

# Microservices with Java Spring Boot and Spring Cloud

C M Abdullah Khan

Senior Software Engineer

BJIT Ltd

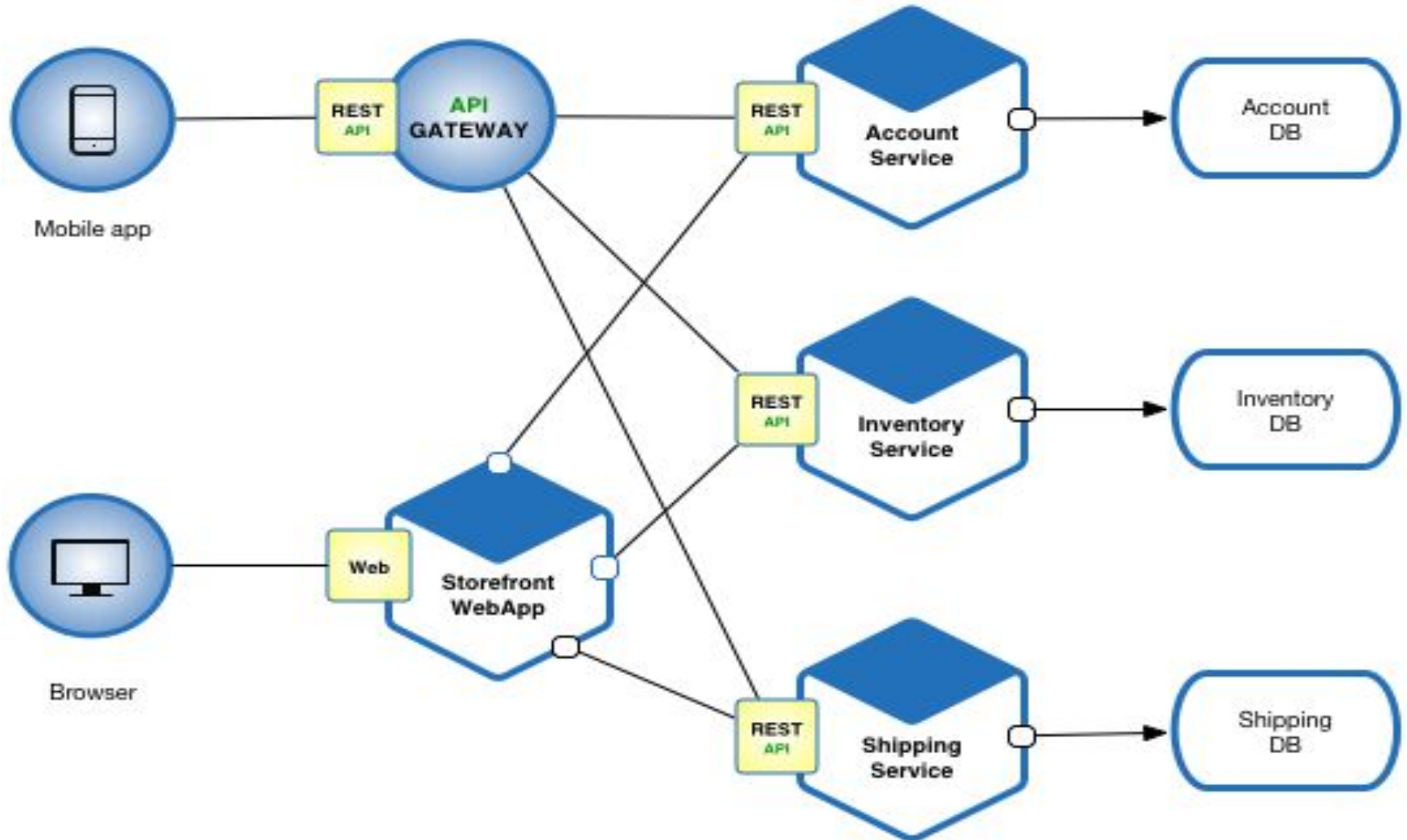
# Agenda

- MicroServices Fundamental
- Monolith vs N-Layer MicroService
- Microservice Architecture
- Microservice Advantage/Limitations and when to use!
- Framework and tools
- Intra service communications

# Day 1

- a. Introduction to Microservice Architecture
- b. Implementation of MicroServices & service discovery

