

## API Gateway

### 👉 What is an API Gateway?

An **API Gateway** serves as a **single entry** point for all client requests to a microservices-based application. It acts as a reverse proxy, routing incoming requests to the appropriate microservices while providing **cross-cutting** functionalities like **authentication**, **logging**, and **monitoring**.

In simpler terms, an API Gateway is like a receptionist for your microservices architecture - it directs visitors to the right department and handles common tasks that would otherwise be repeated across services.

### 👉 Why Do We Need an API Gateway?

#### Single Entry Point for Users

From a user's perspective, they're interacting with one application, not multiple microservices. Users don't care (or need to know) that we have separate quiz and question services. They just want to use the application through a single, consistent URL.

Without an API Gateway:

- Users need to know different port numbers for different services (8090 for quiz service, 8080 for question service)
- URLs change depending on which service they're accessing
- The distributed nature of the application becomes visible to users

#### Handling Cross-Cutting Concerns

API Gateway helps solve several critical challenges in microservices architecture:

1. **Authentication:** Without an API Gateway, each microservice would need to implement its own authentication, forcing users to log in multiple times as they navigate between services.
2. **Logging & Monitoring:** An API Gateway provides a centralized place to log all requests and monitor traffic patterns.
3. **Other Cross-Cutting Concerns:** Rate limiting, caching, request transformation, and other common concerns can be handled centrally.

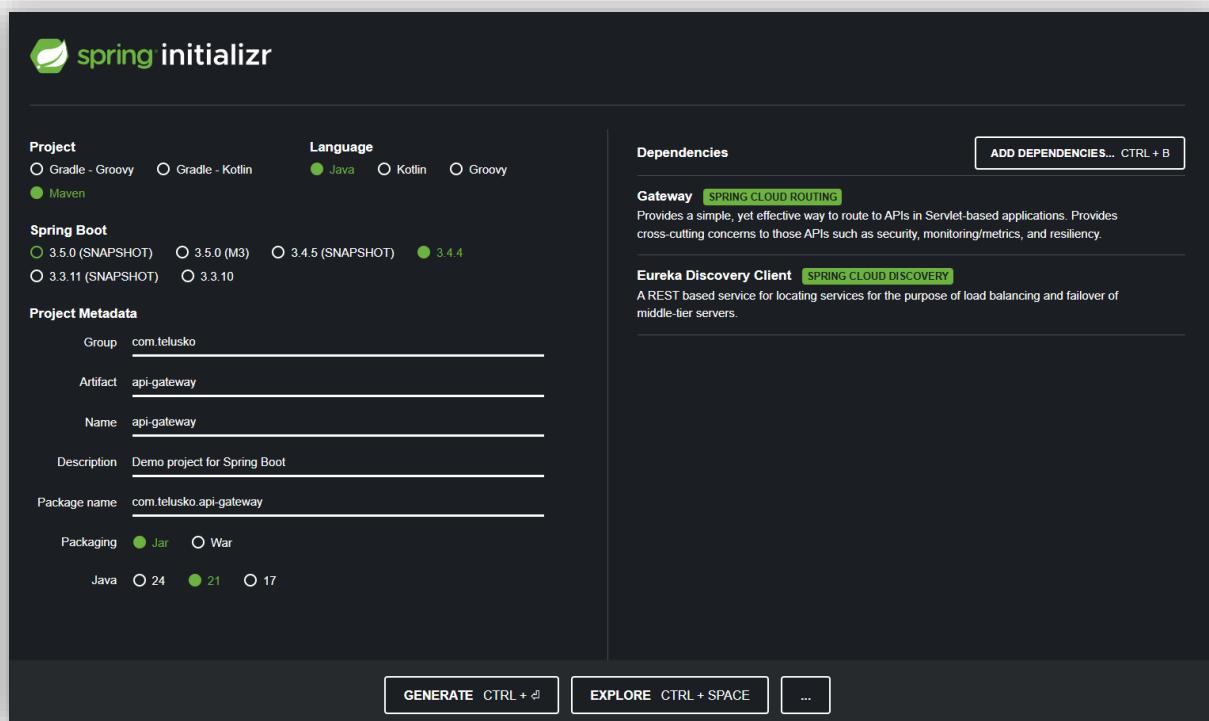
## 👉 Implementing API Gateway with Spring Cloud Gateway

Spring Cloud provides a ready-to-use API Gateway that integrates seamlessly with our microservices ecosystem.

