



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 cqrss-es



Microservices Architecture

1 Introduction Microservices Architecture

2 Architecture

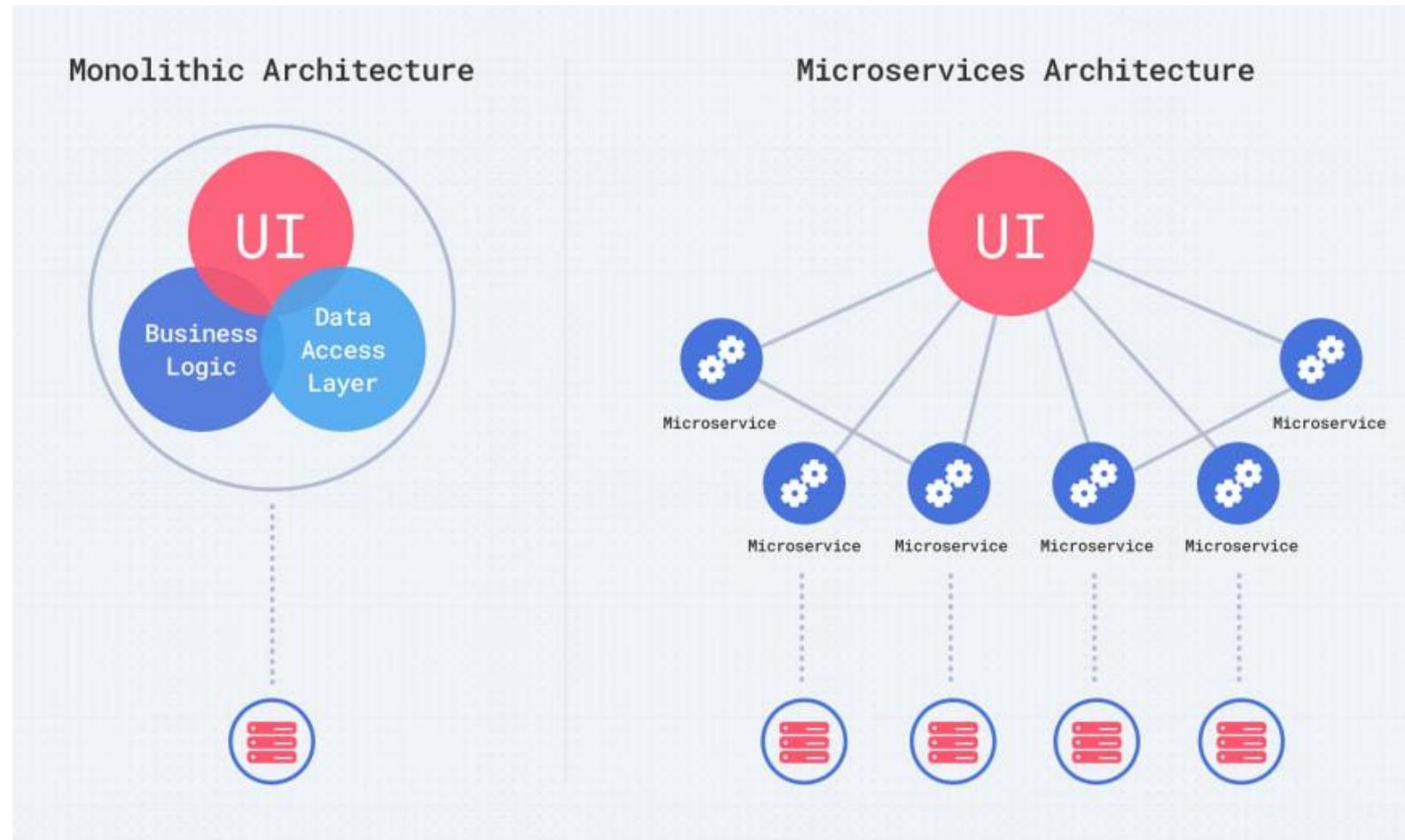
3 Communicating between

4 Securing Microservice



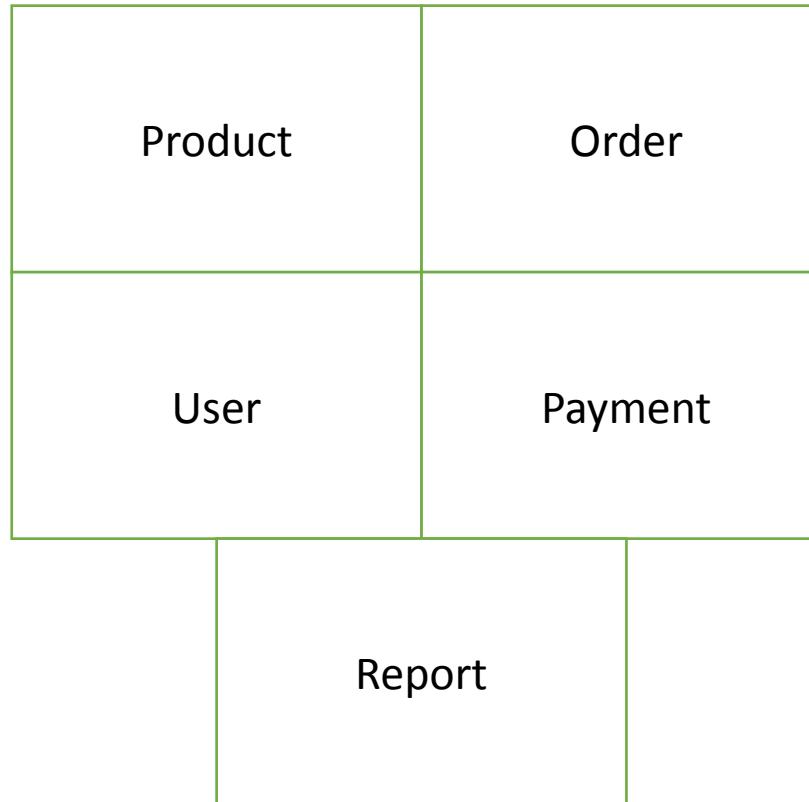
Introduction

Micro -> Nhỏ
Service -> Dịch vụ





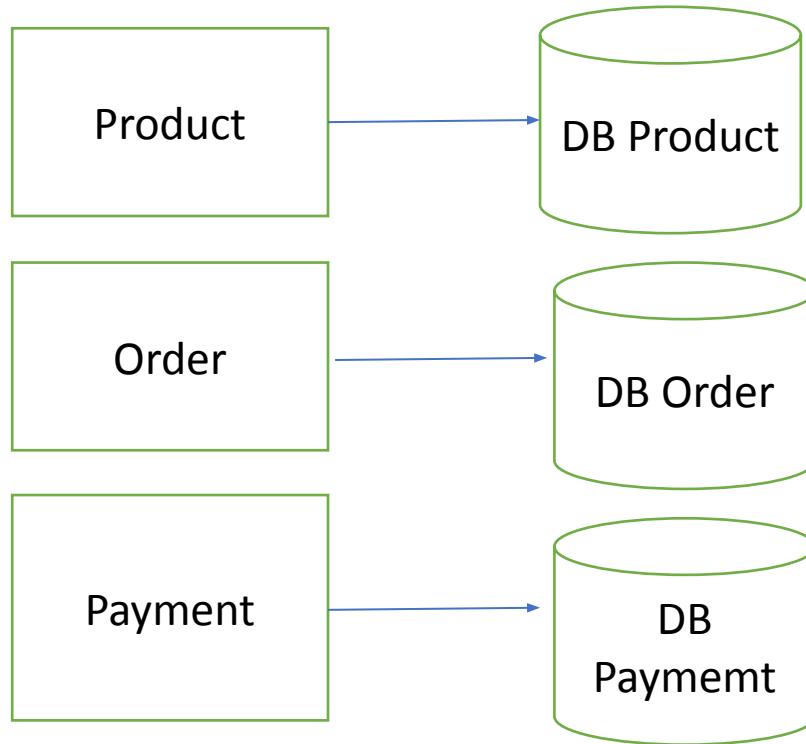
Problem Monolithic



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



Solution



Microservice Architecture

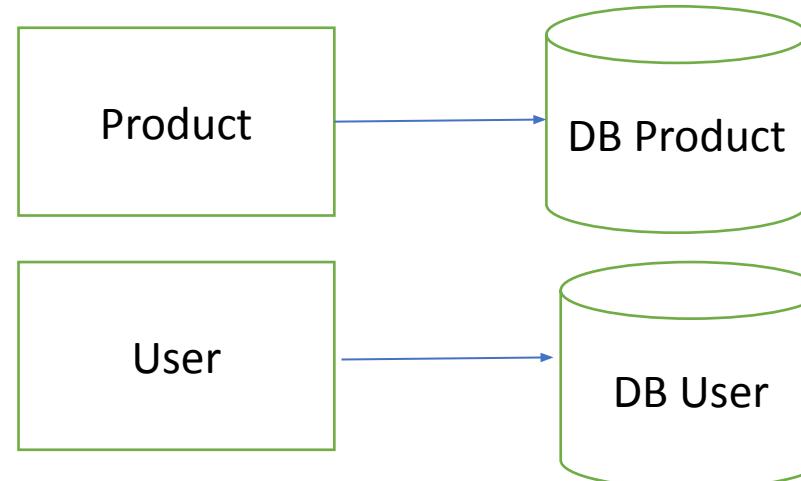
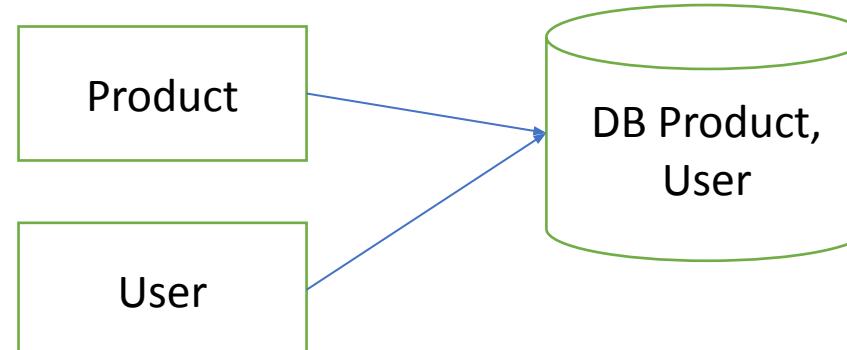
- **Ưu điểm :**
- Dễ dàng scale tiết kiệm chi phí
- Dễ dàng deploy
- Tự do sử dụng các công nghệ ở mỗi service
- Khó sập hơn monolithic

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



Architecting Microservices

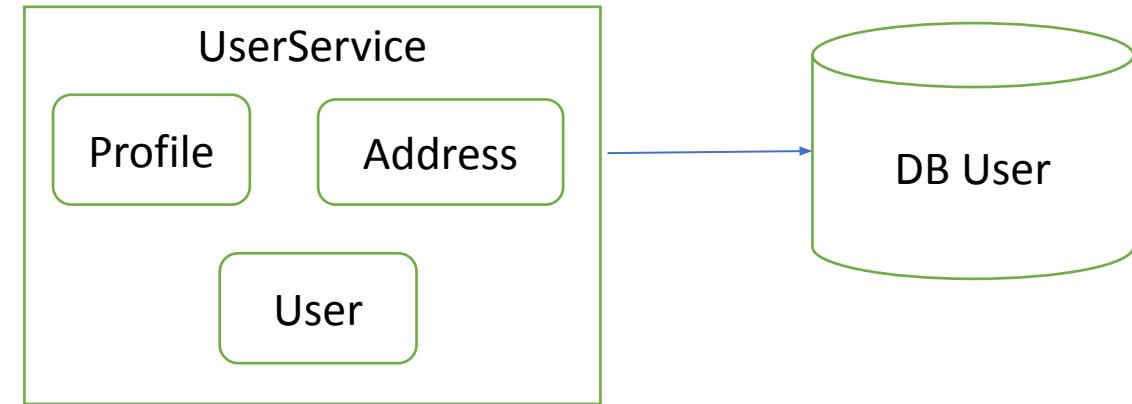
- Mỗi microservice đều phải có cơ sở dữ liệu riêng
- Tính độc lập





Architecting Microservices

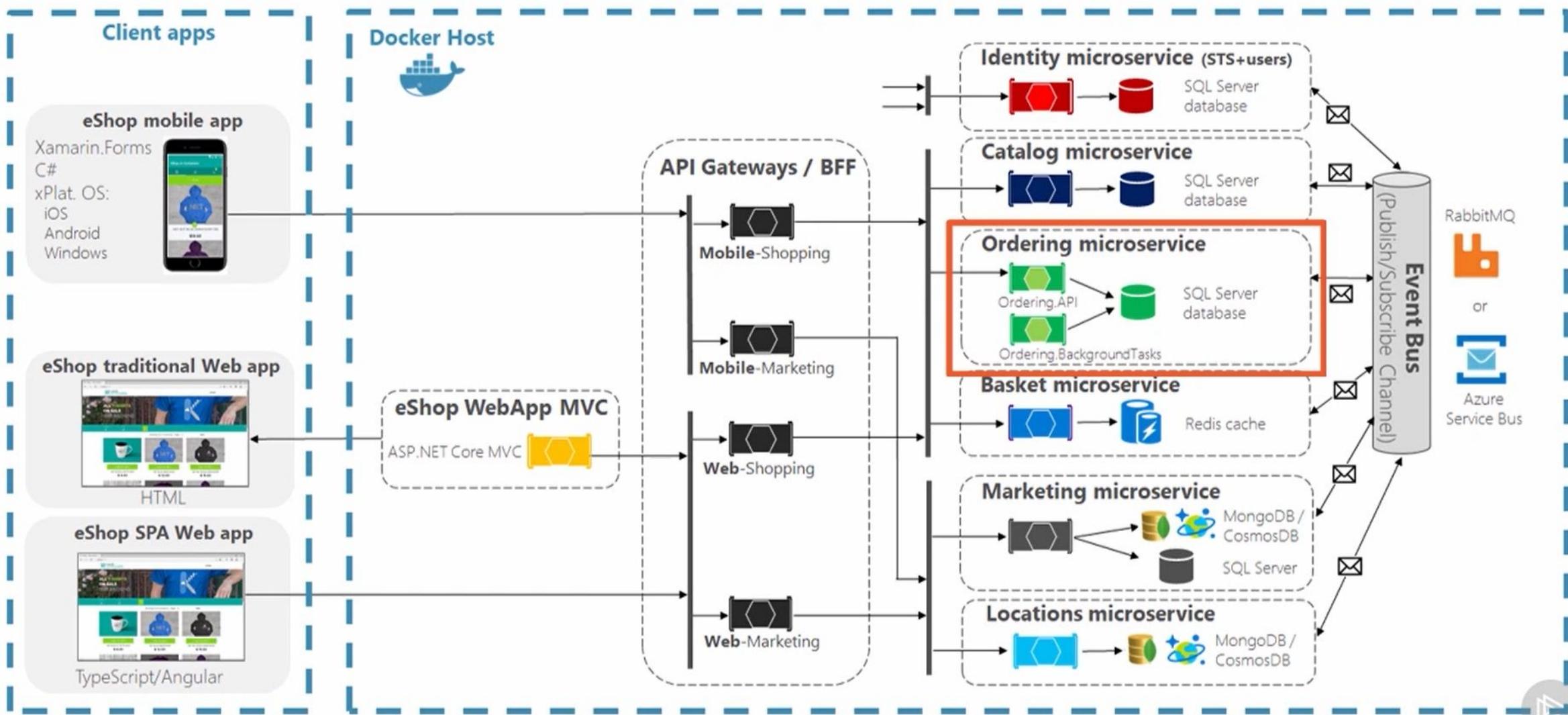
- Xác định ranh giới Service
- Giảm thiểu việc gọi request ở nhiều nơi
- Quản lý code dễ dàng





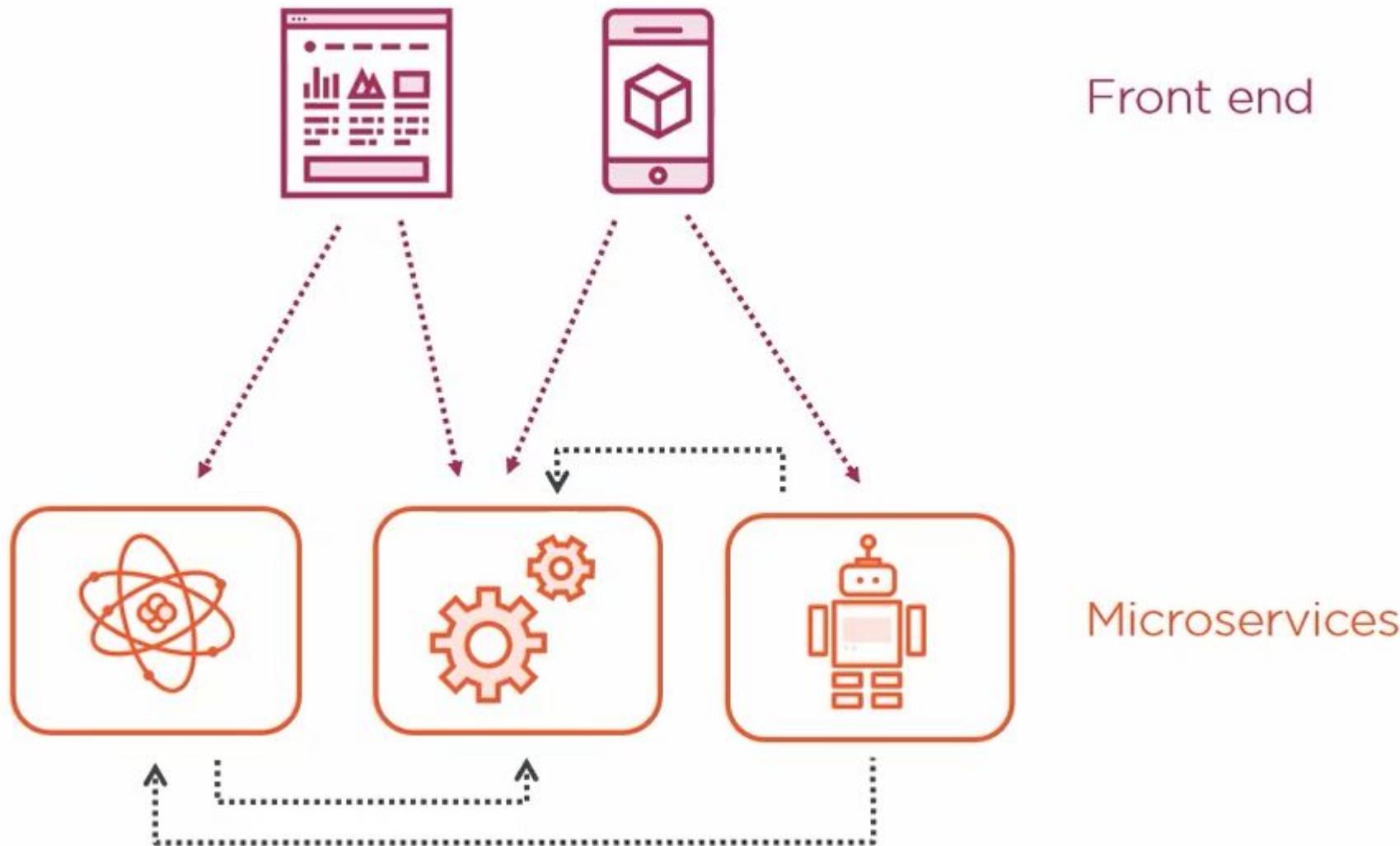
Architecting Microservices

eShopOnContainers Architecture



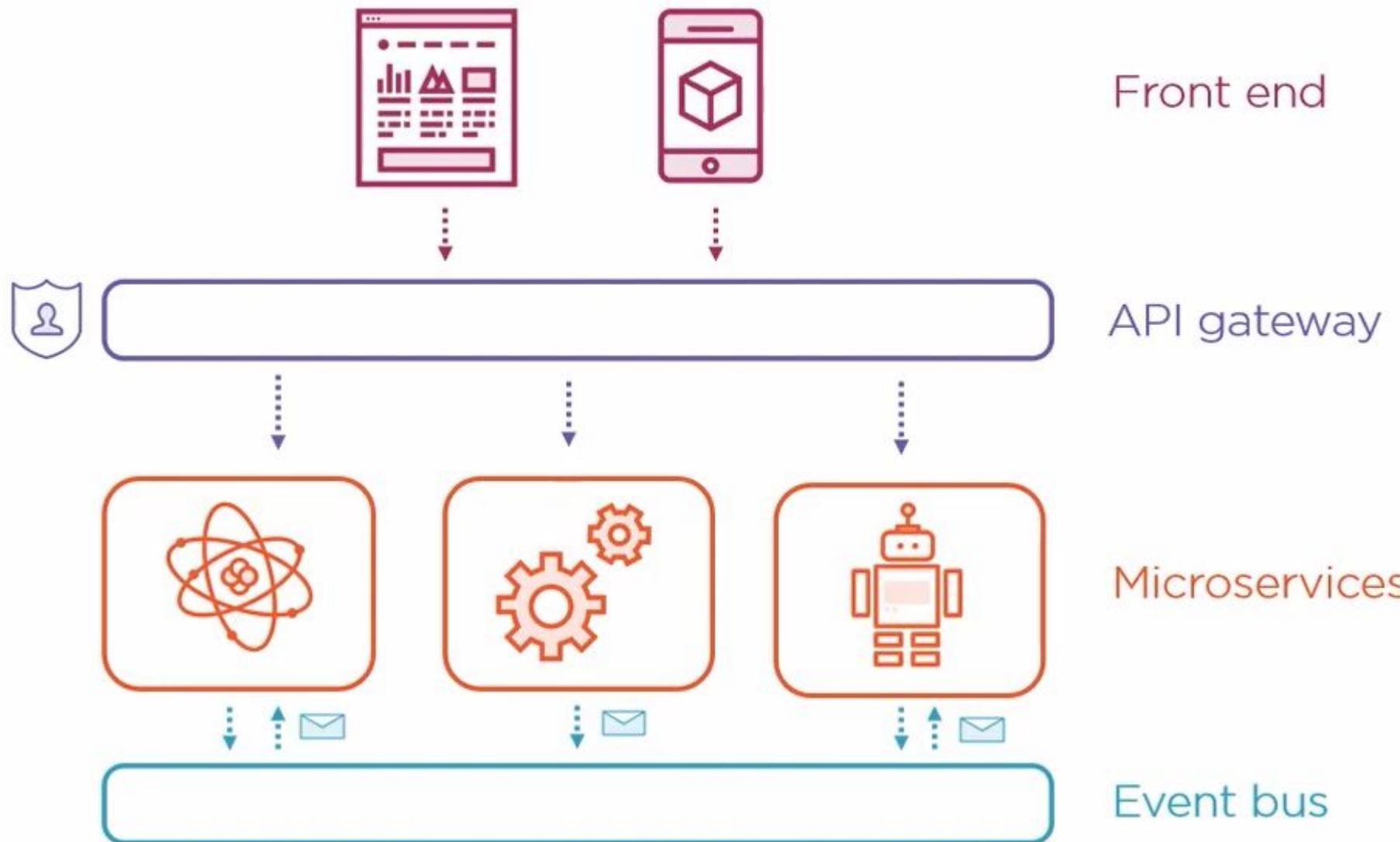


Communication Microservices



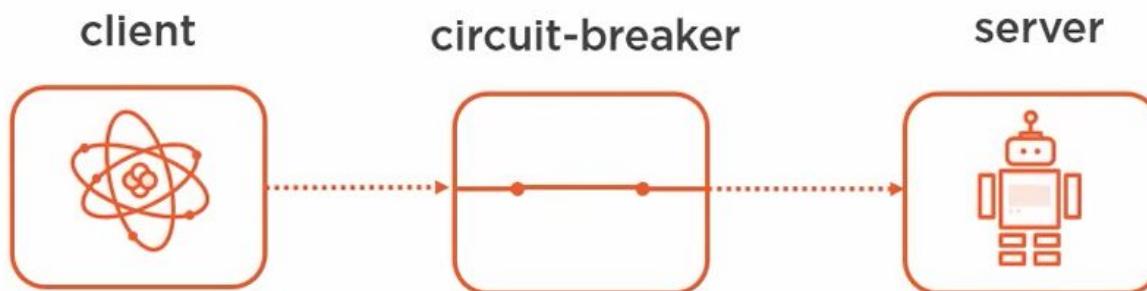
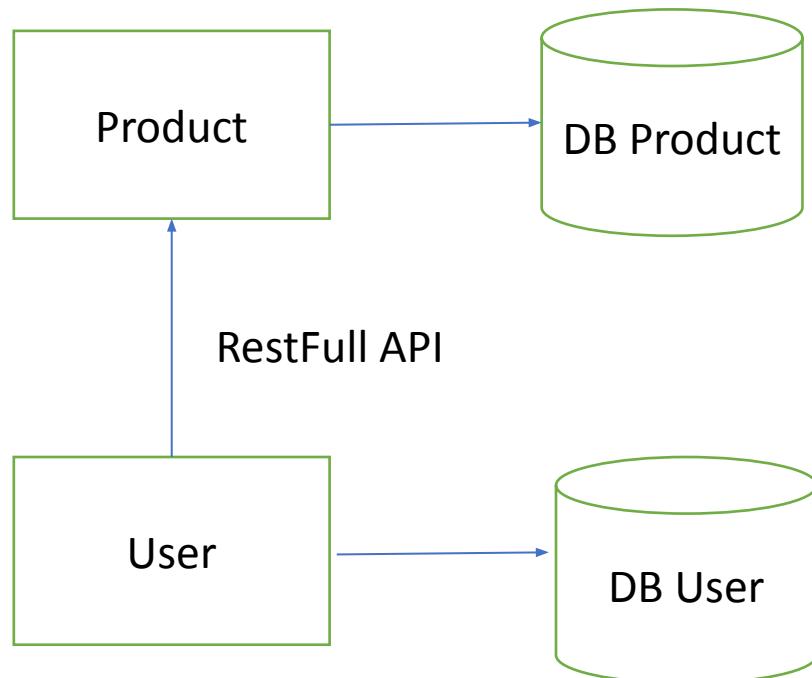


Communication Microservices



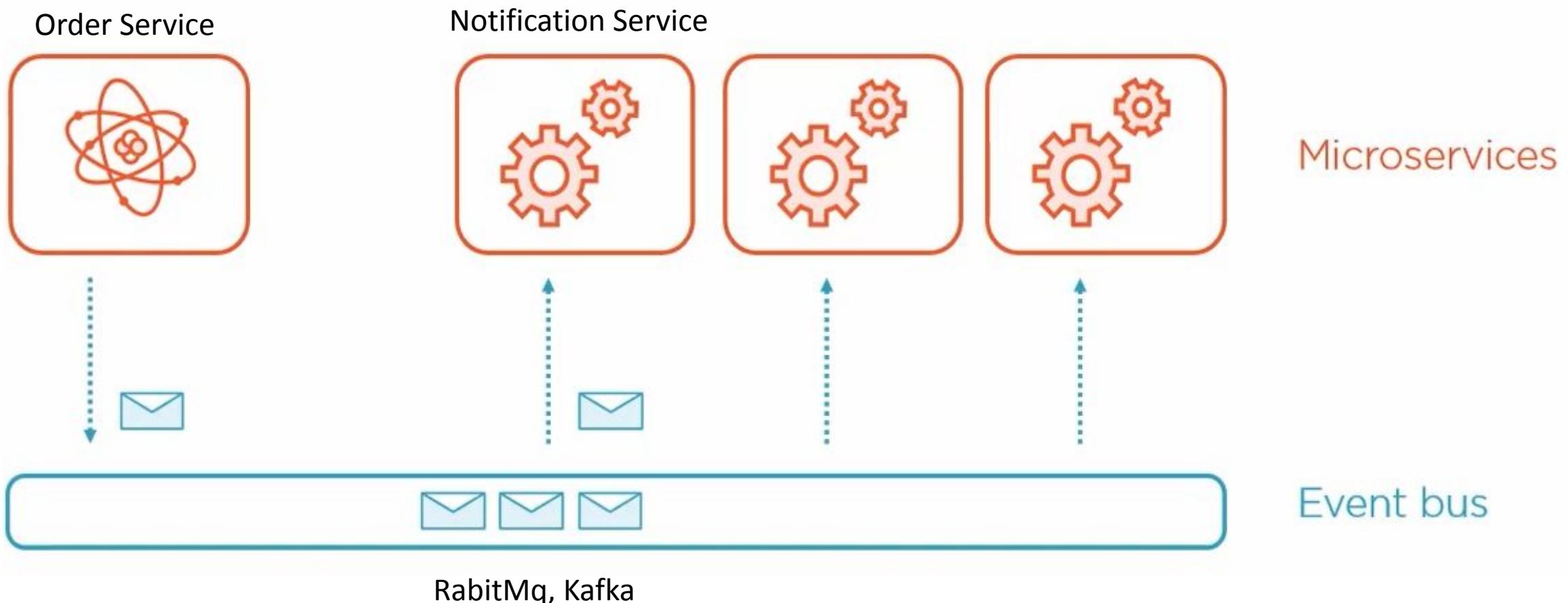


Synchronous Communication



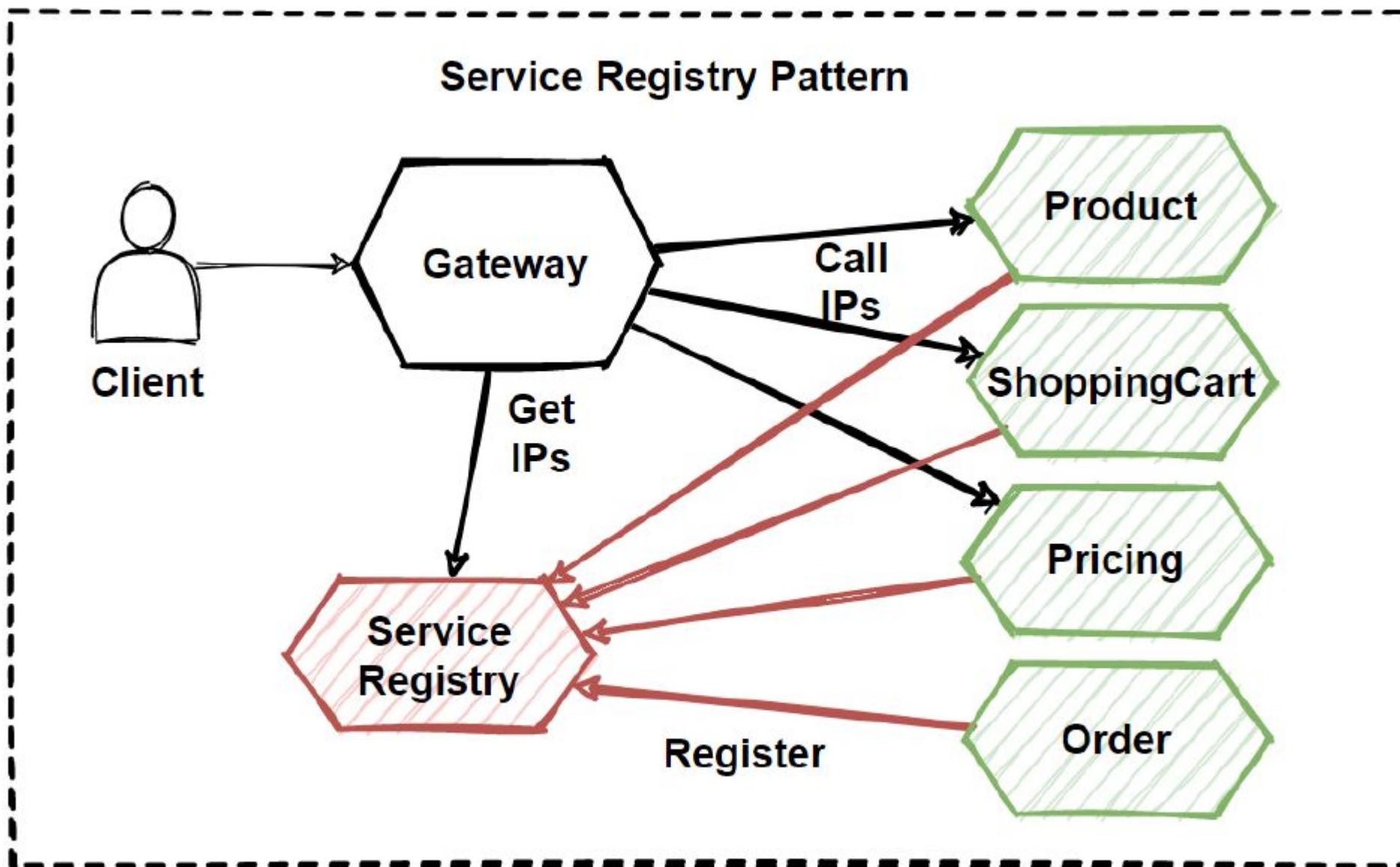


Asynchronous Communication





Service Discovery





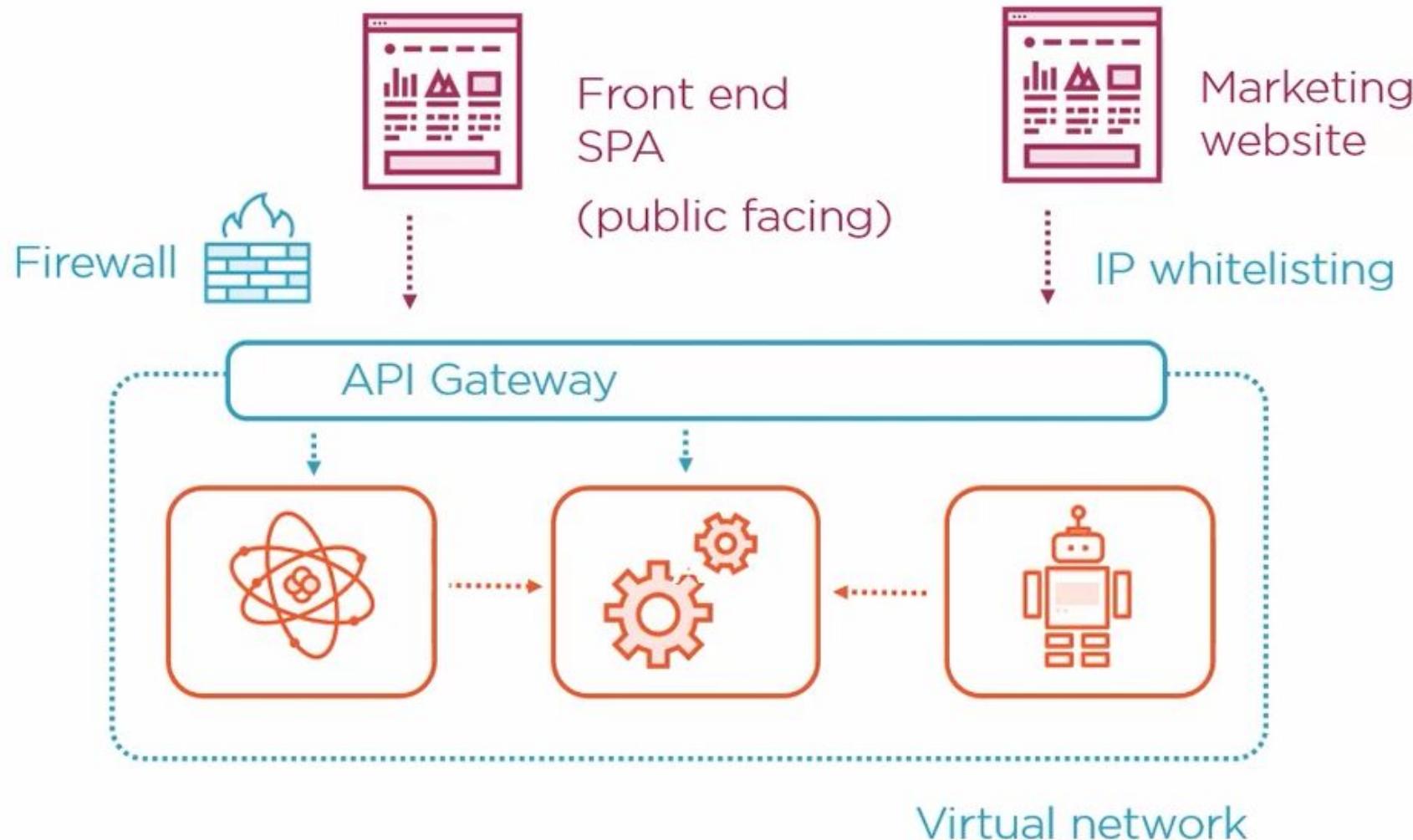
Securing Microservice

- 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.
-

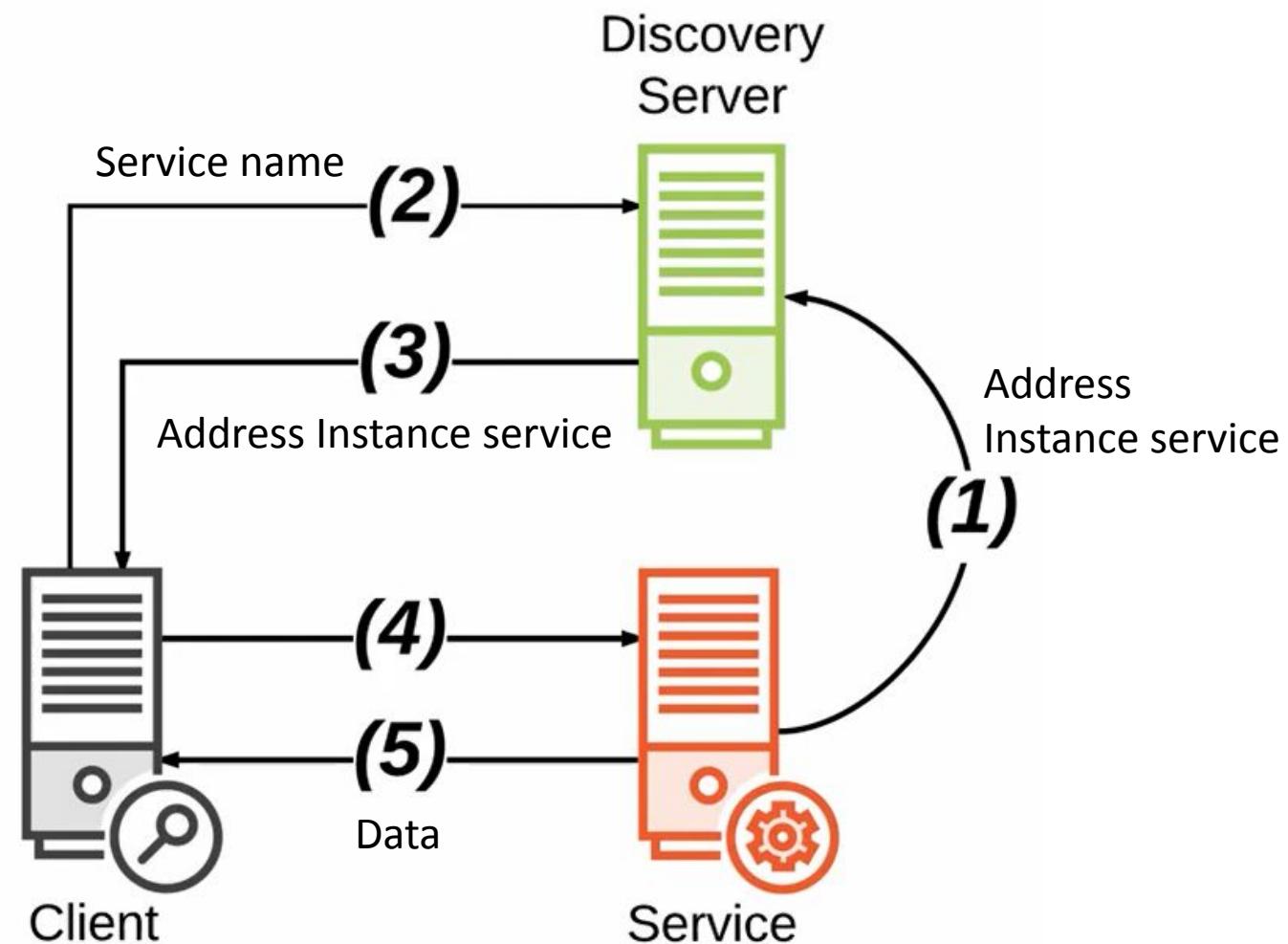


Spring Cloud

Provides a set of tools for common patterns in distributed systems. Useful for building and deploying microservices.



