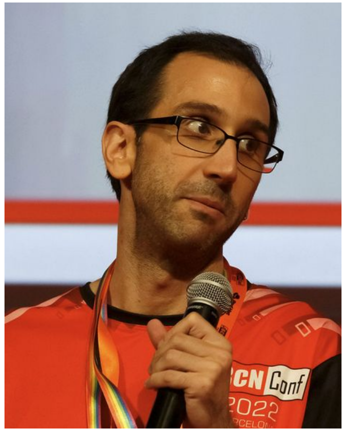
# Beyond routing 🚀 Spring Cloud Gateway with style

Marta Medio Menéndez
@tpdeoro

Abel Salgado Romero
@abelsromero

## Who are we?

- Senior Software Engineer(s) @ VMware
- Working with Spring Cloud + Kubernetes



Barcelona JUG
Asciidoctor Contributor
Pedantic Linux User

Coding from Asturias
❤ Open Source
Non-Pedantic Sommelier

## Why are we here?

Goals

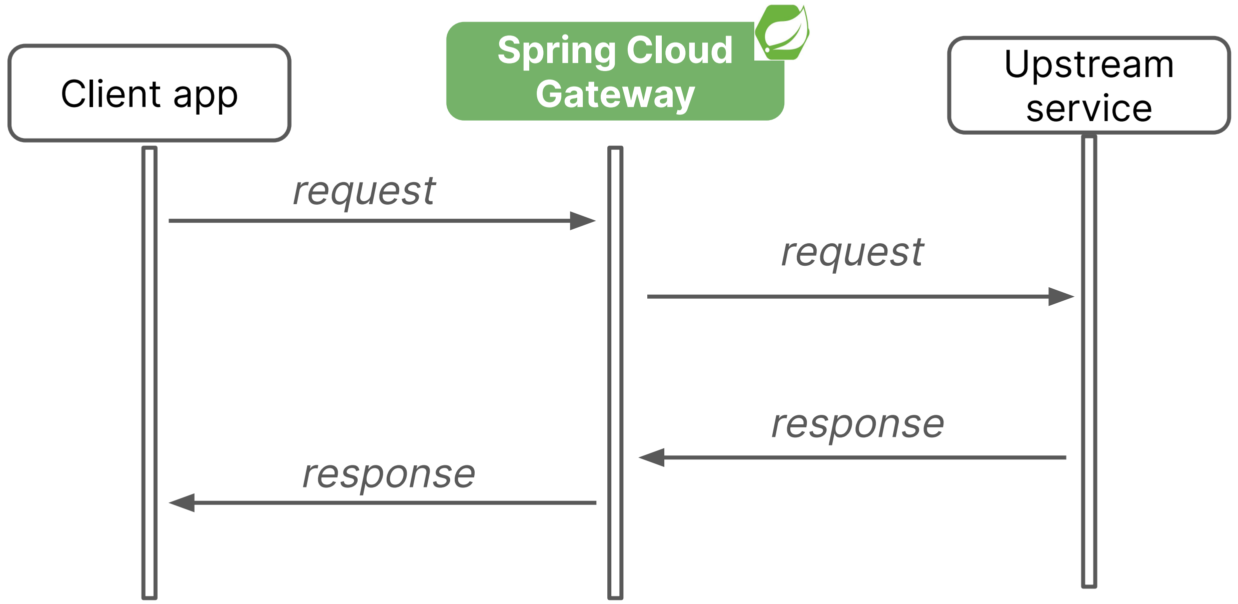
- Learn a few tricks
- Know how to exploit Spring Cloud Gateway’s potential
- Have fun

What is it not

- 101 on Spring Cloud Gateway

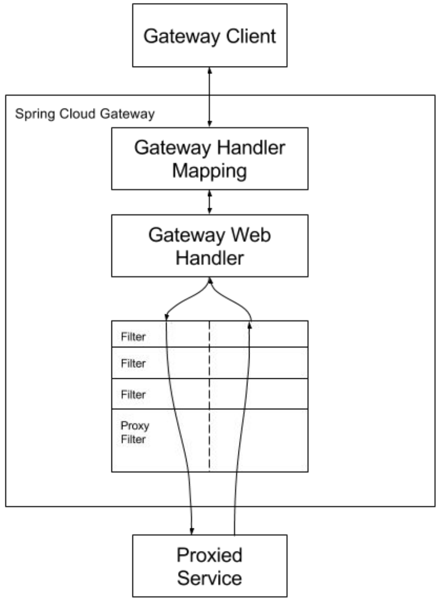
## How does Spring Cloud Gateway work?