### Setting Up the API Gateway Project

#### 1. Create a new project with Spring Initializer:

- Dependencies: Gateway and Eureka Client
- The Eureka Client dependency is necessary so the gateway can register itself with Eureka Server and discover other services



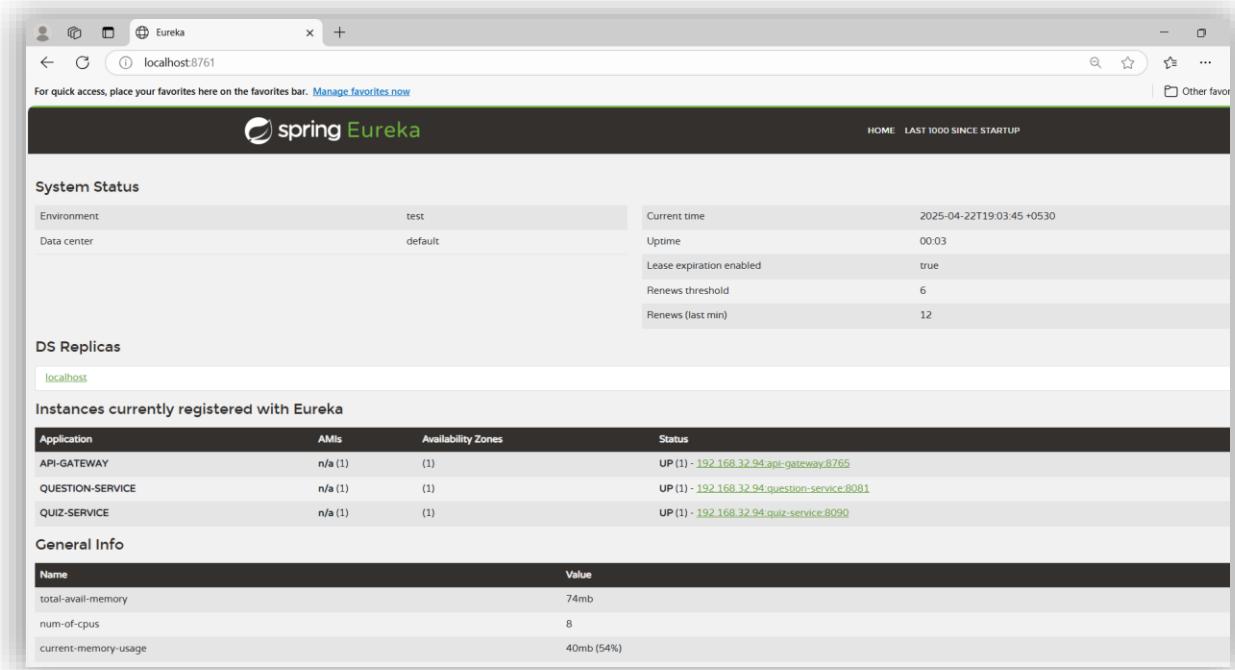
### Configuring the API Gateway

After opening the project in your IDE, configure the gateway in the `application.properties` file:

```
# Name and port for the service
spring.application.name=api-gateway
server.port=8765
```

## Verifying Gateway Registration

When we run the API Gateway, it registers itself with Eureka Server just like our other services:



The screenshot shows the Spring Eureka dashboard at [localhost:8761](http://localhost:8761). The top navigation bar includes links for Home, Last 1000 since startup, and Help. The main content area is divided into sections: System Status, DS Replicas, and General Info.

**System Status** table:

Environment	test	Current time	2025-04-22T19:03:45 +0530
Data center	default	Uptime	00:03
		Lease expiration enabled	true
		Renews threshold	6
		Renews (last min)	12

**DS Replicas** table:

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - 192.168.32.94.api-gateway.8765
QUESTION-SERVICE	n/a (1)	(1)	UP (1) - 192.168.32.94.question-service.8081
QUIZ-SERVICE	n/a (1)	(1)	UP (1) - 192.168.32.94.quiz-service.8090

**General Info** table:

Name	Value
total-avail-memory	74mb
num-of-cpus	8
current-memory-usage	40mb (54%)

## Testing the Gateway

Now we can access our services through the API Gateway:

