# Alfresco Authentication with Spring Cloud Gateway

(SSO & OAuth2)


Daniel Gradečak

10 March 2021

# Agenda

*Alfresco Authentication with Spring Cloud Gateway*

*10/03/2020*

- What is Spring Cloud Gateway?
- Spring Security with OAuth2
- Spring Cloud Gateway and Alfresco Authentication
- Demo
- Resources and links
- Q&A

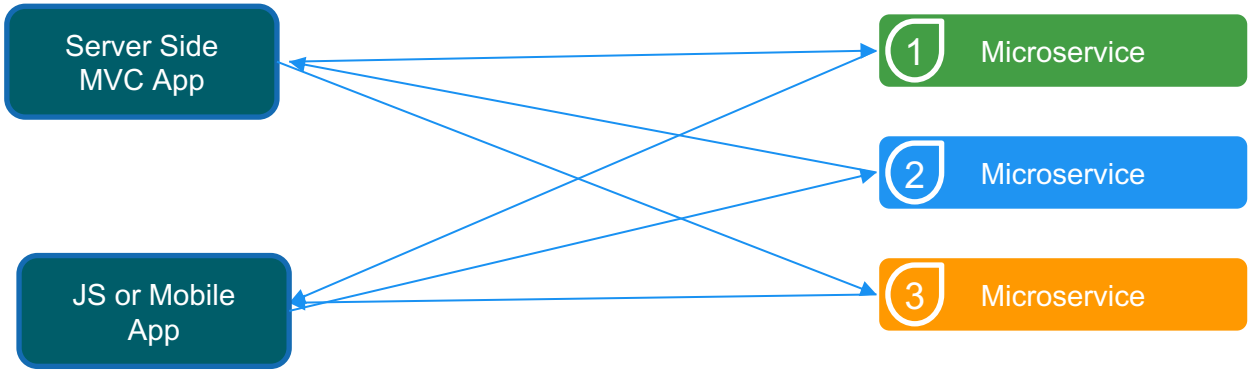
Not included synchronization of users and groups

# What is Spring Cloud Gateway?

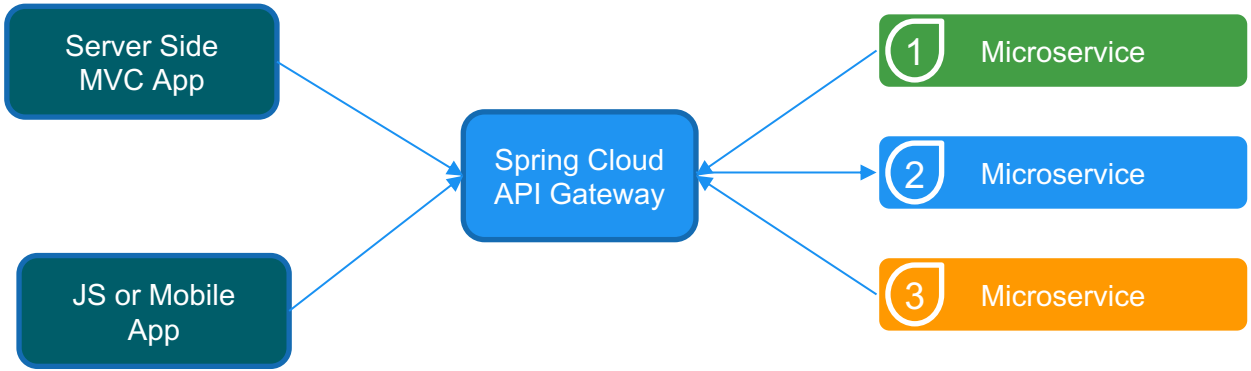
NGINX is a proxy

Spring Cloud Gateway is a proxy on steroids

# Spring Cloud Gateway – why using an API Gateway?

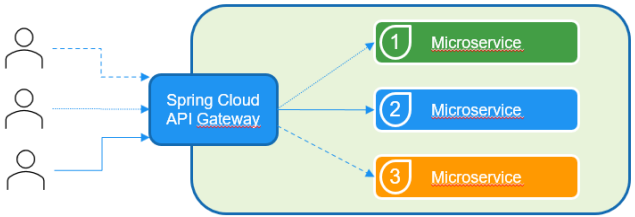


# Spring Cloud Gateway – why using an API Gateway?



# Spring Cloud Gateway – why using an API Gateway?

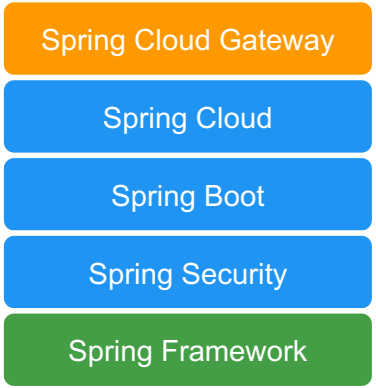
- An API gateway provides a single, unified entry point across one or more internal REST services.
- A microservice-based architecture may have from 10 to 100 or more services. An API gateway can help provide a unified entry point for external consumers, independent of the number and composition of internal microservices.