The gateway is nothing else than a *http_proxy* (on steroids)



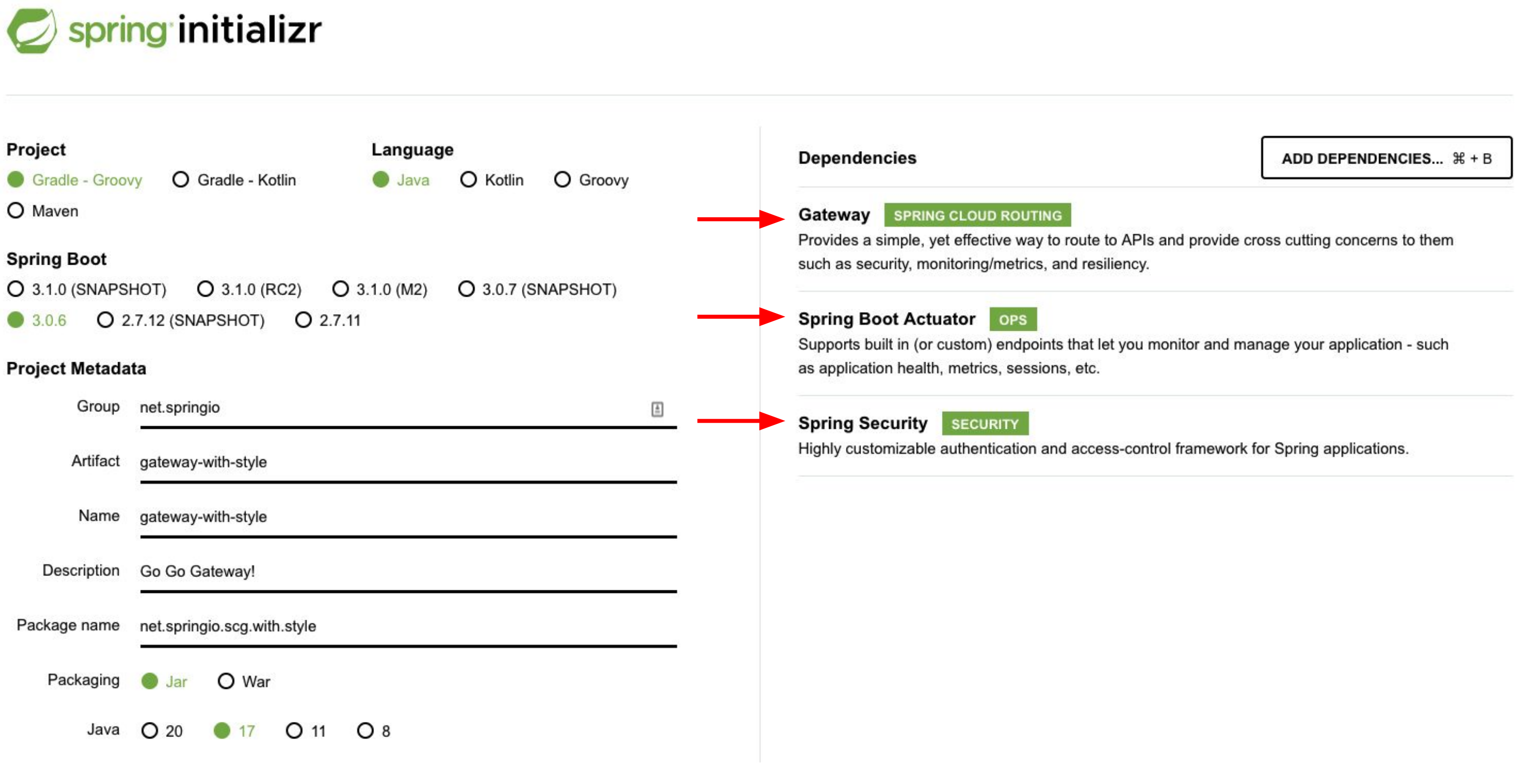
## How does Spring Cloud Gateway work?

**Filters** act on requests and responses through a **chain**



## Lets go!

# What we need to begin?



spring initializr

**Project**

[✓ Gradle - Groovy] Gradle - Kotlin Maven

**Language**

[✓ Java] Kotlin Groovy

**Spring Boot**

3.1.0 (SNAPSHOT) 3.1.0 (RC2) 3.1.0 (M2) 3.0.7 (SNAPSHOT) [✓ 3.0.6] 2.7.12 (SNAPSHOT) 2.7.11

**Project Metadata**

| | |
|---|---|
| Group | net.springio |
| Artifact | gateway-with-style |
| Name | gateway-with-style |
| Description | Go Go Gateway! |
| Package name | net.springio.scg.with.style |
| Packaging | [✓ Jar] War |
| Java | 20 [✓ 17] 11 8 |