Service discovery





Discovery server

- Tạo discovery server với spring cloud Eureka



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.tanthanh2706

Artifact discoveryserver

Name discoveryserver

Description Demo project for Spring Boot

Package name com.tanthanh2706.discoveryserver

Packaging Jar War

Java 18 17 11 8

Dependencies ADD DEPENDENCIES... CTRL + B

Eureka Server SPRING CLOUD DISCOVERY
spring-cloud-netflix Eureka Server.

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.

Spring Boot DevTools DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.



Discovery server

DiscoverserverApplication.java ×

```
1 package com.tananh.discoverserver;
2
3+import org.springframework.boot.SpringApplication;□
4
5
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

localhost:8761

[spring Eureka](#) [Toggle navigation](#)

System Status

Environment	test	Current time	2022-05-24T16:01:55 +0700
Data center	default	Uptime	00:01
		Lease expiration enabled	false
		Renews threshold	1
		Renews (last min)	0

DS Replicas

[localhost](#)

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

General Info

Name	Value
total-avail-memory	154mb
num-of-cpus	4



Application Services

start.spring.io

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.tananh2706

Artifact discoveryserver

Name discoveryserver

Description Demo project for Spring Boot

Package name com.tananh2706.discoveryserver

Packaging Jar War

Java 18 17 11 8

Dependencies

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 Web WEB

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

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.

Spring Boot DevTools DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

GENERATE CTRL + ↵ EXPLORE CTRL + SPACE SHARE...



Application Services

ServiceApplication.java ×

```
1 package com.tanthanh.service;  
2  
3+import org.springframework.boot.SpringApplication;  
4  
5 @EnableDiscoveryClient  
6 @SpringBootApplication  
7 public class ServiceApplication {  
8  
9     public static void main(String[] args) {  
10         SpringApplication.run(ServiceApplication.class, args);  
11     }  
12 }  
13  
14 }
```

application.properties ×

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



Application Services

ServiceController.java ×

```
1 package com.tanthanh.service.controllers;  
2  
3+import org.springframework.beans.factory.annotation.Value;□  
6  
7 @RestController  
8 public class ServiceController {  
9  
10    @Value("${service.instance.name}")  
11    private String instance;  
12  
13    @RequestMapping("/")  
14    public String sayHello() {  
15        return "hello from "+instance;  
16    }  
17}  
18
```



Application Services

Tai Lieu Download - service/src/main/java/com/tanhanh/service

File Edit Source Refactor Navigate Search

Package Explorer X

- > client
- > discoverserver [boot] [devtools]
- ✓ service [boot] [devtools]
 - ✓ src/main/java
 - ✓ com.tanhanh.service
 - > ServiceApplication.java
 - ✓ com.tanhanh.service.controllers
 - > ServiceController.java
 - ✓ src/main/resources
 - static
 - templates
 - application.properties
 - > src/test/java
 - > JRE System Library [JavaSE-11]
 - > Maven Dependencies
 - target/generated-sources/annotations
 - target/generated-test-sources/test-an
 - > src
 - > target
 - HELP.md

Boot Dashboard X

Type tags, projects, or working set names to

> local

Run Configurations

Create, manage, and run configurations

Errors present in application properties.

Name: instance1

Spring Boot Arguments JRE Classpath Source Environment Common

Project service

Main type com.tanhanh.service.ServiceApplication

Profile

Enable debug output Hide from Boot Dash

Fast startup ANSI console output

Enable JMX Port: 0

Enable Life Cycle Management. Termination timeout (ms): 15000

Override properties:

```
server.port=9001
service.instance.name=service1
```

Revert Apply Run Close

Filter matched 26 of 26 items

ng-tool-suite-4-4.14.1.RELEASE-e4.23.0-win32:

uration.java:297) ~[spring-cloud]



Type here to search



34°C Mưa nhỏ 4:11 PM
ENG 5/24/2022

Capgemini 2021. All rights reserved



Application Services



Toggle navigation

System Status

Environment	test	Current time	2022-05-24T16:14:49 +0700
Data center	default	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;...
14
15 @RestController
16 public class ClientController {
17
18
19@Autowired
20 private EurekaClient client;
21
22@Autowired
23 private RestTemplateBuilder templatebuilder;
24
25
26@RequestMapping("/")
27 public String callService() {
28     InstanceInfo instance = client.getNextServerFromEureka("service", false);
29     String url = instance.getHomePageUrl();
30
31     RestTemplate restTemplate = templatebuilder.build();
32     ResponseEntity<String> response = restTemplate.exchange(url, HttpMethod.GET, null, String.class);
33     return response.getBody();
34 }
35 }
36 }
```



Application Client

```
ClientController.java ×
import org.springframework.beans.factory.annotation.Autowired;...
14
15 @RestController
16 public class ClientController {
17
18
19@Autowired
20 private EurekaClient client;
21
22@Autowired
23 private RestTemplateBuilder templatebuilder;
24
25
26@RequestMapping("/")
27 public String callService() {
28     InstanceInfo instance = client.getNextServerFromEureka("service", false);
29     String url = instance.getHomePageUrl();
30
31     RestTemplate restTemplate = templatebuilder.build();
32     ResponseEntity<String> response = restTemplate.exchange(url, HttpMethod.GET, null, String.class);
33     return response.getBody();
34 }
35 }
36 }
```



Application Client

Tai Lieu Download - client/src/main/java/com/tananh/client/Controllers/ClientController.java - Spring Tool Suite 4

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

client [boot] [devtools]
discoverserver [boot] [devtools]
service [boot] [devtools]

ClientController.java x ClientApplication.java

```
14  
15 @RestController  
16 public class ClientController {  
17  
18    @Autowired  
19    private EurekaClient client;  
20  
21    @Autowired  
22    private RestTemplateBuilder templatebuilder;  
23  
24  
25    @RequestMapping("/")  
26    public String callService() {  
27        InstanceInfo instance = client.getNextServerFromEureka("service", false);  
28        String url = instance.getHomePageUrl();  
29  
30        RestTem  
31        Respons  
32        return  
33    }  
34 }  
35 }  
36 }
```

RestTemplateBuilder url= "http://LVN01000155.corp.capgemini.com:9002/" (id=134)
coder= 0
hash= 0
hashIsZero= false
value= (id=141)

http://LVN01000155.corp.capgemini.com:9002/

Boot Dashboard x

Type tags, projects, or working set names to match (incl. * and ? wildcards)

local

Problems @ Javadoc

- Daemon Thread
- Daemon Thread
- Daemon Thread
- Daemon Thread [http-nio-8080-exec-2] (Running)
- Daemon Thread [http-nio-8080-exec-3] (Running)
- Daemon Thread [http-nio-8080-exec-4] (Suspended)
 - owns: NioEndpoint\$NioSocketWrapper (id=126)
 - ClientController.callService() line: 31

Writable Smart Insert 31 : 1 : 953



Type here to search



34°C Mưa nhỏ ENG 4:30 PM
5/24/2022

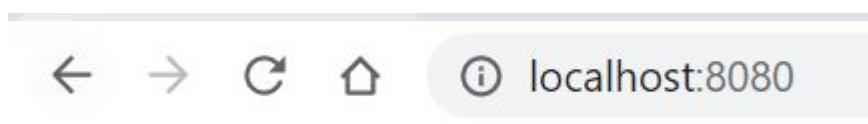
gemini 2021. All rights reserved



Application Client



hello from service1

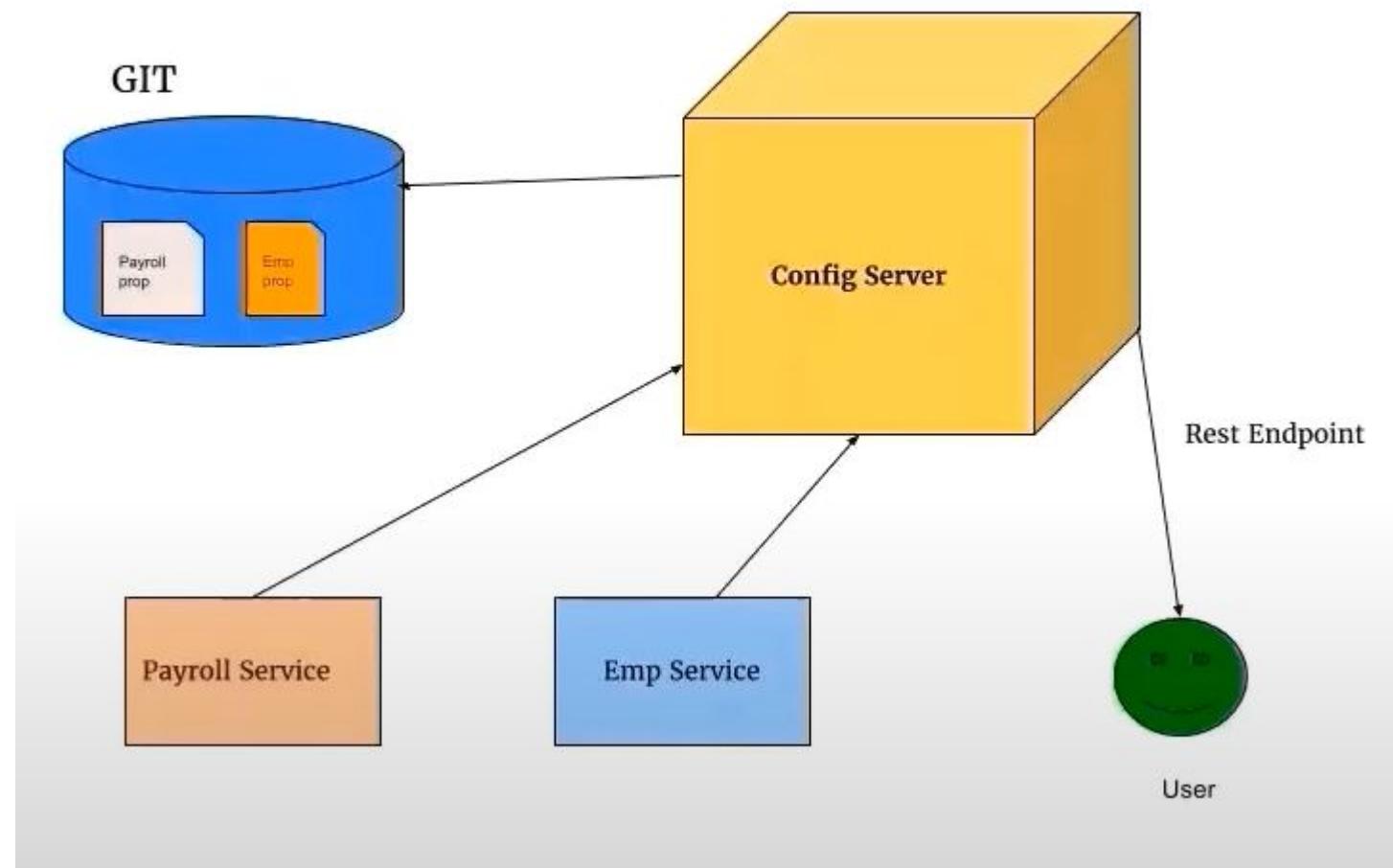


hello from service2



Distributed Configuration

Spring Cloud Config Server





Spring cloud config

start.spring.io

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.tanhanh2706

Artifact configserver

Name configserver

Description Demo project for Spring Boot

Package name com.tanhanh2706.configserver

Packaging Jar War

Java 18 17 11 8

Dependencies

[ADD DEPENDENCIES... CTRL + B](#)

Config Server SPRING CLOUD CONFIG
Central management for configuration via Git, SVN, or HashiCorp Vault.

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

Search or jump to... / Pull requests Issues Marketplace Explore

bingo2706 / config-server Public Pin Unwatch 1 Fork 0 Star 0

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 0 tags Go to file Add file Code About

No description, website, or topics provided.

bingo2706 first commit 357f5ed 1 hour ago 1 commit

application.properties first commit 1 hour ago

config-client-prod.properties first commit 1 hour ago

config-client.properties first commit 1 hour ago

Help people interested in this repository understand your project by adding a README. Add a README

0 stars 1 watching 0 forks

Releases No releases published Create a new release

Packages

No packages published
Publish your first package



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 ×

```
1 package com.tanthanh2706.configserver;
2
3+import org.springframework.boot.SpringApplication;□
4
5 @EnableDiscoveryClient
6 @EnableConfigServer
7 @SpringBootApplication
8 public class ConfigserverApplication {
9
10     public static void main(String[] args) {
11         SpringApplication.run(ConfigserverApplication.class, args);
12     }
13 }
```

application.properties ×

```
1 spring.application.name=configserver
2 server.port=8888
3 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
4 spring.cloud.config.server.git.uri=https://github.com/bingo2706/config-server.git
5 spring.cloud.config.server.git.default-label=master
6
7
```



Spring cloud config



Endpoint

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



Example

- /myapp/dev/master



Spring cloud config

← → C ⌂ ① localhost:8888/abcd/default

```
{  
  name: "abcd",  
  profiles: [  
    "default"  
  ],  
  label: null,  
  version: "357f5edc2b8d62c010dc159e43e38fd0edfd92a3",  
  state: null,  
  propertySources: [  
    {  
      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

localhost:8888/config-client/default

```
{  
  name: "config-client",  
  profiles: [  
    "default"  
  ],  
  label: null,  
  version: "5398bc5fe5e50a46a82d332211d5a6be24641552",  
  state: null,  
  propertySources: [  
    {  
      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

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



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 WarJava 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 ×

```
1 package com.tanthanh.configclient;
2
3+import org.springframework.boot.SpringApplication;
4
5
6 @EnableDiscoveryClient
7 @SpringBootApplication
8 public class ConfigclientApplication {
9
10    public static void main(String[] args) {
11        SpringApplication.run(ConfigclientApplication.class, args);
12    }
13
14
15 }
```

bootstrap.properties ×

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



Spring cloud config

```
ClientConfig.java x
1 package com.tanthanh.configclient;
2
3 import org.springframework.boot.context.properties.ConfigurationProperties;
4
5
6 @Component
7 @ConfigurationProperties(prefix = "sample")
8 public class ClientConfig {
9     private String property1;
10
11     public String getProperty1() {
12         return property1;
13     }
14
15     public void setProperty1(String property1) {
16         this.property1 = property1;
17     }
18 }
19
```



Spring cloud config

```
ConfigClientController.java ×  
3 import org.springframework.beans.factory.annotation.Autowired;  
4 import org.springframework.beans.factory.annotation.Value;  
5 import org.springframework.web.bind.annotation.GetMapping;  
6 import org.springframework.web.bind.annotation.RequestMapping;  
7 import org.springframework.web.bind.annotation.RestController;  
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 }
```

← → ⌂ ⌂ localhost:8080/config

sample property1 config-client1 prod1 sample property2 config-client2 prod2



Refresh Config

bootstrap.properties ×

```
1spring.application.name=config-client
2eureka.client.service-url.defaultZone=http://localhost:8761/eureka
3spring.cloud.config.discovery.enabled=true
4spring.profiles.active=prod
5
6management.endpoints.web.exposure.include=*
```

2 lines (2 sloc) | 128 Bytes

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



Refresh Config

http://localhost:8080/actuator/refresh



POST

http://localhost:8080/actuator/refresh

Send

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings Cookies

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (5) Test Results

🌐 200 OK 6.54 s 255 B Save Response ▾

Pretty Raw Preview Visualize

JSON ▾



```
1 [ "sample.property1",  
2   "config.client.version",  
3   "sample.property2"  
4 ]
```



localhost:8080/config

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



Refresh Config

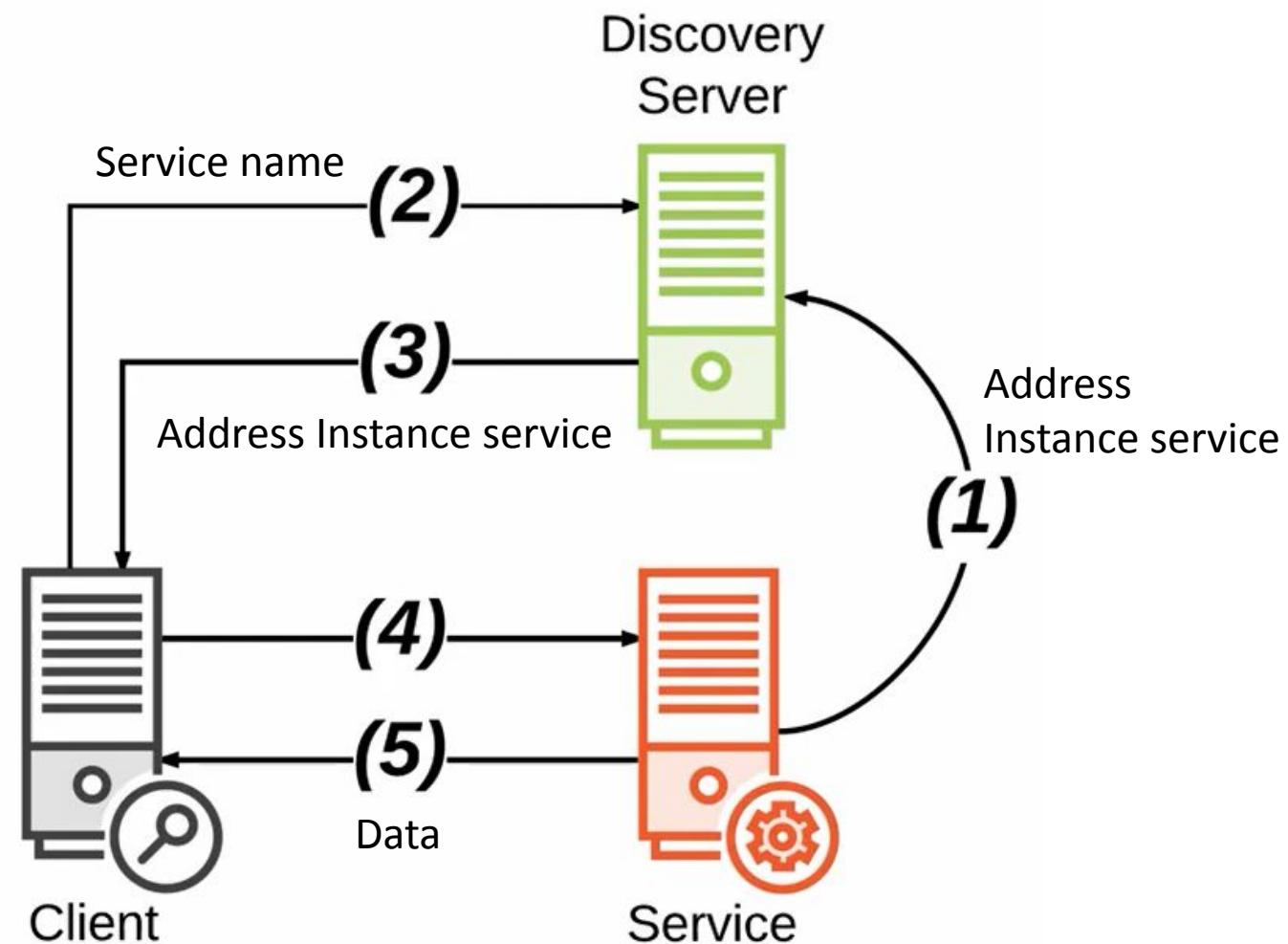
```
@RestController  
@RequestScope  
public class ConfigClientController {  
  
    @Autowired  
    ClientConfig clientConfig;  
  
    @Value("${sample.property2}")  
    private String property2;  
  
    @GetMapping("/config")  
    public String printConfig() {  
        return clientConfig.getProperty1() + " " + property2;  
    }  
}
```

← → ⌂ ⌂ ⓘ localhost:8080/config

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

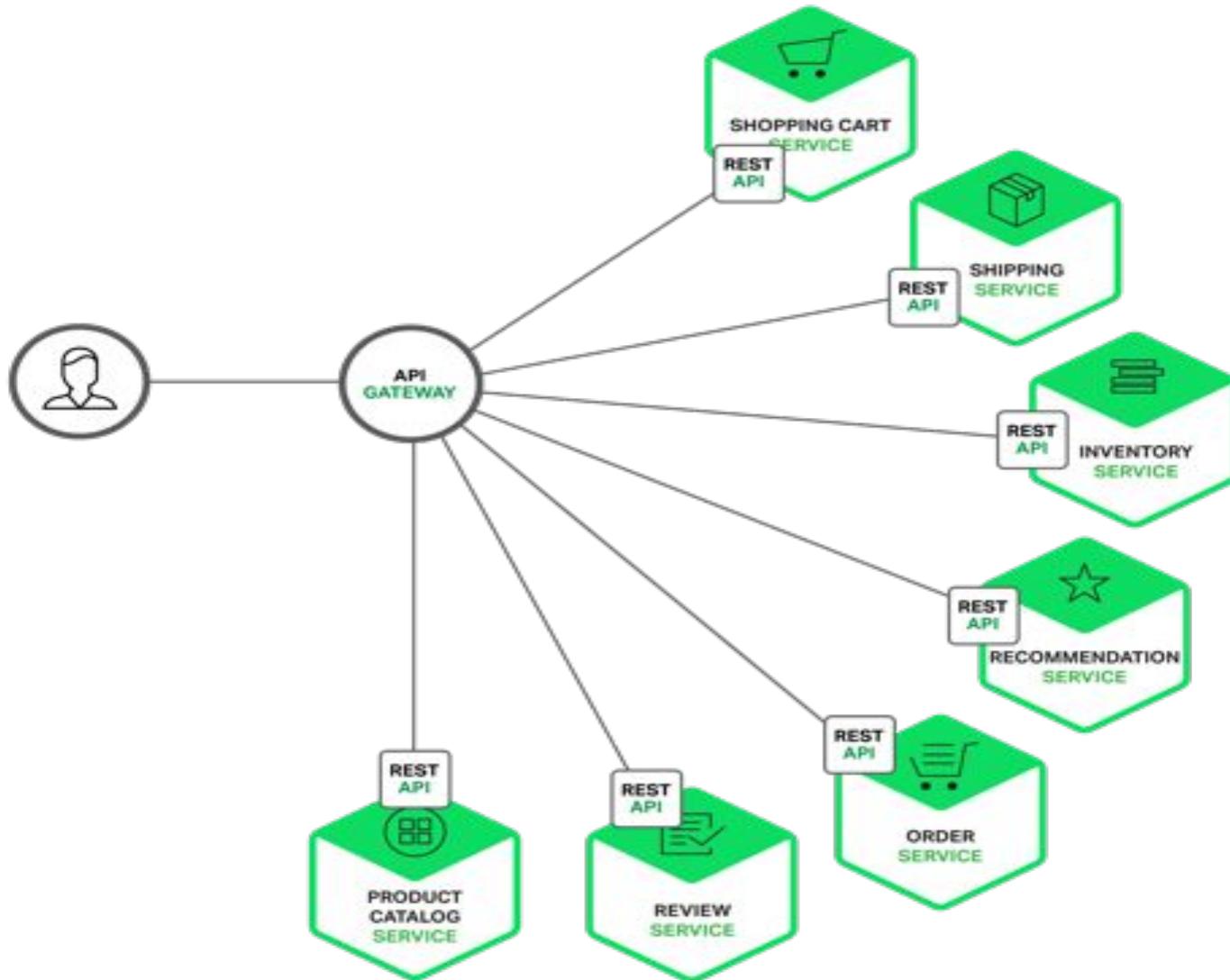


Intelligent routing





Intelligent routing





Intelligent routing

 **spring initializr**

Project Maven Project Gradle Project **Language** Java Kotlin Groovy

Spring Boot
 2.4.0 (SNAPSHOT) 2.4.0 (M4) 2.3.5 (SNAPSHOT) 2.3.4
 2.2.11 (SNAPSHOT) 2.2.10 2.1.18 (SNAPSHOT) 2.1.17

Project Metadata

Group	com.test
Artifact	service1
Name	service1
Description	Demo project for Spring Boot
Package name	com.test.service1
Packaging	<input checked="" type="radio"/> Jar <input type="radio"/> War
Java	<input checked="" type="radio"/> 15 <input type="radio"/> 11 <input type="radio"/> 8

Dependencies [ADD DEPENDENCIES... CTRL + B](#)

Zuul [Maintenance] SPRING CLOUD ROUTING
Intelligent and programmable routing with Spring Cloud Netflix Zuul. 

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. 

Group II © Capgemini 2021. All rights reserved



Intelligent routing

GatewayserviceApplication.java

```
1 package com.test.gatewayservice;
2
3+import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 @EnableZuulProxy
8 @EnableDiscoveryClient
9
10 public class GatewayserviceApplication {
11
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

localhost:8080/service1

← → ⌂ ⌂ i Apps IFI_MDM_Work Home RemoteWork

Hello from service 1



localhost:8080/service2

← → ⌂ ⌂ i Apps IFI_MDM_Work Home RemoteWork

Hello from service 2





Intelligent routing

```
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

A screenshot of a web browser window. The address bar shows the URL `localhost:8080/service1`. Below the address bar is a navigation bar with icons for back, forward, refresh, and home. The main content area displays the text: `Hello from service 1. Request ID: 671f5ef6-14cc-40d9-b8d3-e51ae15f7878`. At the bottom of the browser window, there is a horizontal menu bar with several items: Apps, IFI_MDM_Work, Home, RemoteWork, Spring, and GET_VD.

Hello from service 1. Request ID: 671f5ef6-14cc-40d9-b8d3-e51ae15f7878



Client-side Load Balancing



Project

Maven Project Gradle Project

Language

Java Kotlin Groovy

Spring Boot

2.4.0 (SNAPSHOT) 2.4.0 (M4) 2.3.5 (SNAPSHOT) 2.3.4
 2.2.11 (SNAPSHOT) 2.2.10 2.1.18 (SNAPSHOT) 2.1.17

Project Metadata

Group com.test

Artifact demoapp

Name demoapp

Description Demo project for Spring Boot

Package name com.test.demoapp

Packaging Jar War

Java 15 11 8

Dependencies

[ADD DEPENDENCIES... CTRL + B](#)

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 Web WEB

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

Ribbon [Maintenance] SPRING CLOUD ROUTING

Client-side load-balancing with Spring Cloud Netflix and Ribbon.



Client-side Load Balancing

```
@Configuration
public class DemoConfiguration {

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

@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 {

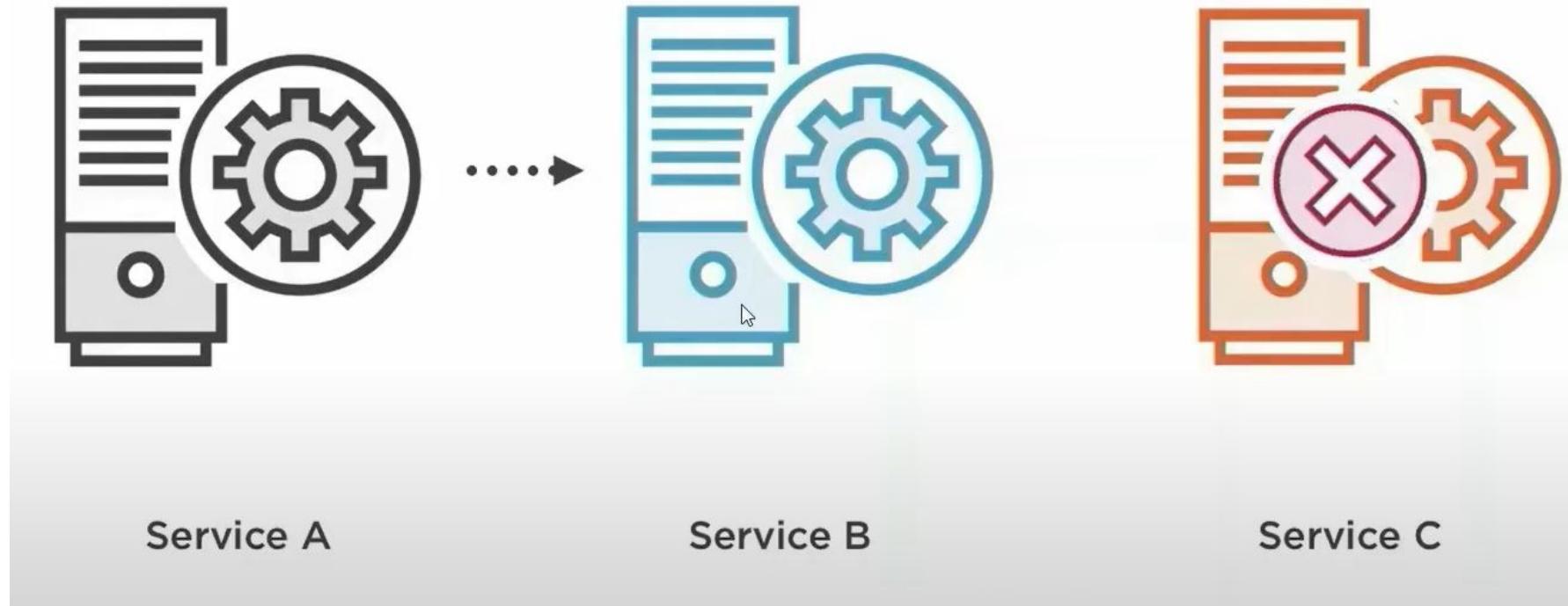
    @Bean
    @LoadBalanced
    public RestTemplate getRestTemplate()
    {
        return new RestTemplate();
    }

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



Circuit Breaker

- Cascading failure





Circuit Breaker



Project
 Maven Project Gradle Project

Language
 Java Kotlin Groovy

Spring Boot
 2.4.0 (SNAPSHOT) 2.4.0 (RC1) 2.3.6 (SNAPSHOT) 2.3.5
 2.2.12 (SNAPSHOT) 2.2.11

Project Metadata

Group com.test
Artifact demoapp
Name demoapp
Description Demo project for Spring Boot
Package name com.test.demoapp
Packaging Jar War
Java 15 11 8

Dependencies

[ADD DEPENDENCIES...](#) CTRL + B

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 Web WEB

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

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.

Hystrix [Maintenance] SPRING CLOUD CIRCUIT BREAKER

Circuit breaker with Spring Cloud Netflix Hystrix.



Circuit Breaker

```
1  
@EnableDiscoveryClient  
@EnableCircuitBreaker  
@SpringBootApplication  
public class DemoappApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(DemoappApplication.class, args);  
    }  
  
}  
  
@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 ràng.



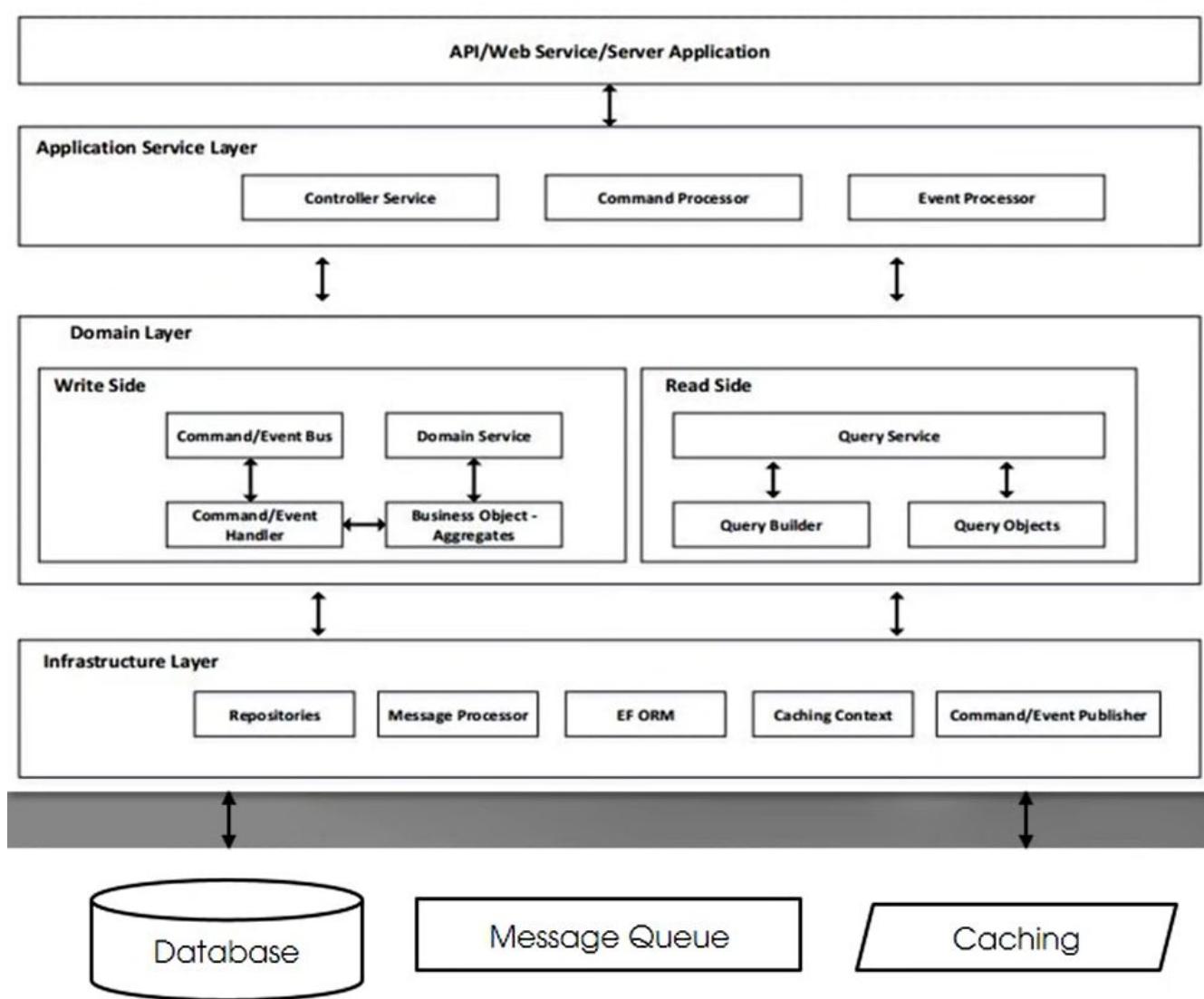
DDD Layered Architecture

* 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

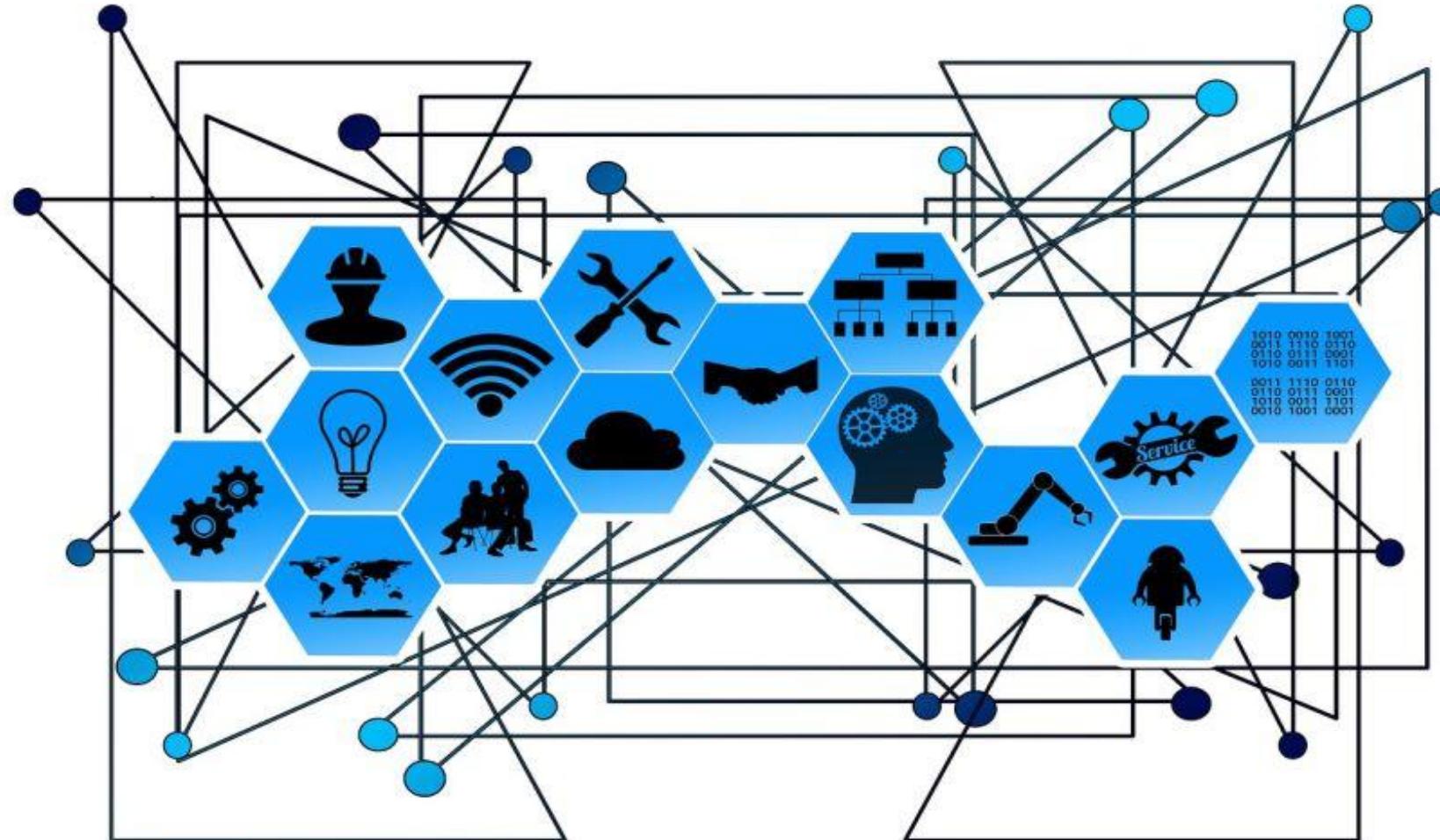


DDD Layered Architecture





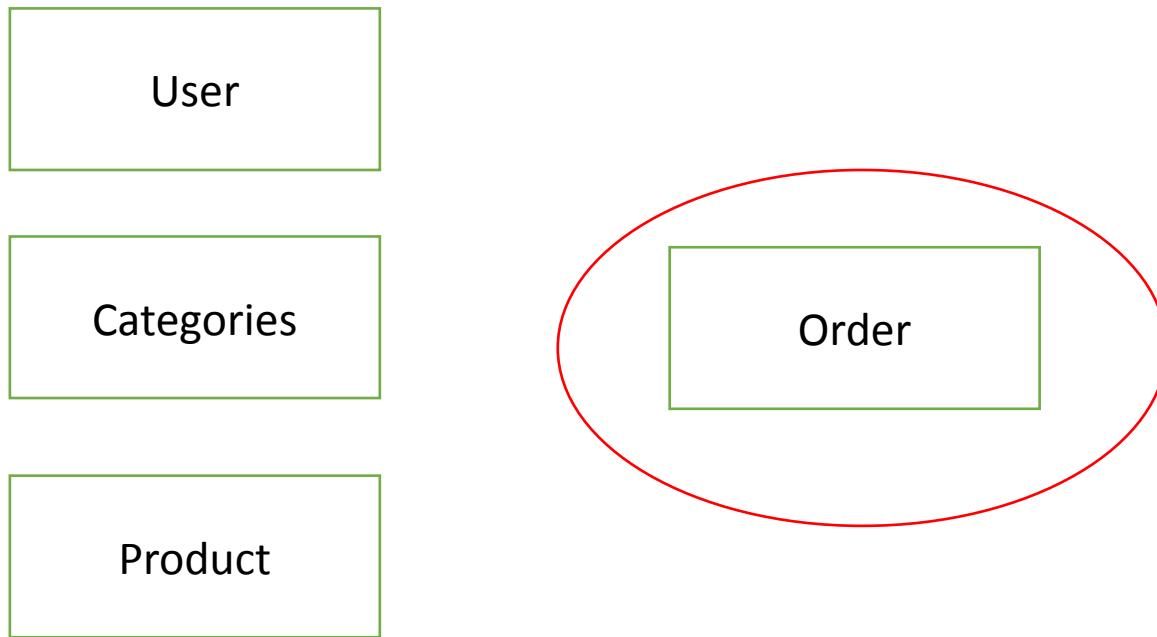
Domain model





Domain model

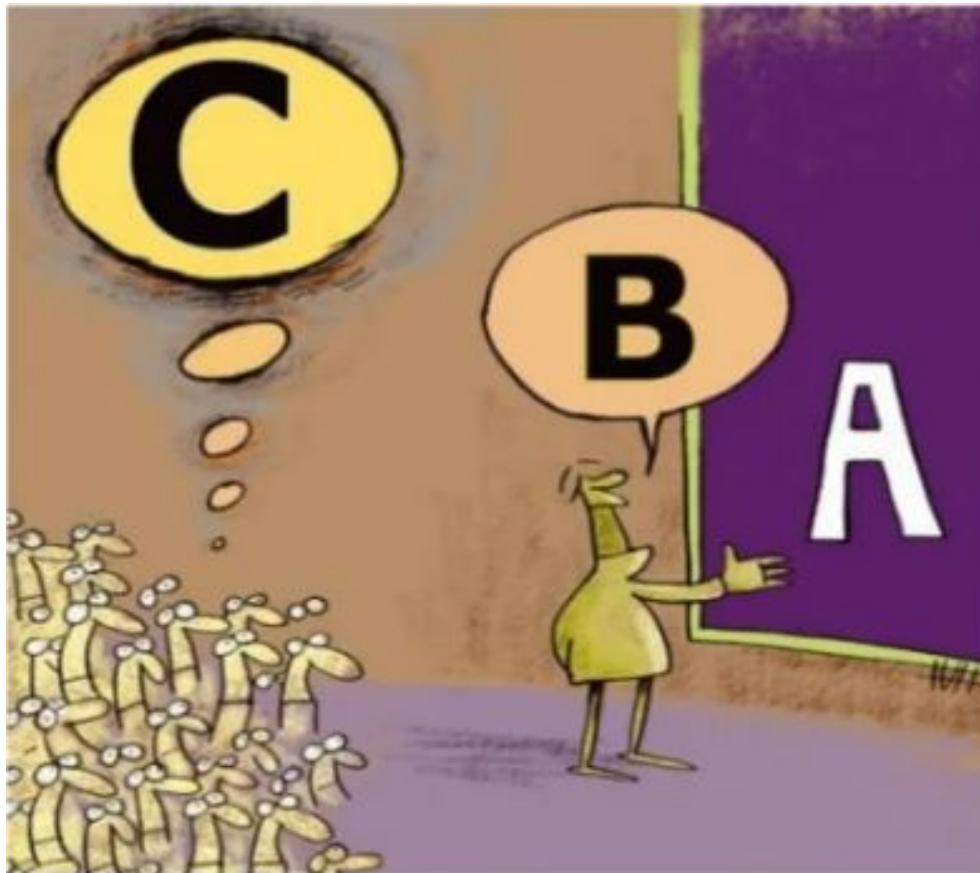
- Phân tách nghiệp vụ thành các nghiệp vụ con và đào sâu và nó
- Tập trung vào từng sub domain





Domain model

Ngôn ngữ chung (Ubiquitous Language)





Domain model

Ngôn ngữ chung (Ubiquitous Language)

Client = Customer, consumer, Khách hàng

Voucher = coupon, gift card

Delete the booking



Submit the order



Update the job order



Create the invoice



Set state of the game



Cancel the booking

Checkout

Extend the job order

Register/Accept the invoice

Start/Pause the game



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)...

