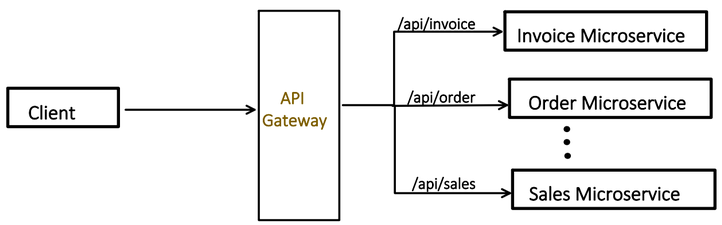
API Gateway - Part1

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:

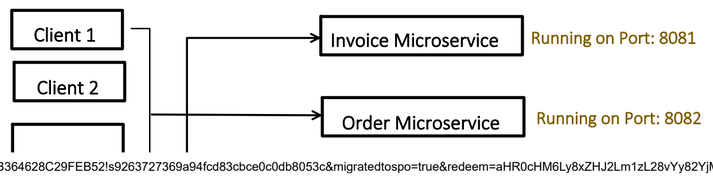
1. Routing : forward requests to right microservices.
2. Load balancing
3. Authentication like JWT
4. Rate Limiting
5. Resilience features (Circuit breaker, Retry etc.)
6. Request/Response Transformation
7. Monitoring and Logging.

   Etc..

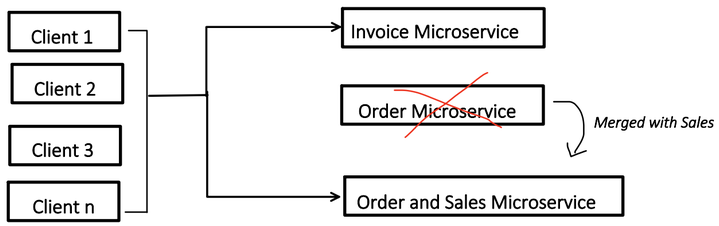
Let's see 1 by 1, how we can achieve all of the above with the API Gateway:

1. Routing : forward requests to right microservices.

Assume this:

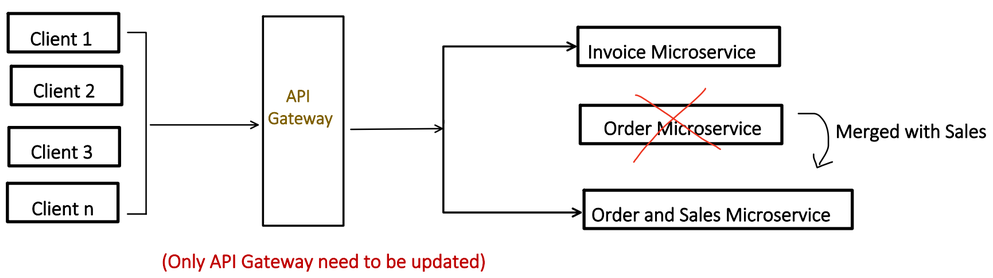


If 1 microservice is further divided into 2 or Multiple microservices merged together, then all clients need to update the routing logic.



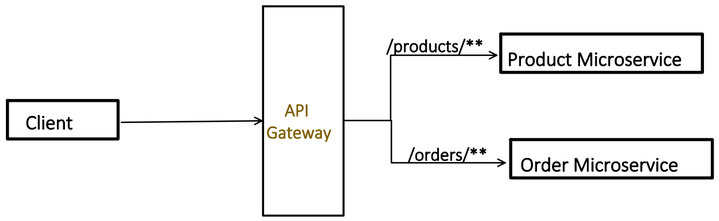
(All clients need to update the routing logic)

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:

