



# Unified authorization in microservices

(http and kafka-based interactions)

September 2021



# STANISLAV DEVIATOV

## ***Solution Architect***

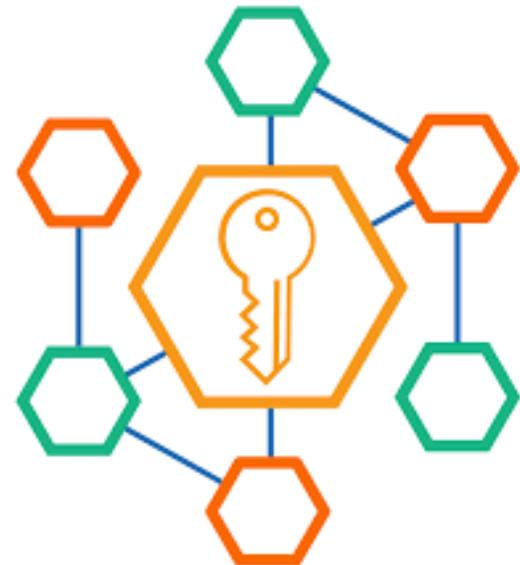
- 14 years experience in integration projects:
  - Mulesoft Anypoint Platform
  - Oracle Integration Stack (OSB, SOA/BPM Suites, ODI, OWB)
  - Open Source based solutions (Apache Camel, Apache Kafka)
- Wide experience: ERP and CRM implementation, ECM, IRM, MOLAP, Oracle database administration & development, java custom development, and microservice based projects.
- EPAM API & Integration CC expert
- Public Integration Community driver



# Agenda

---

- Authentication and authorization
  - Basics
  - Sync & async interactions details
  - Typical cases
- HTTP security overview
- Kafka security overview
- Unified solution based on OAuth2.0 & OIDC



# Assumptions

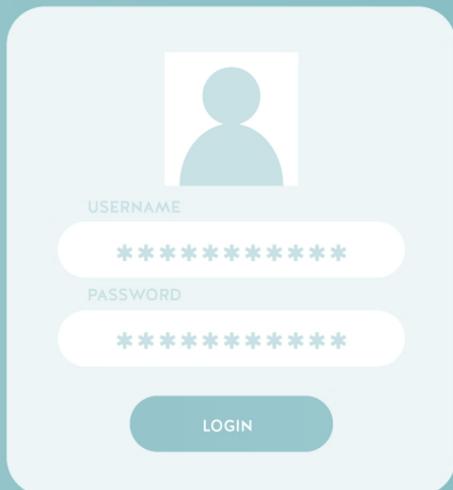
---

I assume that the audience is familiar with:

- HTTP
- Apache Kafka
- JWT
- OAuth2
  - OpenID Connect (OIDC)
  - User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization



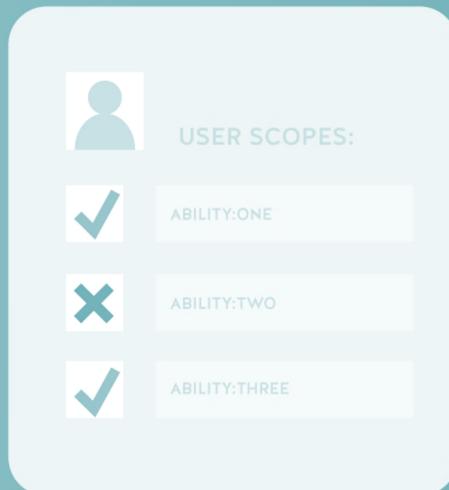
## AUTHENTICATION



WHO ARE YOU?

## AUTHORIZATION

VS

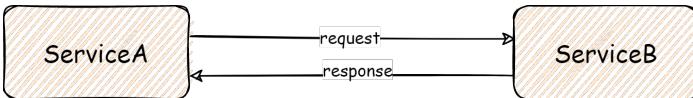


CAN YOU DO THAT?

# Differences in authorization in sync and async communications

## SYNC

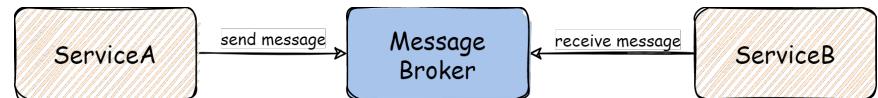
- Direct communication:



- Simplified model
  - The callee gets authentication info from the caller

## ASYNC

- Broker based communication:

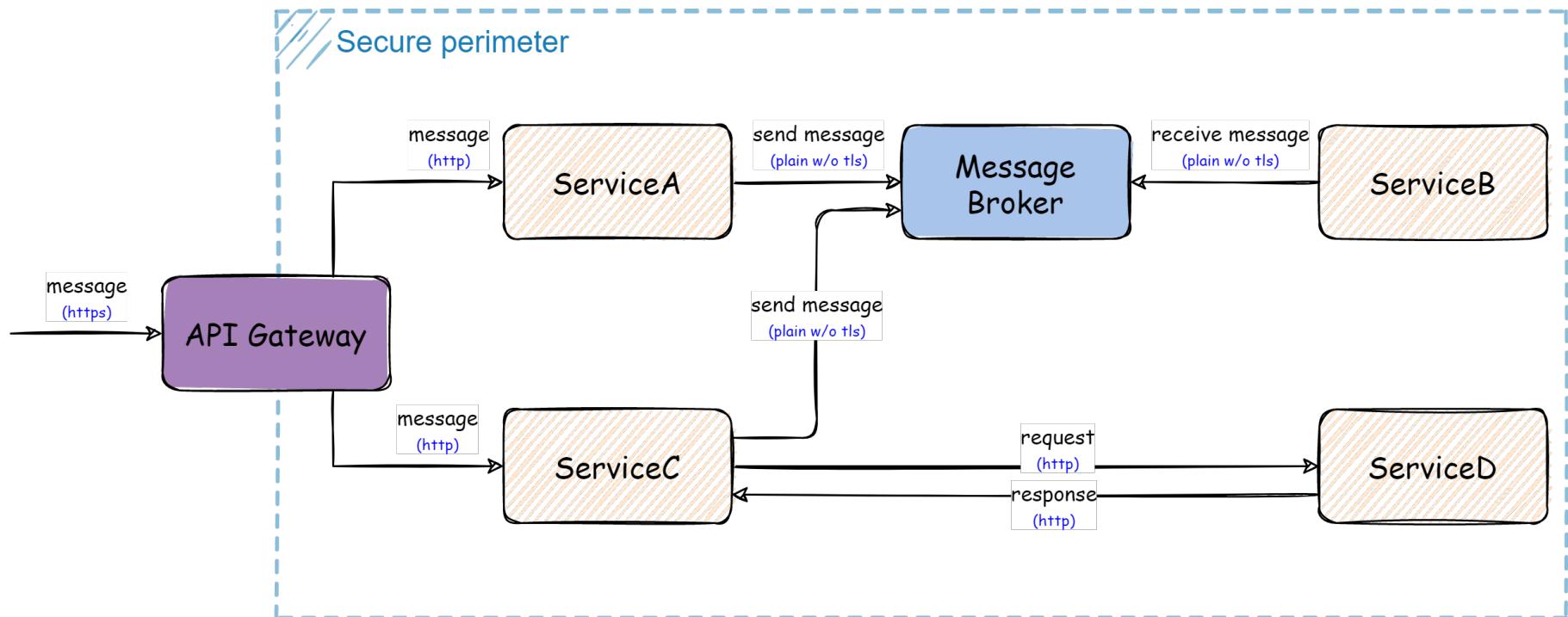


- Simplified model
  - Producer has a write (or send) permission to the destination (queue/topic)
  - Consumer has a read (or receive) permission to the destination (queue/topic)
  - Consumer doesn't get authentication info from the producer