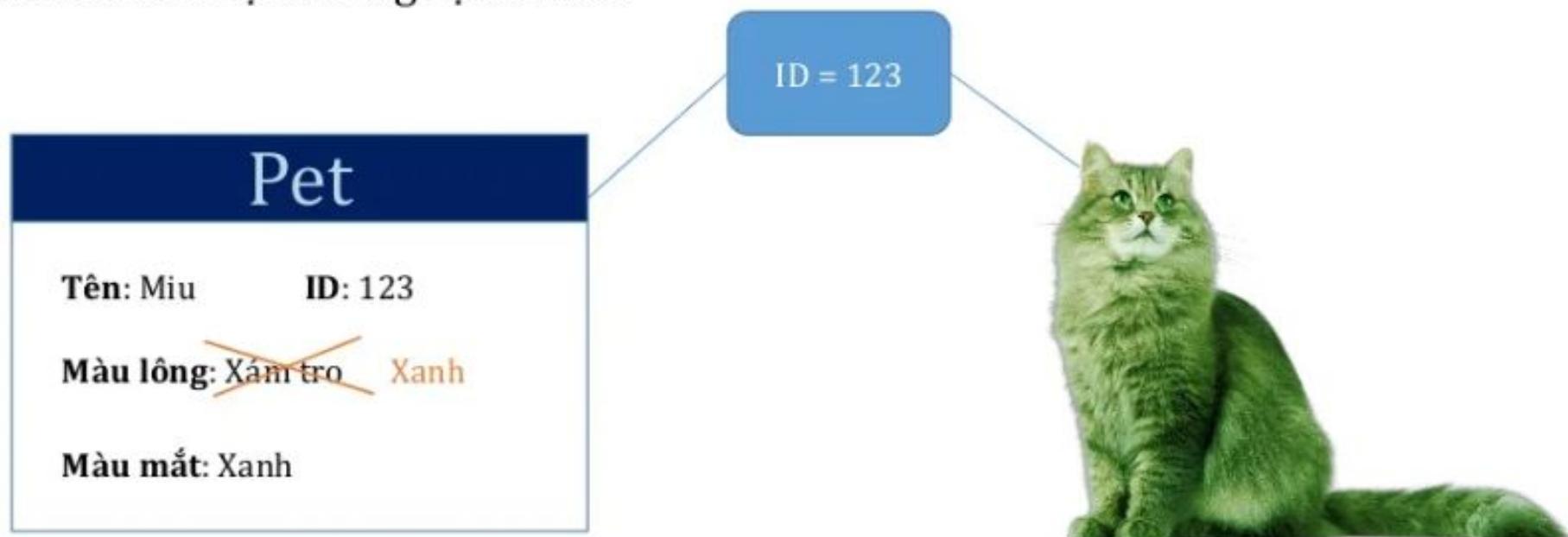
    5 references | 0/3 passing
    public void Confirm(DateTime dateConfirmed)...
}
```



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

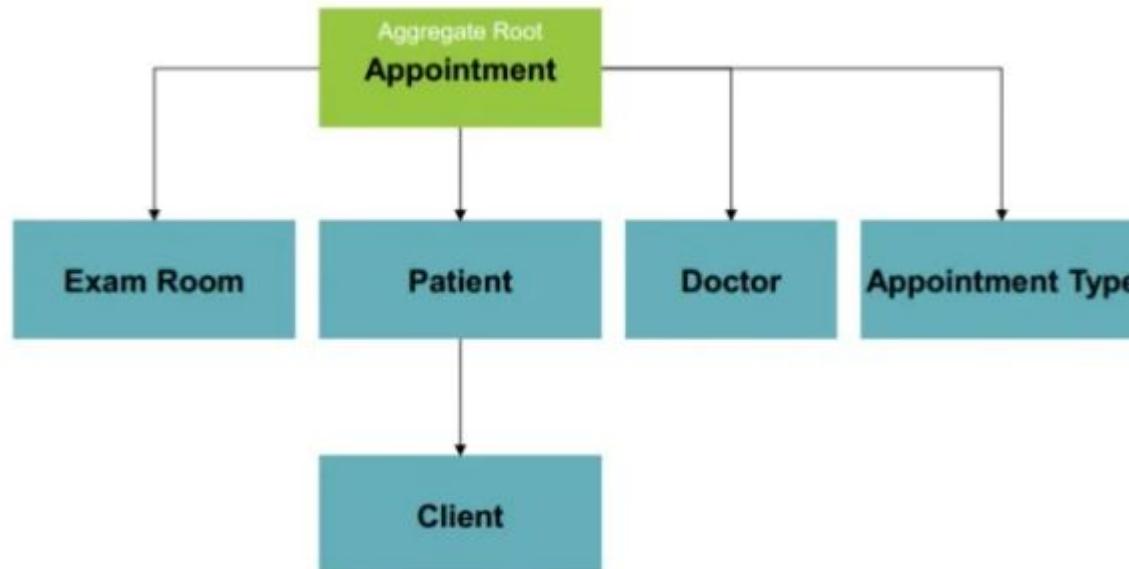




Domain model

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

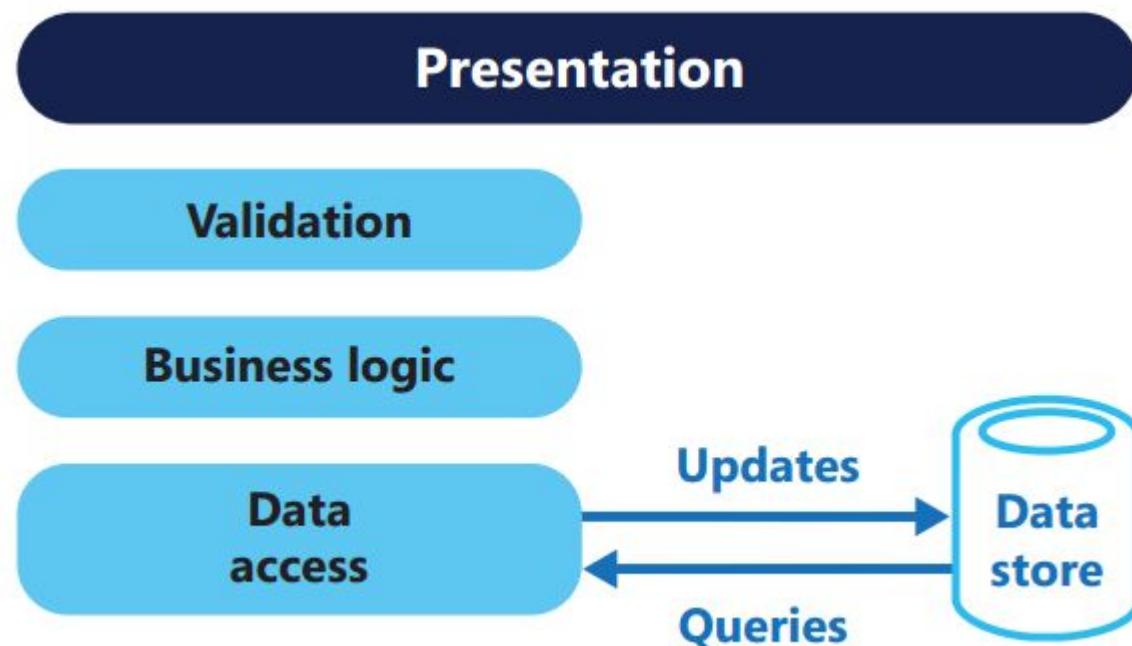
- **Repository:** Repository đảm nhiệm việc lấy ra những đối tượng đã được lưu trữ.





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



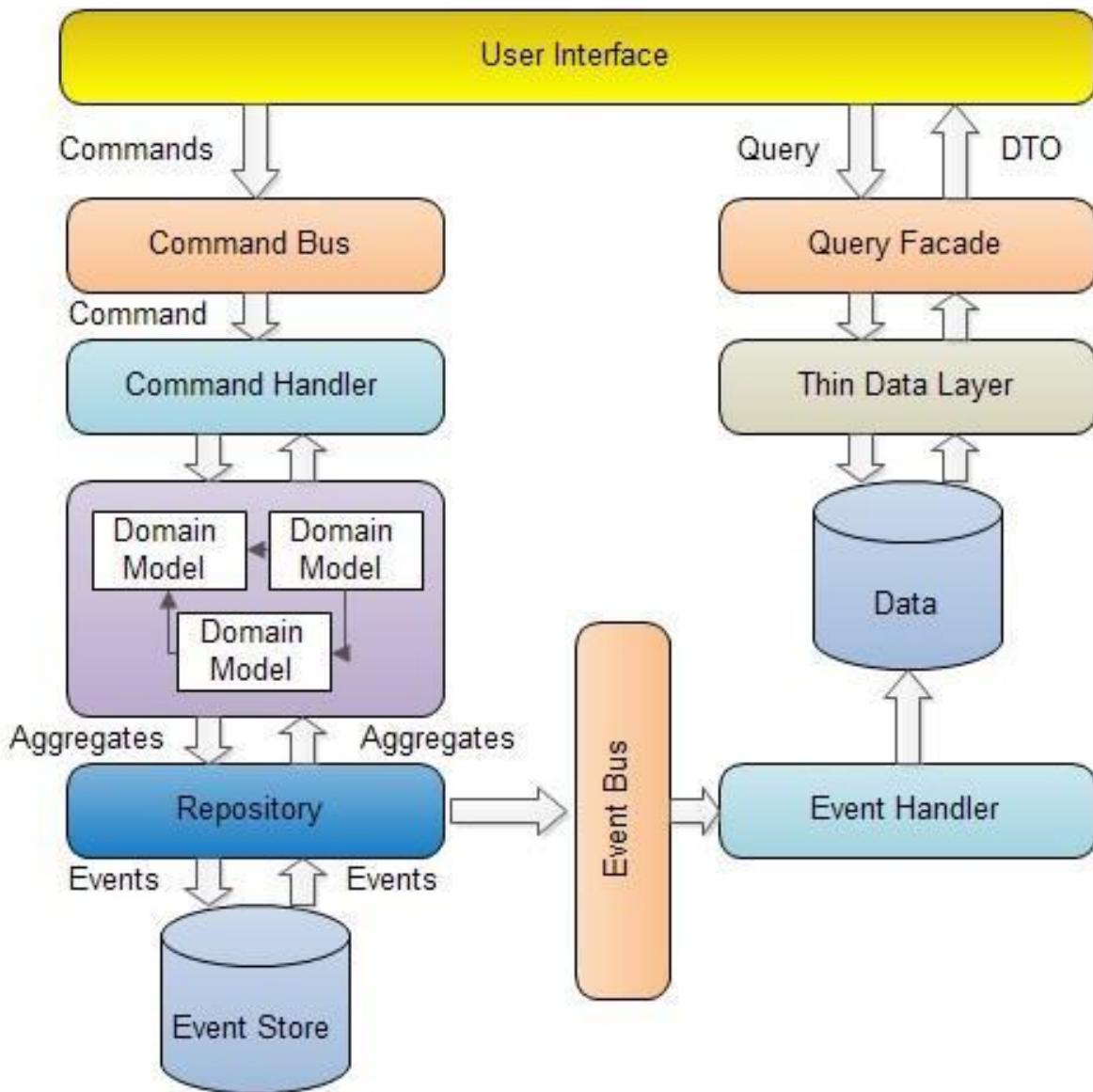
CQRS

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

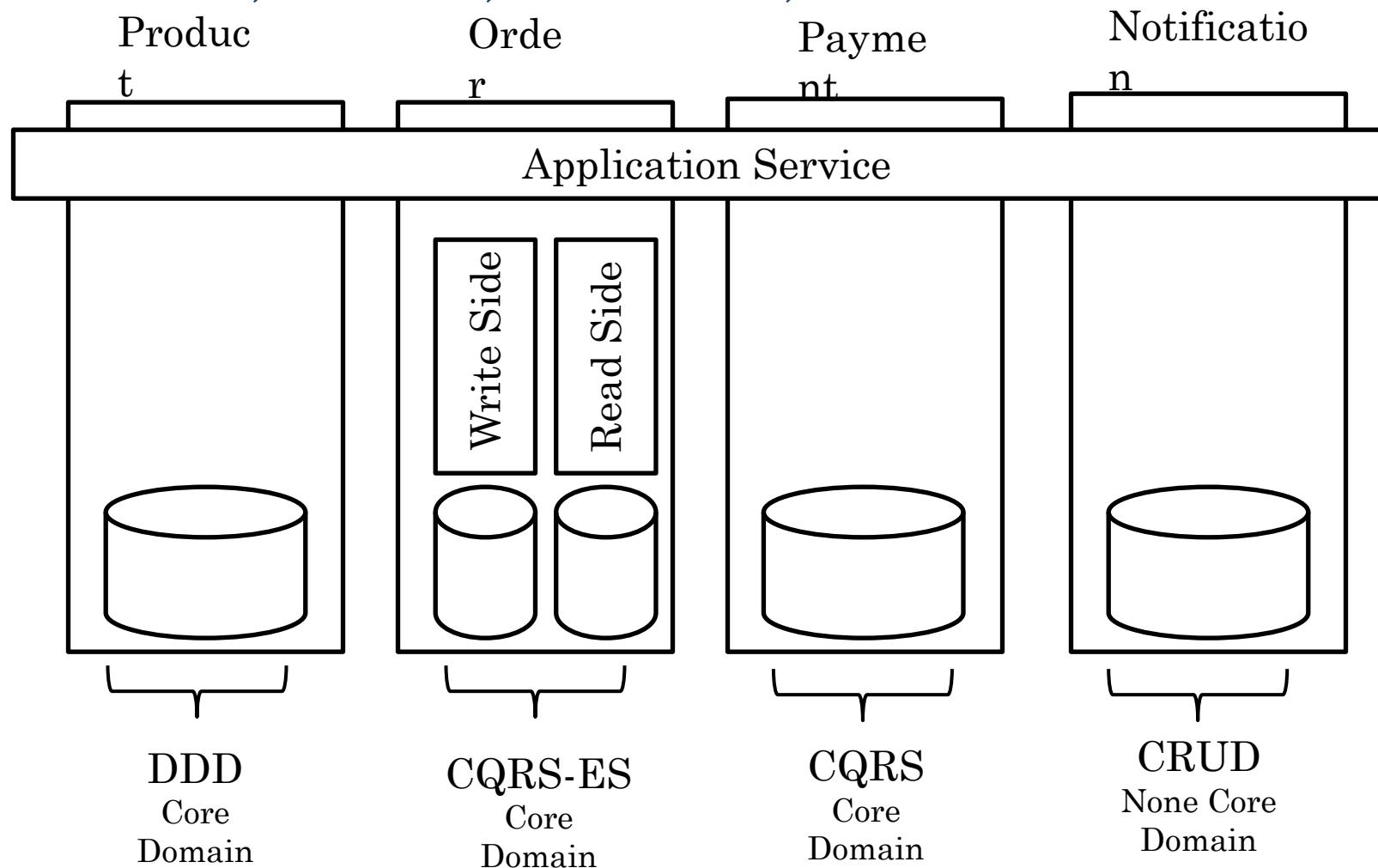
XÉT VÍ DỤ CỤ THỂ

- 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



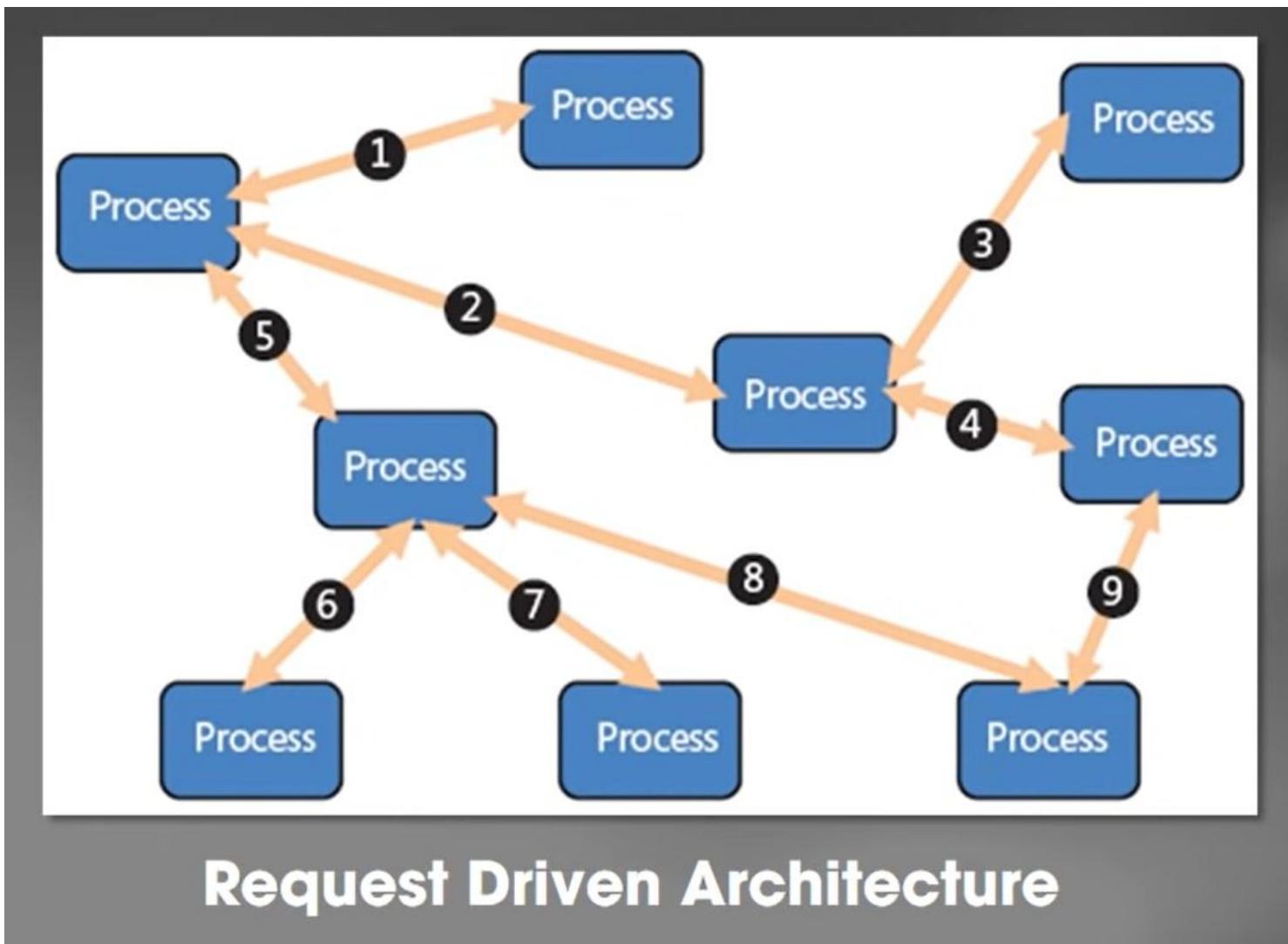
CQRS

PRODUCT, ORDER, PAYMENT, NOTIFICATION



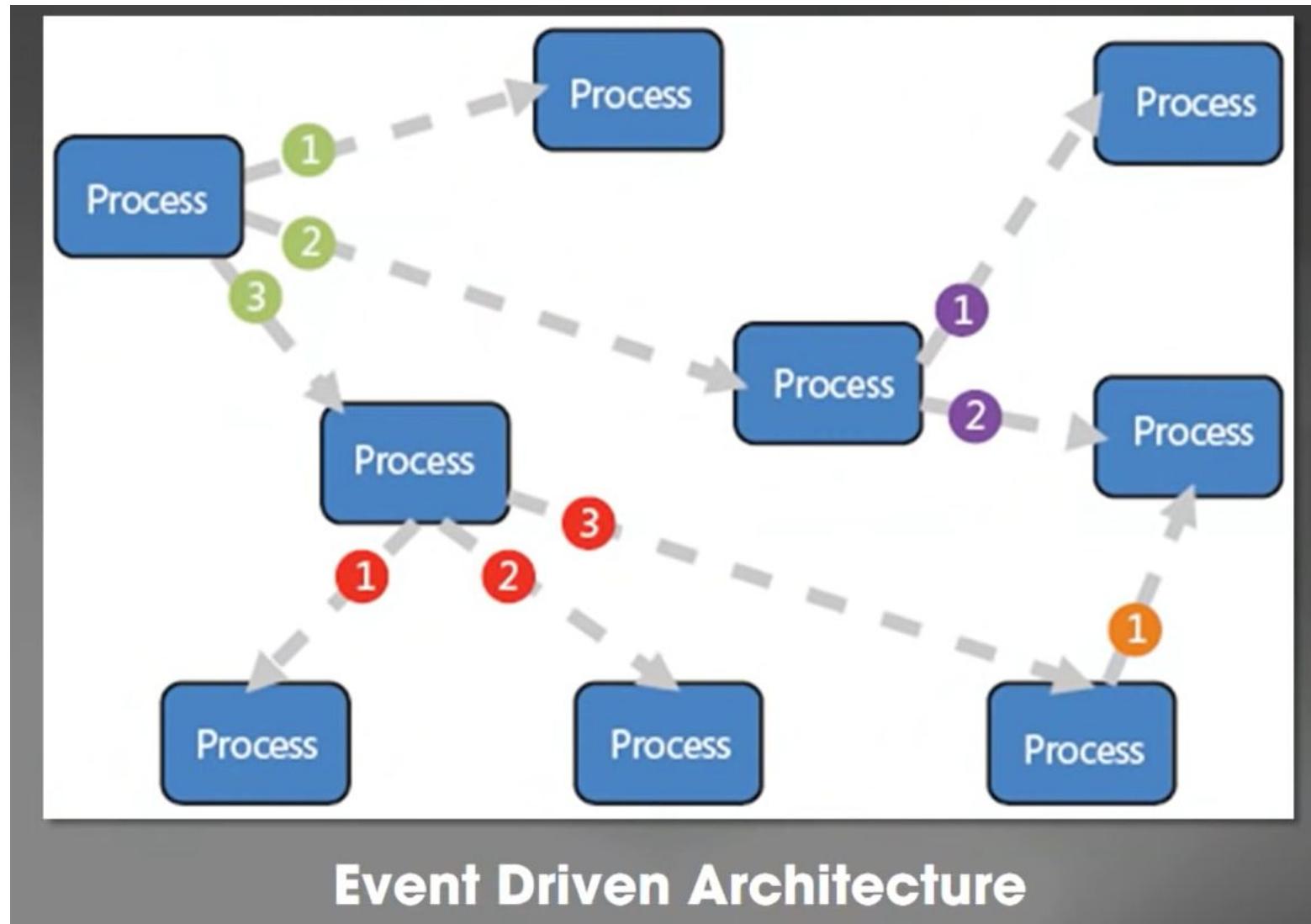


Event Driven Architecture



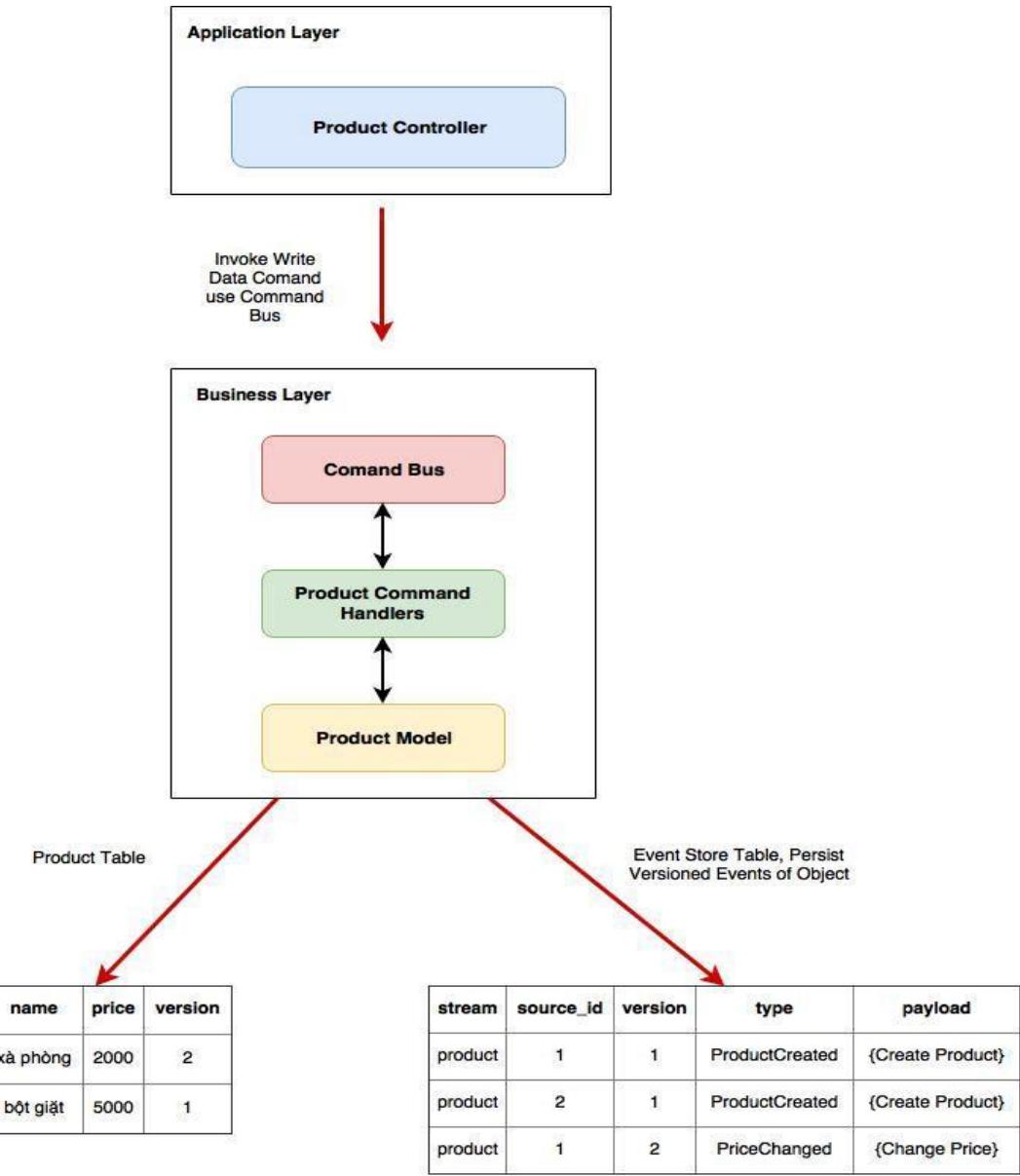


Event Driven Architectur





Event Sourcing

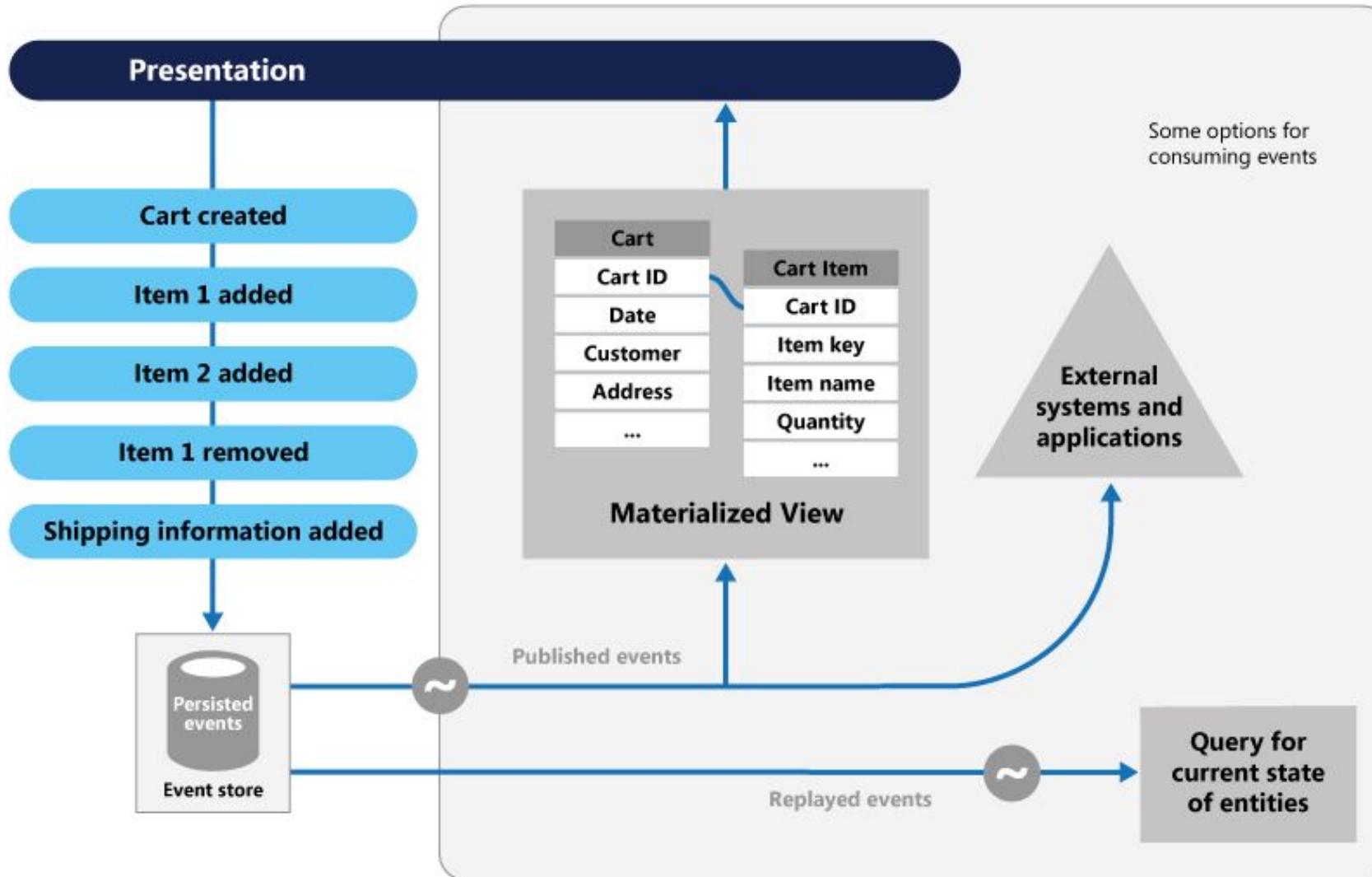


Mô hình Event Sourcing—ES, là mô hình thiết kế mà trạng thái của object sẽ được lưu trữ dưới dạng chuỗi các sự kiện thay đổi

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

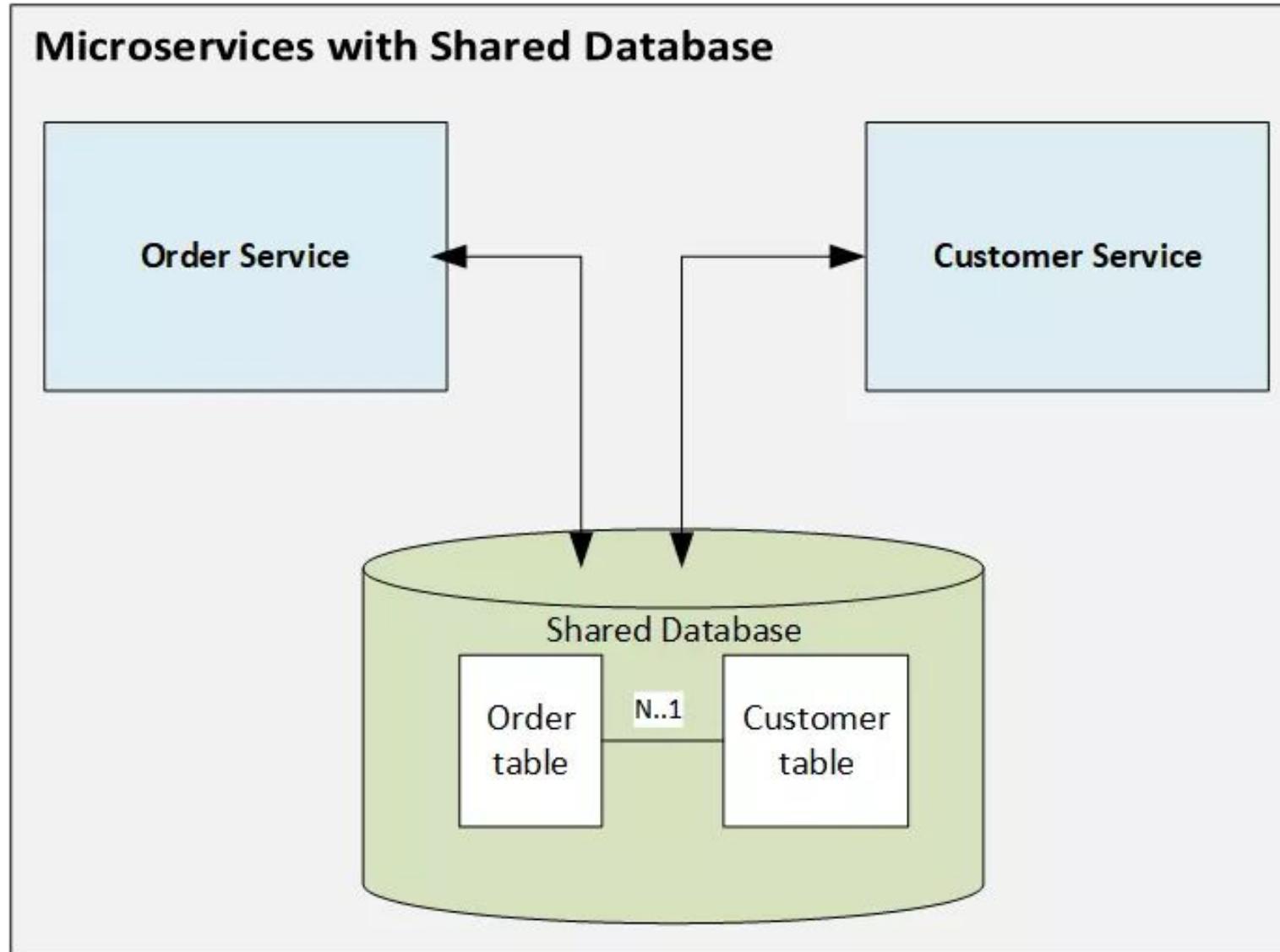


Event Sourcing





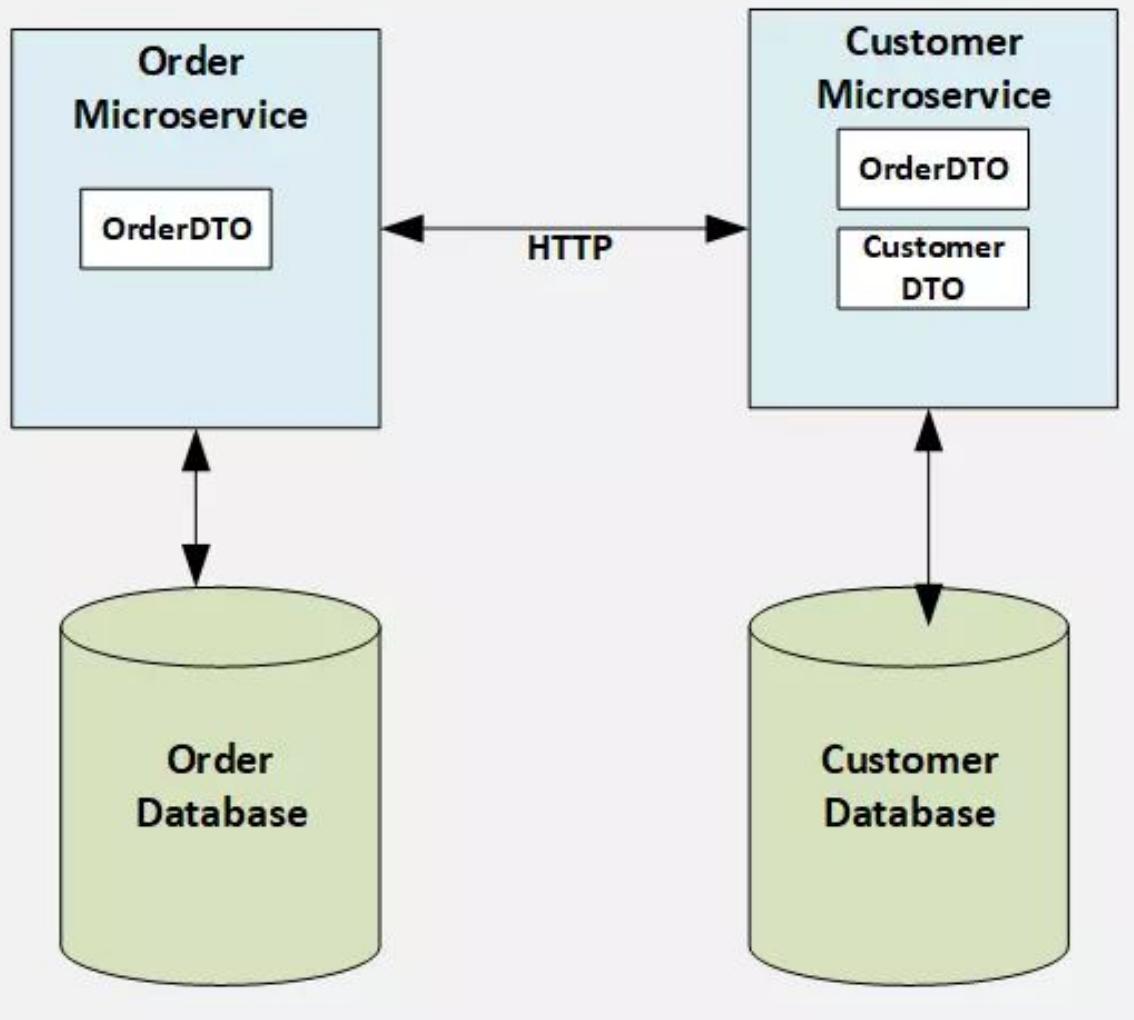
Event Sourcing





Event Sourcing

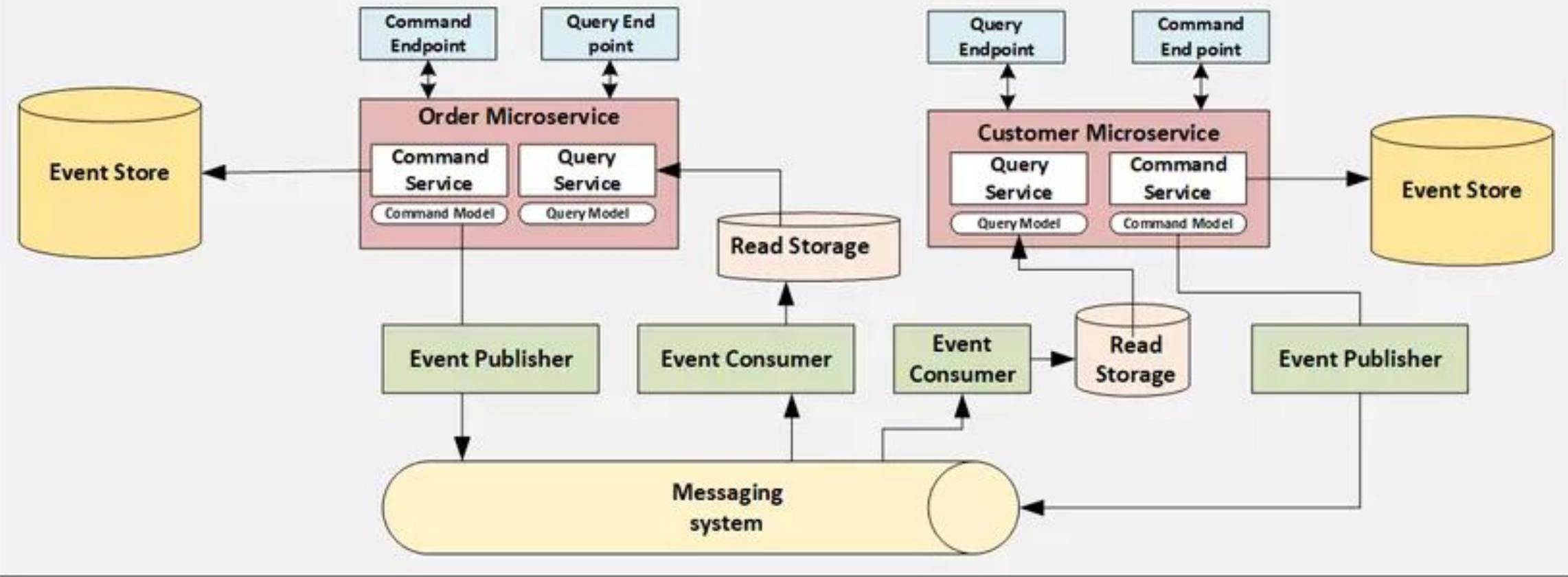
Microservices with dedicated database





Event Sourcing

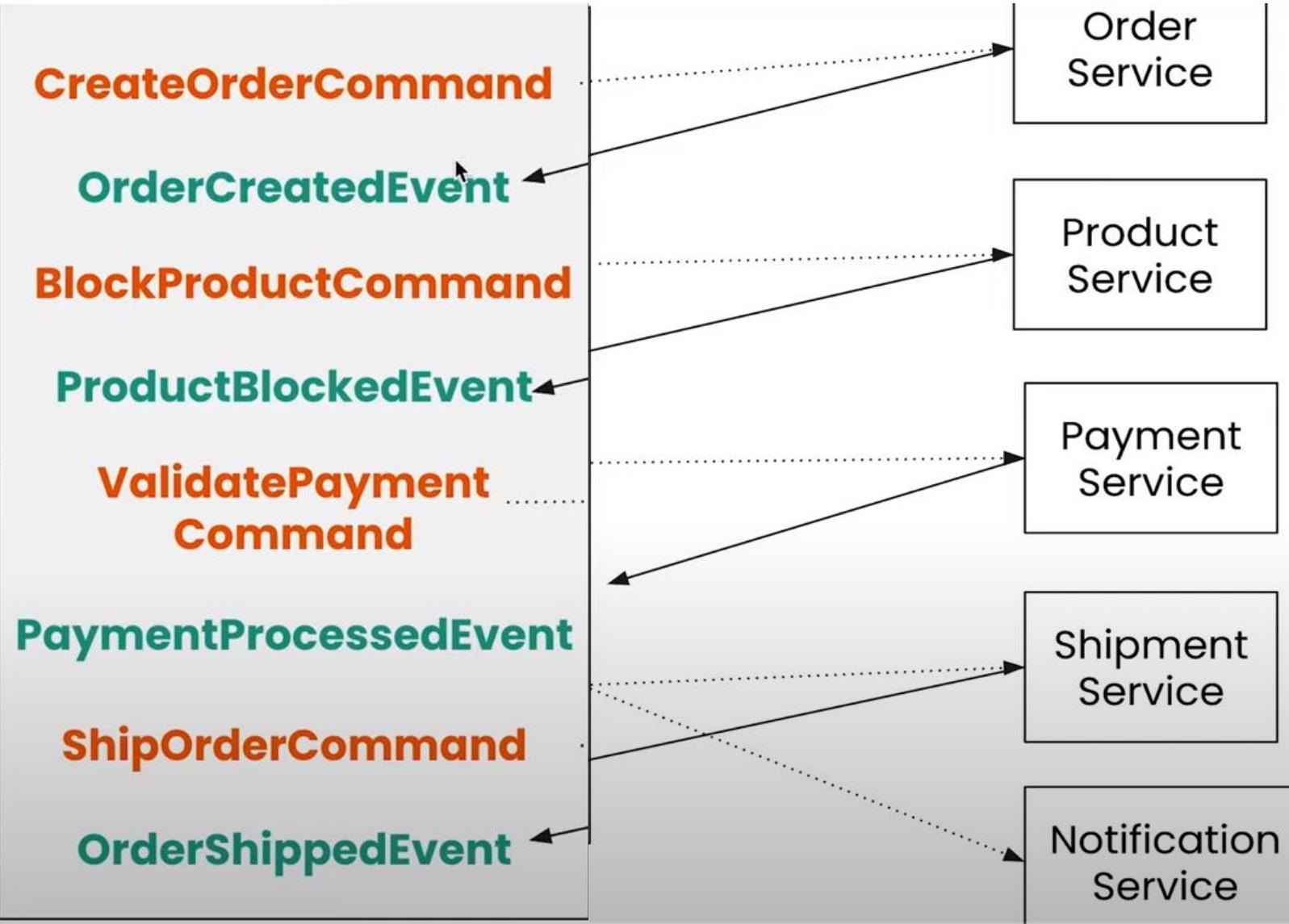
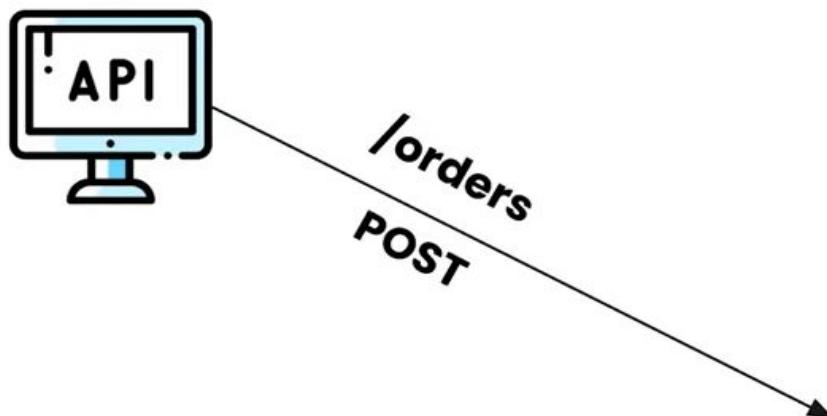
Microservices with CQRS and Event Sourcing





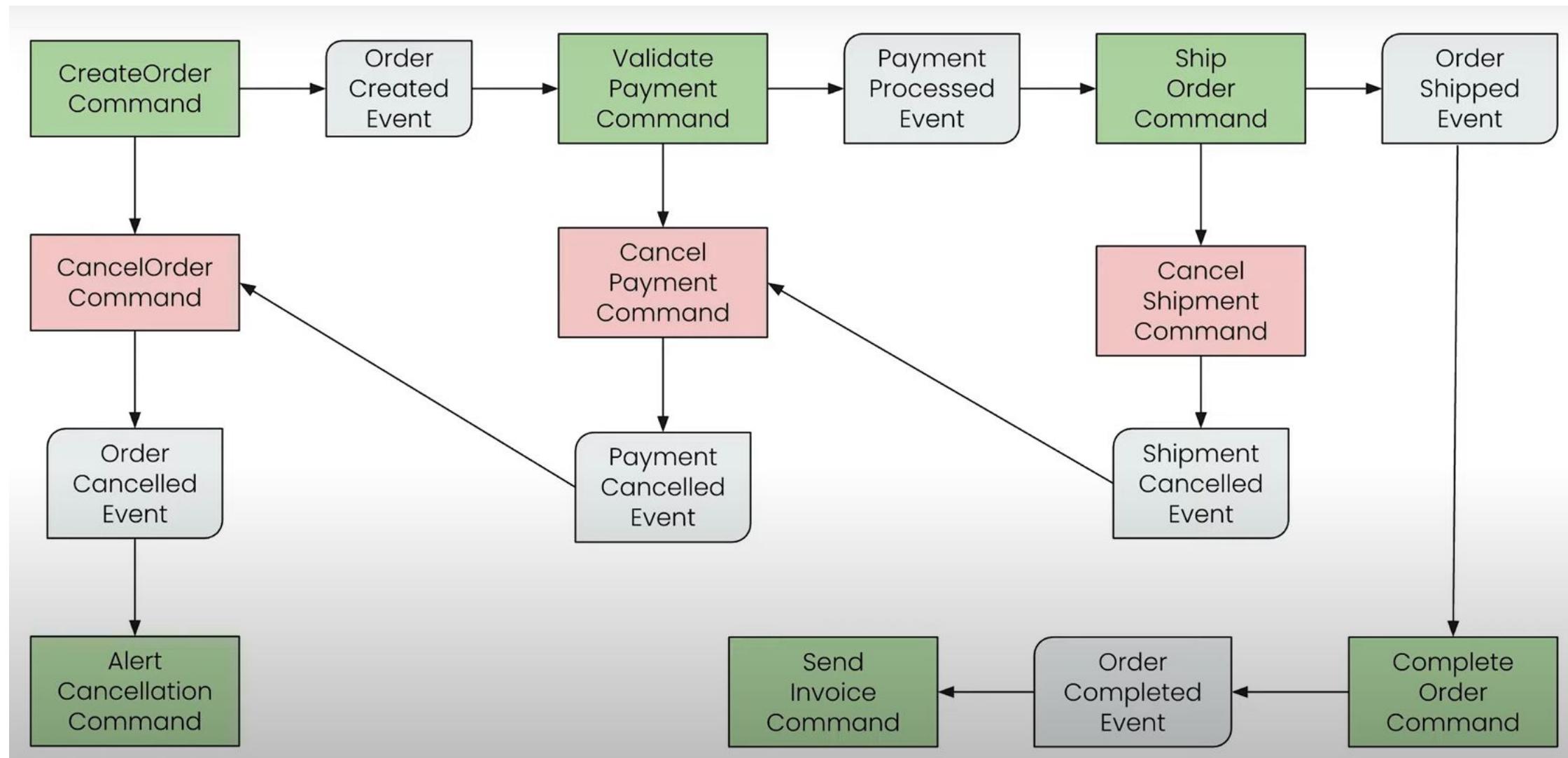
Event Sourcing

Saga Orchestration





Event Sourcing



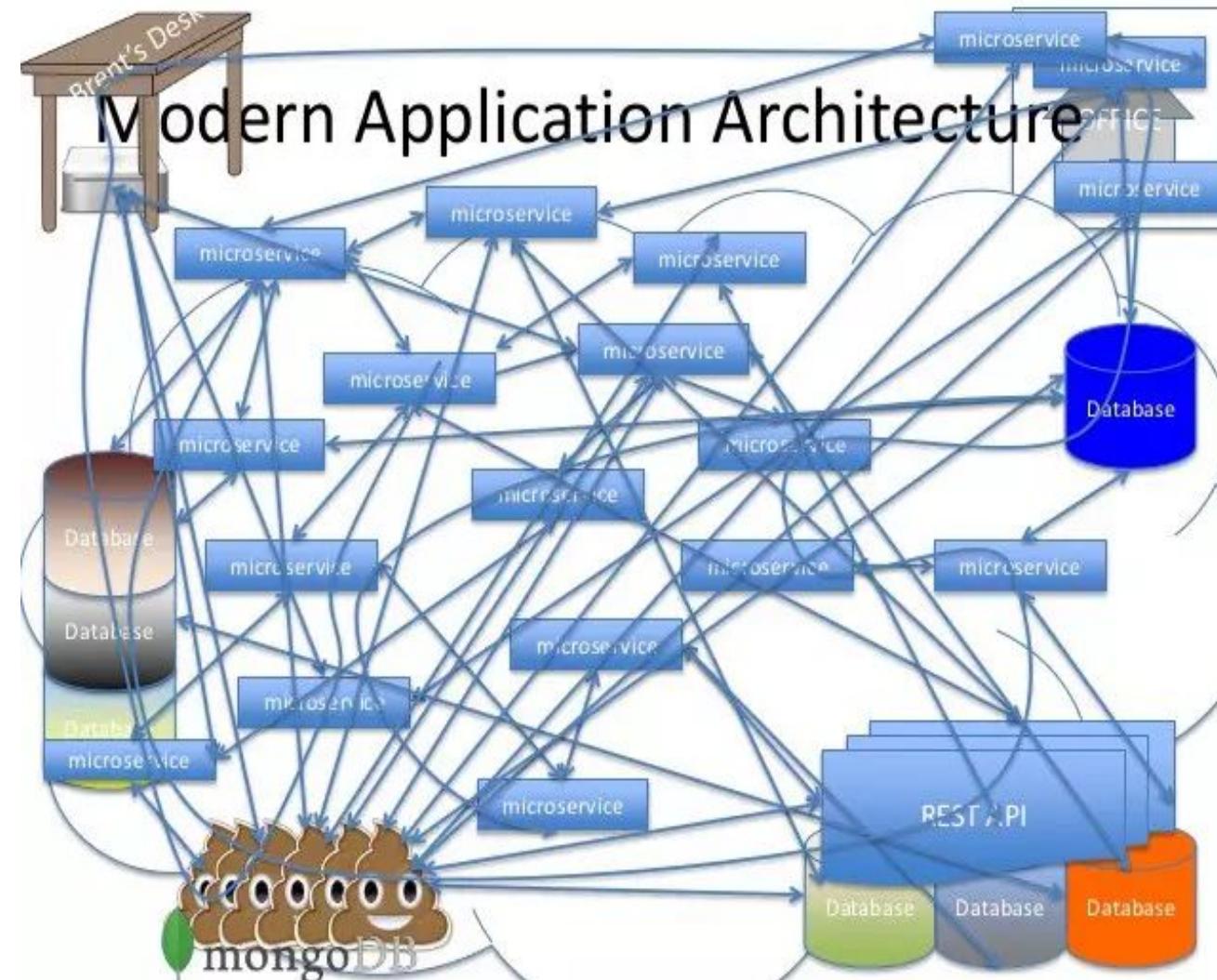


Xây dựng event sourcing với Axon framework

- 1 **Demo Event Sourcing**
- 2 **Demo Event Sourcing with Saga**



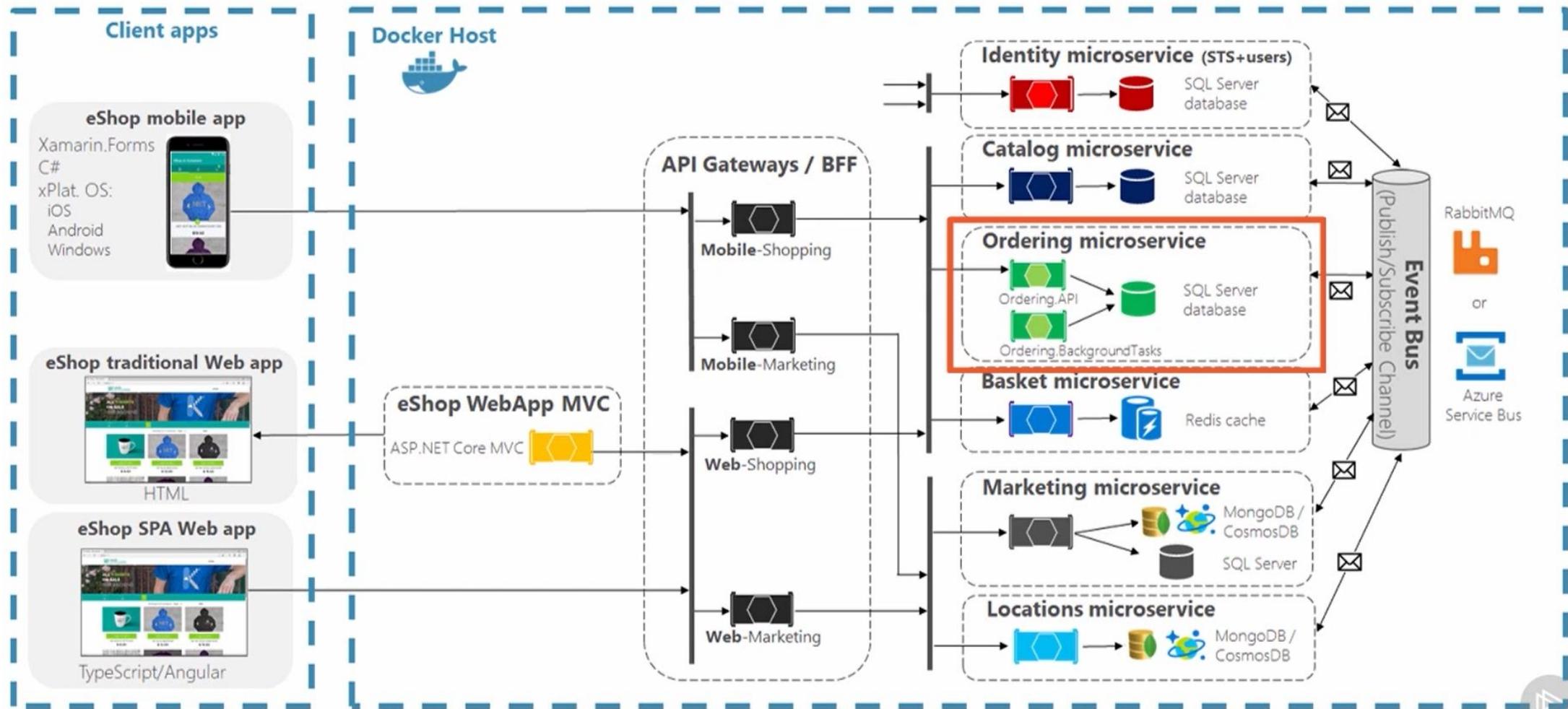
What is API Gateway





What is API Gateway

eShopOnContainers Architecture



[-- HIDE SIDEBAR](#)

API Gateway

Dashboard

APIs

Consumers

Plugins

Upstreams

Certificates

SNIs

Vitals

Overview

Security

RBAC

Kong Admin v0.0.11

Kong v0.31

Total Requests ⓘ

Timeframe: Last 5 Minutes

UTC

View: Hostname

Show All Nodes

cluster

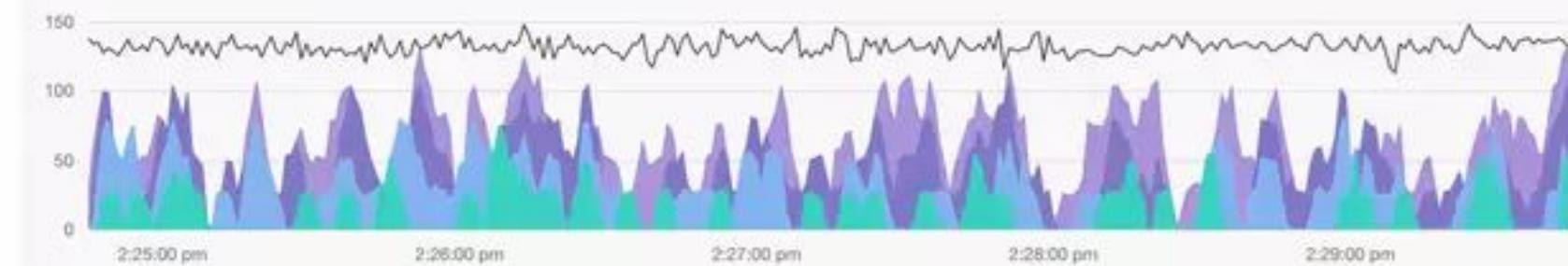
kong-rc-76547984b7-5d997

kong-rc-76547984b7-9t2cm

kong-rc-76547984b7-1p7zf

kong-rc-76547984b7-pvcn2