# MicroServices Fundamental

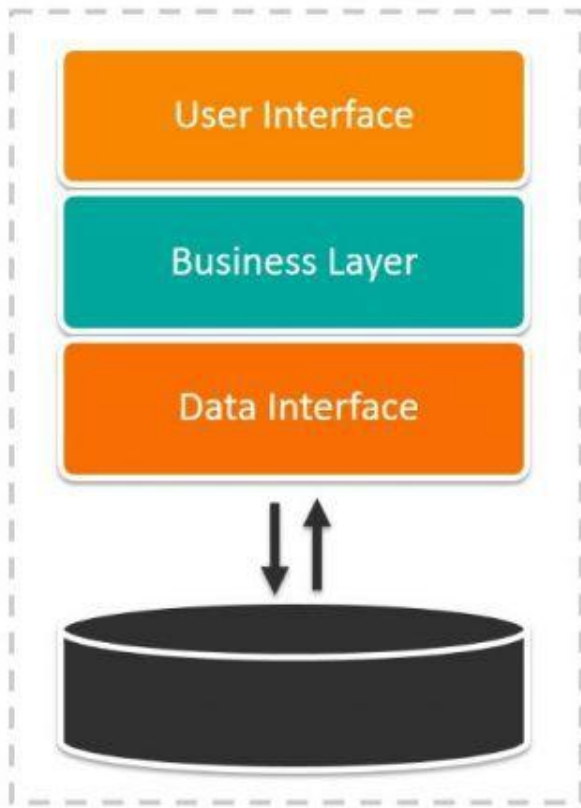


# MicroServices Fundamental

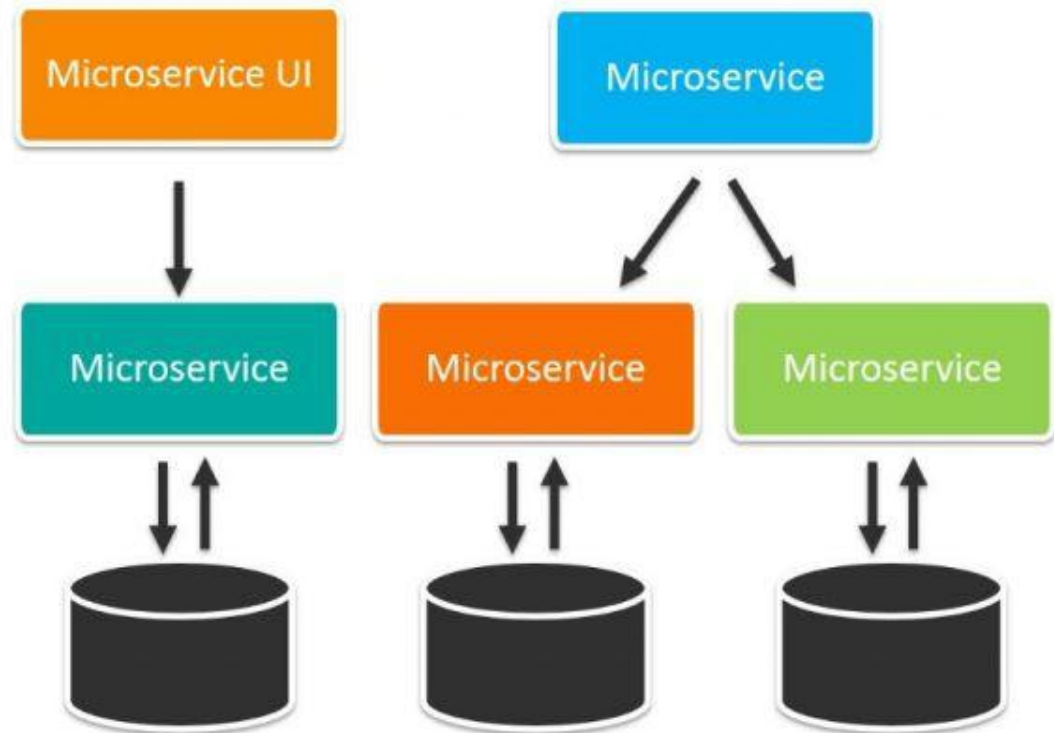
- Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are
  - I. Loosely coupled multiple services
  - II. Independently deployable
  - III. Organized around business capabilities
  - IV. Owned by a small team
  - V. Highly maintainable and testable
  - VI. Ex: Rakuten, Amazon

# Monolith vs N-Layer MicroService

**Monolithic Architecture**



**Microservices Architecture**



# Monolith vs N-Layer MicroService

- The monolithic architecture pattern has been the architectural style used in the past, pre-Kubernetes and cloud services days.
- In a monolithic architecture, the software is a single application distributed in a single place
- Characteristics of a monolithic architecture:
  - Changes are slow
  - Changes are costly
  - Hard to adapt to a specific, or changing, product line
  - Monolithic structures make changes to the application extremely slow. Modifying just a small section of code can require a completely rebuilt and deployed version of software.



# Use case

1. per product application can serve 250 queries per second  
4 core cpu, 1gb ram with 3 gb heap memory
2. per order application can serve 250 queries per second  
4 core cpu, 1gb ram with 6 gb heap memory