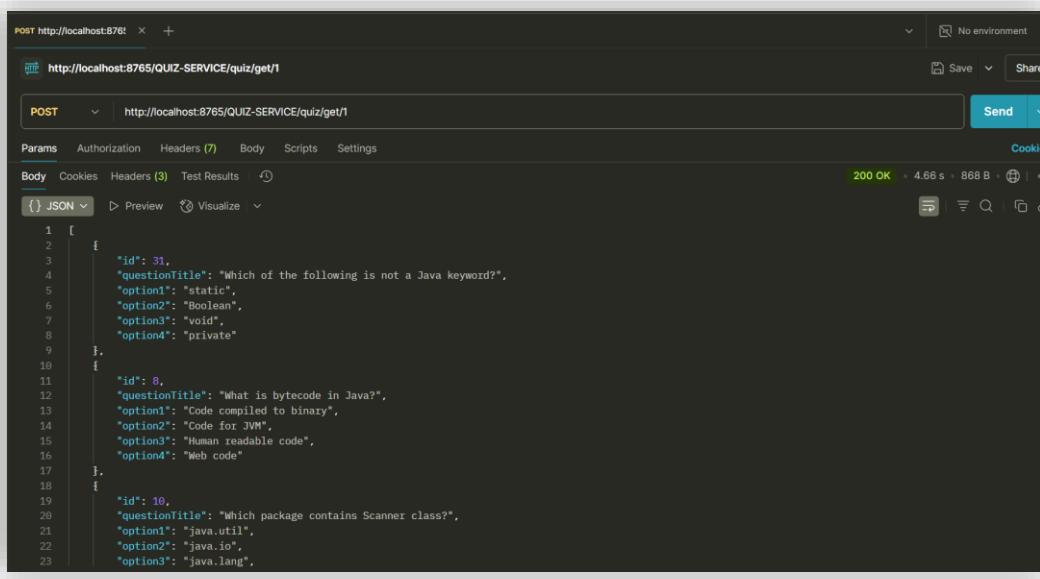
- Instead of calling <http://localhost:8090/quiz/get/2> directly
- We call <http://localhost:8765/QUIZ-SERVICE/quiz/get/2>

However, this first attempt will result in a 404 error because we need to enable the discovery locator properly.

```
# Name and port for the service
spring.application.name=api-gateway
server.port=8765

# Enable service discovery for routing
spring.cloud.gateway.discovery.locator.enabled=true
```

With this configuration, the gateway can now route requests to the appropriate services by their names:

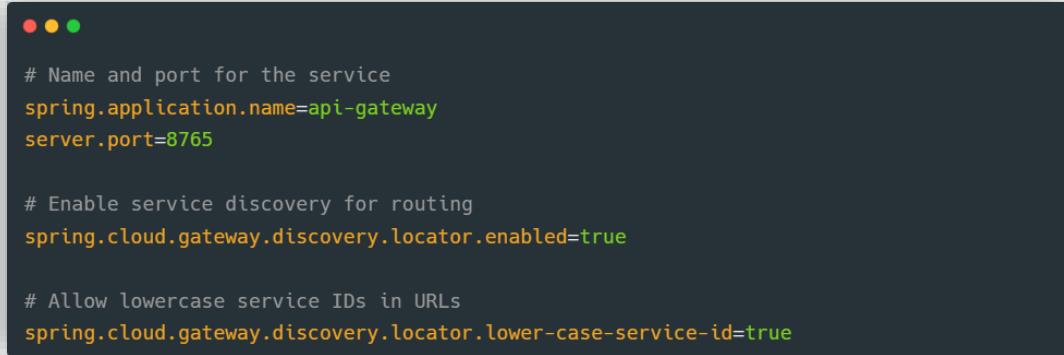


```
POST http://localhost:8765/QUIZ-SERVICE/quiz/get/1
200 OK
```

```
[{"id": 31, "questionTitle": "Which of the following is not a Java keyword?", "option1": "static", "option2": "Boolean", "option3": "void", "option4": "private"}, {"id": 8, "questionTitle": "What is bytecode in Java?", "option1": "Code compiled to binary", "option2": "Code for JVM", "option3": "Human readable code", "option4": "Web code"}, {"id": 10, "questionTitle": "Which package contains Scanner class?", "option1": "java.util", "option2": "java.io", "option3": "java.lang"}, {"id": 11, "questionTitle": "What is the difference between static and final keywords?", "option1": "final can be overridden", "option2": "final can be modified", "option3": "final is used for methods", "option4": "final is used for classes"}, {"id": 12, "questionTitle": "What is the purpose of the finally block?", "option1": "To catch exceptions", "option2": "To release resources", "option3": "To execute code", "option4": "To terminate the program"}, {"id": 13, "questionTitle": "What is the difference between System.out.println() and System.out.print()?", "option1": "System.out.println() adds a new line", "option2": "System.out.print() adds a new line", "option3": "System.out.println() adds a space", "option4": "System.out.print() adds a space"}, {"id": 14, "questionTitle": "What is the difference between String and StringBuilder?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 15, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 16, "questionTitle": "What is the difference between String and StringBuilder?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 17, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 18, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 19, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 20, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 21, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 22, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 23, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}]
```