kong-rc-76547984b7-zp7mw kong-rc-76547984b7-h6q7n kong-rc-76547984b7-ztdtm kong-rc-76547984b7-xb6qs



Node Details

Address	0.0.0.0:8001
Version	0.31-enterprise-edition
Hostname	kong-rc-76547984b7-zp7mw

Datastore Details

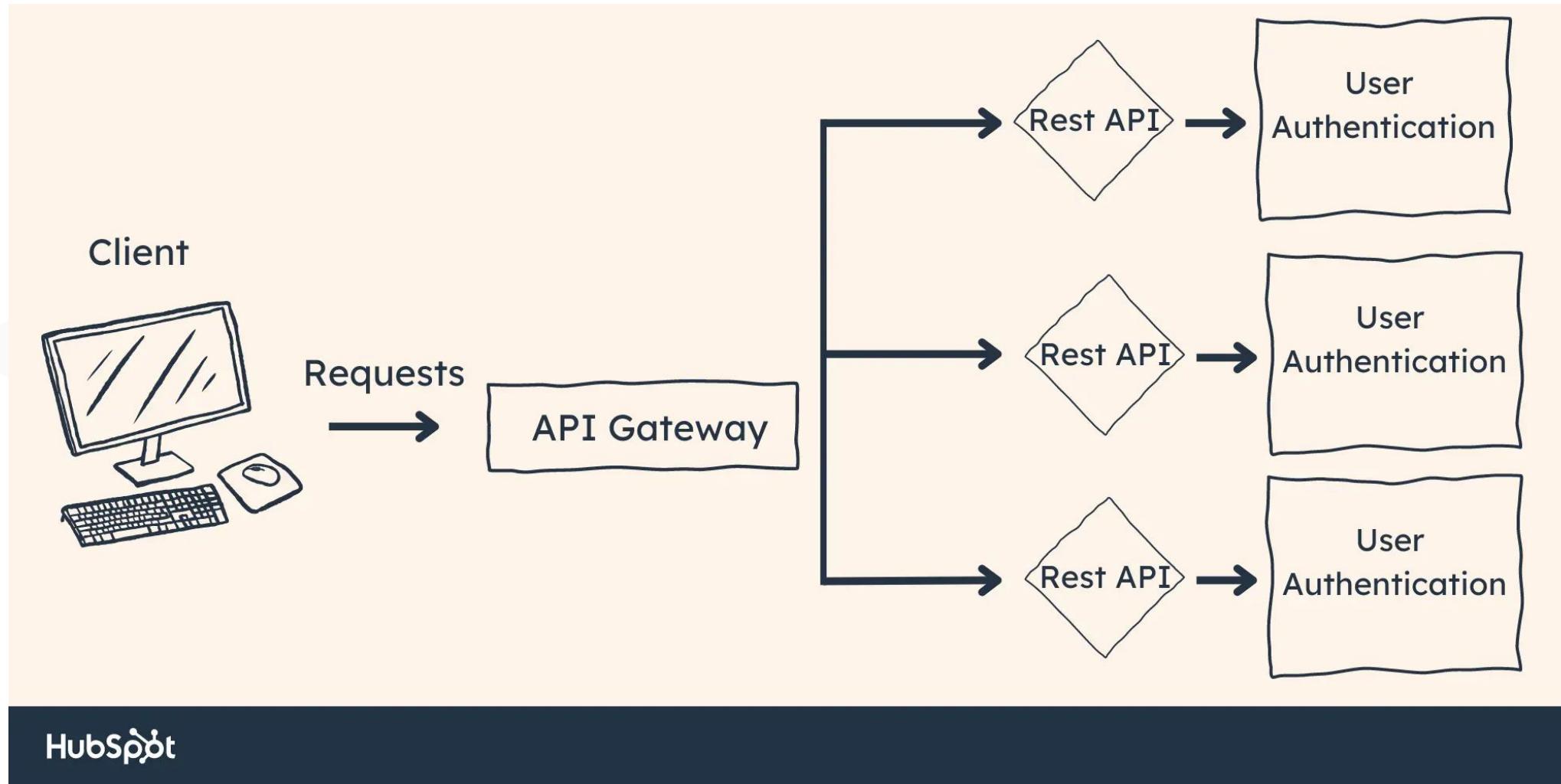
Type	postgres
Host/Port	postgres:5432
User	kong
SSL	false

Port Details

Admin Port	8001
Admin SSL Port	8444
Port	
Proxy Port	8000
Proxy SSL Port	8443



Load balancing



HubSpot



Authentication



JWT

Verify and authenticate
JSON Web Tokens

Support by:
 Kong Inc.



Key Authentication

Add key authentication to
your Services

Support by:
 Kong Inc.



LDAP Authentication Advanced

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

Support by:
 Kong Inc.

Enterprise



LDAP Authentication

Integrate Kong with a
LDAP server

Support by:
 Kong Inc.



OAuth 2.0 Introspection

Integrate Kong with a
third-party OAuth 2.0
Authorization Server

Support by:
 Kong Inc.



OAuth 2.0 Authentication

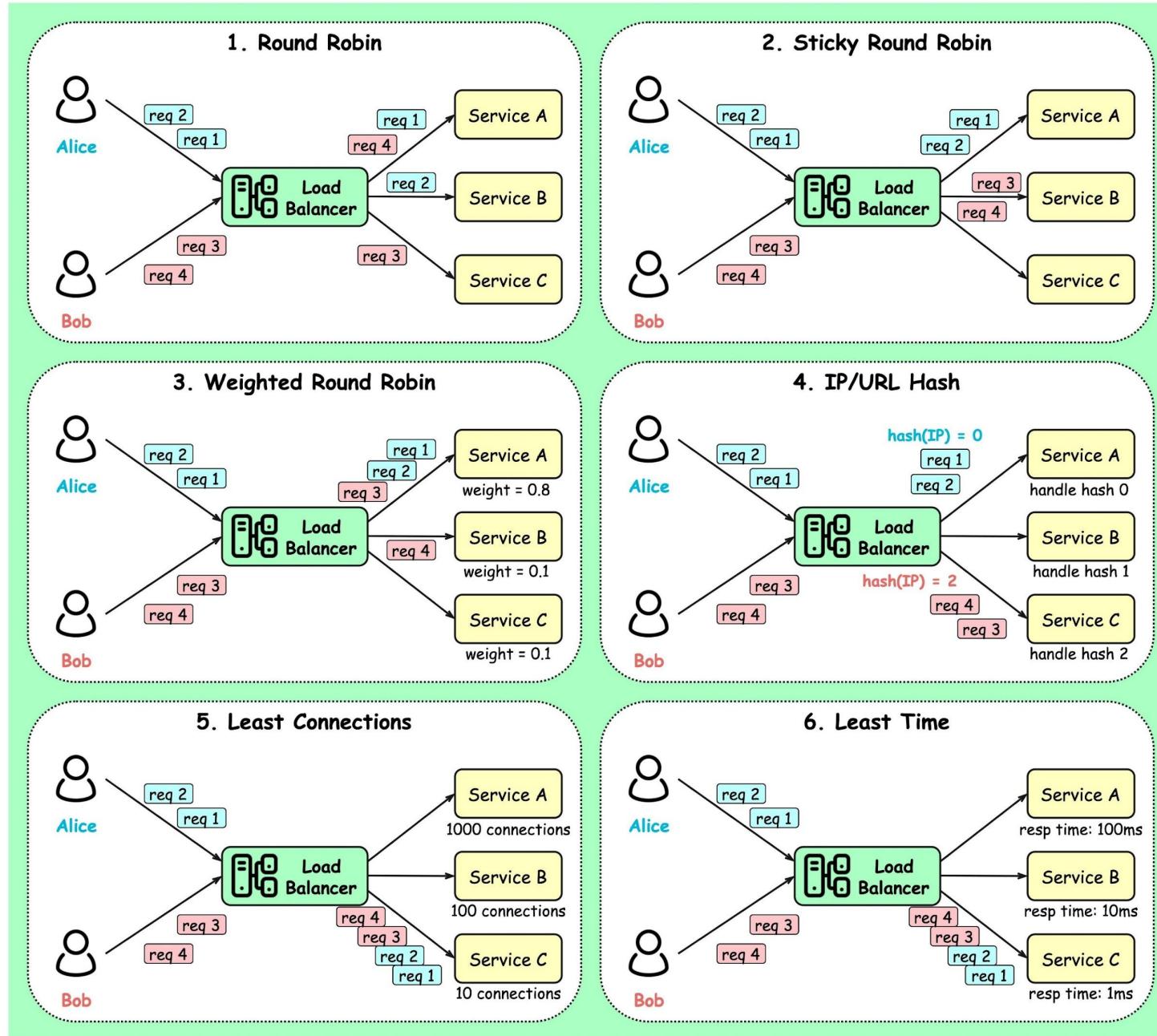
Add OAuth 2.0
authentication to your
Services

Support by:
 Kong Inc.



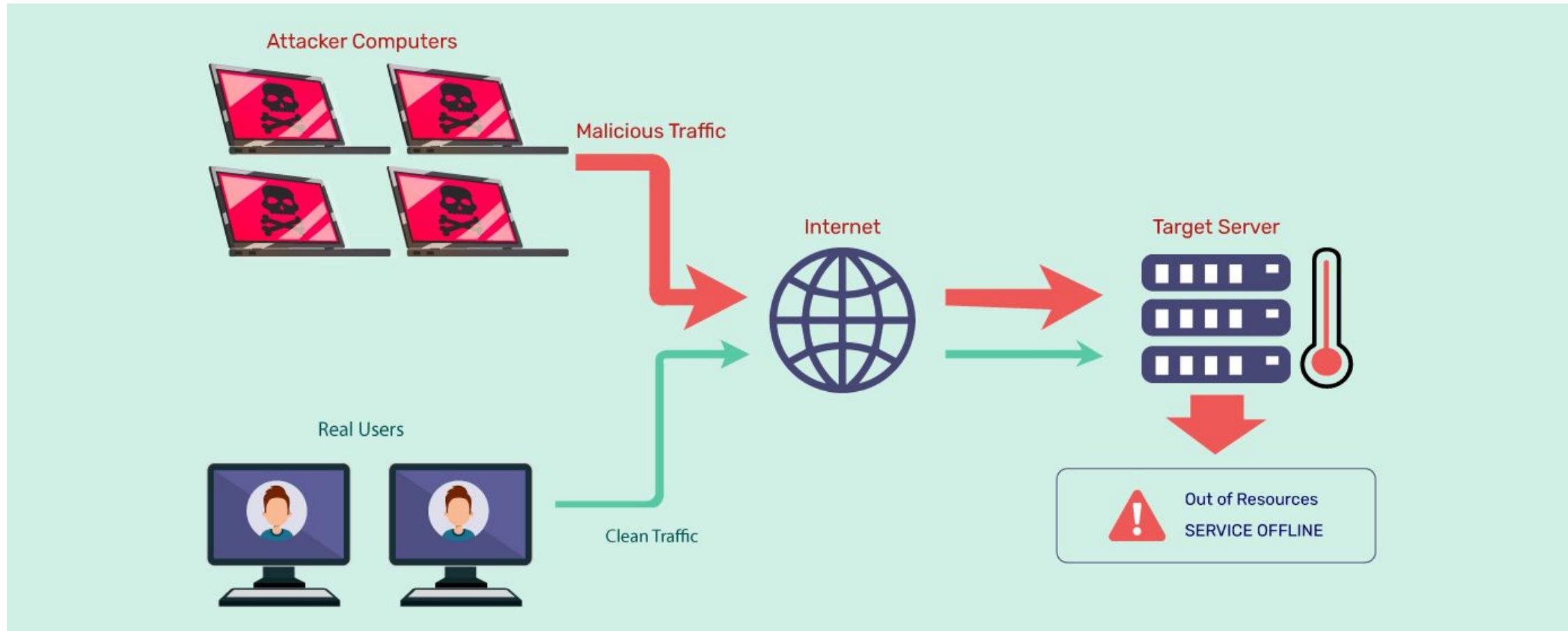
Nhược điểm API Gateway





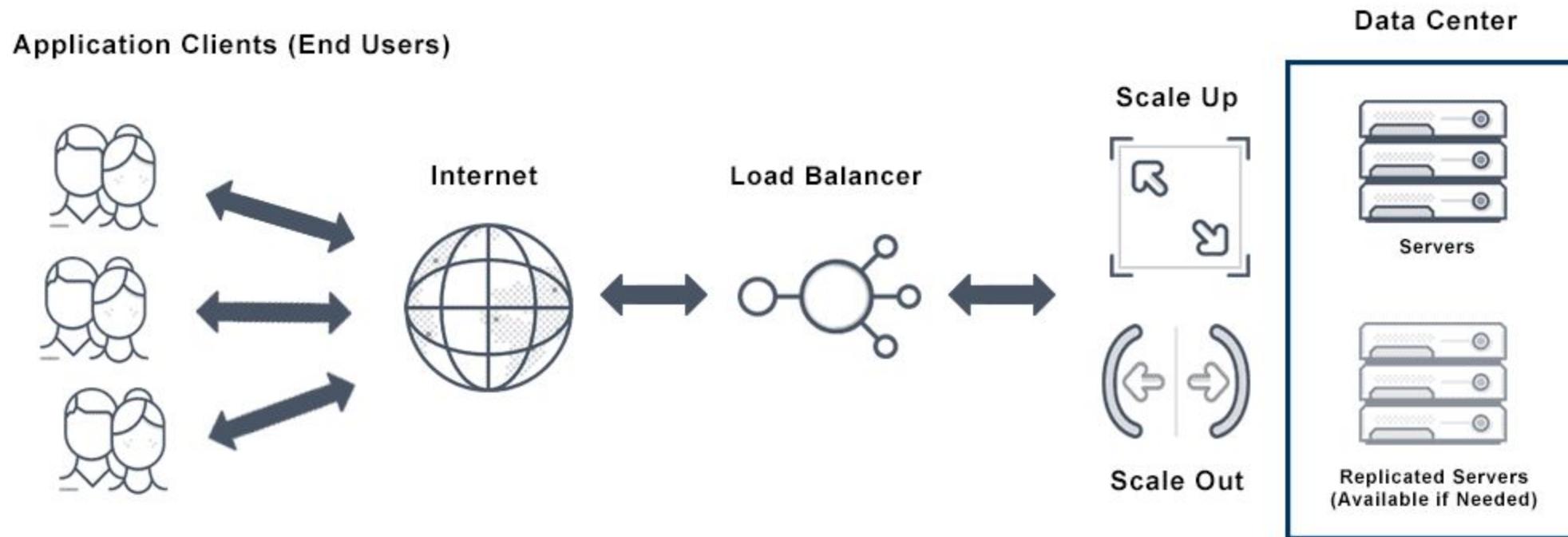


Why we need Rate limiting



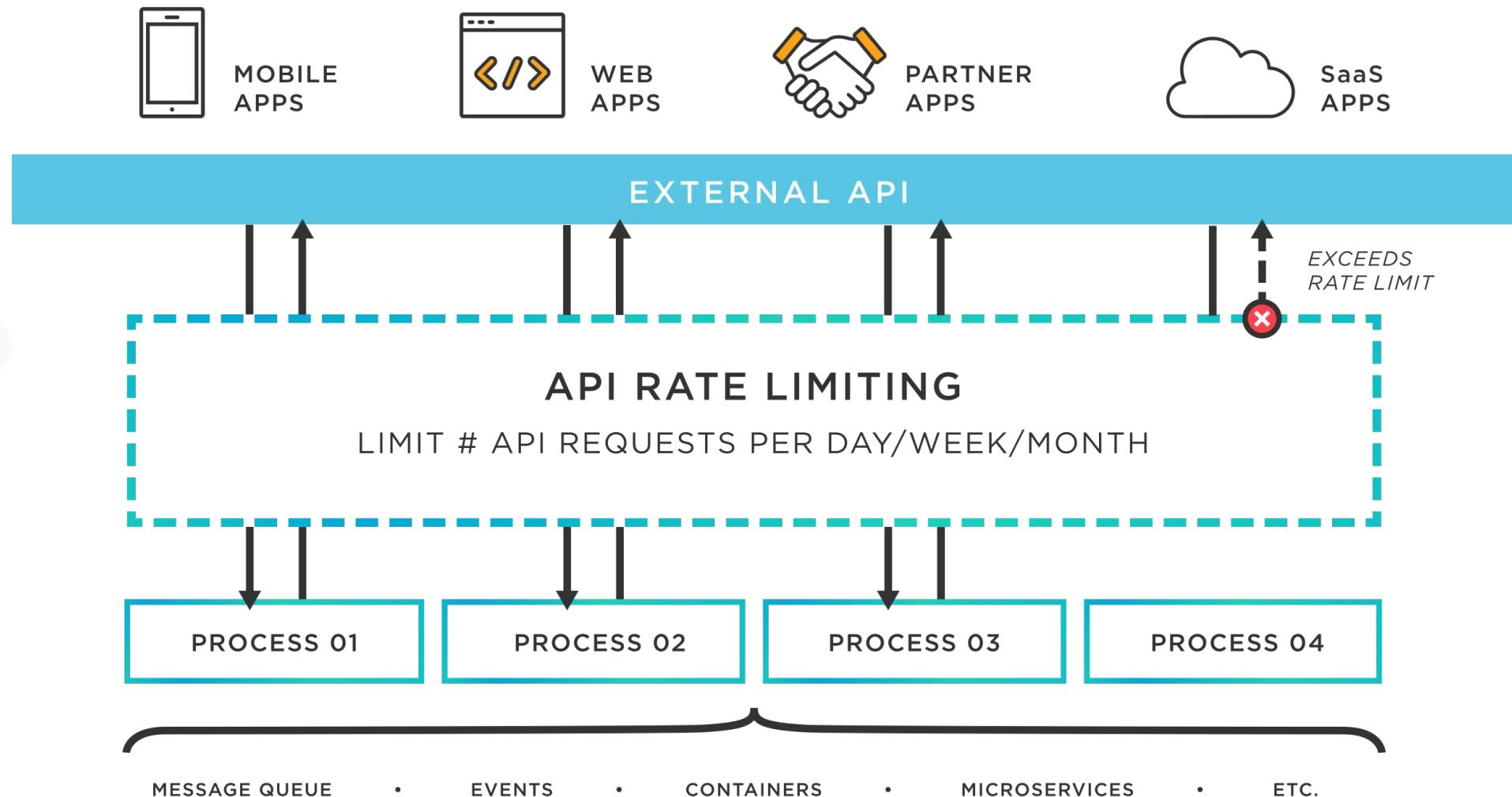


Auto Scale ?





Rate limiting





Throttling là gì ?

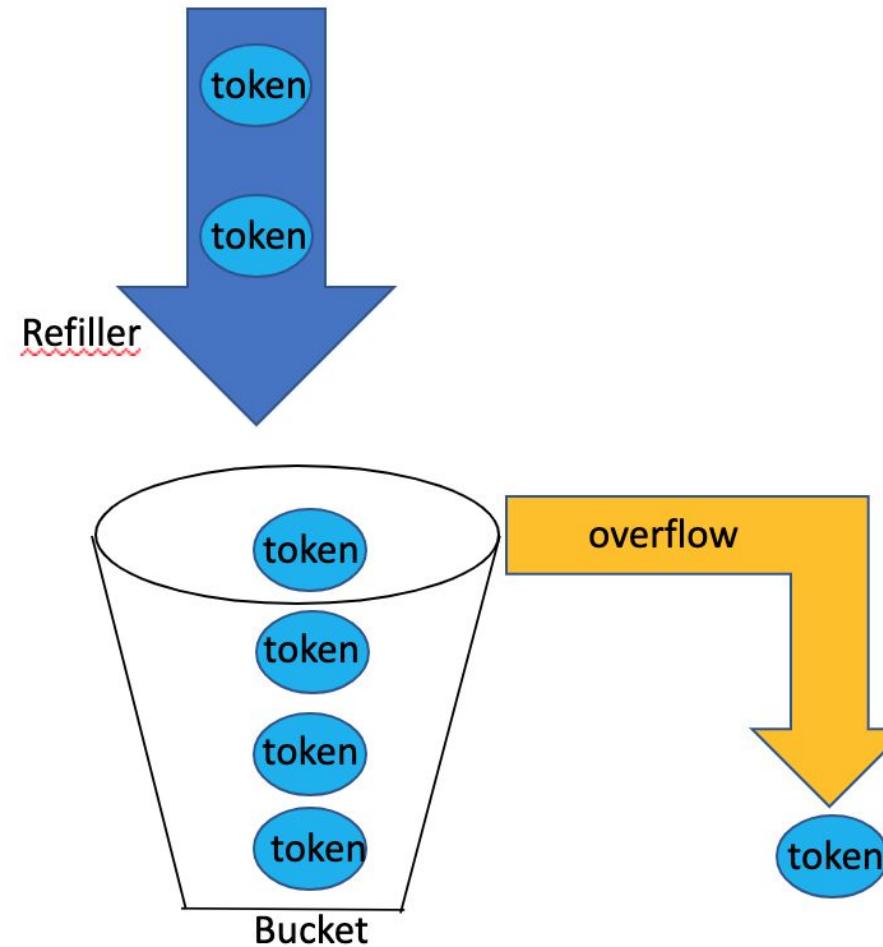
- User-Level Rate Limiting
- Server-Level Rate Limiting
- Geography-Based Rate Limiting





What Are the Algorithms Used for Rate Limiting?

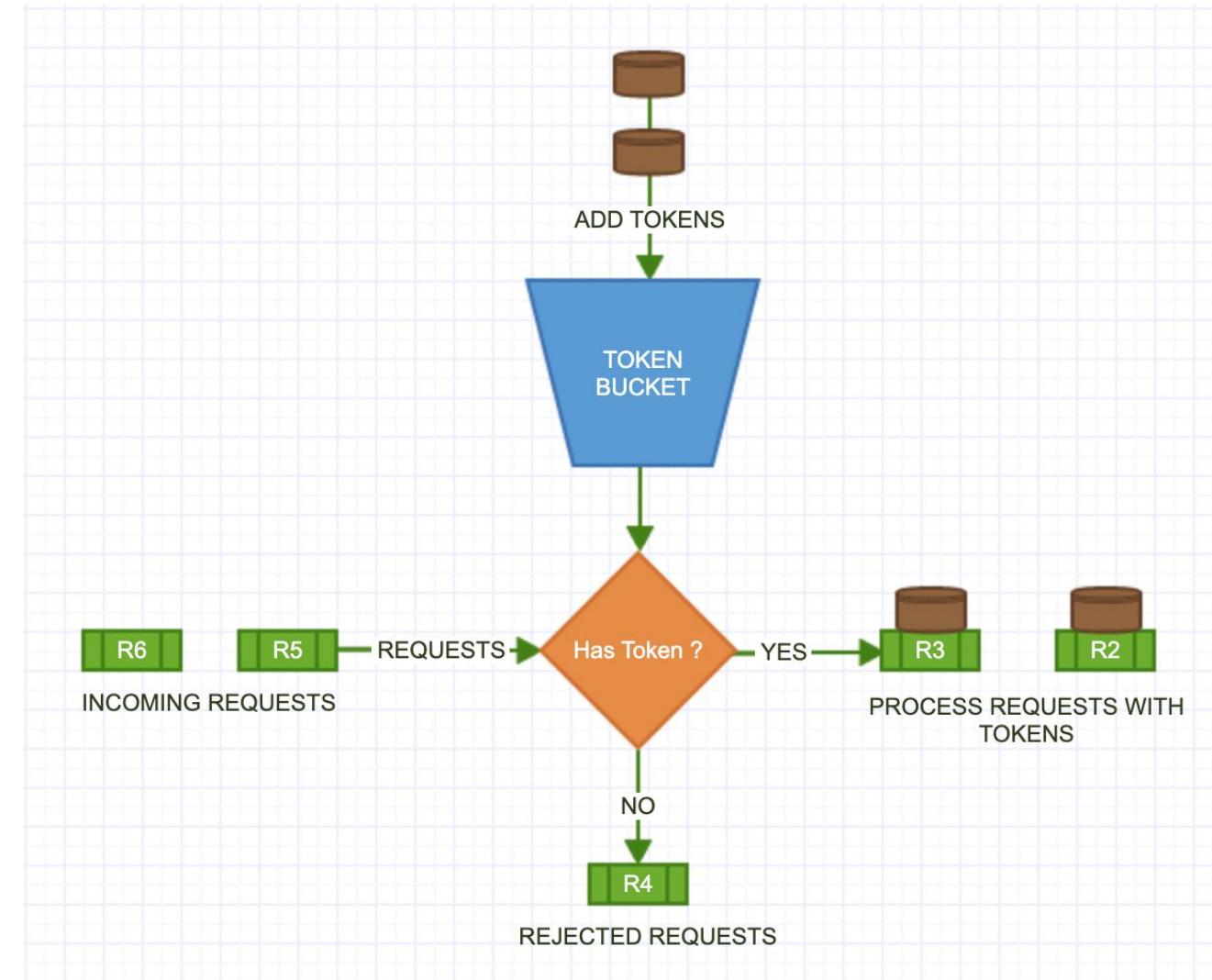
Thuật toán Token Bucket





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.
- Nay, 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

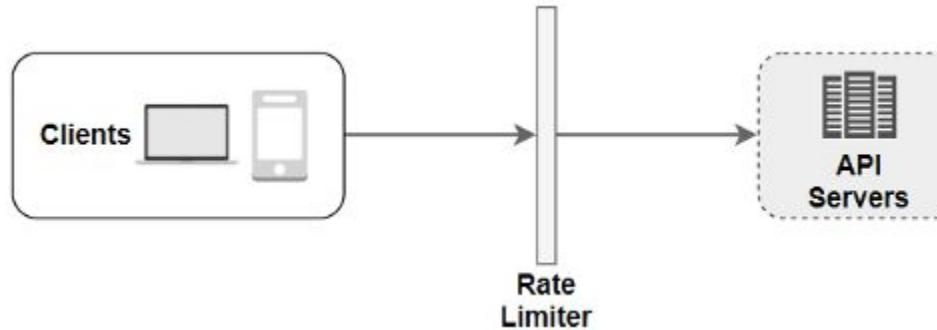




Xây dựng Rate Limiting

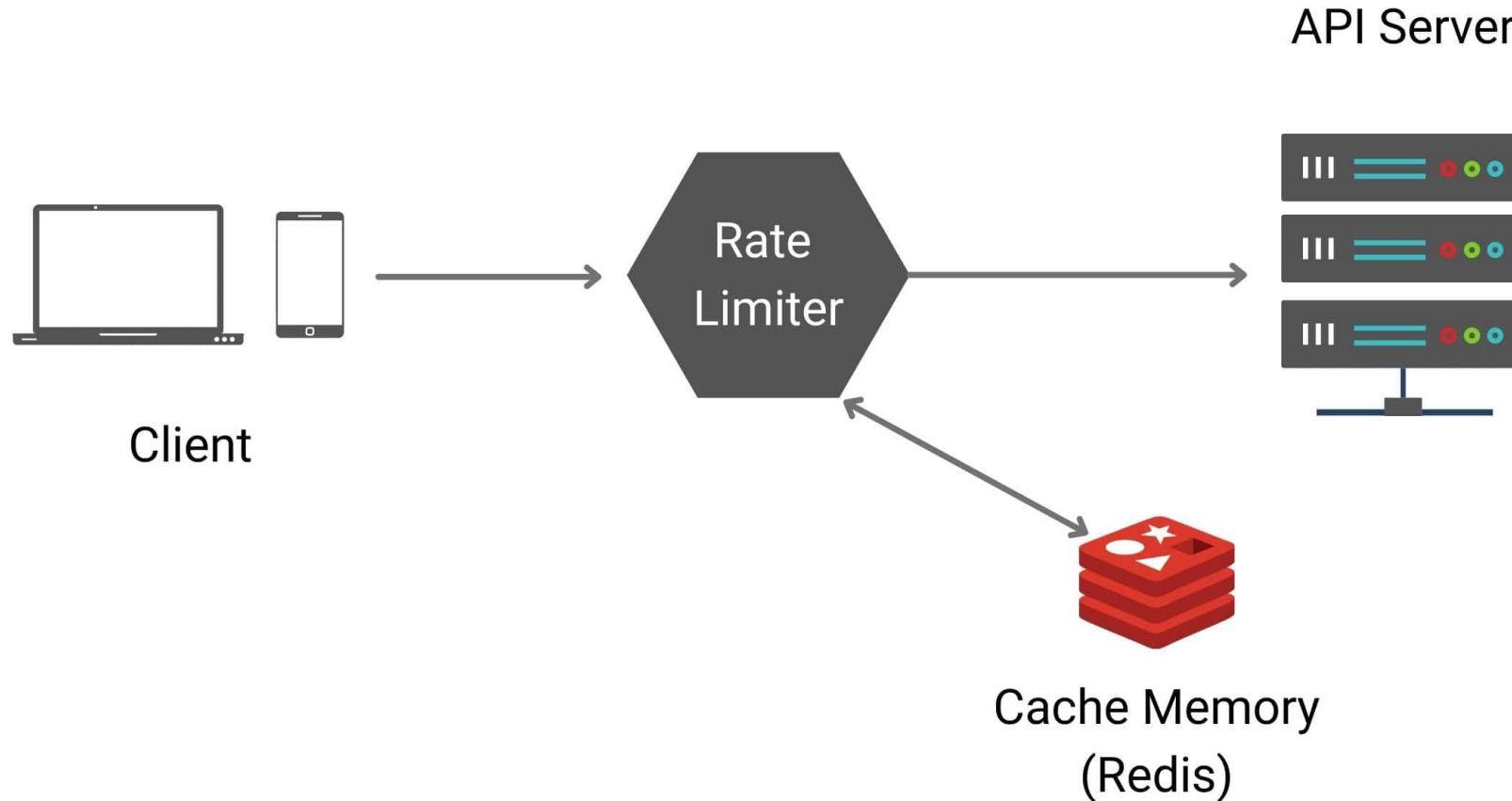
Where to put the Rate Limiter ?

- Client side
- Server side
- Middleware



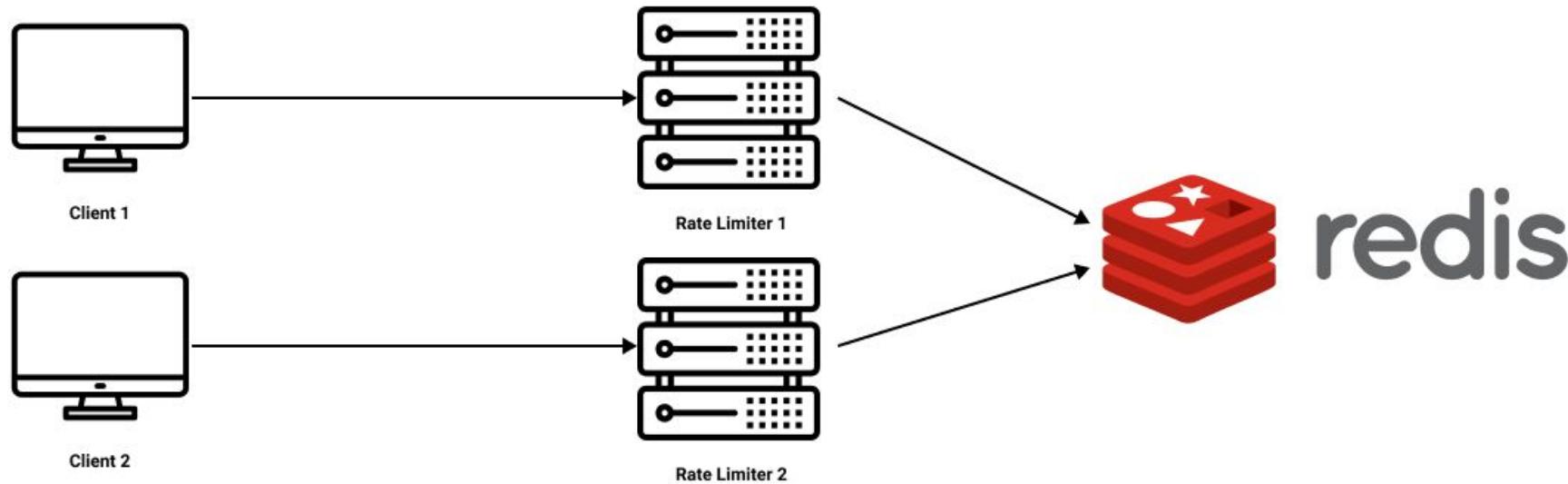


Xây dựng Rate Limiting



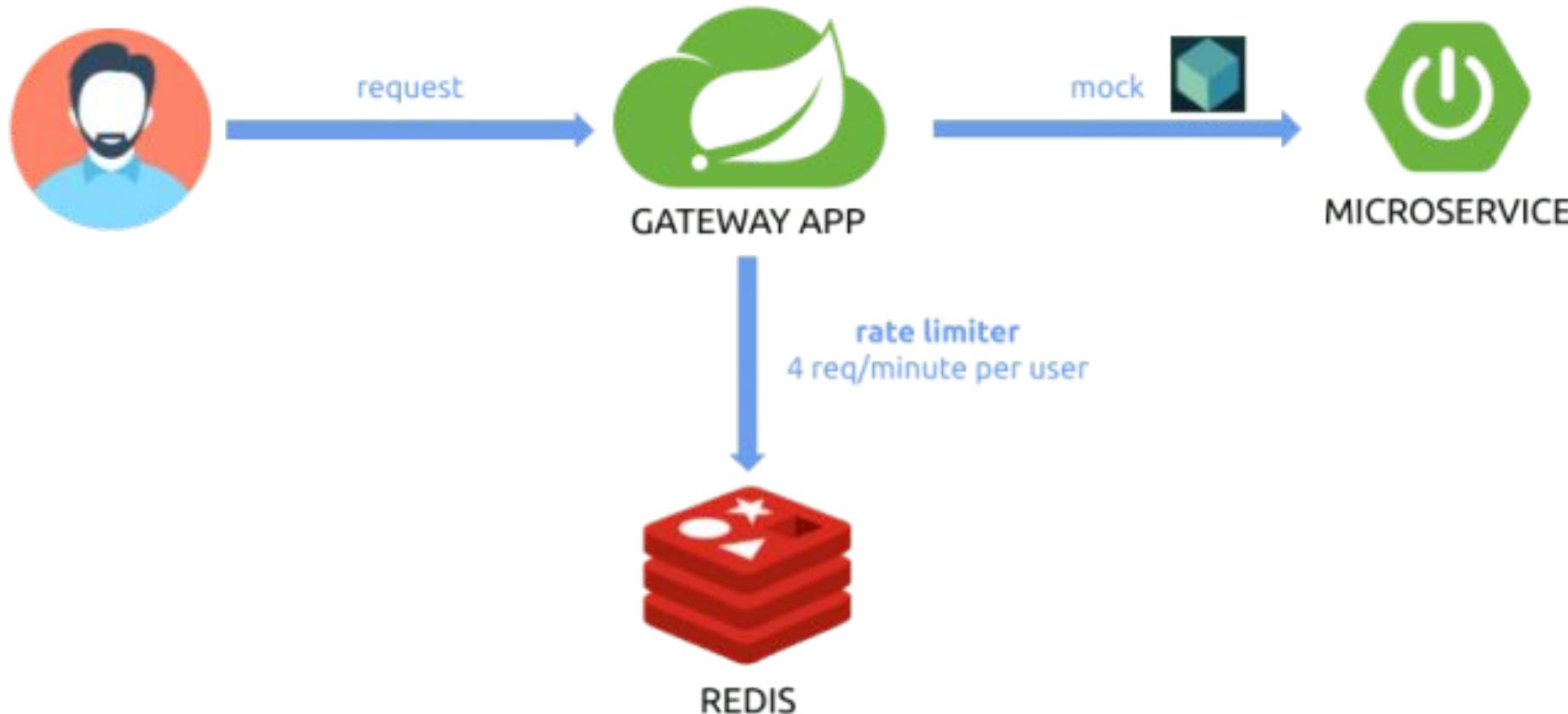


Xây dựng Rate Limiting



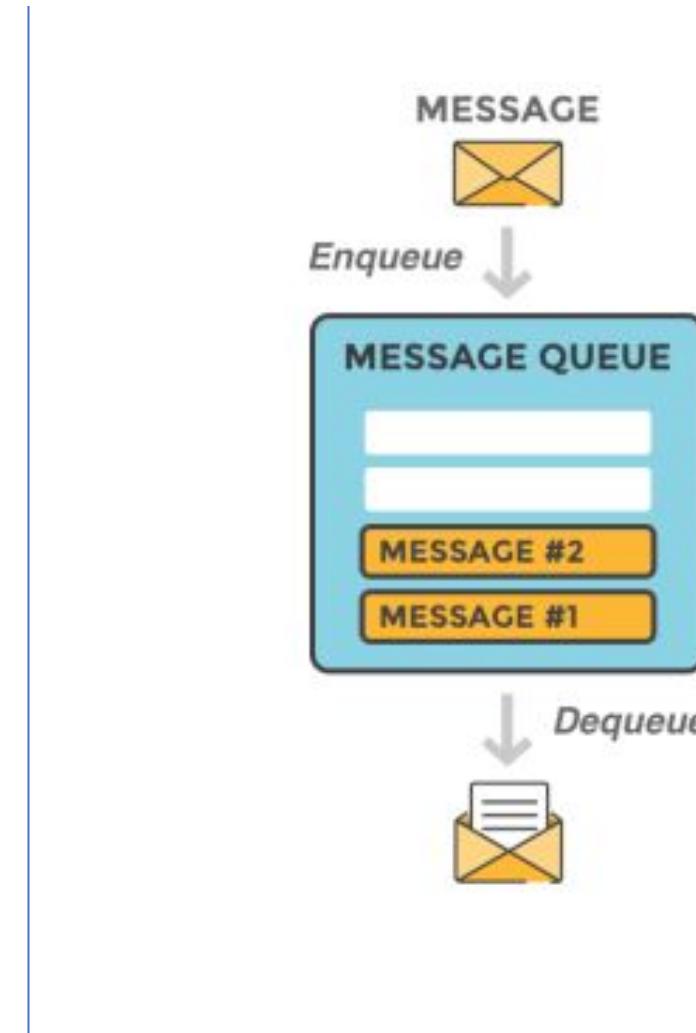
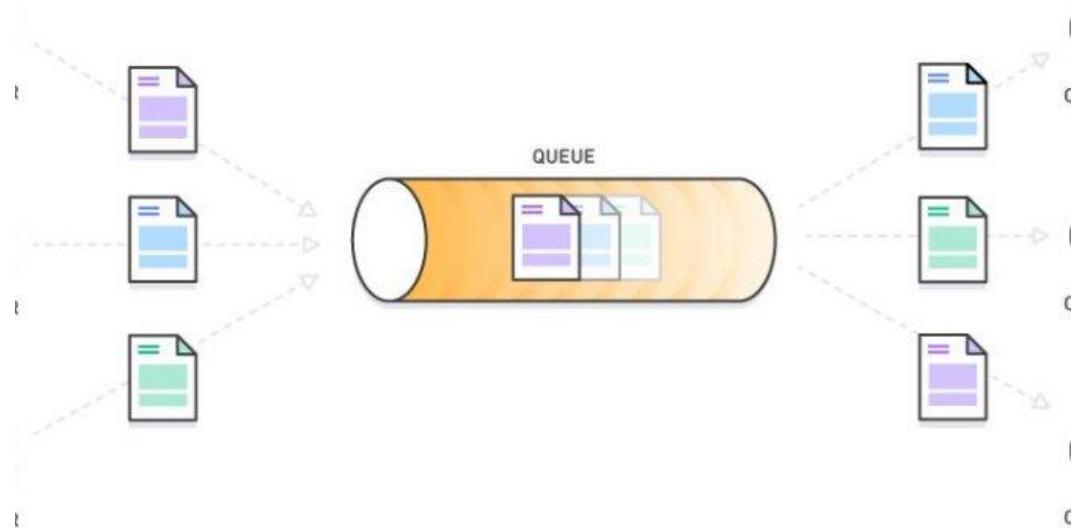


Xây dựng Rate Limiting với Spring cloud gateway





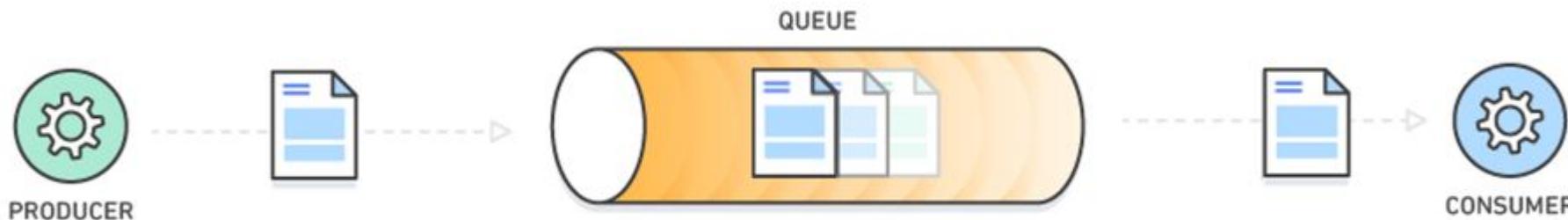
Message Queue là gì ?





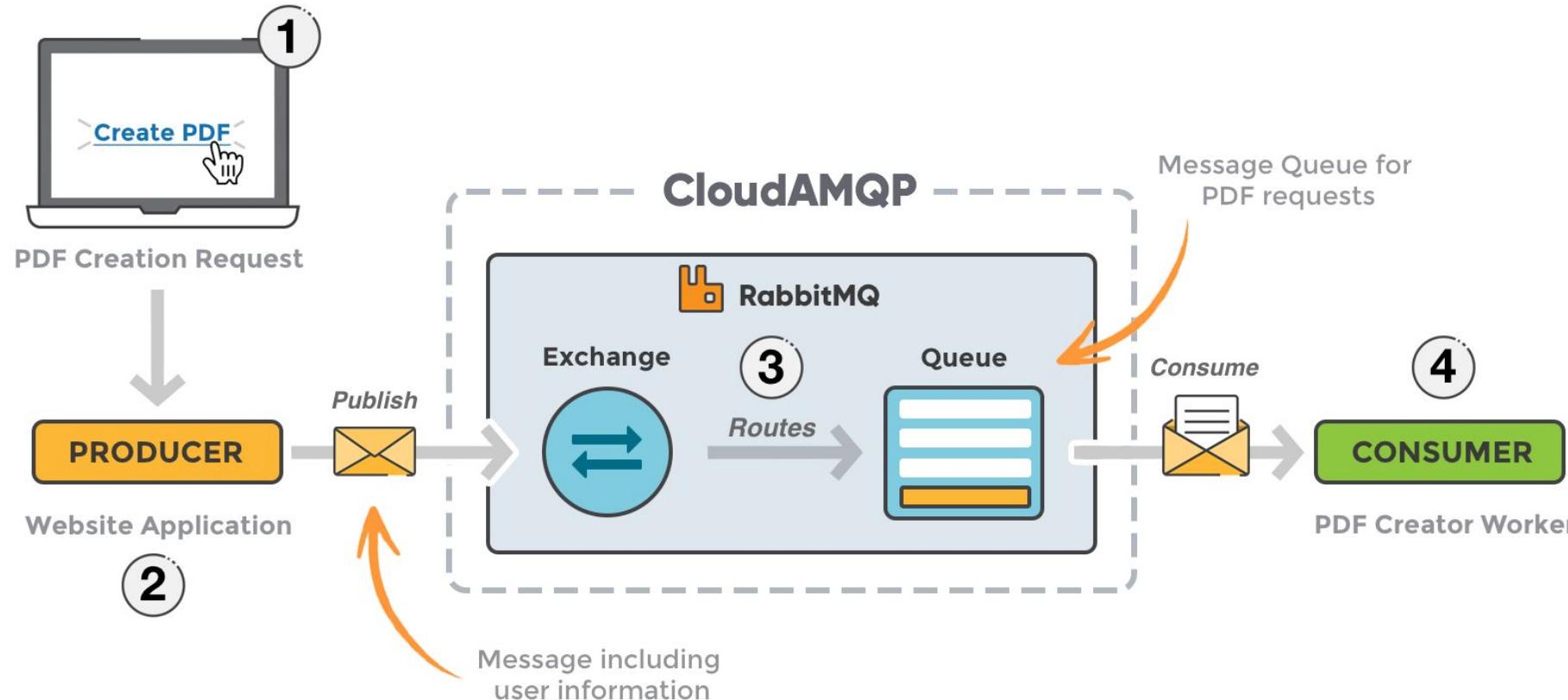
Thành phần của Message Queue ?

- **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

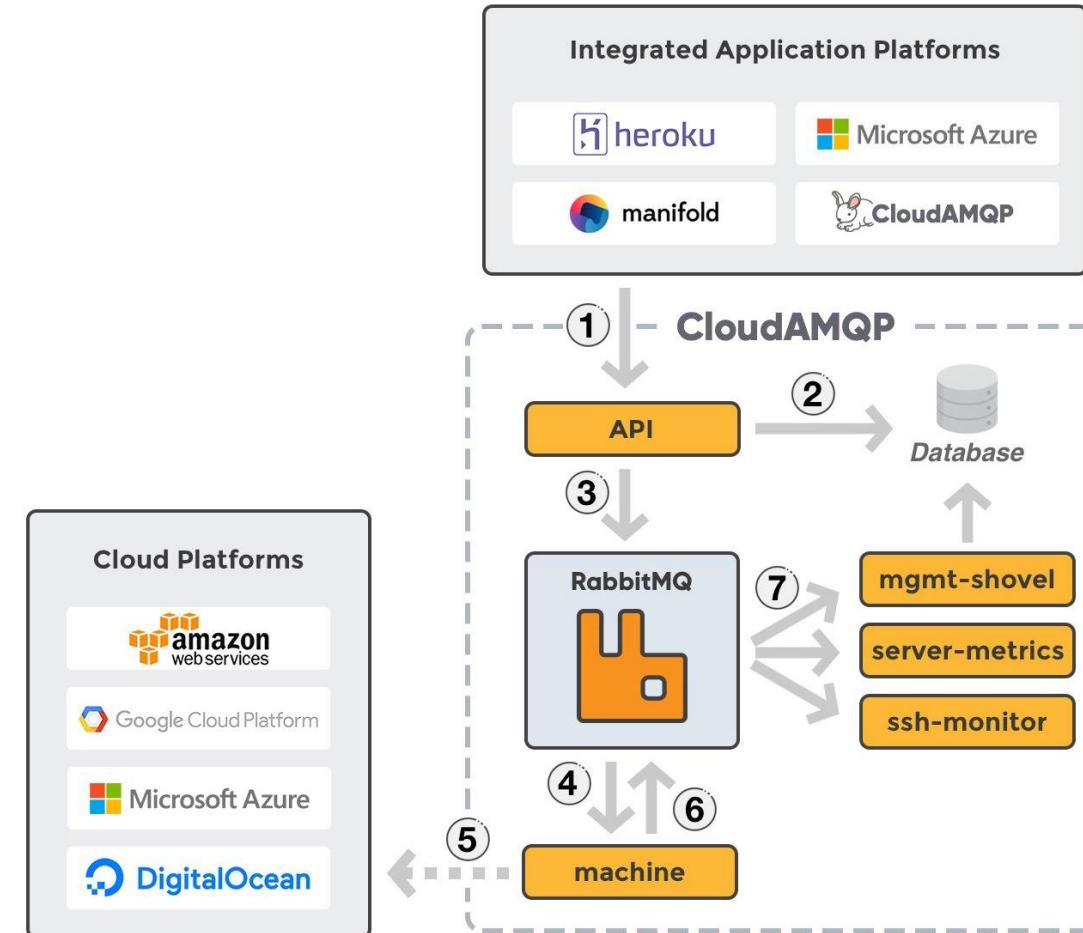
