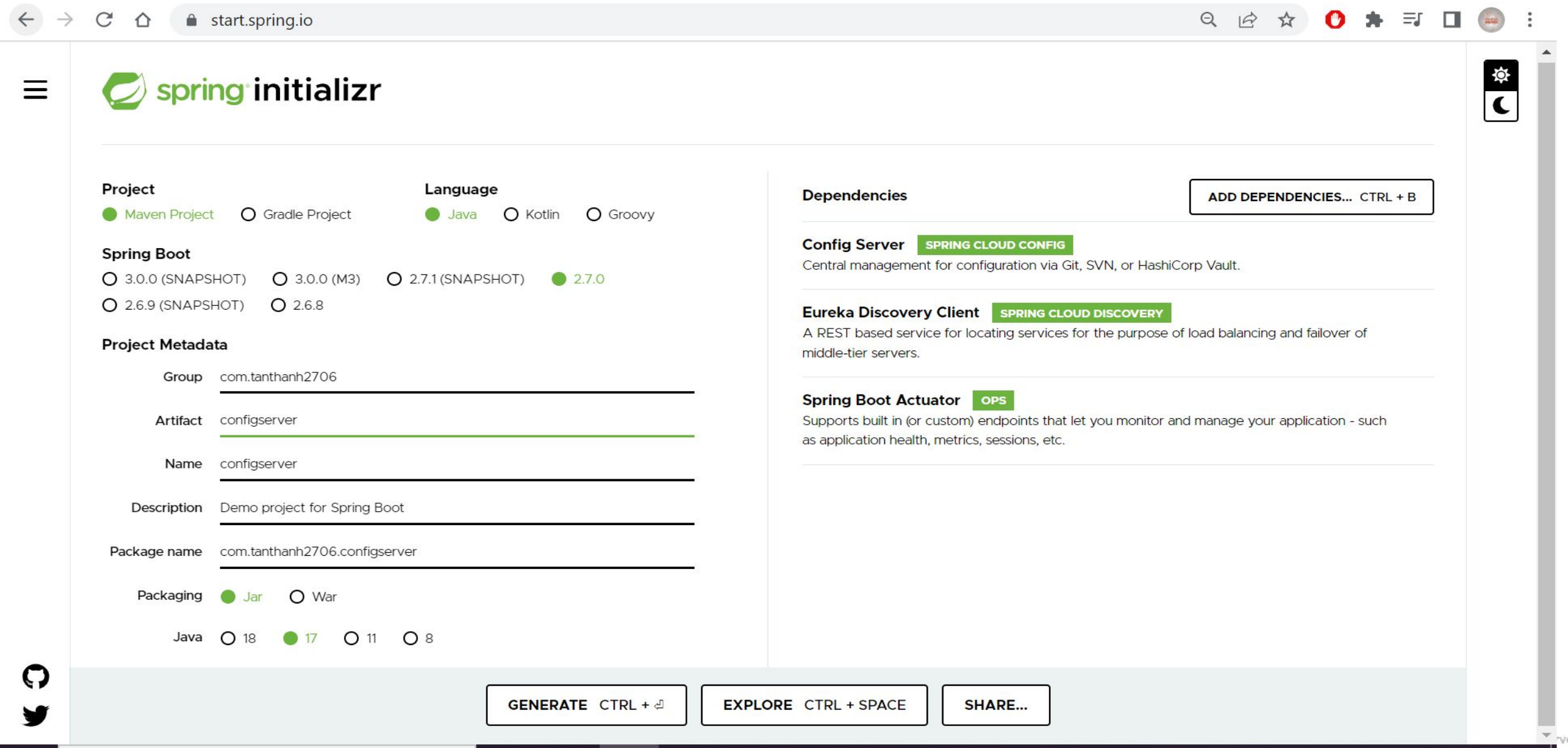




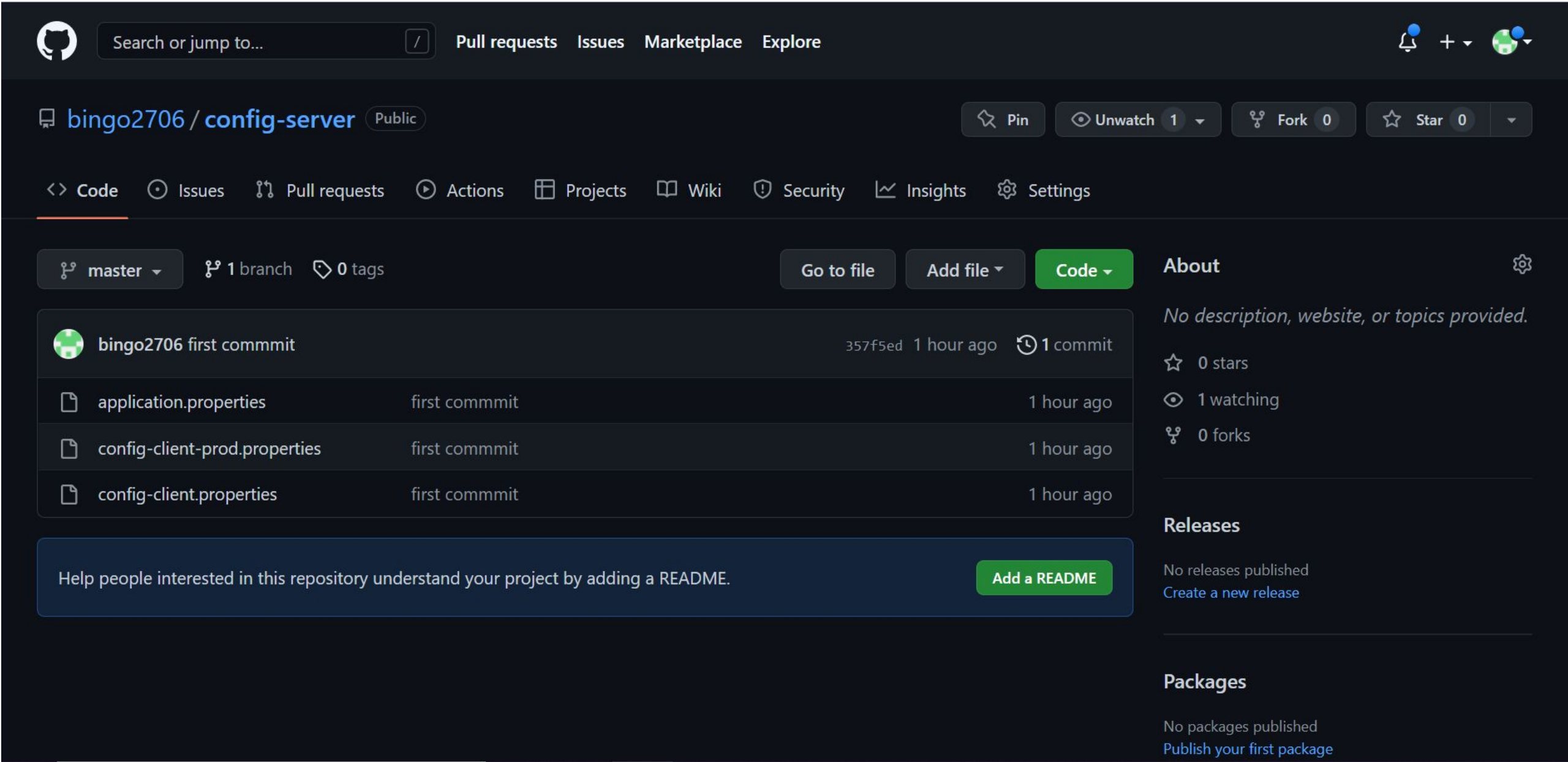
Distributed Configuration



Spring cloud config



Spring cloud config



Spring cloud config

```
2 lines (2 sloc)  67 Bytes
1  sample.property1=sample property1
2  sample.property2=sample property2
```

Application.properties

```
2 lines (2 sloc)  98 Bytes
1  sample.property1=sample property1 config-client1
2  sample.property2=sample property2 config-client2
```

Config-client.properties

```
2 lines (2 sloc)  110 Bytes
1  sample.property1=sample property1 config-client1 prod1
2  sample.property2=sample property2 config-client2 prod2
```

Config-client-prod.properties

Spring cloud config

ConfigserverApplication.java

```java
package com.tanthanh2706.configserver;

import org.springframework.boot.SpringApplication;

@EnableDiscoveryClient
@EnableConfigServer
@SpringBootApplication
public class ConfigserverApplication {

    public static void main(String[] args) {
        SpringApplication.run(ConfigserverApplication.class, args);
    }

}
```

application.properties

```
spring.application.name=configserver
server.port=8888
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
spring.cloud.config.server.git.uri=https://github.com/bingo2706/config-server.git
spring.cloud.config.server.git.default-label=master
```

Spring cloud config

Endpoint

```
GET /{application}/{profile}[/{label}]
```

Example

- /myapp/dev/master

Spring cloud config

Spring cloud config

Spring cloud config

localhost:8888/config-client/prod


```
{
  name: "config-client",
  profiles: [
    "prod"
  ],
  label: null,
  version: "843460a127d0fd3128623e52a4f97c7fe387a5a8",
  state: null,
  propertySources: [
    {
      name: https://github.com/bingo2706/config-server.git/file:C:\\Users\\thando\\AppData\\Local\\Temp\\config-repo-9957999627645791082\\config-client-prod.properties,
      source: {
        "sample.property1": "sample property1 config-client1 prod1",
        "sample.property2": "sample property2 config-client2 prod2"
      }
    },
    {
      name: https://github.com/bingo2706/config-server.git/file:C:\\Users\\thando\\AppData\\Local\\Temp\\config-repo-9957999627645791082\\config-client.properties,
      source: {
        "sample.property1": "sample property1 config-client1",
        "sample.property2": "sample property2 config-client2"
      }
    },
    {
      name: https://github.com/bingo2706/config-server.git/file:C:\\Users\\thando\\AppData\\Local\\Temp\\config-repo-9957999627645791082\\application.properties,
      source: {
        "sample.property1": "sample property1",
        "sample.property2": "sample property2"
      }
    }
  ]
}
```

Spring cloud config

spring initializr

Project
- (●) Maven Project
- ( ) Gradle Project

Language
- (●) Java
- ( ) Kotlin
- ( ) Groovy

Spring Boot
- ( ) 3.0.0 (SNAPSHOT)
- ( ) 3.0.0 (M3)
- ( ) 2.7.1 (SNAPSHOT)
- (●) 2.7.0
- ( ) 2.6.9 (SNAPSHOT)
- ( ) 2.6.8

Project Metadata

Group	com.tanthanh
Artifact	configclient
Name	configclient
Description	Demo project for Spring Boot
Package name	com.tanthanh.configclient
Packaging	(●) Jar ( ) War
Java	( ) 18 ( ) 17 (●) 11 ( ) 8

Dependencies — ADD DEPENDENCIES... CTRL + B

Config Client SPRING CLOUD CONFIG
Client that connects to a Spring Cloud Config Server to fetch the application's configuration.

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Eureka Discovery Client SPRING CLOUD DISCOVERY
A REST based service for locating services for the purpose of load balancing and failover of middle-tier servers.

Spring Boot Actuator OPS
Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

GENERATE CTRL + ⏎ | EXPLORE CTRL + SPACE | SHARE...

Spring cloud config

ConfigclientApplication.java

```java
package com.tanthanh.configclient;

import org.springframework.boot.SpringApplication;

@EnableDiscoveryClient
@SpringBootApplication
public class ConfigclientApplication {

    public static void main(String[] args) {
        SpringApplication.run(ConfigclientApplication.class, args);
    }

}
```

bootstrap.properties

```
spring.application.name=config-client
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
spring.cloud.config.discovery.enabled=true
spring.profiles.active=prod
```

Spring cloud config

ClientConfig.java

```java
package com.tanthanh.configclient;

import org.springframework.boot.context.properties.ConfigurationProperties;

@Component
@ConfigurationProperties(prefix = "sample")
public class ClientConfig {
    private String property1;

    public String getProperty1() {
        return property1;
    }

    public void setProperty1(String property1) {
        this.property1 = property1;
    }
}
```

Spring cloud config

ConfigClientController.java

```
3  import org.springframework.beans.factory.annotation.Autowire
4  import org.springframework.beans.factory.annotation.Value;
5  import org.springframework.web.bind.annotation.GetMapping;
6  import org.springframework.web.bind.annotation.RequestMappir
7  import org.springframework.web.bind.annotation.RestControlle
8
9  @RestController
10 public class ConfigClientController {
11
12     @Autowired
13     ClientConfig clientConfig;
14
15     @Value("${sample.property2}")
16     private String property2;
17
18     @GetMapping("/config")
19     public String printConfig() {
20         return clientConfig.getProperty1() + " "+property2;
21     }
22 }
```

Refresh Config

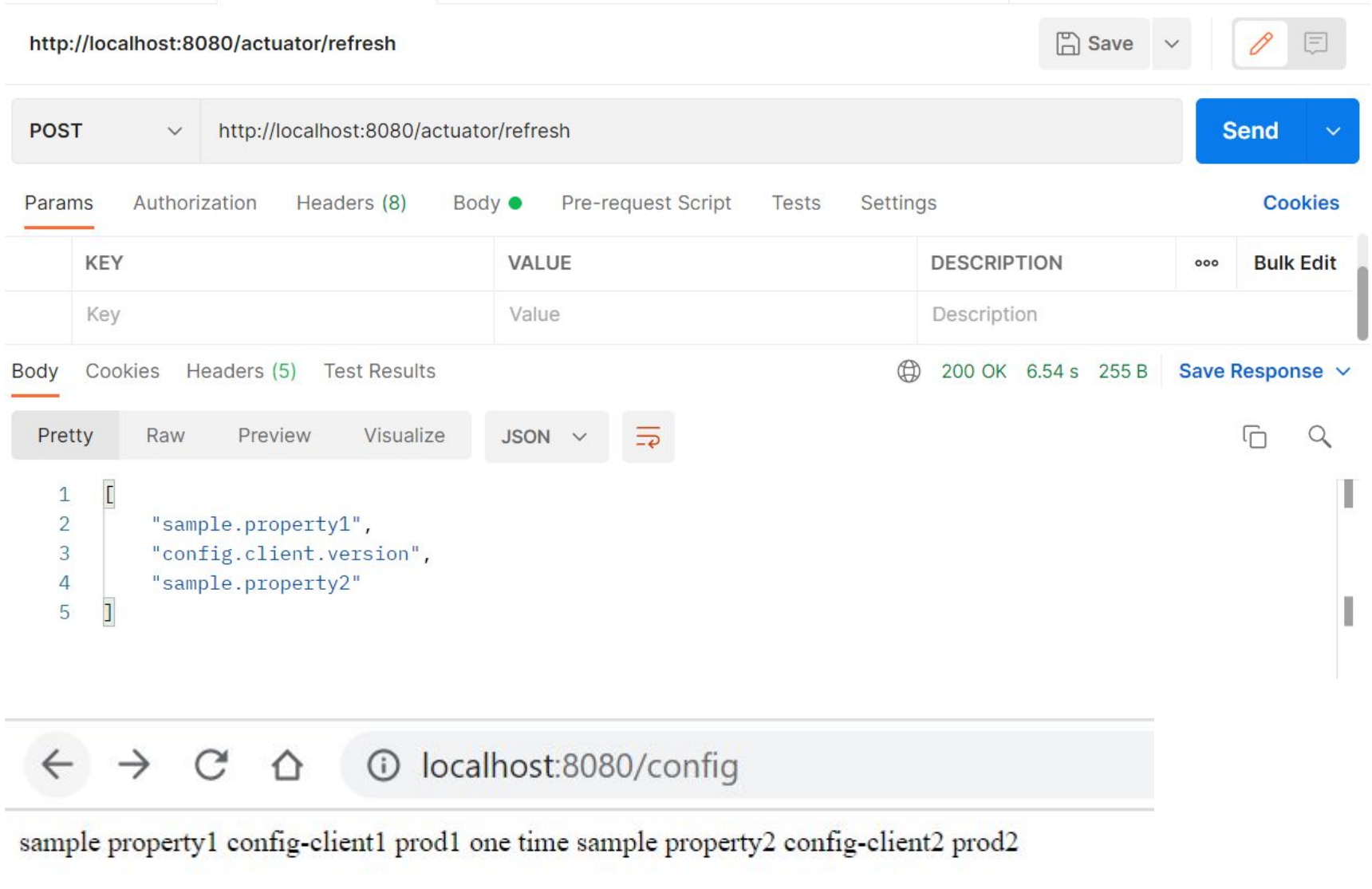
bootstrap.properties

```
spring.application.name=config-client
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
spring.cloud.config.discovery.enabled=true
spring.profiles.active=prod

management.endpoints.web.exposure.include=*
```

2 lines (2 sloc) | 128 Bytes

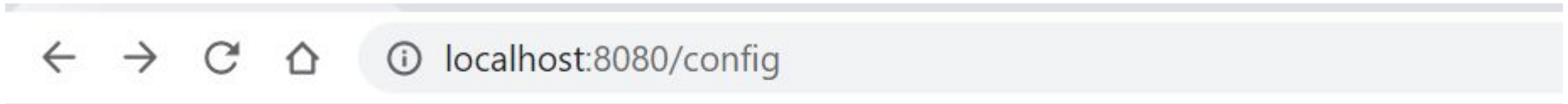
```
sample.property1=sample property1 config-client1 prod1 one time
sample.property2=sample property2 config-client2 prod2 one time
```

Refresh Config



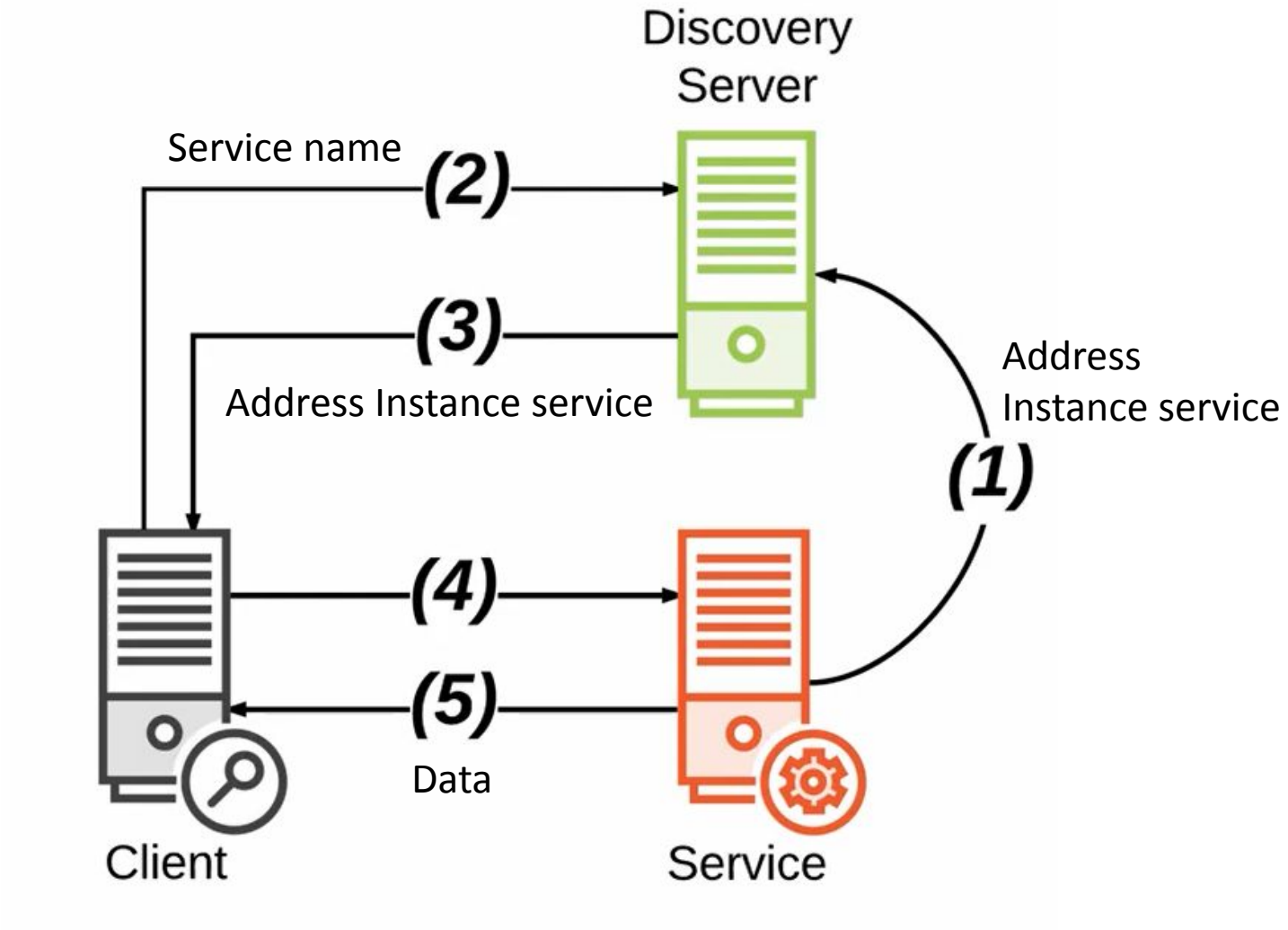
Refresh Config

```java
@RestController
@RequestScope
public class ConfigClientController {

    @Autowired
    ClientConfig clientConfig;


    @Value("${sample.property2}")
    private String property2;

    @GetMapping("/config")
    public String printConfig() {
        return clientConfig.getProperty1() + " "+property2;
    }
}
```

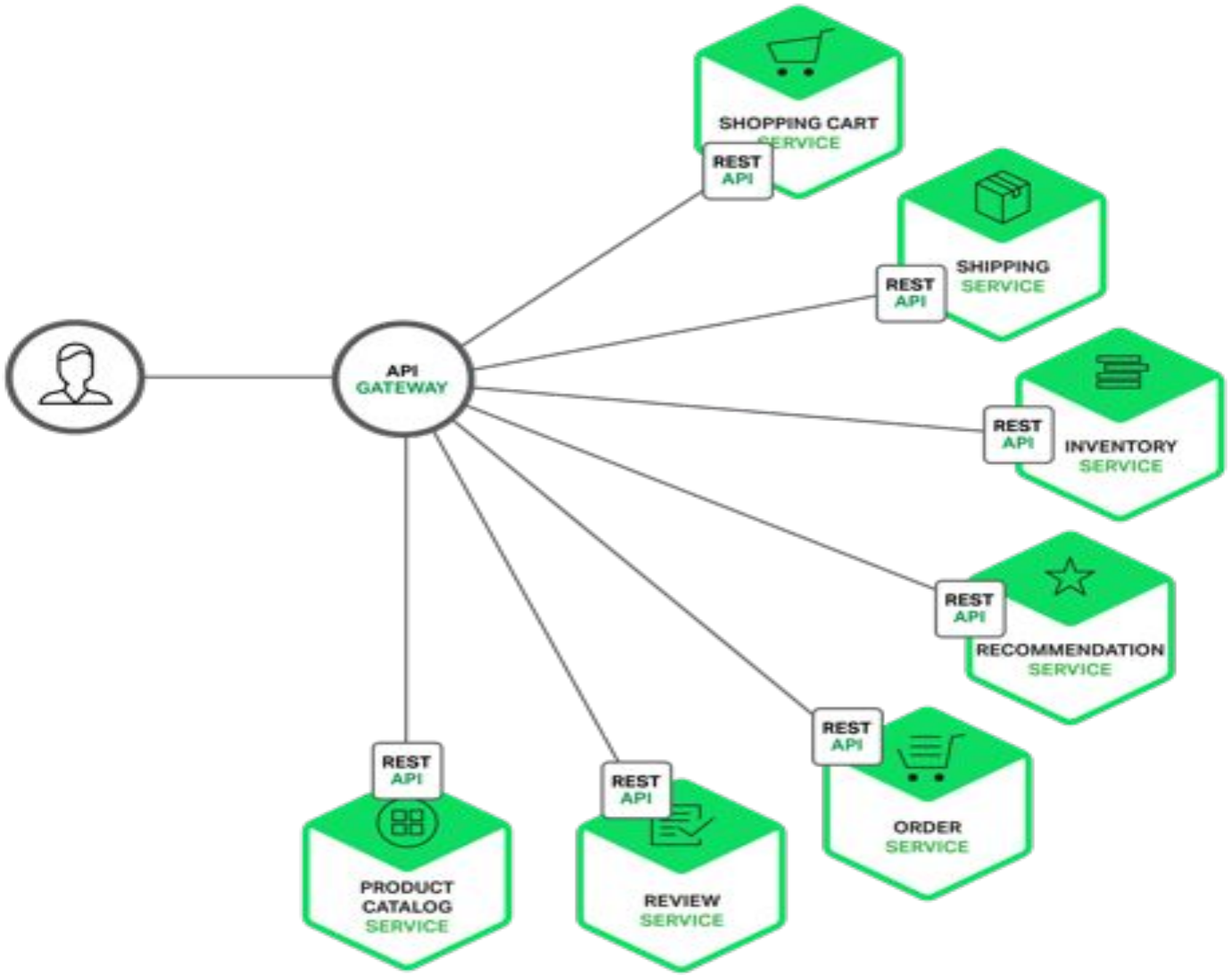


sample property1 config-client1 prod1 one time second time sample property2 config-client2 prod2 one time second time

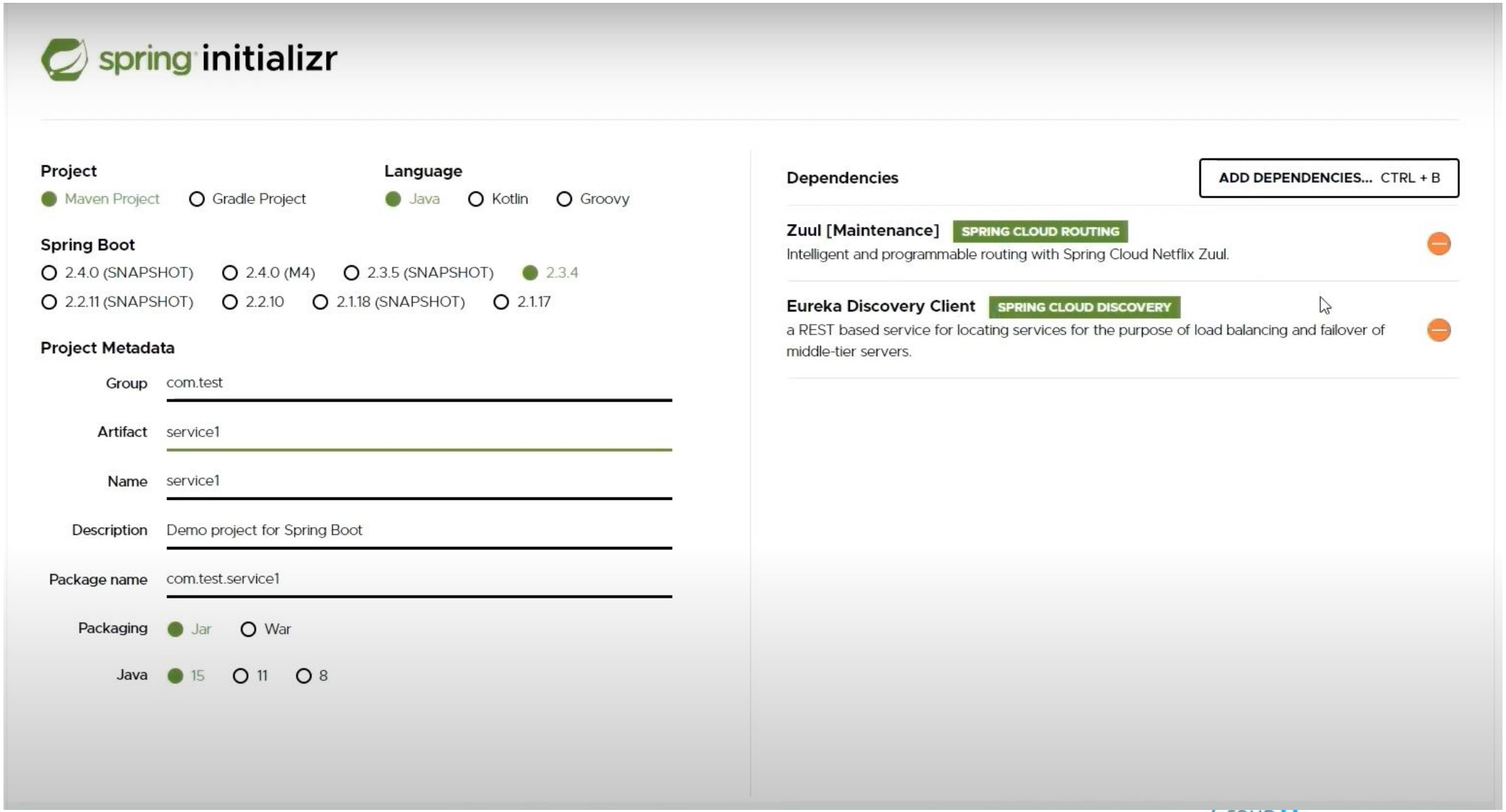
Intelligent routing



Intelligent routing



Intelligent routing



Intelligent routing

GatewayserviceApplication.java

```
1  package com.test.gatewayservice;
2
3  import org.springframework.boot.SpringApplication;
7
8  @SpringBootApplication
9  @EnableZuulProxy
10 @EnableDiscoveryClient
11 public class GatewayserviceApplication {
12
13     public static void main(String[] args) {
14         SpringApplication.run(GatewayserviceApplication.class, args);
15     }
16
17 }
18
```

GatewayserviceApplication.java | application.properties

```
1  spring.application.name=gateway-service
2  eureka.client.service-url.defaultZone=http://localhost:8761/eureka
3
```

Intelligent routing

Service 1

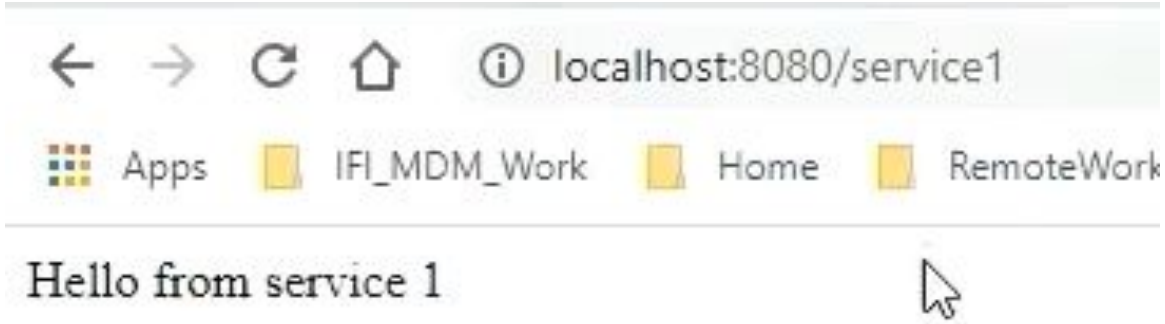
```
@RestController
public class TestController {

    @RequestMapping
    public String sayHello()
    {
        return "Hello from service 1";
    }
}
```

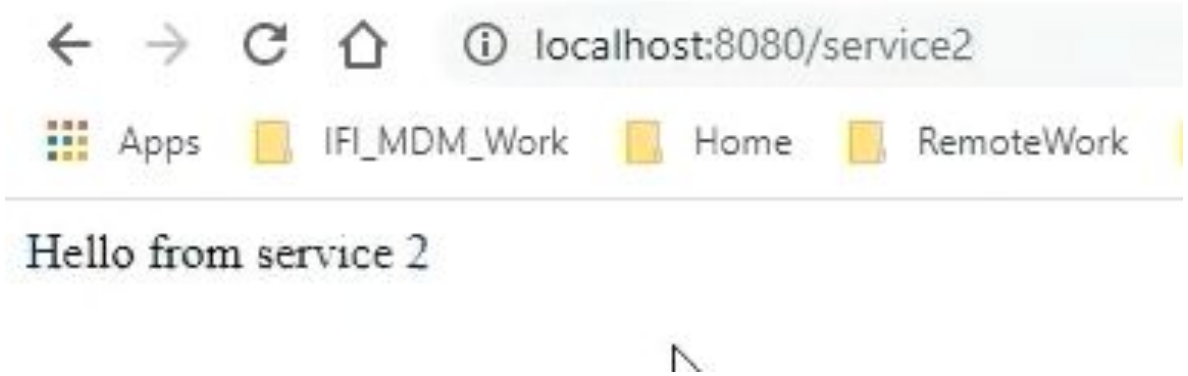
Service 2

```
@RestController
public class TestController {

    @RequestMapping
    public String sayHello()
    {
        return "Hello from service 2";
    }
}
```

Application	AMIs	Availability Zones	Status
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - localhost:gateway-service
SERVICE1	n/a (1)	(1)	UP (1) - localhost:service1:1111
SERVICE2	n/a (1)	(1)	UP (1) - localhost:service2:2222

Intelligent routing





Intelligent routing

```java
public class AddRequestIdFilter extends ZuulFilter {

    @Override
    public boolean shouldFilter() {
        return true;
    }

    @Override
    public Object run() throws ZuulException {
        RequestContext context = RequestContext.getCurrentContext();
        String requestId = UUID.randomUUID().toString();
        context.addZuulRequestHeader("requestId", requestId);
        return null;
    }

    @Override
    public String filterType() {
        return FilterConstants.PRE_TYPE;
    }

    @Override
    public int filterOrder() {
        // TODO Auto-generated method stub
        return 0;
    }

}
```

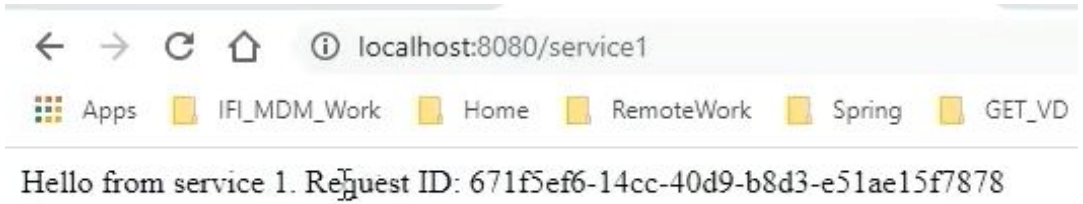
Intelligent routing

```
@Configuration
public class GatewayServiceConfiguration {

    @Bean
    public AddRequestIdFilter addRequestIdFilter()
    {
        return new AddRequestIdFilter();
    }
}
```

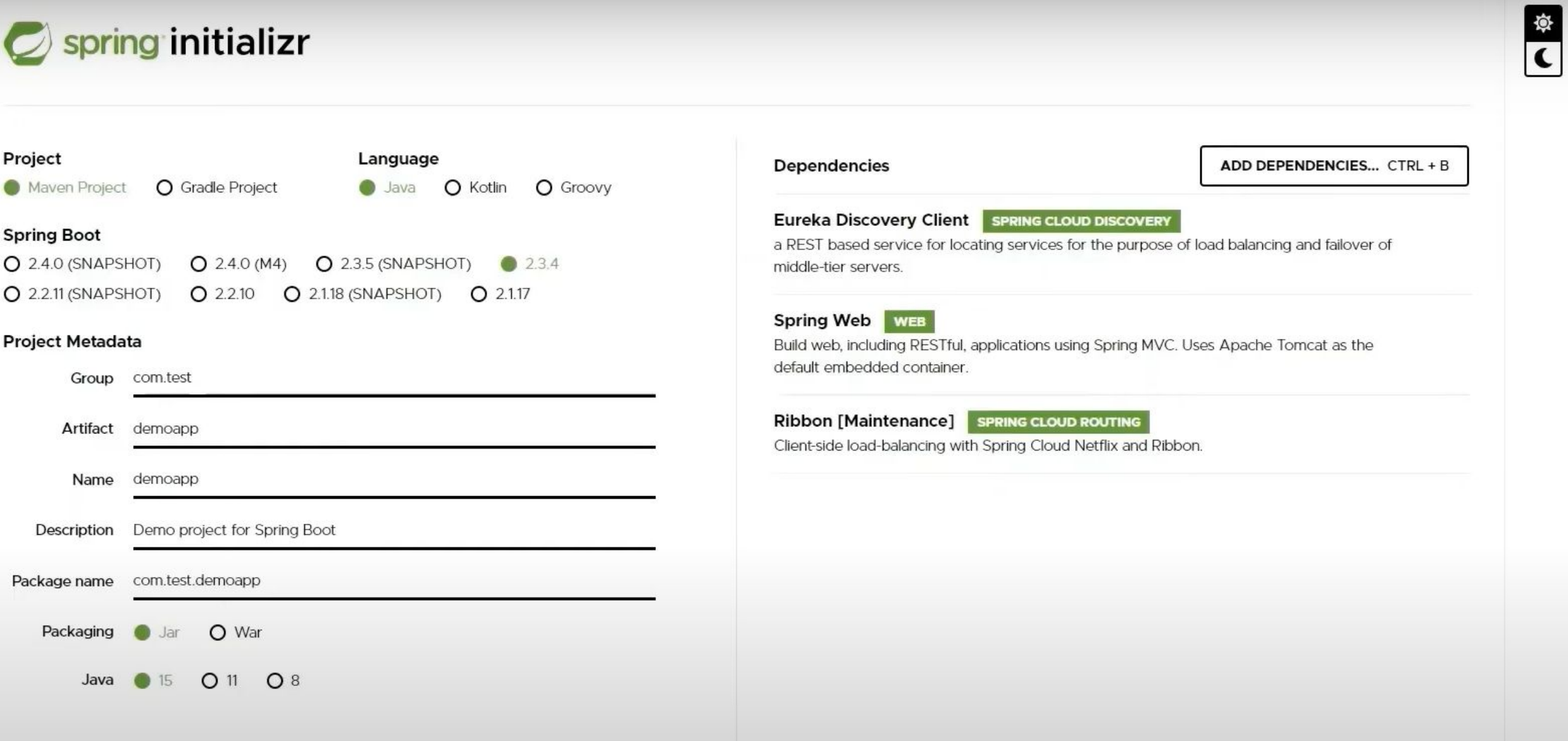
Gateway Service

```
@RestController
public class TestController {

    @RequestMapping
    public String sayHello(@RequestHeader("requestId") String requestId)
    {
        return "Hello from service 1. Request ID: " + requestId;
    }
}
```

Service 1



Client-side Load Balancing



Client-side Load Balancing

```java
@Configuration
public class DemoConfiguration {

    @Bean
    @LoadBalanced
    public RestTemplate getRestTemplate()
    {
        return new RestTemplate();
    }
}
```

```java
@RestController
public class DemoAppController {

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping
    public String getServiceInstanceInfo()
    {
        return this.restTemplate.getForEntity("http://demoservice", String.class).getBody();
    }
}
```