/product/list-> 900M -> 10,416.66 queries/S

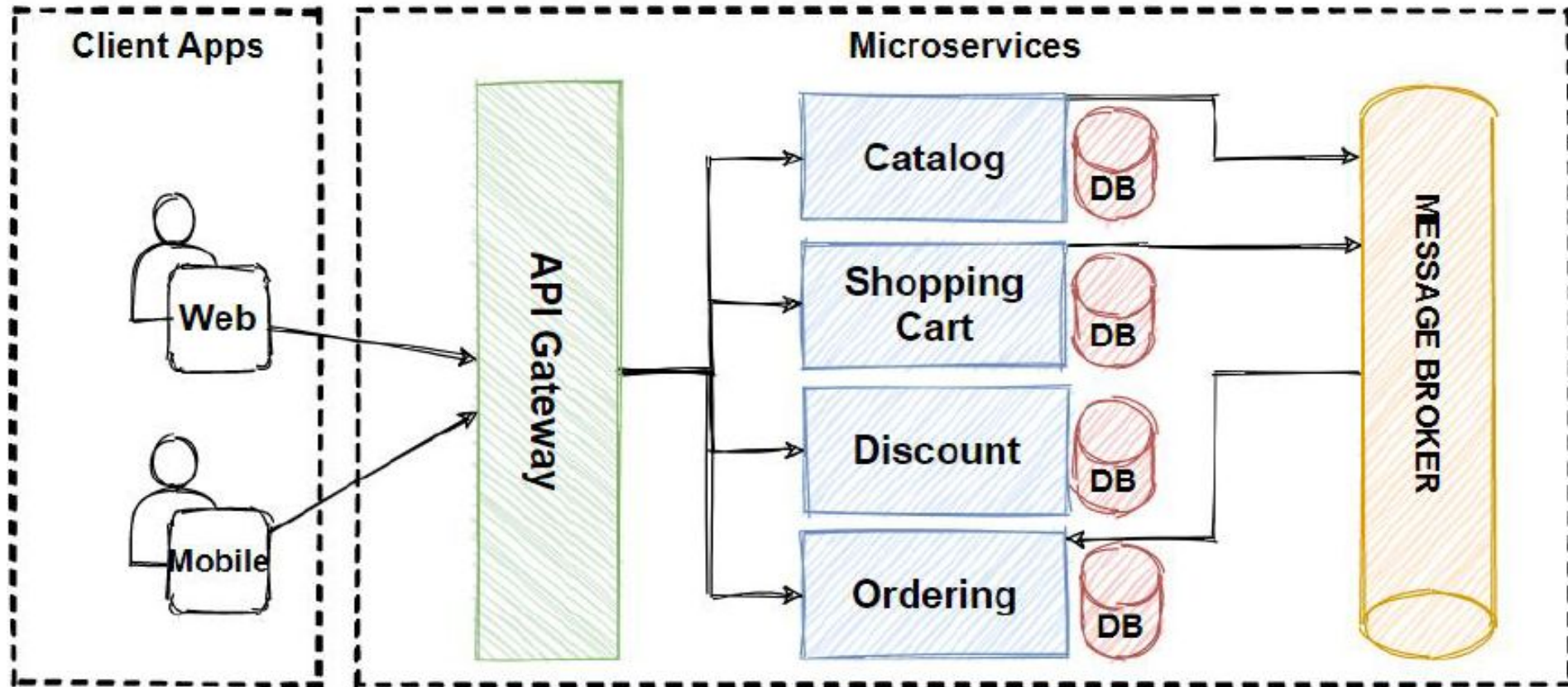
/order/submit -> 270M

system accepts 30% of orders corresponding to products view

**count the number of instances.**

**count the number of instances at the peak time.**

# Microservice Architecture



# Advantages to Microservices

- Applications built as a set of independent modular components.
- easier to test, maintain, and understand.
- Decrease the amount of time to build
- Developer independence
- Isolation & resilience
- Scalability

# Disadvantages of microservices

- Each microservice's size
- Optimal boundaries and connection points between microservices
- The right framework to integrate services
- More broadly, microservices have these drawbacks:
  - Increased complexity
  - More expensive
  - Greater security risks

