**Spring Cloud**

Now lets us come to the Spring Cloud area. As discussed Spring Boot is used to write Standalone Java web applications. Then comes the question of how do I configure it to use with the external systems or components? The answer to this is the Spring Cloud. Yes, you heard that right.

Spring Cloud provides the following functionalities which are very crucial for the modern distributed systems

**1. Distributed/versioned configuration**

* A centralized configuration server to maintain the application configuration
* This can be maintained separately compared to the code repo and is easily manageable

2. **Service registration and discovery**

* All the services will register their identity to a registry server and the famous one is Eureka.
* The **clients** that connect to the central server can update and retrieve the address.

**3. Service-to-Service calls**

The services can discover each other to communicate using the following steps

* Registering the service
* Fetching the Registry
* Finding the downstream service
* Resolving the Underlying IP address
* Call the rest Endpoint using RestTemplate

**4. Load balancing**

* The objective of load balancing is to maximize throughput, minimize response time, increase efficiency, and optimize resource uses.
* Using multiple components with load balancing may increase **reliability** and **availability** through redundancy.
* Eureka dependency has an inbuilt load balancer called **Ribbon**which balances the load between the instances of a single service

**5. Circuit Breakers**

* Circuit breakers are used as a fallback mechanism when an API call to another microservice / another system fails.
* They enable opening and closing the circuit and what to do in case of failure. The circuit breaker handles these failures gracefully.
* The widely used circuit breaker for Spring Boot is **resilience4j**

**5. API Gateway**

Spring Cloud Gateway is the API gateway designed for the Spring boot application. This is part of the Spring Cloud. It has the following features

* Built on Spring framework 5, project reactor and Spring Boot 2.0
* Able to match routes on any requested attribute
* Predicates and filters are specific to routes
* Circuit Breaker integration
* Spring Cloud Discovery Client integration
* Easy to write Predicates and filters
* Request Rate Limiting
* Path rewriting

**6. Spring Cloud Security**

* Spring Security integration to provide Authentication and Authorization
* Support for JWT Token Validation
* Spring Cloud Security offers a set of primitives for building secure applications using Spring Boot and Spring Security OAuth2
* We can quickly create systems that implement common patterns like single sign-on, token relay, and token exchange.

**7. Distributed Tracing**

* Tracing is very useful in microservices because a request from the client can travel across multiple microservices. It is essential for developers to see the state of the request across the services and Spring Cloud has an answer for this by proving the following dependency for this
* We can add **Spring Cloud Sleuth** library to our project to enable tracing. Sleuth is responsible for recording **timing**, and also generates the request-id which is used for **latency analysis**. With this request-id, we can trace the request across services
* **Zipkin** is a distributed tracing tool specially designed for **analyzing latency problems** inside the microservice architecture. It exposes the HTTP endpoint used for collecting input data. If we are required to add tracing to our project, we should add the **spring-cloud-starter-Zipkin** dependency.

**8. Distributed Messaging**

* Spring Cloud Bus links nodes of a distributed system with a lightweight message broker. This can then be used to broadcast state changes in an application using AMQP and other messaging protocols.
* This is a very common use case in microservices for async communication by pushing msgs to RabbitMQ, [Kafka](https://javarevisited.blogspot.com/2018/04/top-5-apache-kafka-course-to-learn.html), AWS SQS, etc.
* Spring Cloud Stream is another framework in Spring Cloud for building highly scalable event-driven microservices connected with shared messaging systems. This is used for stream processing

As we saw above, Spring Cloud has a lot of functionalities for the communication between distributed systems and we can include the required dependencies and use it very easily.

In a nutshell, Spring Boot is an application used to develop a stand-alone Microservice application and Spring Cloud is used to configure the communication between those services. Hence both are needed to accomplish a production-grade microservices system.