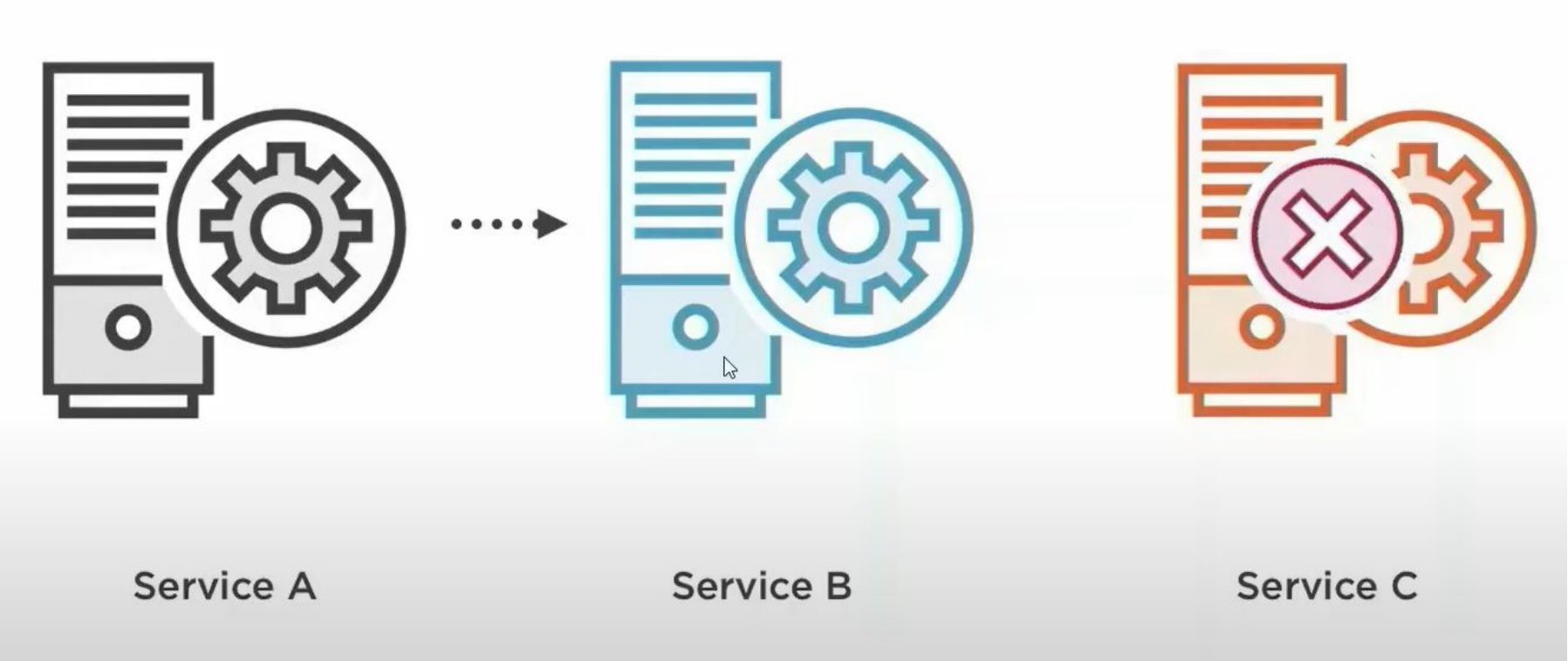
Client-side Load Balancing

- Overide thuật toán của load balancing

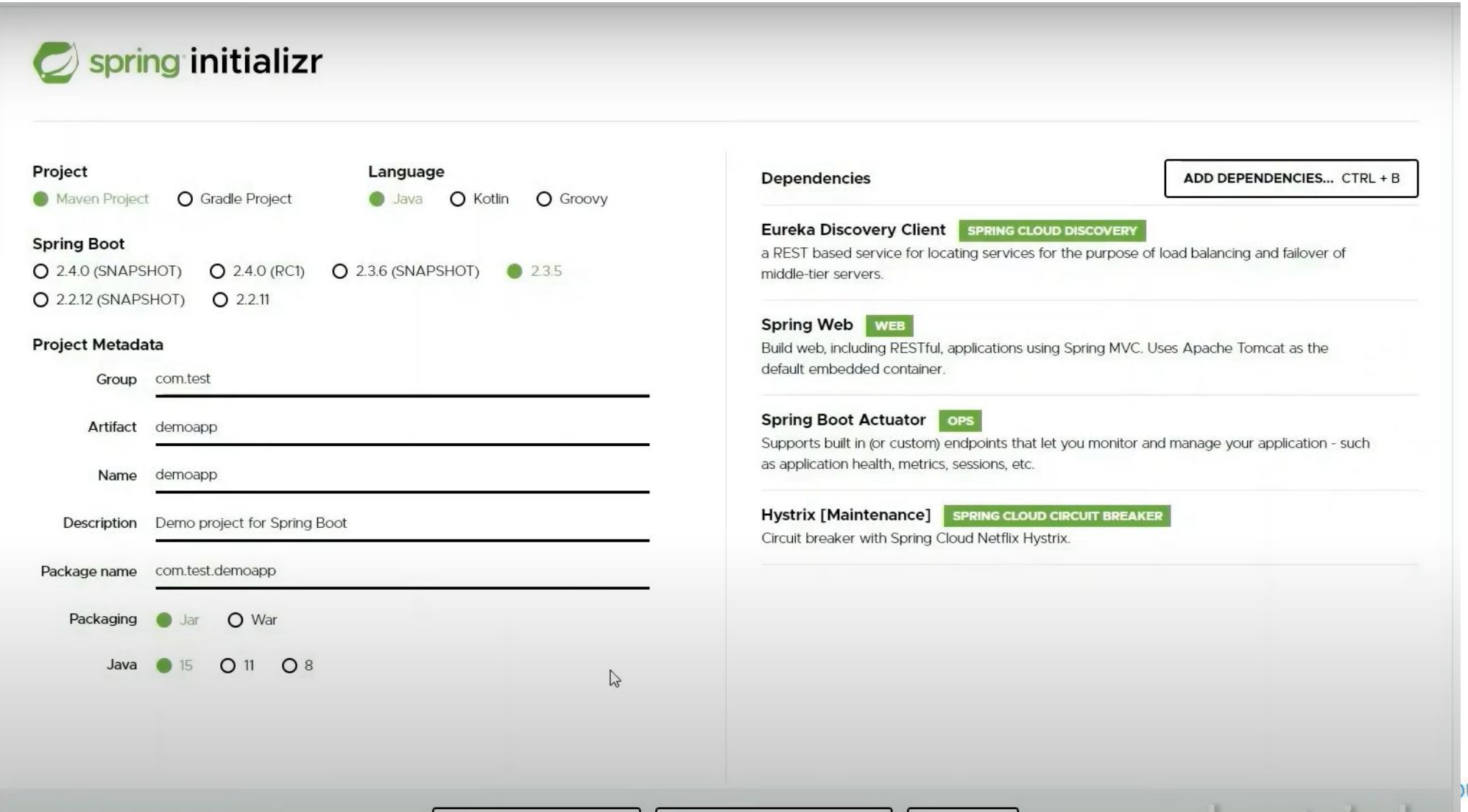
```
@Configuration
public class DemoConfiguration {

    @Bean
    @LoadBalanced
    public RestTemplate getRestTemplate()
    {
        return new RestTemplate();
    }

    @Bean
    public IRule getRule()
    {
        return new RandomRule();
    }
}
```

Circuit Breaker

- Cascading failure



Circuit Breaker



Circuit Breaker

```java
@EnableDiscoveryClient
@EnableCircuitBreaker
@SpringBootApplication
public class DemoappApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoappApplication.class, args);
    }

}
```

```java
@Service
public class DemoAppService {

    @Autowired
    private RestTemplate restTemplate;

    @HystrixCommand(fallbackMethod = "zeroNumber")
    public int getRandomNumber()
    {
        return this.restTemplate.getForEntity("http://numberservice/randomNumber", Integer.class).getBody();
    }

    public int zeroNumber()
    {
        return 0;
    }
}
```

DDD Layered Architecture

*** Là gì ?**

- Là một cách tiếp cận phát triển phần mềm để kết nối các nghiệp vụ phức tạp với phần mềm

*** Tại sao cần quan tâm đến DDD**

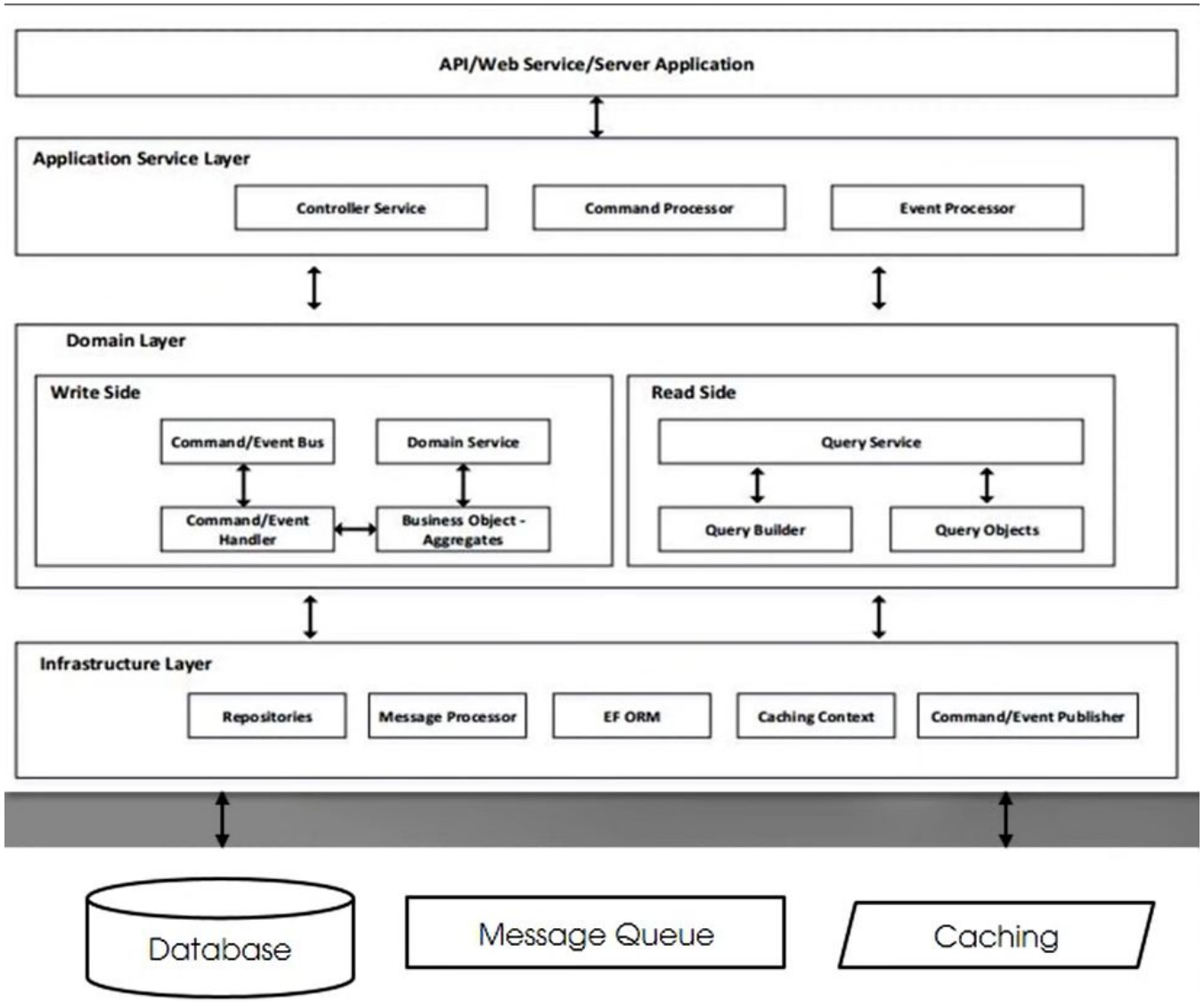
- Nguyên tắc và mẫu thiết kế để giải quyết các vấn đề phức tạp
- Lịch sử thành công với nhiều dự án phức tạp
- Code thể hiện nghiệp vụ rõ rang.

DDD Layered Architecture

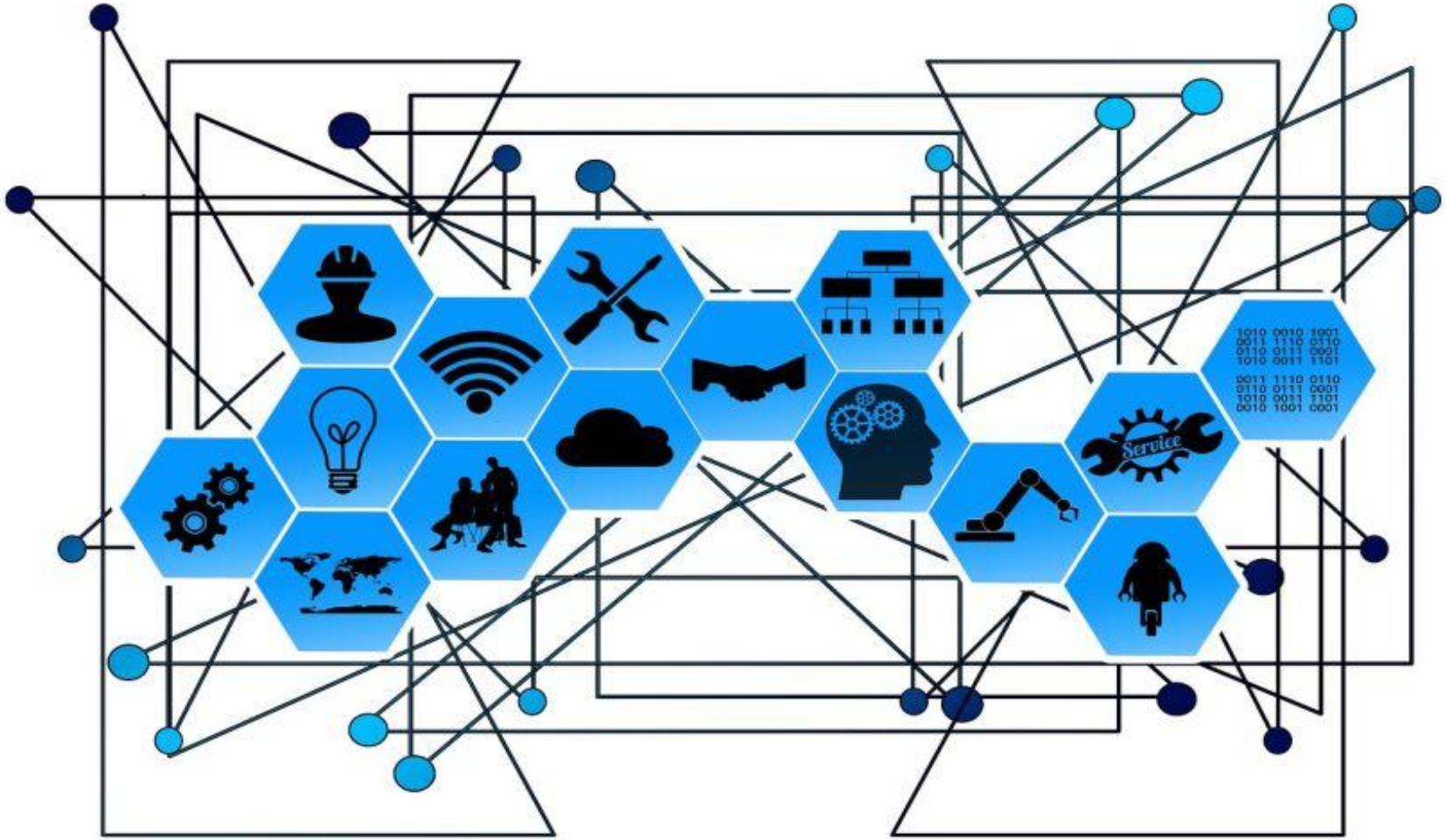
*** Điểm mấu chốt trong DDD**

1) Đào sâu kiến thức với Domain Expert
2) Tập trung vào mô hình của từng sub domain
3) Thực thi từng subdomain

DDD Layered Architecture

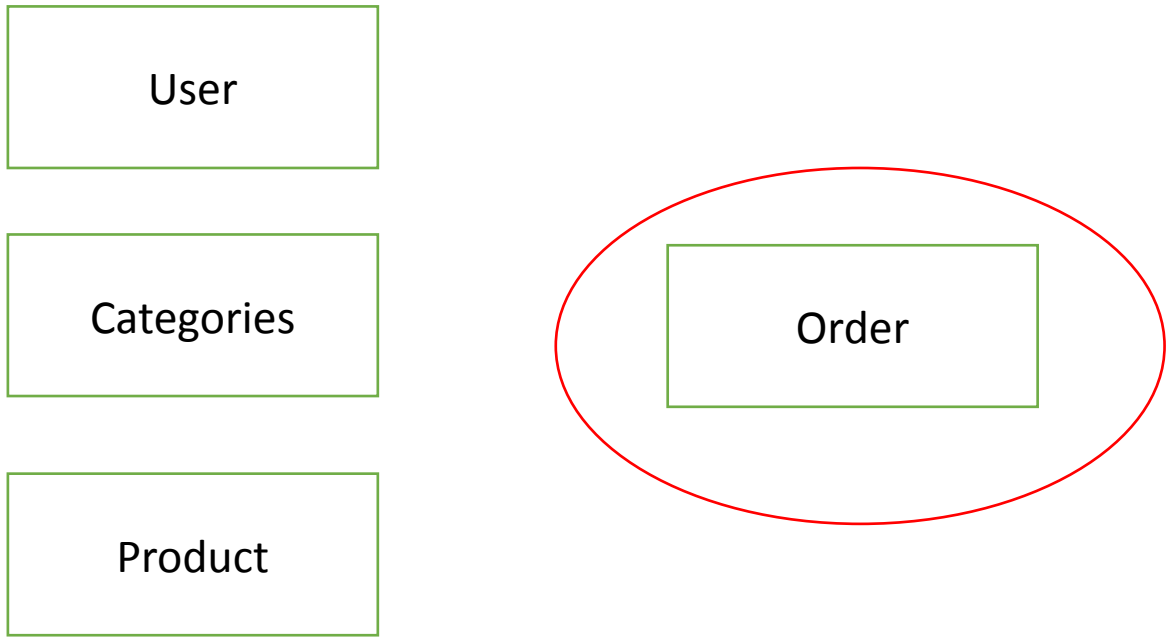


Domain model



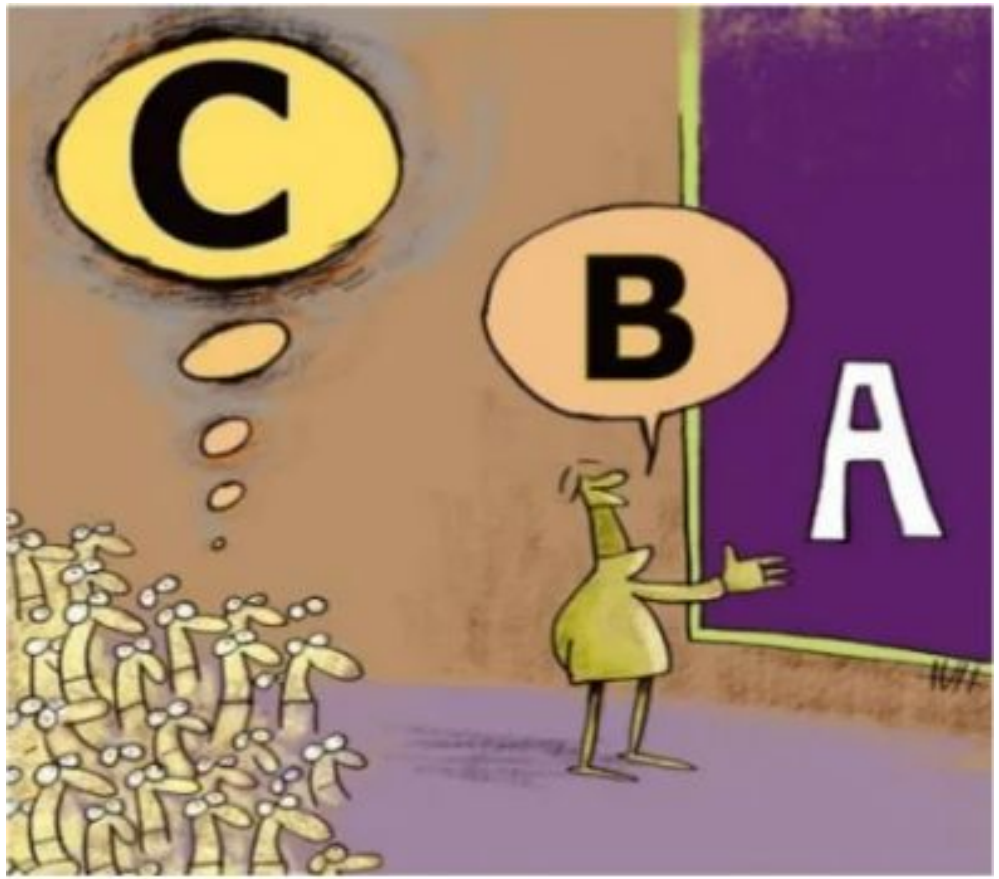
Domain model

- Phân tách nghiệp vụ thành các nghiệp vụ con và đào sâu và nó
-- Tập trung vào từng sub domain



Domain model

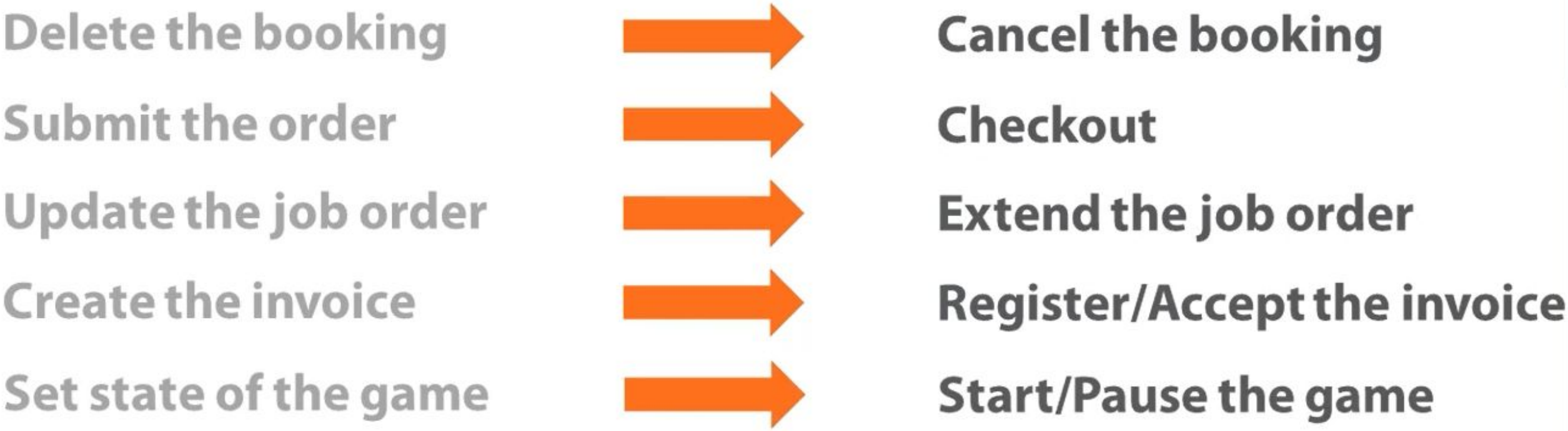
Ngôn ngữ chung (Ubiquitous Language)



Domain model

Ngôn ngữ chung (Ubiquitous Language)

Client = Customer, consumer, Khách hang
Voucher = coupon, gift card



Domain model

Anemic model

- Cấu trúc
- Thuộc tính
- Quan hệ

Rich model

- Cấu trúc
- Thuộc tính
- Quan hệ
- Hành vi

Domain model

Anemic model

```
public class Appointment : Entity<Guid>
{
    2 references
    public Guid ScheduleId { get; private set; }
    4 references | 0/1 passing
    public int ClientId { get; private set; }
    7 references | 0/1 passing
    public int PatientId { get; private set; }
    4 references | 0/2 passing
    public int RoomId { get; private set; }
    3 references
    public int? DoctorId { get; private set; }
    3 references
    public int AppointmentTypeId { get; private set; }
    12 references | 0/3 passing
    public DateTimeRange TimeRange { get; private set; }
    5 references
    public string Title { get; set; }
}
```

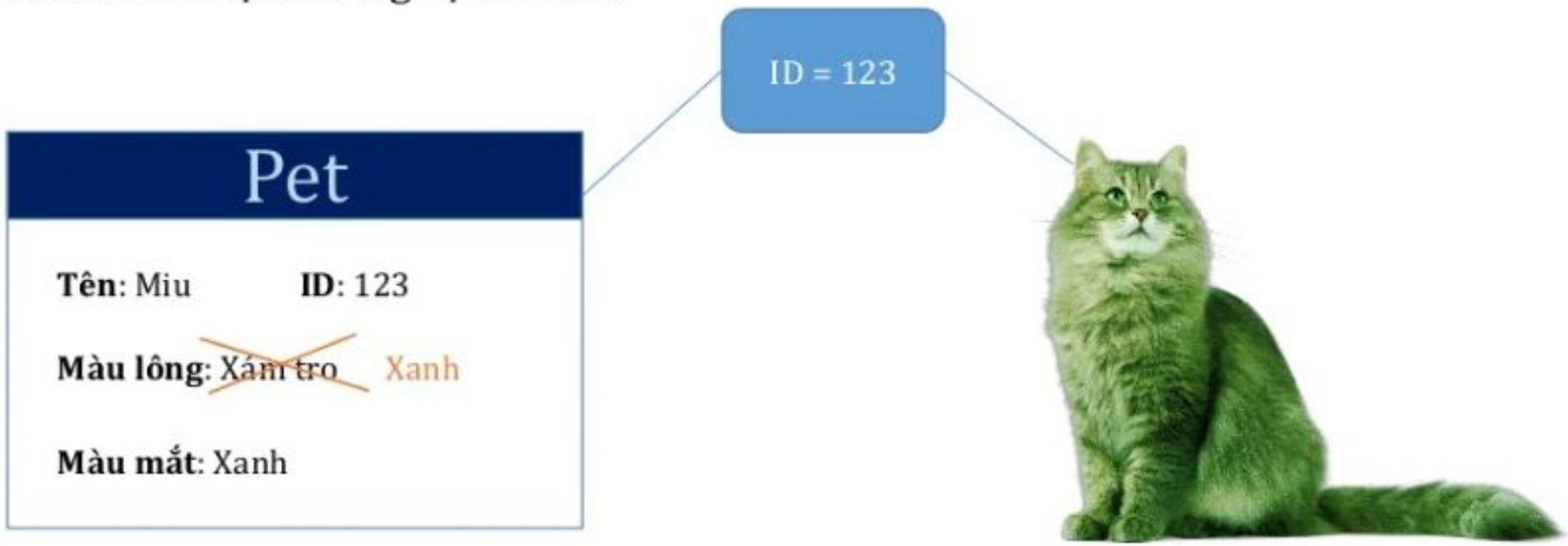
Rich model

```
public class Appointment : Entity<Guid>
{
    2 references
    public Guid ScheduleId { get; private set; }
    5 references | 0/2 passing
    public int ClientId { get; private set; }
    8 references | 0/2 passing
    public int PatientId { get; private set; }
    7 references | 0/3 passing
    public int RoomId { get; private set; }
    5 references | 0/2 passing
    public int? DoctorId { get; private set; }
    4 references | 0/1 passing
    public int AppointmentTypeId { get; private set; }
    14 references | 0/3 passing
    public DateTimeRange TimeRange { get; private set; }
    6 references | 0/1 passing
    public string Title { get; set; }

    More Properties

    4 references | 0/3 passing
    public void UpdateRoom(int newRoomId)...

    5 references | 0/4 passing
    public void UpdateTime(DateTimeRange newStartEnd)...

    5 references | 0/3 passing
    public void Confirm(DateTime dateConfirmed)...
```

Domain model

Entities và value object

- **Entities:** Xác định bằng định danh



Domain model

Entities và value object

- **Value Object:** Xác định bằng giá trị

Domain model

Domain Service

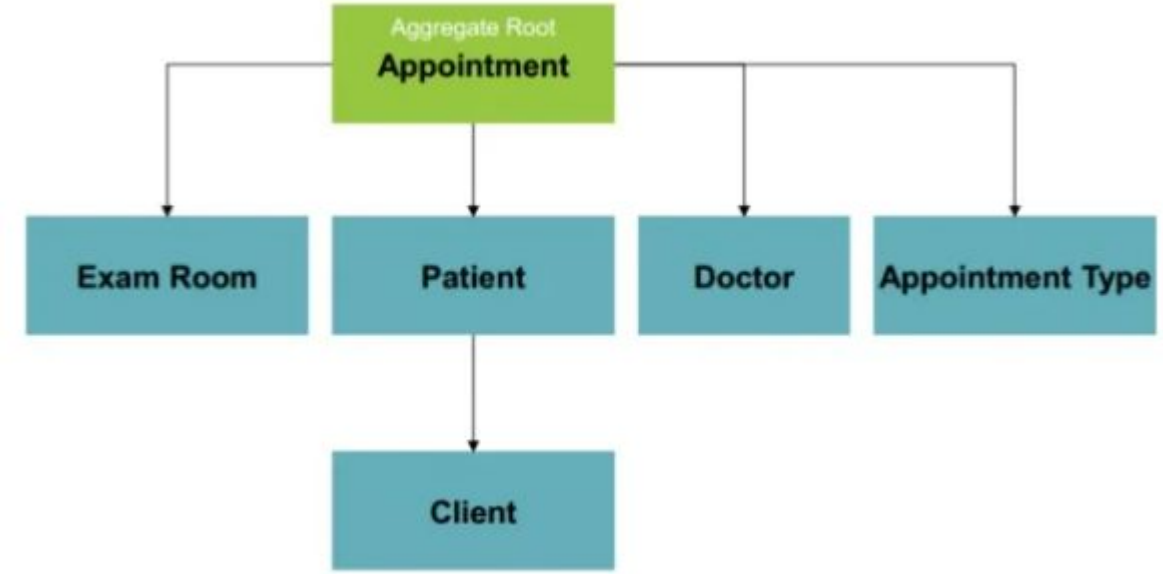
- Không thực sự nằm trong một entity hay value Object nào của domain
- Hoạt động trên nhiều đối tượng khác nhau

Domain model

Aggregate

- Là một nhóm các đối tượng dữ liệu được đối xử như một thể thống nhất trong hệ thống

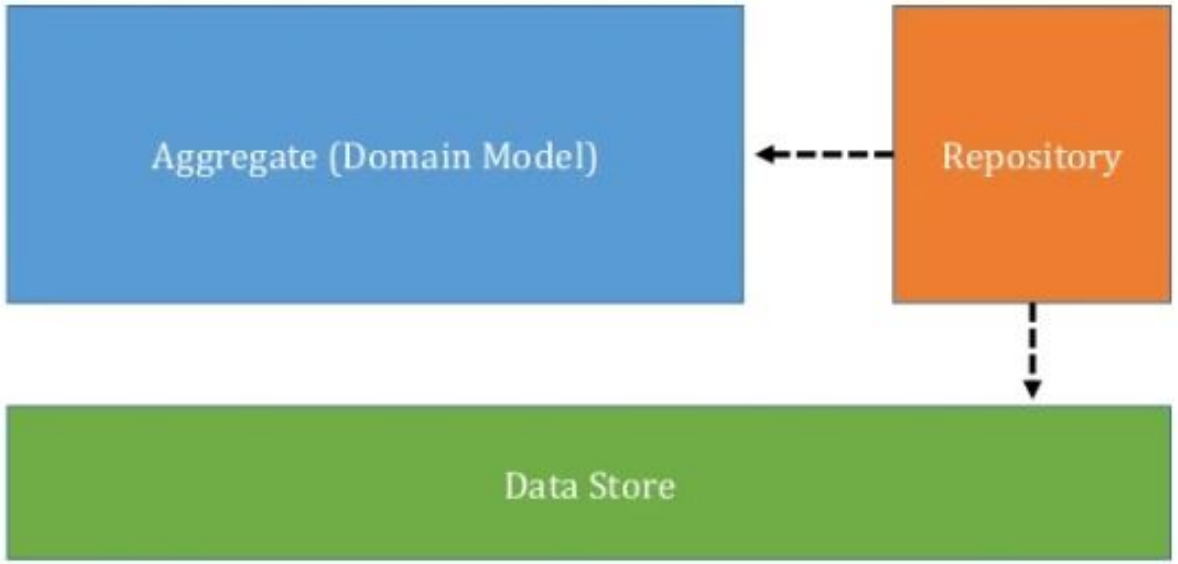
- Vd: User aggregate phải có profile,address,role
- Order aggregate phải có các order item và định dạng là order_id



Domain model

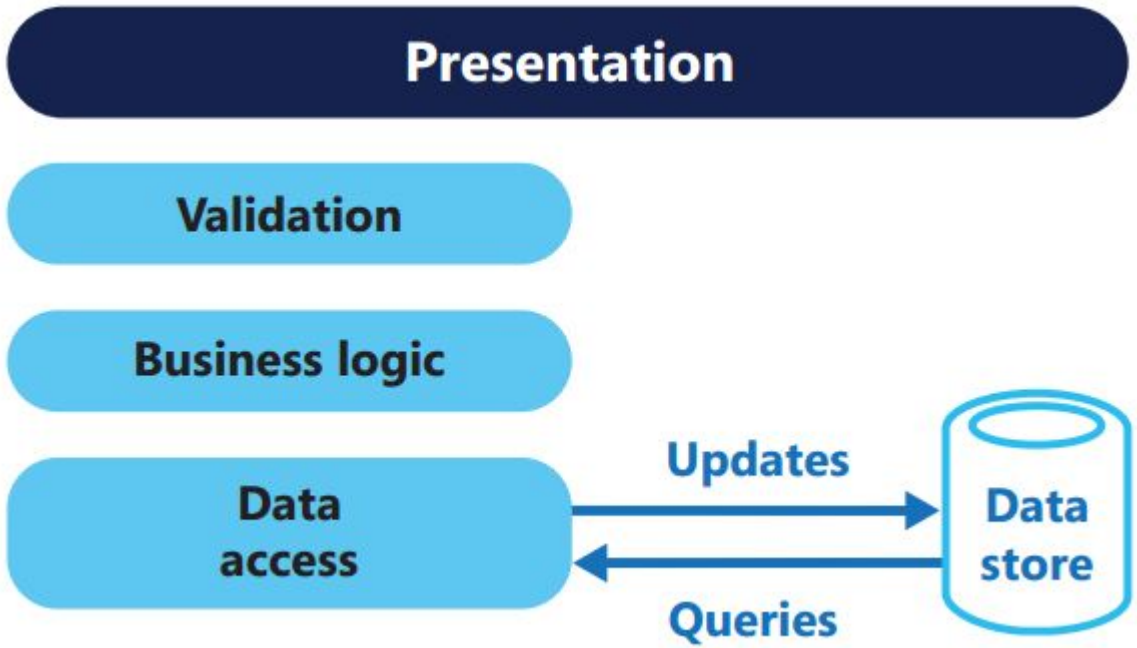
Aggregate

- **Repository:** Repository đảm nhiệm việc lấy ra những đối tượng đã được lưu trữ.



CQRS

Mô hình CRUD



CQRS

Mô hình CRUD

- Có ba tầng chính: Data Layer, Business Layer, Presentation
- DAL query trả lại DTO để Presentation hiển thị
- Presentation thay đổi DTO qua DAL lưu vào vào database
- Model dùng chung cả việc đọc và ghi dữ liệu

CQRS

Mô hình CRUD

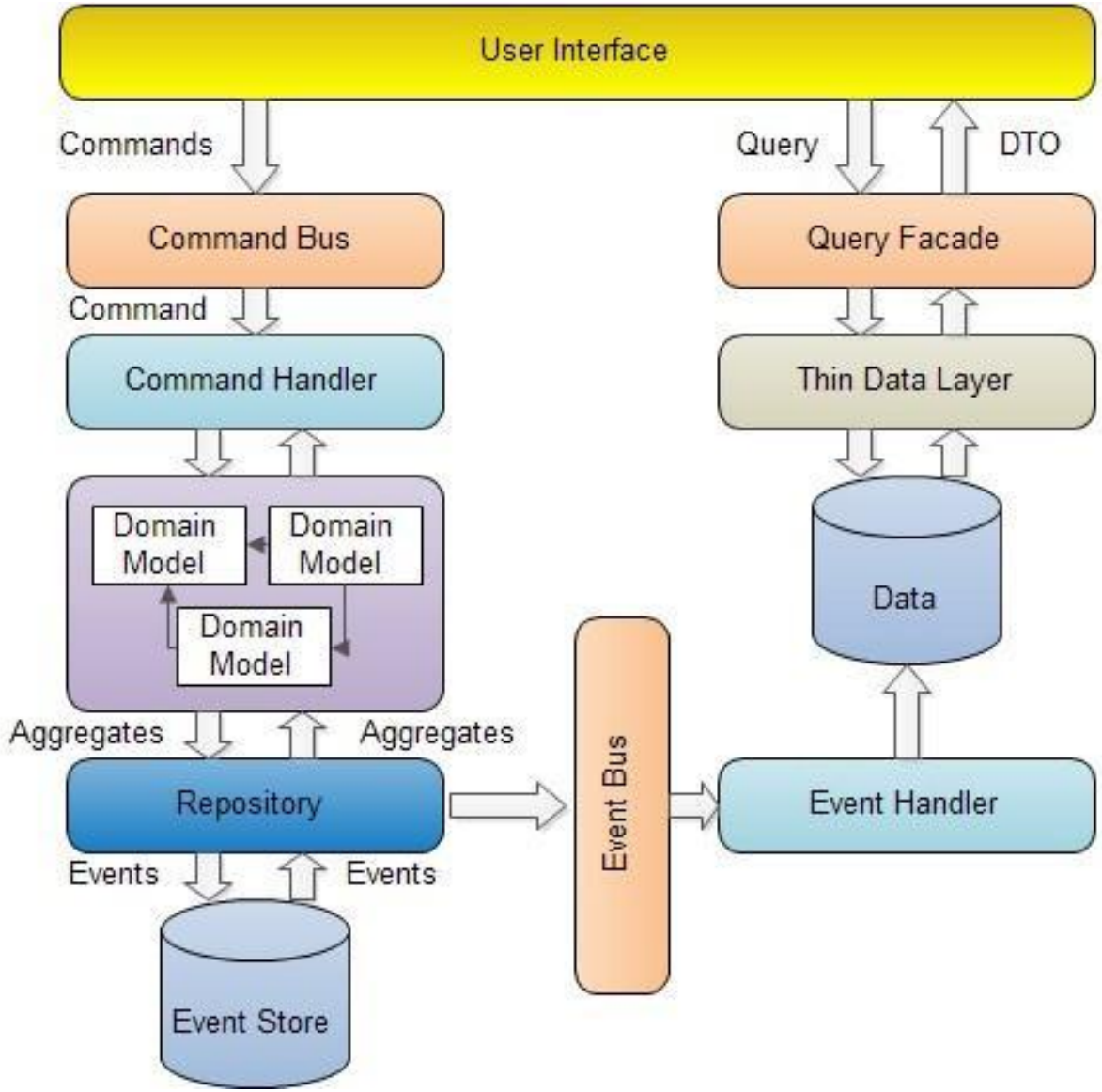
- Ưu điểm: nhanh, không cầu kì thiết kế
- Nhược điểm:
- Model phức tạp do dùng chung cho cả đọc và ghi dữ liệu
- Tăng tranh chấp dữ liệu giữa đọc và ghi dữ liệu
- Không scale được độc lập yêu cầu hiệu năng cho cả phần đọc và ghi

CQRS

Mô hình CQRS

- Người đưa ra: Greg Young – 2006
- CQRS – Command and Query Responsibility Segregation
- Mục tiêu phân chia các luồng ghi và đọc dữ liệu ra riêng biệt
- Scale độc lập các thành phần đọc và ghi dữ liệu

CQRS



CQRS

Các thành phần chính

- Command: lệnh phát sinh xử lý một nghiệp vụ
- Event: báo một quá trình xử lý kết thúc
- Command Bus: đường vận chuyển lệnh command
- Event Bus: đường vận chuyển các event
- Domain Model: mô hình xử lý nghiệp vụ
- DTO: Data Transfer Object
- Query Façade: mô hình query dữ liệu

CQRS

Mô hình hoạt động

- Phát sinh command. Vd: đặt hàng – placeorder
- Command Bus gửi command
- Command Handler nhận command, thực thi xử lý nghiệp vụ tại domain model
- Phát sinh các Event báo kết thúc quá trình xử lý.
- Event Bus publish các event đi