Common API gateway functions include routing, rate limiting, billing, monitoring, analytics and **authentication**, **security**.

# What is Spring Cloud Gateway?

Spring Cloud Gateway is just another Spring Boot application, you have access to the entire Spring ecosystem.




**Spring Cloud Gateway features:**

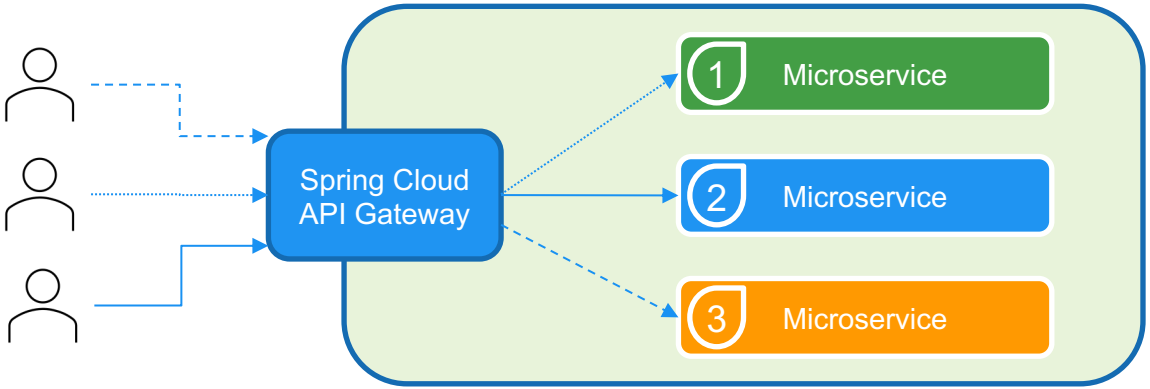
- ☑ Built on Spring Framework 5, Project Reactor and Spring Boot 2.0
- ☑ Able to match routes on any request attribute
- ☑ **Predicates** and **Filters** are specific to routes
- ☑ **Predicates** and **Filters** - Easy to write
- ☑ Circuit Breaker integration
- ☑ Request Rate Limiting
- ☑ Path Rewriting
- ☑ …

# What is Spring Cloud Gateway?

Spring Cloud Gateway provides a library for building API gateways based on Java and on the reactive/non-blocking support in Spring (WebFlux).

It provides a flexible way of routing requests based on a number of criteria, as well as focuses on cross-cutting concerns such as security, resiliency, and monitoring.

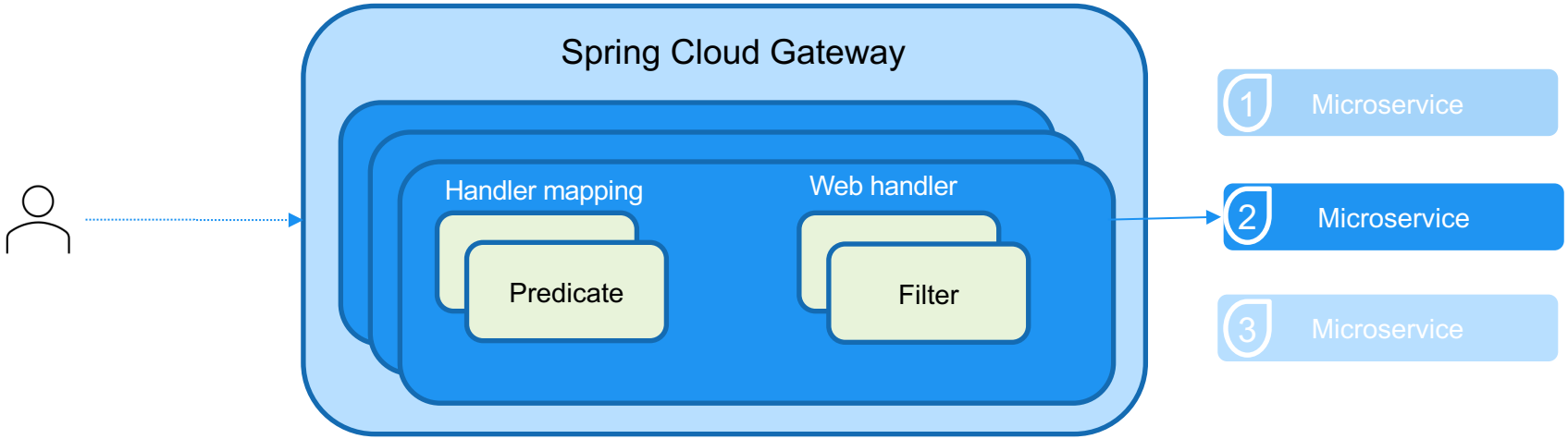
An **API gateway** is a way to decouple the client interface from your backend implementation. When a client makes a request, the **API gateway** breaks it into multiple requests, routes them to the right places, produces a response, and keeps track of everything.