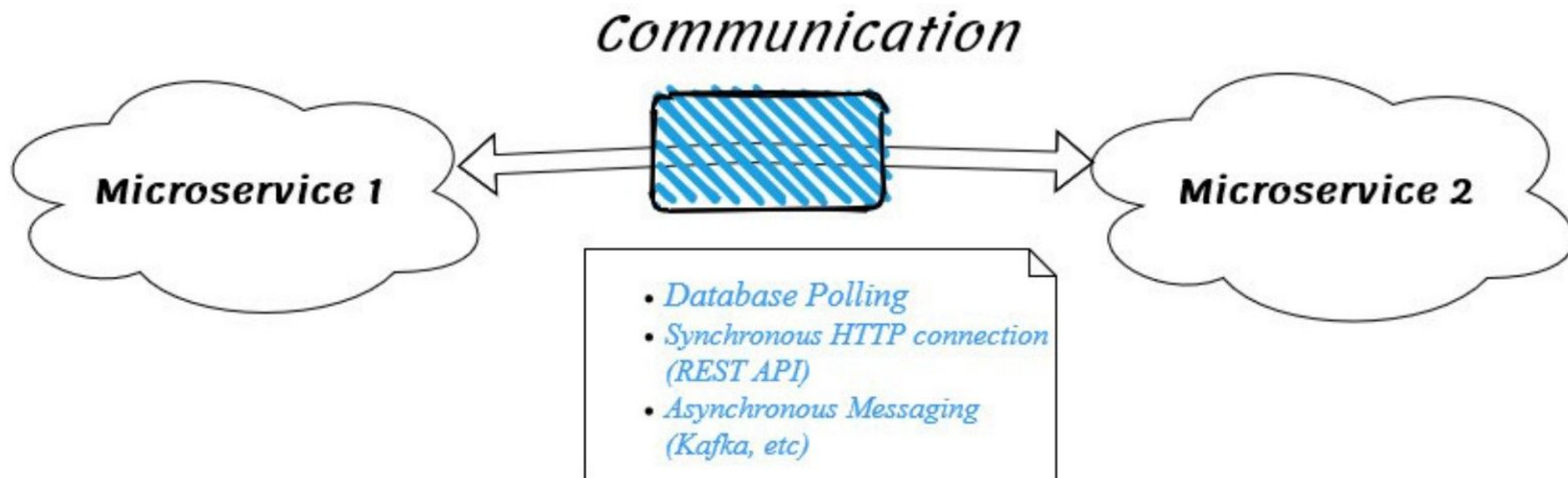
# Framework and tools

- Spring Boot with Spring Cloud
- Data Management — Spring have various modules to easily integrate with popular databases. Spring JDBC, **Spring JPA**.
- Observability — Spring Boot Actuator are powerful, it provides **health check**, view logs, **Metrix**.
- Security — Spring Security, good support for OAuth2, Session management, possible to build stateful and stateless services.
- Distributed config management — **Spring Cloud Config**
- Service Discovery — **Client Side, Server Side**
- Performance — **Caching support, Load Balancing, Clustering with Spring Cloud Cluster**
- Communication Data Format — **JSON**, XML
- Testing — Spring Testing module, **Mocking**, Profiling

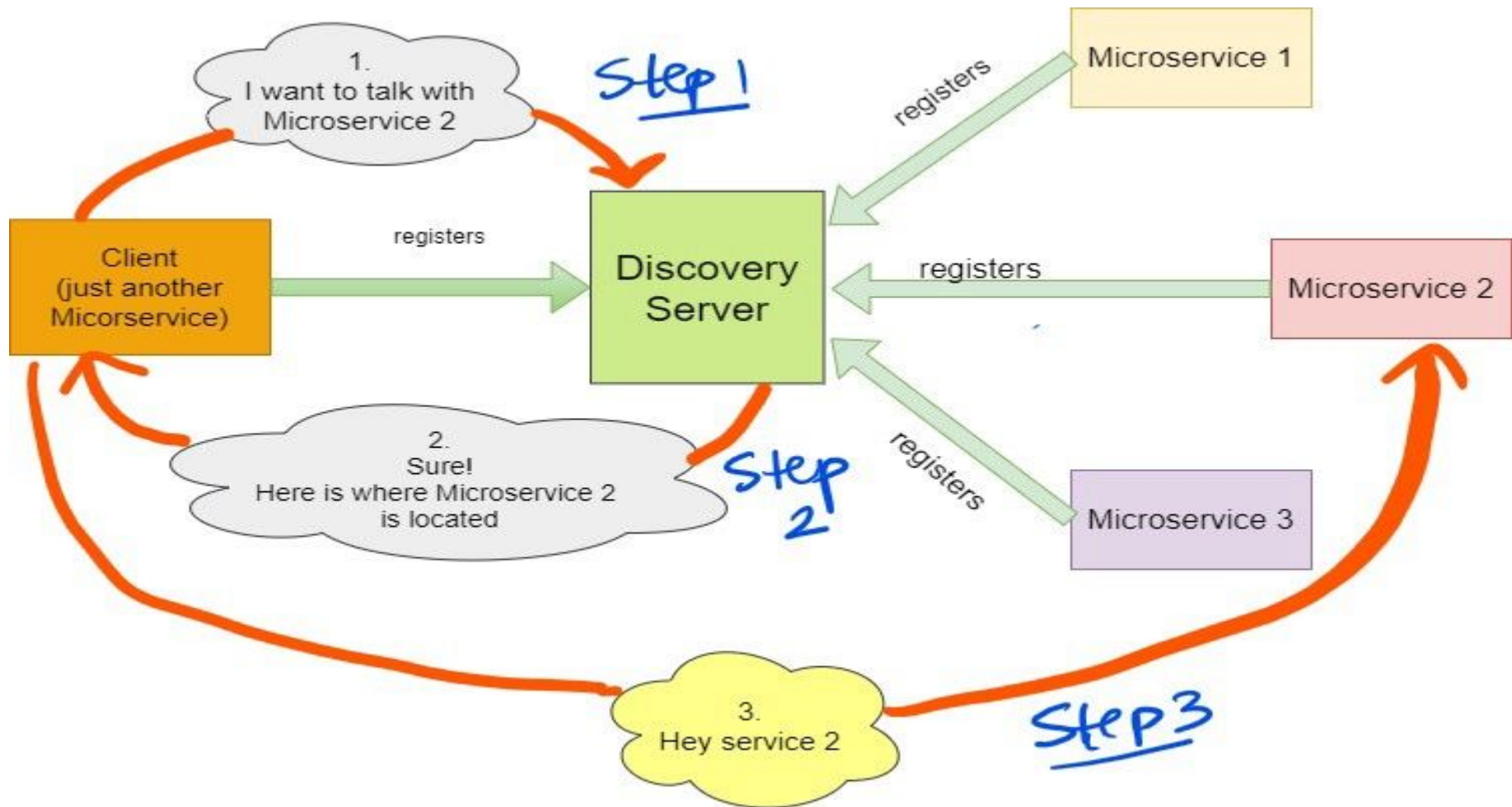
# Framework and tools

- Communication Style — Spring have capabilities to **build reactive app**. It is super easy to build rest API with the help of Spring MVC. Spring is easy to integrate with **MQ to build asynchronous style services**.
- Middle-tier Integration — With the help of various module Spring provide abstraction to integrate with **Apache Kafka**, RabbitMQ
- Integration with Tools — **Prometheus, Grafana**