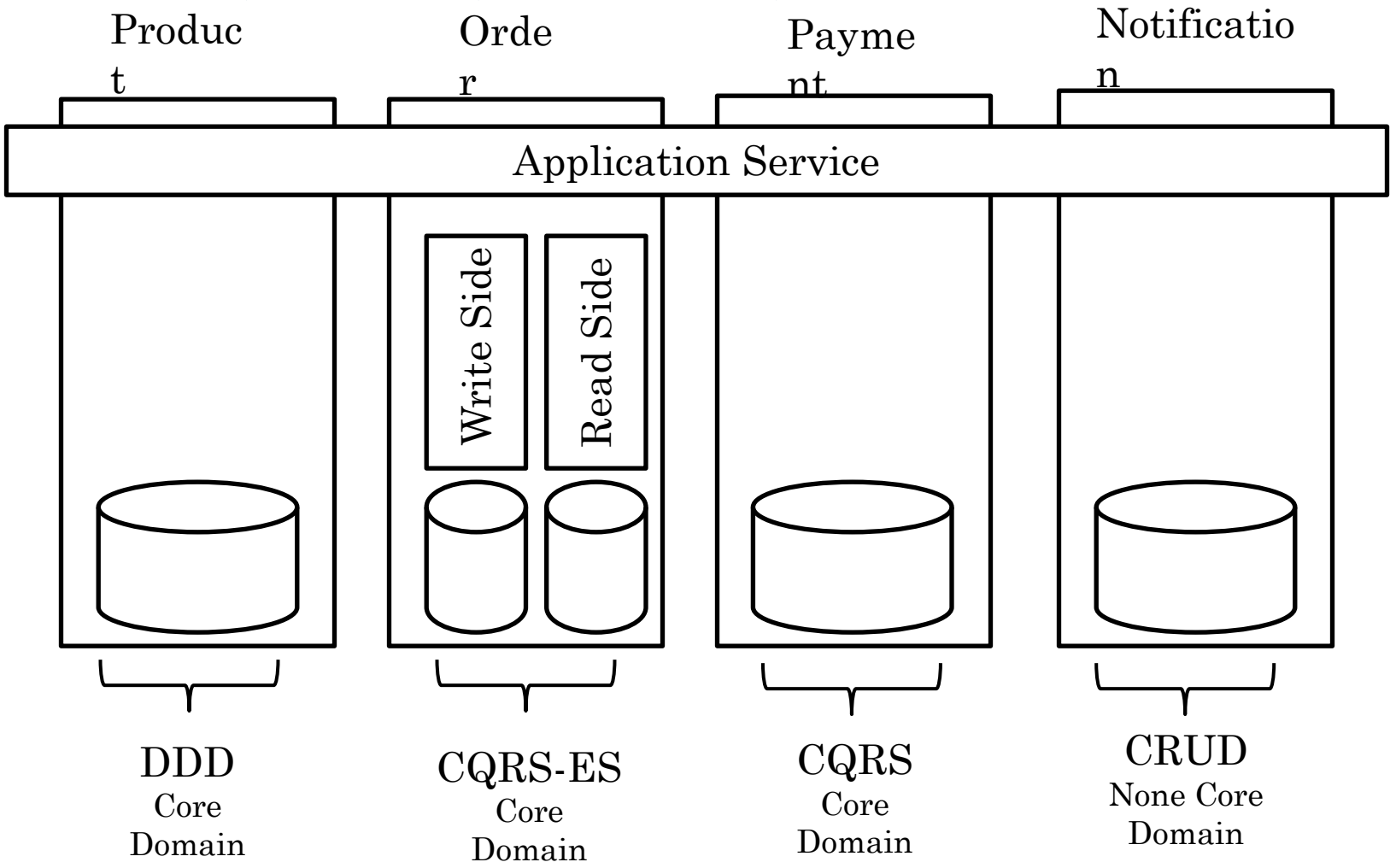
CQRS

Xét ví dụ cụ thể

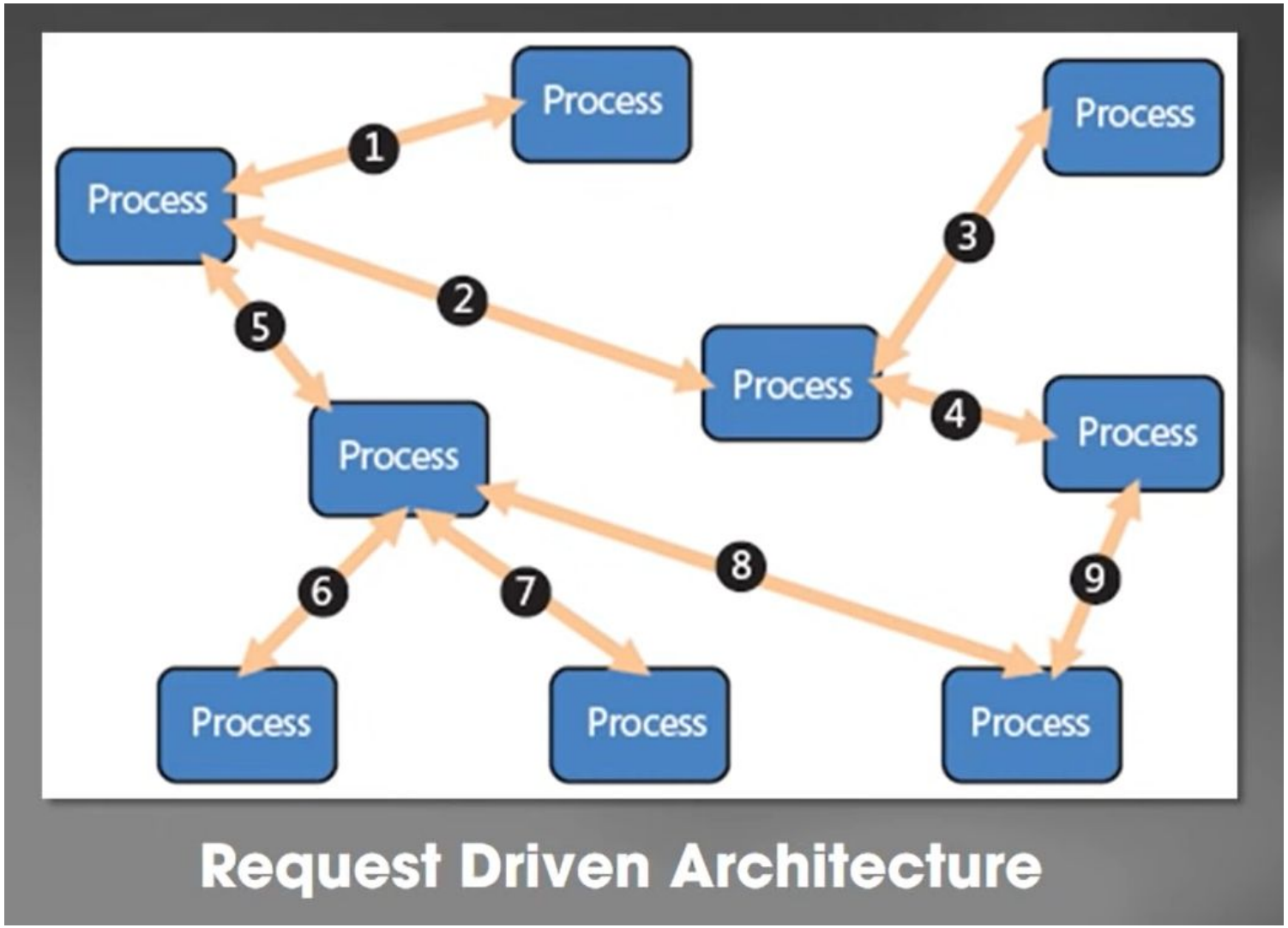
- Xây dựng ứng dụng thương mại điện tử đơn giản: cho phép người dùng xem và mua hàng
- Sau đặt hàng xong người dùng có thể thanh toán online
- Sau khi thanh toán online xong, đơn hàng được xác nhận thì gửi tin nhắn và email xác nhận cho người dùng

CQRS

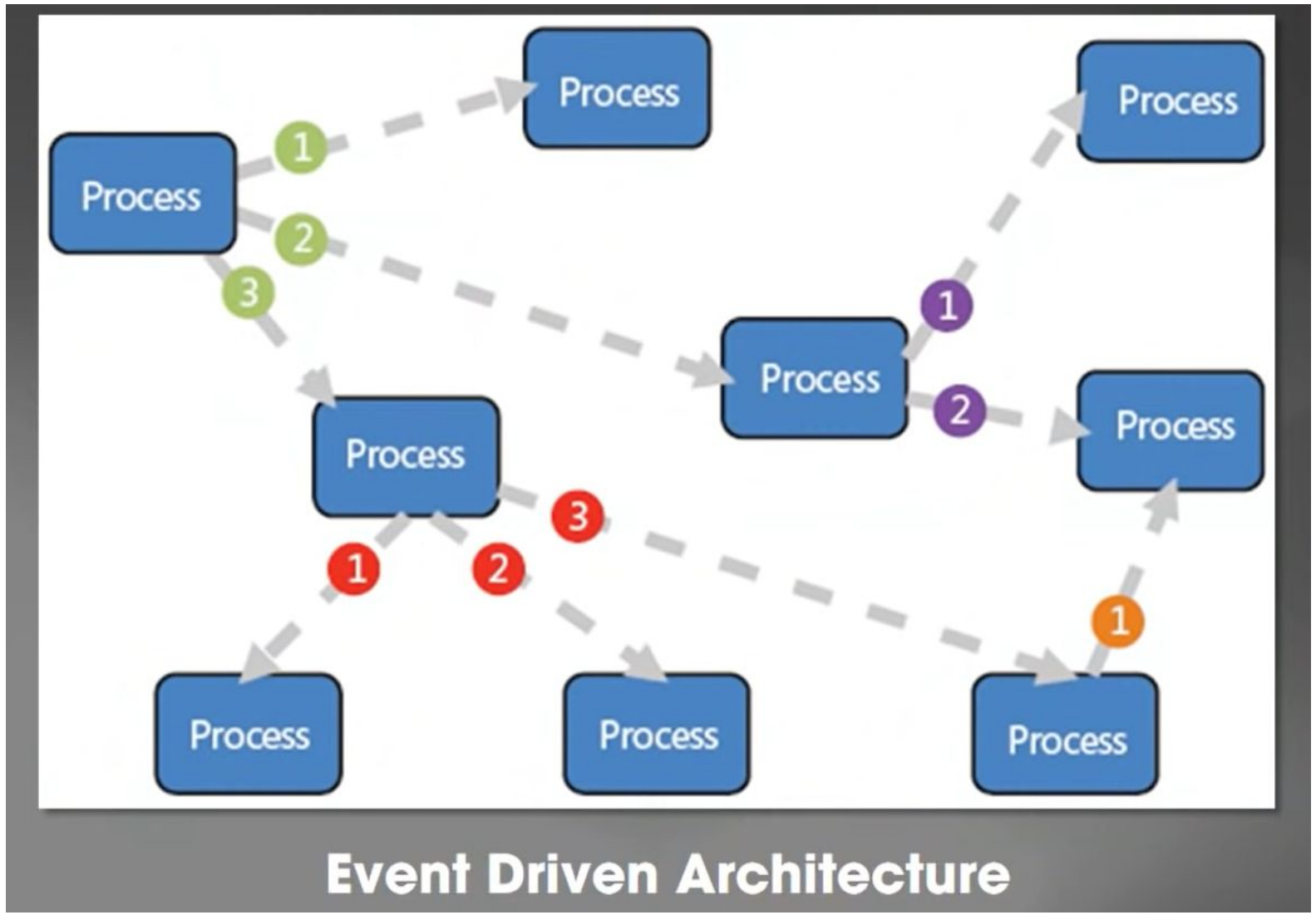
Product, Order, Payment, Notification



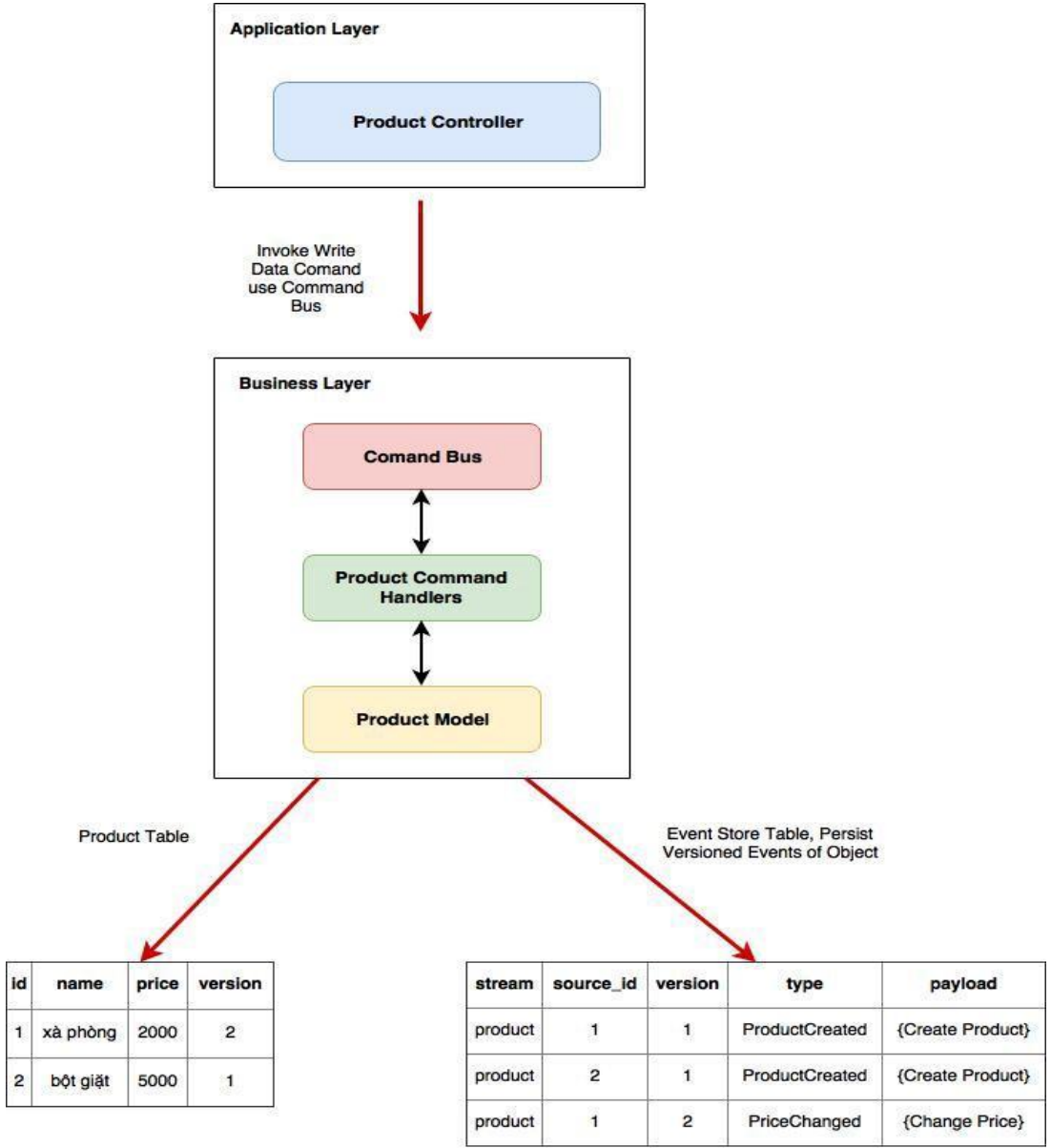
Event Driven Architecture



Event Driven Architectur



Event Sourcing



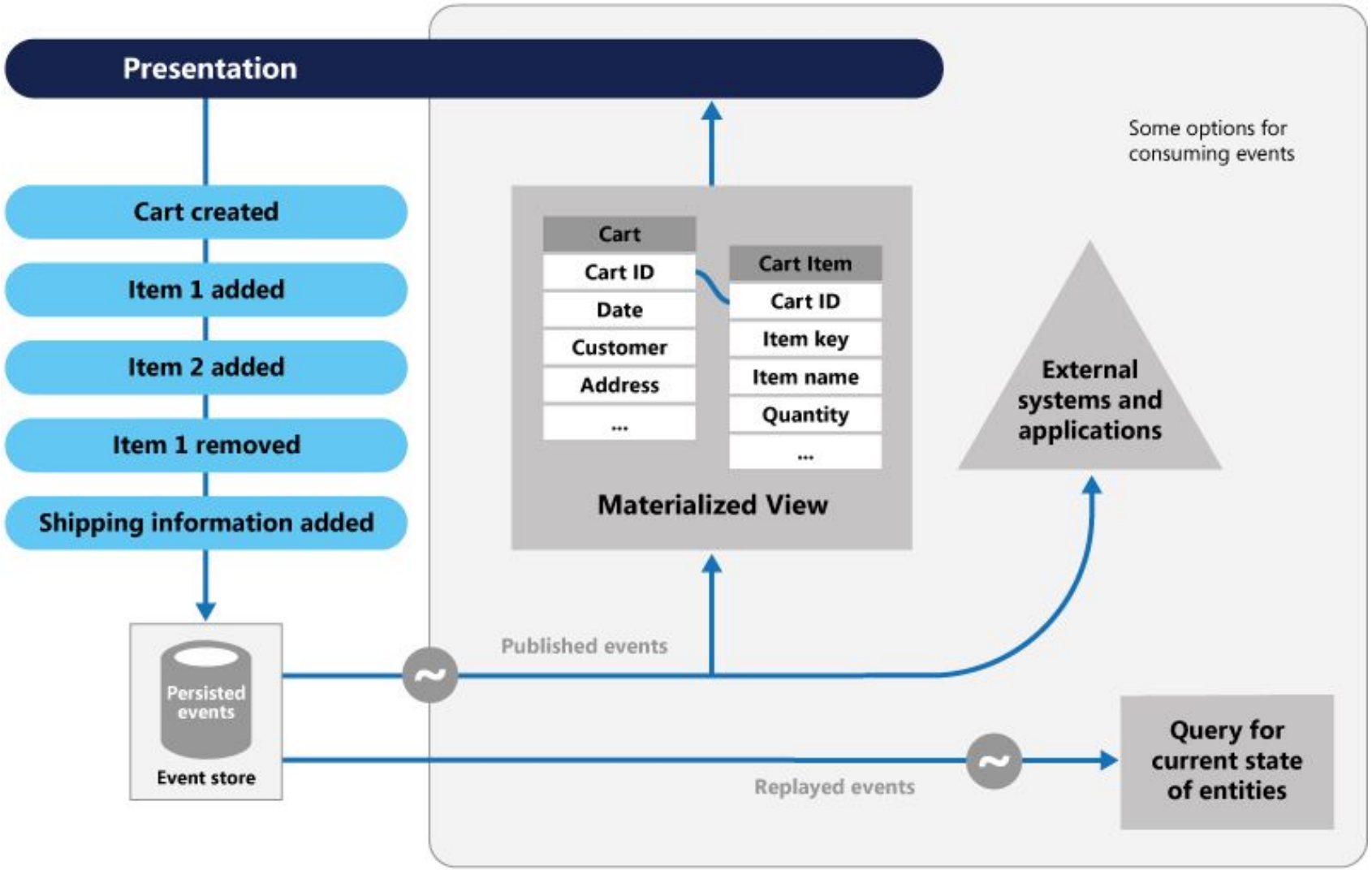
id	name	price	version
1	xà phòng	2000	2
2	bột giặt	5000	1

stream	source_id	version	type	payload
product	1	1	ProductCreated	{Create Product}
product	2	1	ProductCreated	{Create Product}
product	1	2	PriceChanged	{Change Price}

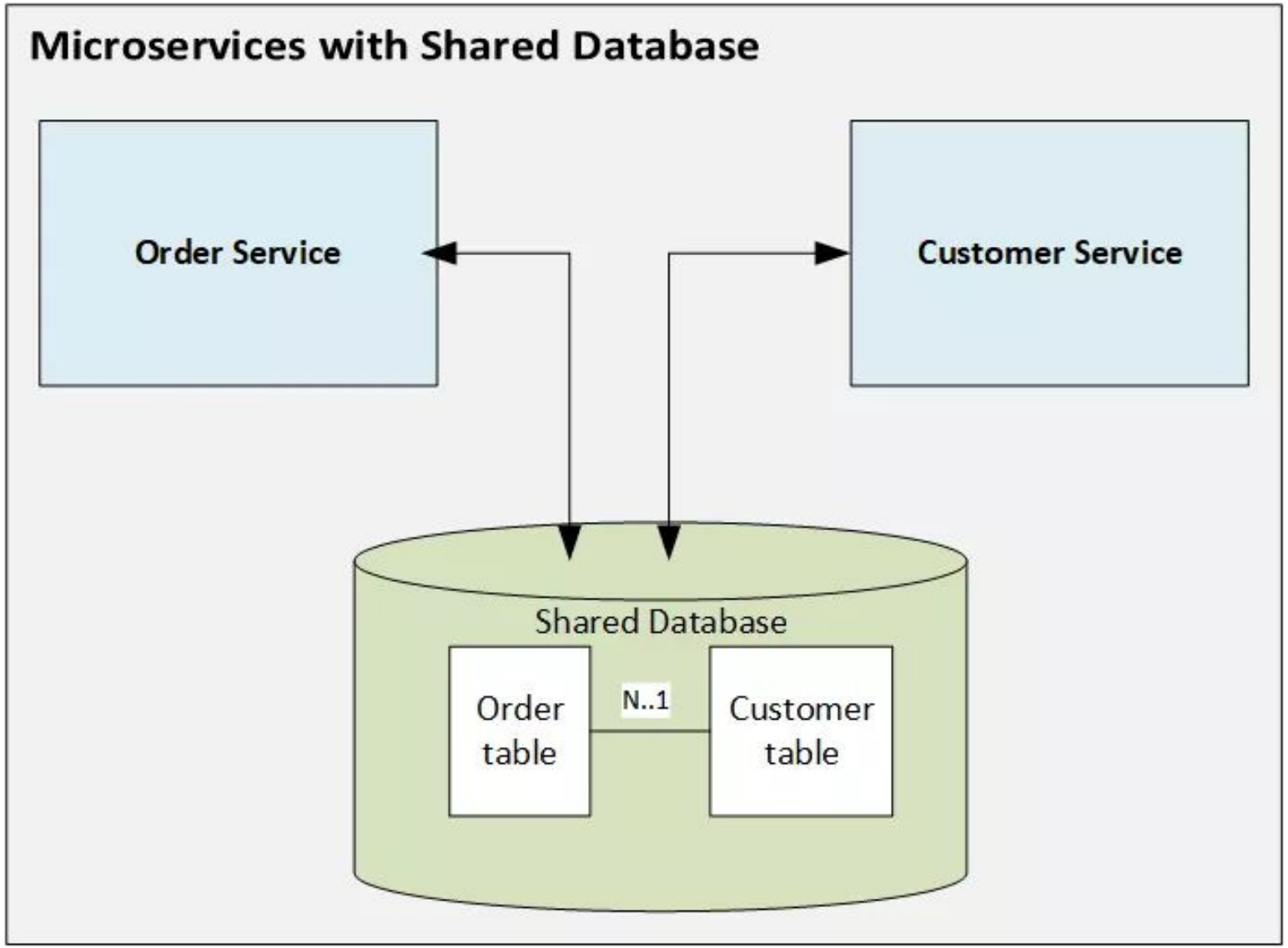
Mô hình Event Sourcing—ES, là mô hình thiết kế mà trạng thái của object sẽ được lưu trữ dưới dạng chuỗi các sự kiện thay đổi

- Lưu trữ được lịch sử thay đổi của các đối tượng.

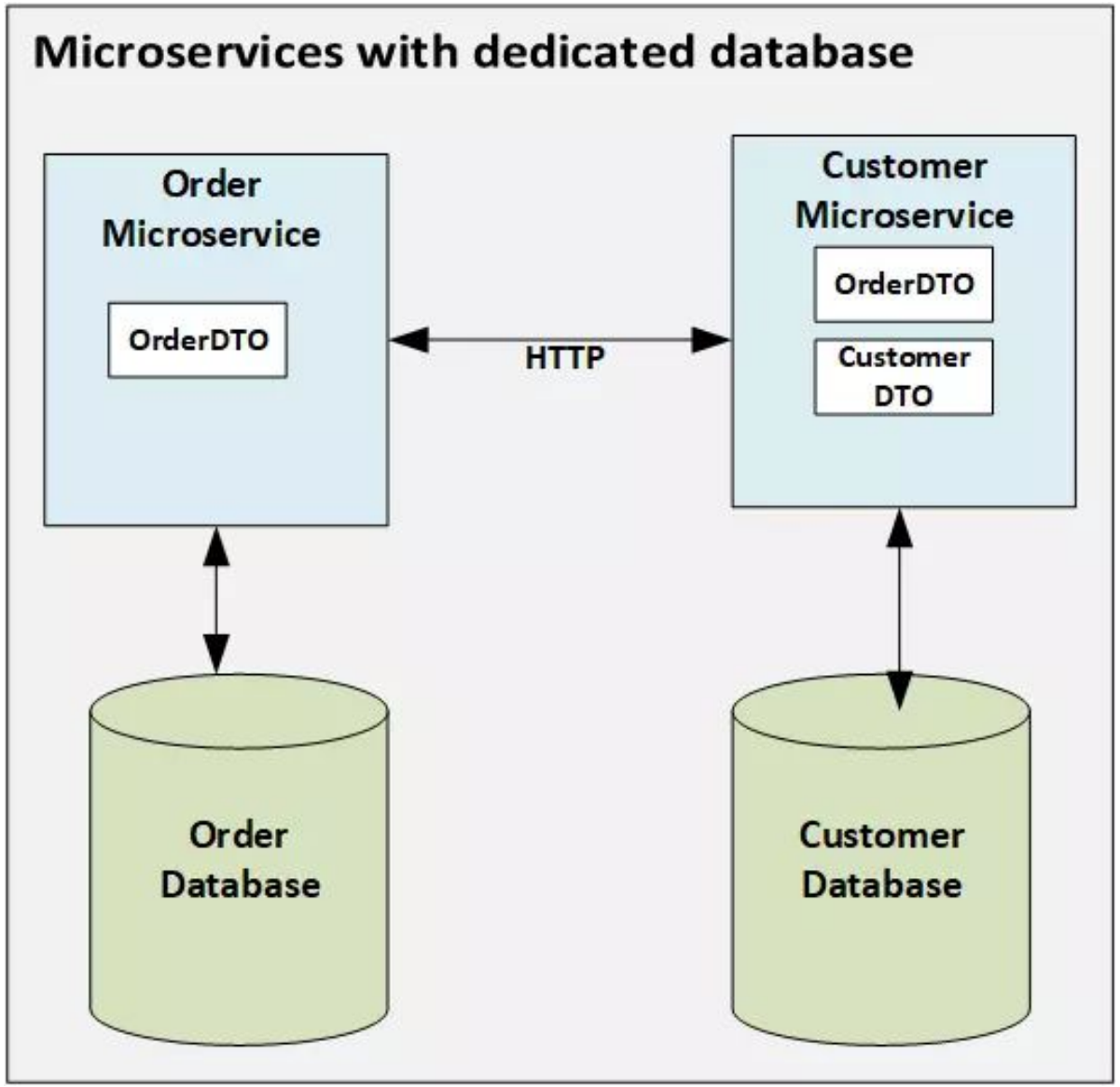
Event Sourcing



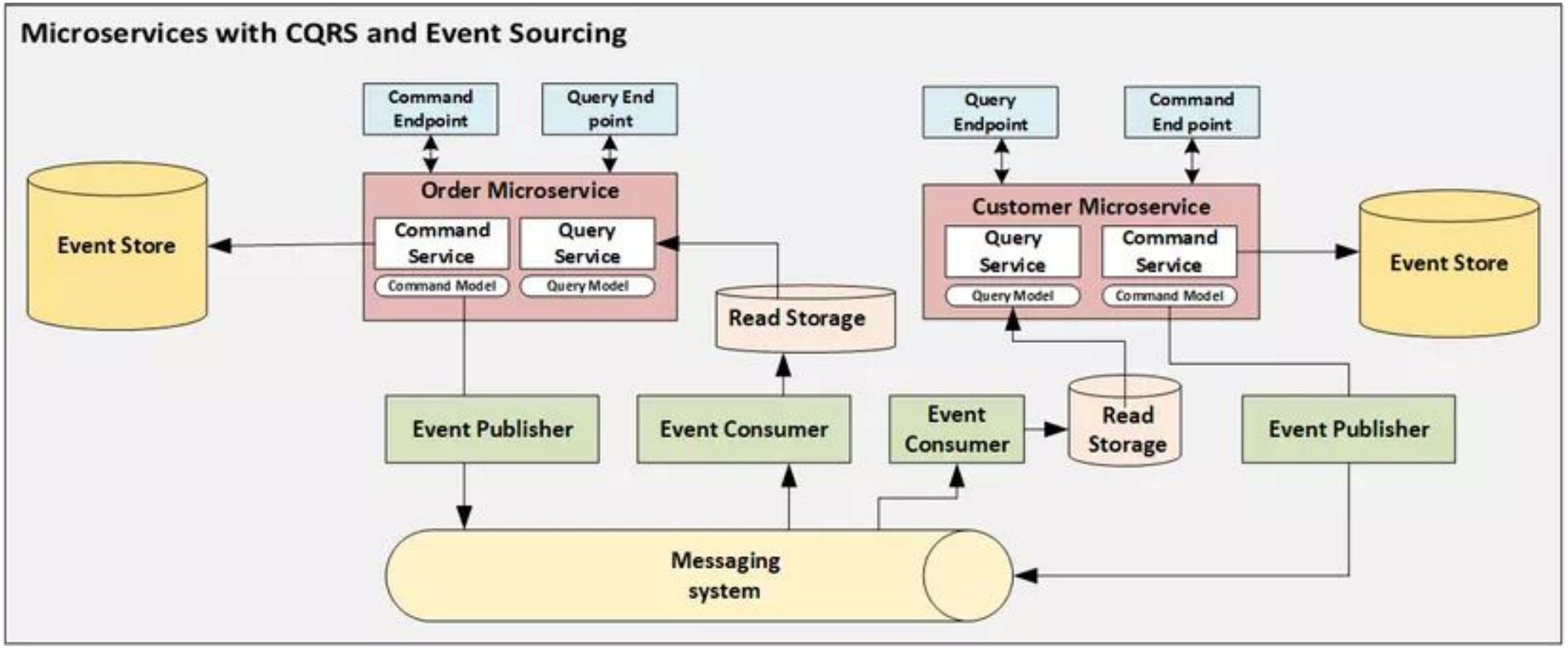
Event Sourcing



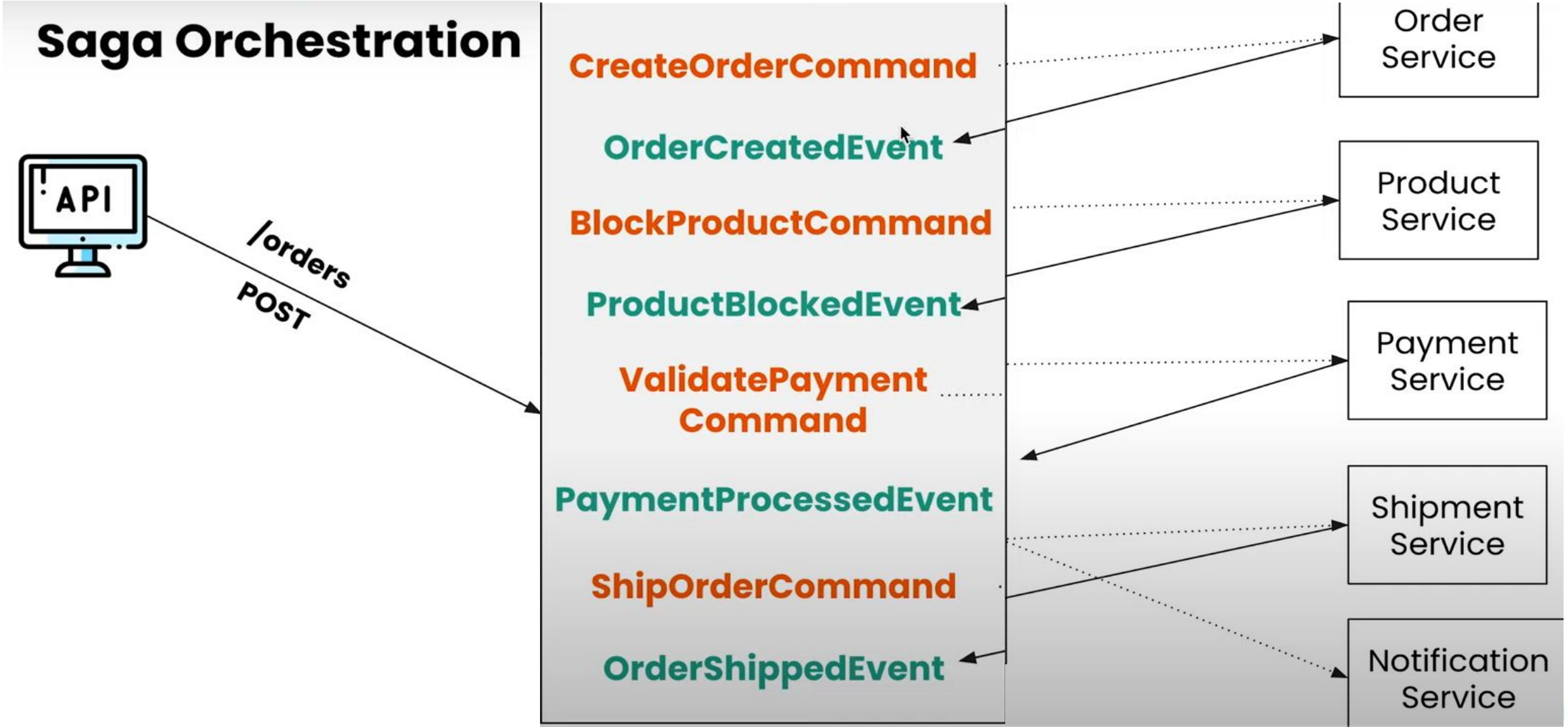
Event Sourcing



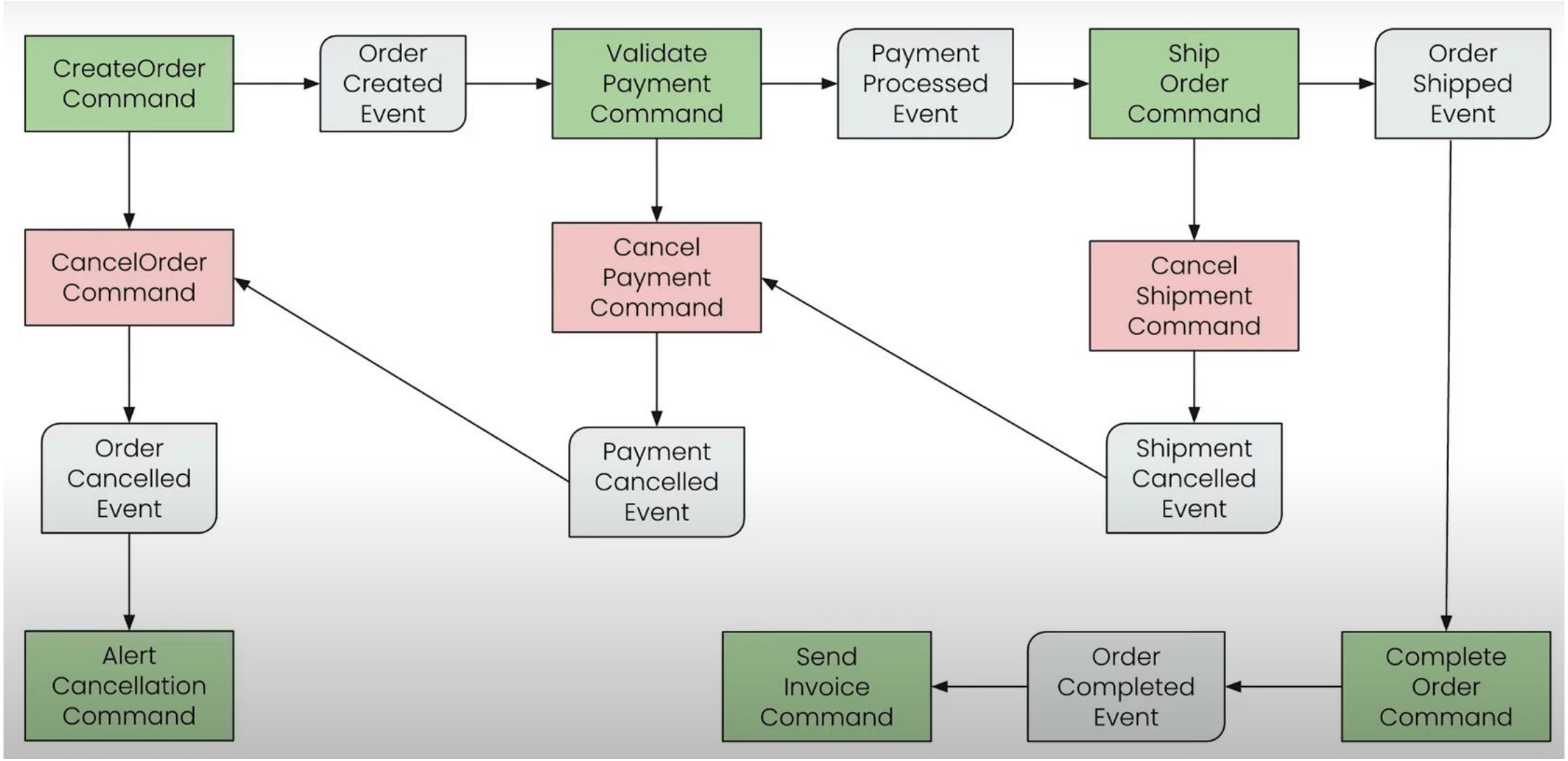
Event Sourcing



Event Sourcing



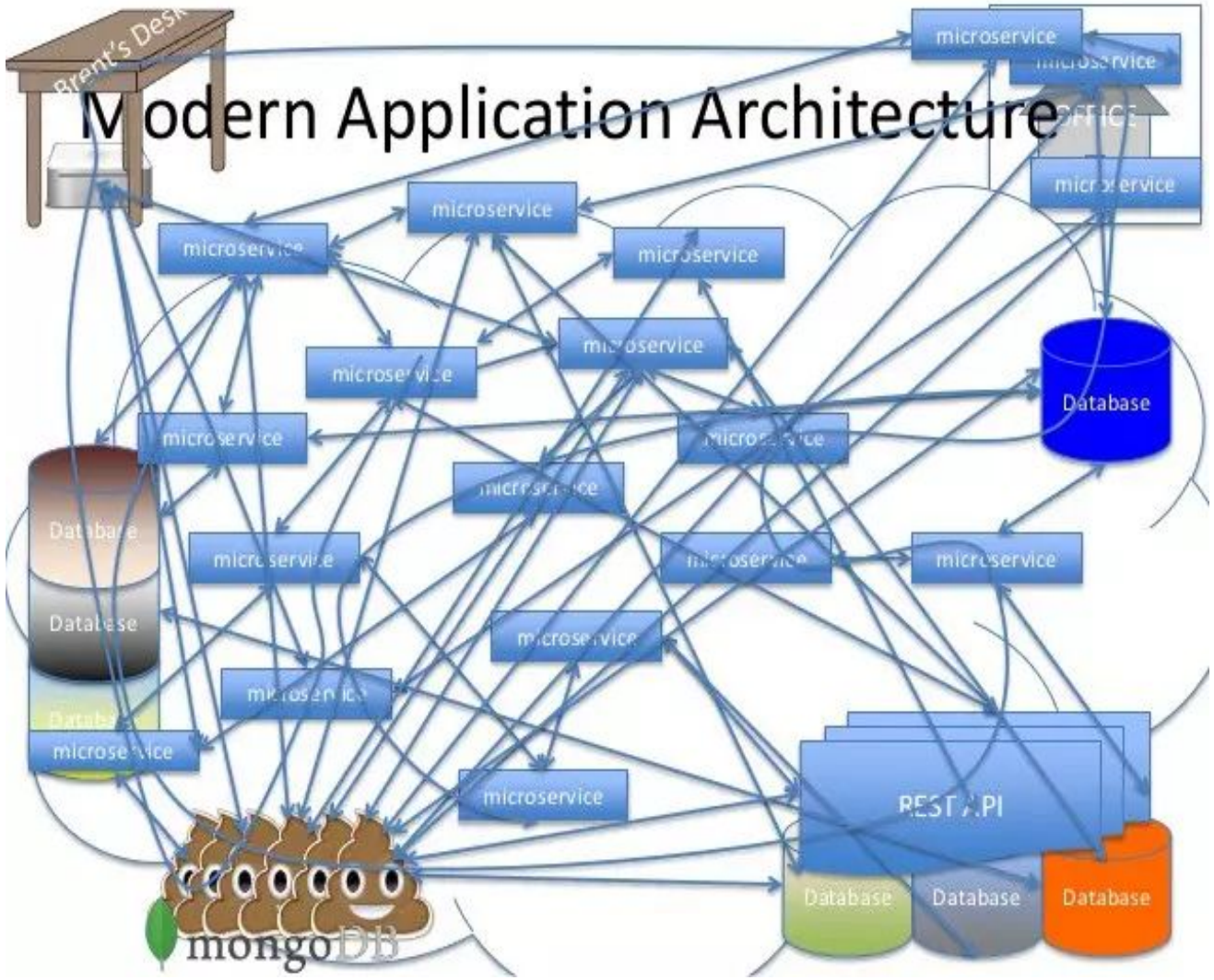
Event Sourcing



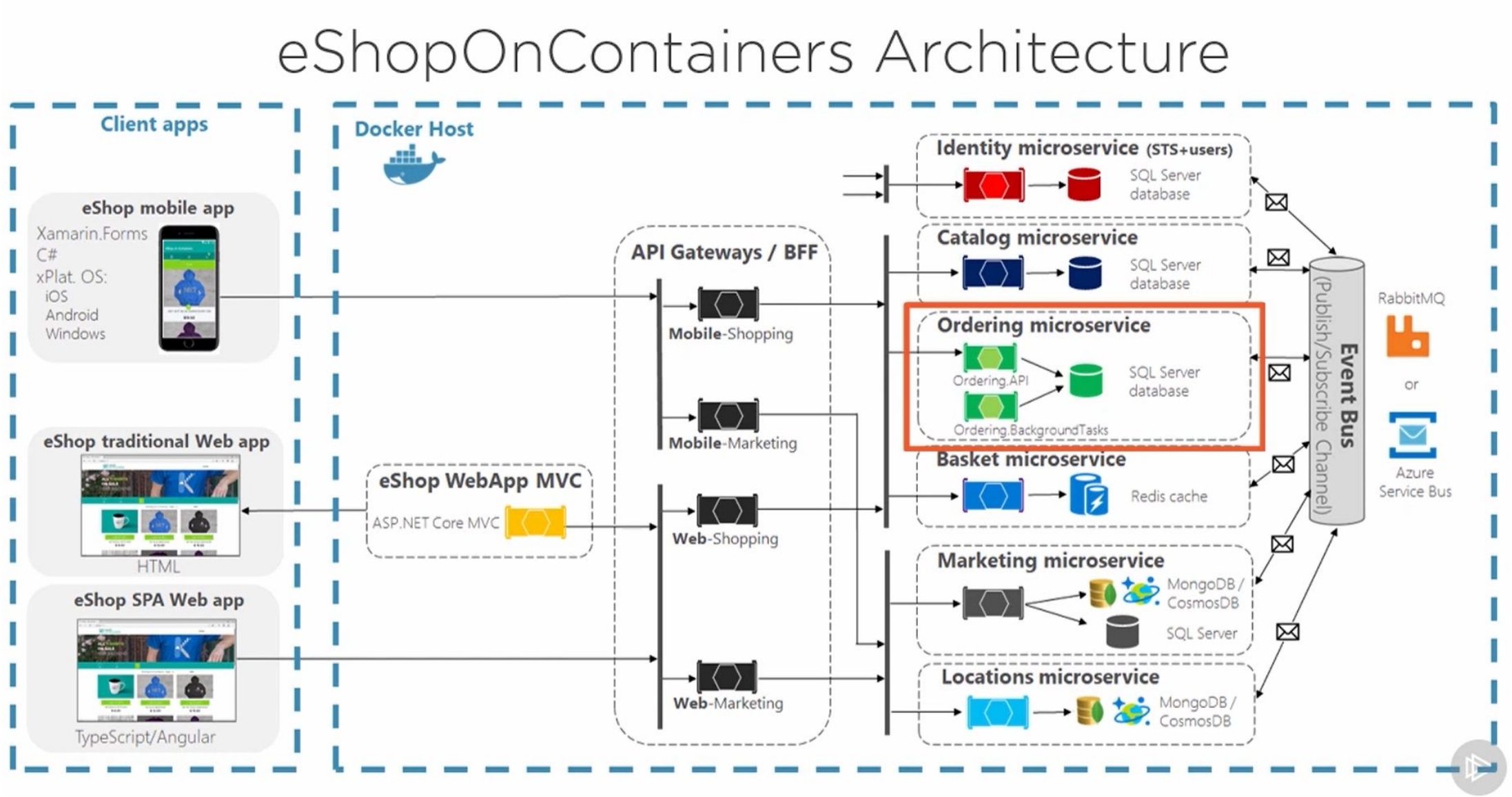
Xây dựng event sourcing với Axon framework