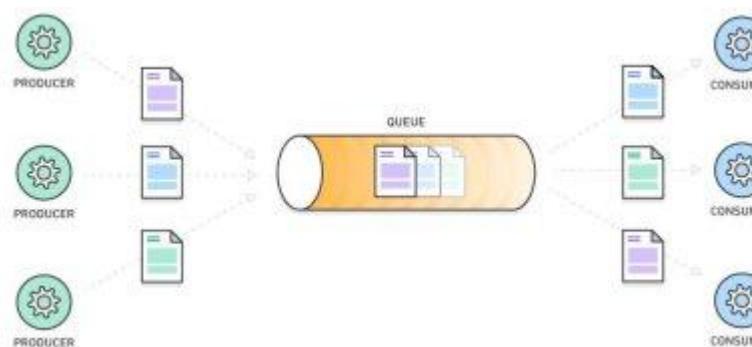




Tại sao phải sử dụng Message Queue

Ưu điểm về Message Queue

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



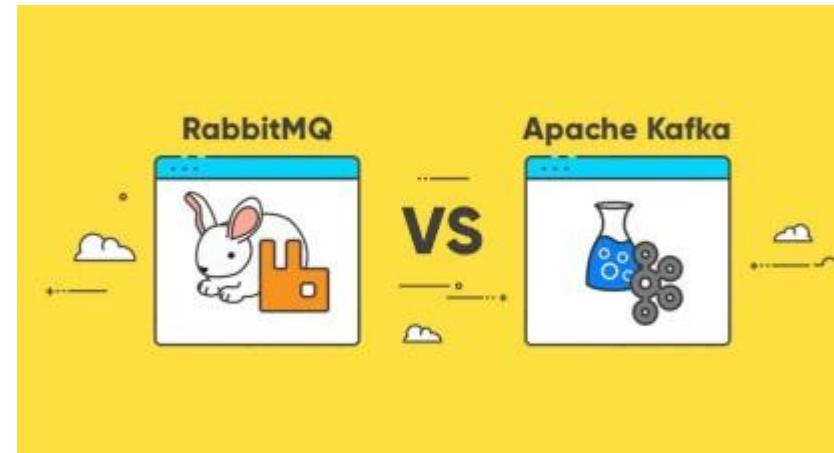


Một số điểm cần lưu ý về MS

- 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 nay bao gồm:

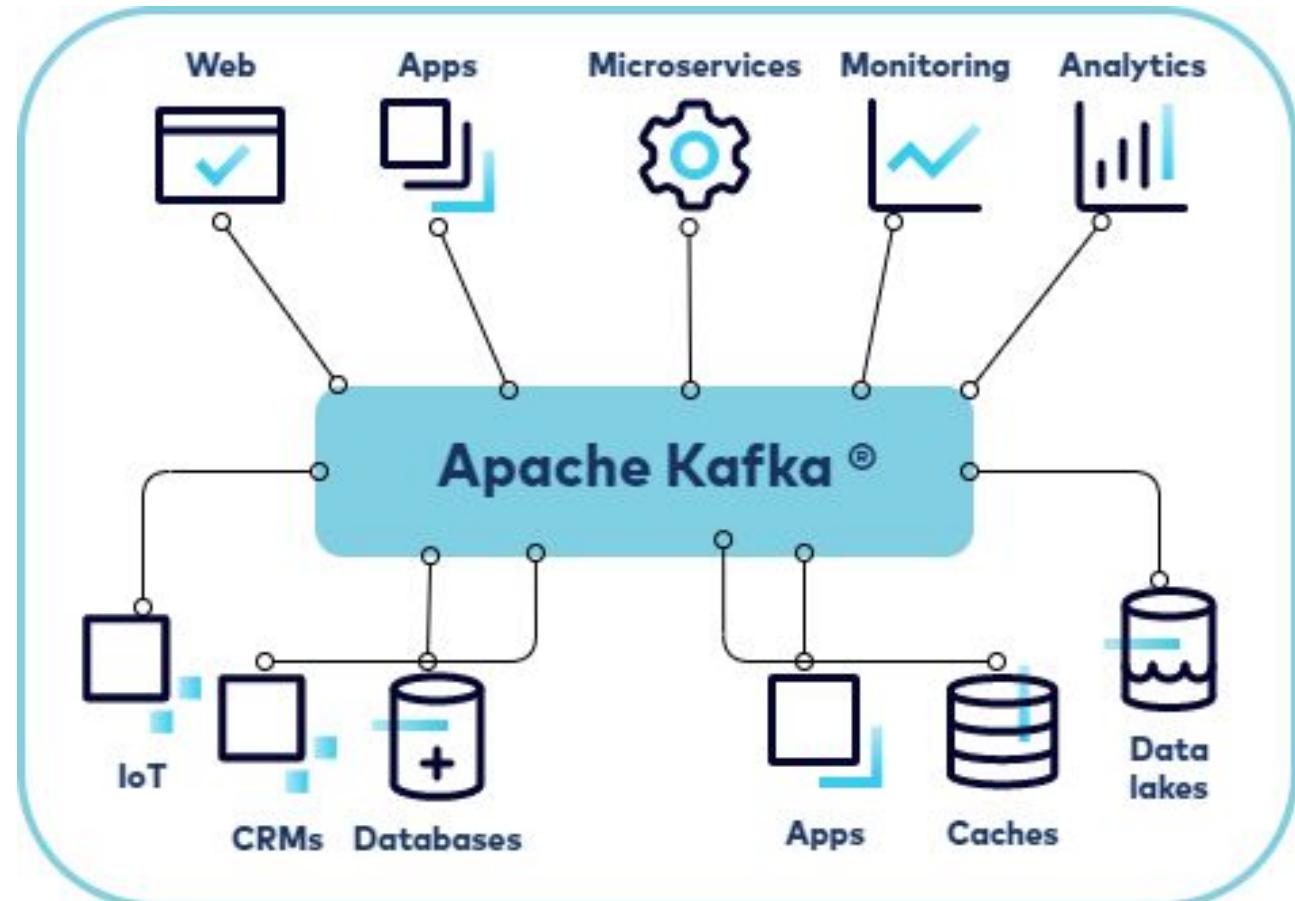
- RabbitMQ
- Kafka
- Amazon SQS
- MSMQ (*Microsoft Message Queuing*)
- RocketMQ
- ZeroMQ





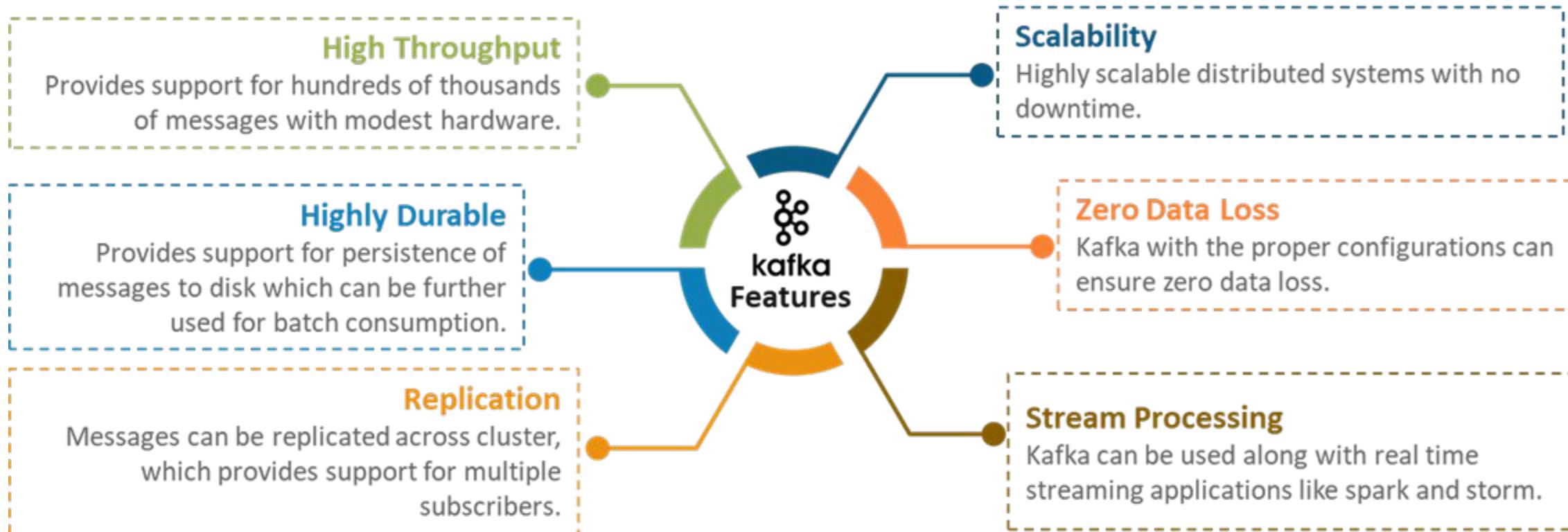
Apache Kafka là gì ?

- 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





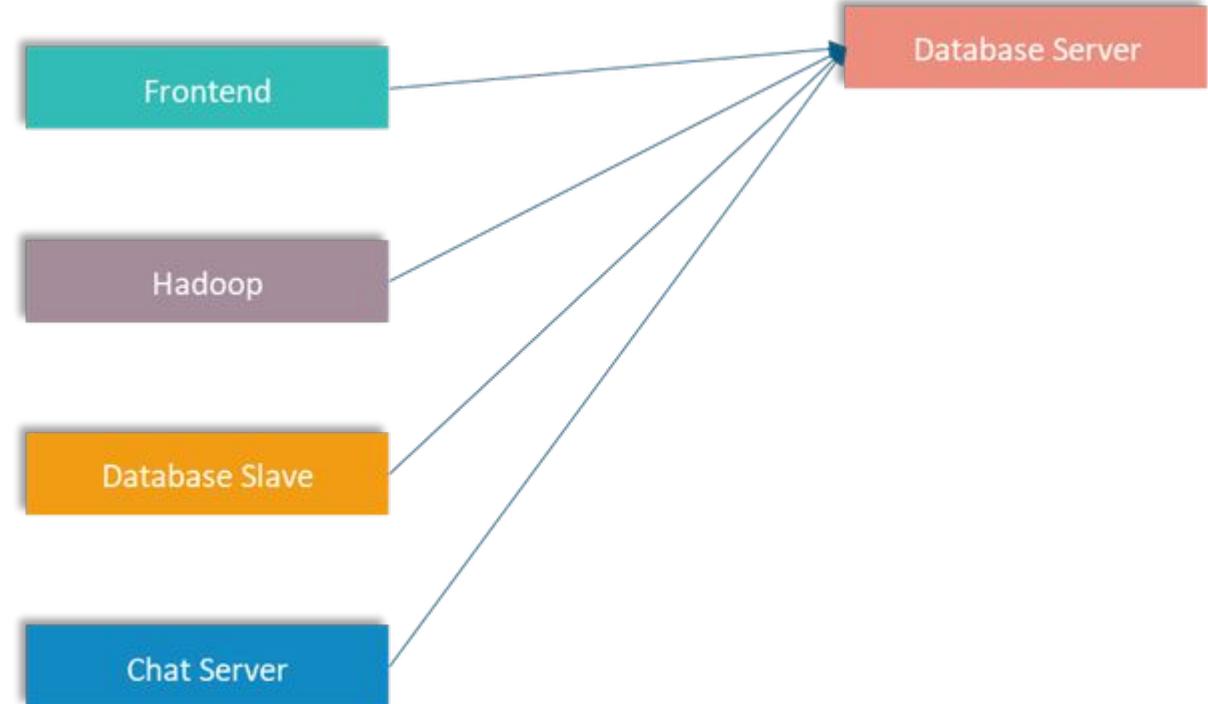
Tính năng của Apache Kafka ?





Tại sao Apache Kafka được sử dụng

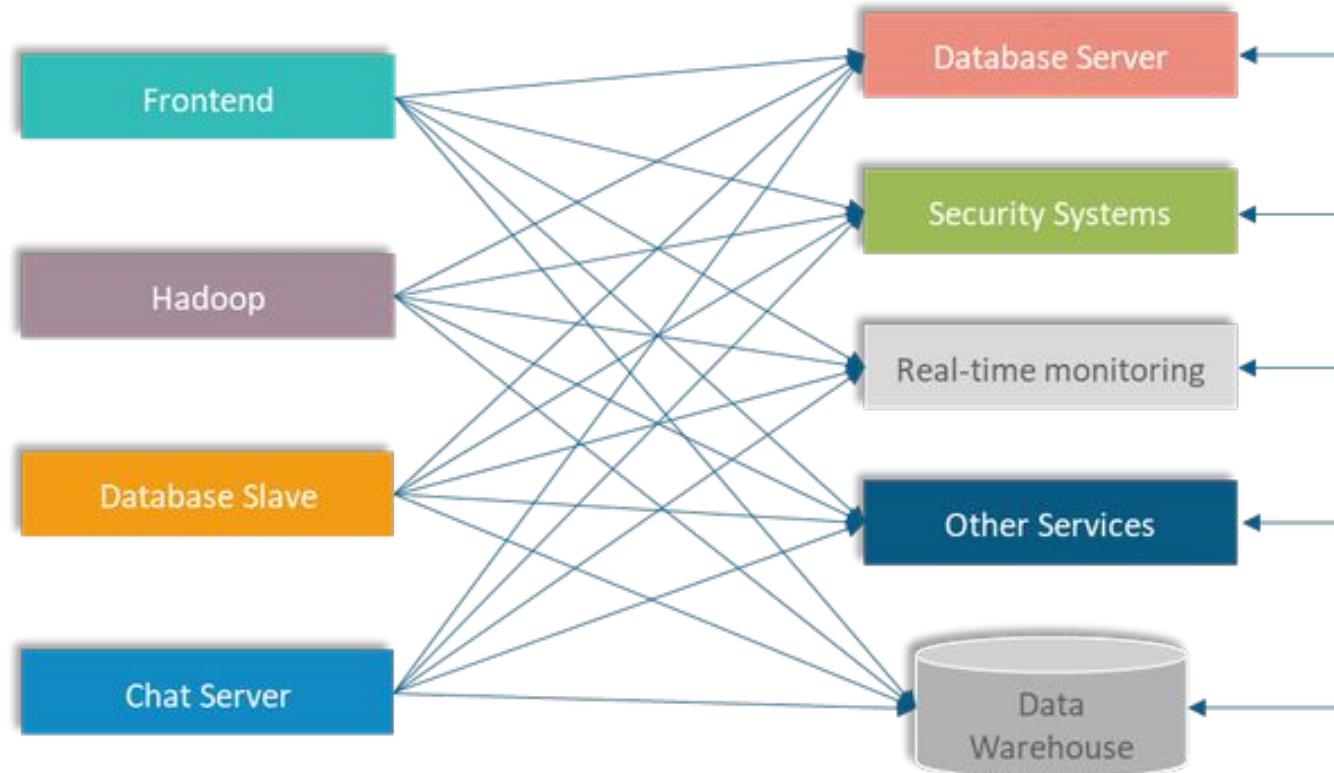
- Website Activity Monitoring: theo dõi hoạt động của website
- Stream Processing: xử lý stream
- Log Aggregation: tổng hợp log
- Metrics Collection: thu thập dữ liệu





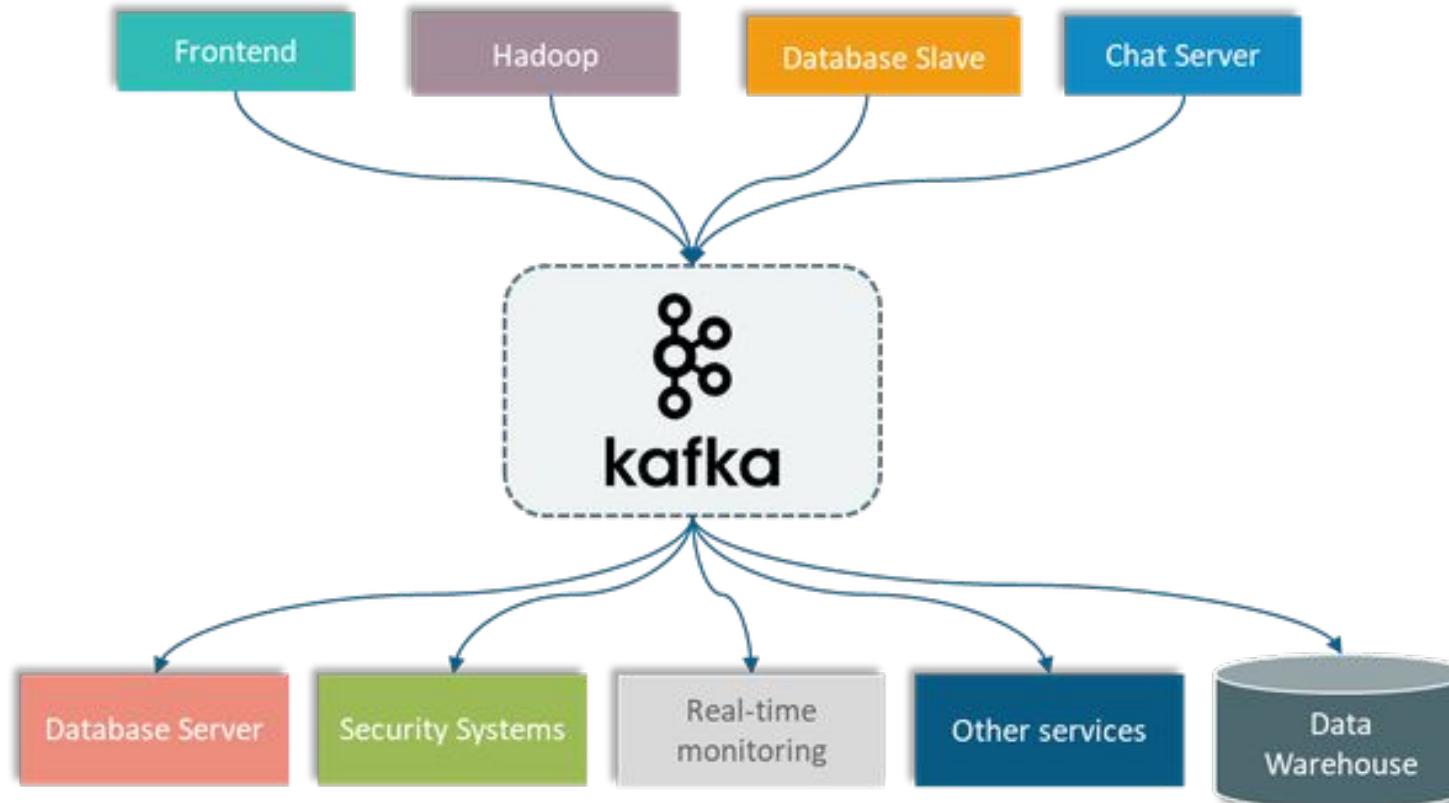
Tại sao Apache Kafka được sử dụng

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





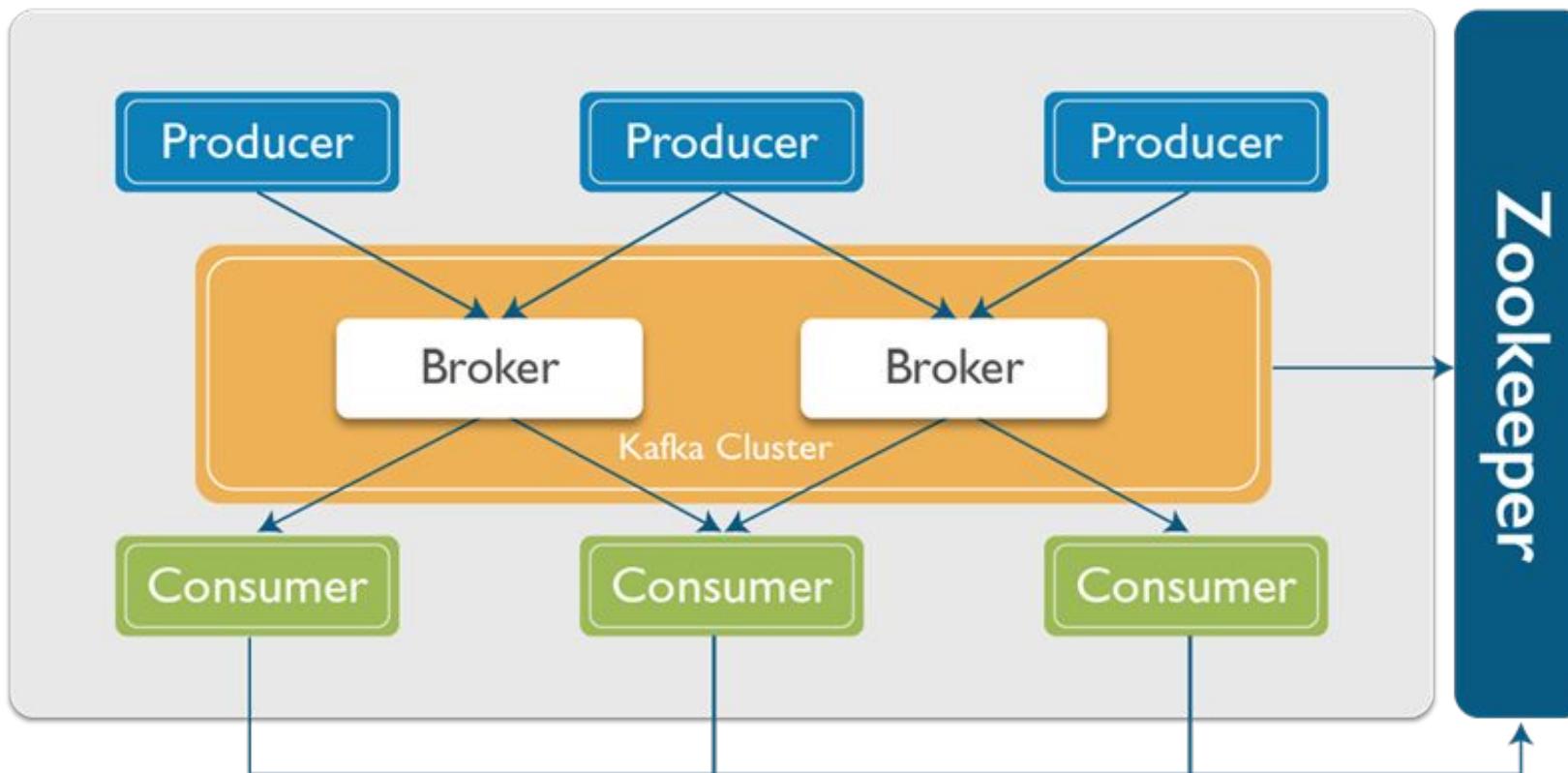
Tại sao Apache Kafka được sử dụng





Cấu trúc của Apache Kafka

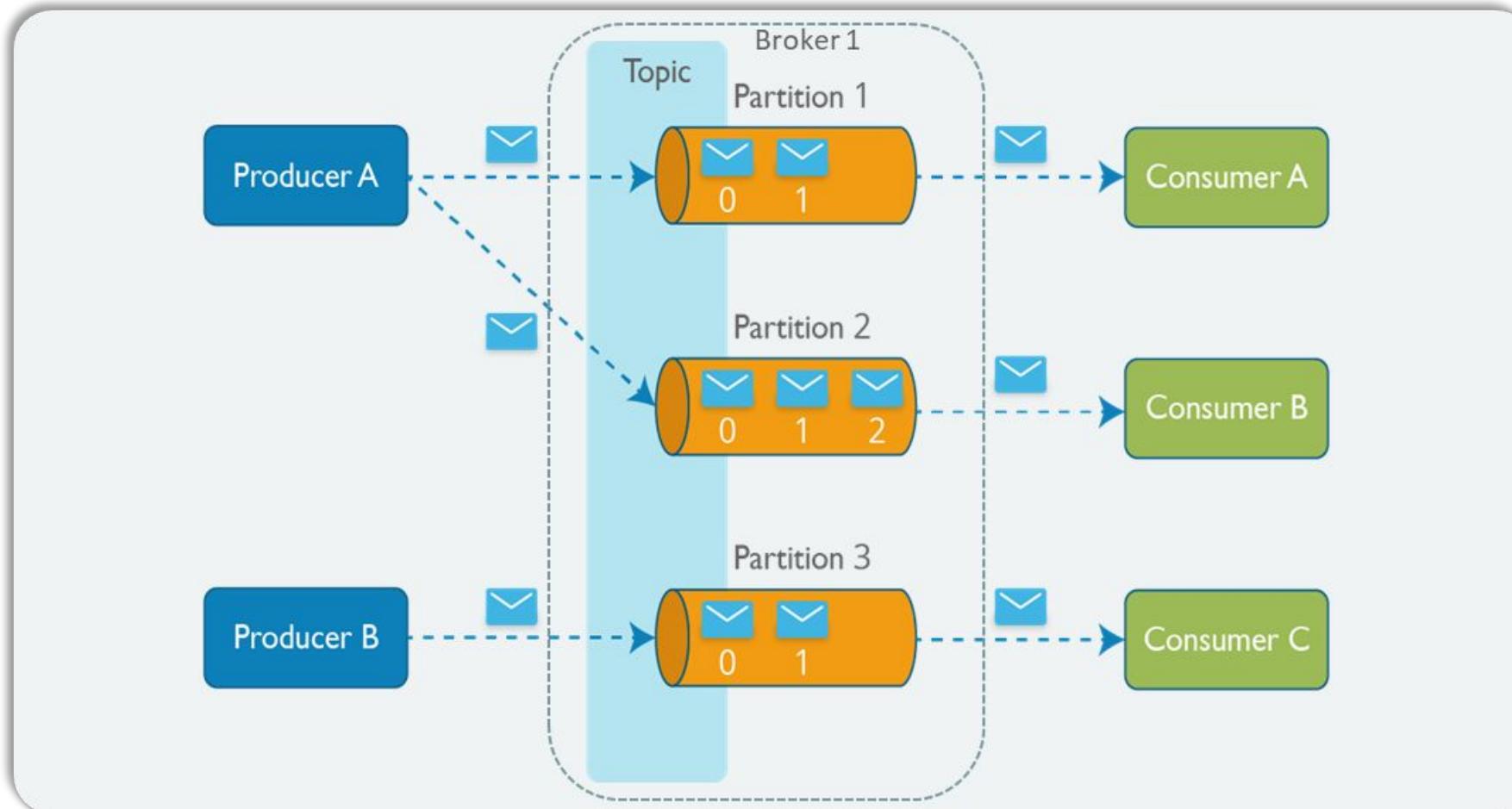
Một mô hình cấu trúc Kafka đơn giản





Cấu trúc của Apache Kafka

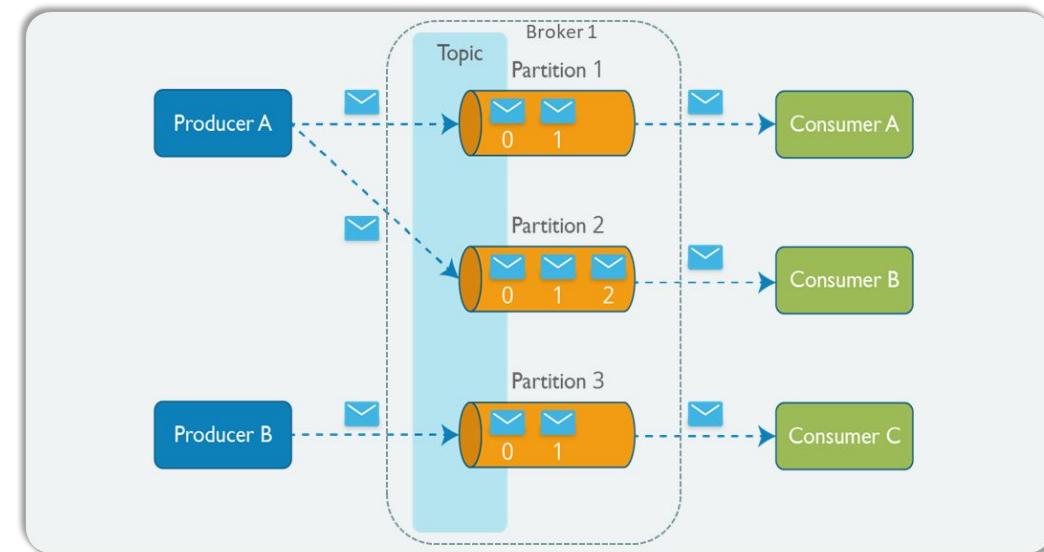
Một mô hình cấu trúc Kafka chi tiết





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.



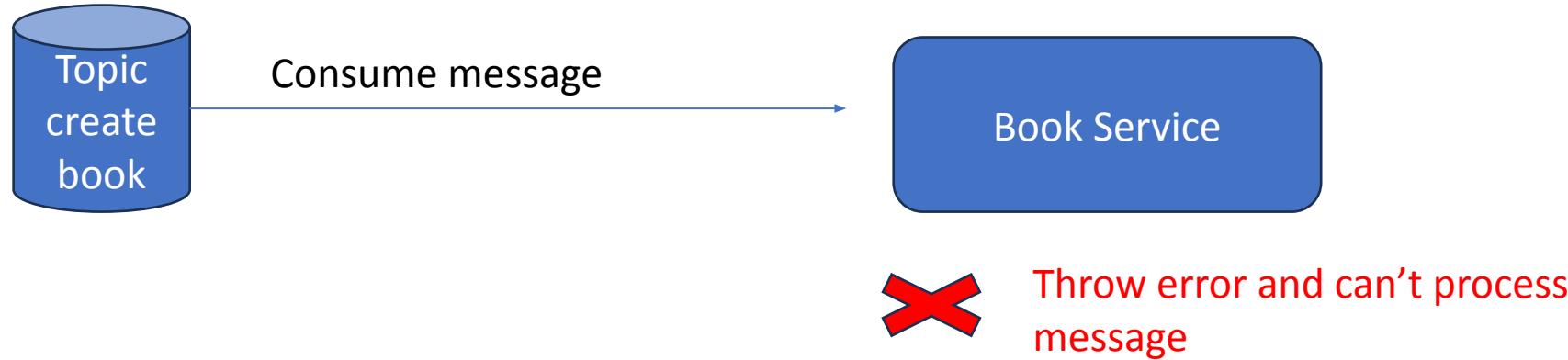


Khuyết điểm của Kafka

- 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:

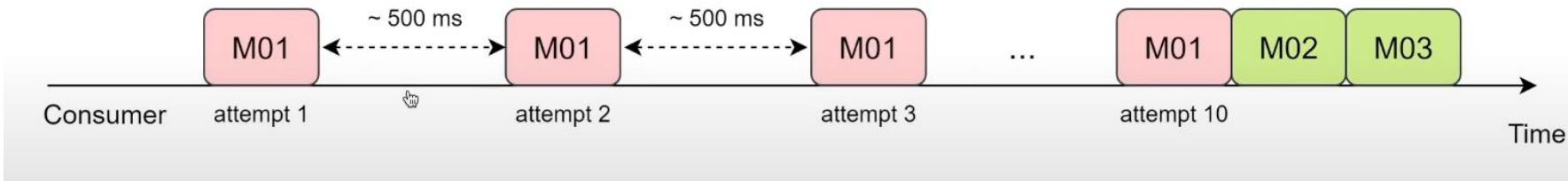
- Lỗi có thể retry
- Lỗi Không thể retry





Kafka – Error Handling

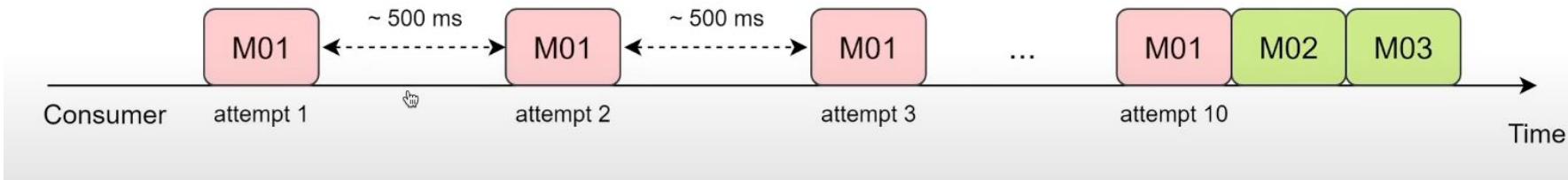
Giải pháp 1: Cơ chế Retry





Kafka – Error Handling

Cơ chế Simple Retry



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

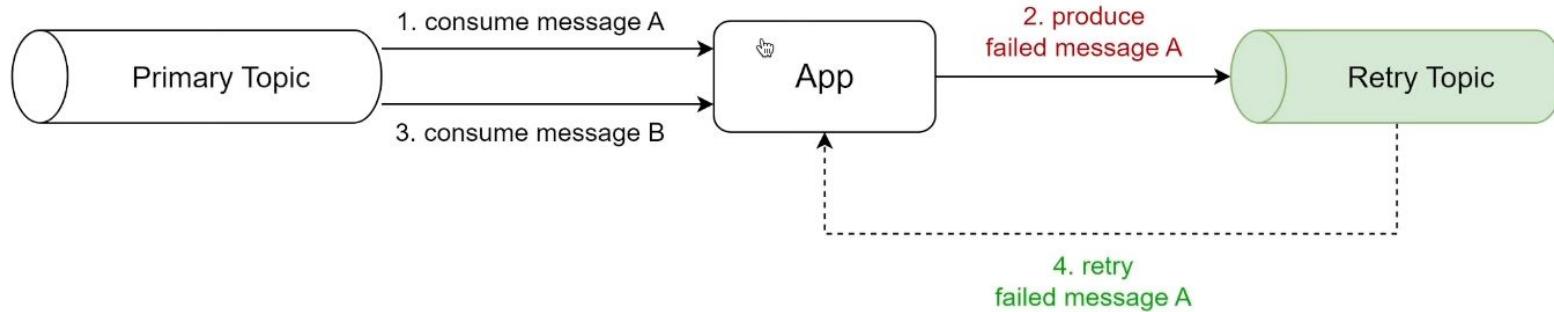
Nhược điểm

- 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

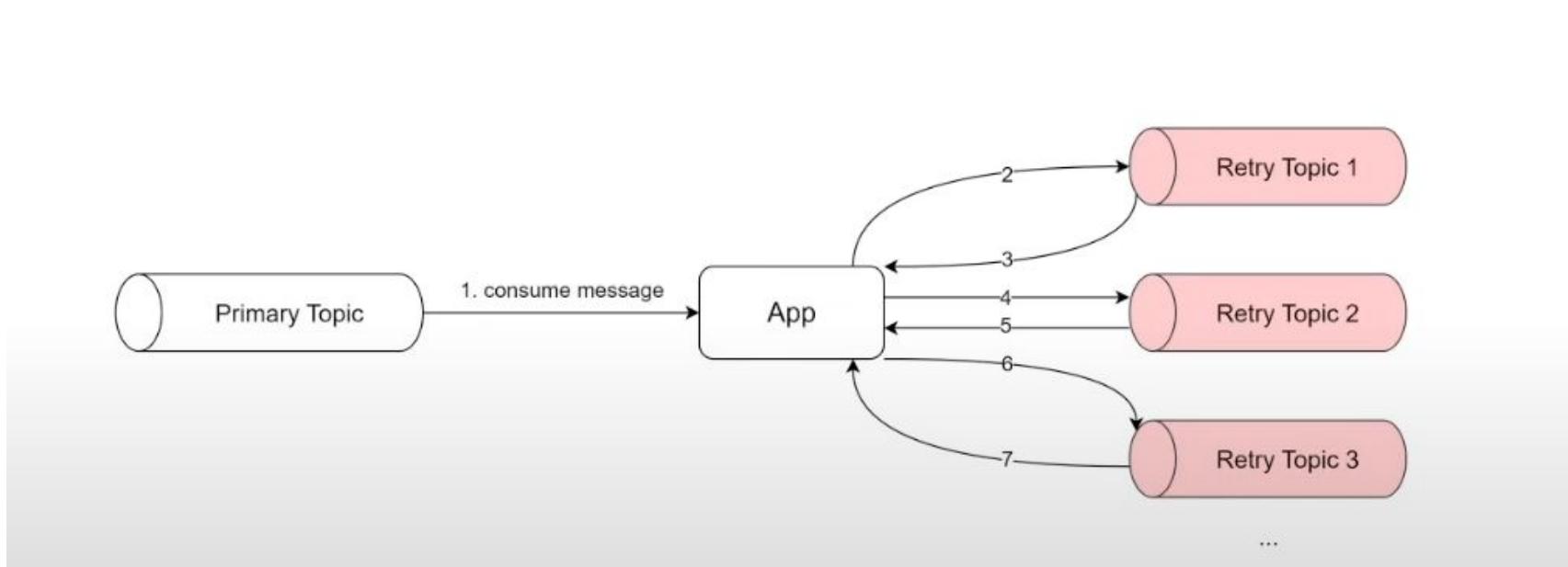


- Consumer sẽ không bị block
- Dễ trace được path của message lỗi



Kafka – Error Handling

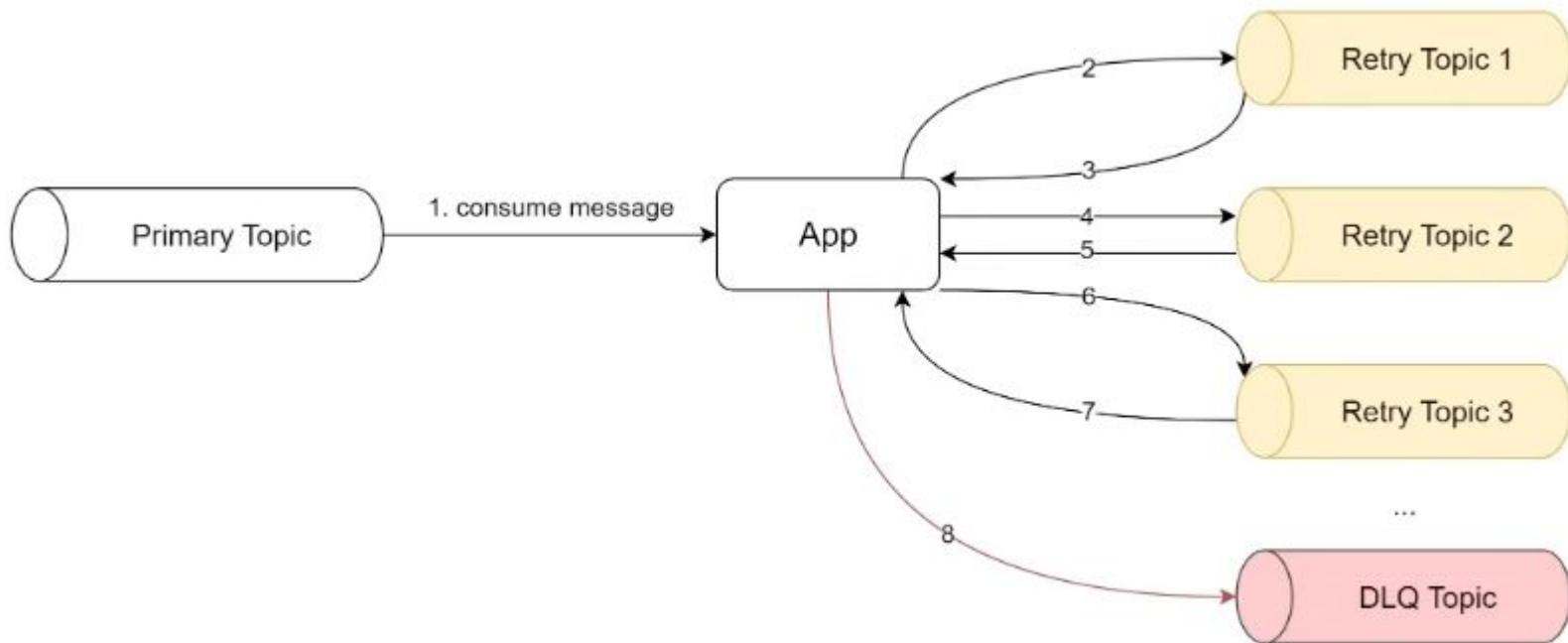
Nếu như vẫn Fail khi retry thì sao ?





Kafka – Error Handling

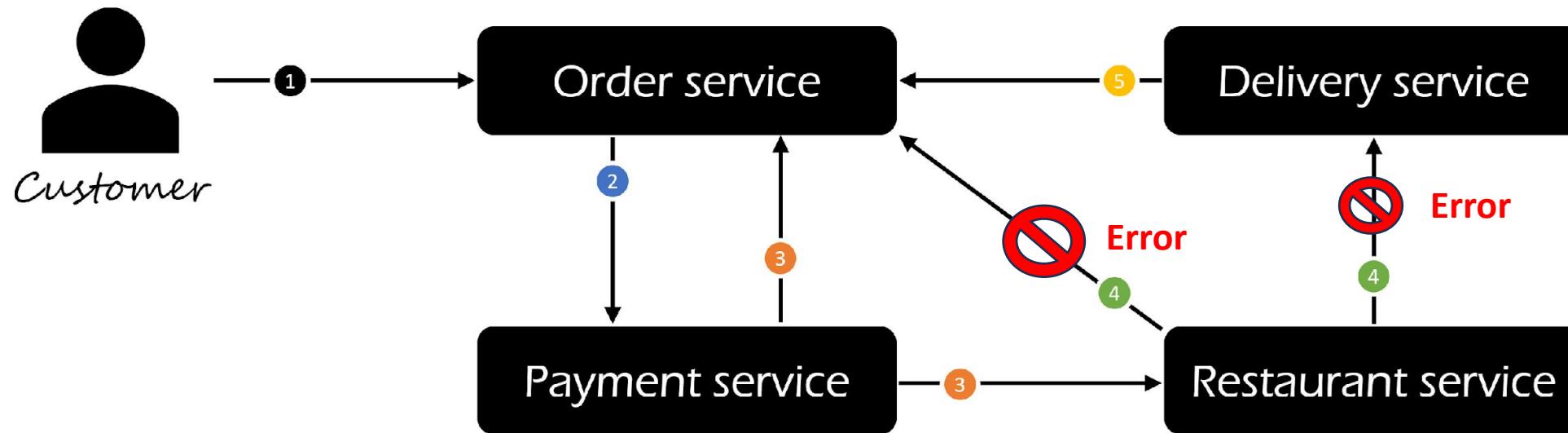
Giải pháp: Dead Letter Queue (DLQ)





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

ALL or NOTHING



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 ...;
```