# Spring Cloud Gateway - architecture

**Cons**:
- It is yet another highly available component that must be developed, deployed, and managed
- There is also a risk that the API Gateway becomes a bottleneck

**Pros**:
- can use Spring HttpSession and Redis for HA if required
- Integrated with Spring Security and OAUTH2 support
- Single point of entry for metrics



# Spring Cloud Gateway - Predicates

**Predicate**:
- tests if the given request fulfills a given condition
- for each route, we can define one or more predicates that, if satisfied, will accept requests for the configured backend after applying the filters.

```
spring.cloud.gateway:
       routes:
        - id: alfresco
          uri: http://localhost:8080/alfresco
          predicates:
            - Path=/alfresco/**
            - Method=GET,POST
            - Header=X-Request-Id, 123
                                    -
Host=**.somehost.org,**.anotherhost.org
            - name=Cookie
              args:
                name=mycookie
                regexp=mycookievalue
            …
```

we are mostly interested in this one (Path)
means: anything that comes to http://gateway/alfresco/

# Spring Cloud Gateway - Filters

**Filter**:
- make the modification of the incoming HTTP request or outgoing HTTP response
- Pre-Filters & Global-Filters
- Post-Filters

```
spring.cloud.gateway:
       routes:
        - id: alfresco
          uri: http://localhost:8080/alfresco
          filters:
            - TokenRelay
            - AddRequestHeader=X-Request-red, blue
            - AddResponseHeader=X-Response-Red, Blue
            - name=CircuitBreaker
              args:
                 name=myCircuitBreaker
                 fallbackUri=forward:/inCaseOfFailureUseThis
          …
```

Unfortunately there is no JWT creation provided out of the box
***we will create a new filter later …***

# Spring Security with OAuth2

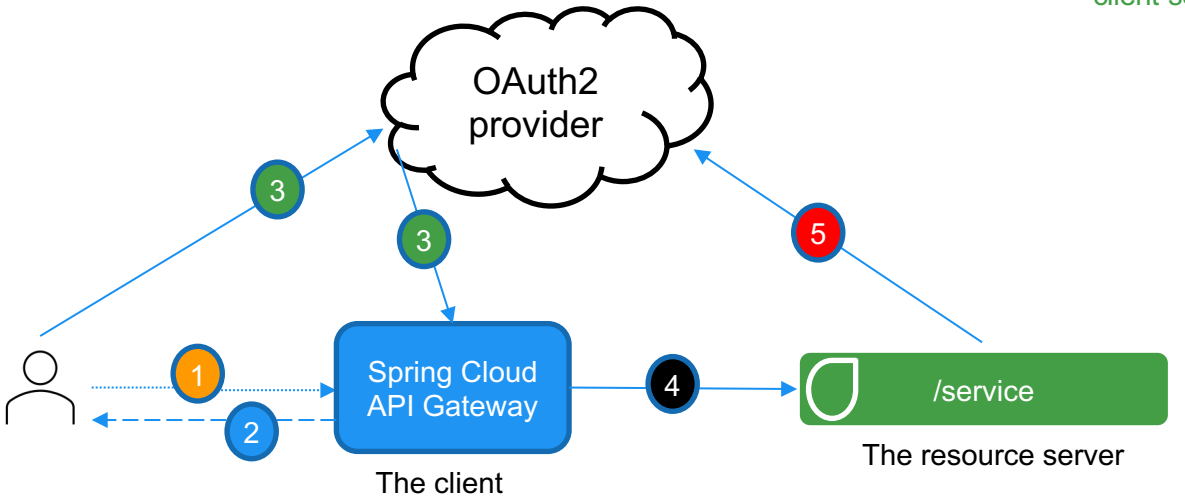
In OAuth2 terms, in short we have

1) a Client
2) an Authorization Server (Google, Facebook, …)
3) and a Resource Server

