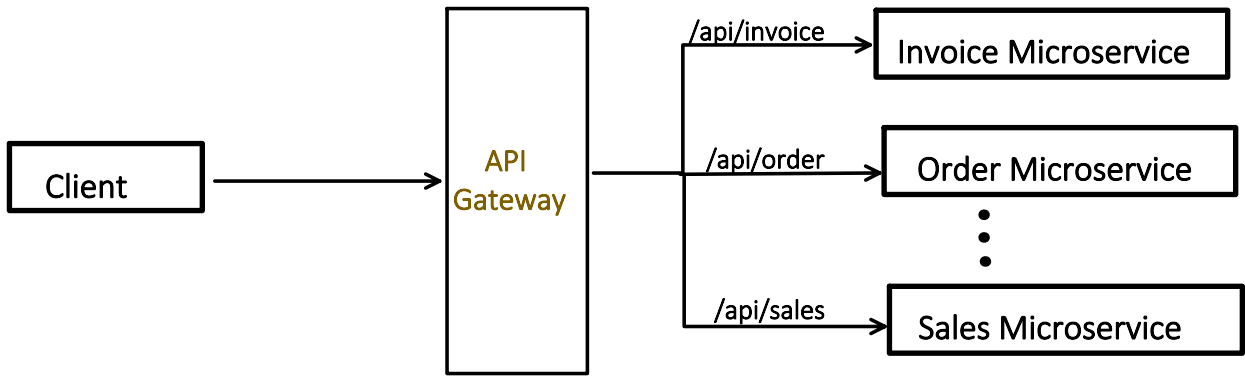


API Gateway - Part1

Wednesday, 27 August 2025 10:17 AM

API Gateway:

It provides a Single entry point to access all microservices.



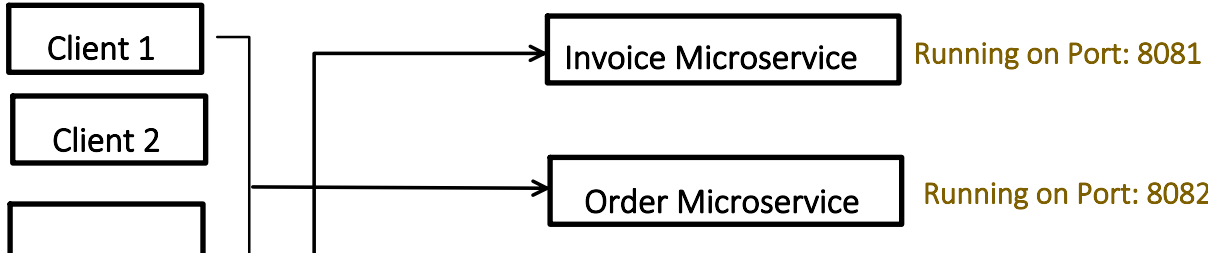
And this Single Entry Point,,provides lot of benefits like:

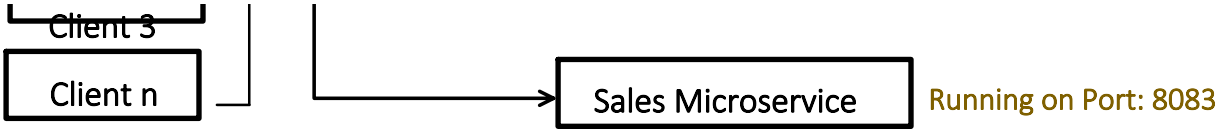
- 1. Routing : forward requests to right microservices.
- 2. Load balancing
- 3. Authentication like JWT
- 4. Rate Limiting
- 5. Resilience features (Circuit breaker, Retry etc.)
- 6. Request/Response Transformation
- 7. Monitoring and Logging.
- Etc..