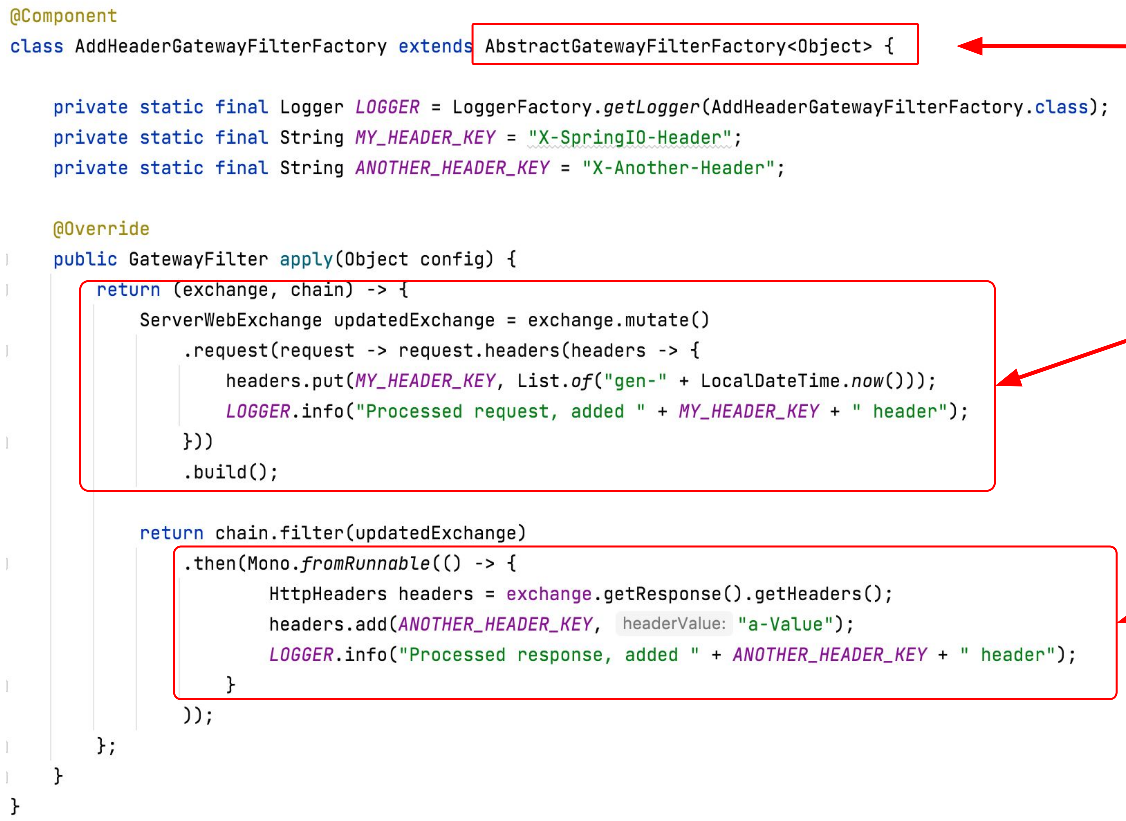
**Dependencies**

**Gateway** SPRING CLOUD ROUTING
Provides a simple, yet effective way to route to APIs and provide cross cutting concerns to them such as security, monitoring/metrics, and resiliency.

**Spring Boot Actuator** OPS
Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

**Spring Security** SECURITY
Highly customizable authentication and access-control framework for Spring applications.

# Our first filter!



```
@Component
class AddHeaderGatewayFilterFactory extends AbstractGatewayFilterFactory<Object> {

    private static final Logger LOGGER = LoggerFactory.getLogger(AddHeaderGatewayFilterFactory.class);
    private static final String MY_HEADER_KEY = "X-SpringIO-Header";
    private static final String ANOTHER_HEADER_KEY = "X-Another-Header";

    @Override
    public GatewayFilter apply(Object config) {
        return (exchange, chain) -> {
            ServerWebExchange updatedExchange = exchange.mutate()
                .request(request -> request.headers(headers -> {
                    headers.put(MY_HEADER_KEY, List.of("gen-" + LocalDateTime.now()));
                    LOGGER.info("Processed request, added " + MY_HEADER_KEY + " header");
                }))
                .build();

            return chain.filter(updatedExchange)
                .then(Mono.fromRunnable(() -> {
                        HttpHeaders headers = exchange.getResponse().getHeaders();
                        headers.add(ANOTHER_HEADER_KEY, headerValue: "a-Value");
                        LOGGER.info("Processed response, added " + ANOTHER_HEADER_KEY + " header");
                    }
                ));
        };
    }
}
```

Gateway hook

Mutate the Request

Modify the Response

## Our first filter!

Not really hard so far...

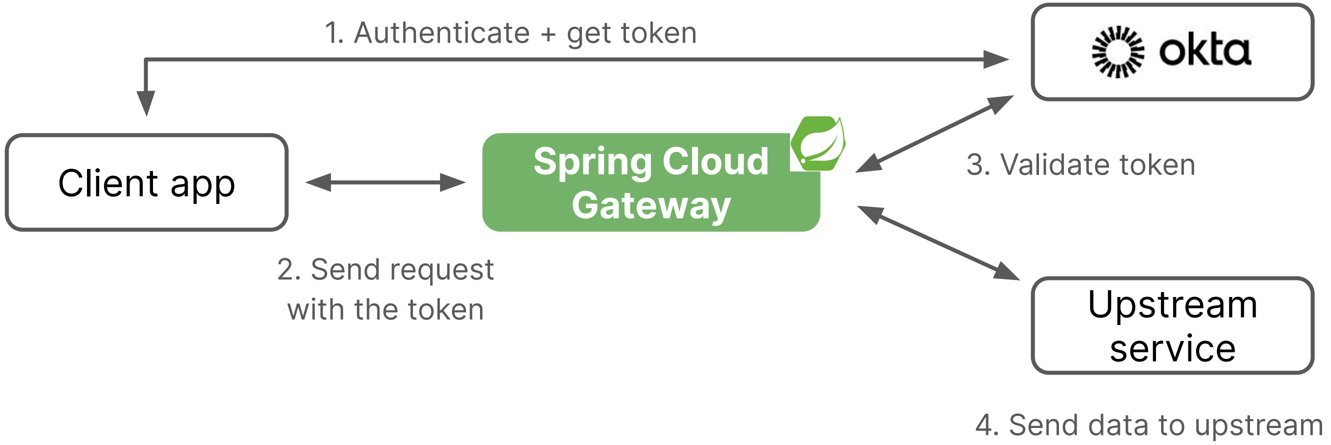
## Let's see our second filter

Raising the stakes...