# Spring Cloud Gateway – Spring Security with OAUTH2

One of the key features in Spring Security 5 is support for writing applications that integrate with services that are secured with OAuth2. This includes the ability to sign into an application by way of an external service such as Google, Facebook, GitHub, Keycloak …

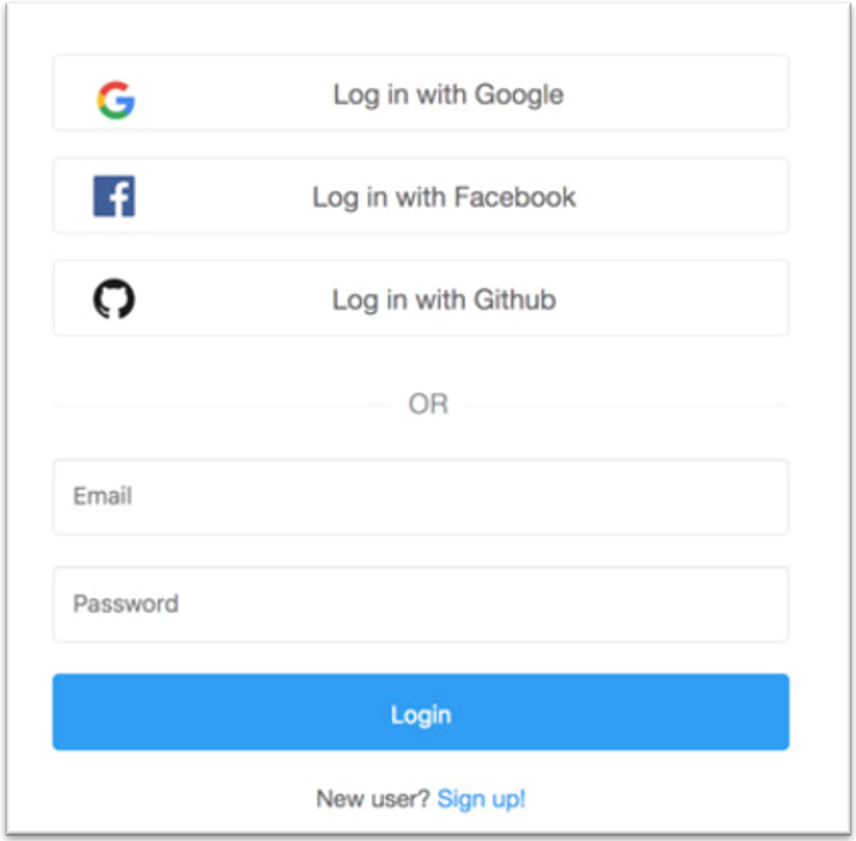
```
spring:
  security:
    oauth2:
      client:
        registration:
          google:
            client-id: CLIENT ID
            client-secret: CLIENT SECRET
```




1. User requests **/service**
2. Spring security redirects to login
3. OAUTH protocol for authentication + autorization
4. **Requests and TOKEN relayed to the service**
5. **Service validates the TOKEN with the provider**