Let's see 1 by 1, how we can achieve all of the above with the API Gateway:

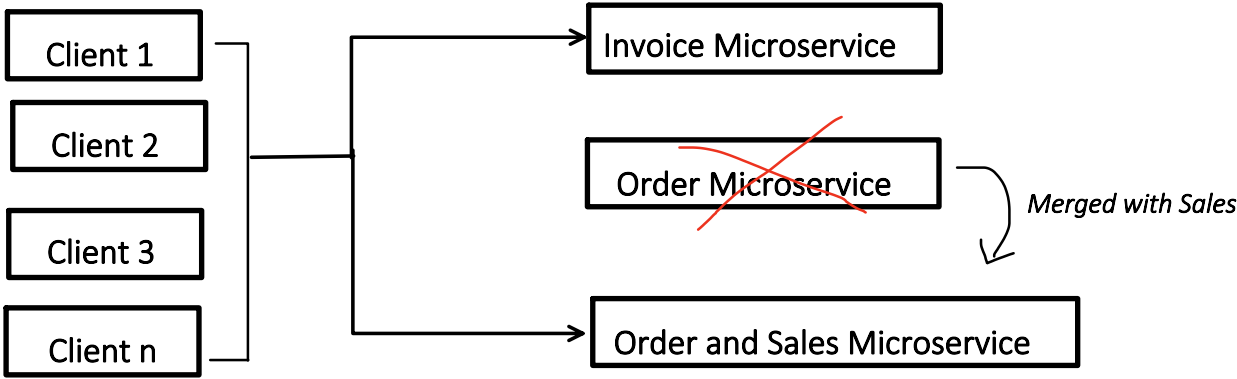
- 1. Routing : forward requests to right microservices.

Assume this:



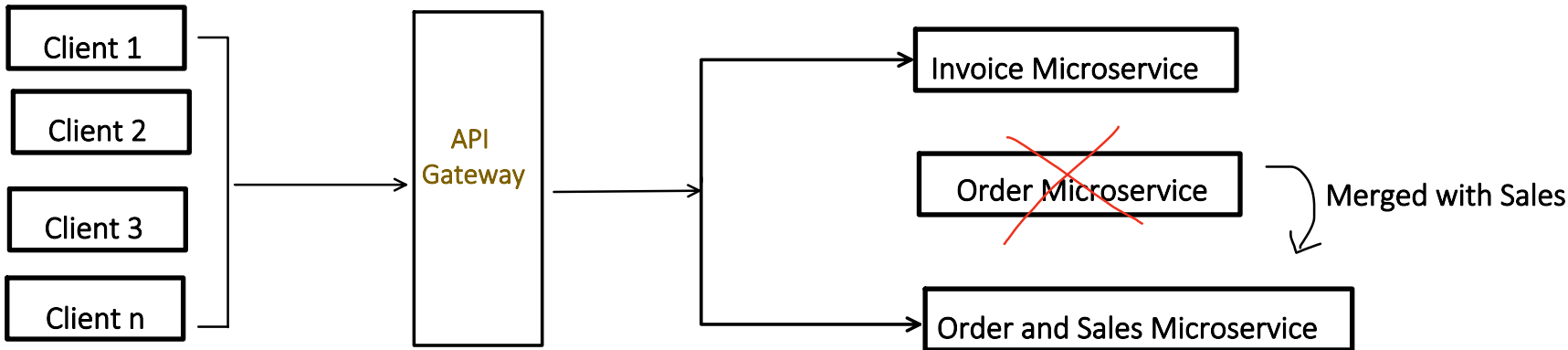


If 1 microservice is further divided into 2 or Multiple microservices merged together, then all clients need to update the routing logic.



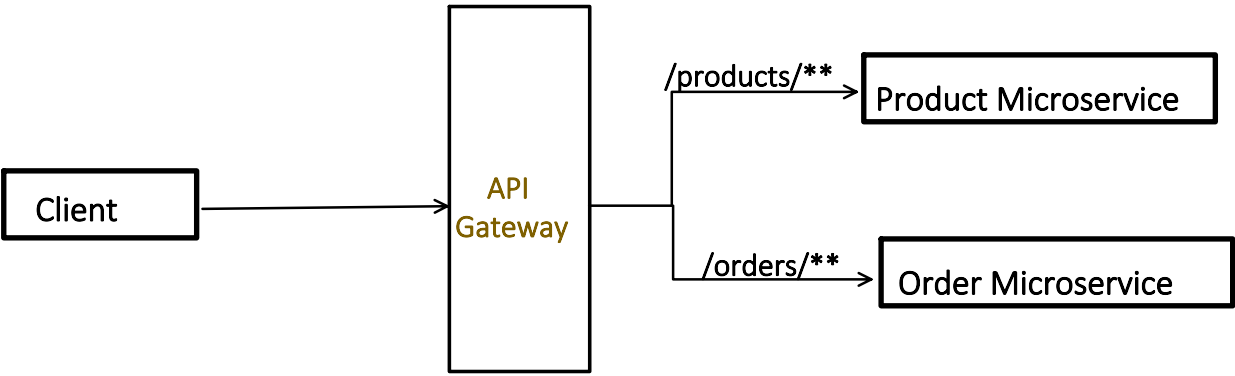
(All clients need to update the routing logic)