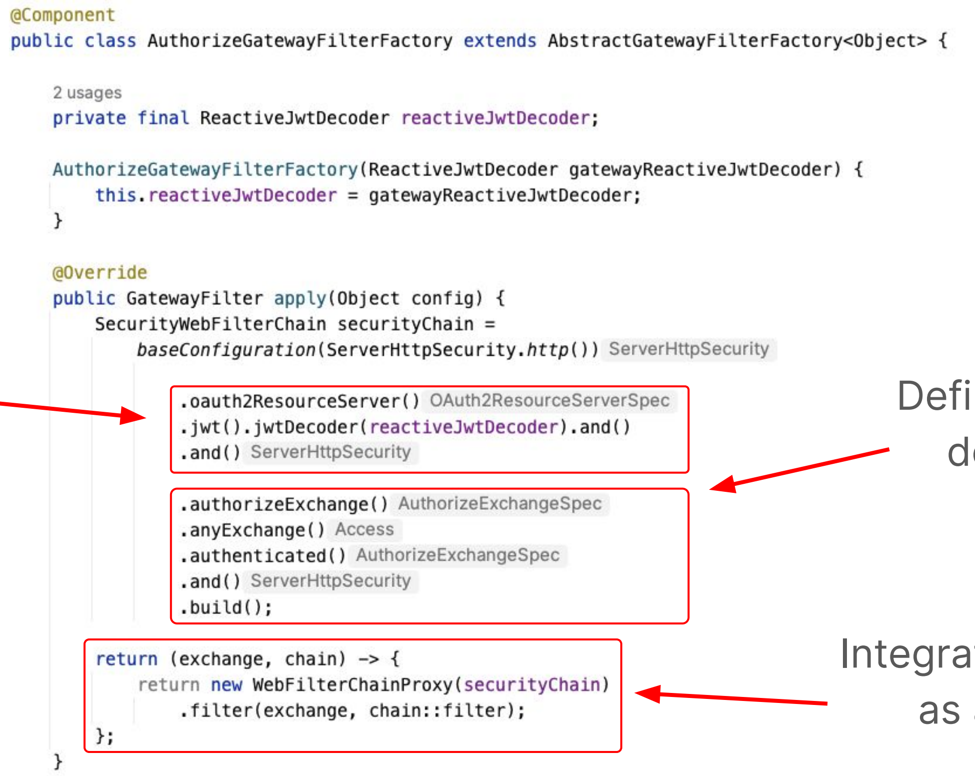
- Can we secure our app with OAuth?
- And extract the information from the token?
- And adapt it for an upstream that doesn't talk OAuth?

# Let's see our second filter: Spring Security Filters!

Authorizing OAuth requests



## Let's see our second filter: Validate Token



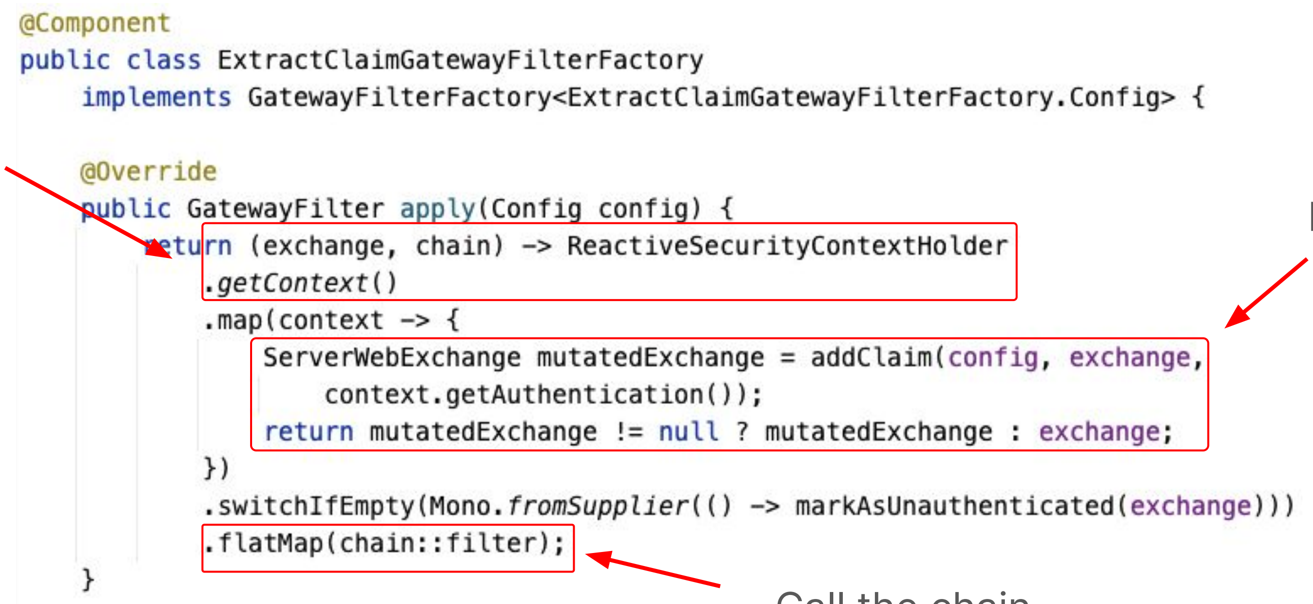
```java
@Component
public class AuthorizeGatewayFilterFactory extends AbstractGatewayFilterFactory<Object> {

    2 usages
    private final ReactiveJwtDecoder reactiveJwtDecoder;

    AuthorizeGatewayFilterFactory(ReactiveJwtDecoder gatewayReactiveJwtDecoder) {
        this.reactiveJwtDecoder = gatewayReactiveJwtDecoder;
    }

    @Override
    public GatewayFilter apply(Object config) {
        SecurityWebFilterChain securityChain =
            baseConfiguration(ServerHttpSecurity.http()) ServerHttpSecurity

                .oauth2ResourceServer() OAuth2ResourceServerSpec
                .jwt().jwtDecoder(reactiveJwtDecoder).and()
                .and() ServerHttpSecurity

                .authorizeExchange() AuthorizeExchangeSpec
                .anyExchange() Access
                .authenticated() AuthorizeExchangeSpec
                .and() ServerHttpSecurity
                .build();

        return (exchange, chain) -> {
            return new WebFilterChainProxy(securityChain)
                .filter(exchange, chain::filter);
        };
    }
```