# Orders to Products service communication



# Service Discovery





# @EnableEurekaServer

```
package phoenix;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class NetflixEurekaNamingServerApplication {

    public static void main(String[] args) {

SpringApplication.run(NetflixEurekaNamingServerApplication.class,
args);
    }

}
```

# spring-cloud-starter-netflix-eureka-server

```
dependencies {
```

```
    //actuator
```

```
    implementation
```

```
    'org.springframework.boot:spring-boot-starter-actuator'
```

```
    //eureka-server
```

```
    implementation
```

```
    'org.springframework.cloud:spring-cloud-starter-netflix-eureka-server'
```

```
    //lombok
```

```
    compileOnly 'org.projectlombok:lombok'
```

```
    annotationProcessor 'org.projectlombok:lombok'
```

```
    testImplementation
```

```
    'org.springframework.boot:spring-boot-starter-test'
```

```
}
```

# Eureka Server Configuration

```
spring.application.name=EurekaNamingServer  
server.port=8761
```

```
eureka.client.register-with-eureka=false  
eureka.client.fetch-registry=false  
#application.properties
```

# Eureka Dashboard

Eureka

localhost:8761

Developer Trai... Internet Spee... xRDP - How t... xRDP - Easy in... Products | c-n... KAFKA Configuring L... redis

spring Eureka

HOME LAST 1000 SINCE STARTUP

## System Status

Environment	test	Current time	2023-01-23T22:32:04 +0600
Data center	default	Uptime	01:47
		Lease expiration enabled	true
		Renews threshold	8
		Renews (last min)	12

## DS Replicas

localhost

## Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
APIGATEWAY	n/a (1)	(1)	UP (1) - localhost:ApiGateway:9090
CONFIG-SERVER	n/a (1)	(1)	UP (1) - 192.168.0.10:config-server:8888
ORDER-APP	n/a (1)	(1)	UP (1) - 192.168.0.10:order-app:8083
PRODUCT-APP	n/a (1)	(1)	UP (1) - 192.168.0.10:product-app:8082

## General Info

Name	Value
total-avail-memory	300mb
num-of-cpus	8
current-memory-usage	173mb (57%)

# Client-side service discovery

Client-side service discovery allows services to find and communicate with each other without hard-coding the hostname and port. The only 'fixed point' in such an architecture is the service registry, with which each service has to register.

# Day2

- a. Introduction of Spring Cloud config & API Gateway
- b. Implementation of Spring Cloud config & API Gateway
- c. Assignment on a hands-on task

# Eureka Client Config

*need to register with eureka server*

```
eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

*need to add dependency as config client*

implementation

```
'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'
```

# Spring Cloud Config

Spring Cloud Config provides server-side and client-side support for externalized configuration in a distributed system. With the Config Server, you have a central place to manage external properties for applications across all environments.



# @EnableConfigServer

```
package phoenix;
```

```
import org.springframework.boot.SpringApplication;
```

```
import
```

```
org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
import
```

```
org.springframework.cloud.config.server.EnableConfigServer;
```

```
@SpringBootApplication
```

```
@EnableConfigServer
```

```
public class ConfigServerApplication {
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.run(ConfigServerApplication.class,  
args);
```

```
    }
```

```
}
```

# Set the repository Path

```
spring.application.name=config-server
```

```
#spring.cloud.config.server.git.uri=file:///home/bjit/Documents/ECSG/MicroserviceEssentials/configRepo
```

```
spring.cloud.config.server.git.uri=file:///home/abdullah/Documents/workspace/MicroserviceEssentials/configRepo
```

```
server.port=8888
```

```
eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

```
#set this config in application.properties
```

# Need to add config server dependency

```
dependencies {
```

```
    //actuator
```

```
    implementation
```

```
'org.springframework.boot:spring-boot-starter-actuator'
```

```
    //eureka-client
```

```
    implementation
```

```
'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'
```

```
    //config server
```

```
    implementation
```

```
'org.springframework.cloud:spring-cloud-config-server'
```

```
    testImplementation
```

```
'org.springframework.boot:spring-boot-starter-test'
```

```
}
```

# Config Server Url End-point

- curl --location --request GET '<http://localhost:8888/order-app/default>'
- curl --location --request GET '<http://localhost:8888/config-server/default>'

# Read this properties from config Client

order-app.properties

- `order-app.applytransaction=true`

Need to make sure application name and config name must be same

File Edit View Navigate Code Refactor Build Run Tools Git Window Help

MicroserviceEssentials > OrderApp > src > main > resources > bootstrap.properties

bootstrap.properties x OrderApp/.../bootstrap.properties x ConfigSei v TransactionConfiguration.java x

```
1 spring.application.name=order-app
2 spring.cloud.config.uri=http://localhost:8888
3 #spring.profiles.active=dev
```

```
1 package phoenix.config;
2
3 import ...
4
5 3 usages Khan, C M Abdullah
6 @Component
7 @ConfigurationProperties("order-app")
8 public class TransactionConfiguration {
9
10 2 usages
11
12 1 usage Khan, C M Abdullah
13 private Boolean applytransaction;
14
15
16 public Boolean getApplytransaction() {
17     return applytransaction;
18 }
19
20 no usages Khan, C M Abdullah
21 public void setApplytransaction(Boolean applytransaction) {
22     this.applytransaction = applytransaction;
23 }
24 }
```