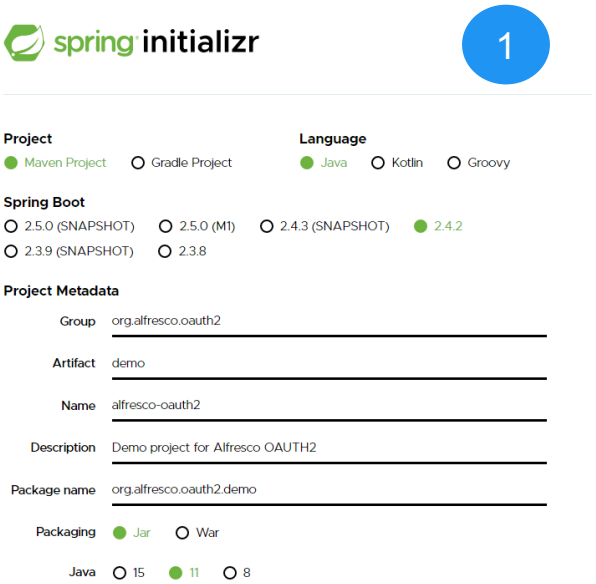
# Spring Security – login page

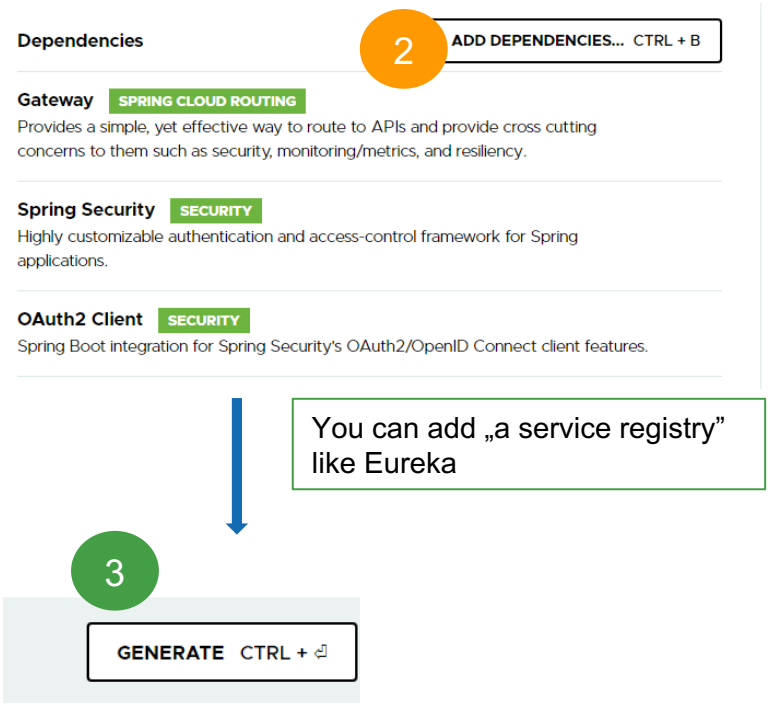
We can either use the default login page (below) or create a custom login page (on the right) with frameworks like Thymeleaf or any other supported by Spring

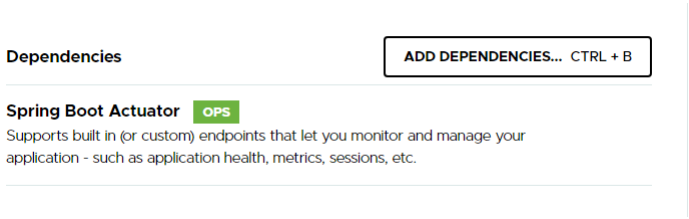
# Spring Boot – create a new Spring Gateway app

Navigate to: https://start.spring.io/



# Spring Boot – actuators and metrics

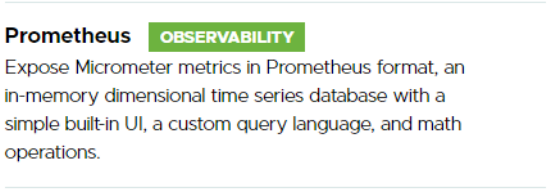



management.endpoints.enabled-by-default=true
management.endpoint.gateway.enabled=true
management.endpoints.web.exposure.include=*

https://gateway/actuator

**Prometheus metrics integration:**
Add dependency io.micrometer:micrometer-registry-prometheus

http://gateway/actuator/prometheus




Spring Boot includes a number of additional features to help you monitor and manage your application when you push it to production. You can choose to manage and monitor your application by using HTTP endpoints or with JMX. Auditing, health, and metrics gathering can also be automatically applied to your application.

# Demo

# Spring Cloud Gateway and Alfresco Authentication

Alfresco as a Resource Server

# Spring Cloud Gateway – Alfresco Authentication

Alfresco Authentication provides the new „Identity Service” auth subsystem

and can be configured **without** the „Alfresco Identity Service – Keycloak server”
in *alfresco-global.properties*

Alfresco 6.1 →

identity-service.authentication.enabled = true
identity-service.authentication.enable-username-password-authentication = false
identity-service.bearer-only = true
identity-service.realm-public-key = YOUR_PUBLIC_KEY