## Improving User Experience with Lowercase Service IDs

The default configuration requires uppercase service names (e.g., **QUIZ-SERVICE**), which isn't user-friendly. We can improve this by enabling lowercase service IDs:

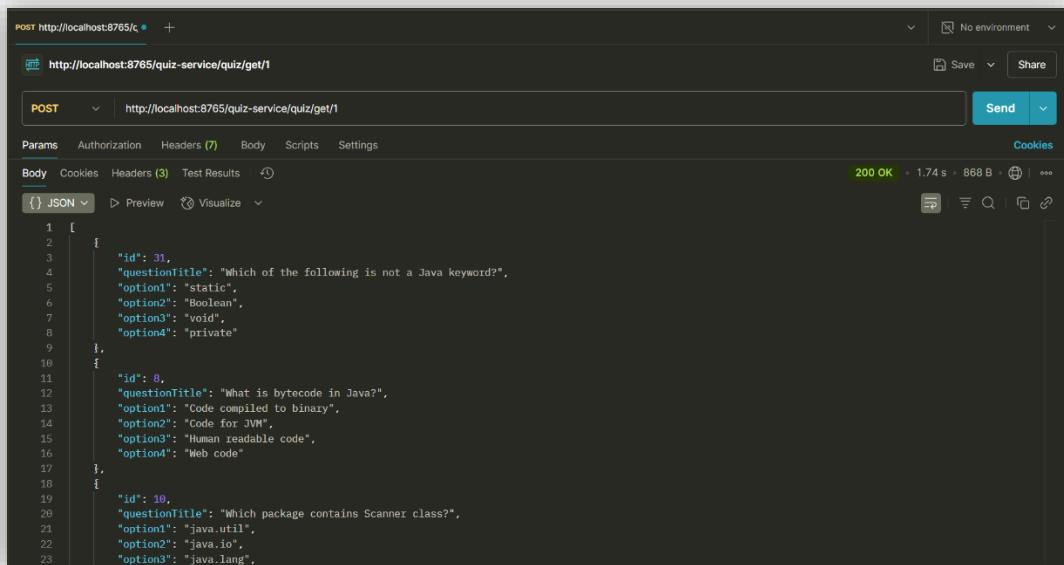


```
# Name and port for the service
spring.application.name=api-gateway
server.port=8765

# Enable service discovery for routing
spring.cloud.gateway.discovery.locator.enabled=true

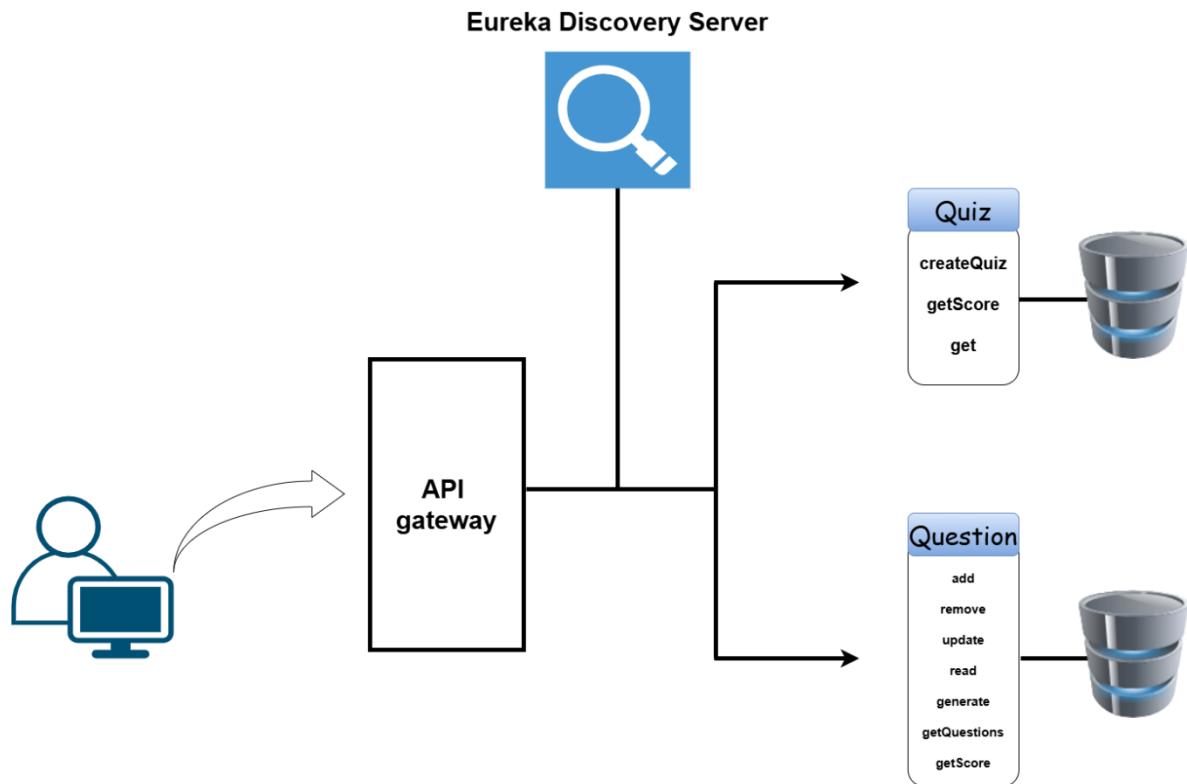
# Allow lowercase service IDs in URLs
spring.cloud.gateway.discovery.locator.lower-case-service-id=true
```