# API Gateway

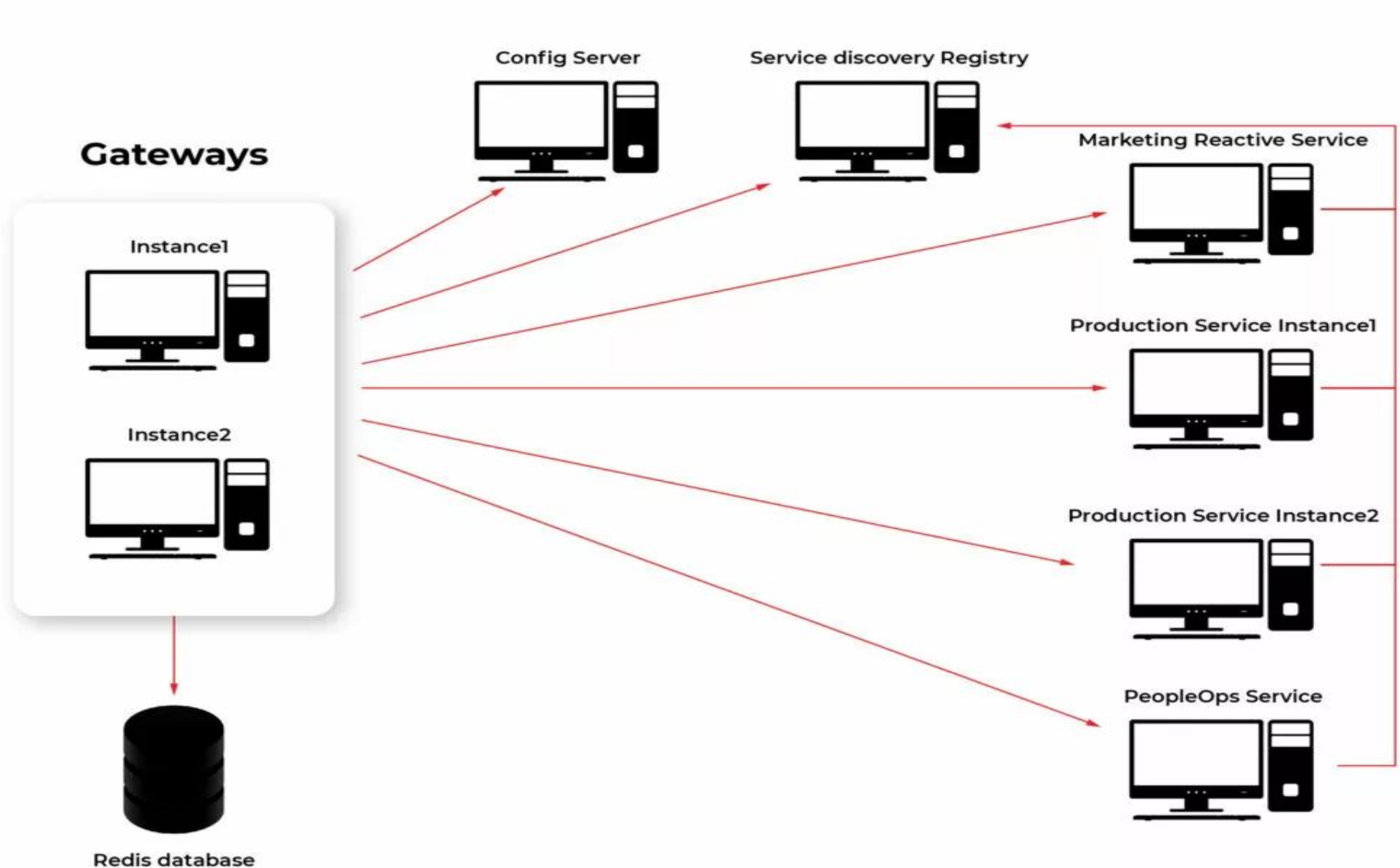
Spring Cloud Gateway aims to provide a simple, yet effective way to route to APIs and provide cross cutting concerns to them such as: security, monitoring/metrics, and resiliency.