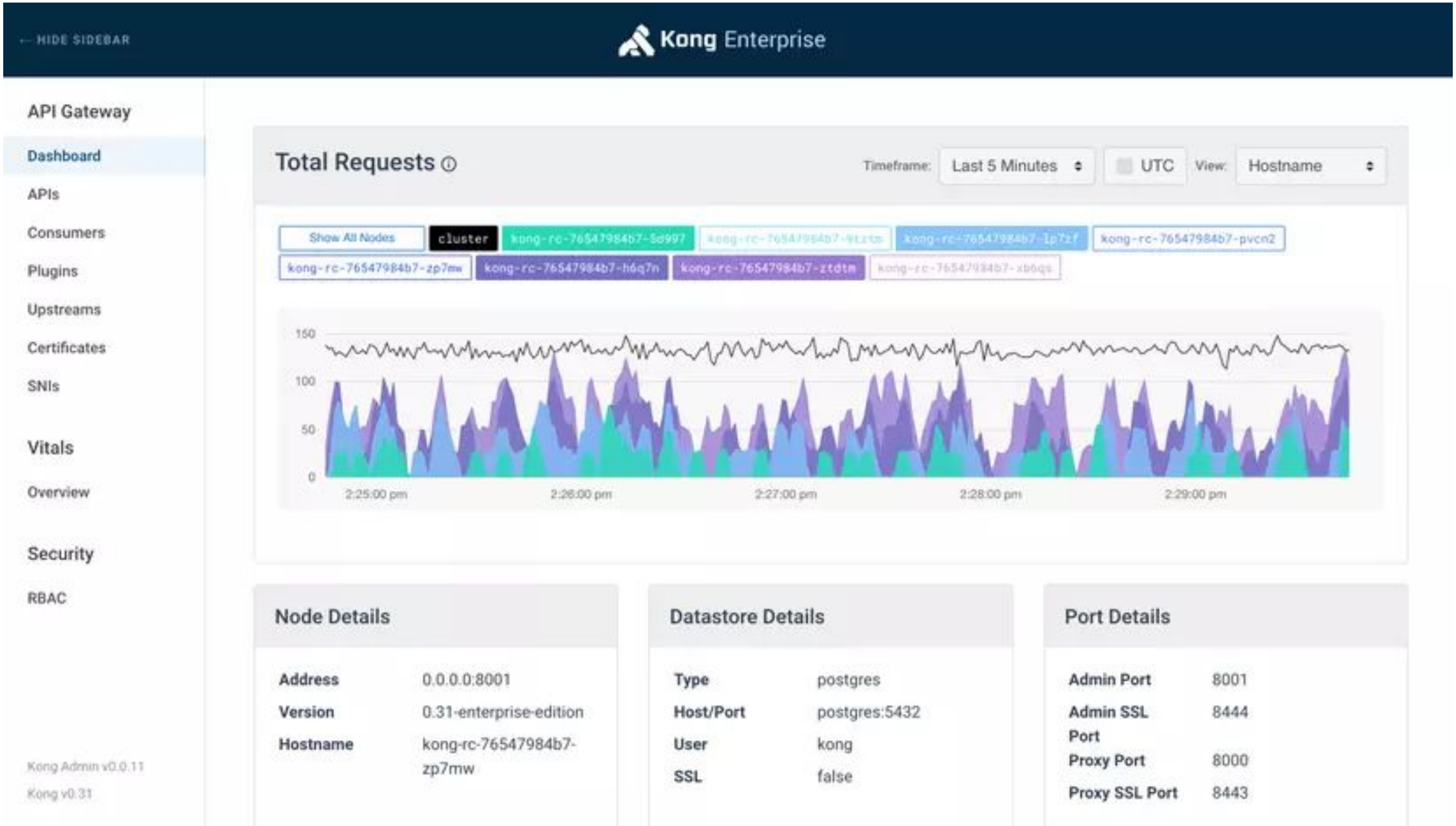
1. Demo Event Sourcing
2. Demo Event Sourcing with Saga

What is API Gateway

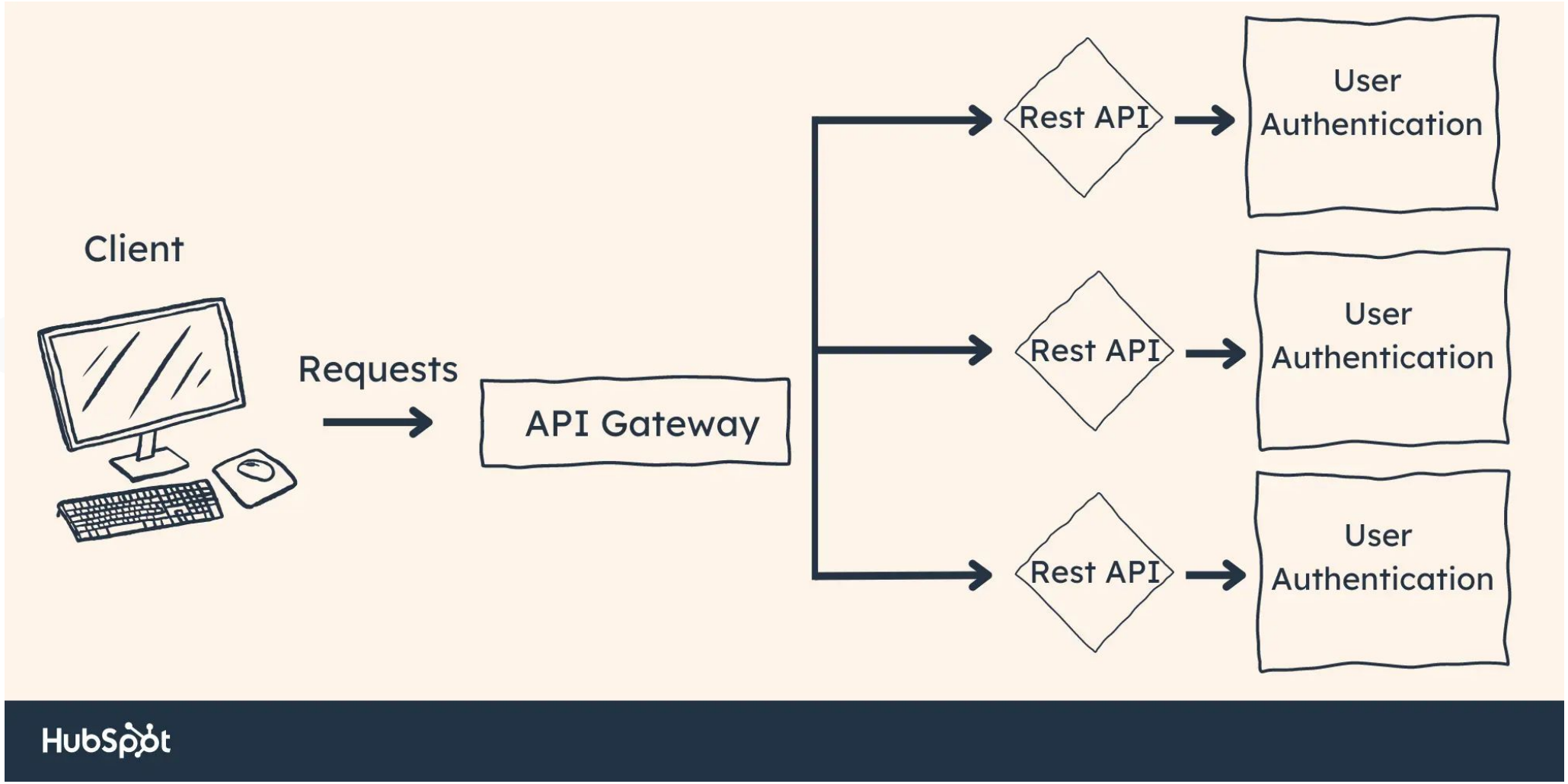


What is API Gateway





Load balancing



Authentication

JWT

Verify and authenticate JSON Web Tokens

Support by:
Kong Inc.

Key Authentication

Add key authentication to your Services

Support by:
Kong Inc.

Enterprise

LDAP Authentication Advanced

Secure Kong clusters, routes and services with username and password protection

Support by:
Kong Inc.

LDAP Authentication

Integrate Kong with a LDAP server

Support by:
Kong Inc.

Enterprise

OAuth 2.0 Introspection

Integrate Kong with a third-party OAuth 2.0 Authorization Server

Support by:
Kong Inc.

OAuth 2.0 Authentication

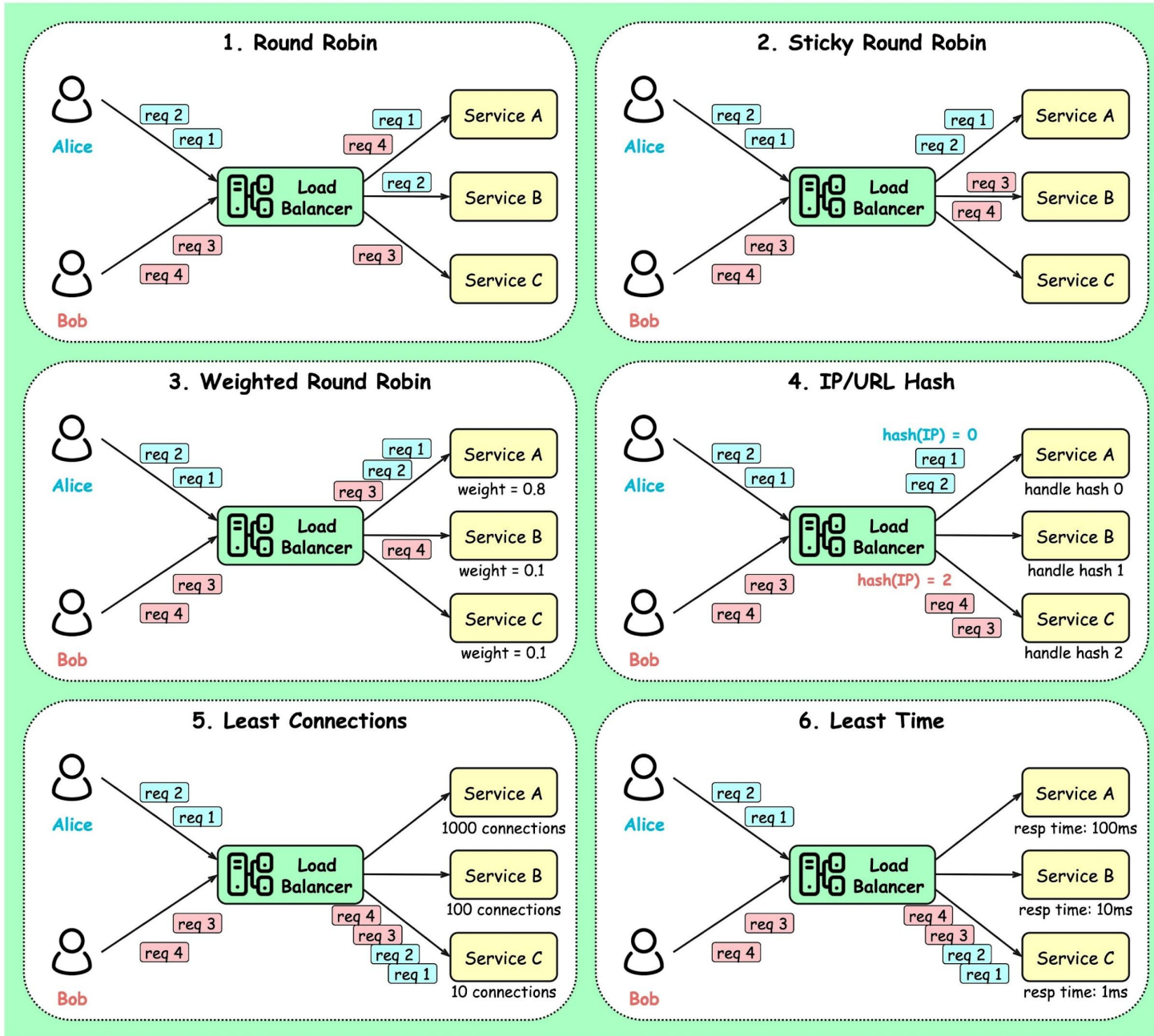
Add OAuth 2.0 authentication to your Services

Support by:
Kong Inc.

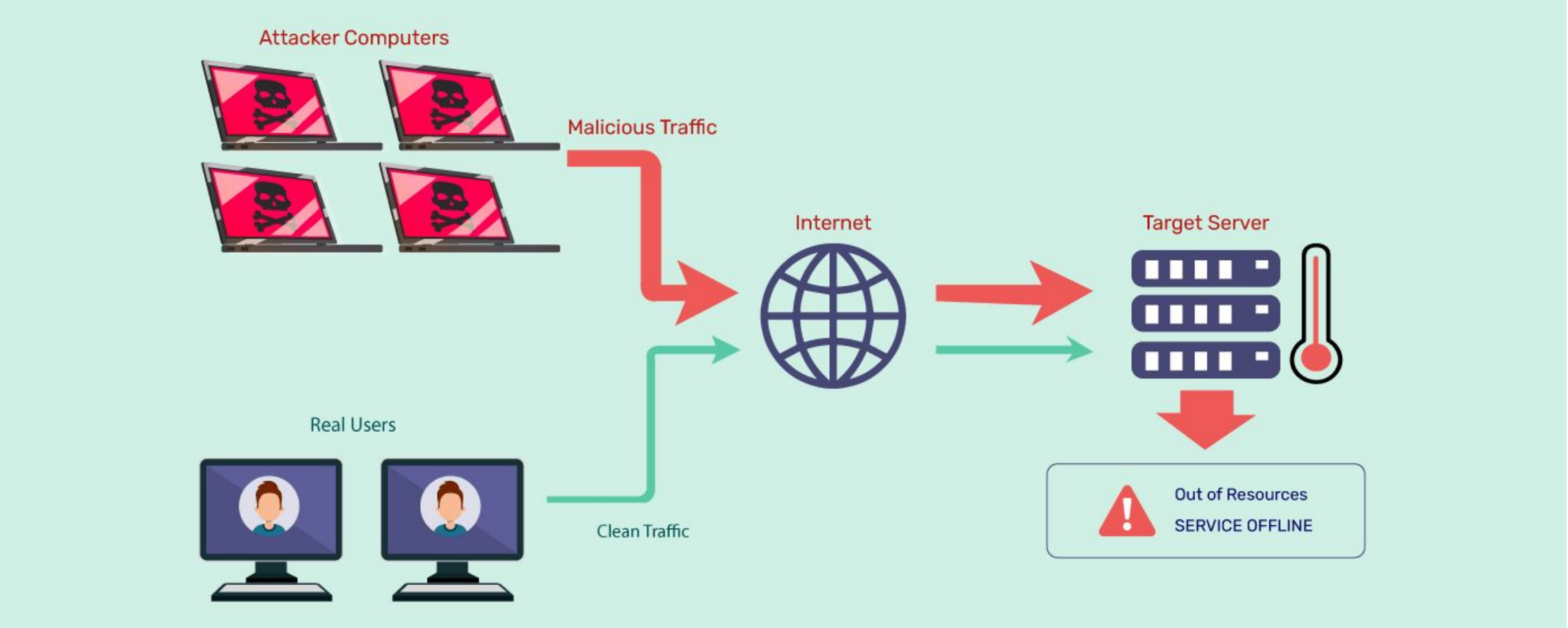
Nhược điểm API Gateway

Load Balancing Algorithms

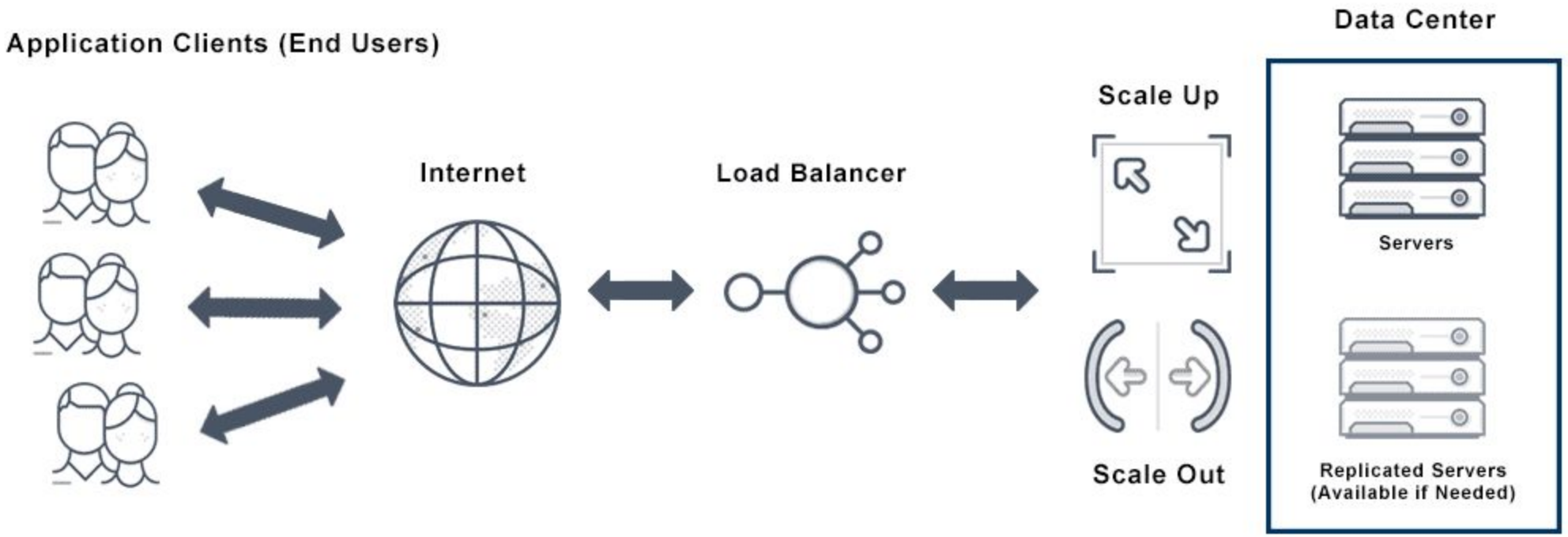
blog.bytebytego.com



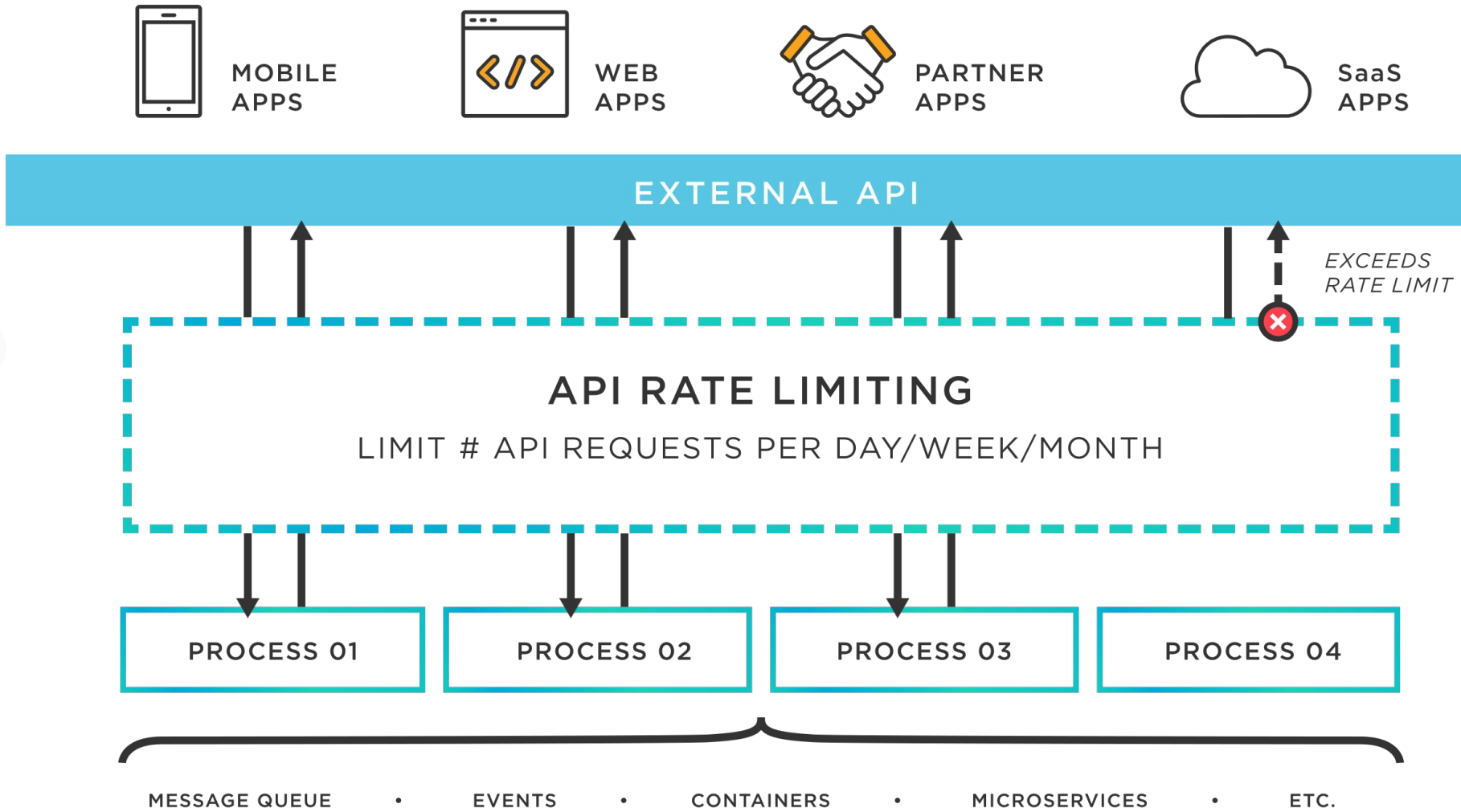
Why we need Rate limiting



Auto Scale ?



Rate limiting

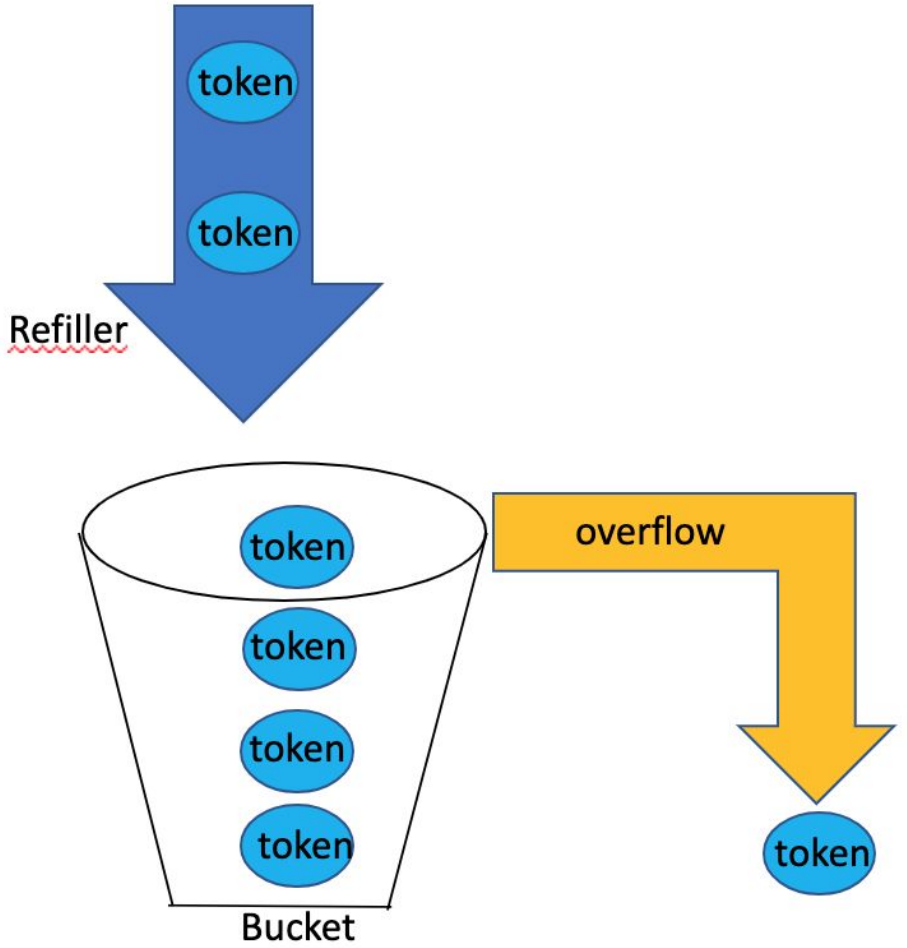


Throttling là gì ?

- User-Level Rate Limiting
- Server-Level Rate Limiting
- Geography-Based Rate Limiting

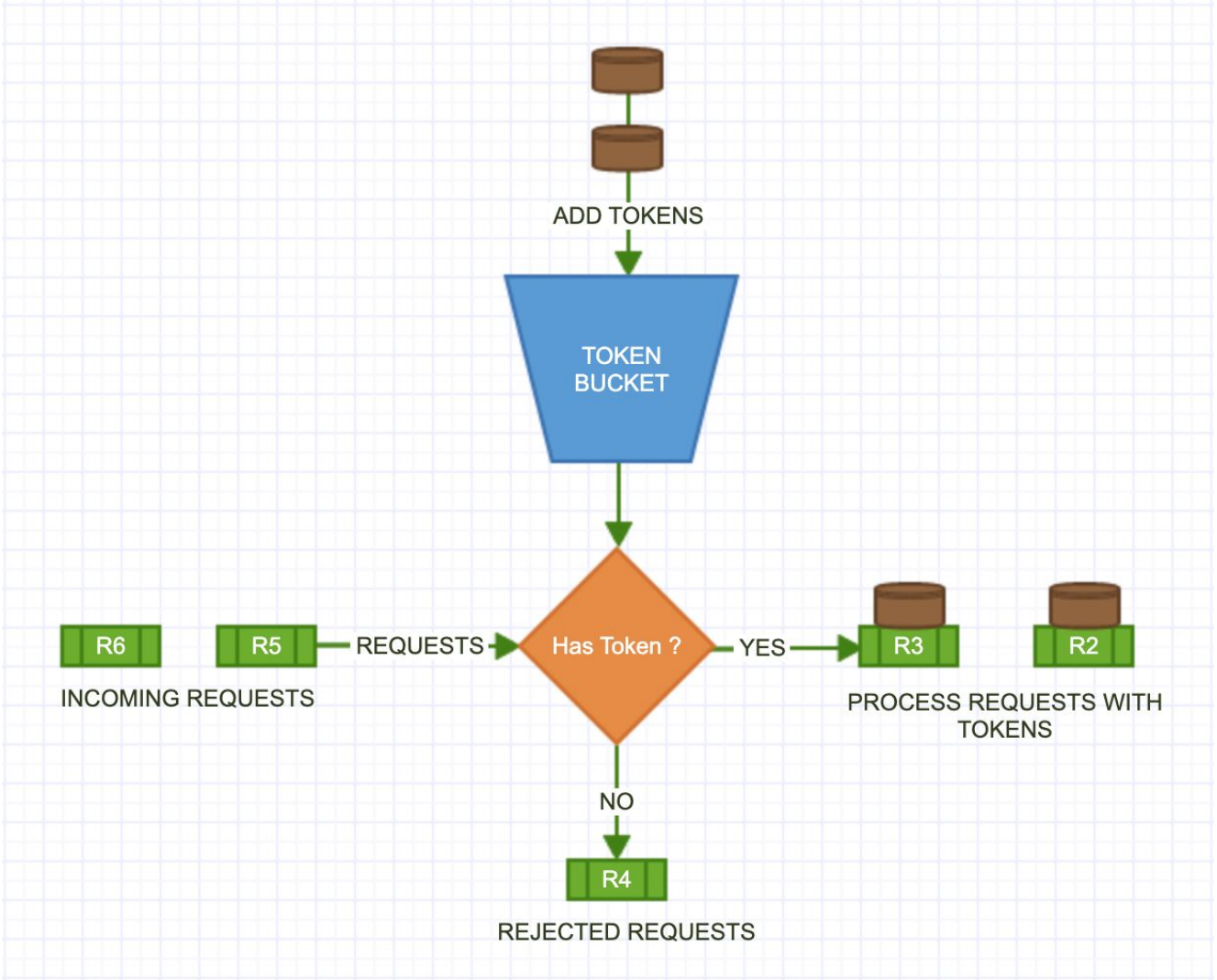
What Are the Algorithms Used for Rate Limiting?

Thuật toán Token Bucket



What Are the Algorithms Used for Rate Limiting?

Thuật toán Token Bucket



What Are the Algorithms Used for Rate Limiting?

Thuật toán Token Bucket

- **Ví dụ:**
  Trong ví dụ bên dưới, mỗi người dùng được phép gửi tối đa 3 yêu cầu mỗi phút.

- Người dùng 1 thực hiện yêu cầu đầu tiên lúc **10:00:00** ; số token khả dụng là 3, do đó yêu cầu được chấp thuận và số lượng token khả dụng giảm xuống còn 2.

- Vào lúc **10:00:10** , yêu cầu thứ hai của người dùng, mã thông báo khả dụng là 2, yêu cầu được chấp nhận và số lượng mã thông báo giảm xuống còn 1.

- Yêu cầu thứ ba đến lúc **10:00:35** , với số lượng mã thông báo là 1, do đó yêu cầu được chấp thuận và giảm đi.

- Yêu cầu thứ 4 sẽ đến lúc **10:00:45** và số lượng khả dụng đã là 0, do đó API bị từ chối.

- Bây giờ, vào lúc **10:01:00** , số lượng sẽ được làm mới với mã thông báo khả dụng

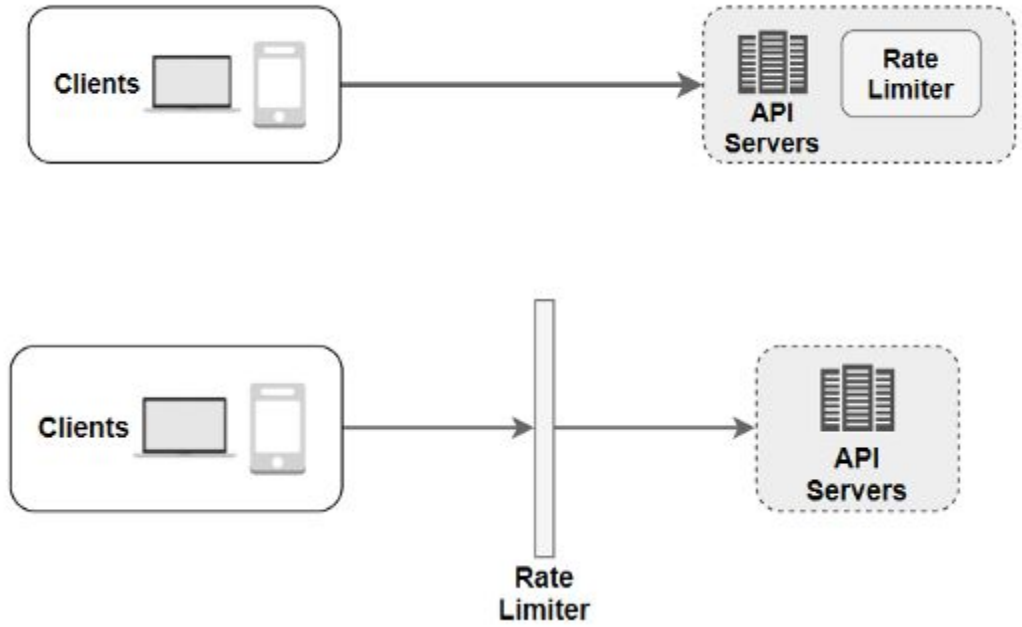
What Are the Algorithms Used for Rate Limiting?

- Leaky Bucket algorithm
- Fixed Window Counter algorithm
- Sliding Window Logs algorithm

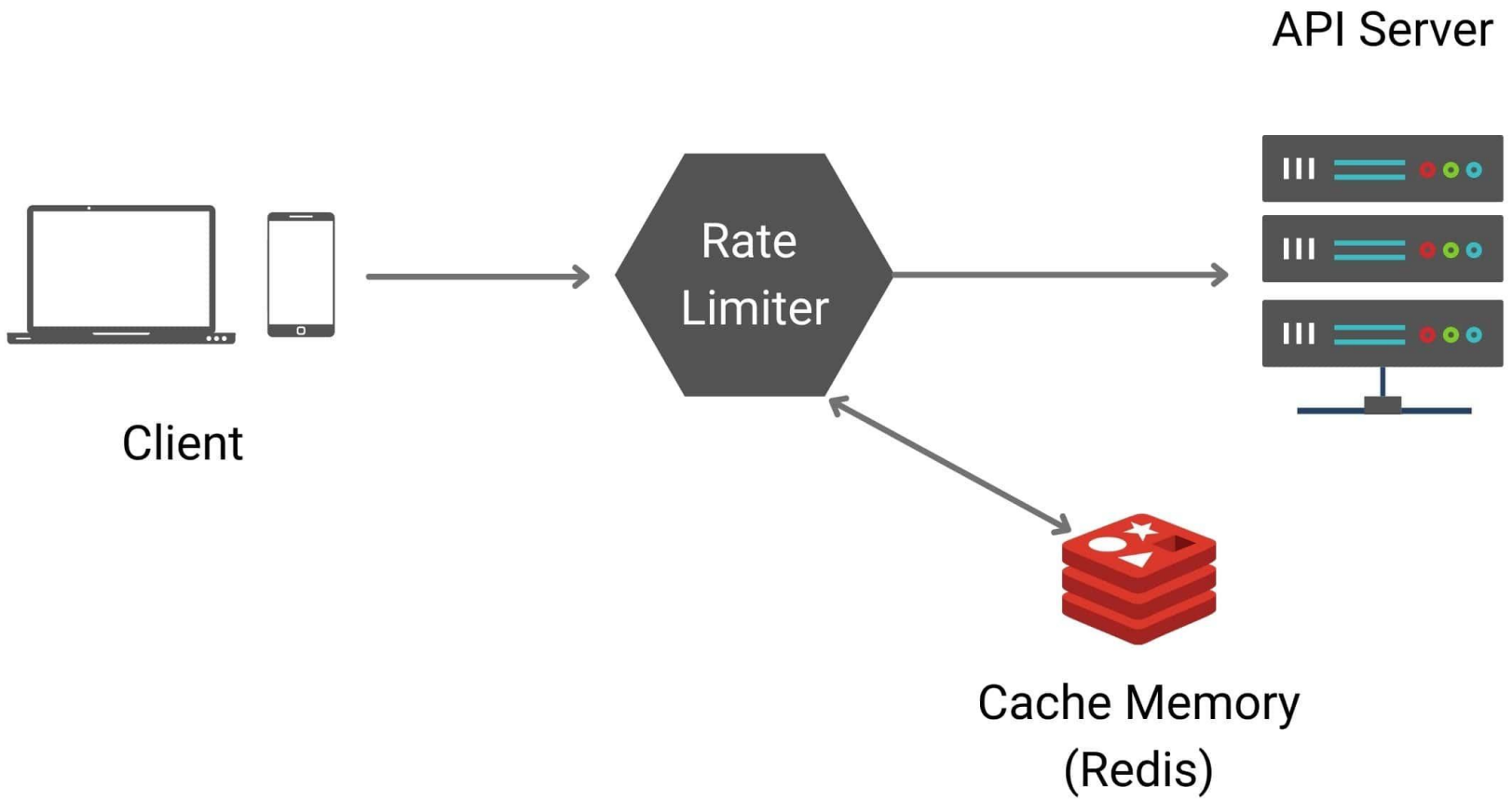
Xây dựng Rate Limiting

Where to put the Rate Limiter ?