But with API Gateway in Mid, changes update required only at 1 service i.e. Gateway



(Only API Gateway need to be updated)

Lets implement this:



Product Controller Class:

```
@RestController
@RequestMapping("/products")
```

```
public class ProductController {  
  
    @GetMapping("/{id}")  
    public ResponseEntity<String> getProduct(@PathVariable String id) {  
        return ResponseEntity.ok().body("fetch the product details with id:" + id);  
    }  
}
```

application.properties

```
server.port=8082  
spring.application.name=product-service
```

OneNote

Project

☐ Gradle - Groovy

☐ Gradle - Kotlin

☒ Maven

Language

☒ Java

☐ Kotlin

☐ Groovy

Spring Boot

☐ 4.0.0 (SNAPSHOT)

☐ 4.0.0 (M2)

☐ 3.5.6 (SNAPSHOT)

☐ 3.5.5

☐ 3.4.10 (SNAPSHOT)

☒ 3.4.9

Project Metadata

Group

com.conceptandcoding

Artifact

product

Name

product

Description

Learning APIGateway

Package name

com.conceptandcoding.product

Packaging

☒ Jar

☐ War

Java

☐ 24

☐ 21

☒ 17

Dependencies

ADD DEPENDENCIES...

⌘ + B

Spring Web

WEB

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

Screenshot 2025-08-18 at 5.25.33 PM

Order Controller Class:

```
@RestController  
@RequestMapping("/orders")  
public class OrderController {  
  
    @GetMapping("/{id}")  
    public ResponseEntity<String> getOrder(@PathVariable String id) {  
        return ResponseEntity.ok().body("fetch the Order details with id:" + id);  
    }  
}
```

application.properties

```
server.port=8081  
spring.application.name=order-service
```

Project

☐ Gradle - Groovy

☐ Gradle - Kotlin

☒ Maven

Language

☒ Java

☐ Kotlin

☐ Groovy

Spring Boot

☐ 4.0.0 (SNAPSHOT)

☐ 4.0.0 (M2)

☐ 3.5.6 (SNAPSHOT)

☐ 3.5.5

☐ 3.4.10 (SNAPSHOT)

☒ 3.4.9

Project Metadata

Group

com.conceptandcoding

Artifact

order

Name

order

Description

Learning APIGateway

Package name

com.conceptandcoding.order

Packaging

☒ Jar

☐ War

Java

☐ 24

☐ 21

☒ 17

Dependencies

ADD DEPENDENCIES...

⌘ + B

Spring Web

WEB

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

Create a new Gateway Application via Spring Initializr:

Project

☐ Gradle - Groovy

☐ Gradle - Kotlin

☒ Maven

Language

☒ Java

☐ Kotlin

☐ Groovy

Spring Boot

☐ 4.0.0 (SNAPSHOT)

☐ 4.0.0 (M2)

☐ 3.5.6 (SNAPSHOT)

☐ 3.5.5

☐ 3.4.10 (SNAPSHOT)

☒ 3.4.9

Project Metadata

Group

com.conceptandcoding

Artifact

gateway

Name

gateway

Description

Learning APIGateway

Package name

com.conceptandcoding.gateway

Packaging

☒ Jar

☐ War

Java

☐ 24

☐ 21

☒ 17

Dependencies

ADD DEPENDENCIES...