Enable token processing

Define what to do with it

Integrate the Security filter as a Gateway filter

# Let's see our second filter: Send data to upstream



```java
@Component
public class ExtractClaimGatewayFilterFactory
    implements GatewayFilterFactory<ExtractClaimGatewayFilterFactory.Config> {

    @Override
    public GatewayFilter apply(Config config) {
        return (exchange, chain) -> ReactiveSecurityContextHolder
                .getContext()
                .map(context -> {
                    ServerWebExchange mutatedExchange = addClaim(config, exchange,
                        context.getAuthentication());
                    return mutatedExchange != null ? mutatedExchange : exchange;
                })
                .switchIfEmpty(Mono.fromSupplier(() -> markAsUnauthenticated(exchange)))
                .flatMap(chain::filter);
    }
```

Get the Security Context

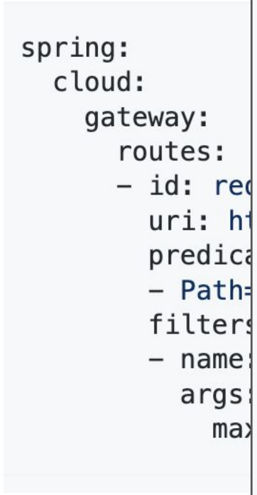
Mutate the request

Call the chain

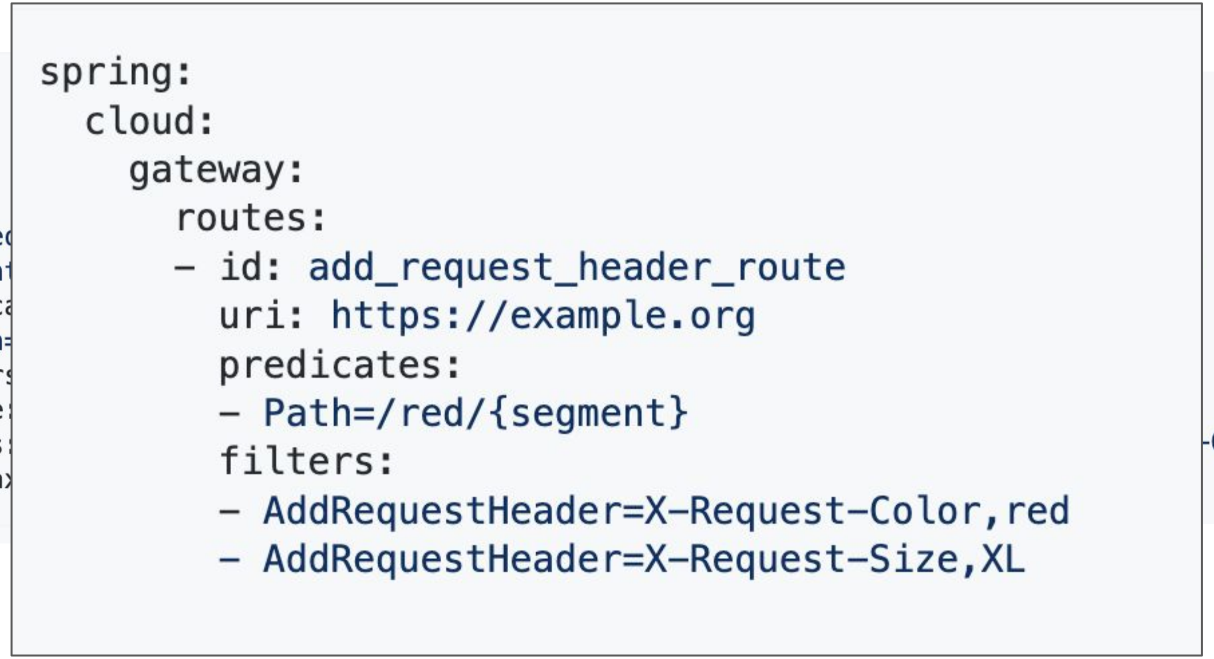
## Hacking Security with a couple of filters!