Alfresco 6.2 →

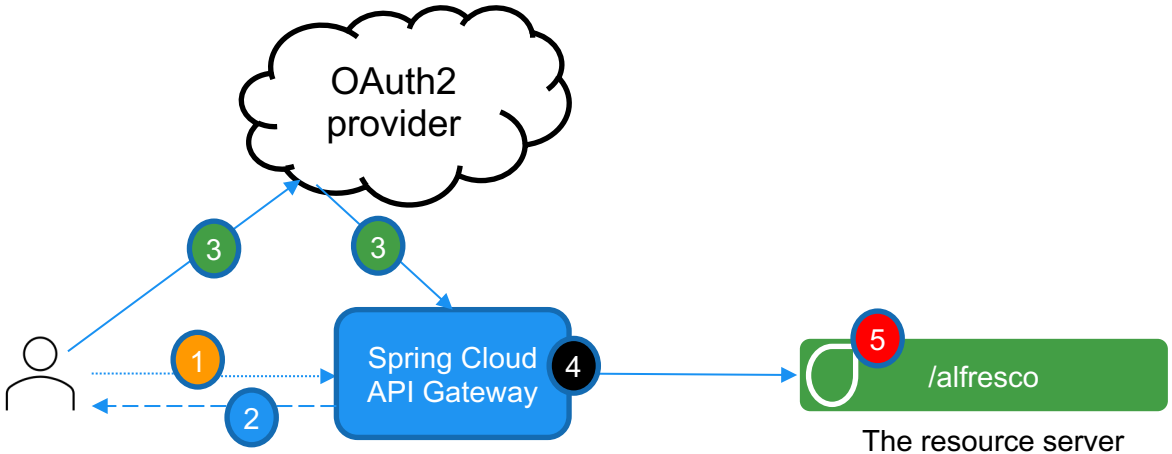
identity-service.register-node-at-startup = true

During this session we will not handle provisioning or LDAP synchronization for users and groups

# Spring Cloud Gateway – Alfresco Authentication

identity-service.realm-public-key = YOUR_PUBLIC_KEY 5 ⟵ Is the *IMPORTANT* configuration

4 ⟵ Uses the private key to sign our JWT




1 User requests **/service**

2 Spring security redirects to login

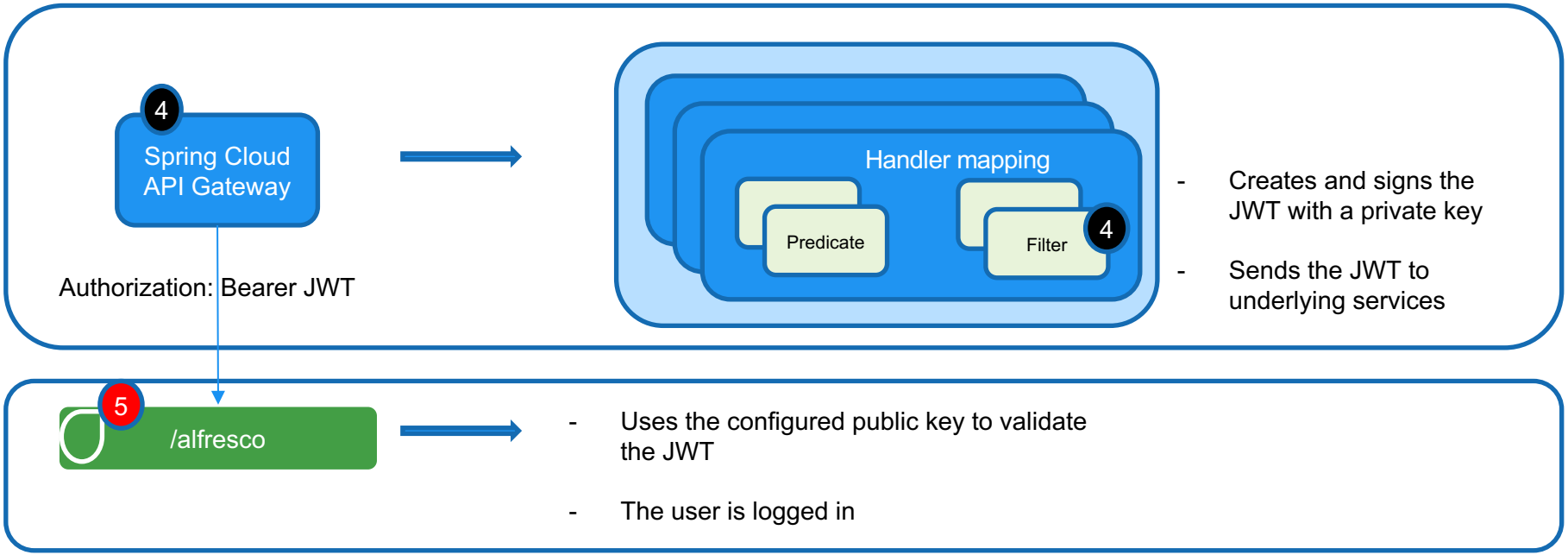
3 OAUTH protocol for authentication + autorization