⌘ + B

Reactive Gateway

SPRING CLOUD ROUTING

Provides a simple, yet effective way to route to APIs in reactive applications. Provides cross-cutting concerns to those APIs such as security, monitoring/metrics, and resiliency.

api-gateway

Name

apigateway

Description

Learning API Gateway

Package name

com.conceptandcoding.apigateway

Packaging

Jar

War

Java

24

21

17

In pom.xml, below dependency will get added

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
```

application.properties

```
1  spring.application.name=apigateway
2  server.port=8083
3
4  # Route: Forward /products/** to product-service
5  spring.cloud.gateway.routes[0].id=product-service
6  spring.cloud.gateway.routes[0].uri=http://localhost:8082
7  spring.cloud.gateway.routes[0].predicates[0]=Path=/products/**
8
9  # Route: Forward /orders/** to order-service
10 spring.cloud.gateway.routes[1].id=order-service
11 spring.cloud.gateway.routes[1].uri=http://localhost:8081
12 spring.cloud.gateway.routes[1].predicates[0]=Path=/orders/**
13
```

Id: This is just a unique name to this route

Uri: Microservice host and port number

predicates: pattern matching to decide the routing.
apart from Path, we can also have predicate on Method like:
spring.cloud.gateway.routes[0].predicates[1]=Method=GET,POST

means GET and POST request to /products/** will match this route.

Start all 3 servers: Product, Order and API Gateway.

Product service is running on 8082

Order service is running on 8081

```
main] c.c.p.ProductServiceApplication : Starting ProductServiceApplication using Java 17.0.
main] c.c.p.ProductServiceApplication : No active profile set, falling back to 1 default pr
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8082 (http)
main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.20]
main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization complete
```