# Let's talk about configuration

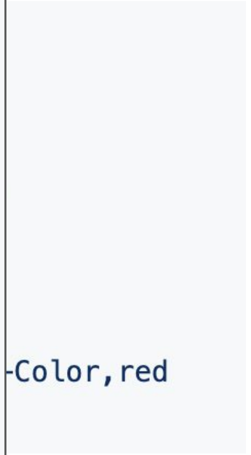
Common configuration of filters



```
spring:
  cloud:
    gateway:
      routes:
      - id: re
        uri: h
        predic
        - Path
        filter
        - name
          args
            ma
```



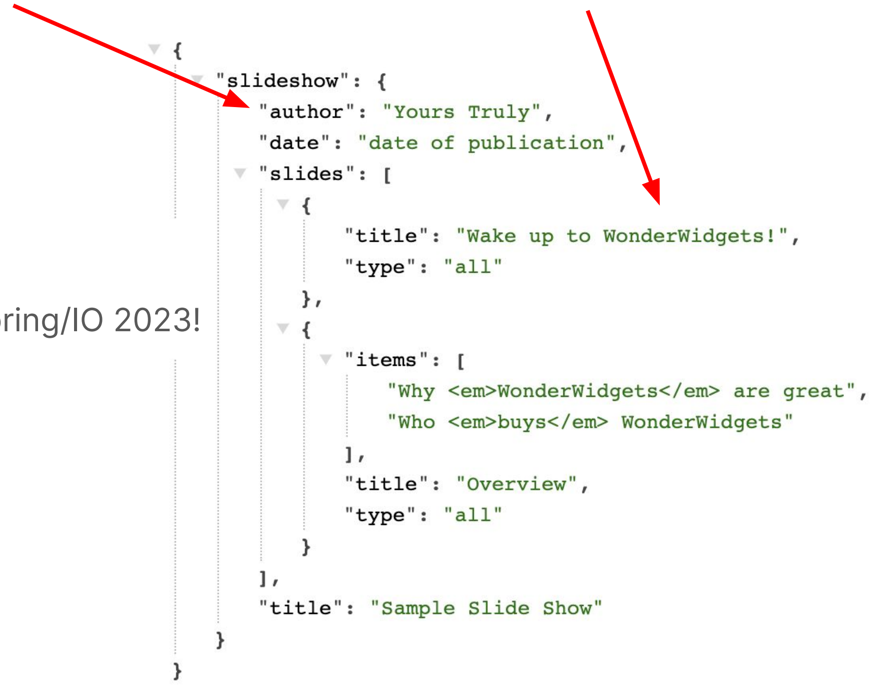
```
spring:
  cloud:
    gateway:
      routes:
      - id: add_request_header_route
        uri: https://example.org
        predicates:
        - Path=/red/{segment}
        filters:
        - AddRequestHeader=X-Request-Color,red
        - AddRequestHeader=X-Request-Size,XL
```



```
-Color,red
```

## Let's talk about configuration

RewriteJsonResponseBody=slideshow.author:Marta,slideshow.slides[0].title:Spring/IO 2023!

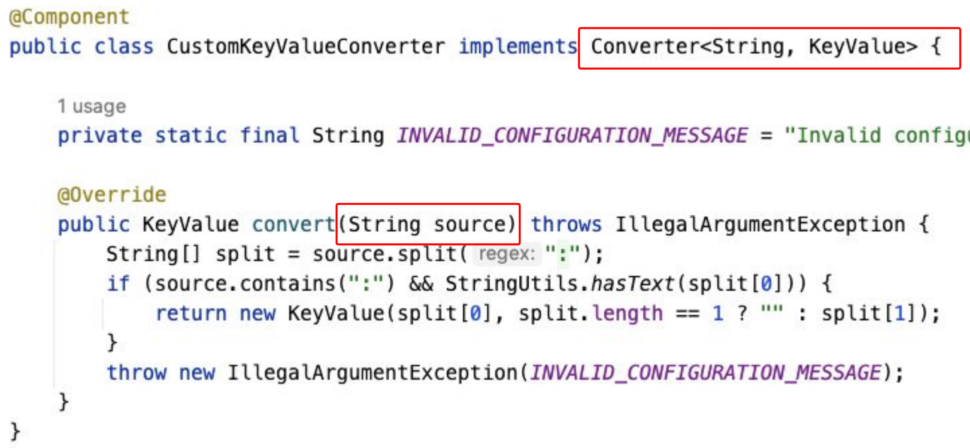


```
{
  "slideshow": {
    "author": "Yours Truly",
    "date": "date of publication",
    "slides": [
      {
        "title": "Wake up to WonderWidgets!",
        "type": "all"
      },
      {
        "items": [
          "Why <em>WonderWidgets</em> are great",
          "Who <em>buys</em> WonderWidgets"
        ],
        "title": "Overview",
        "type": "all"
      }
    ],
    "title": "Sample Slide Show"
  }
}
```