4 Create the **JWT** relayed to the service

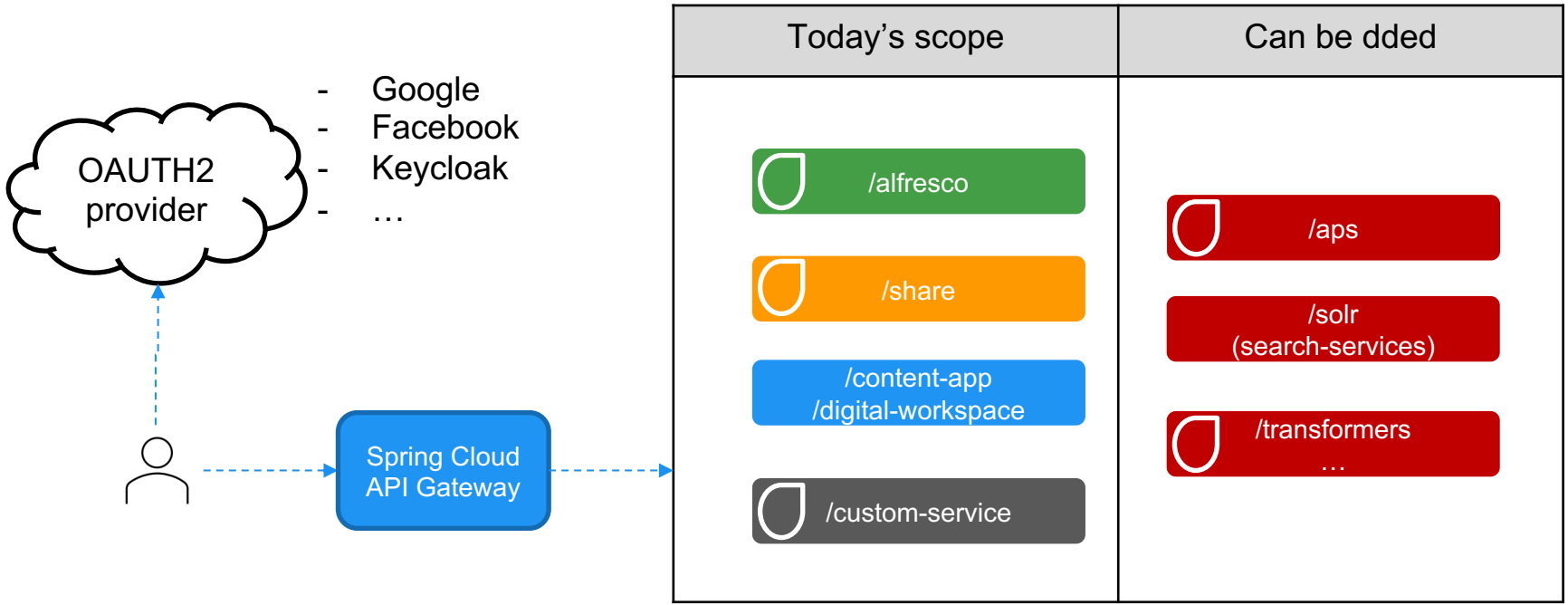
5 **Alfresco validates the JWT**

# Spring Cloud Gateway – Alfresco Authentication

What happened in 4 and 5 ?



# Spring Cloud Gateway – Alfresco Authentication




Can be used with AOS (Edit in MS Office)

# Spring Cloud Gateway – define a custom JWT filter

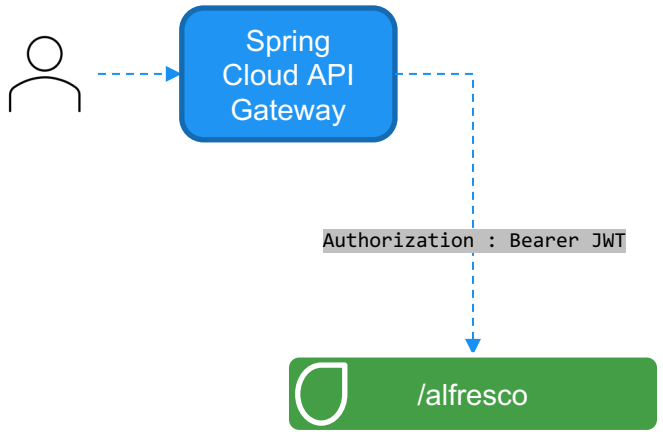
```
public class JwtBearerAuthorizationHeaderGatewayFilterFactory
extends AbstractGatewayFilterFactory<AbstractGatewayFilterFactory.NameConfig> {

   @Override
   public GatewayFilter apply(NameConfig config) {
      … create the JWT and add it to the Authorization header
   }
}
```

Alfresco *identity-service authentication subsystem* expects some predefined claims in the JWT

```
new JWTClaimsSet.Builder().subject(username)
.claim("authorities",Collections.emptyList())
.audience("pleosoft").claim("username", username)
.claim("typ", "Bearer").issuer("http://localhost:8180/auth/realms/alfresco")
.claim("preferred_username", username).build();
```