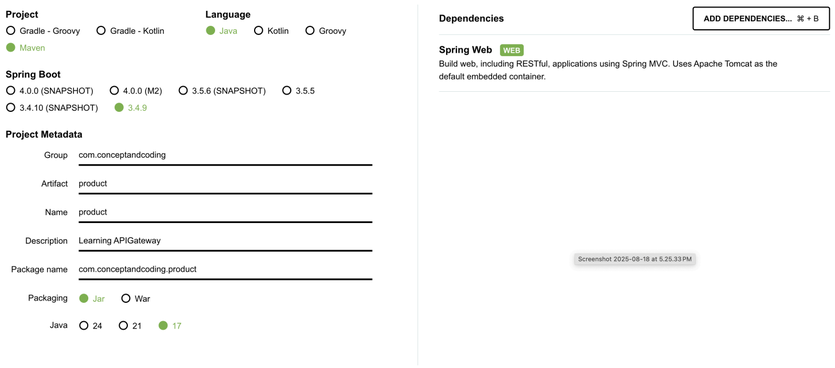
```
@RestController
@RequestMapping("/products")
```

```
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
```



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
```


Create a new Gateway Application via Spring Initializr:

In pom.xml, below dependency will get added

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

application.properties

```
spring.application.name=apigateway
server.port=8083

# Route: Forward /products/** to product-service
spring.cloud.gateway.routes[0].id=product-service
spring.cloud.gateway.routes[0].uri=http://localhost:8082
spring.cloud.gateway.routes[0].predicates[0]=Path=/products/**

# Route: Forward /orders/** to order-service
spring.cloud.gateway.routes[1].id=order-service
spring.cloud.gateway.routes[1].uri=http://localhost:8081
spring.cloud.gateway.routes[1].predicates[0]=Path=/orders/**
```

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.

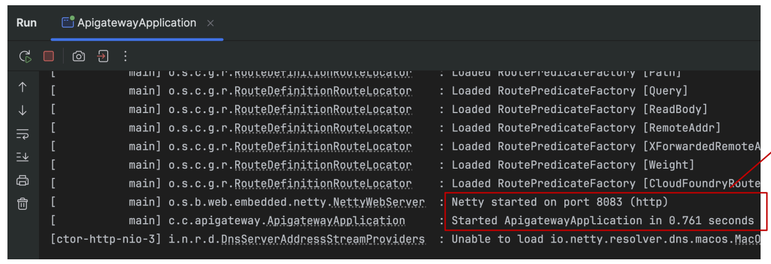
Start all 3 servers: Product, Order and API Gateway

Product service is running on 8082

Order service is running on 8081

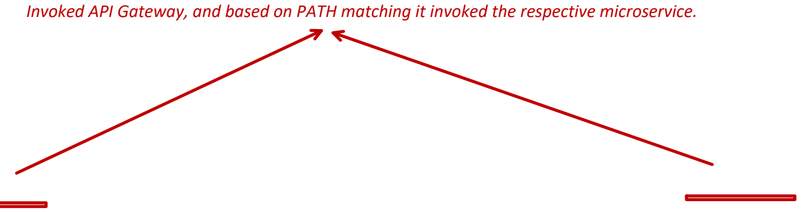
```
main] c.c.p.ProductserviceApplication          : Starting ProductserviceApplication using Java 17.0.
main] c.c.p.ProductserviceApplication          : No active profile set, falling back to 1 default pr
main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8082 (http)
main] 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
```

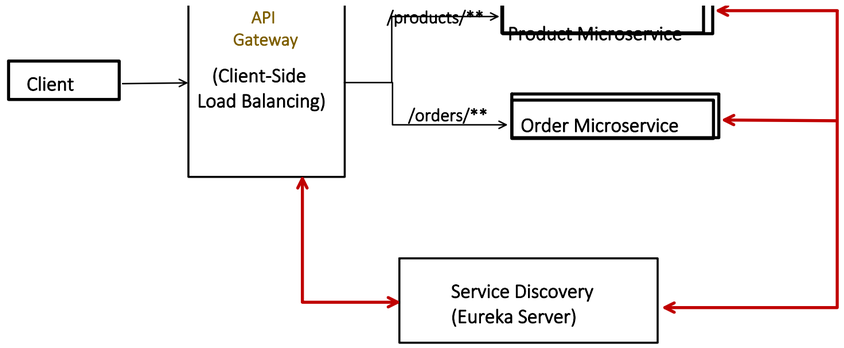


API Gateway service is running on 8083

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

```java
@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