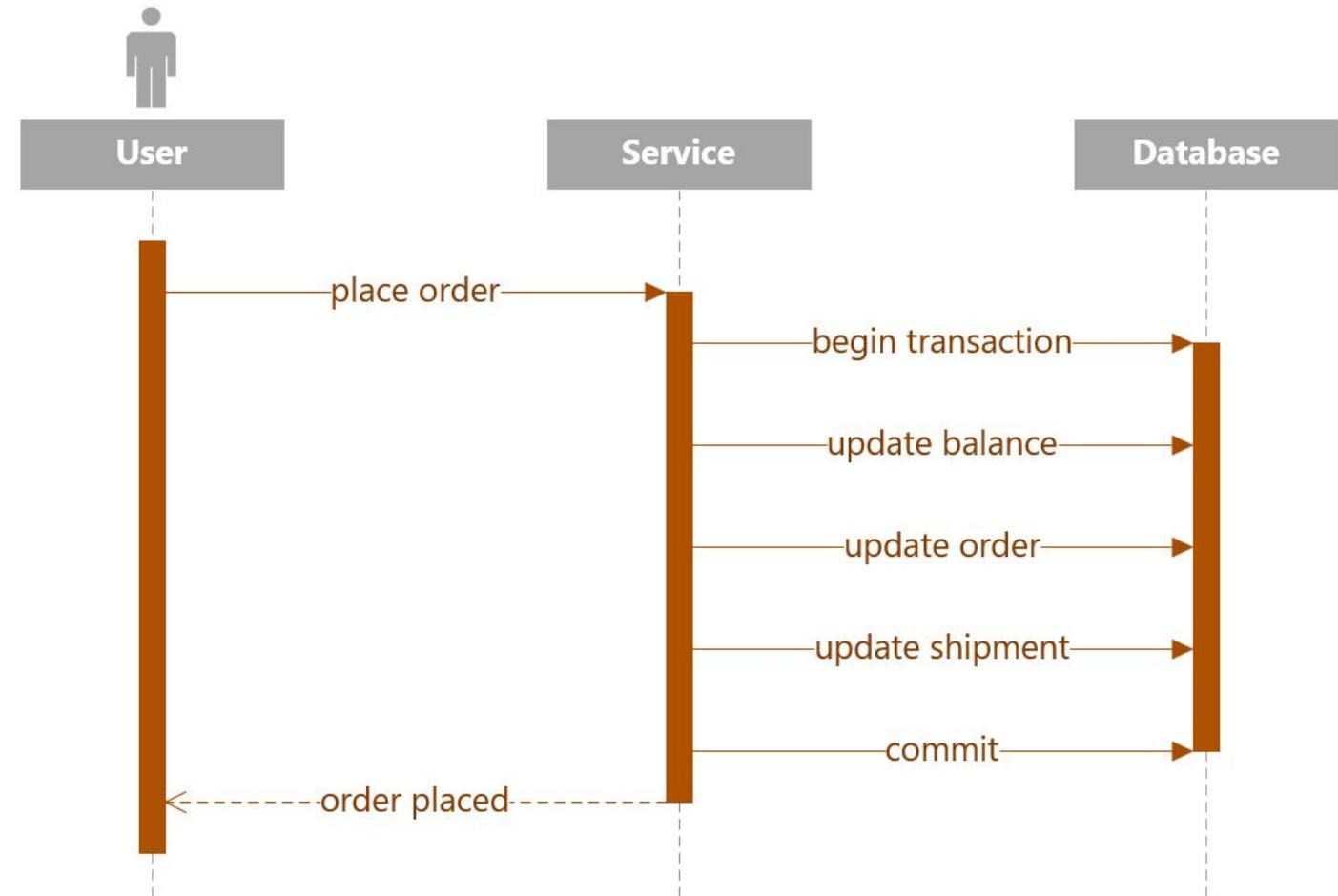
```
COMMIT;
```



Distributed Transaction

Monolithic Architecture

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

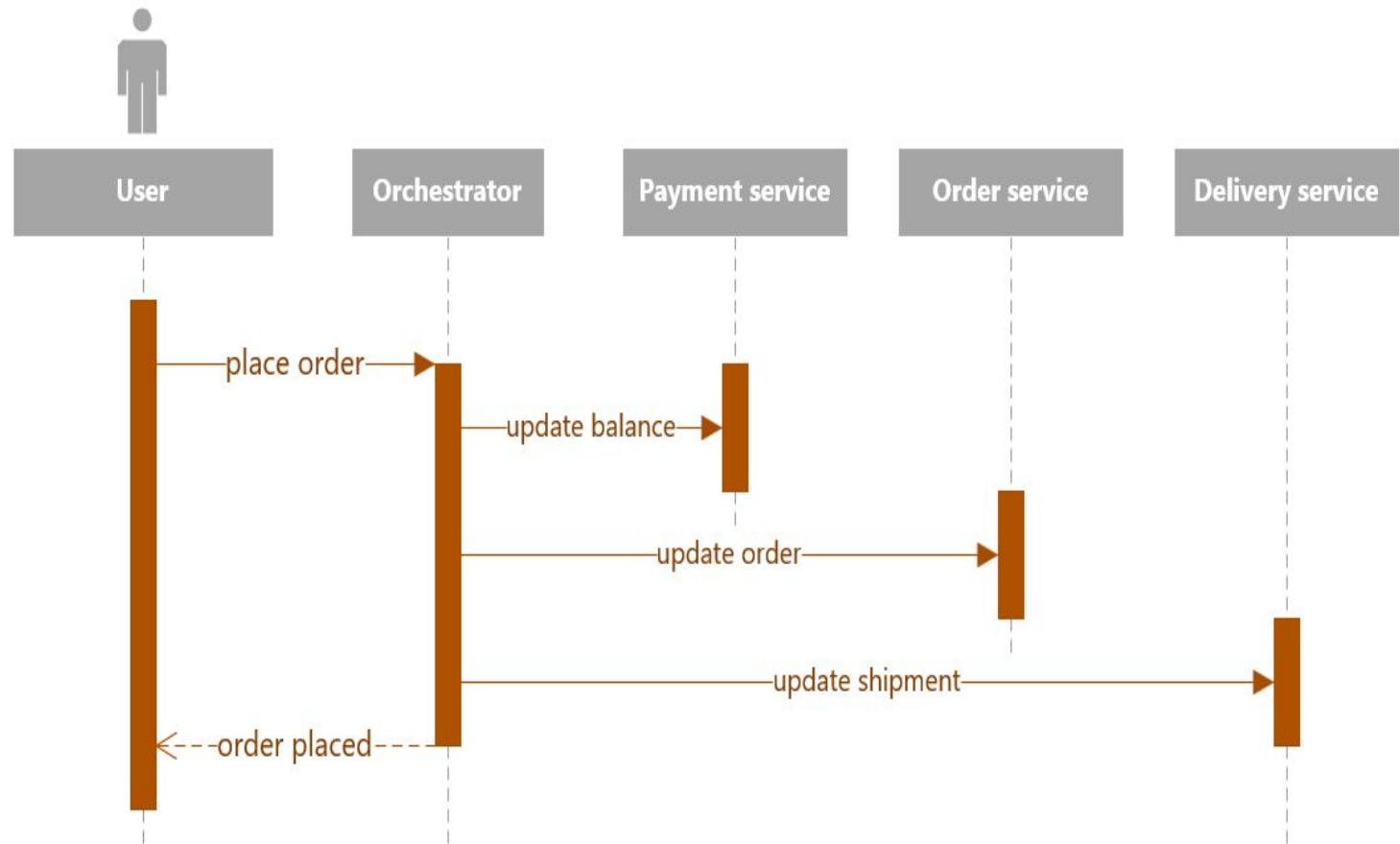




Distributed Transaction

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.





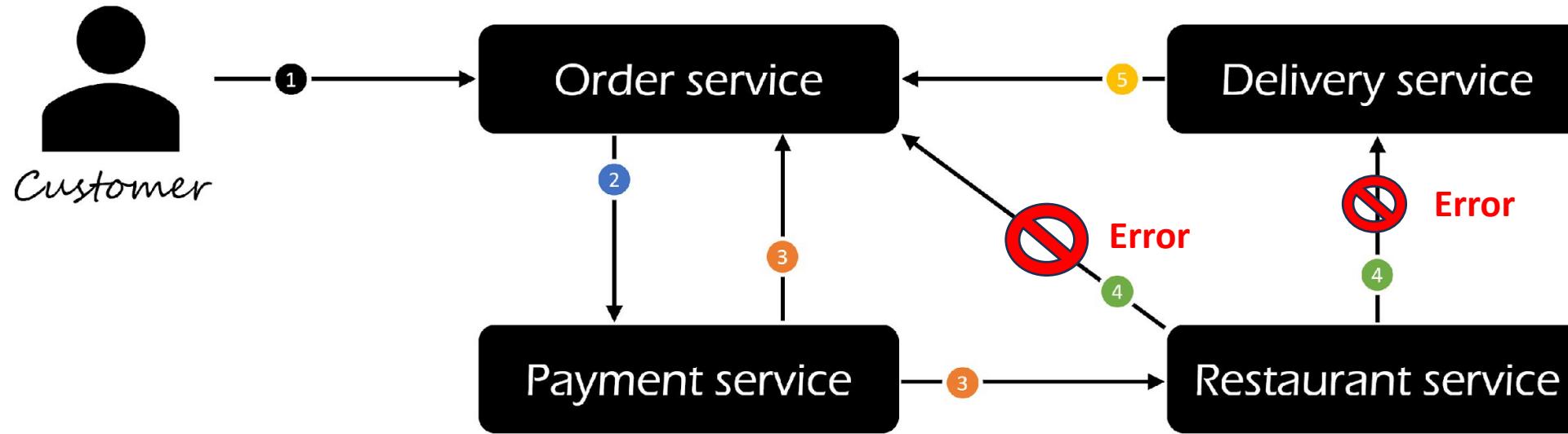
Distributed Transaction

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.