# What is Spring Cloud Gateway?

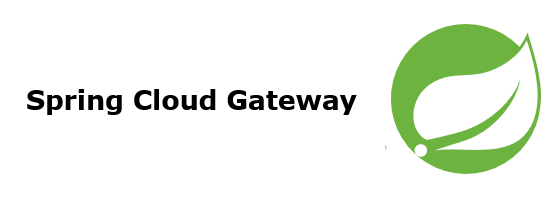
Hello everyone. This article is a continuation of the previous article

## [What is Spring Cloud and how it is different from Spring and Spring Boot?](https://blog.devgenius.io/what-is-spring-cloud-and-how-it-is-different-from-spring-and-spring-boot-128d276a1432" \t "_blank)

### **[Hello everyone. Let us discuss Spring Cloud in this article. There are a lot of terms like Spring, Spring Boot, Spring…](https://blog.devgenius.io/what-is-spring-cloud-and-how-it-is-different-from-spring-and-spring-boot-128d276a1432" \t "_blank)**

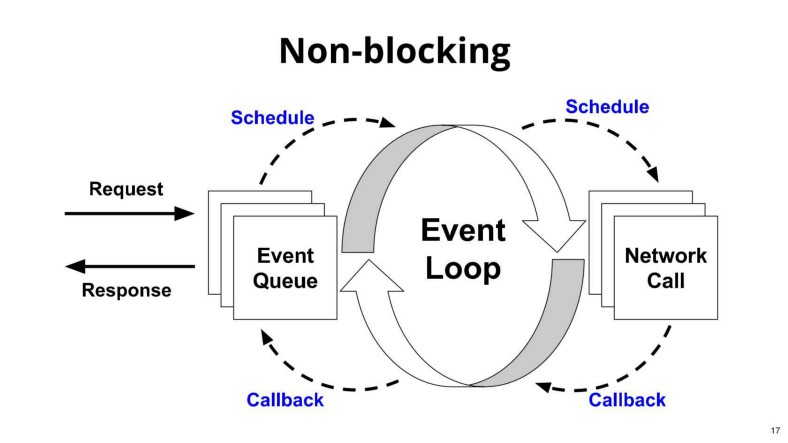
[blog.devgenius.io](https://blog.devgenius.io/what-is-spring-cloud-and-how-it-is-different-from-spring-and-spring-boot-128d276a1432" \t "_blank)

If you have not read it, please read that and come back here to have a continuation.

[[](https://javarevisited.blogspot.com/2018/04/top-5-spring-cloud-courses-for-java.html)](https://javarevisited.blogspot.com/2018/04/top-5-spring-cloud-courses-for-java.html)

Spring Cloud Gateway

Spring Cloud Gateway is an [API Gateway](https://javarevisited.blogspot.com/2021/09/microservices-design-patterns-principles.html) / Backend For the FrontEnd (BFF) framework. It is based on [Spring 5](https://medium.com/javarevisited/top-10-free-courses-to-learn-spring-framework-for-java-developers-639db9348d25), [Spring Boot 2](https://medium.com/javarevisited/top-10-courses-to-learn-spring-boot-in-2020-best-of-lot-6ffce88a1b6e), and Project Reactor / Webflux and works on a **non-blocking API model.**Basically, it is a Spring Webflux application and I will write another article on [**Spring Webflux**](https://medium.com/javarevisited/7-best-webflux-and-reactive-spring-boot-courses-for-java-programmers-33b7c6fa8995) to explain it in depth.

[[](https://javarevisited.blogspot.com/2021/09/microservices-design-patterns-principles.html)](https://javarevisited.blogspot.com/2021/09/microservices-design-patterns-principles.html)

When using a non-blocking API model, a thread is always available to process the incoming request. These requests are then processed asynchronously in the background and once completed the response is returned.

This will save a lot of overhead compared to the traditional blocking servers where all the incoming requests will be waiting until the response. These are the following dependencies required to build a [Spring Cloud API Gateway](https://www.java67.com/2021/01/spring-cloud-interview-questions-with-answers-java.html) application.