# spring-cloud-starter-gateway

```
dependencies {  
    implementation  
    'org.springframework.cloud:spring-cloud-starter-gateway'  
    implementation  
    'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'  
  
    //lombok  
    compileOnly 'org.projectlombok:lombok'  
    annotationProcessor 'org.projectlombok:lombok'  
    testImplementation  
    'org.springframework.boot:spring-boot-starter-test'  
}
```



# Request Routing



# Global Filter

```
@Configuration
public class APIGatewayCustomFilter implements GlobalFilter
{
    Logger logger =
    LoggerFactory.getLogger(APIGatewayCustomFilter.class);

    @Override
    public Mono<Void> filter(ServerWebExchange exchange,
    GatewayFilterChain chain) {
        logger.info("Global Pre Filter executed ");
        logger.info("Request Headers = " +
    exchange.getRequest().getHeaders());
        //return chain.filter(exchange);
        return
    chain.filter(exchange).then(Mono.fromRunnable(() -> {
        logger.info("Global Post Filter executed: " +
    exchange.getResponse().getStatusCode());
        })));
    }
}
```

# Distributed Tracing

Add a Unique ID to our logs

- Trace Id: unique id is the same on the entire request
- Span Id: one trace id will contain multiple span IDs, by which we can identify individual services.
-

# Aggregate logs through Zipkin

The screenshot shows the Zipkin web interface in a browser. The address bar displays a URL for localhost:9411/zipkin with various query parameters. The interface includes a navigation bar with links like 'Investigate system behavior', 'Find a trace', 'View Saved Trace', and 'Dependencies'. Below this is a search filter section with dropdowns for 'Service Name' (set to 'all'), 'Span Name' (set to 'all'), and 'Lookback' (set to '1 hour'). There are also input fields for 'Annotation Query', 'Duration (µs) >=' (with an example '100ms or 5s'), 'Limit' (set to '10'), and 'Sort' (set to 'Longest First'). A 'Find Traces' button is present, along with a 'Showing: 7 of 7' indicator and a 'Services: all' tag. A 'JSON' download button is also visible. The main content area displays a list of traces, each with a total duration, number of spans, and a list of individual spans with their durations. The traces are sorted by duration, with the longest at the top. The first trace has a duration of 620.228ms and 3 spans. The last trace has a duration of 20.860ms and 3 spans. The interface is dark-themed.

Zipkin - Index

localhost:9411/zipkin/?serviceName=all&spanName=all&lookback=3600000&startTs=1674632725337&endTs=1674636325337&annotationQuery=&minDuration=&limit=10&sortOrder=duration-desc

essential docu... prometheus -> How to fix -D... How to Install... Running Sona... limit -Elastics... Projects GRAPHQL common-todo ECSG Tech ENT-4 Rangam... Huddel BFF

Investigate system behavior Find a trace View Saved Trace Dependencies

Try Lens UI Go to trace Search

Service Name Span Name Lookback

all all 1 hour

Annotation Query Duration (µs) >= Limit Sort

For example: http.path=/foo/bar/ and cluster=foo and cache.miss Ex: 100ms or 5s 10 Longest First

Find Traces ?

Showing: 7 of 7 Services: all JSON

620.228ms 3 spans

order-app x2 620.228ms product-app x1 279.114ms 6 minutes ago

24.899ms 3 spans

order-app x2 24.899ms product-app x1 12.664ms 5 minutes ago

22.247ms 3 spans

order-app x2 22.247ms product-app x1 10.803ms 5 minutes ago

22.086ms 3 spans

order-app x2 22.086ms product-app x1 10.763ms 5 minutes ago

21.719ms 3 spans

order-app x2 21.719ms product-app x1 11.259ms 5 minutes ago

21.534ms 3 spans

order-app x2 21.534ms product-app x1 10.562ms 5 minutes ago

20.860ms 3 spans

order-app x2 20.860ms product-app x1 9.790ms 2 minutes ago

# Order Success

traceld list

Order App		Product App	
traceld	A	traceld	A
spanId	A	spanId	B
parentId		parentId	A

# Order App call

Investigate system behavior Find a trace View Saved

Duration: 68.659ms Services: 2

Expand All Collapse All

order-app x2 product-app x1

Services

- order-app
- product-app

13.7

68.659ms : post /api/order/submit

13.633ms : post /api/product/updatequantity

order-app.post /api/order/submit: 68.659ms

Services: order-app

Date Time	Relative Time	Annotation	Address
1/26/2023, 3:58:19 PM		Server Start	172.20.0.1 (order-app)
1/26/2023, 3:58:19 PM	68.659ms	Server Finish	172.20.0.1 (order-app)

Key	Value
http.method	POST
http.path	/api/order/submit
mvc.controller.class	OrderController
mvc.controller.method	addProducts
Client Address	:::1]:57644

Show IDs

traceld	1a3aa81ee8dcc13f
spanId	1a3aa81ee8dcc13f
parentId	

# Product-App call

product-app.post /api/product/updatequantity: 13.633ms

Duration: 68.659ms

Services: 2

Expand All

Collapse All

order-app x2 product-app x1

Services 13.7

order-app 68.659ms : post /api/order/submit  
product-app 13.633ms : post /api/product/updatequantity

Services: order-app,product-app

Date Time	Relative Time	Annotation	Address
1/26/2023, 3:58:19 PM	2.393ms	Client Start	172.20.0.1 (order-app)
1/26/2023, 3:58:19 PM	5.175ms	Server Start	172.20.0.1 (product-app)
1/26/2023, 3:58:19 PM	16.026ms	Client Finish	172.20.0.1 (order-app)
1/26/2023, 3:58:19 PM	16.093ms	Server Finish	172.20.0.1 (product-app)