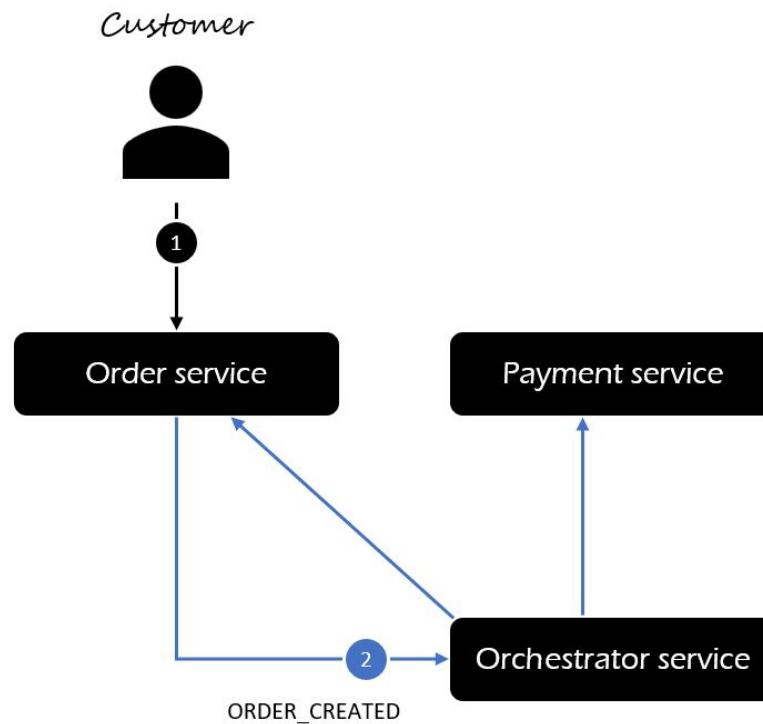
Distributed Transaction – SAGA Pattern

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.

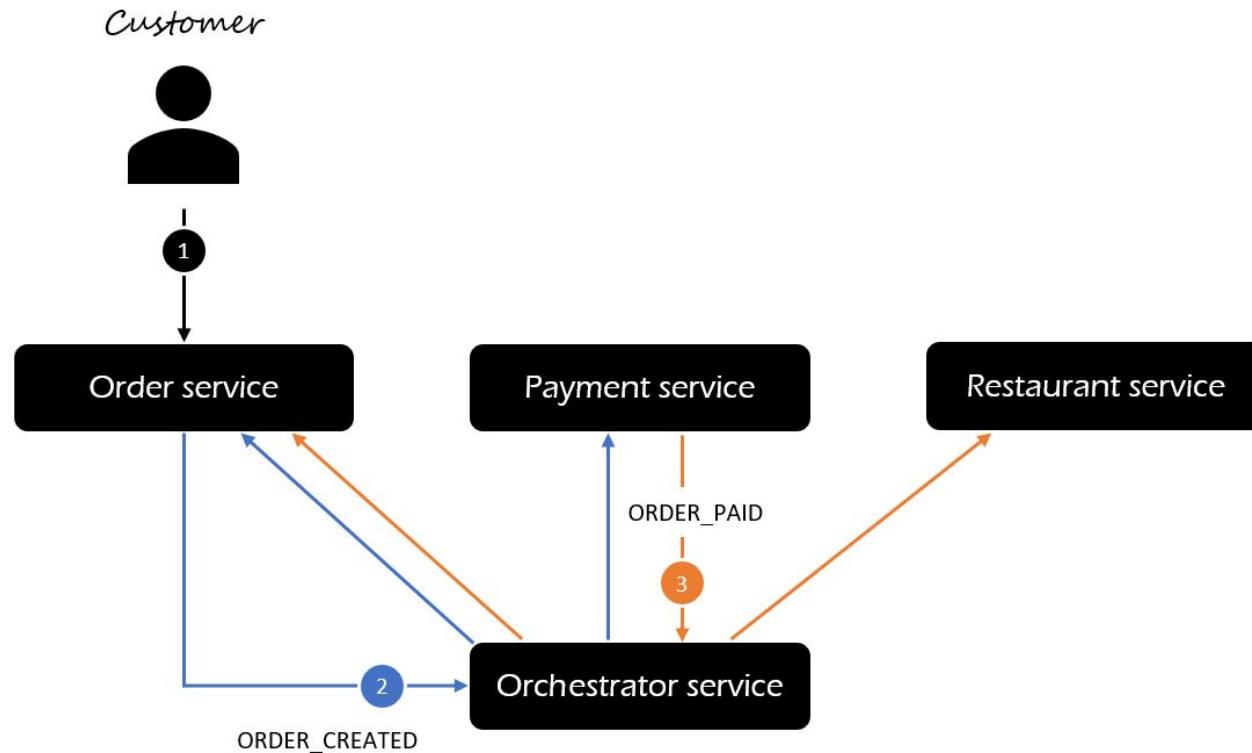


Distributed Transaction – SAGA Pattern



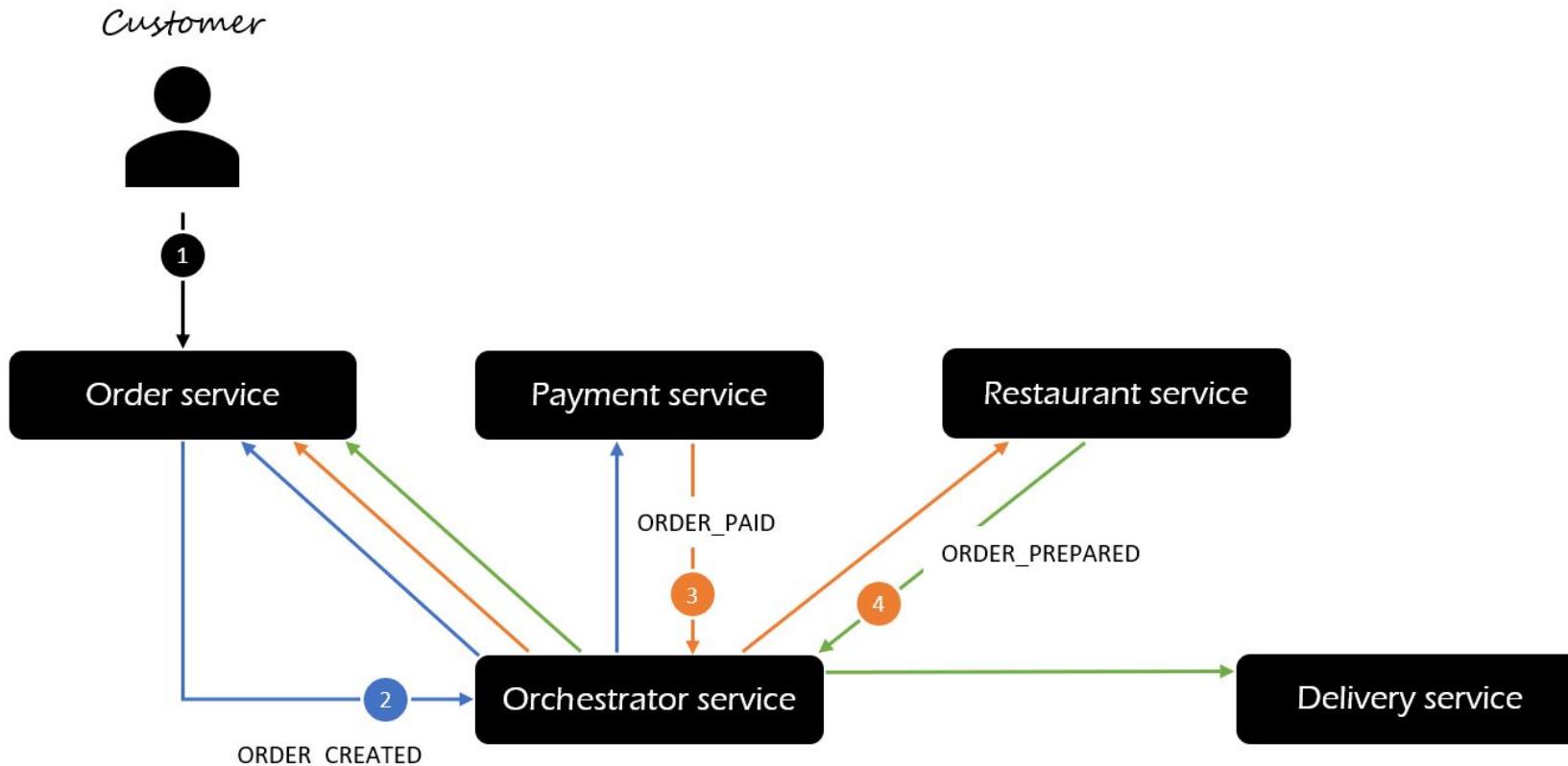


Distributed Transaction – SAGA Pattern

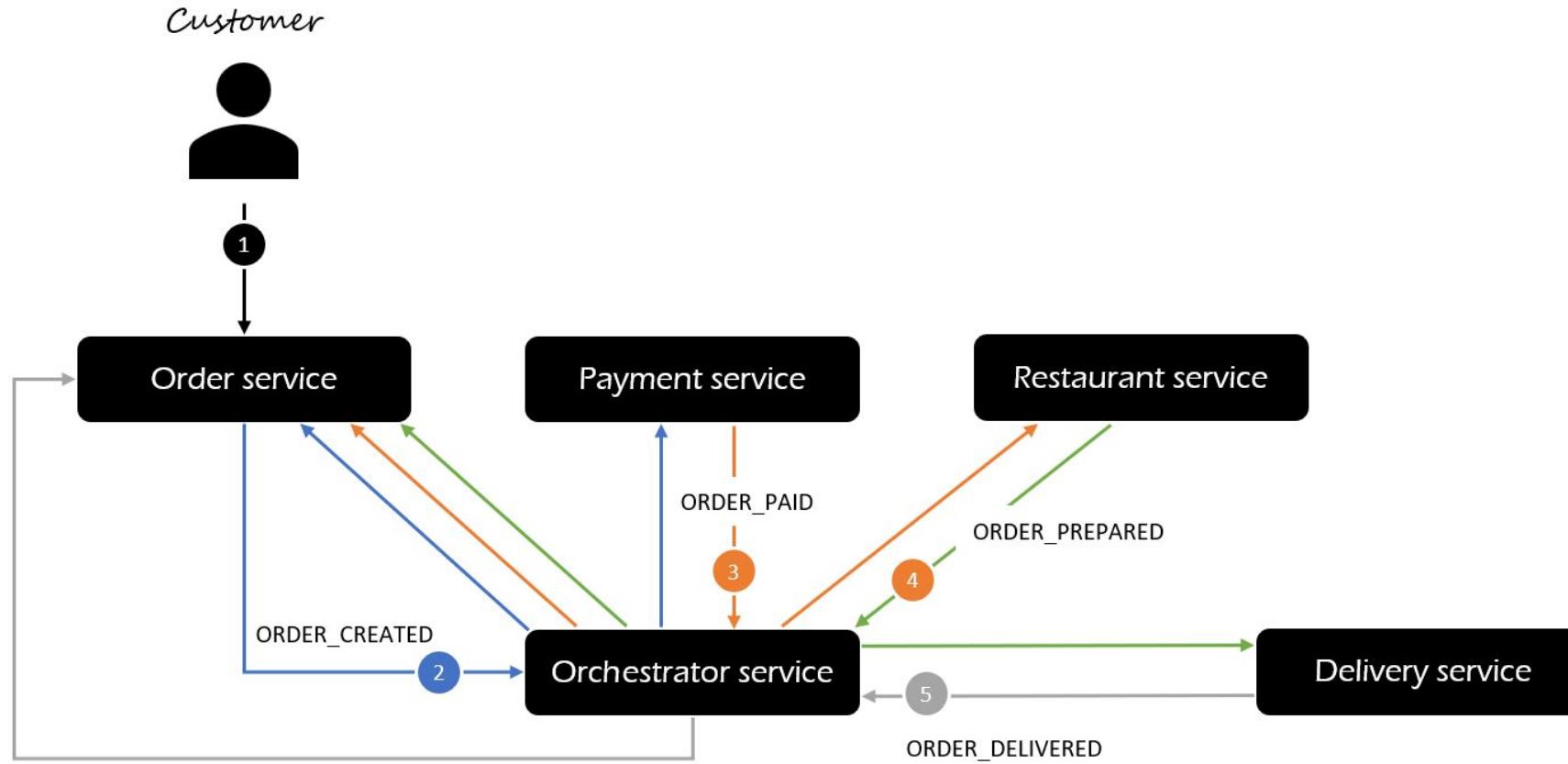




Distributed Transaction – SAGA Pattern



Distributed Transaction – SAGA Pattern





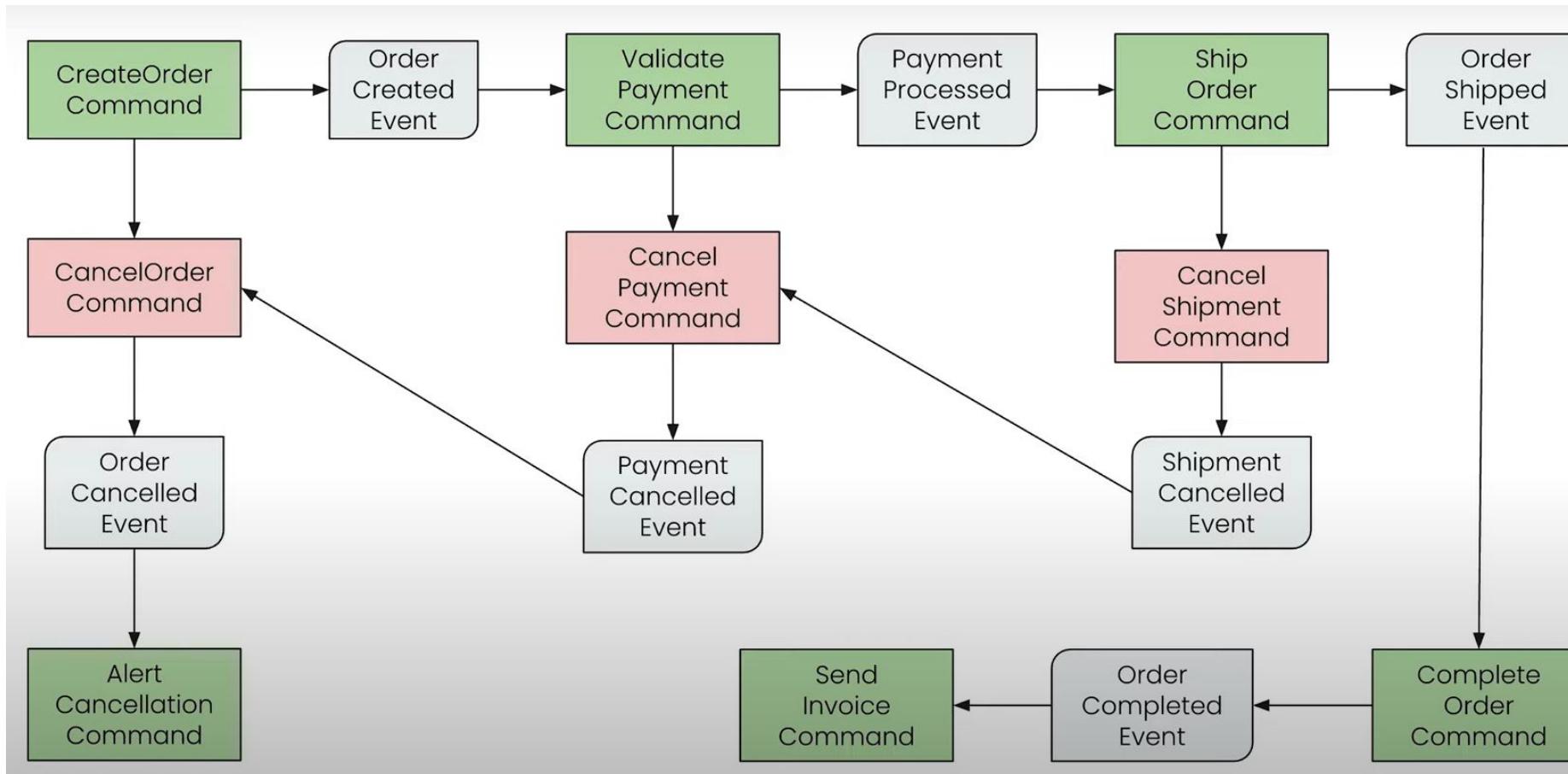
Exception handling

- **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...



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 / Latency:



Docker là gì ?

Docker là một nền tảng
containerization





Containerization là gì ?

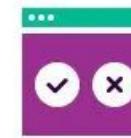
Environments



DEVELOPMENT

Write code

Developers



TESTING

Verify code

QA



STAGING

Test infrastructure

Product & Stakeholders



PRODUCTION

Deploy live

End Users



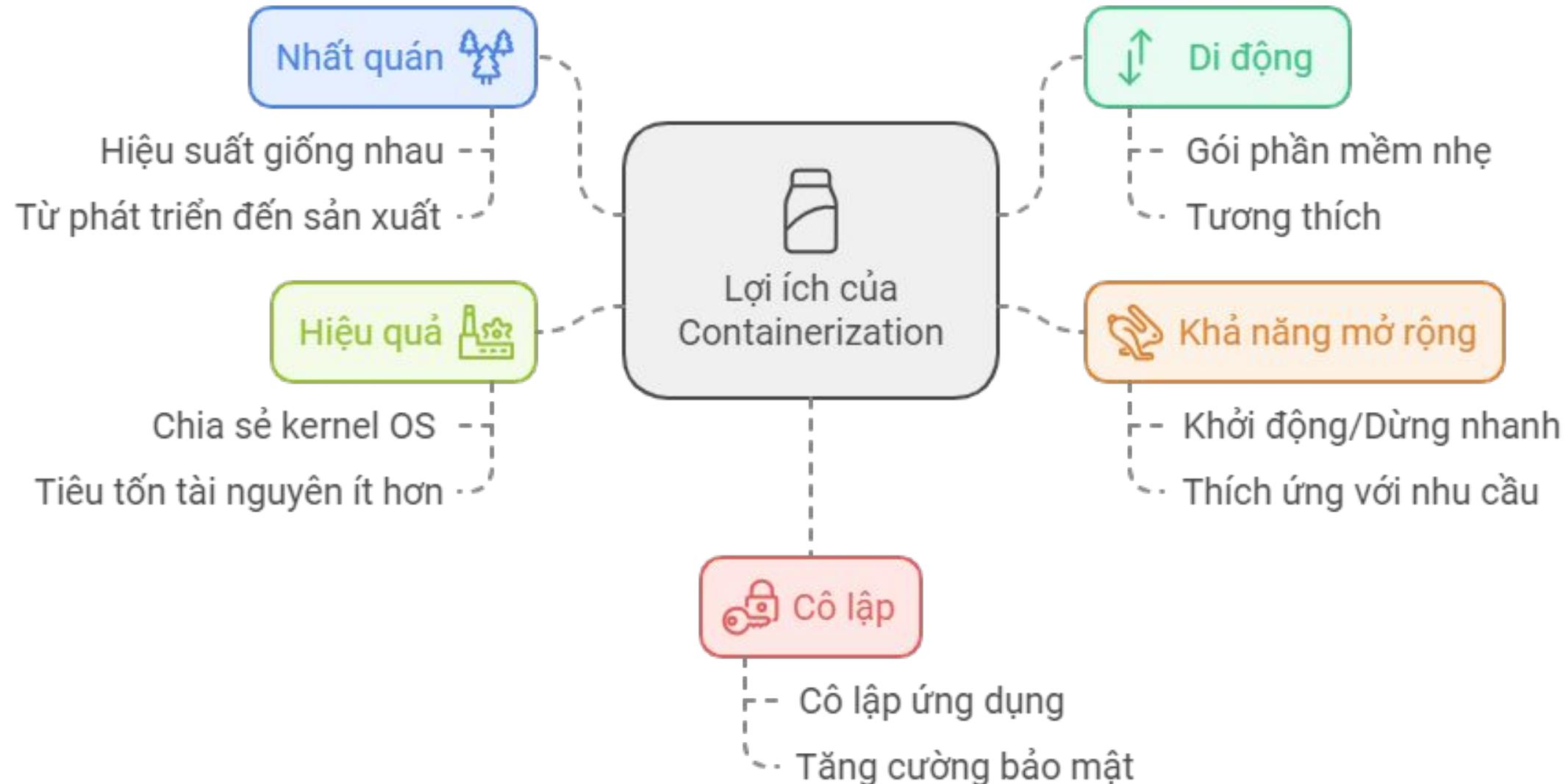
Containerization là gì ?

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)





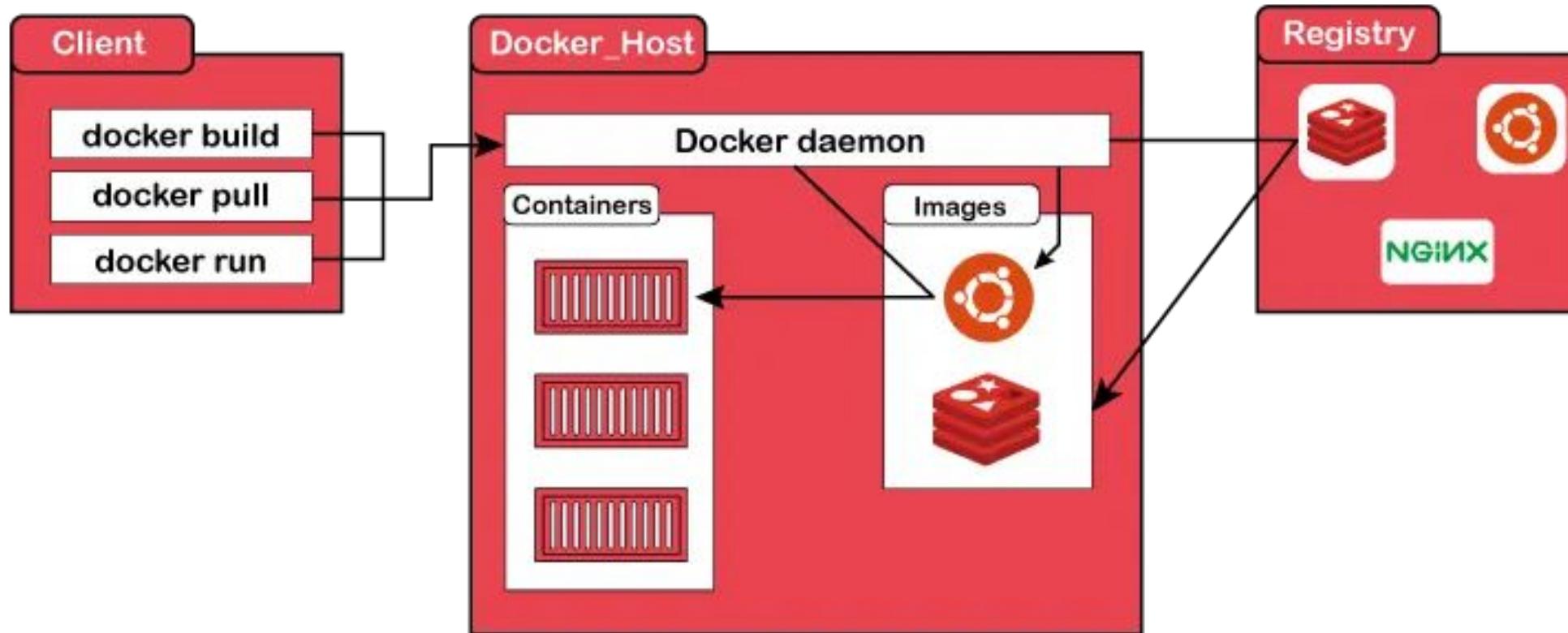
Vai trò của containerization ?





Các thành phần chính của Docker ?

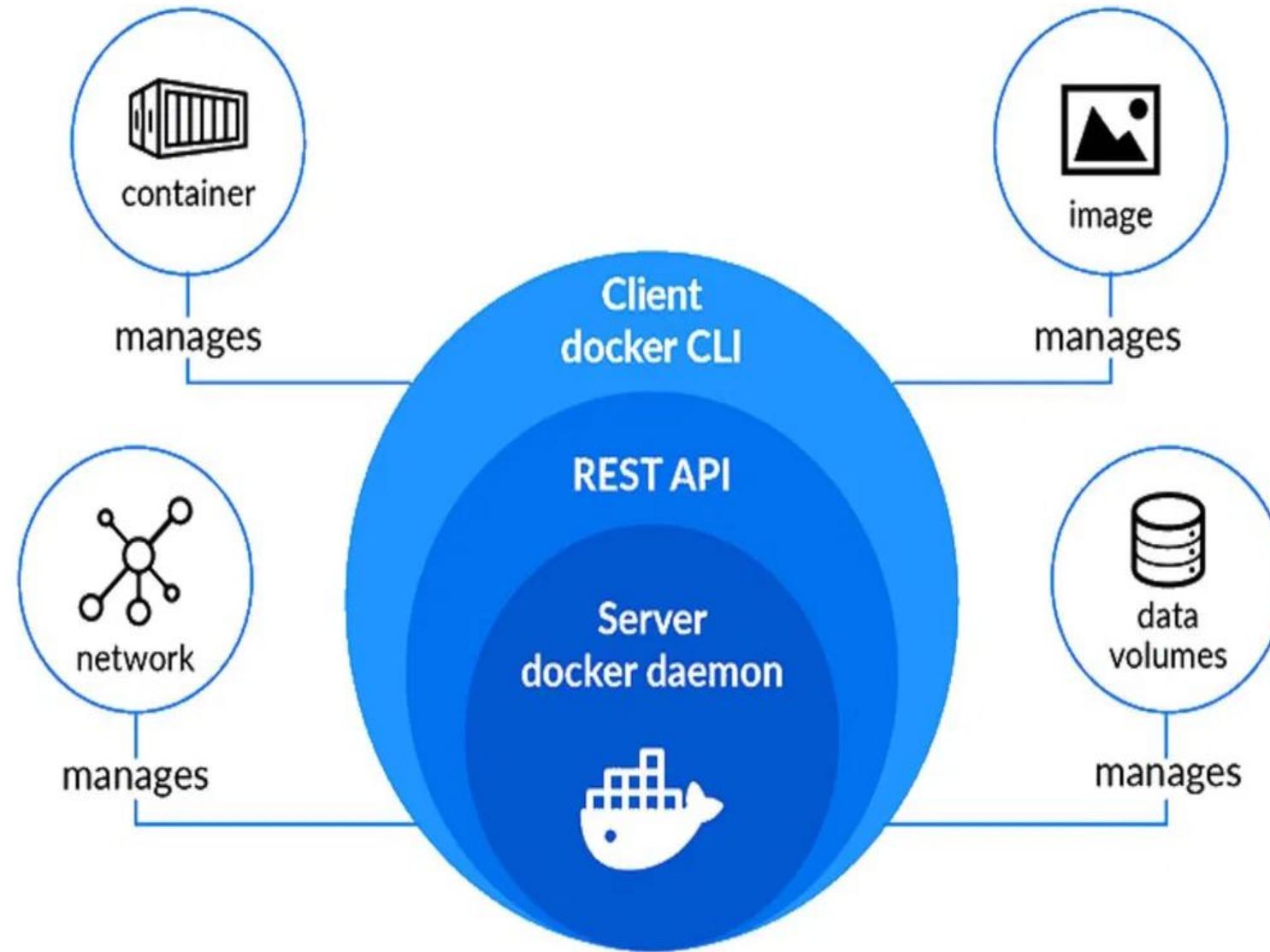
Để hiểu rõ cách thức hoạt động của Docker, bạn cần làm quen với các thành phần chính của nó





Docker Engine

- Docker CLI
- Docker API
- Docker Daemon





Docker Image

Quá trình chuyển đổi Docker Image thành Container

Tạo Docker
Image



Image trở nên
không thay đổi



Chạy Image
như Container

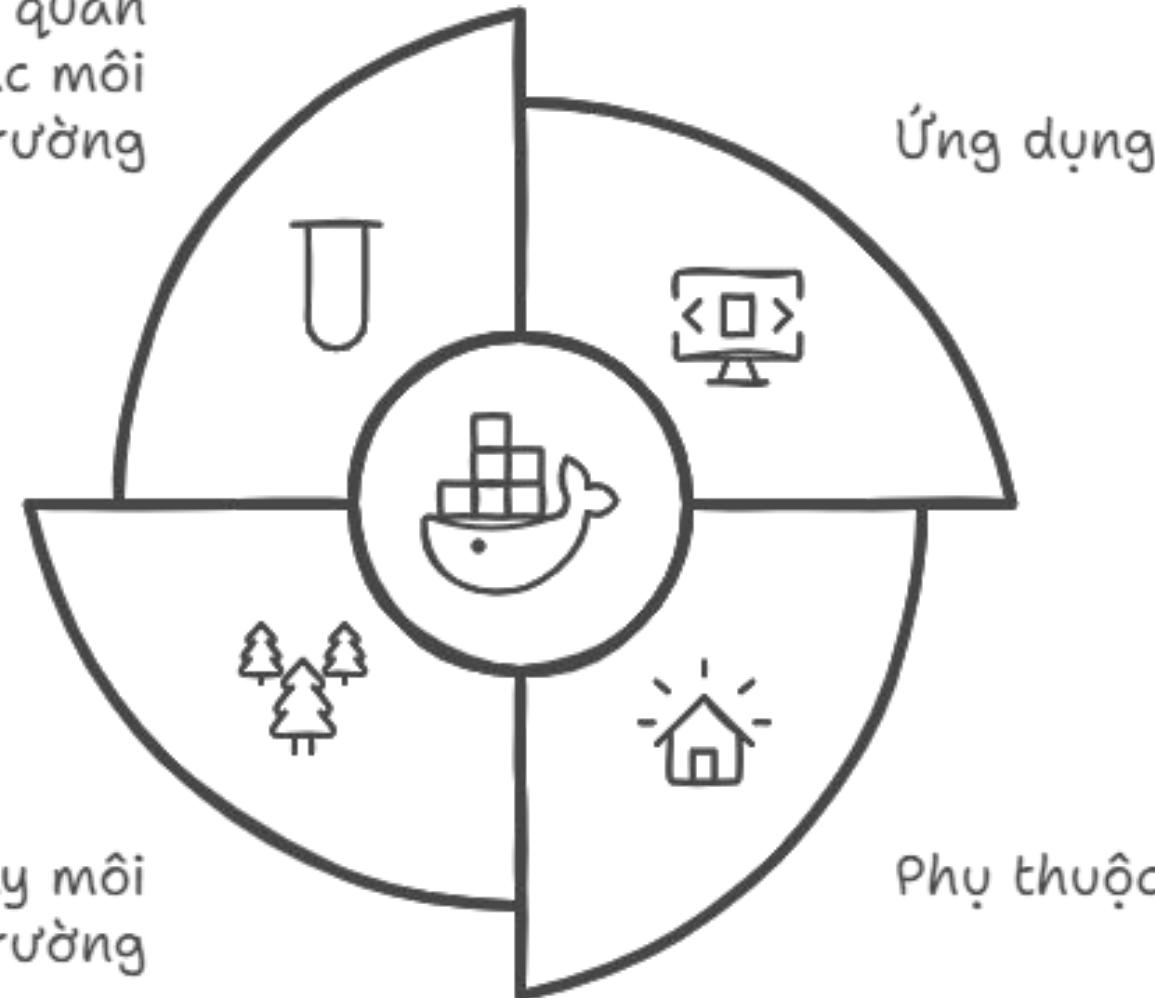




Docker Containers

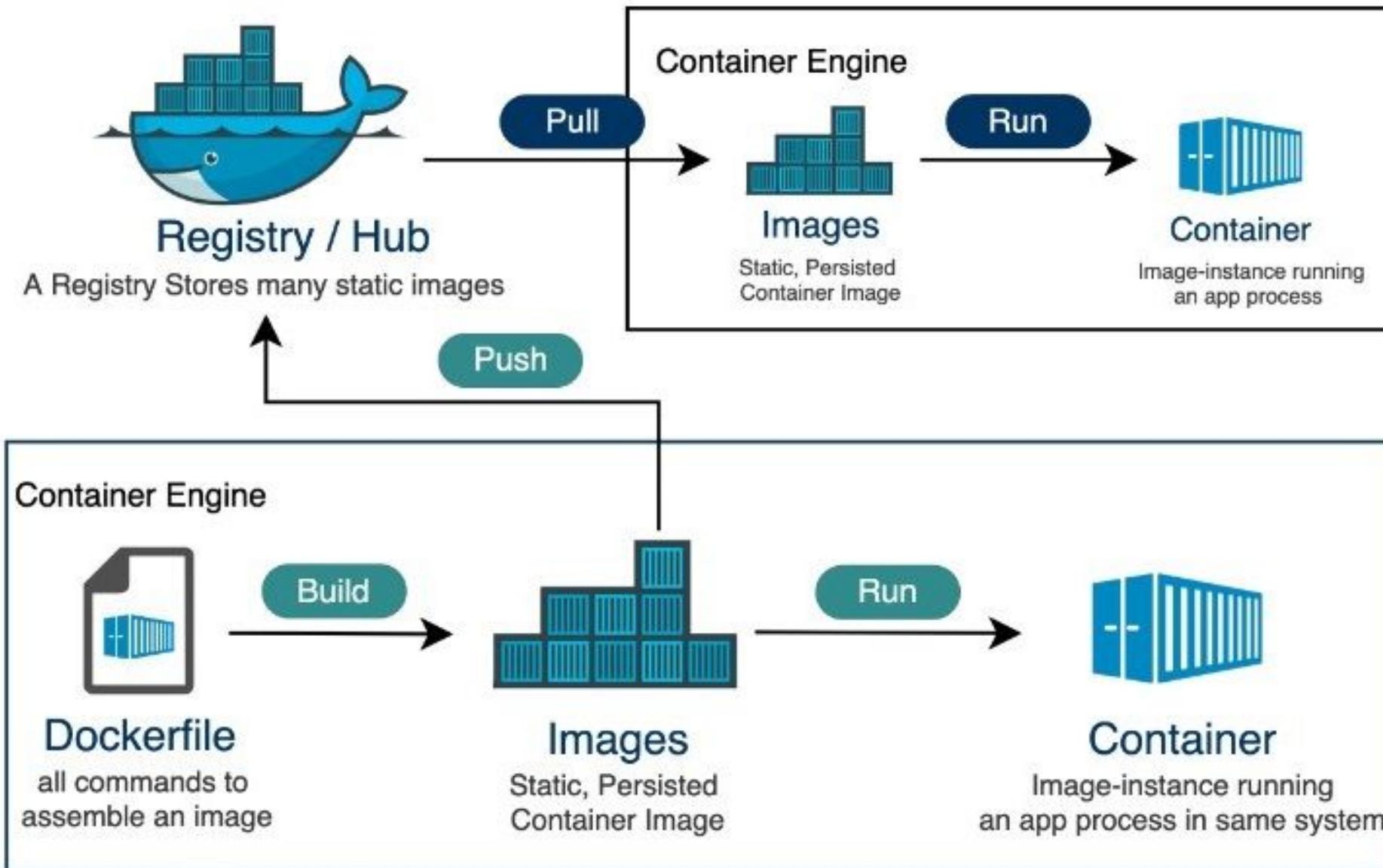
Hiểu về Docker Containers

Tính nhất quán
giữa các môi
trường





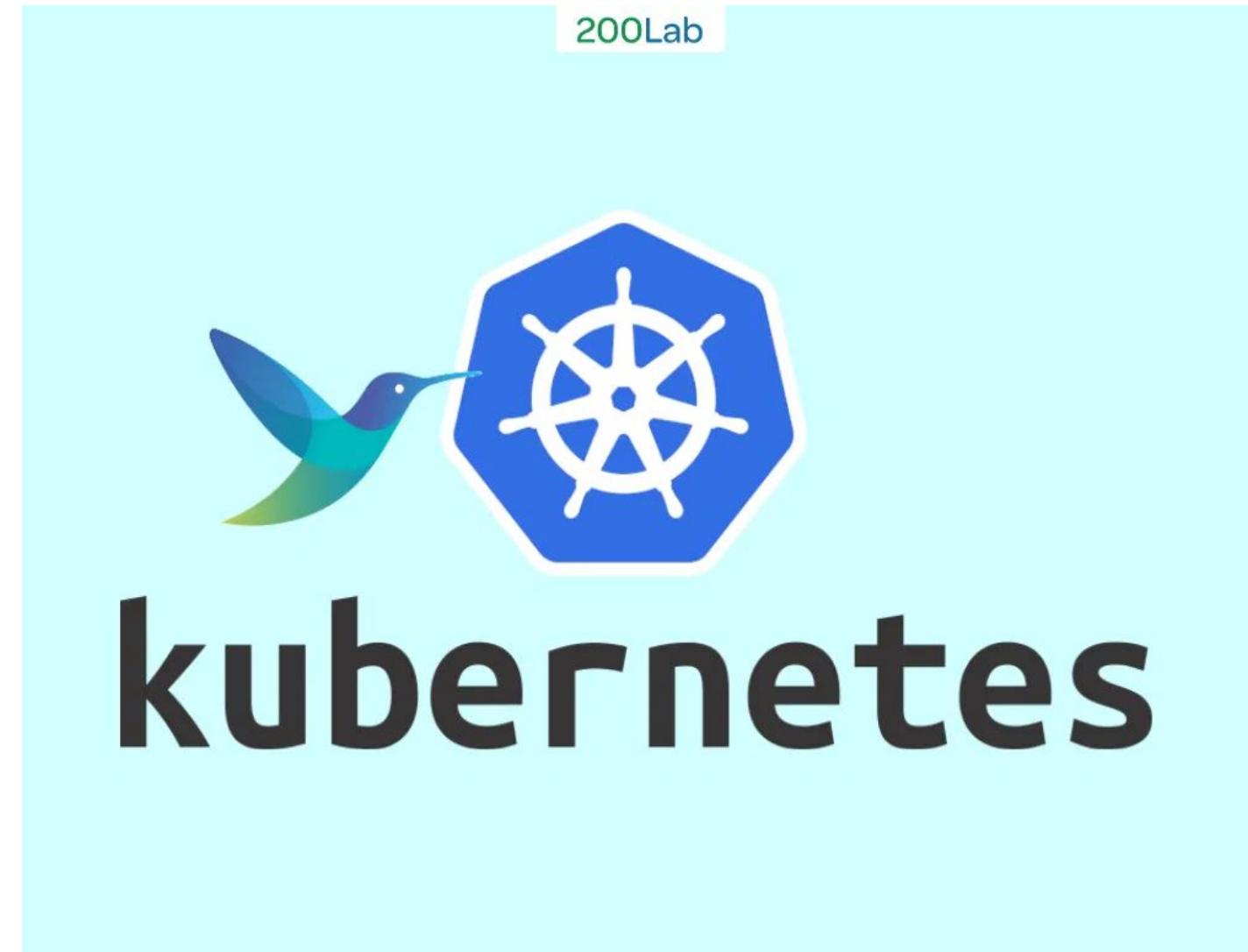
Docker Hub





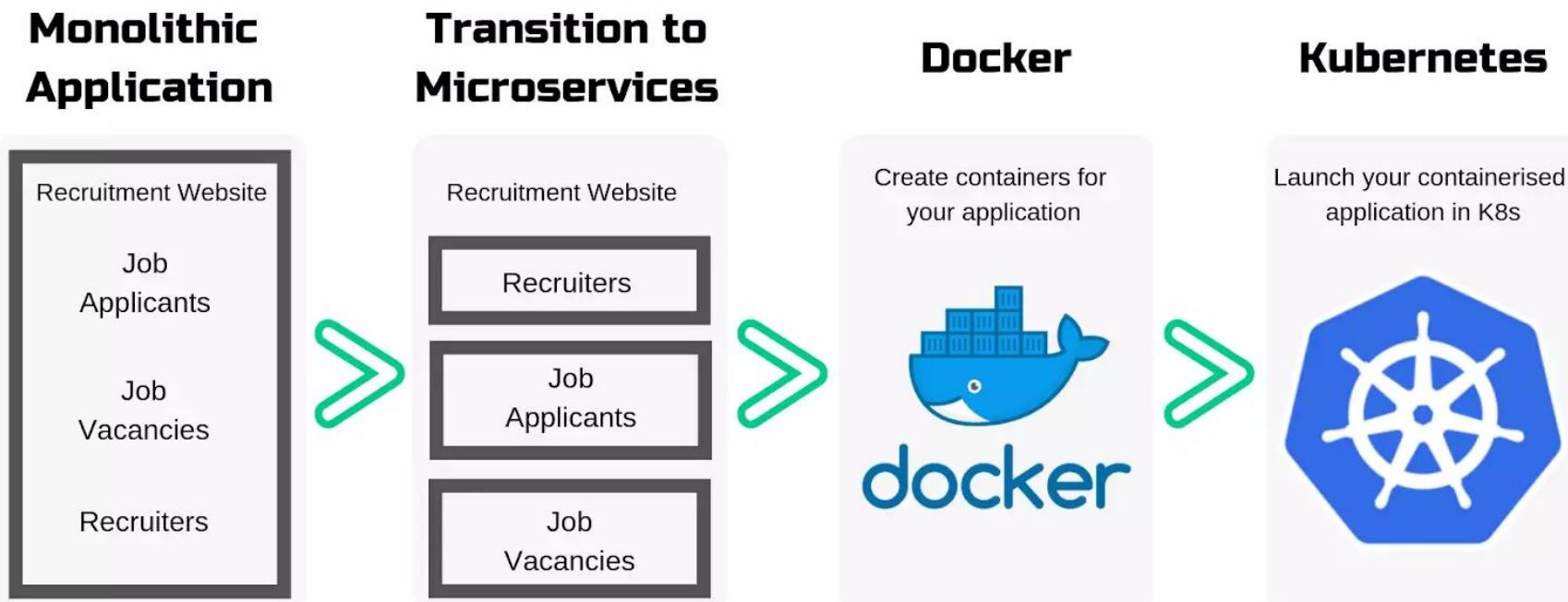
Kubernetes Là Gì ?