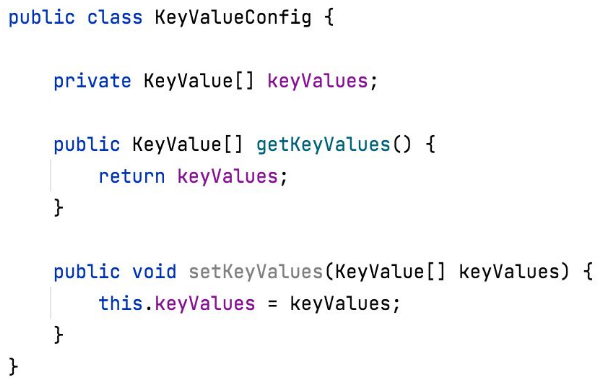
RewriteJsonResponseBody=slideshow.author,Marta

RewriteJsonResponseBody=slideshow.slides[0].title,Spring/IO 2023!

# Let's talk about configuration: Custom Converters!

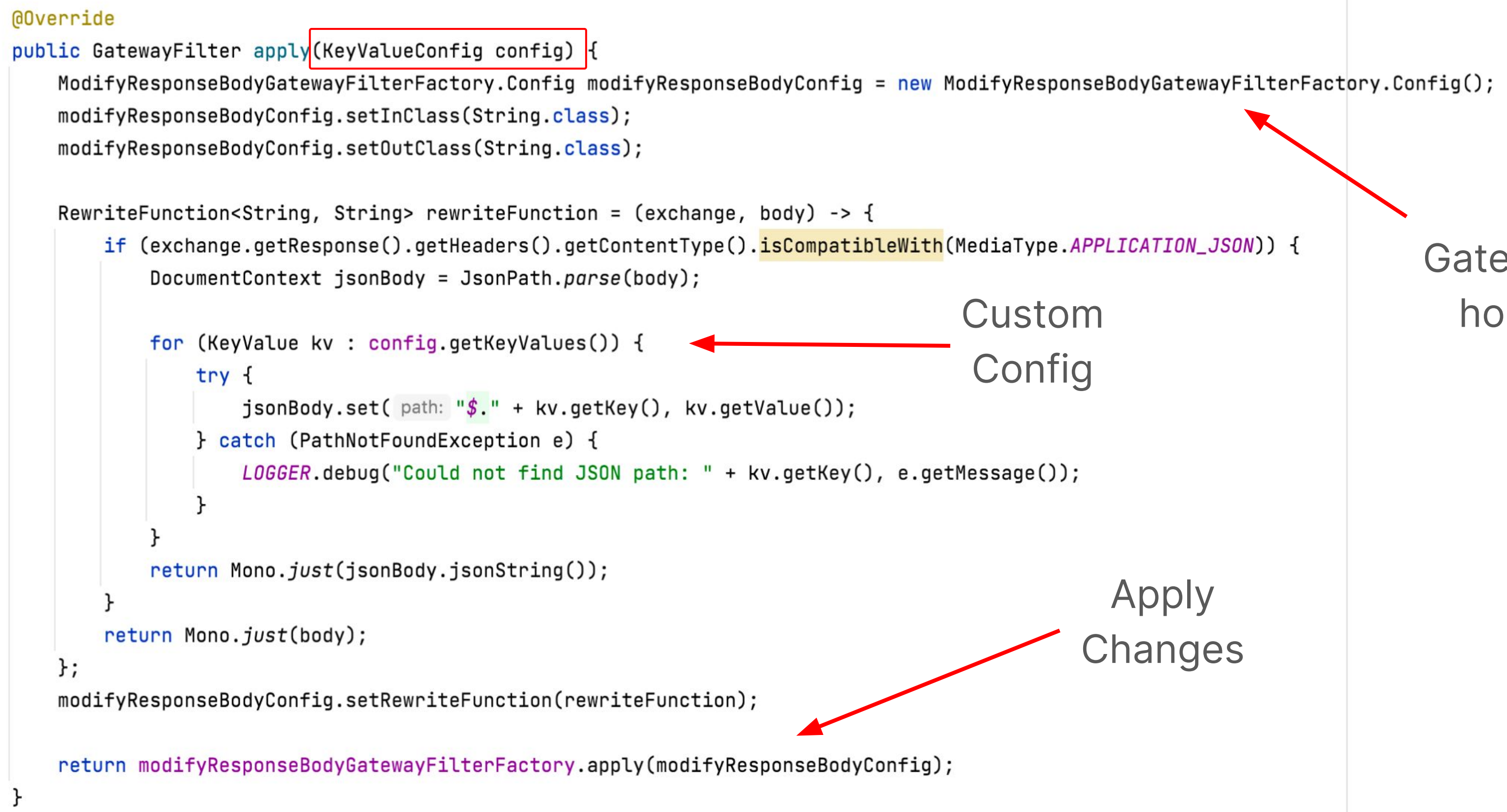


```java
@Component
public class CustomKeyValueConverter implements Converter<String, KeyValue> {

    1 usage
    private static final String INVALID_CONFIGURATION_MESSAGE = "Invalid configuration, expected format: 'key:value'";

    @Override
    public KeyValue convert(String source) throws IllegalArgumentException {
        String[] split = source.split( regex: ":");
        if (source.contains(":") && StringUtils.hasText(split[0])) {
            return new KeyValue(split[0], split.length == 1 ? "" : split[1]);
        }
        throw new IllegalArgumentException(INVALID_CONFIGURATION_MESSAGE);
    }
}
```



```java
public class KeyValueConfig {

    private KeyValue[] keyValues;

    public KeyValue[] getKeyValues() {
        return keyValues;
    }

    public void setKeyValues(KeyValue[] keyValues) {
        this.keyValues = keyValues;
    }
}
```