Now we can use more natural URLs:



```
POST http://localhost:8765/quiz-service/quiz/get/1
200 OK
```

```
[{"id": 31, "questionTitle": "Which of the following is not a Java keyword?", "option1": "static", "option2": "Boolean", "option3": "void", "option4": "private"}, {"id": 8, "questionTitle": "What is bytecode in Java?", "option1": "Code compiled to binary", "option2": "Code for JVM", "option3": "Human readable code", "option4": "Web code"}, {"id": 10, "questionTitle": "Which package contains Scanner class?", "option1": "java.util", "option2": "java.io", "option3": "java.lang"}, {"id": 11, "questionTitle": "What is the difference between static and final keywords?", "option1": "final can be overridden", "option2": "final can be modified", "option3": "final is used for methods", "option4": "final is used for classes"}, {"id": 12, "questionTitle": "What is the purpose of the finally block?", "option1": "To catch exceptions", "option2": "To release resources", "option3": "To execute code", "option4": "To terminate the program"}, {"id": 13, "questionTitle": "What is the difference between System.out.println() and System.out.print()?", "option1": "System.out.println() adds a new line", "option2": "System.out.print() adds a new line", "option3": "System.out.println() adds a space", "option4": "System.out.print() adds a space"}, {"id": 14, "questionTitle": "What is the difference between String and StringBuilder?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 15, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 16, "questionTitle": "What is the difference between String and StringBuilder?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 17, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 18, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 19, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 20, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 21, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 22, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}, {"id": 23, "questionTitle": "What is the difference between String and StringBuffer?", "option1": "String is immutable", "option2": "String is mutable", "option3": "String is thread-safe", "option4": "String is not thread-safe"}]
```



### The Complete Architecture with API Gateway

With the API Gateway in place, our architecture now follows the industry-standard pattern for microservices:

1. **Clients** send all requests to the API Gateway
2. **API Gateway** routes requests to the appropriate service based on the URL
3. **Microservices** handle specific functionality and communicate with each other as needed
4. **Service Registry** keeps track of all available service instances

This approach provides:

- A simplified experience for clients (single entry point)
- Centralized handling of common concerns
- Abstraction of the underlying microservices structure
- Flexibility to evolve the microservices architecture without impacting clients