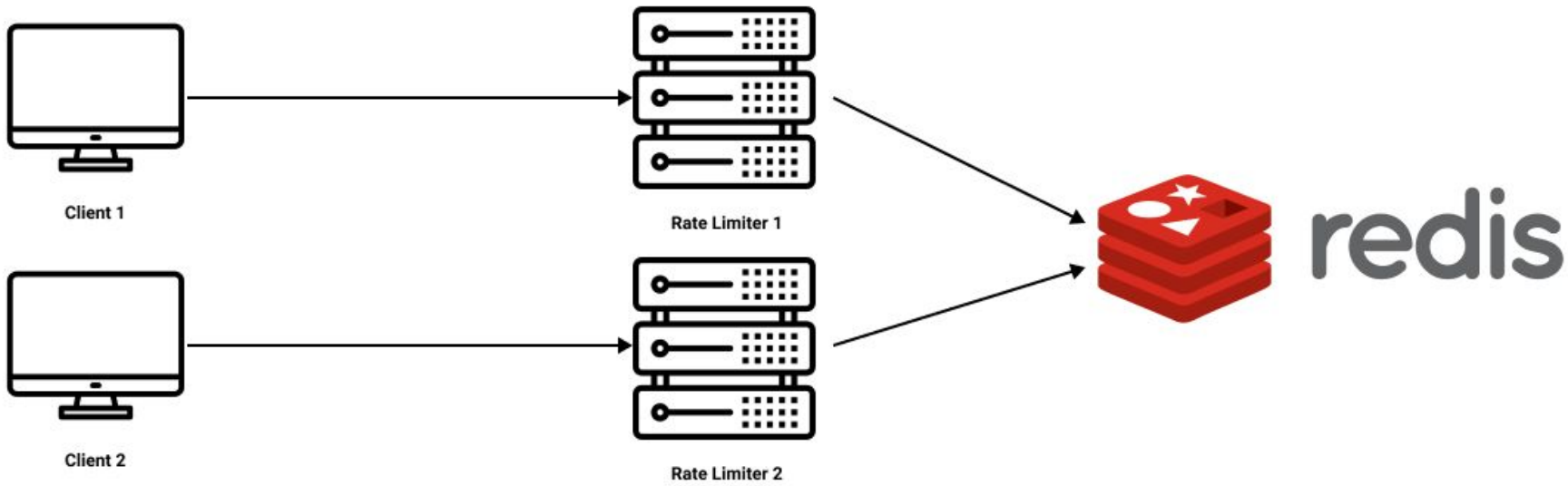
- Client side
- Server side
- Middleware



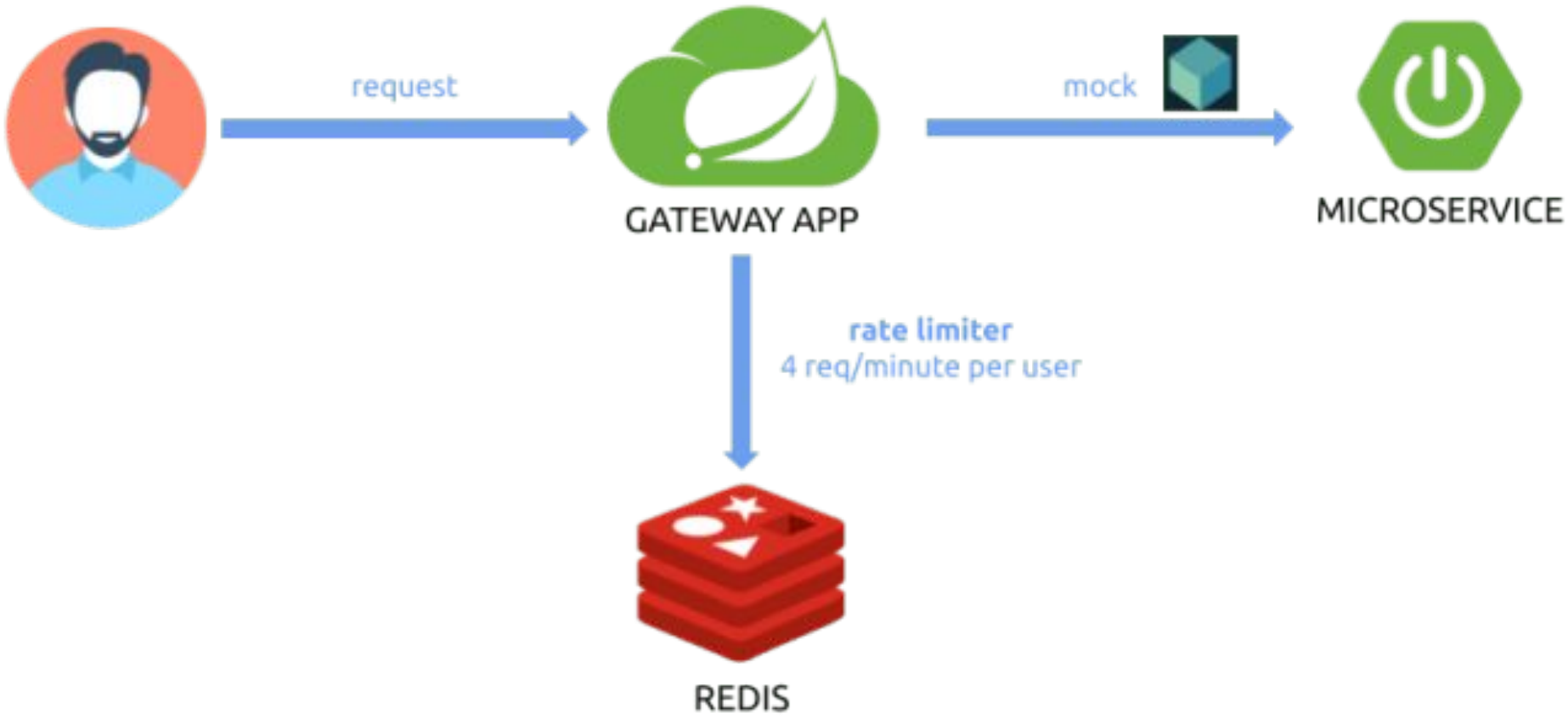
Xây dựng Rate Limiting



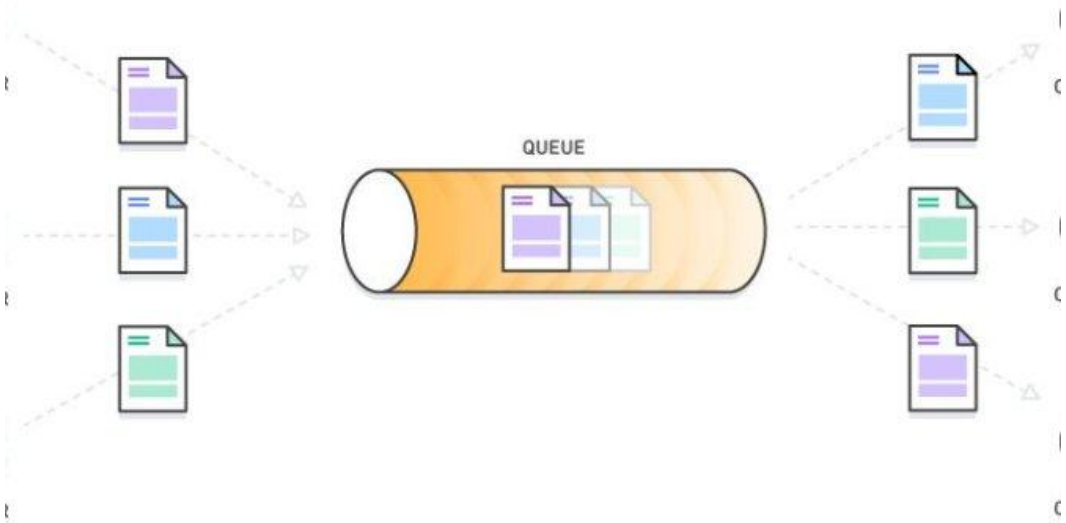
Xây dựng Rate Limiting



Xây dựng Rate Limiting với Spring cloud gateway



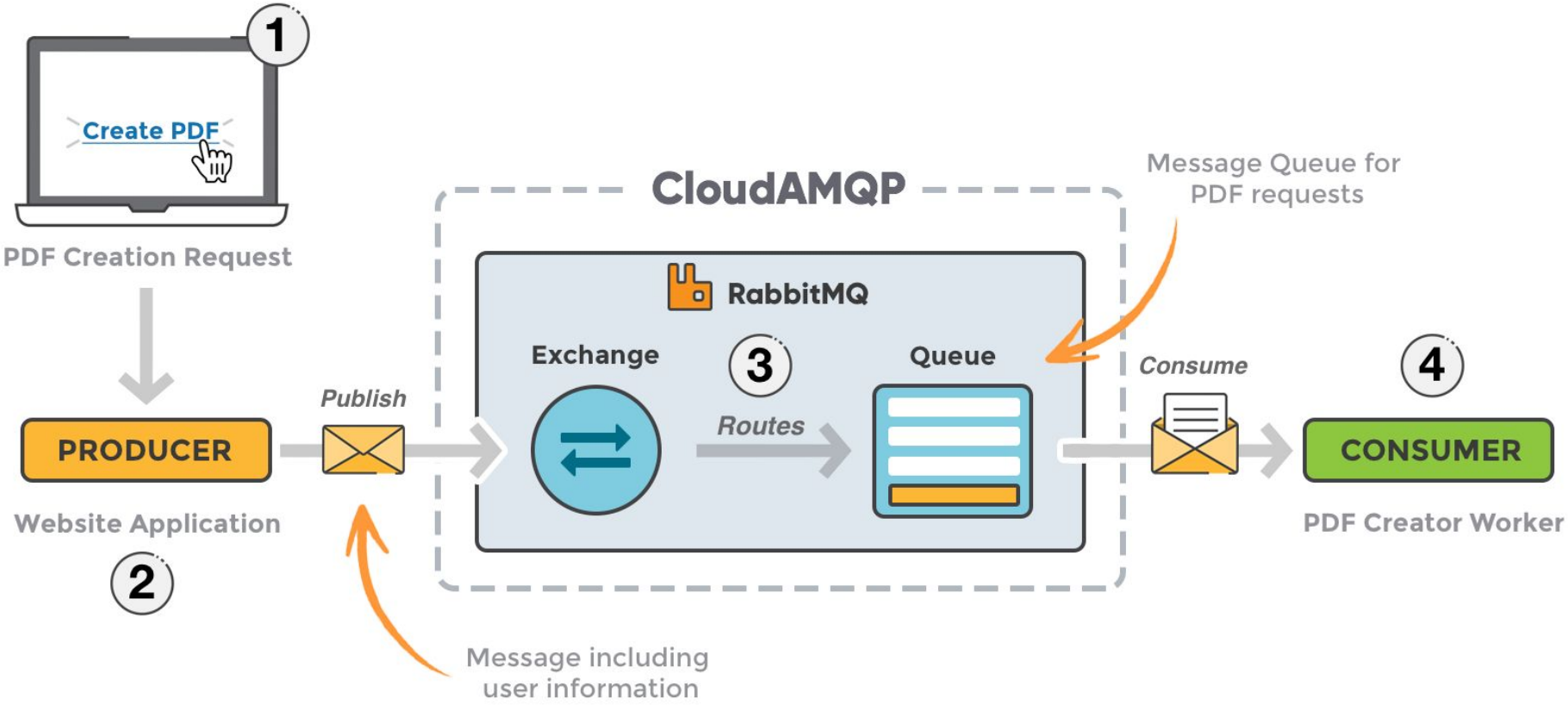
Message Queue là gì ?



Thành phần của Message Queue ?

- **Message**: Thông tin được gửi (có thể là text, binary hoặc JSON)
- **Message Queue**: Nơi chứa những message này, cho phép producer và consumer có thể trao đổi với nhau
- **Producer**: Service tạo ra thông tin, đưa thông tin vào message queue
- **Consumer**: Service nhận message từ message queue và xử lý
- Một service có thể vừa làm **producer**, vừa làm **consumer**

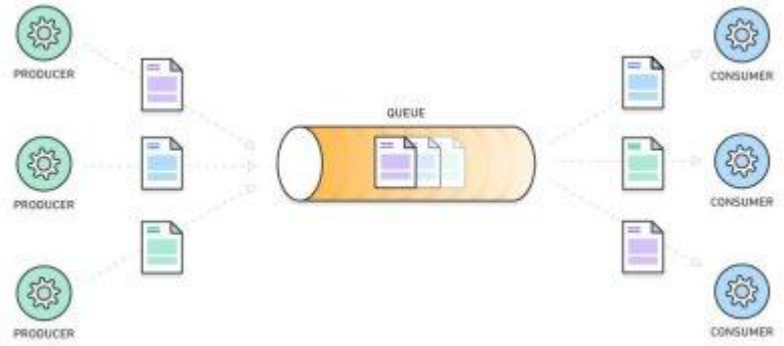
Usecase của Message Queue

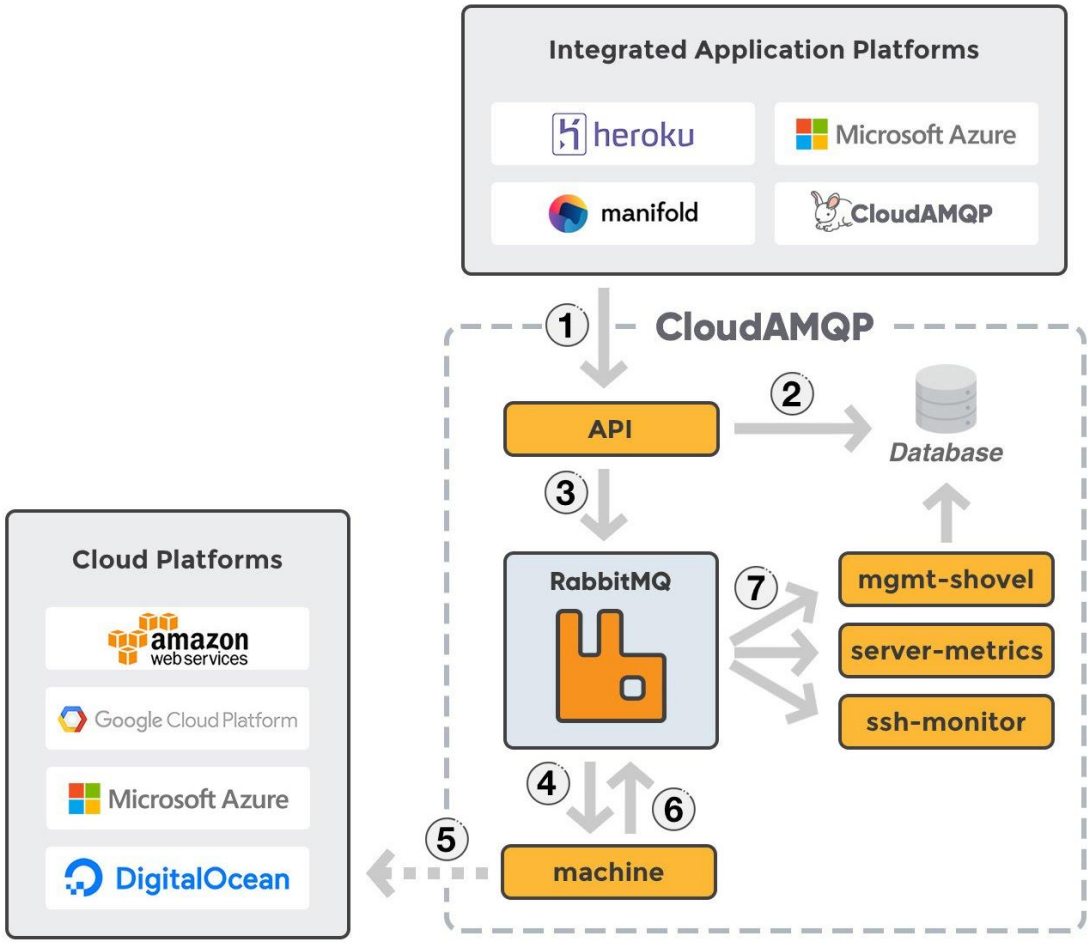


Tại sao phải sử dụng Message Queue

Ưu điểm về Message Queue

- Dễ scaling hệ thống:
- Phân tán hệ thống
- Đảm bảo duration/recovery:
- Hỗ trợ rate limit, batch process





Một số điểm cần lưu ý về MS

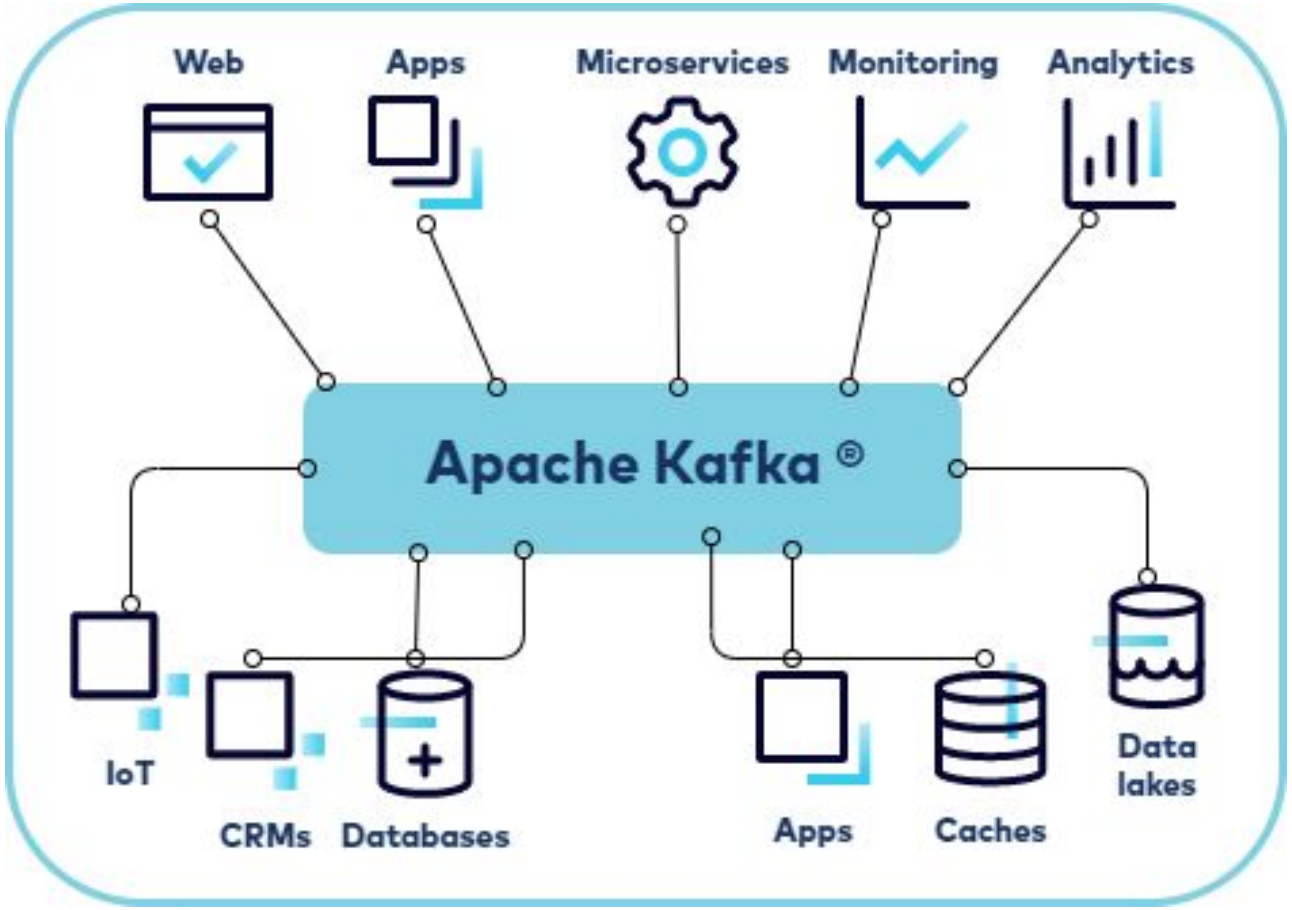
- Khó xử lý đồng bộ
- Làm hệ thống phức tạp hơn
- Cần đảm bảo message format
- Cần Monitoring Queue

Một số message queue hay được dùng hiện này bao gồm:

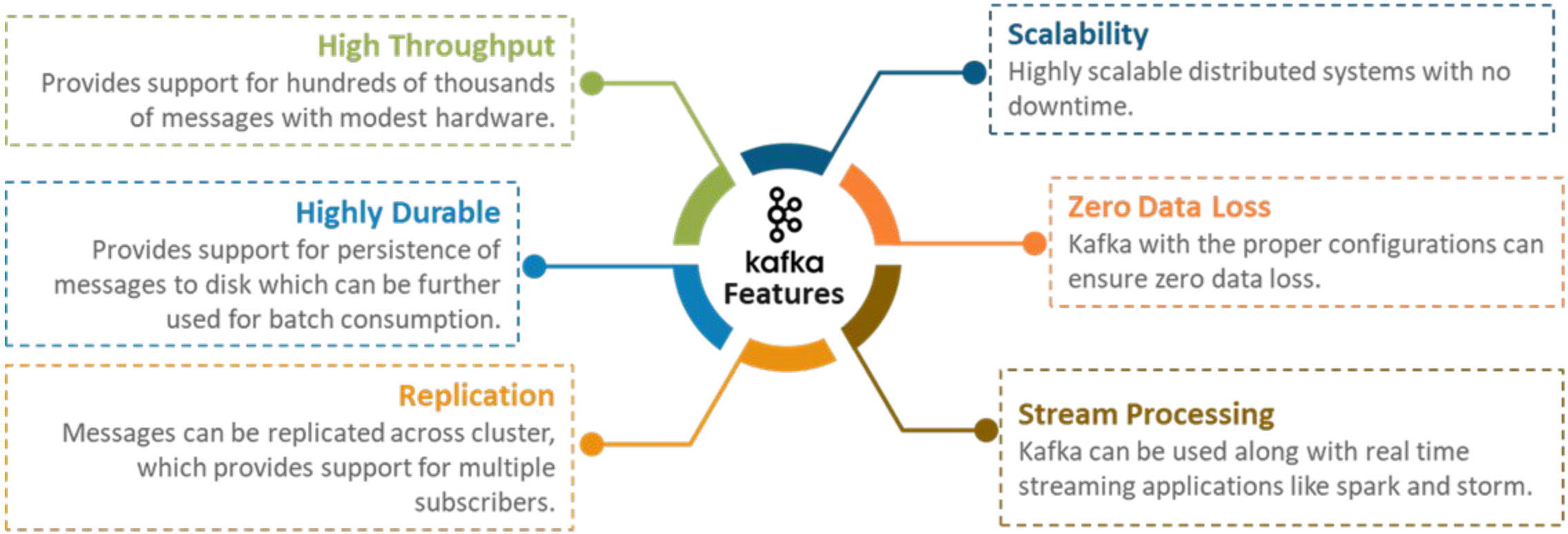
- *RabbitMQ*
- *Kafka*
- *Amazon SQS*
- *MSMQ (Microsoft Message Queuing)*
- *RocketMQ*
- *ZeroMQ*

Apache Kafka là gì ?

- Hệ thống message pub/sub phân tán
- Open source (distributed messaging system)
- Được phát triển bởi *Apache Software Foundation*
- Được viết bằng Java và Scala

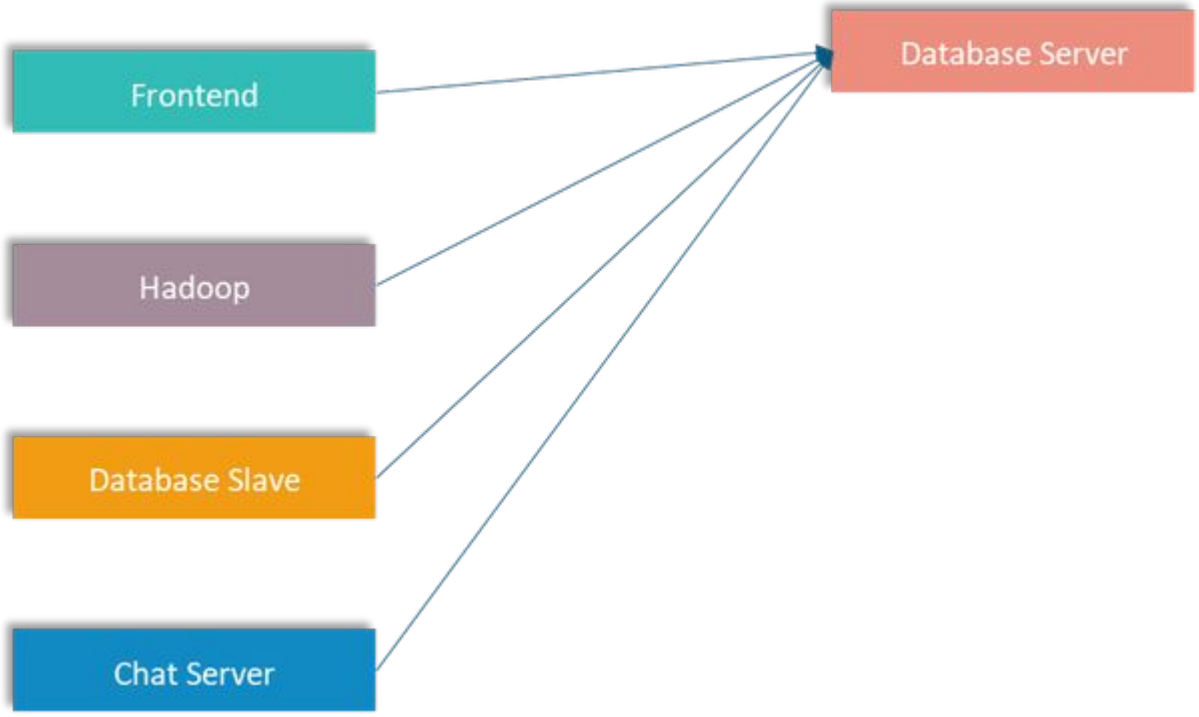


Tính năng của Apache Kafka ?



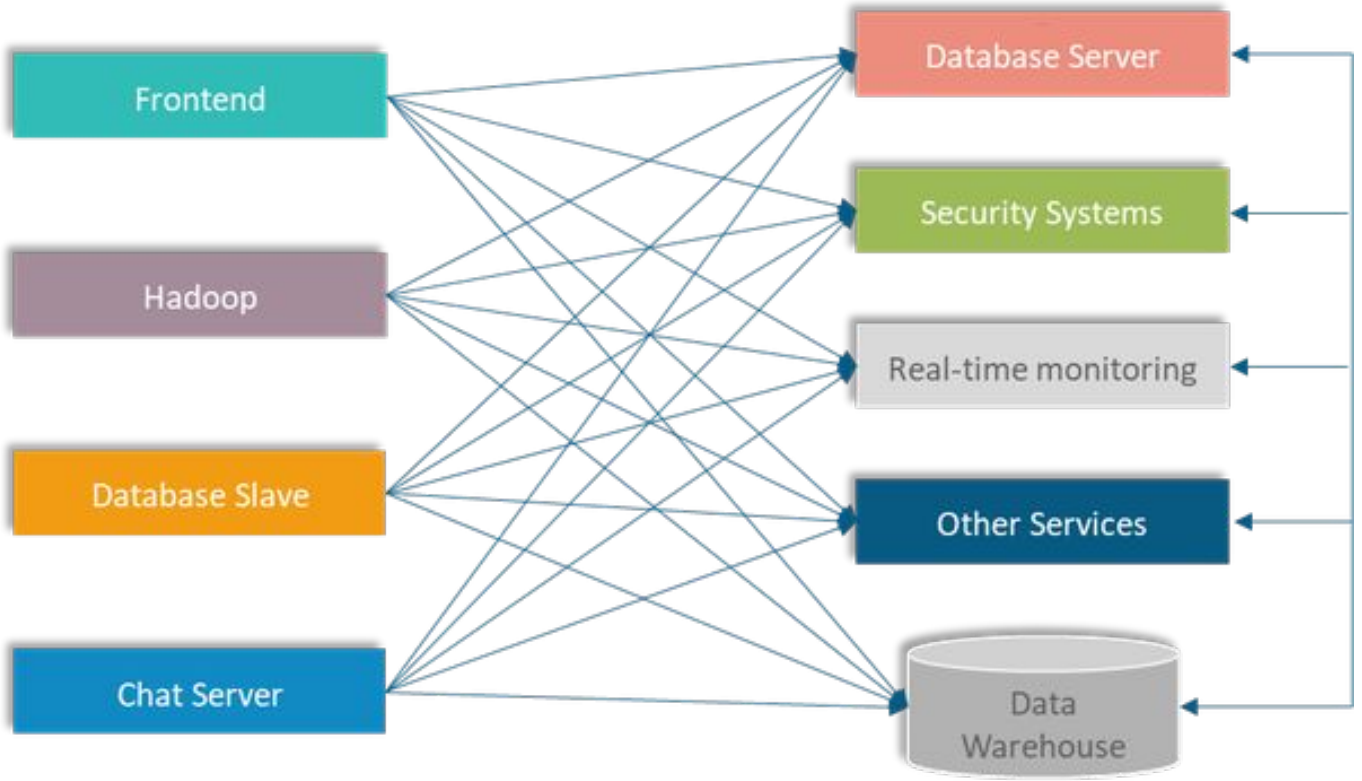
Tại sao Apache Kafka được sử dụng

- Website Activity Monitoring: theo dõi hoạt động của website
- Stream Processing: xử lý stream
- Log Aggregation: tổng hợp log
- Metrics Collection: thu thập dữ liệu

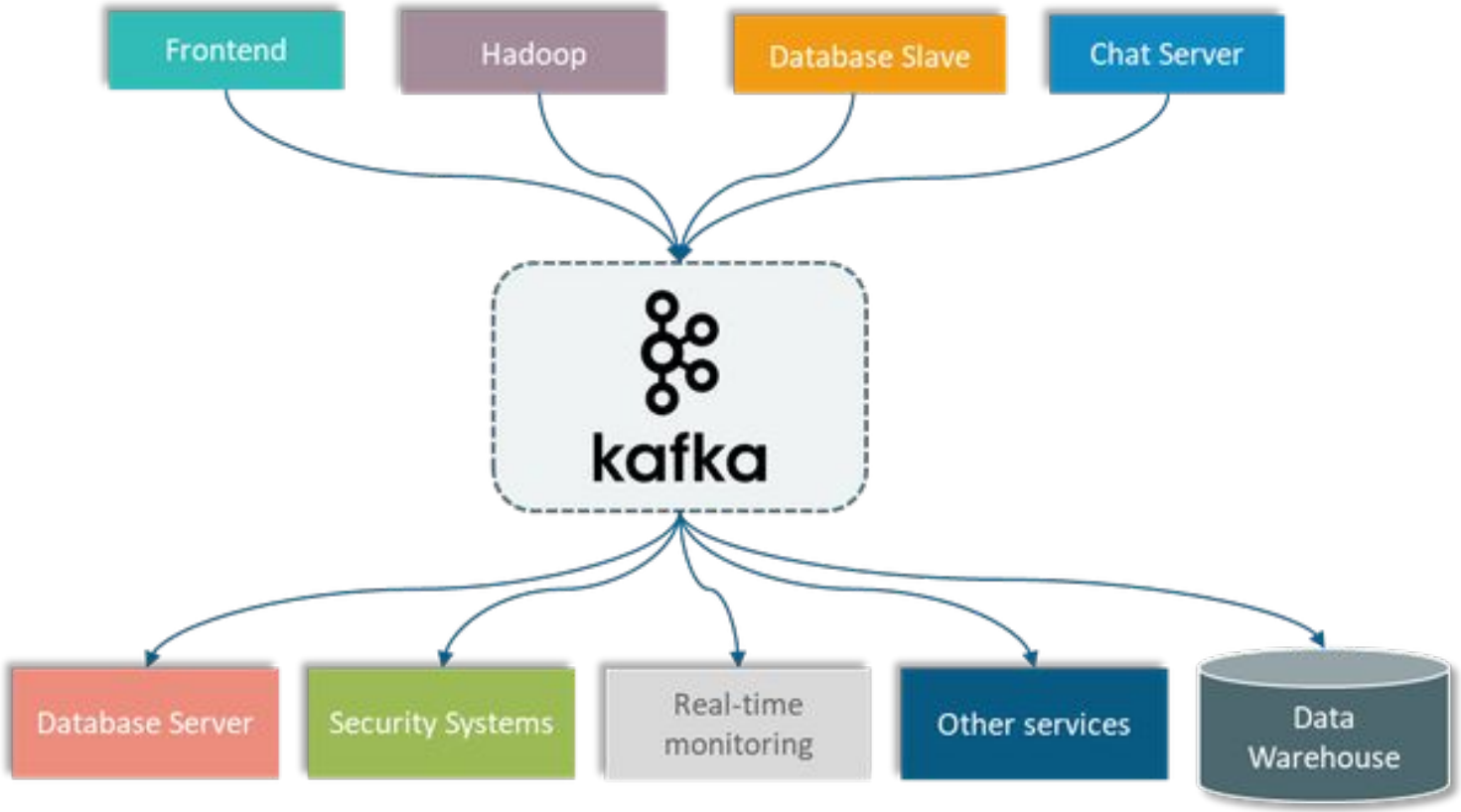


Tại sao Apache Kafka được sử dụng

Nhưng trong thực tế, hệ thống thương mại điện tử sẽ còn phải kết nối đến một vài server khác nữa như là

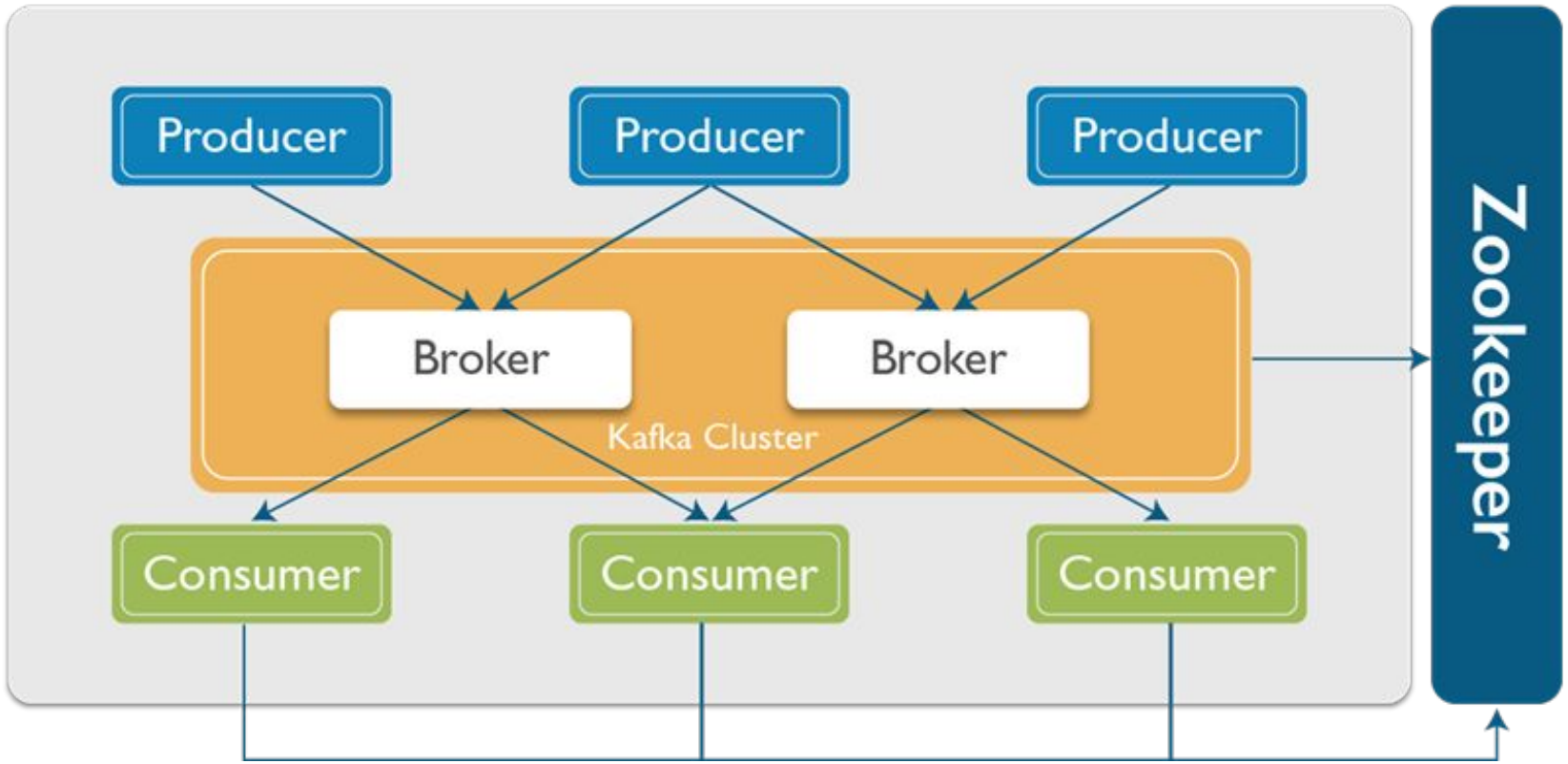


Tại sao Apache Kafka được sử dụng



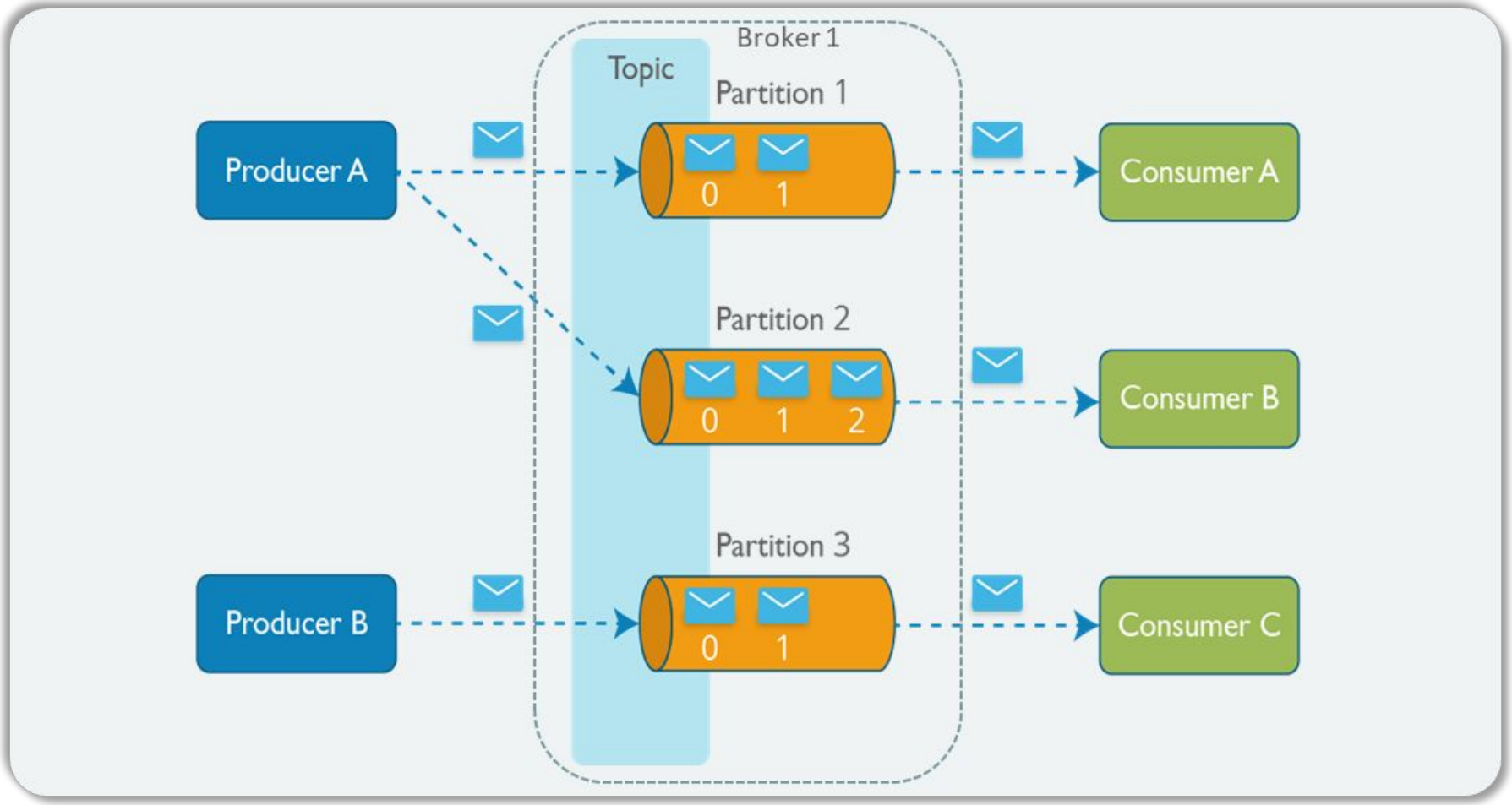
Cấu trúc của Apache Kafka

Một mô hình cấu trúc Kafka đơn giản



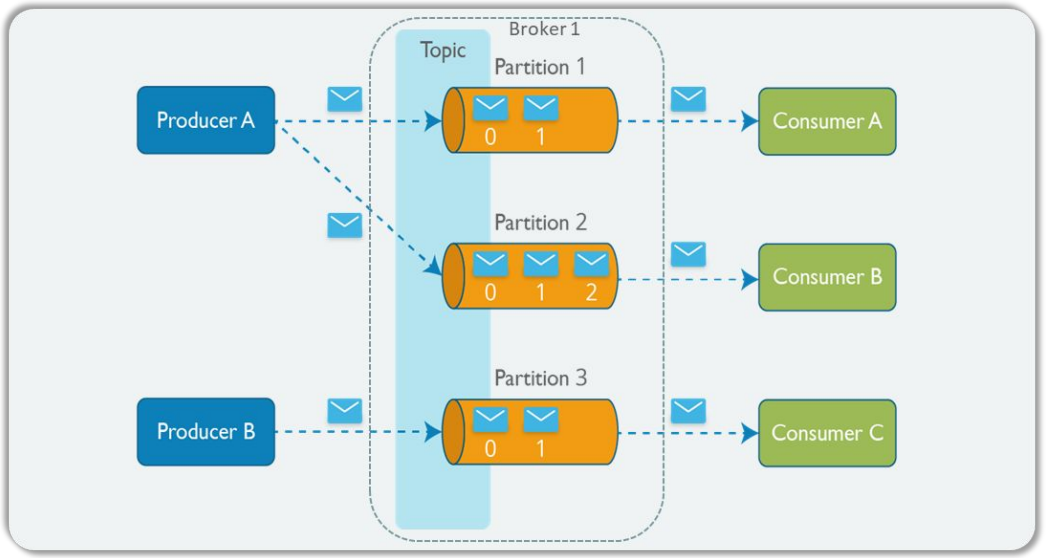
Cấu trúc của Apache Kafka

Một mô hình cấu trúc Kafka chi tiết



Cấu trúc của Apache Kafka

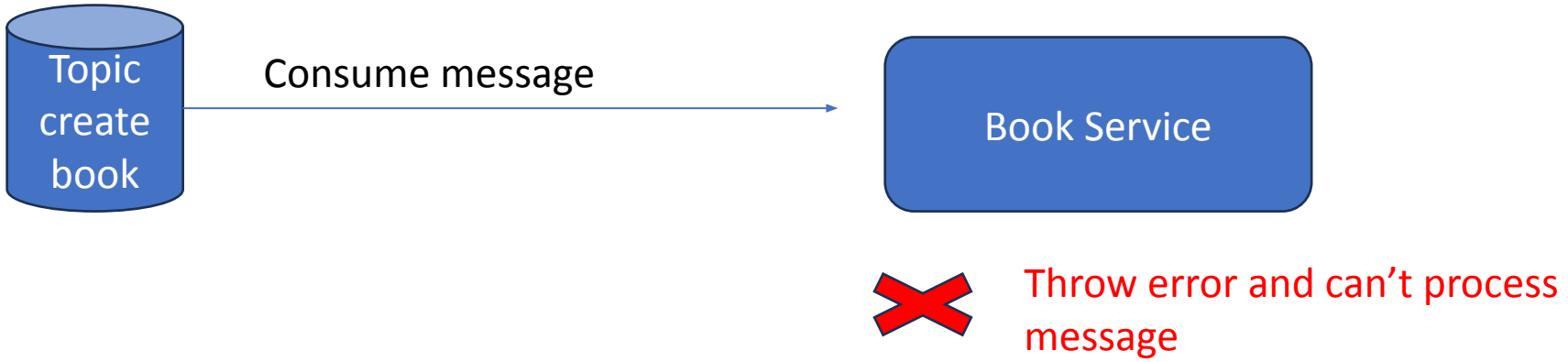
- Producer: Một producer có thể là bất kì ứng dụng nào có chức năng publish message vào một topic.
- Messages: Messages đơn thuần là byte array và developer có thể sử dụng chúng để lưu bất kì object với bất kì format nào - thông thường là String, JSON và Avro
- Topic: Một topic là một category hoặc feed name nơi mà record được publish.
- Partitions: Các topic được chia nhỏ vào các đoạn khác nhau, các đoạn này được gọi là partition
- Consumer: Một consumer có thể là bất kì ứng dụng nào có chức năng subscribe vào một topic và tiêu thụ các tin nhắn.
- Broker: Kafka cluster là một set các server, mỗi một set này được gọi là 1 broker
- Zookeeper: được dùng để quản lý và bố trí các broker.