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



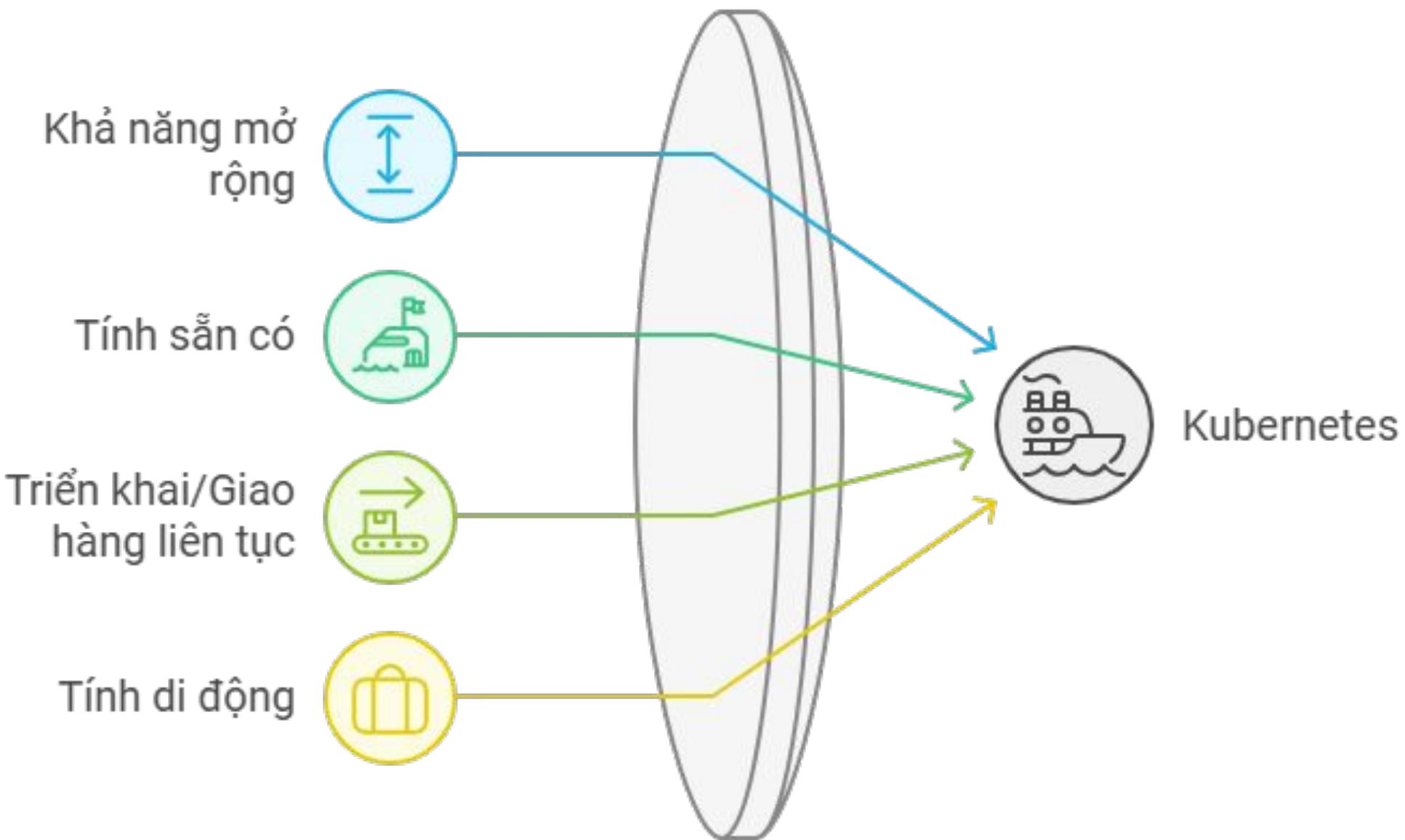


Nên dùng Kubernetes khi nào



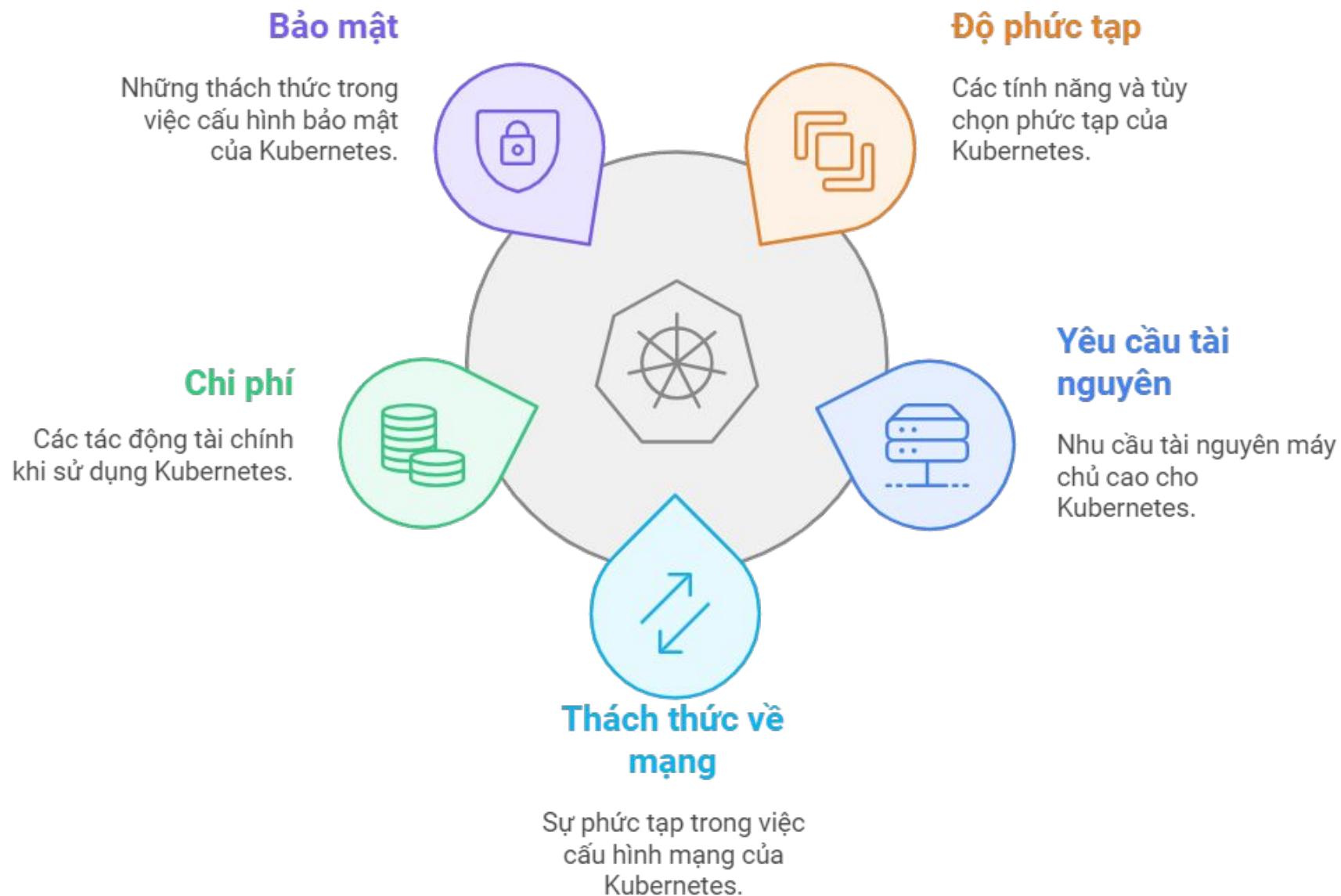


Lợi ích của Kubernetes



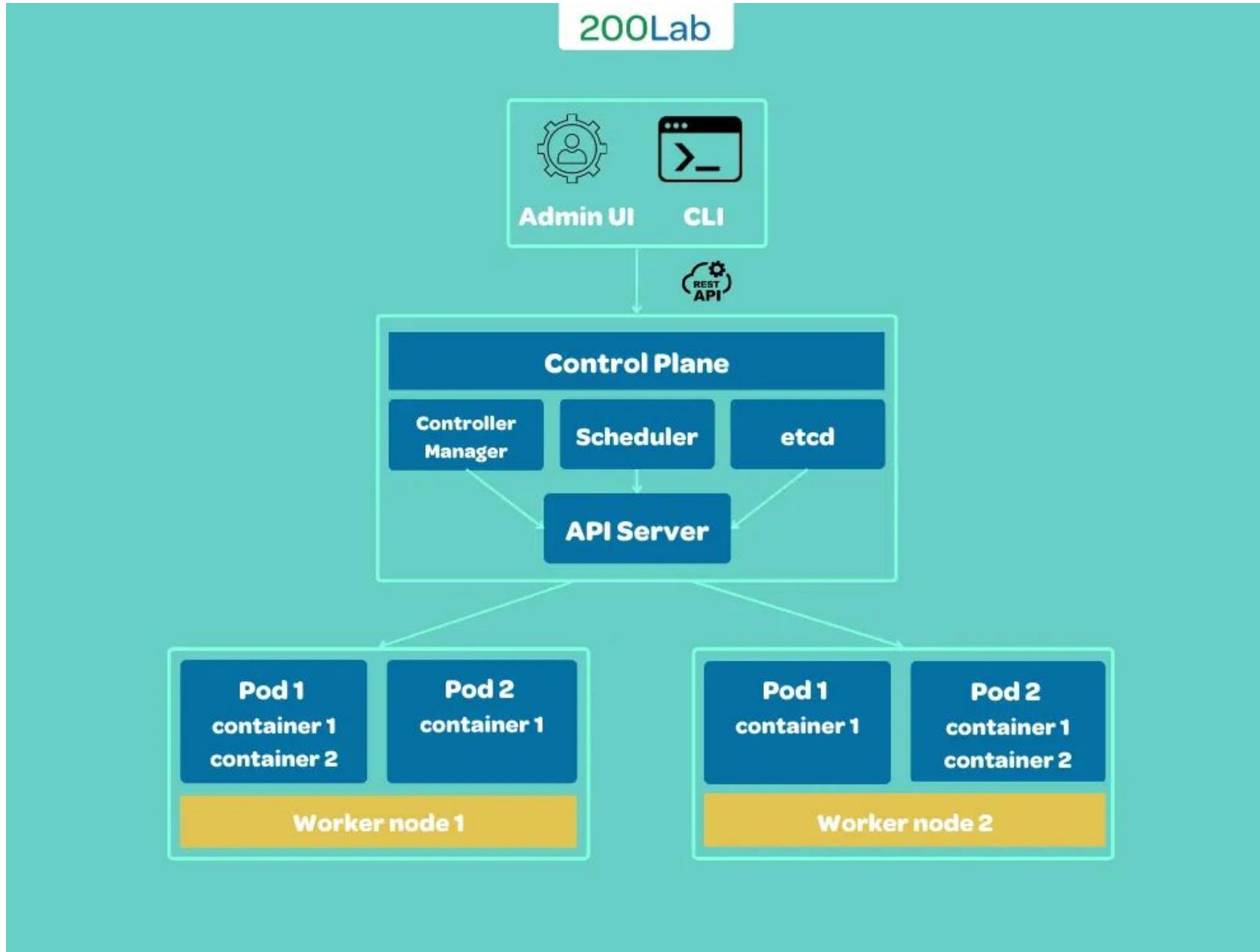


Thách thức trong việc triển khai Kubernetes





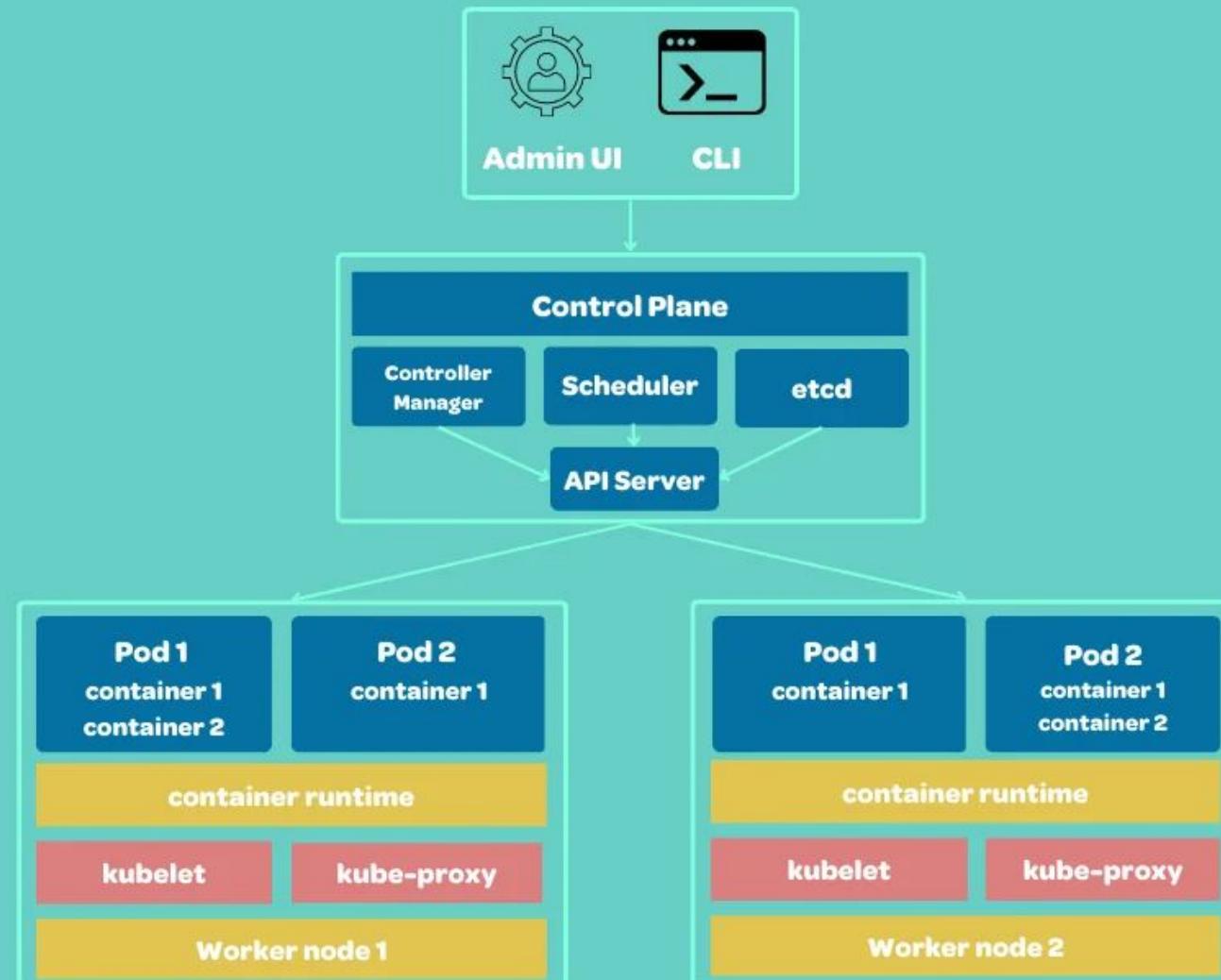
Kubernetes Architecture





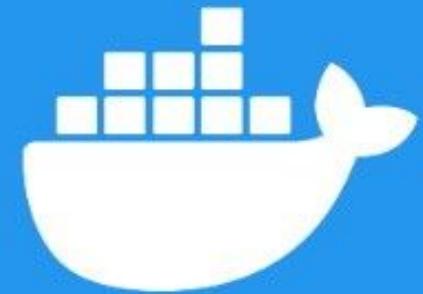
Kubernetes Architecture

200Lab





Kubernetes vs Docker



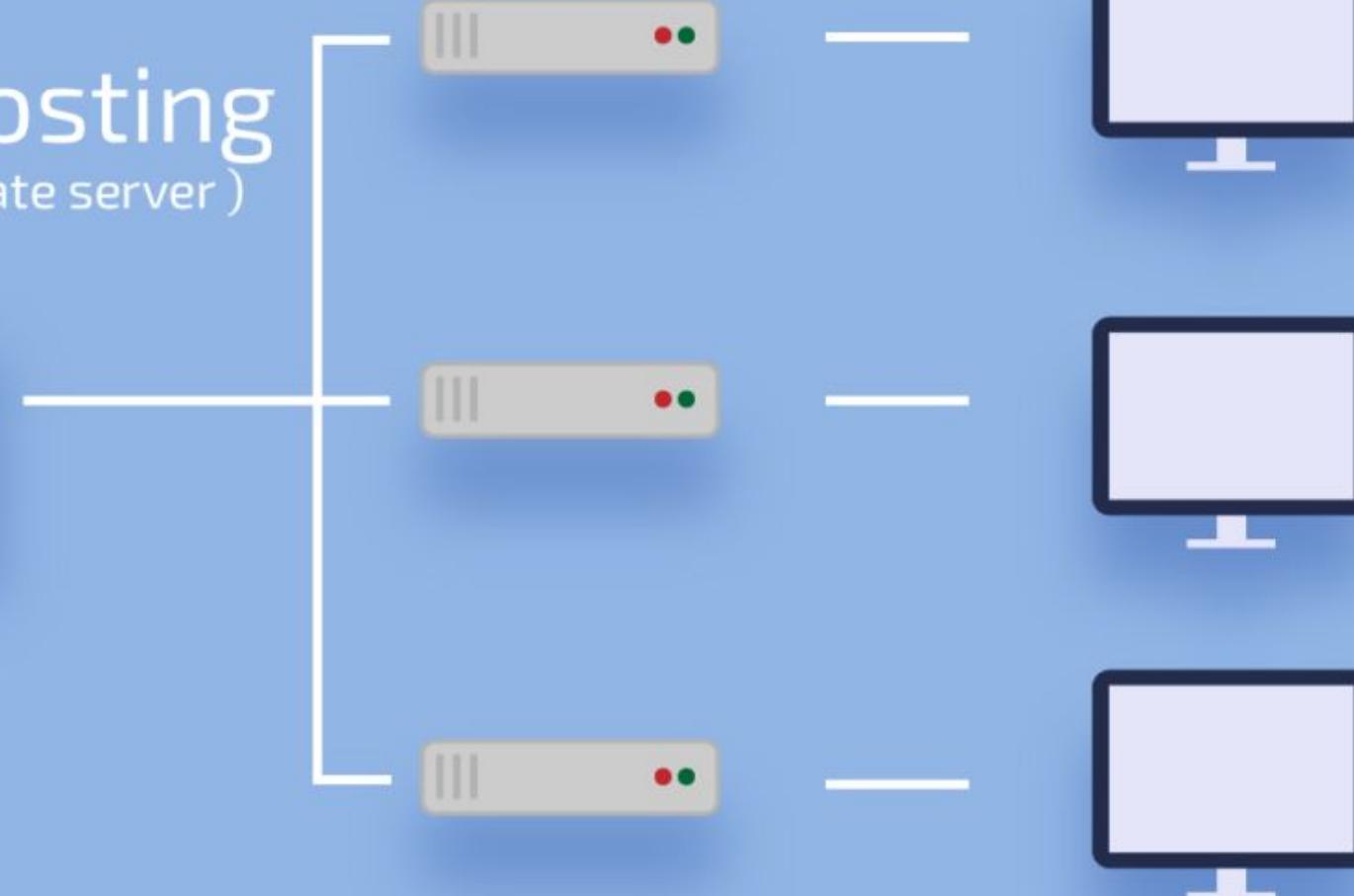
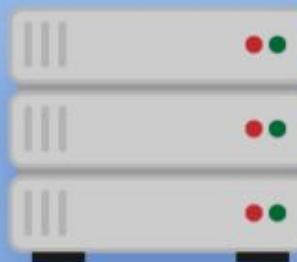
docker



kubernetes



VSP Hosting (virtual private server)





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.



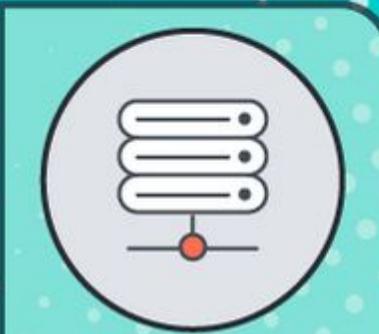
Ứng dụng đa dạng

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



Types of Web hosting

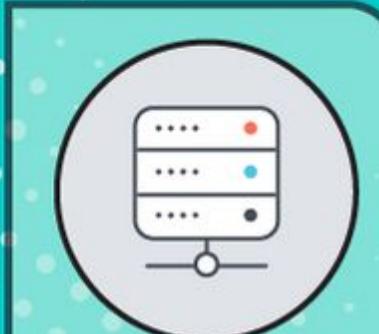
Types & Differences



Shared
Hosting



VPS

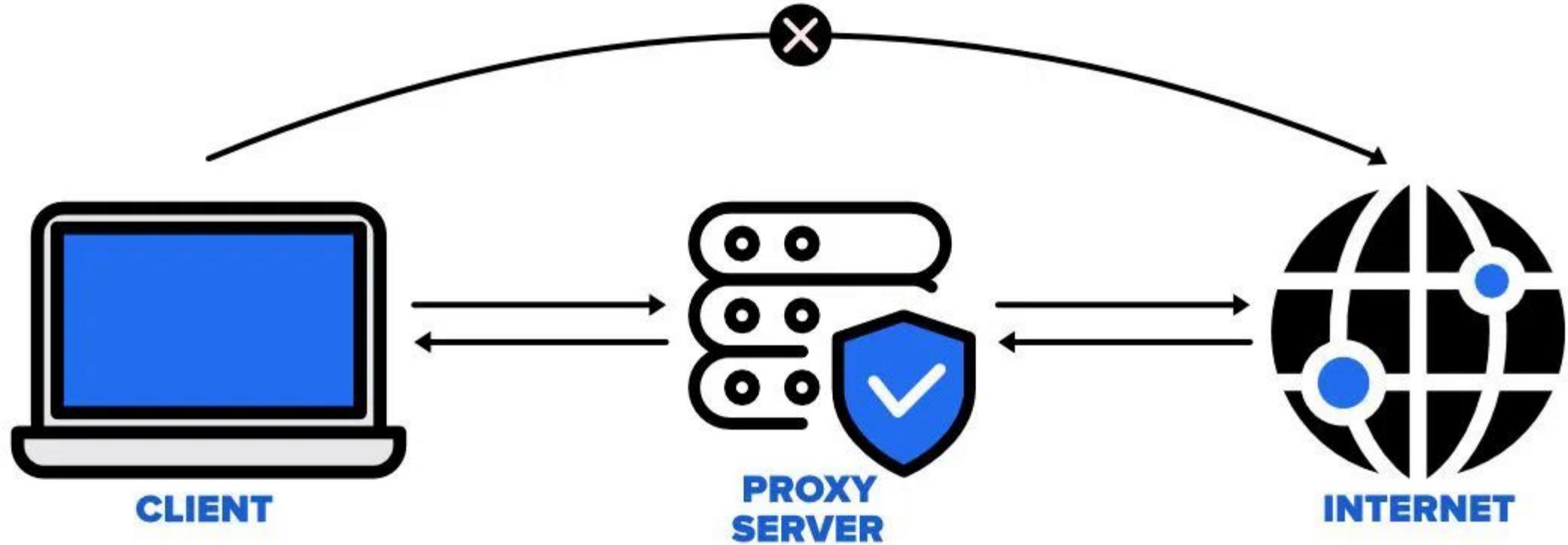


Dedicated
Server

OO Mhgoz



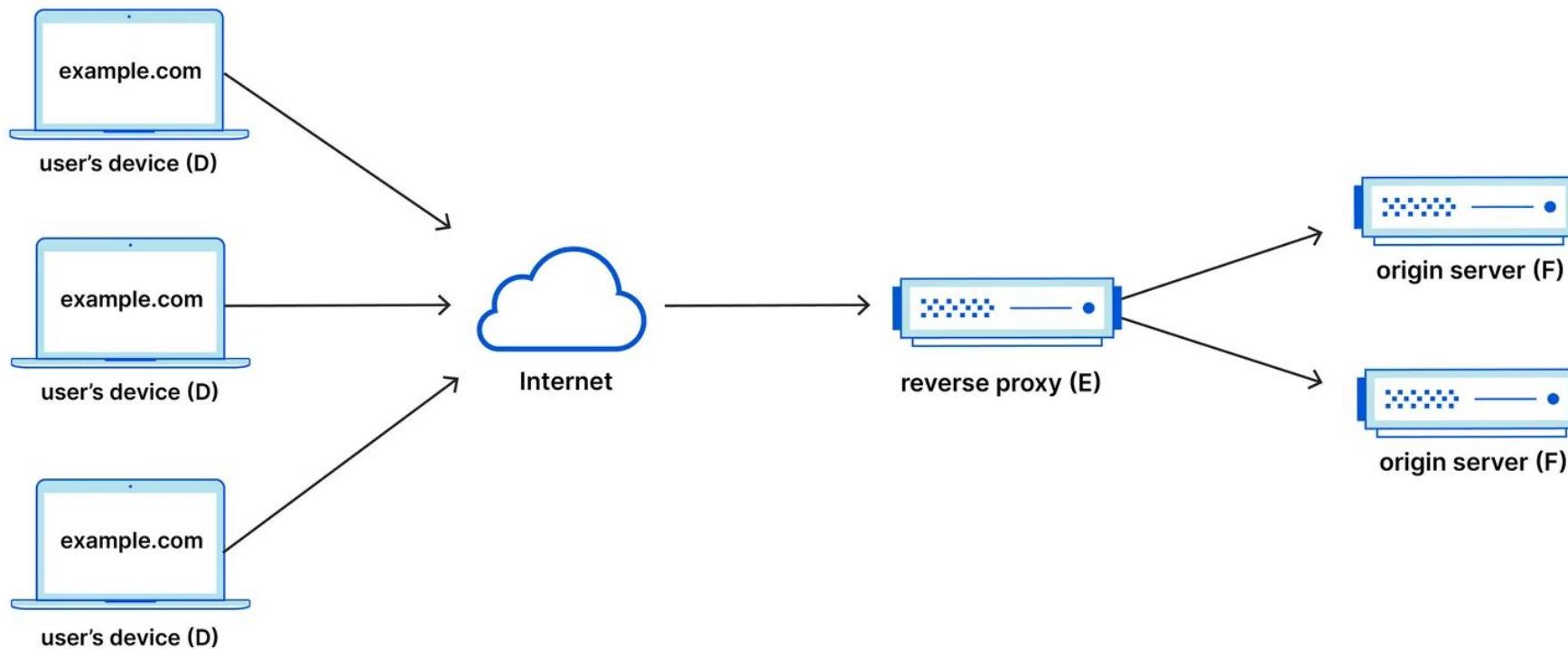
Proxy là gì ?





Reverse Proxy là gì ?

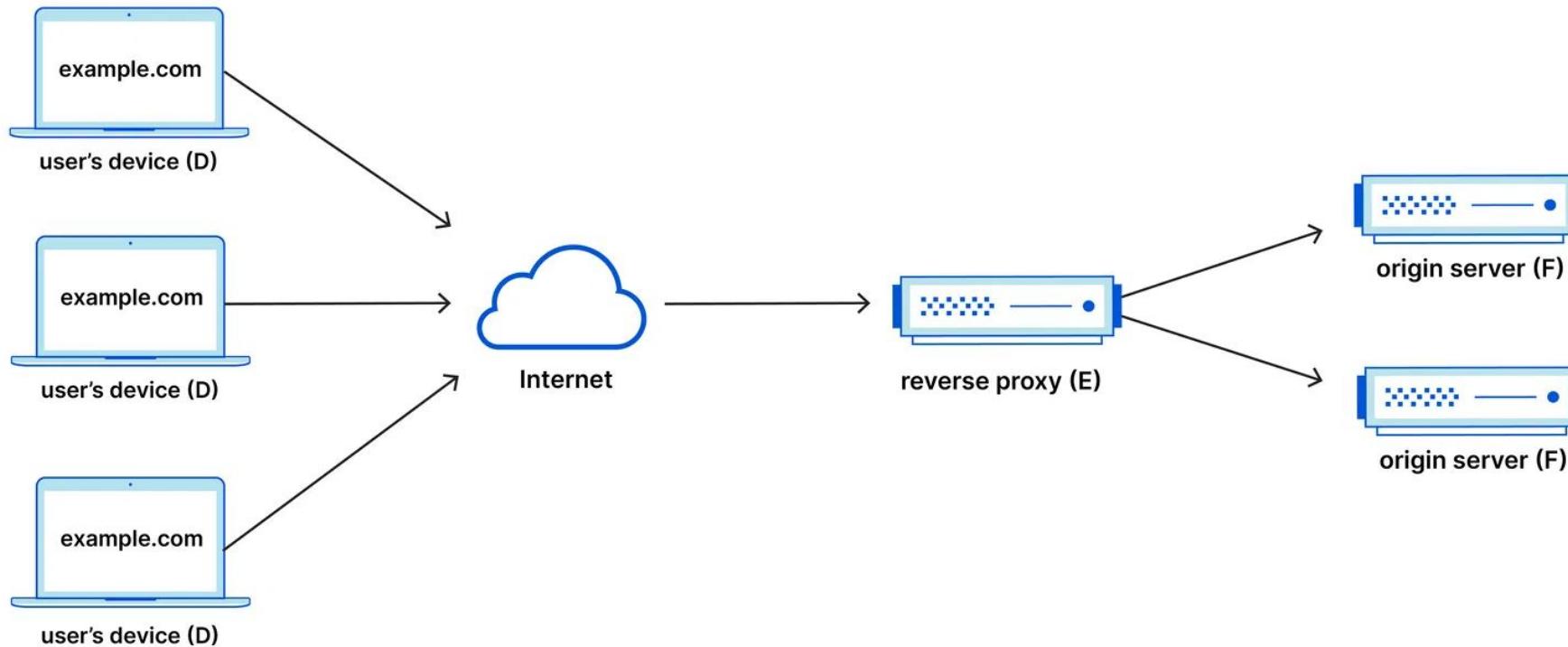
Reverse Proxy Flow





Reverse Proxy là gì ?

Reverse Proxy Flow





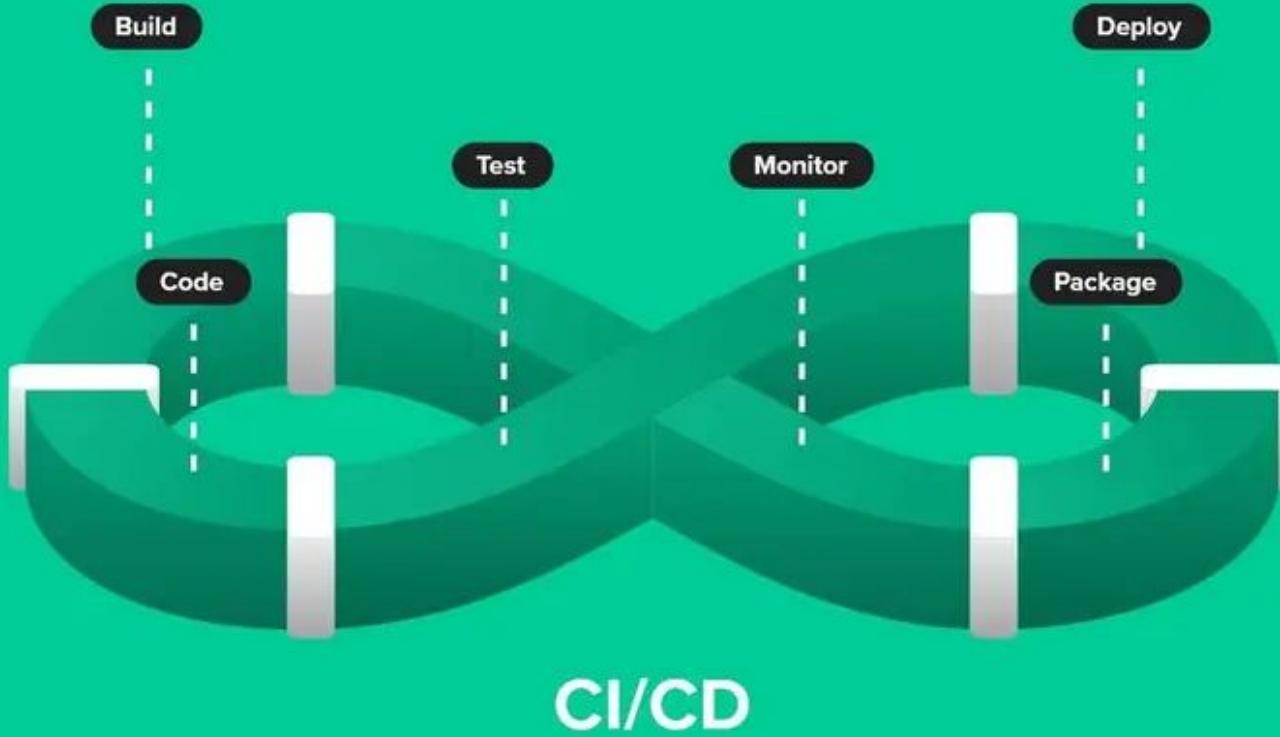
Tại sao cần Reverse Proxy ?

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





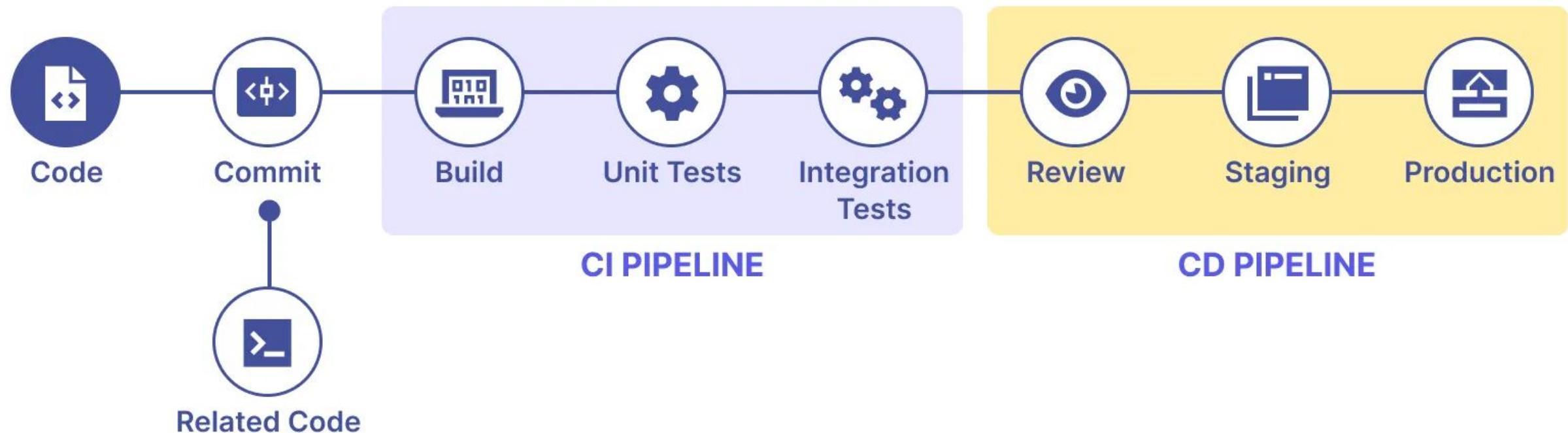
CI/CD là gì ?





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

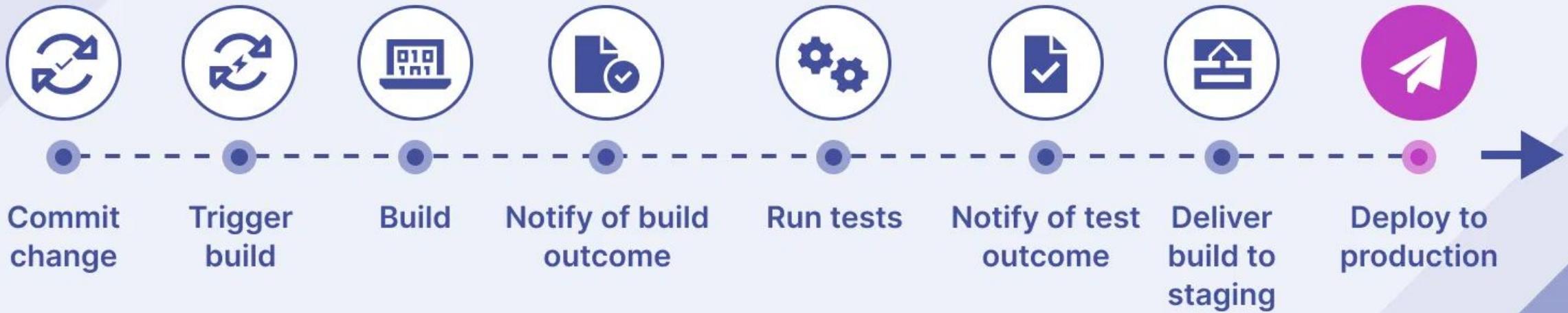
- Functional tests: Kiểm tra xem phần mềm có hoạt động đúng theo mong đợi hay không. Đây thường là các bài kiểm tra tự động mà đội ngũ QA (Quality Assurance) phát triển để đảm bảo tính năng hoạt động đúng. VD: Người dùng có thể thêm sản phẩm vào giỏ hàng và thực hiện thanh toán thành công hay không.
- 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

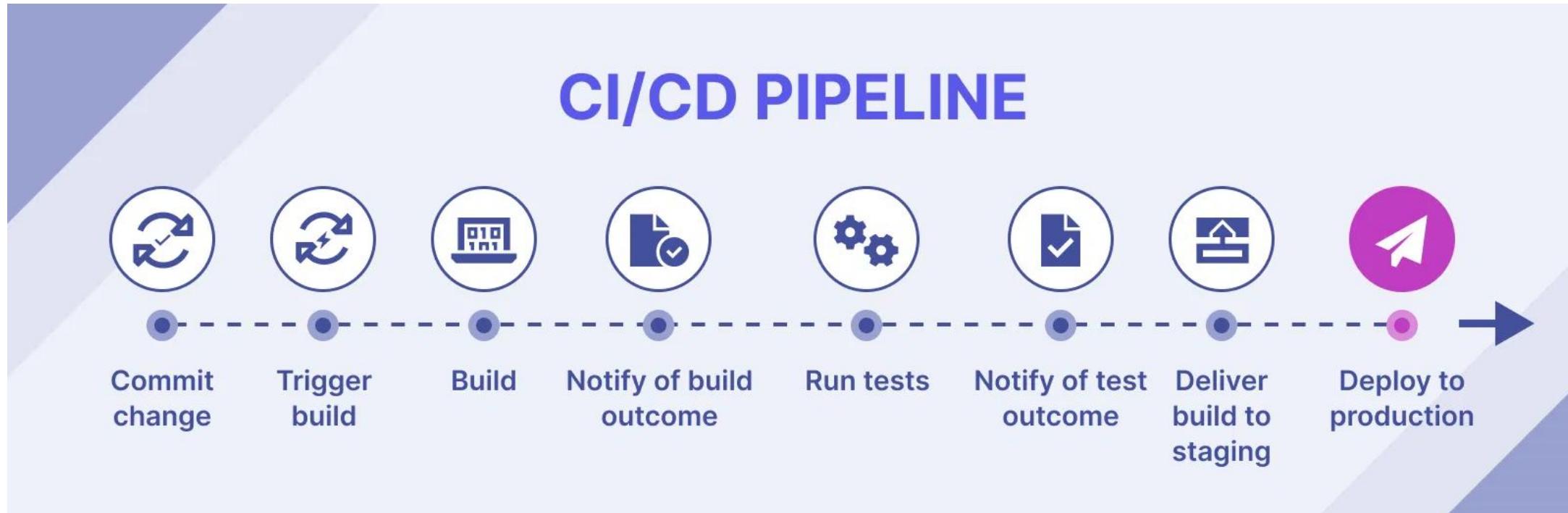
CI/CD PIPELINE





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."