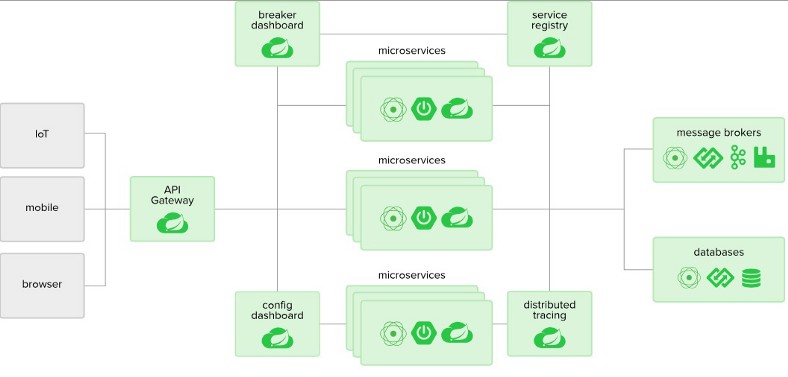
<dependency> <groupId>org.springframework.cloud</groupId> <artifactId>spring-cloud-starter-gateway</artifactId></dependency><dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-webflux</artifactId></dependency><dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka- client</artifactId>  
</dependency>

The Spring Cloud Gateway uses **Netty** as the embedded server instead of the [Tomcat server](https://www.java67.com/2019/07/spring-boot-3-ways-to-change-port-of-tomcat.html) because**Tomcat is based on the blocking model** and **Netty is based on the non-blocking model.**

**The functionality of Spring Cloud Gateway**

* Built on [Spring framework 5](https://medium.com/javarevisited/top-10-pluralsight-courses-to-learn-spring-framework-for-java-developers-3d35c4a1dc2), project reactor and [Spring Boot 2.0](https://medium.com/javarevisited/10-free-spring-boot-tutorials-and-courses-for-java-developers-53dfe084587e)
* Able to match routes on any requested attribute
* Predicates and filters are specific to routes
* Circuit Breaker integration
* Spring Cloud Discovery Client integration
* Easy to write Predicates and filters
* Request Rate Limiting
* Path rewriting
* Spring Security Integration

[Spring Cloud gateway](https://javarevisited.blogspot.com/2022/06/spring-cloud-interview-questions-answers.html) works very well with the Spring ecosystem and is an obvious choice for the microservices developed using the Spring framework. Moreover, it acts as a single facade layer to any type of client like mobile, web, etc.

[[](https://javarevisited.blogspot.com/2022/06/spring-cloud-interview-questions-answers.html)](https://javarevisited.blogspot.com/2022/06/spring-cloud-interview-questions-answers.html)

As seen from the above diagram, all the front-end clients talk to a single component and Spring Cloud Gateway takes care of routing the request to the respective microservice and gets the response to the client. In a cloud environment, Spring Cloud Gateway runs in the Public subnet (public IP) and all other microservices run in the private subnet (private IP).

Spring Cloud gateway is also a discovery client along with other microservices and hence it discovers the [microservice](https://medium.com/javarevisited/10-best-java-microservices-courses-with-spring-boot-and-spring-cloud-6d04556bdfed)from the service discovery registry and routes to those services using its private IP. By doing so, we are reducing the risk of exposing the entire microservices architecture to the public.

Spring Cloud Gateway can also handle Authentication and Authorization. Authentication is done by integrating [Spring Security](https://medium.com/javarevisited/top-10-courses-to-learn-spring-security-and-oauth2-with-spring-boot-for-java-developers-8f0222d6066d). Authorization of the JWT tokens can be done here at a centralized place thereby reducing the overhead for every other microservice.

We can also route in the RouteLocator bean java file or as the configuration in the property file. These routes will decide where to send the request when the incoming request pattern is matched. For every request, rate limiters can be applied specifically as well.