Khuyết điểm của Kafka

- Không có bộ công cụ giám sát hoàn chỉnh: Có khá nhiều tool khác nhau, mỗi một tool lại hỗ trợ một việc quản lý khác nhau cho từng mục đích quản lý, vd như:
  - Kafka tool (offset manager) GUI tool dùng để quản lý topic và consumer
  - Lense: dùng để hỗ trợ việc query message trong Kafka
  - Akhq là toolbox để quản lý Kafka cũng như view data bên trong Kafka
- Không hỗ trợ chọn topic theo wildcard: Kafka không hỗ trợ việc sử dụng và lựa chọn từng topic theo wildcard mà phải sử dụng chính xác tên topic để xử lý message.
- Reduces performance – Giảm hiệu suất: Vấn đề xảy ra kích thước message tăng lên, buộc consumer và producer phải compress và decompress message, làm cho bộ nhớ sẽ chậm, dẫn đến ảnh hưởng tới throughput và hiệu suất.
- Behave clumsy – Xử lý hơi vụng về: Đôi khi do số lượng queues trong Kafka cluster tăng lên nhiều, Kafka thường có hiện tượng xử lý chậm chạp hơn.

Kafka – Error Handling



Kafka – Error Handling

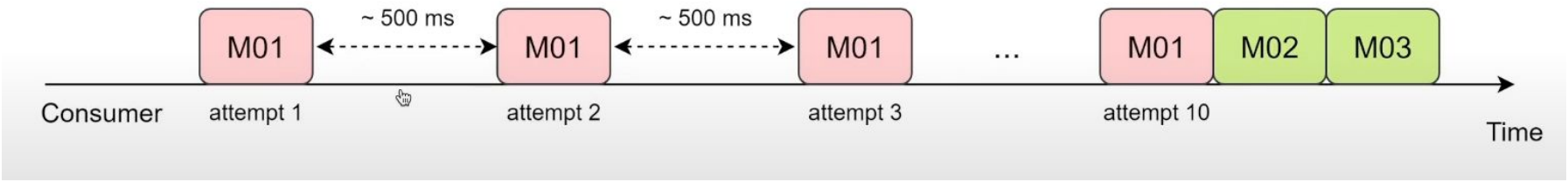
Giải pháp 1: Cơ chế Retry

Lỗi chia làm 2 loại:

- Lỗi có thể retry
- Lỗi Không thể retry

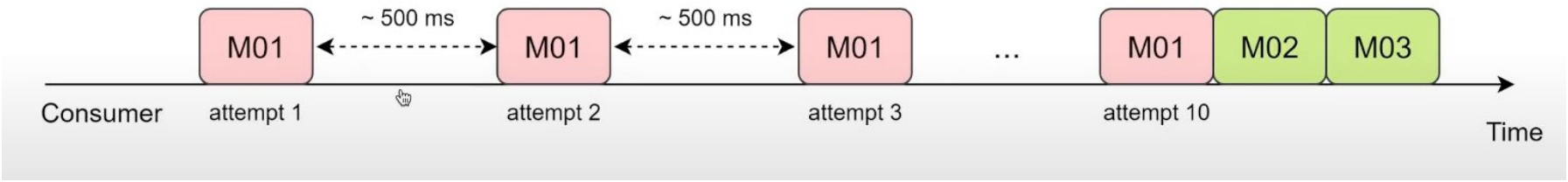
Kafka – Error Handling

Giải pháp 1: Cơ chế Retry



Kafka – Error Handling

Cơ chế Simple Retry



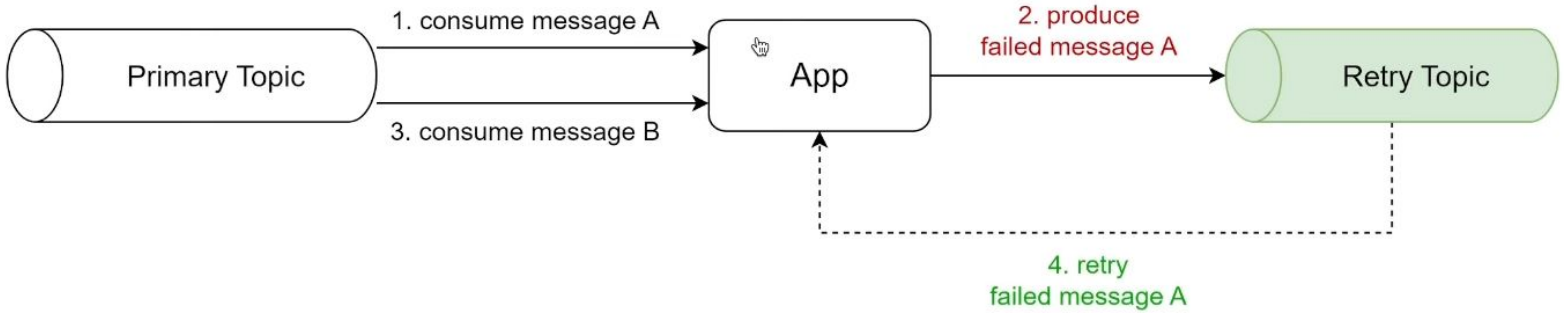
- Consumer sẽ bị blocking
- Sau 10 attempts consumer mới xử lý message tiếp theo

Nhược điểm

- Ảnh hưởng tới latency của các message khác
- Chiếm tài nguyên của consumer
- Khó trace path của message lỗi
- Khó lấy các metadata khi retry
- Không linh hoạt

Kafka – Error Handling

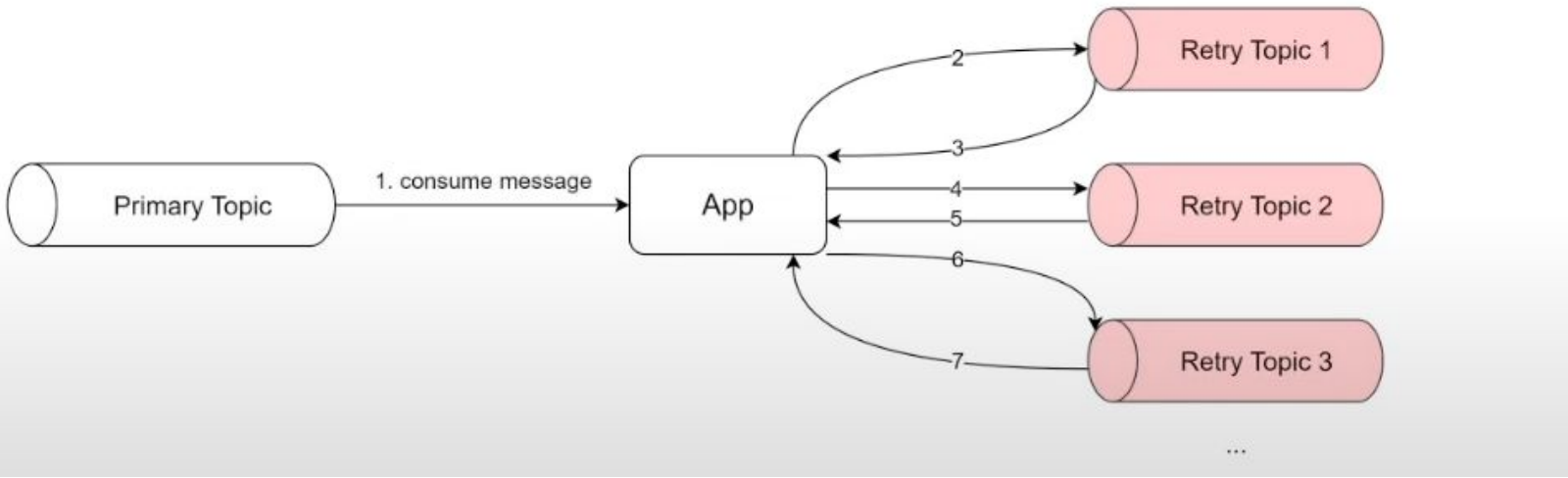
Cơ chế Separate Retry Queue



- Consumer sẽ không bị block
- Dễ trace được path của message lỗi

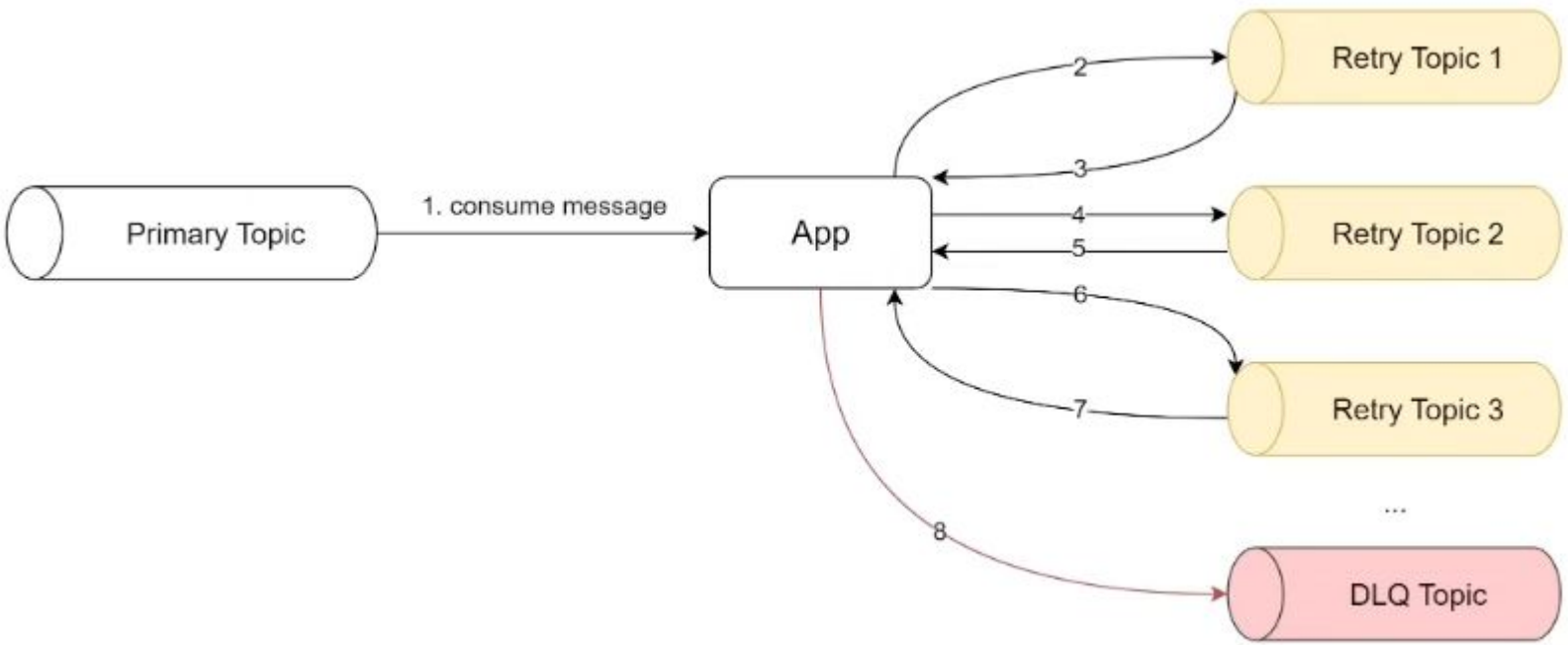
Kafka – Error Handling

Nếu như vẫn Fail khi retry thì sao ?



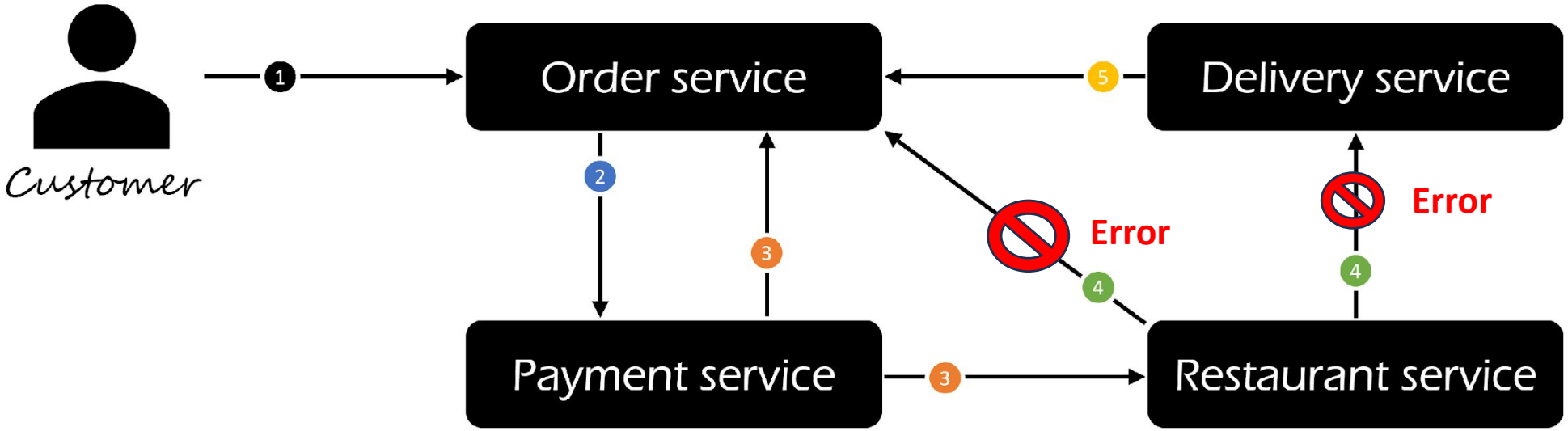
Kafka – Error Handling

Giải pháp: Dead Letter Queue (DLQ)



Distributed Transaction

Dual write problem



Distributed Transaction

Như chúng ta biết, **transaction** bao gồm 4 tính chất **ACID**:

- Atomicity.
- Consistency.
- Isolation.
- Durability.

Distributed Transaction

CONCEPT

Distributed Transaction

CONCEPT: All or Nothing

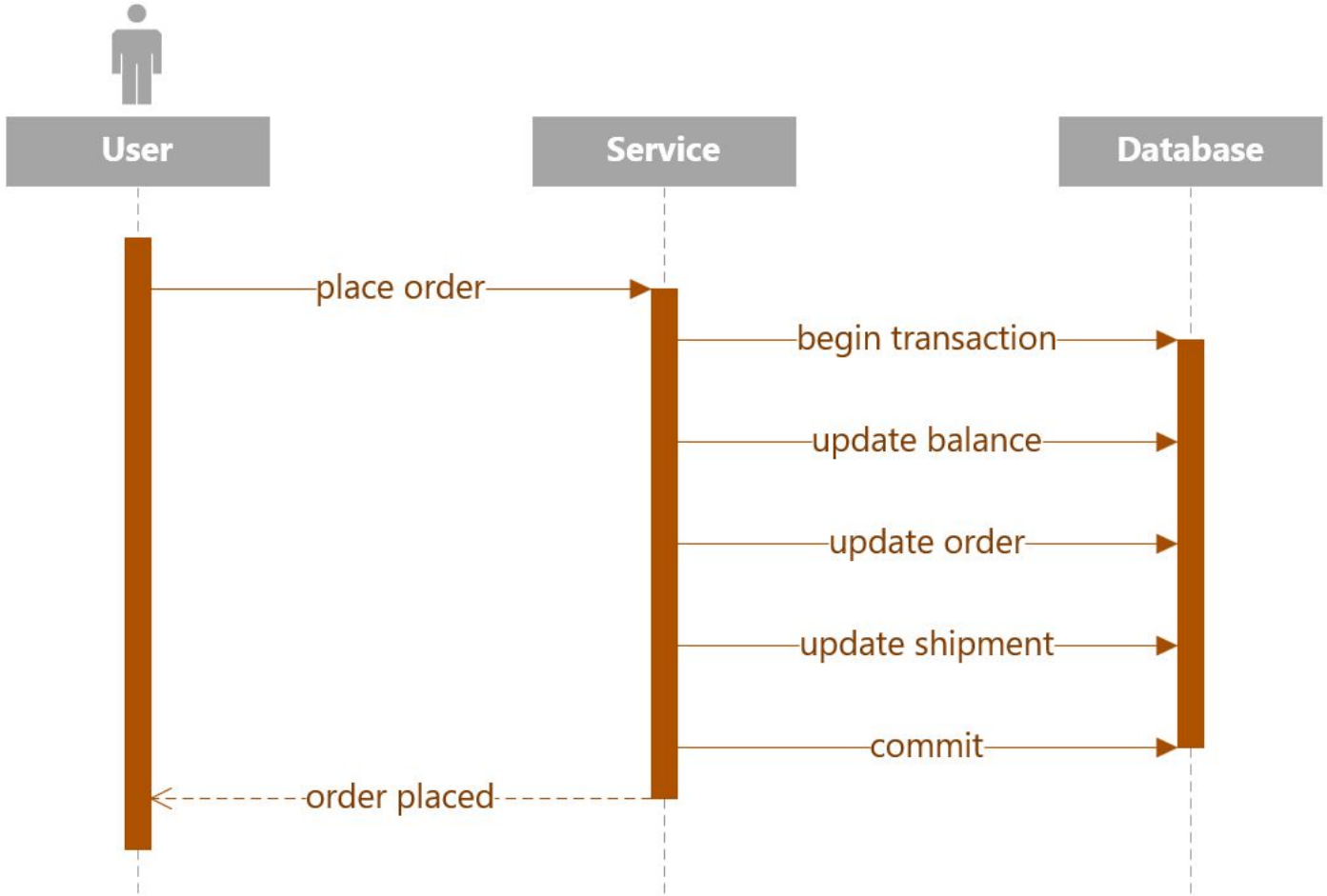
Để đảm bảo được concept trên với **single database** là một chuyện hết sức đơn giản:

```
BEGIN TRANSACTION;

INSERT INTO...;
UPDATE ...;
DELETE ...;

COMMIT;
```

Distributed Transaction

Monilithic Architecture

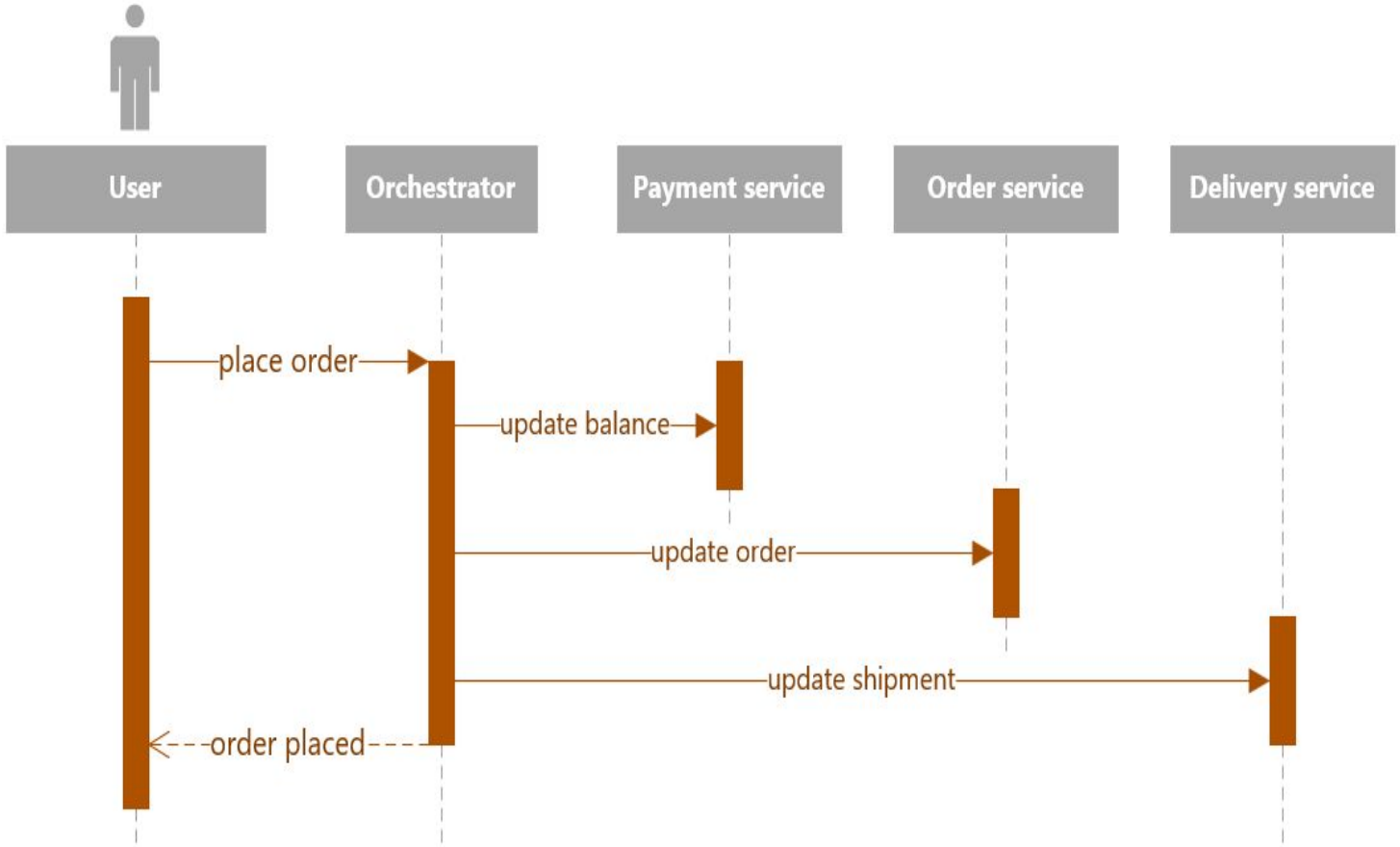
- Khởi tạo transaction.
- Trừ tiền trong tài khoản.
- Tạo đơn hàng.
- Cập nhật thông tin giao hàng.
- Commit.



Distributed Transaction

Microservices Architecture

- **Payment service**: xử lý liên quan đến kiểm tra tài khoản, cộng trừ số dư.
- **Order service**: tạo và quản lý order.
- **Delivery service**: tạo và quản lý công việc liên quan đến giao hàng.

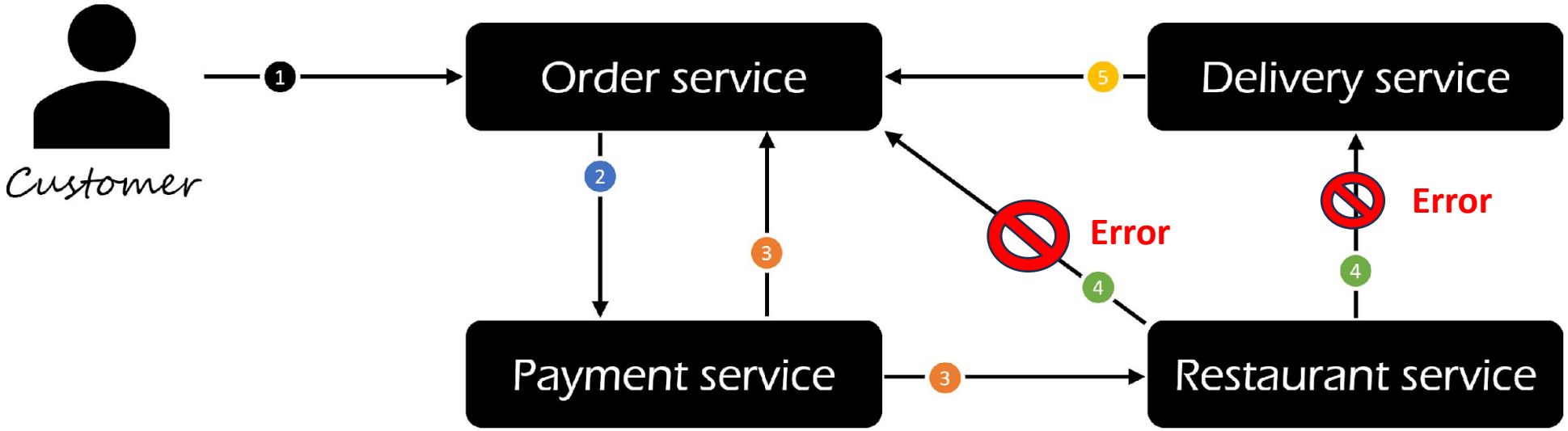


Distributed Transaction

Có nhiều các thuật toán, cơ chế, pattern khác nhau để implement **distributed transaction** có thể kể tên:

- Two-phase commit.
- Three-phase commit.
- **Saga pattern** with **Orchestration** & Choreography.
- Parallel pipeline.

Distributed Transaction – SAGA Pattern



Distributed Transaction – SAGA Pattern

SAGA pattern xử lí distributed transaction với 2 cách thức là **choreography** và **orchestration**.

- **Choreography - Event based**.
- **Orchestration - Command based**.

Distributed Transaction – SAGA Pattern

Trước khi đi vào chi tiết thì cần hiểu **event** và **command** là gì đã.

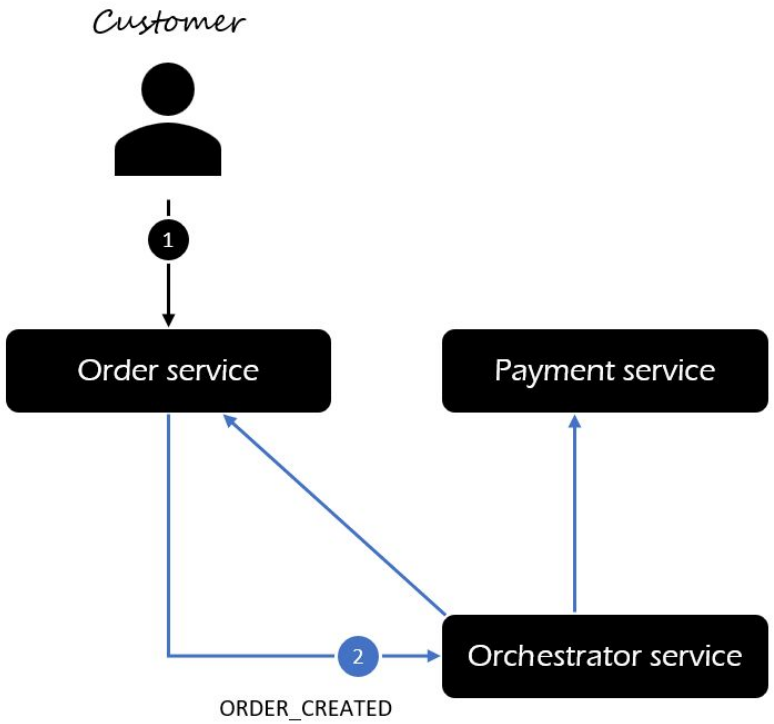
- **Event**: về cơ bản **event** nói về một điều gì đó đã xảy ra. Mang tính chất thông báo, giống như **notification**.
- **Command**: là một yêu cầu, phía nhận command phải thực hiện, và phía yêu cầu mong muốn có response.

Distributed Transaction – SAGA Pattern

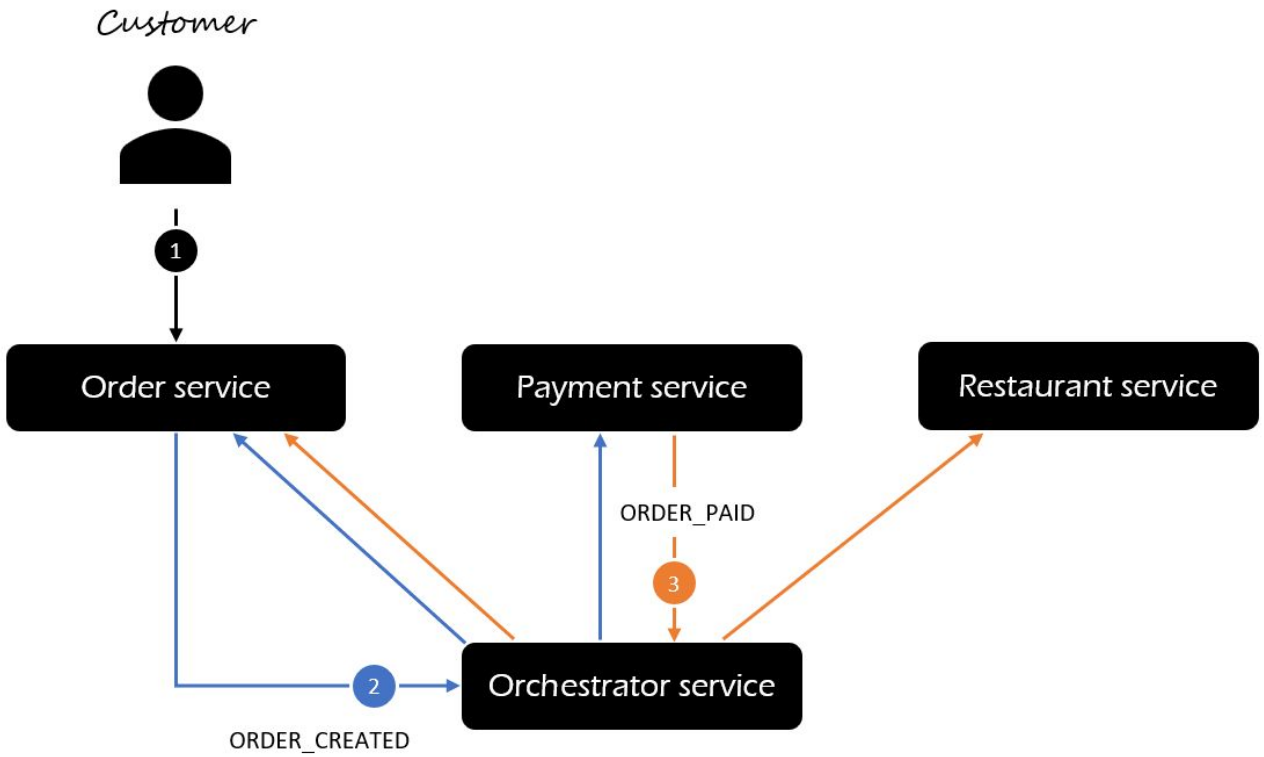
Orchestration - Command based

SAGA orchestration là cách điều phối các local transaction để các transaction được execute hoặc rollback đúng thứ tự. Việc execute hay rollback phụ thuộc vào yêu cầu của orchestrator.

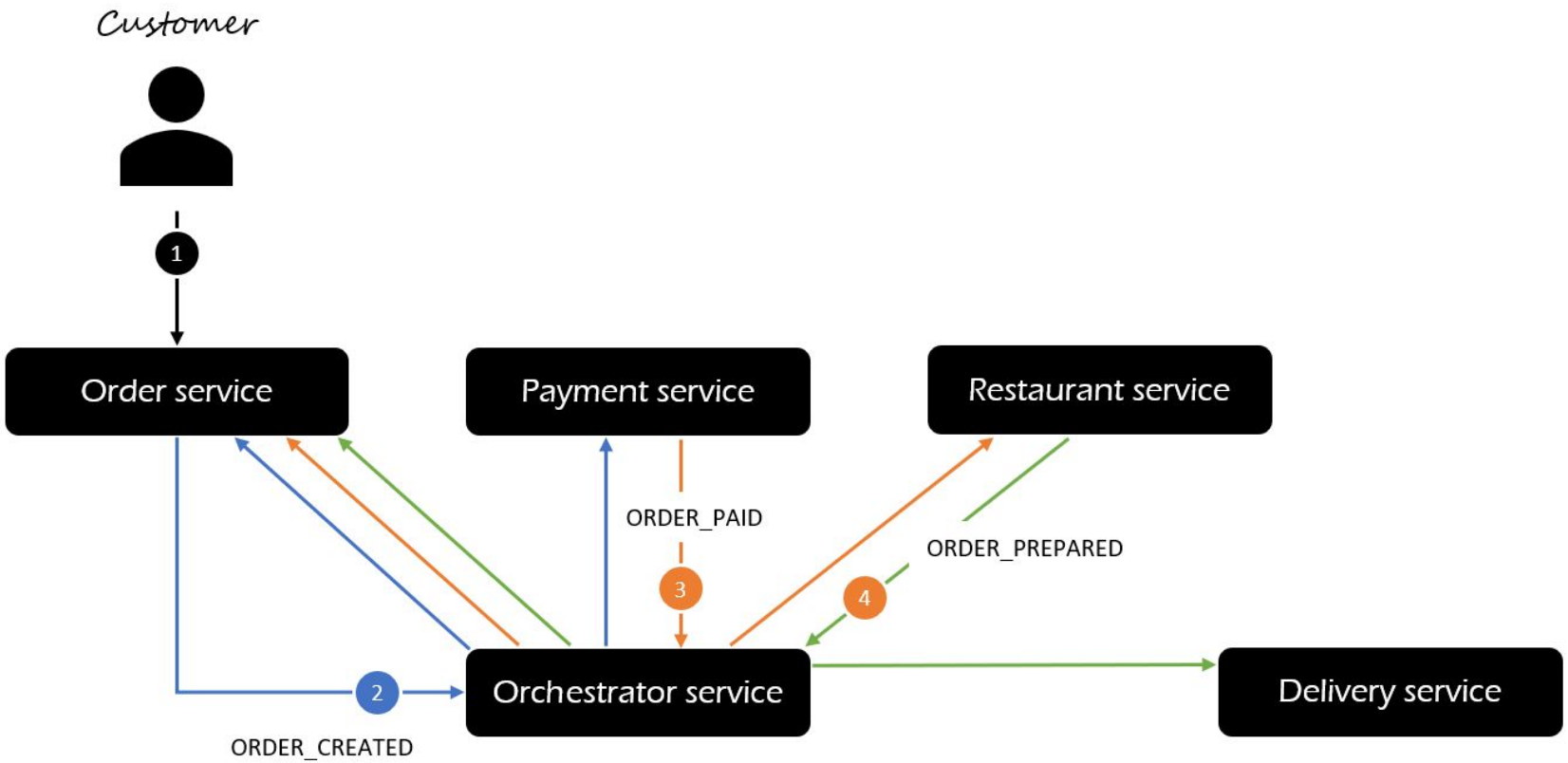
Distributed Transaction – SAGA Pattern



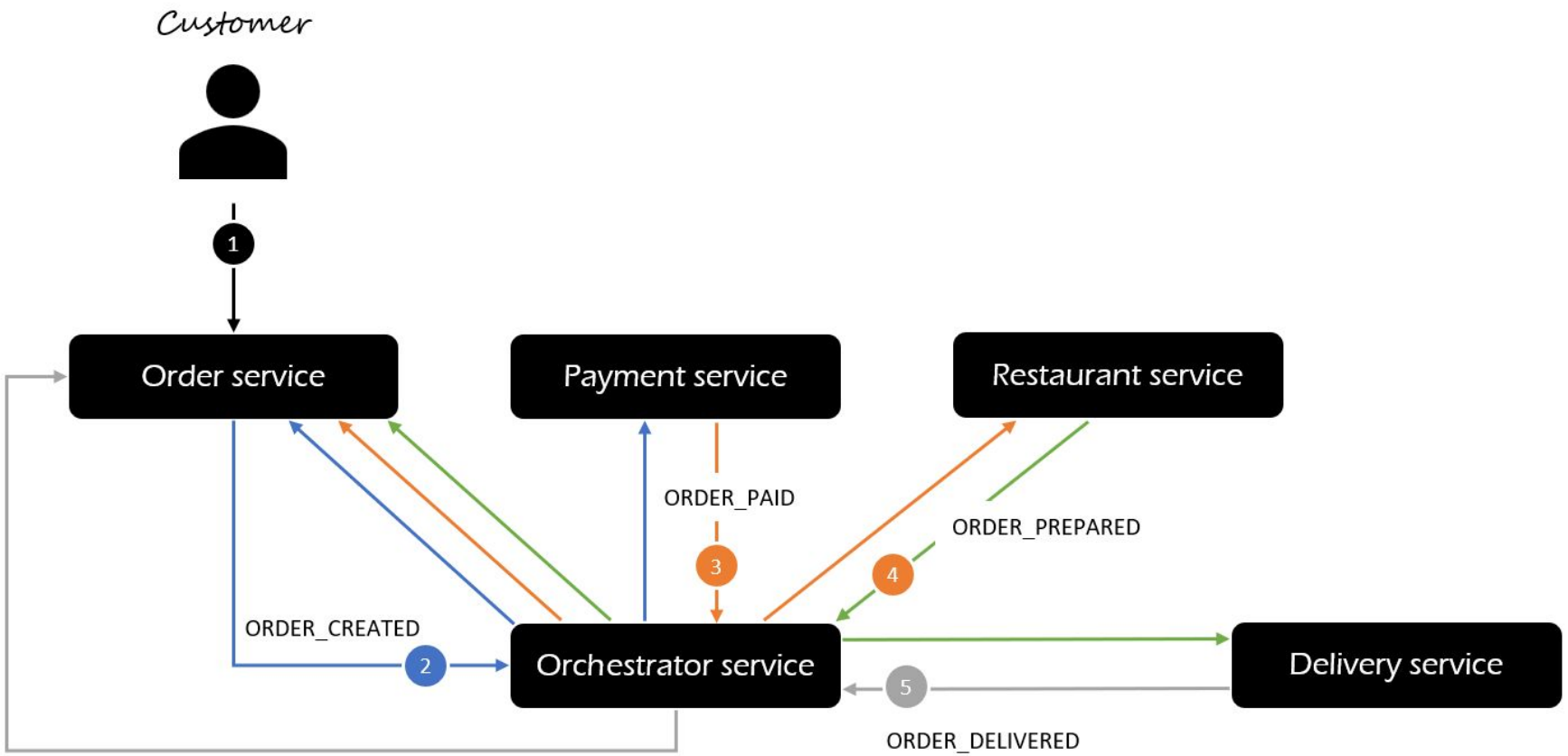
Distributed Transaction – SAGA Pattern



Distributed Transaction – SAGA Pattern



Distributed Transaction – SAGA Pattern



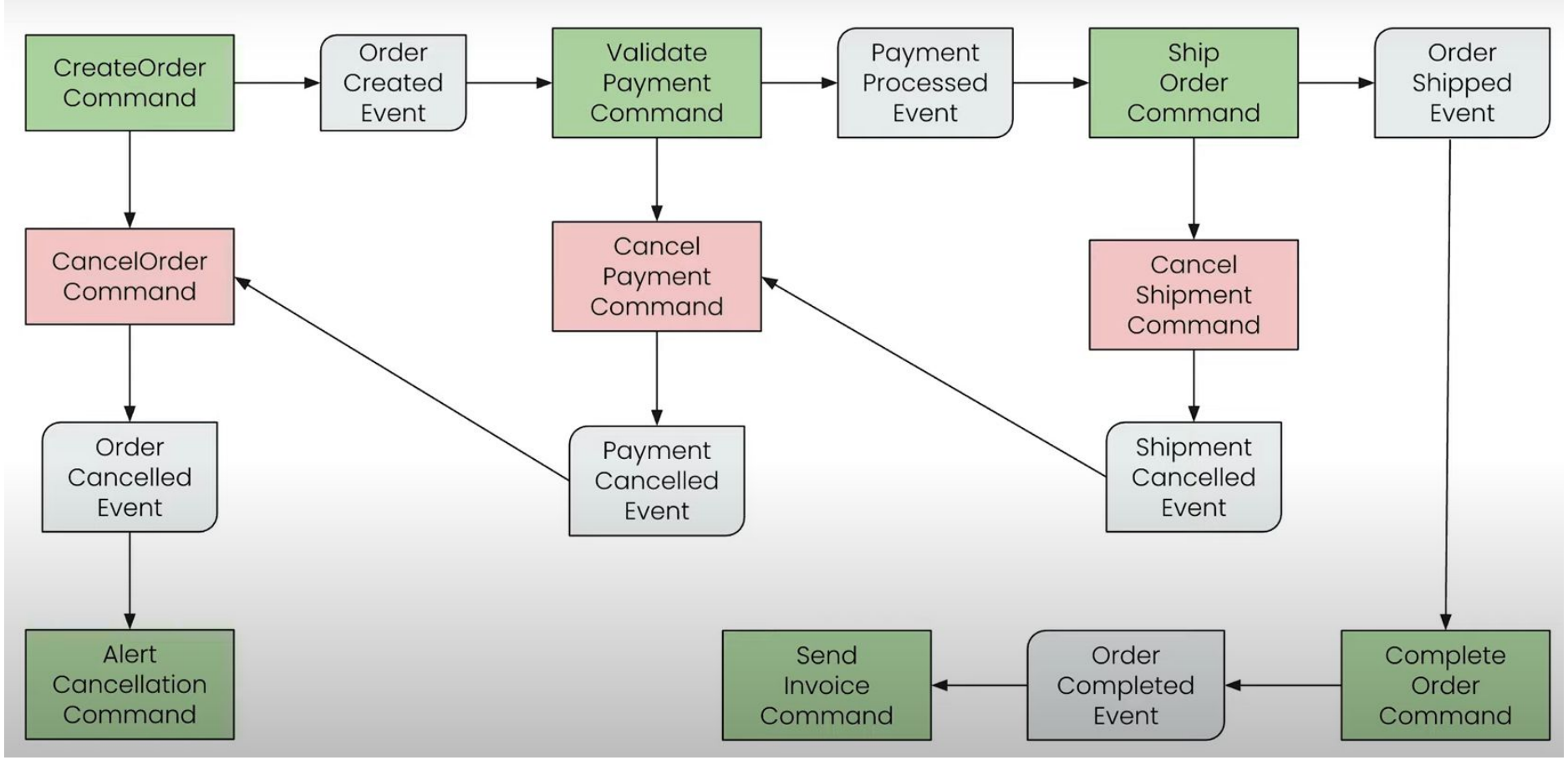
Distributed Transaction – SAGA Pattern

Exception handling

- **Retry**: **Restaurant** sai nhân viên đi mua nguyên liệu nhanh còn kịp. Sau khi có đầy đủ nguyên liệu thì thực hiện tiếp công việc.
- **Abort**: báo lại cho Karla (**orchestrator**) rằng nhà bếp không thể xử lí được tình huống này. Karla nhận thông tin và báo lại cho (**payment service**) hoàn trả tiền vào thẻ cho khách. Sau đó báo tiếp đến (**order service**) update order, gọi điện xin lỗi khách.