# Modifying the Response content



```java
@Override
public GatewayFilter apply(KeyValueConfig config) {
    ModifyResponseBodyGatewayFilterFactory.Config modifyResponseBodyConfig = new ModifyResponseBodyGatewayFilterFactory.Config();
    modifyResponseBodyConfig.setInClass(String.class);
    modifyResponseBodyConfig.setOutClass(String.class);

    RewriteFunction<String, String> rewriteFunction = (exchange, body) -> {
        if (exchange.getResponse().getHeaders().getContentType().isCompatibleWith(MediaType.APPLICATION_JSON)) {
            DocumentContext jsonBody = JsonPath.parse(body);

            for (KeyValue kv : config.getKeyValues()) {
                try {
                    jsonBody.set( path: "$." + kv.getKey(), kv.getValue());
                } catch (PathNotFoundException e) {
                    LOGGER.debug("Could not find JSON path: " + kv.getKey(), e.getMessage());
                }
            }
            return Mono.just(jsonBody.jsonString());
        }
        return Mono.just(body);
    };
    modifyResponseBodyConfig.setRewriteFunction(rewriteFunction);

    return modifyResponseBodyGatewayFilterFactory.apply(modifyResponseBodyConfig);
}
```

Gateway hook

Custom Config

Apply Changes

## With great power comes great responsibility

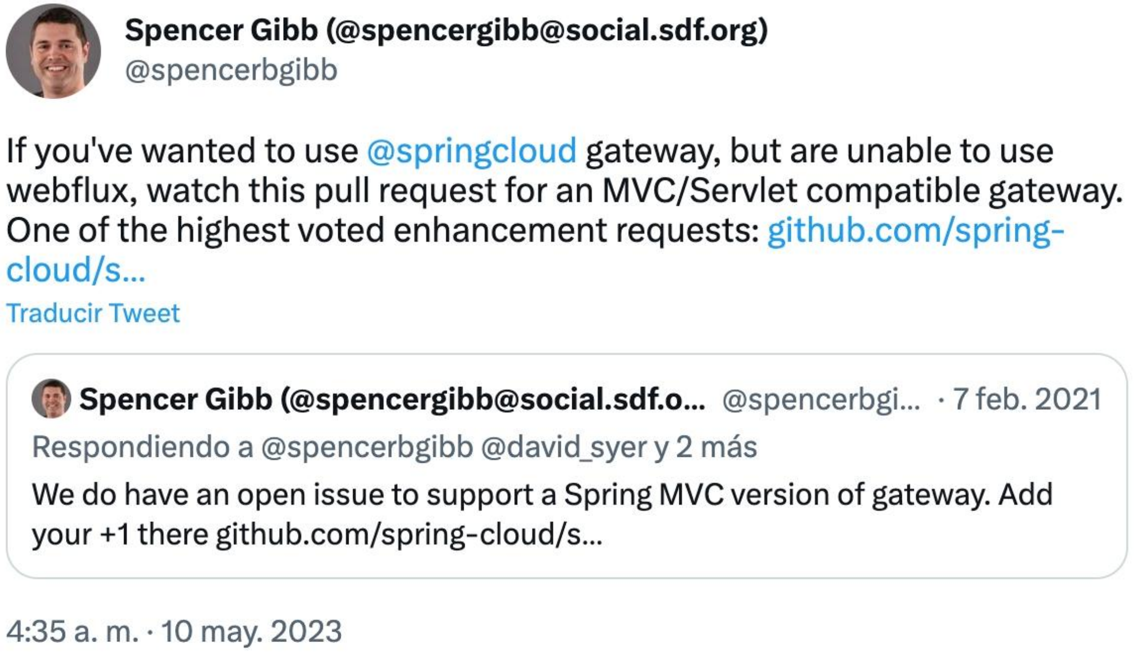
## Conclusions

- Know the difference between Request and Response
- Mind the Order
- Do not reinvent the wheel
- Gateway abstraction helps with feature encapsulation

## Demos

- https://github.com/abelsromero/gateway-with-style

## Everything we say, maybe be a bit useless 😊



**Spencer Gibb (@spencergibb@social.sdf.org)**
@spencerbgibb

If you've wanted to use @springcloud gateway, but are unable to use webflux, watch this pull request for an MVC/Servlet compatible gateway. One of the highest voted enhancement requests: github.com/spring-cloud/s...

Traducir Tweet

**Spencer Gibb (@spencergibb@social.sdf.o...** @spencerbgi... · 7 feb. 2021
Respondiendo a @spencerbgibb @david_syer y 2 más
We do have an open issue to support a Spring MVC version of gateway. Add your +1 there github.com/spring-cloud/s...

4:35 a. m. · 10 may. 2023

# THANK YOU! Q&A

Marta Medio Menéndez
@tpdeoro

Abel Salgado Romero
@abelsromero