# Spring Cloud Gateway – send the JWT to Alfresco

```
spring.cloud.gateway:
    routes:
        - id: alfresco
          uri: http://localhost:8080/alfresco/
          predicates:
            - Path=/alfresco/**
          filters:
            - JwtBearerAuthorizationHeader
```

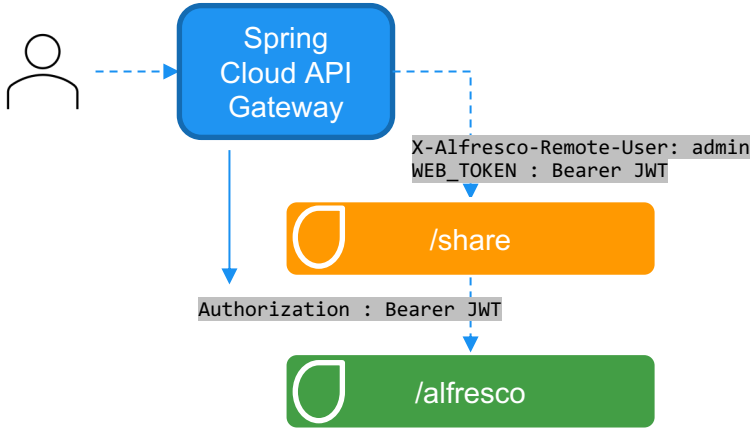



```
public class JwtBearerAuthorizationHeaderGatewayFilterFactory{
    …
    adding to the request => Authorization: Bearer + generated JWT
}
```

# Spring Cloud Gateway – send the JWT to Share

- Share has no JWT support ☹
- But we can reuse the external authentication config for Share
- And create another Gateway filter : `UsernameHeader`


```
spring.cloud.gateway:
    routes:
        - id: share
          uri: http://localhost:9090/share/
          predicates:
            - Path=/share/**
          filters:
            - UsernameHeader=X-Alfresco-Remote-User
            - JwtBearerAuthorizationHeader=WEB_TOKEN
```



# Spring Cloud Gateway – send the JWT to Share


Alfresco Share

```
<connector>
   <id>alfrescoHeader</id>
   <name>Alfresco Connector</name>
   <class>com.gradecak.alfresco.jwt.authorization.JwtAuthorizationAlfrescoConnector</class>
   <userHeader>X-Alfresco-Remote-User</userHeader>
   <jwtHeader>WEB_TOKEN</jwtHeader>
</connector>
```

```
public class JwtAuthorizationAlfrescoConnector extends SlingshotAlfrescoConnector {
   …
   remoteClient.setRequestProperties(Collections.singletonMap("Authorization", JWT (from the header WEB_TOKEN)));
   …
}
```

# Spring Cloud Gateway – send the JWT to ADF applications

To enable SSO and bypass the normal login, set the withCredentials property to true in the auth section of the ***app.config.json***

```
"auth": { "withCredentials": true }
```

Alfresco Development Framework

```
spring.cloud.gateway:
       routes:
         - id: aca
          uri: http://localhost:4200
          predicates:
            - Path=/aca/**
          filters:
            - StripPrefix=1
            - JwtBearerAuthorizationHeader
```

http://gateway/aca/

/aca/ is being stripped otherwise it would go to http://localhost:4200/aca/

# Spring Cloud Gateway – send the JWT to custom services

```
spring.cloud.gateway:
       routes:
        - id: customservice
          uri: http://localhost:7070/
          predicates:
            - Path=/customservice/*
          filters:
            - StripPrefix=1
            - JwtBearerAuthorizationHeader
```

Custom service

PdfRenderer transformer (T-engine)

```
spring.cloud.gateway:
       routes:
        - id: pdfrenderer
          uri: http://localhost:8070/
          predicates:
            - Path=/pdfrenderer/*
          filters:
            - StripPrefix=1
            - JwtBearerAuthorizationHeader
```

# Demo

# Summary

We talked about …

**Spring Cloud Gateway**

**Spring Boot & Spring Security**


**Alfresco Repository**
Identity Services – without Keycloak

**Alfresco Share & ADF applications**
- Content App
- Digital workspaces
- AOS (Edit in MS Office)

**Secure other Alfresco service like:**
- Search Services
- T-engines
- …

**OAuth2 & SSO integrations**

# Resources and Links

## Links

https://spring.io/projects/spring-cloud-gateway

https://github.com/dgradecak/alfresco-jwt-auth

## Blog

https://gradecak.com

## Contact

https://pleosoft.com

daniel@pleosoft.com

# Thank you

Questions???