Distributed Transaction – SAGA Pattern

Exception handling



Distributed Transaction – SAGA Pattern

Exception handling

Đấy là tưởng tượng thì nó thế thôi, còn implement phức tạp hơn nhiều lần, có vô số các thứ cần phải xử lí như:

- Retry như thế nào? State cần lưu ở đâu để đảm bảo vẫn có thể retry nếu service crash...
- Retry bao lâu, trường hợp nào thì retry, trường hợp nào abort...
- Đấy là còn chưa bàn đến backbone communication: REST API, gRPC, Message broker...

Distributed Transaction – SAGA Pattern

Characteristic

Bất kì cách tiếp cận nào cũng có 2 phần là **ưu điểm** và **nhược điểm**, vấn đề là cần nắm rõ 2 phần này để áp dụng sao cho tối ưu được ưu điểm và hạn chế được nhược điểm.

- **Good for complex workflow / Less coupling**
- **Separation of concerns**:

Distributed Transaction – SAGA Pattern

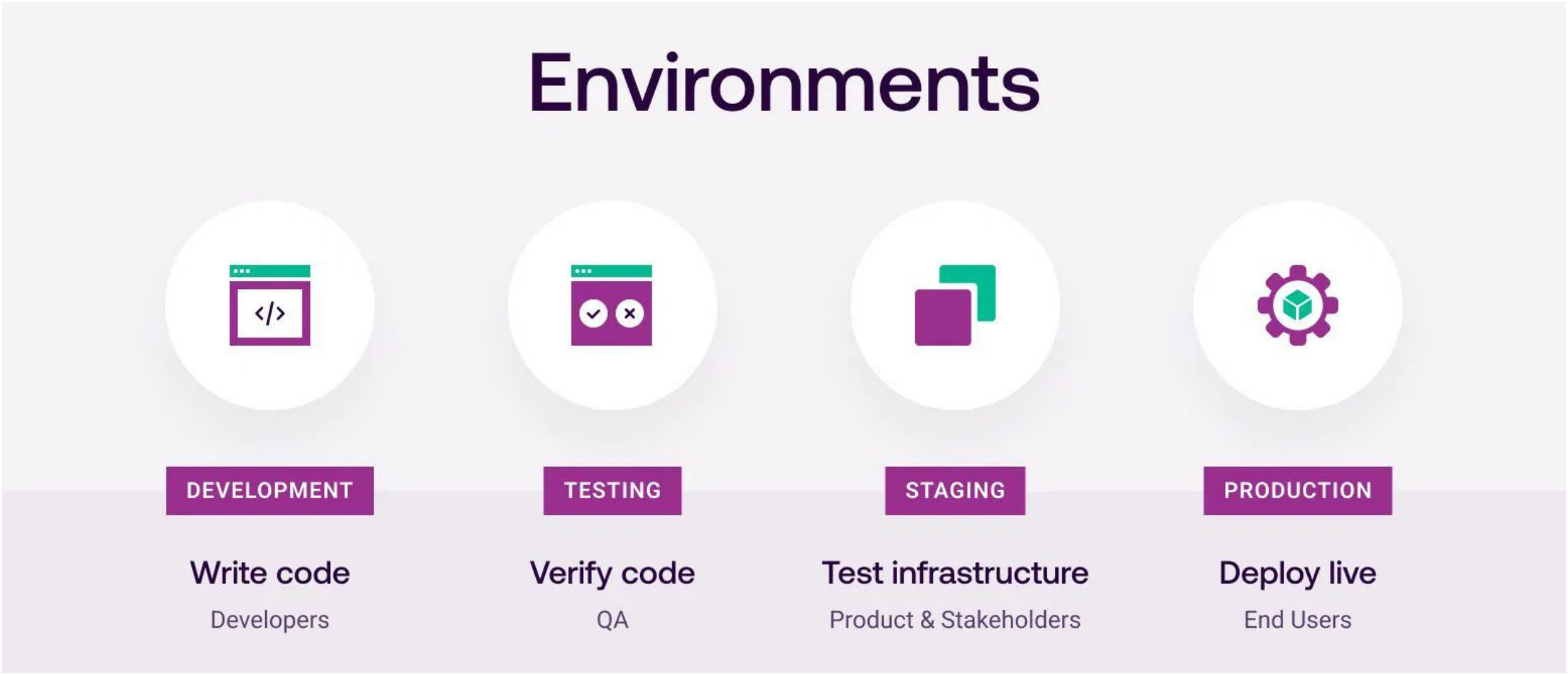
Drawbacks

- **Single-point failure**:**Separation of concerns**:
- **Bottle neck / Letancy**:

Docker là gì ?

Docker là một nền tảng containerization

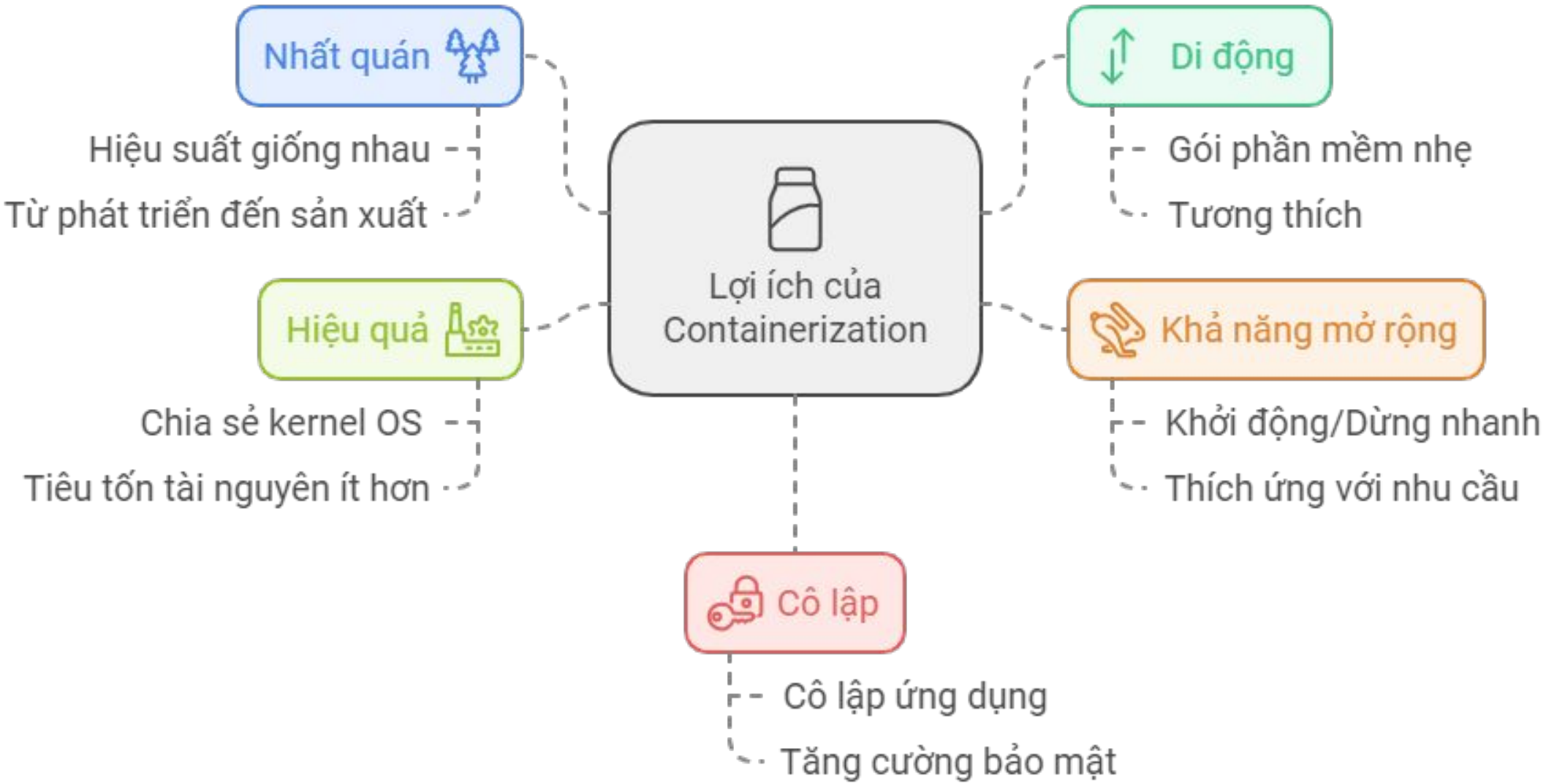
Containerization là gì ?



Containerization là gì ?

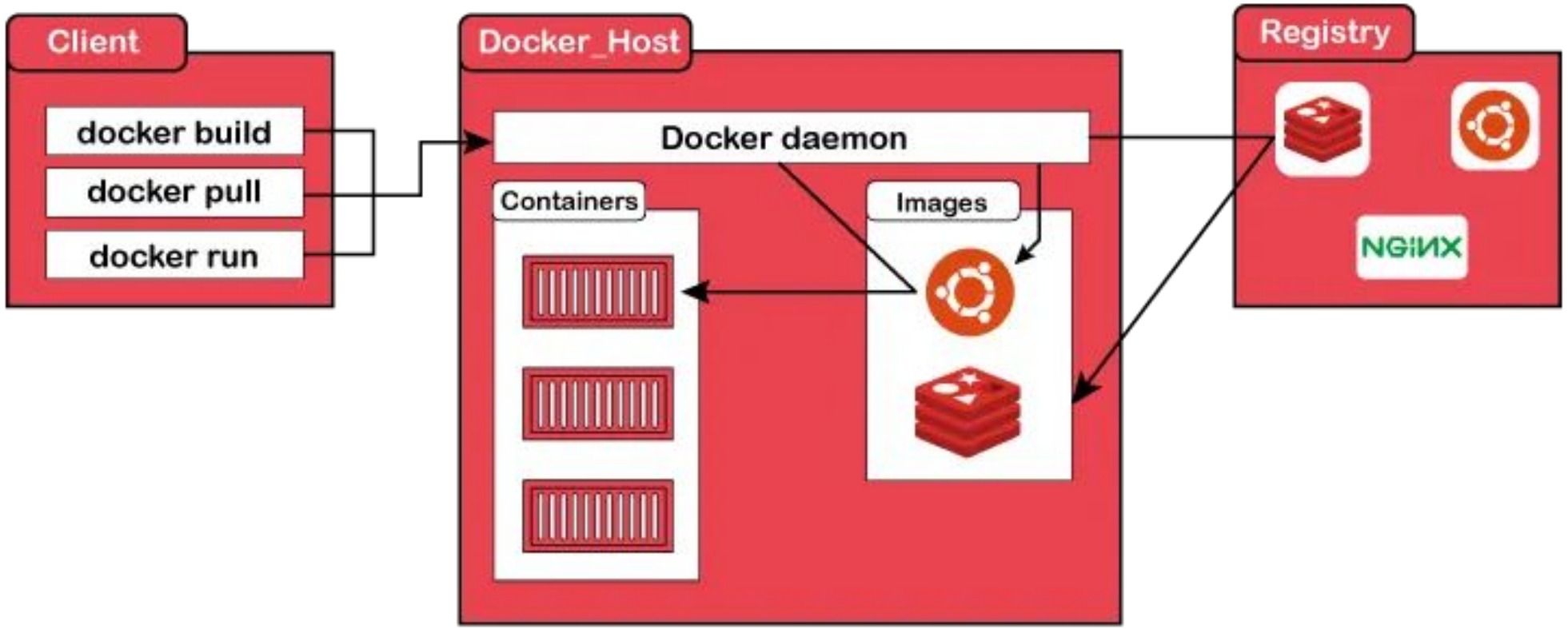
Containerization là một công nghệ ảo hoá, giúp giải quyết vấn đề này bằng cách đóng gói ứng dụng và tất cả các phụ thuộc (dependencies

Vai trò của conatainerization ?



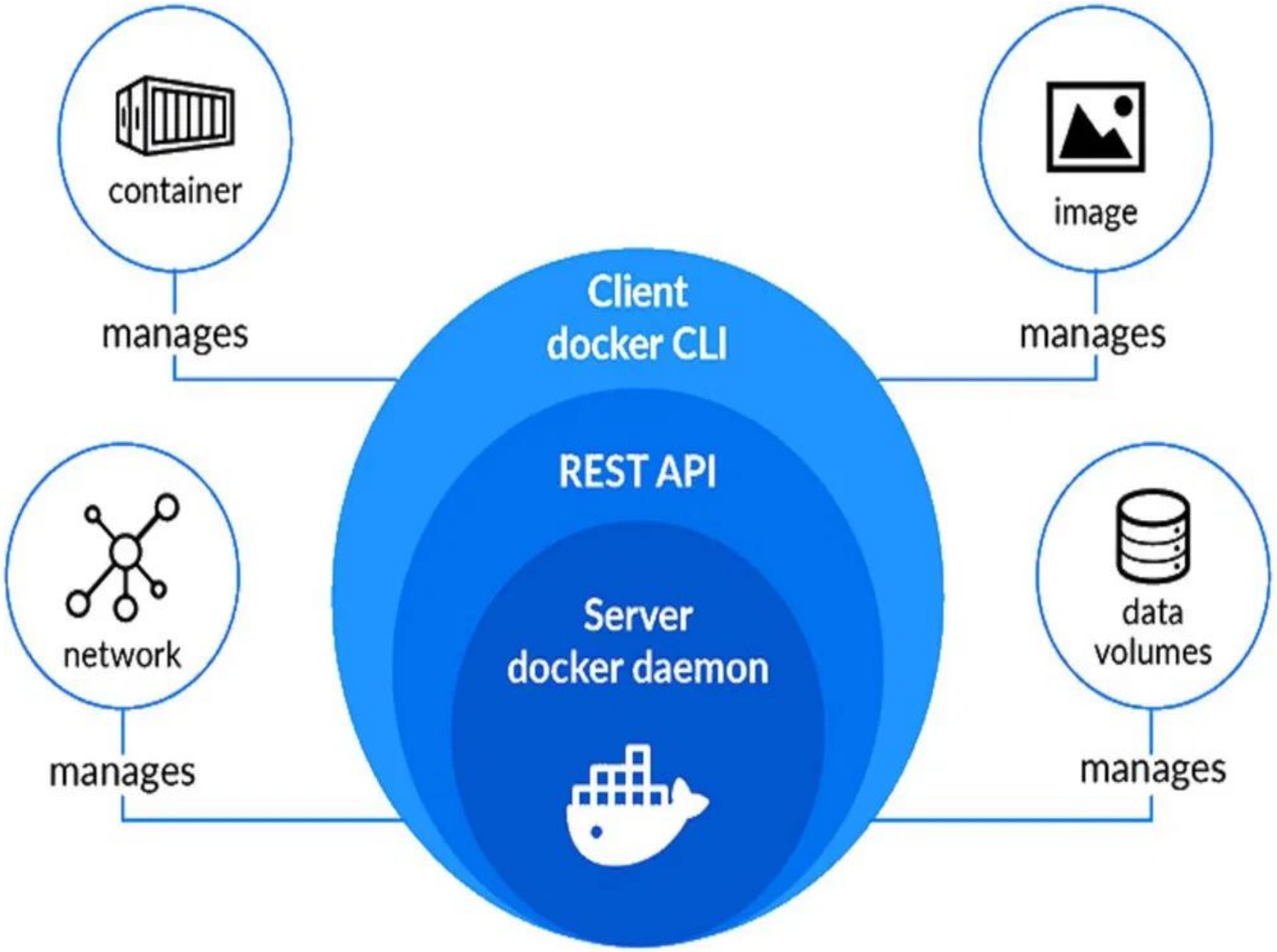
Các thành phần chính của Docker ?

Để hiểu rõ cách thức hoạt động của Docker, bạn cần làm quen với các thành phần chính của nó



Docker Engine

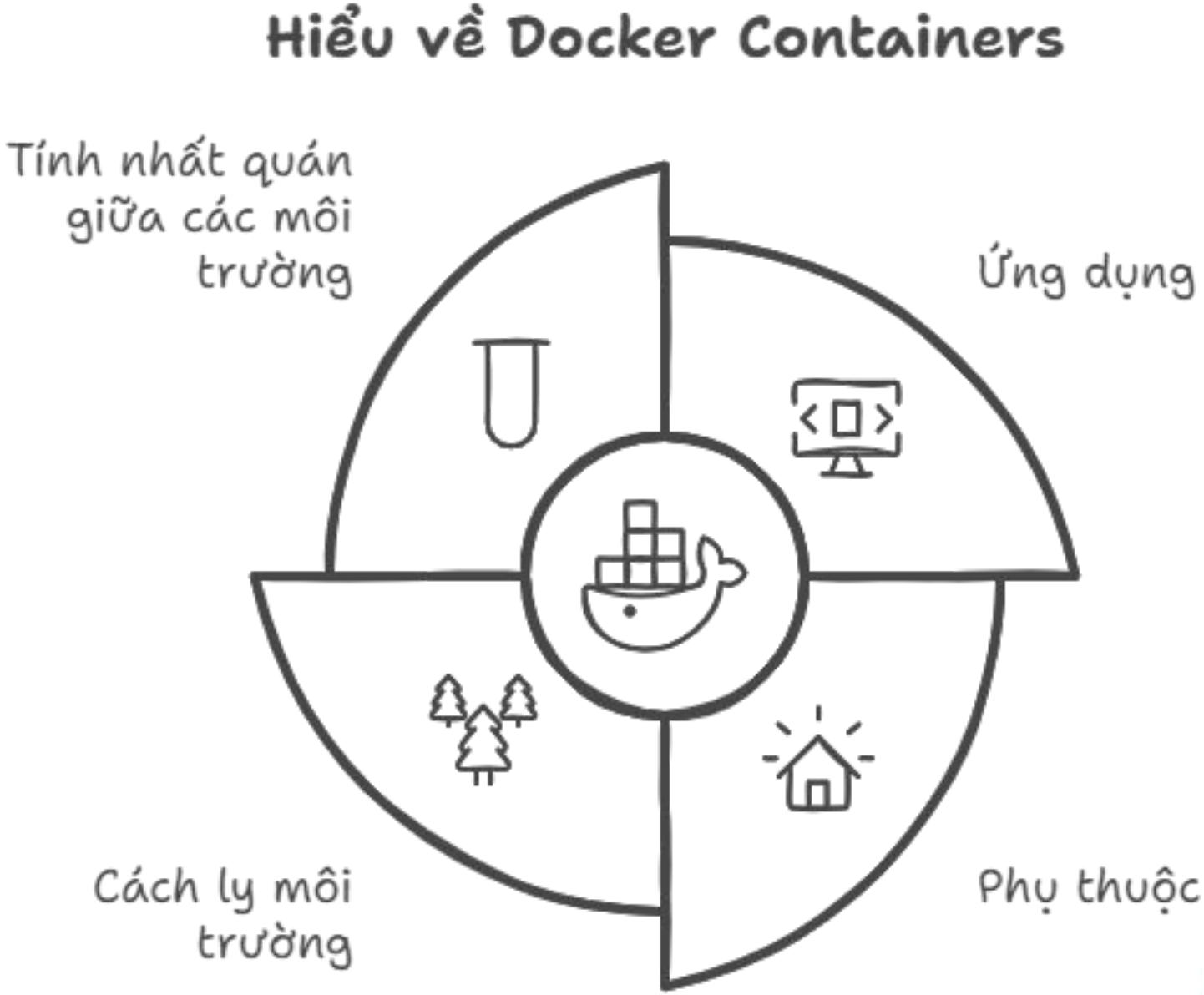
- Docker CLI
- Docker API
- Docker Daemon



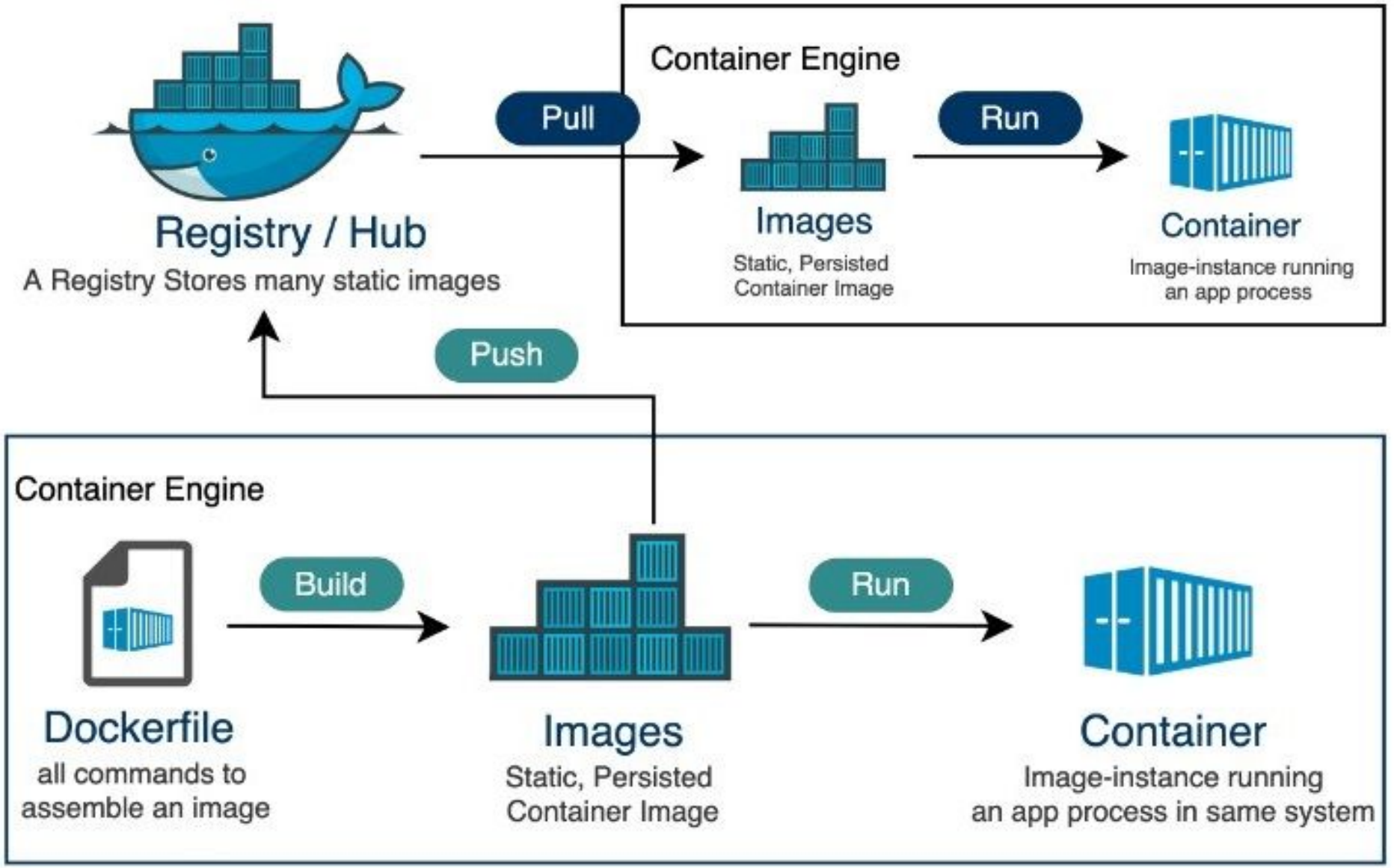
Docker Image

Quá trình chuyển đổi Docker Image thành Container

Docker Containers



Docker Hub

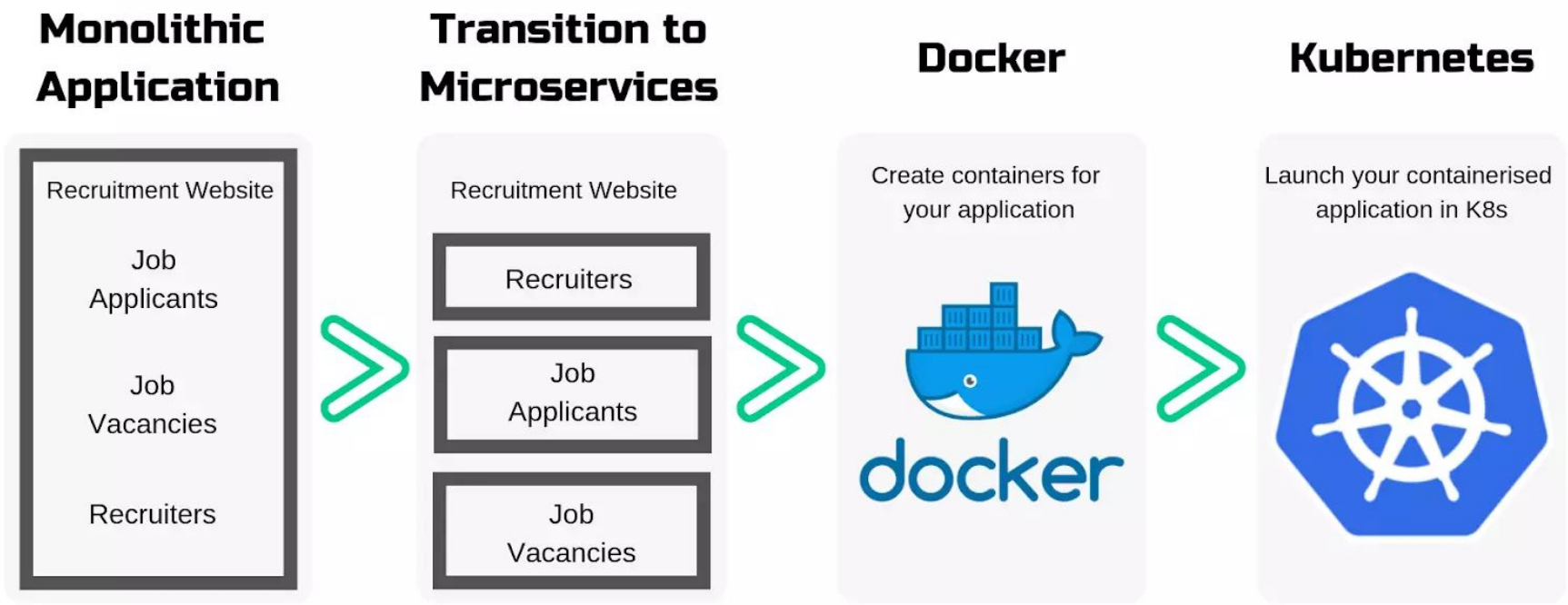


Kubernetes Là Gì ?

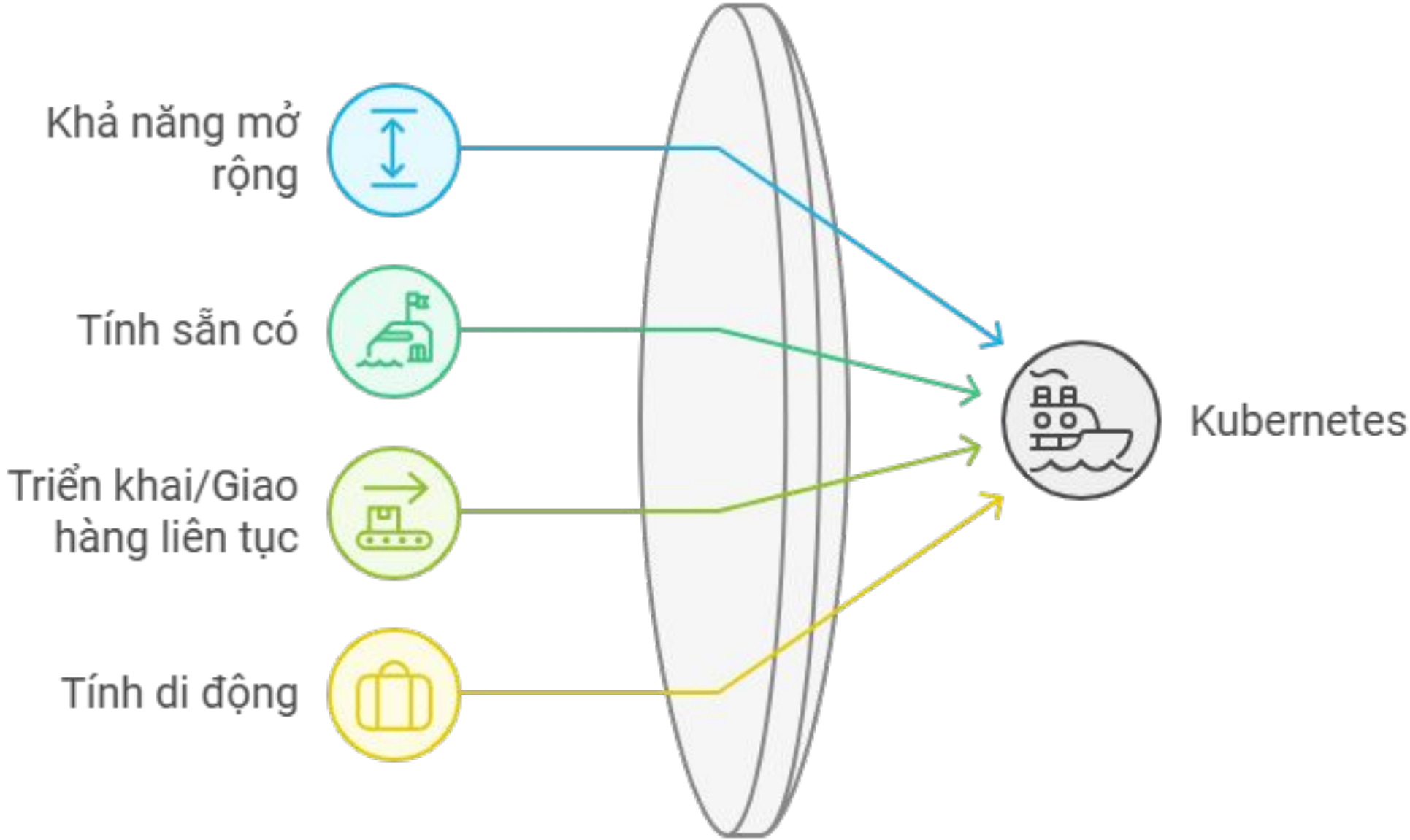
Kubernetes viết tắt K8s là một hệ thống điều phối container mã nguồn mở (open sources container orchestration tool) được phát triển bởi Google.

Nên dùng Kubernetes khi nào

Bytemark



Lợi ích của Kubernetes



Thách thức trong việc triển khai Kubernetes

Bảo mật

Những thách thức trong việc cấu hình bảo mật của Kubernetes.

Độ phức tạp

Các tính năng và tùy chọn phức tạp của Kubernetes.

Chi phí

Các tác động tài chính khi sử dụng Kubernetes.

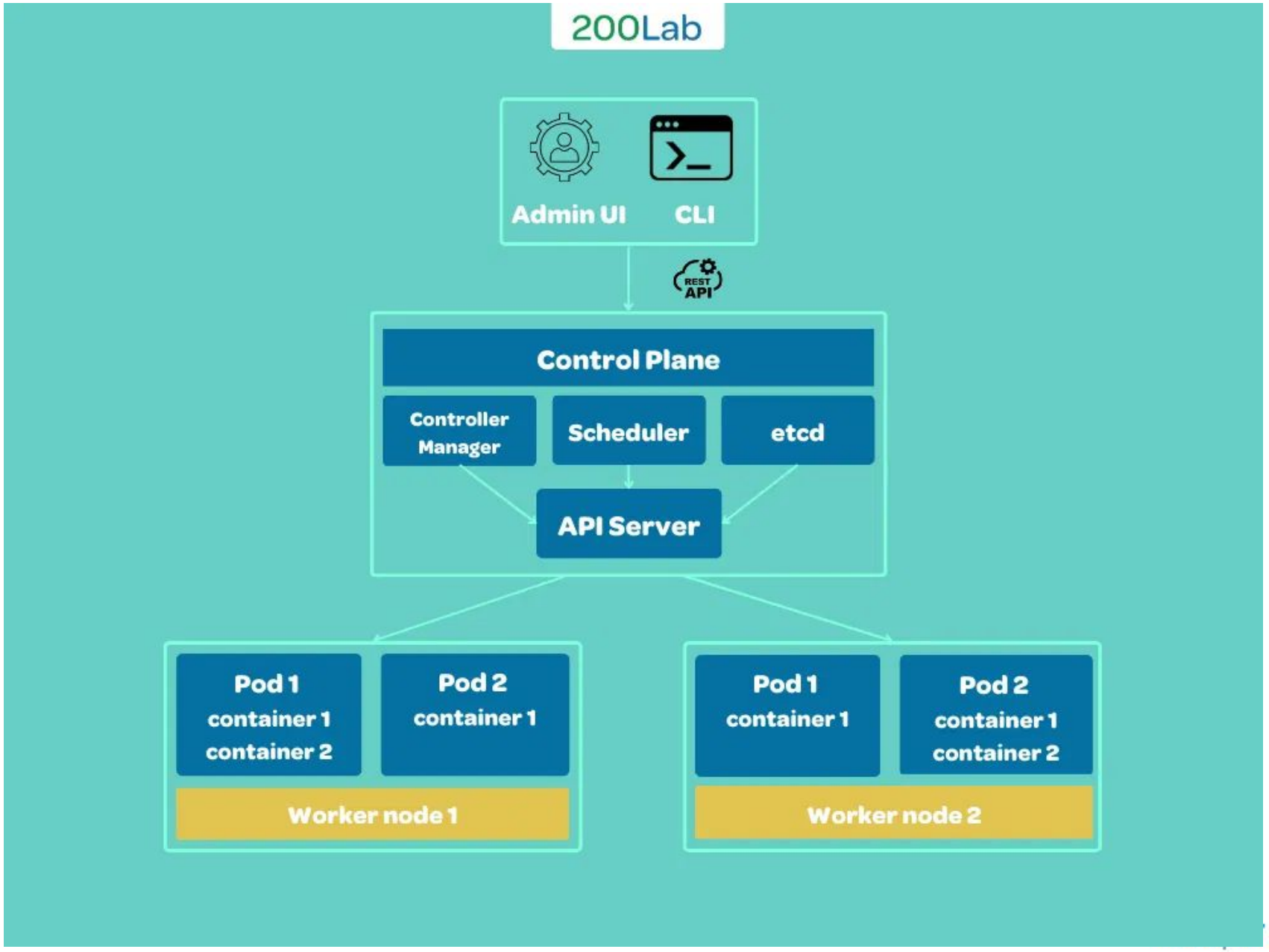
Yêu cầu tài nguyên

Nhu cầu tài nguyên máy chủ cao cho Kubernetes.