Key	Value
http.method	POST
http.path	/api/product/updateQuantity
mvc.controller.class	ProductController
mvc.controller.method	updateProductsQuantity
Client Address	192.168.96.217:45132

Show IDs

traceld	1a3aa81ee8dcc13f
spanId	4c7f23a840ee1136
parentId	1a3aa81ee8dcc13f

# Order Failed

traceId list

Order App [/api/order/submit]		Product App [/api/product/updateQuantity]		Product App [/api/product/revokeQuantity]	
traceId	A	traceId	A	traceId	A
spanId	A	spanId	B	spanId	C
parentId		parentId	A	parentId	A



# When Order Failed

Investigate system behavior Find a trace View Saved

Duration: 31.755ms Services: 2

Expand All Collapse All

order-app x3 product-app x2

Services

- order-app 31.755ms : post /error
- product-app 11.590ms : post /api/product
- product-app

order-app.post /error: 31.755ms

Services: order-app

Date Time	Relative Time	Annotation	Address
1/26/2023, 4:08:14 PM		Server Start	172.20.0.1 (order-app)
1/26/2023, 4:08:14 PM	31.755ms	Server Finish	172.20.0.1 (order-app)

Key	Value
error	order fauled
http.method	POST
http.path	/api/order/submit
http.status_code	500
mvc.controller.class	BasicErrorController
mvc.controller.method	error
Client Address	:::1]:50838

Show IDs

traceld	9b23694847b8b869
spanId	9b23694847b8b869
parentId	

# Product app Update call /api/product/updateQuantity

Investigate system behavior Find a trace View Saved

Duration: **31.755ms** Services: **2**

Expand All Collapse All

order-app x3 product-app x2

Services

- order-app 31.755ms : post /error
- product-app 11.590ms : post /api/product/...
- product-app

product-app.post /api/product/updatequantity: 11.590ms

Services: order-app,product-app

Date Time	Relative Time	Annotation	Address
1/26/2023, 4:08:14 PM	3.044ms	Client Start	172.20.0.1 (order-app)
1/26/2023, 4:08:14 PM	4.021ms	Server Start	172.20.0.1 (product-app)
1/26/2023, 4:08:14 PM	14.634ms	Client Finish	172.20.0.1 (order-app)
1/26/2023, 4:08:14 PM	14.791ms	Server Finish	172.20.0.1 (product-app)

Key	Value
http.method	POST
http.path	/api/product/updateQuantity
mvc.controller.class	ProductController
mvc.controller.method	updateProductsQuantity
Client Address	192.168.96.217:59522

Show IDs

traceld	9b23694847b8b869
spanId	b5100de70d1be331
parentId	9b23694847b8b869

# Product app Update call /api/product/revokeQuantity

Investigate system behavior Find a trace View Saved

Duration: 31.755ms Services: 2

Expand All Collapse All

order-app x3 product-app x2

Services

order-app 31.755ms : post /error

product-app 11.590ms : post /api/product/

product-app

product-app.post /api/product/revokequantity: 11.841ms

Services: order-app,product-app

Date Time	Relative Time	Annotation	Address
1/26/2023, 4:08:14 PM	16.120ms	Client Start	172.20.0.1 (order-app)
1/26/2023, 4:08:14 PM	18.153ms	Server Start	172.20.0.1 (product-app)
1/26/2023, 4:08:14 PM	27.961ms	Client Finish	172.20.0.1 (order-app)
1/26/2023, 4:08:14 PM	28.321ms	Server Finish	172.20.0.1 (product-app)

Key	Value
http.method	POST
http.path	/api/product/revokeQuantity
mvc.controller.class	ProductController
mvc.controller.method	revertProductsQuantity
Client Address	192.168.96.217:59522

Show IDs

traceld	9b23694847b8b869
spanId	53141536898e2d2f
parentId	9b23694847b8b869

# Task

write a microservice named **transaction** service  
it will be called with specific parameters,

1. transaction service will return true or false by reading config properties from the config server.
2. after that order will be placed successfully or failed.
3. transaction service will be registered with service discovery server

```
—— //call product service
```

```
—— ResponseEntity<List<ProductResponseDto>> apiResponse =
```

```
—— productFeignClient.updateProductsQuantity(productOrderRequestDto);
```

```
—— List<ProductResponseDto> requestedProductList =
```

```
—— apiResponse.getBody() != null ? apiResponse.getBody() : Collections.emptyList();
```

```
//—— callProductService.callProductService(productOrderRequestDto,
```

```
//—— "http://localhost:8082/api/product/updateQuantity");
```

```
—— //call payment app
```

```
—— //if success then return
```

```
//else revert as failure
```

```
if (transaction()) {
```

```
—— //transaction success
```

# Day3

- a. Various Profile management using Spring Cloud config
- b. Follow up hands-on task
- c. Q & A

- Ref:
- <https://www.bmc.com/blogs/microservices-architecture/>
- <https://microservices.io/>
- <https://medium.com/microservices-architecture/top-10-microservices-framework-for-2020-eefb5e66d1a2>
- <https://www.baeldung.com/spring-cloud-netflix-eureka>
- <https://grapeup.com/blog/big-picture-of-spring-cloud-gateway/#>
- <https://www.baeldung.com/spring-cloud-custom-gateway-filters>