**Global Filters — Pre and Post Filter**

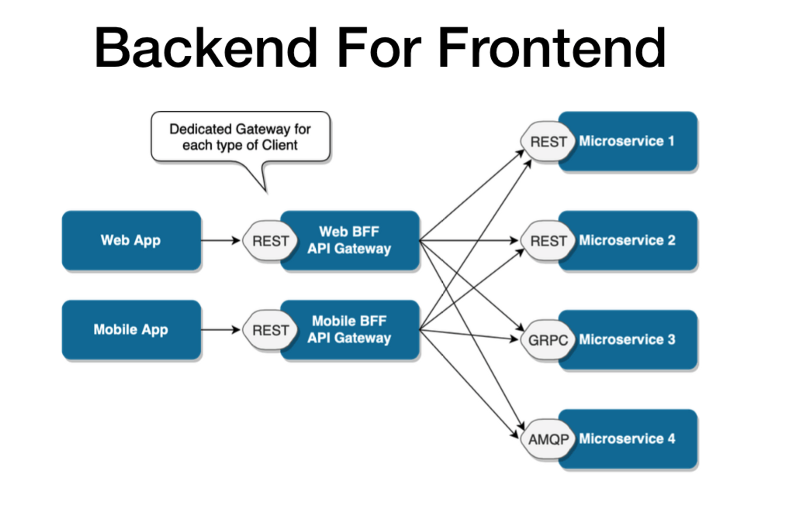
Global filters are executed for every route defined in the API [Gateway](https://blog.knoldus.com/spring-cloud-api-gateway/). The main difference between pre-filter and post-filter classes is that the pre-filter code is executed before Spring Cloud API Gateway routes the request to a destination web service endpoint.

While the post-filter code will be executed after Spring Cloud API Gateway has routed the HTTP request to a destination web service endpoint. Pre-Filter is usually used to do some [Authentication](https://javarevisited.blogspot.com/2018/01/how-to-enable-http-basic-authentication-spring-security-java-xml-configuration.html)/ [Authorization](https://javarevisited.blogspot.com/2013/07/role-based-access-control-using-spring-security-ldap-authorities-mapping-mvc.html)for all requests before they are routed to the specific microservice. Post-Filter is used to alter the response like decorating the response and send to the front end.

**What is Backend for FrontEnd (BFF) Pattern?**

The BFF is a design pattern by which every client will talk to its own API gateway instead of the common gateway. In the diagram above, there is a **single Spring Cloud API gateway application** that is shared by three different clients namely [IoT](https://medium.com/javarevisited/my-favorite-courses-to-learn-internet-of-things-iot-in-2020-best-of-lot-8517aa9fc838), Mobile, and Web.

If we use a **specific API Gateway**for each of these clients, then there will be different Gateway applications running in the public subnet. These API gateways are collectively called **Backend for FrontEnd**

[[](https://javarevisited.blogspot.com/2019/01/top-5-online-courses-to-become-web-developer.html)](https://javarevisited.blogspot.com/2019/01/top-5-online-courses-to-become-web-developer.html)

**AWS API Gateway vs Spring Cloud Gateway**

There are a lot of API gateways available in the market like Spring Cloud API Gateway, Zuul 2, Apigee, Kong, AWS API Gateway and so on. But I would like to answer the commonly asked question. Which one I should use **AWS API Gateway or Spring Cloud Gateway?**

The answer is

Use AWS API Gateway if you need following

* [Serverless Infrastructure](https://medium.com/javarevisited/7-best-serverless-and-aws-lambda-courses-to-learn-in-2021-de1820111c85)
* Easy and quick to deploy APIs
* Simple Routing rules
* Quicker Request Response

Use Spring Cloud Gateway for the following

* If you need granular control on your API Gateway
* If you want to add custom filters and complex routes
* If your API takes a long time to respond. AWS API Gateway will timeout if the response takes more than 29 seconds per request
* If you use the Spring ecosystem, Spring Cloud Gateway will fit in well