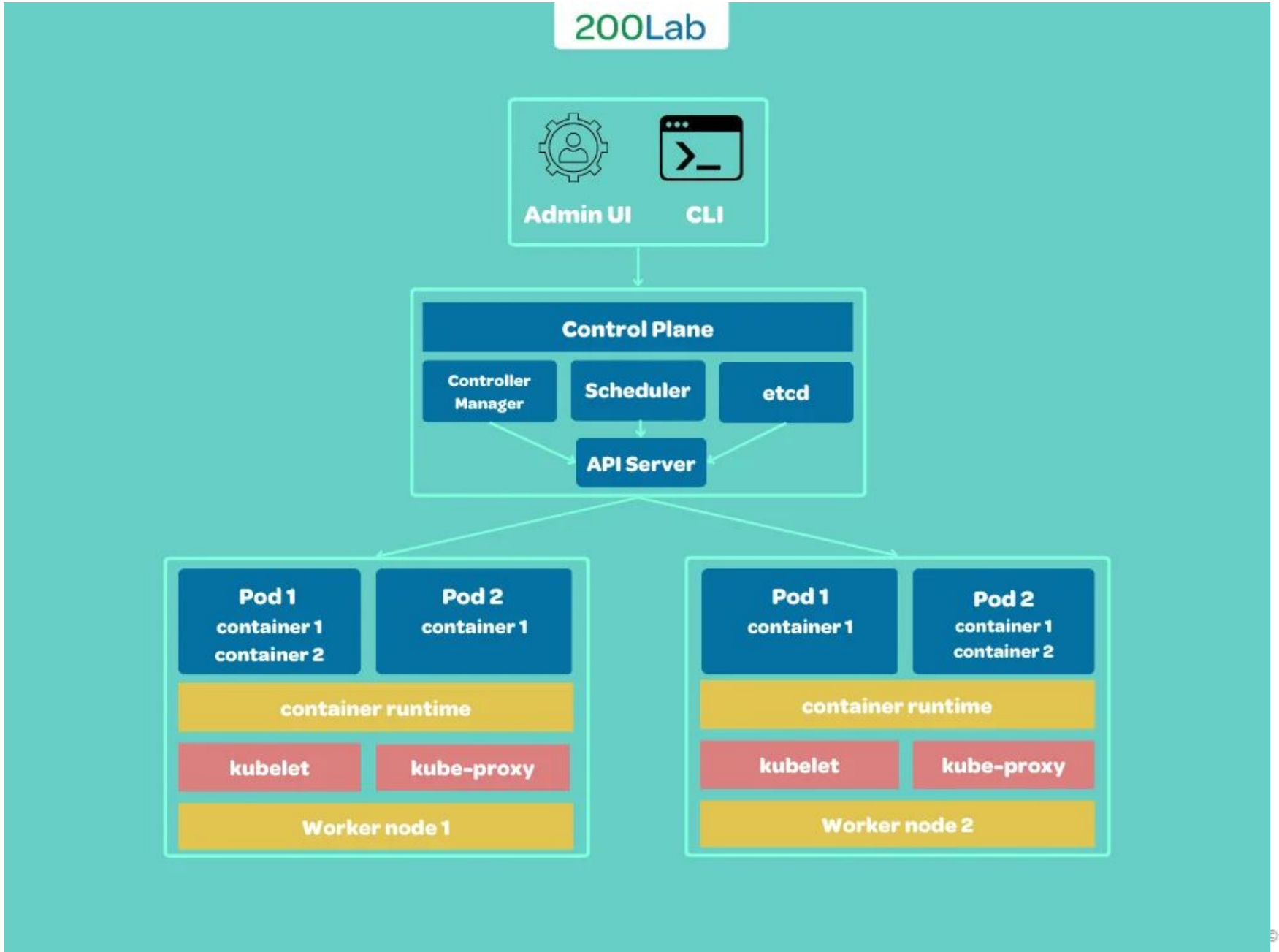
Thách thức về mạng

Sự phức tạp trong việc cấu hình mạng của Kubernetes.

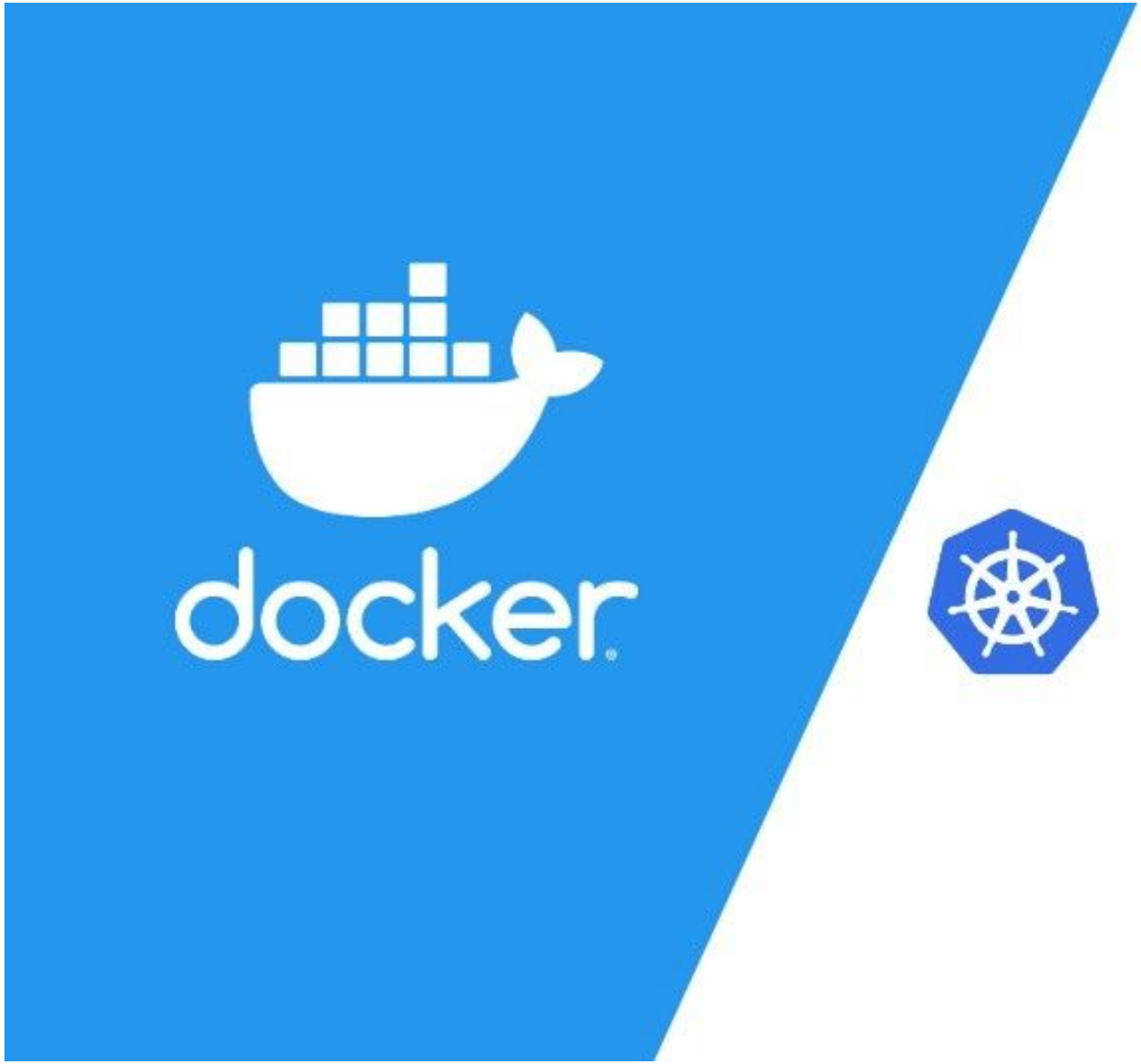
Kubernetes Architecture



Kubernetes Architecture

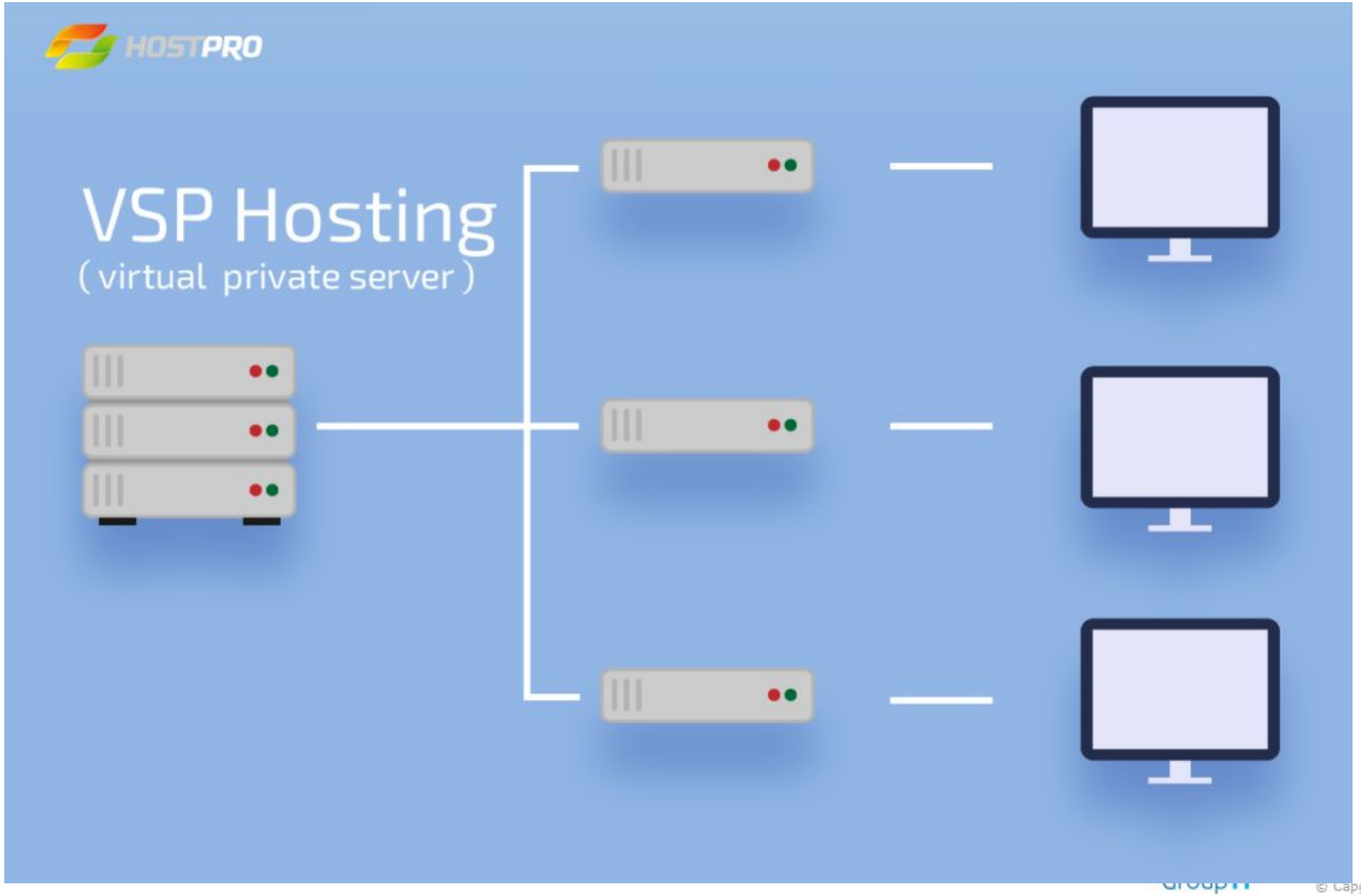


Kubernetes vs Docker

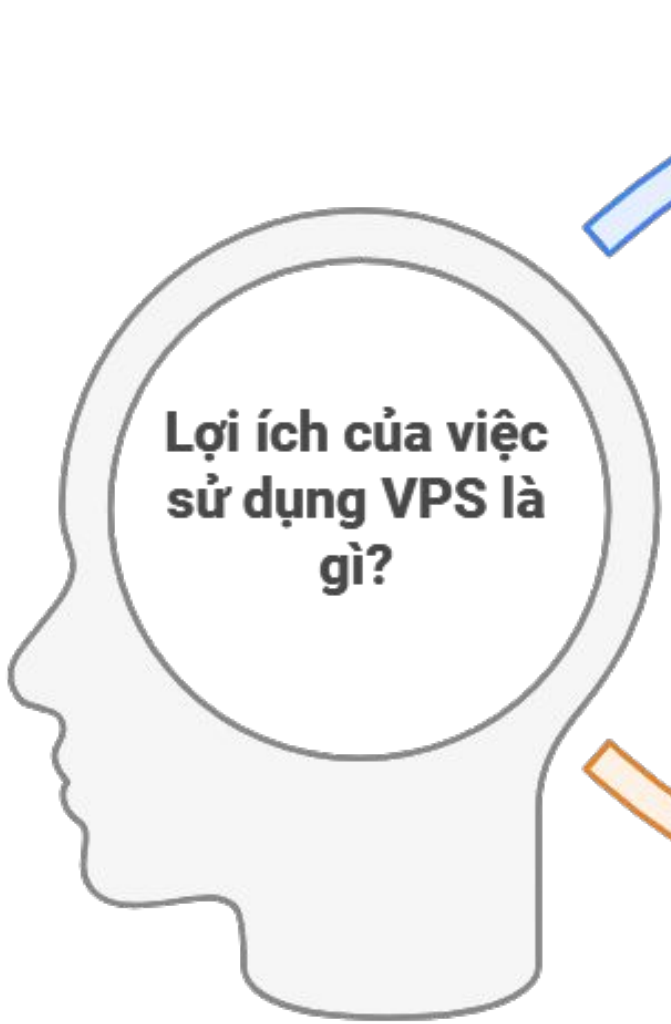


kubernetes

VPS



VPS



Hiệu suất

Cung cấp tài nguyên riêng cho hiệu suất tốt hơn.

Quản lý dễ dàng

Cung cấp quyền truy cập root để cài đặt phần mềm.

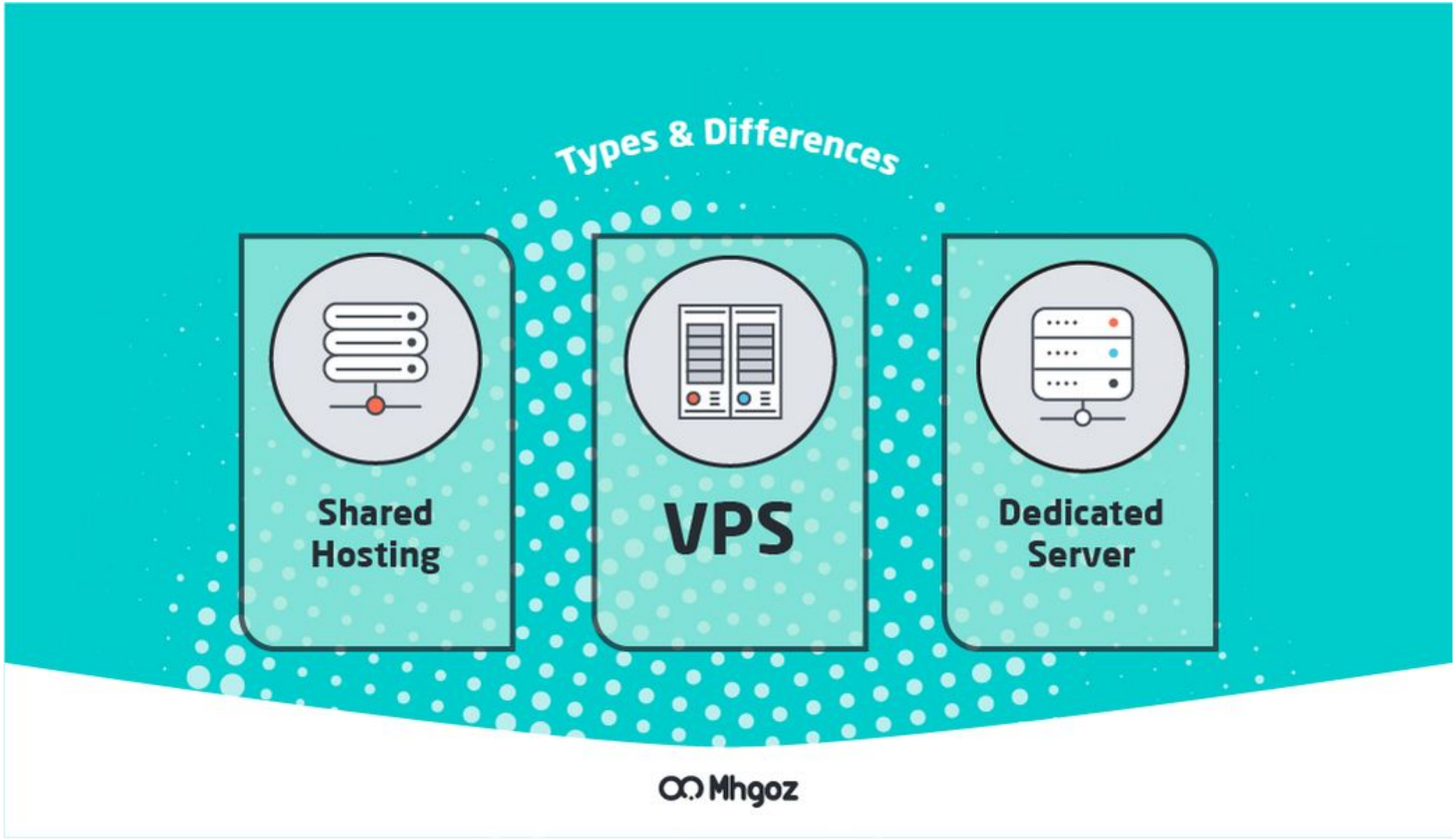
Chi phí hợp lý

Rẻ hơn so với máy chủ vật lý.

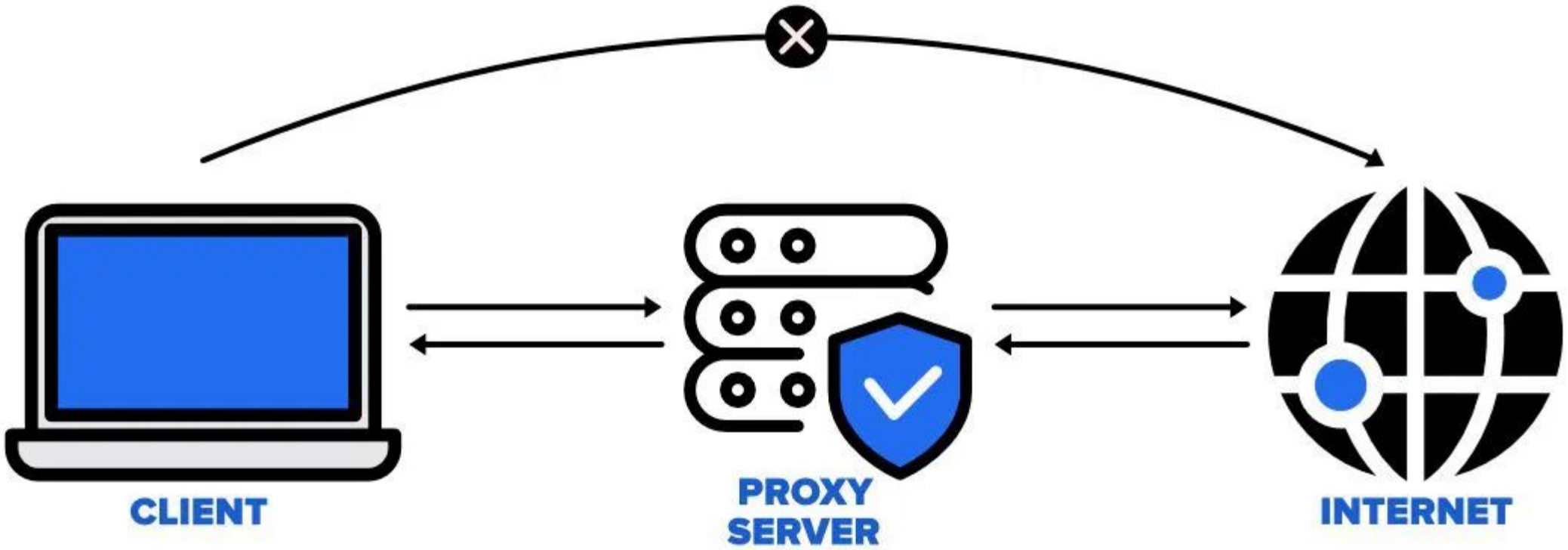
Ứng dụng đa dạng

Phù hợp cho nhiều mục đích như lưu trữ và phát triển.

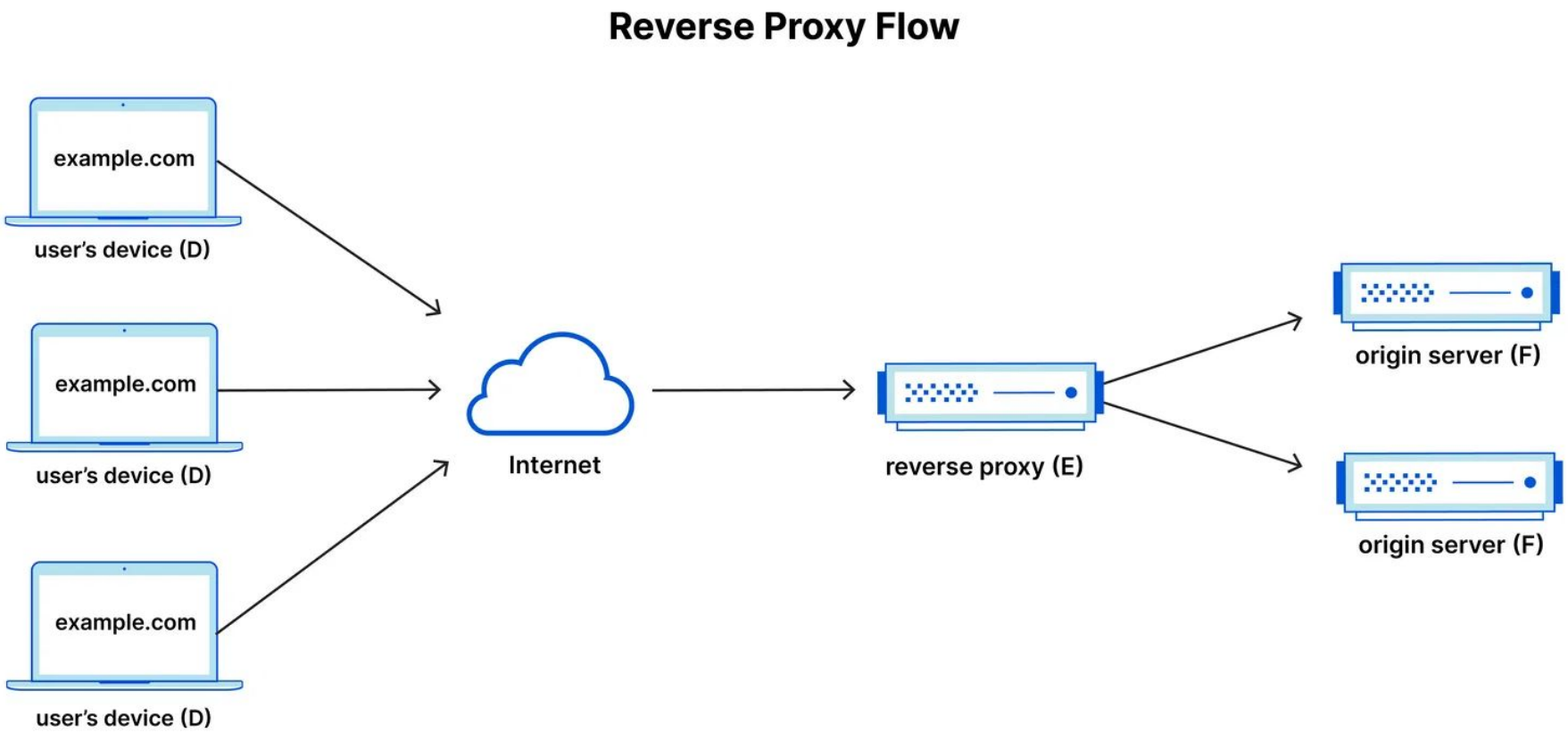
Types of Web hosting



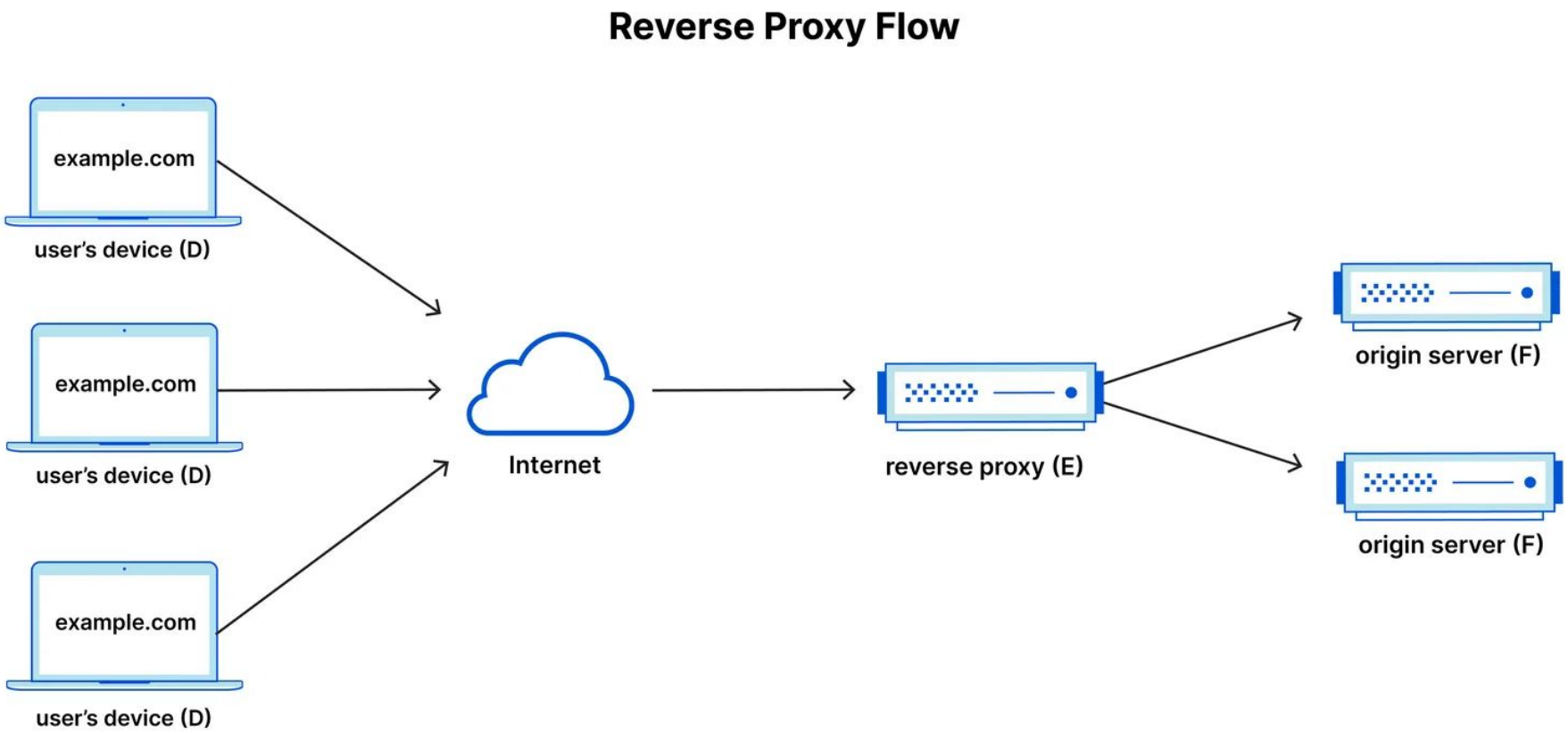
Proxy là gì ?



Reverse Proxy là gì ?



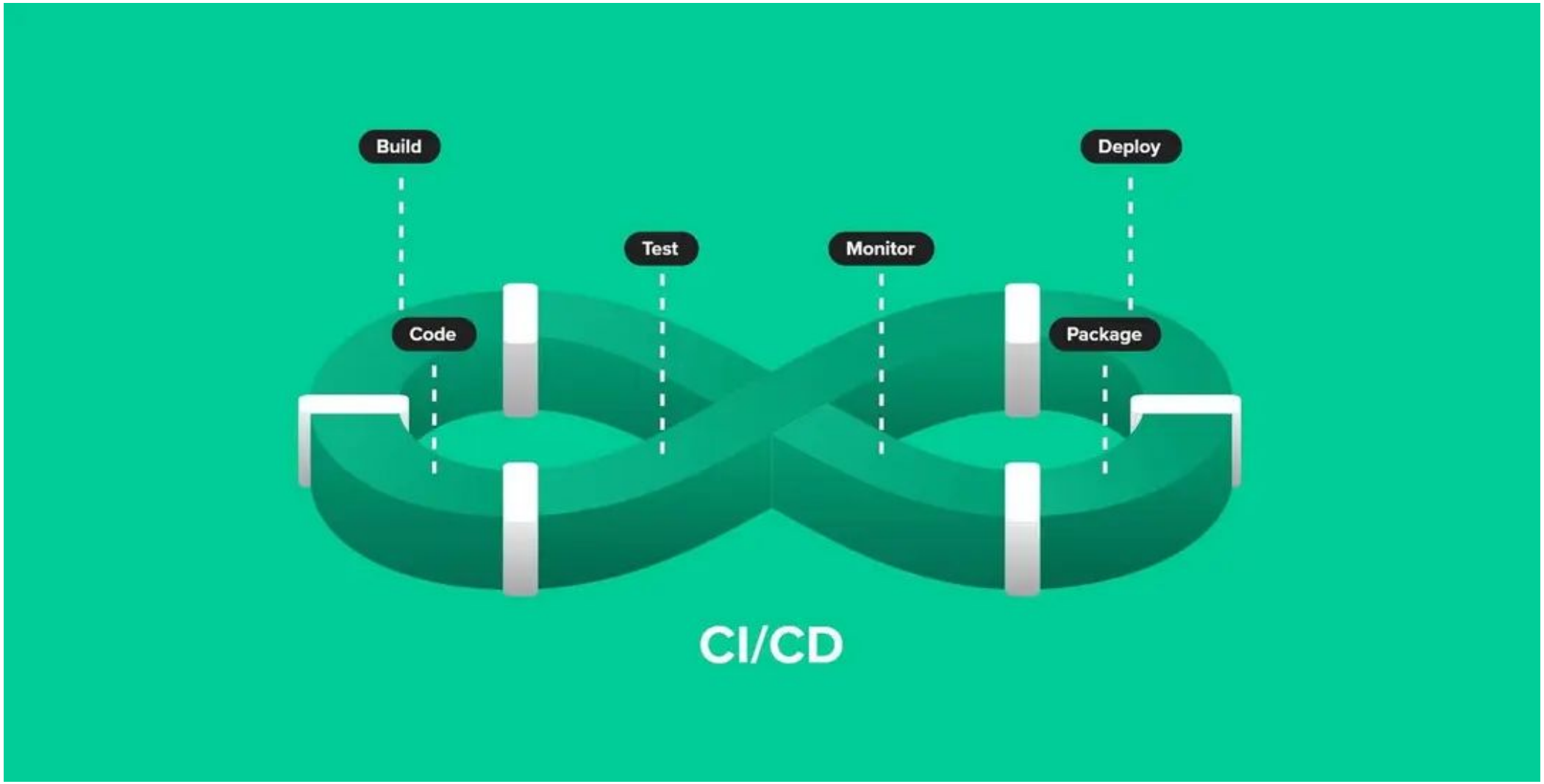
Reverse Proxy là gì ?



Tại sao cần Reverse Proxy ?

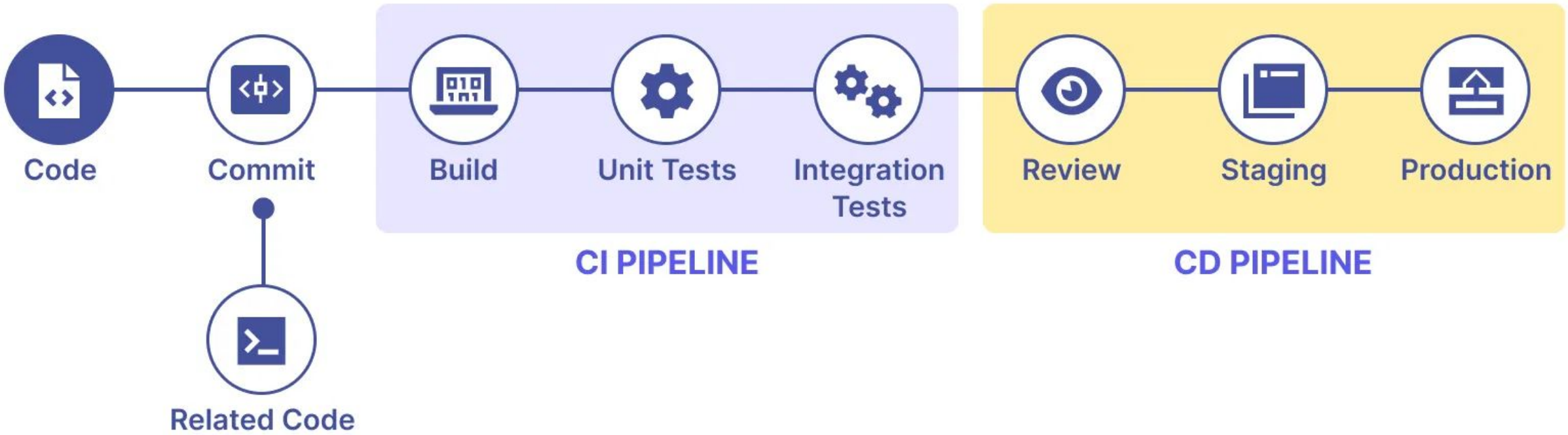
1. Phân phối tải hiệu quả:
2. Tăng tốc độ tải trang:
3. Bảo mật tốt
4. Chấm dứt SSL
5. Chuyển hướng linh hoạt

CI/CD là gì ?



CI là gì ?

CI (Continuous Integration) là quá trình liên tục kiểm tra và tích hợp code mới vào code chính của một dự án.

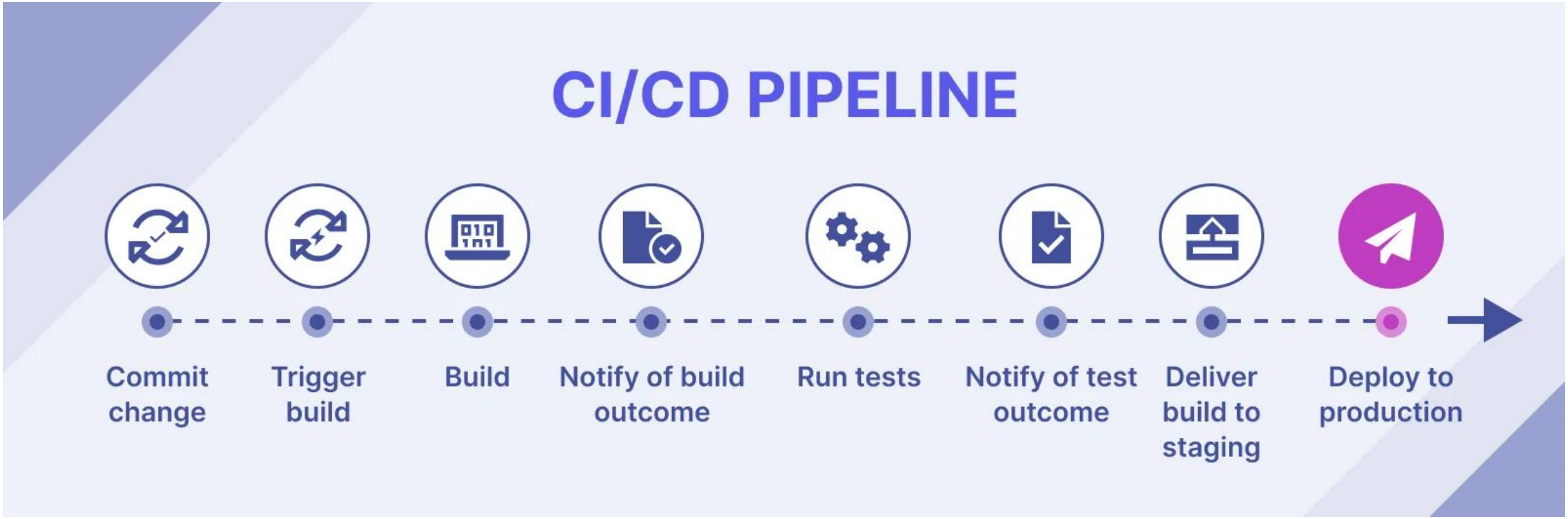


Những bước cơ bản trong CI

- Functional tests: Kiểm tra xem phần mềm có hoạt động đúng theo mong đợi hay không. Đây thường là các bài kiểm tra tự động mà đội ngũ QA (Quality Assurance) phát triển để đảm bảo tính năng hoạt động đúng. VD: Người dùng có thể thêm sản phẩm vào giỏ hàng và thực hiện thanh toán thành công hay không.
- Security scans: Kiểm tra code có lỗ hổng bảo mật nào không, chẳng hạn như khả năng bị SQL Injection hay XSS (Cross-Site Scripting).
- Code quality scans: Đảm bảo code tuân thủ các tiêu chuẩn, ví dụ như độ dài hàm, cách sử dụng khoảng trắng, và các quy tắc coding style. VD: Tiêu chuẩn như đặt tên biến.
- Performance tests: Kiểm tra xem code có đáp ứng được yêu cầu về hiệu suất không, chẳng hạn như thời gian xử lý yêu cầu hoặc khả năng chịu tải.
- License scanning: Kiểm tra xem tất cả các thư viện hoặc công cụ bạn sử dụng có giấy phép phù hợp hay không, để tránh các vấn đề pháp lý.
- Fuzz testing: Gửi các dữ liệu bất thường, như chuỗi ký tự quá dài hoặc số không hợp lệ, vào ứng dụng để kiểm tra xem nó có bị crash hay gặp lỗi bất ngờ không.

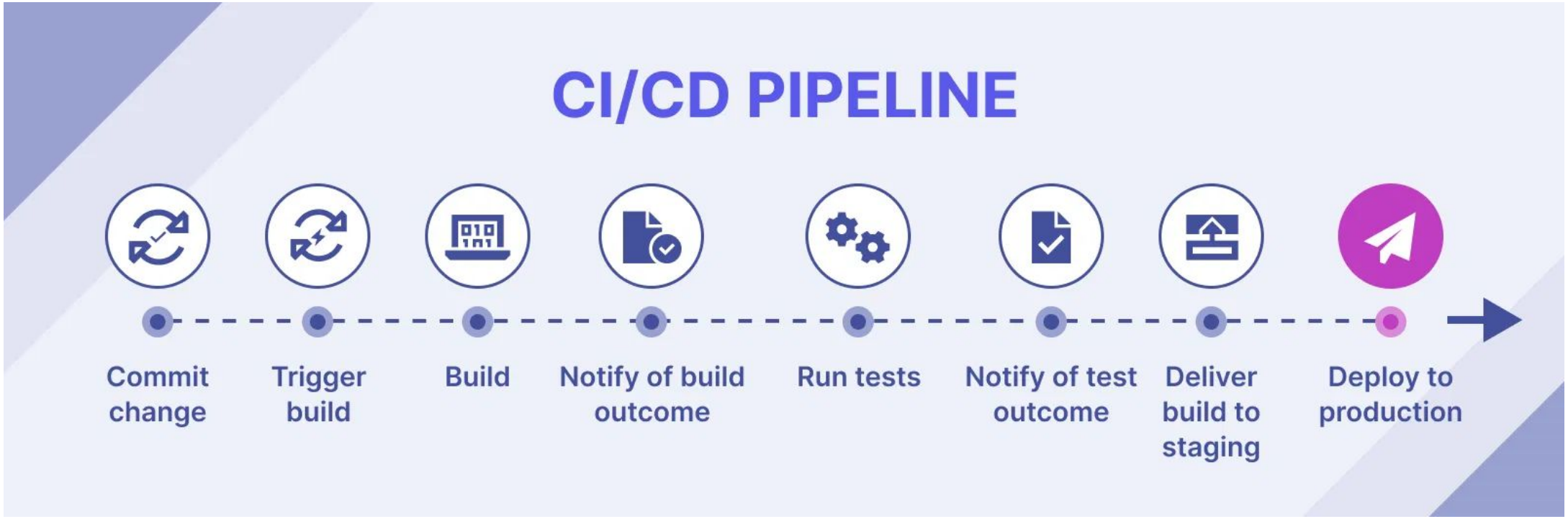
CD là gì ?

CD là quá trình đưa code từ nơi bạn viết lên một nơi mà người dùng có thể sử dụng được, như website hoặc ứng dụng di động



Những bước cơ bản trong CD

- Build: Trước khi triển khai code, đôi khi chúng ta cần phải thực hiện quá trình build cho các ngôn ngữ biên dịch (compiled language) như C, C++, Java, ...
- Deploy: Triển khai code: đẩy Docker image lên repository, dùng dòng lệnh để đưa code lên AWS, ... Triển khai có thể được tự động hoặc thủ công, tùy thuộc vào cách bạn thiết lập.



Lợi ích của CI/CD là gì ?

- **Thứ nhất**, giảm lỗi phát sinh. Nhờ vào kiểm tra tự động, chúng ta có thể phát hiện lỗi ngay từ sớm.
- **Thứ hai**, tăng tốc độ phát triển. Thay vì phải chờ đợi kiểm tra thủ công, mọi thứ được tự động hóa.
- **Thứ ba**, tạo sự ổn định. Chúng ta không lo triển khai nhầm phiên bản hoặc quên bước kiểm tra."