```
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8082 (http) with context pat
main] c.c.p.ProductServiceApplication : Started ProductServiceApplication in 0.583 seconds
-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherSe
-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 0 ms
```

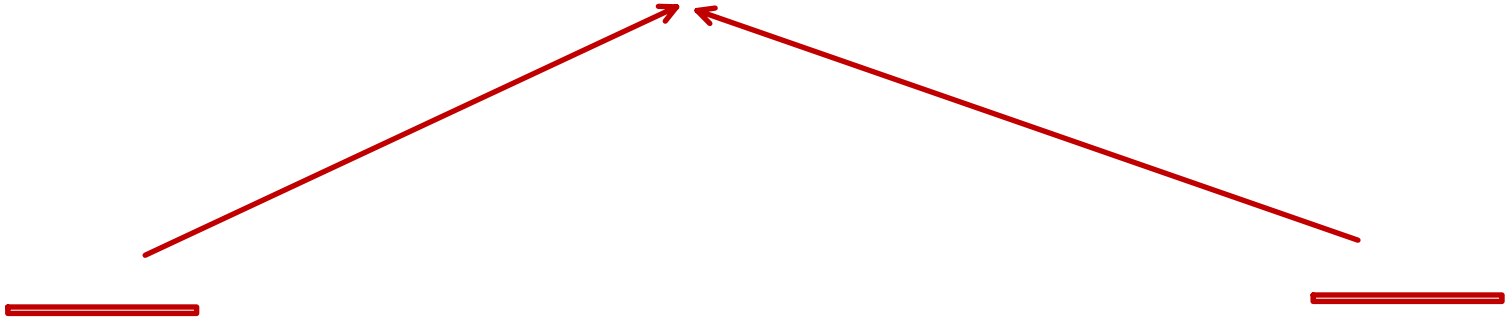
OneNote



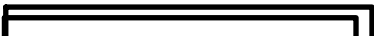
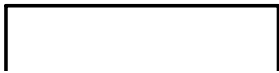
```
Run ApigatewayApplication x
[main] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [Path]
[main] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [Query]
[main] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [ReadBody]
[main] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [RemoteAddr]
[main] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [XForwardedRemoteA
[main] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [Weight]
[main] o.s.c.g.r.RouteDefinitionRouteLocator : Loaded RoutePredicateFactory [CloudFoundryRoute
[main] o.s.b.web.embedded.netty.NettyWebServer : Netty started on port 8083 (http)
[main] c.c.apigateway.ApigatewayApplication : Started ApigatewayApplication in 0.761 seconds
[ctor-http-nio-3] i.n.r.d.DnsServerAddressStreamProviders : Unable to load io.netty.resolver.dns.macos.MacO
```

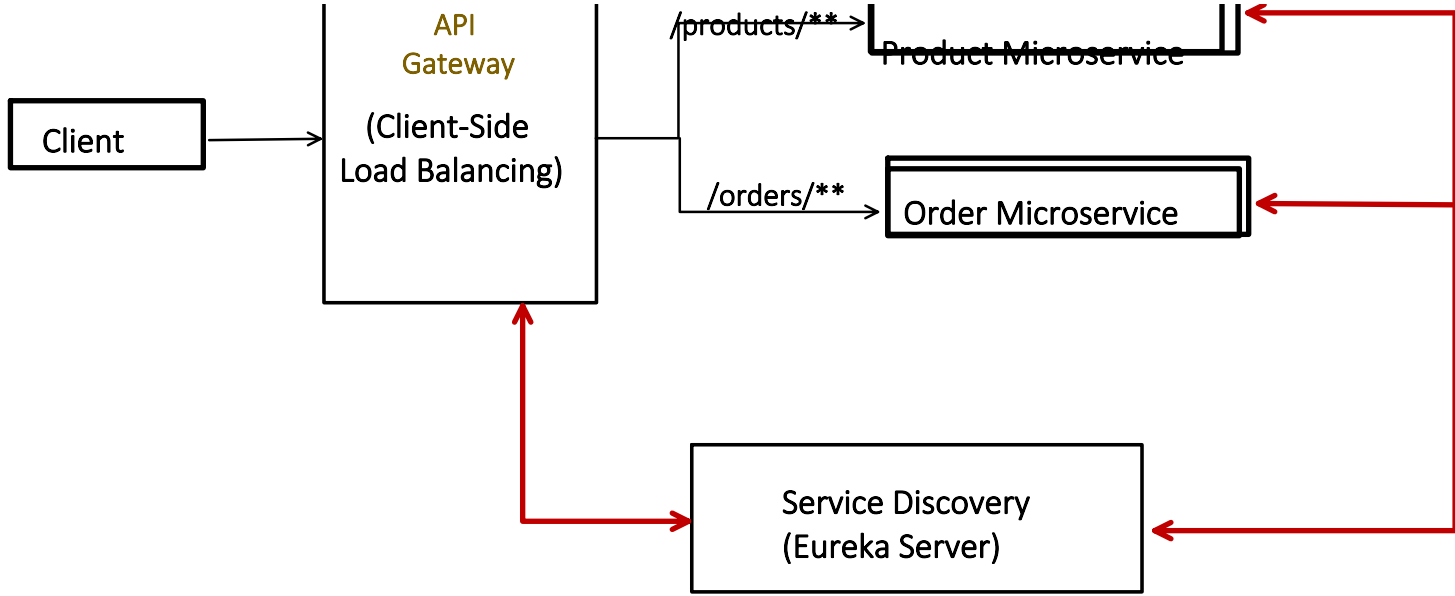
API Gateway service is running on 8083

Invoked API Gateway, and based on PATH matching it invoked the respective microservice.



2. API Gateway and Load Balancer





Already covered **Service Discovery** and **Client Side Load Balancer** topic in this playlist, if you have any doubt with below topics, kindly check it out first.

Product Service:

Pom.xml

application.properties

Order Service:

Pom.xml

application.properties

Controller class

```
@RestController
@RequestMapping("/products")
public class ProductController {

    @GetMapping("/{id}")
    public ResponseEntity<String> getProduct(@PathVariable String id) {
        return ResponseEntity.ok().body("fetch the product details with id:" + id);
    }
}
```

Controller class

Eureka Server (Service Discovery):

Pom.xml

application.properties

API Gateway

Pom.xml

application.properties

Replaced the hardcoded URI: <http://localhost:8082>
- Now internally it will use Spring Cloud LoadBalancer framework to perform the load balancing.
- first will fetch the instances from Service Discovery and
- Perform Load balancing algorithm