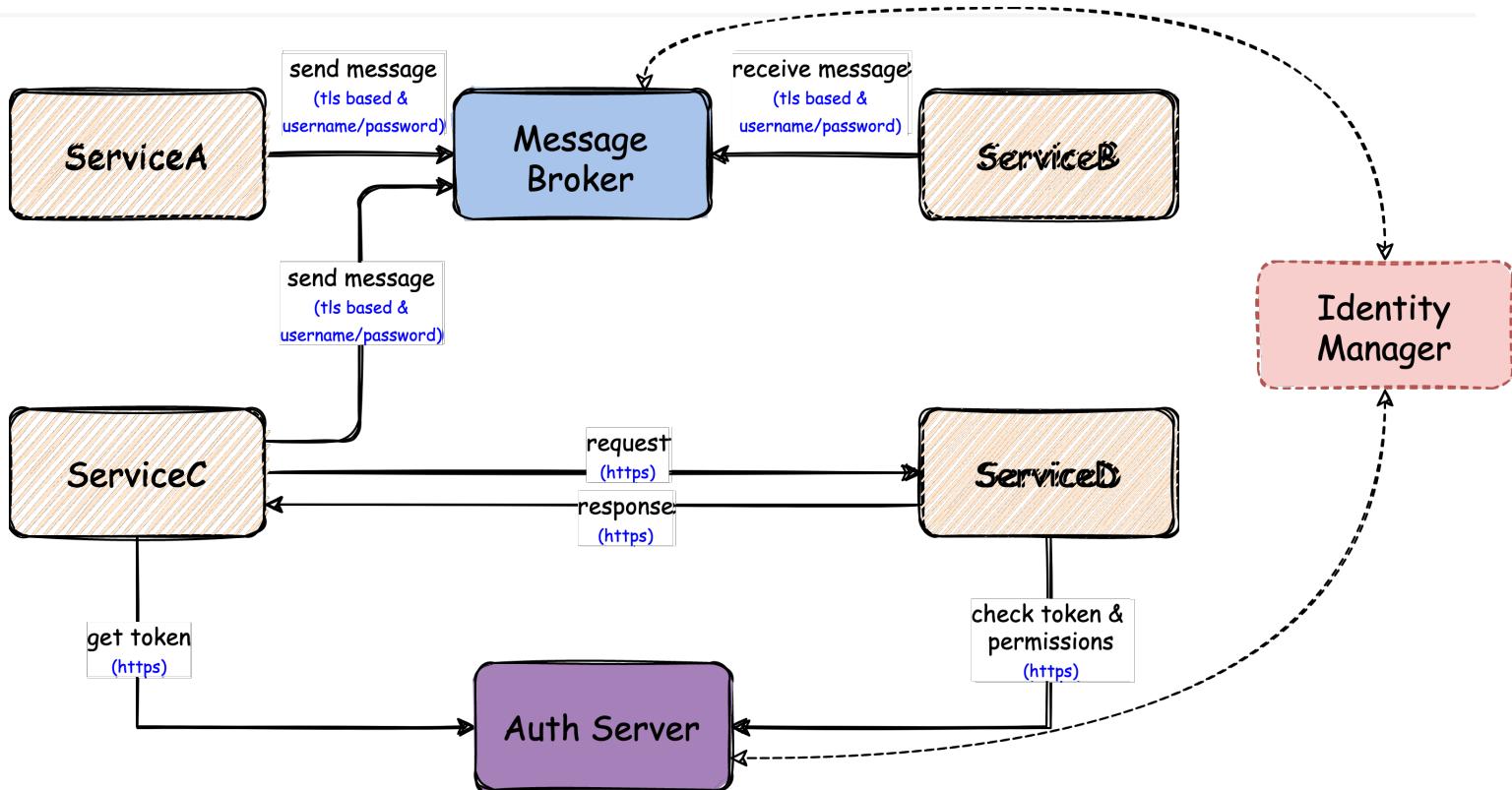
## TYPICAL AAA\* IN MICROSERVICE BASED PROJECTS

\* *authentication, authorization, and accounting*

## Option 1: perimeter security only



## Option 2: transport and application-level security



## Option 3: your case

---



## **HTTP AND KAFKA SECURITY**

# Security Tokens

---

## JWT

- JSON Web Tokens (pronounced /dʒɒt/, same as the word "jot")
- RFC-7523 published in 2014
- Based on JSON
- JWT, OAuth2, and OpenID Connect are common in the Social Media, but is now finding its way into the enterprise
- It's used in:
  - OpenID Connect (OIDC)
  - OAuth 2.0

## SAML

- Security Assertion Markup Language Tokens
- SAML 2 introduced in 2005
- Based on XML
- SAML 2 is the standard for Enterprise SSO
- It's used commonly in:
  - WS-Security
  - WS-Trust
  - WS-Federation
  - Etc.
- It uses:
  - XML Digital Signature
  - XML Encryption
  - XML Schema (XSD)

# HTTP Security

---

## TRANSPORT LEVEL

- TLS based communications

## APPLICATION LEVEL

- HTTP Strict Transport Security (HSTS)
- Security Headers
- Cross-origin resource sharing (CORS)
- Security Tokens
- API Keys
- Input validation
- etc.

# Kafka Security – Authentication & authorization

---

## AUTHENTICATION

- Authentication of connections to brokers from clients (producers and consumers), other brokers and tools, using either SSL or SASL\*. Kafka supports the following SASL mechanisms:
  - SASL/GSSAPI (Kerberos) - starting at version 0.9.0.0
  - SASL/PLAIN - starting at version 0.10.0.0
  - SASL/SCRAM-SHA-256 and SASL/SCRAM-SHA-512 - starting at version 0.10.2.0
  - SASL/OAUTHBEARER - starting at version 2.0

## AUTHORIZATION

- Kafka ships with a **pluggable** Authorizer and an out-of-box authorizer implementation that uses zookeeper to store all the ACLs.

\* Simple Authentication and Security Layer (SASL) is defined by [RFC 4422](#)

# Kafka Security Model

---

## TOPIC

- Write
- Read
- Describe
- Create
- Delete
- DescribeConfigs
- AlterConfigs
- IdempotentWrite

## CLUSTER

- Create
- Describe
- Alter
- DescribeConfigs
- AlterConfigs
- IdempotentWrite
- ClusterAction

## TRANSACTIONAL ID

- Describe
- Write

## GROUP

- Read
- Describe
- Delete

## DELEGATION TOKEN

- Describe

# Kafka Security - SASL/OAUTHBEARER

---

## KIP-255

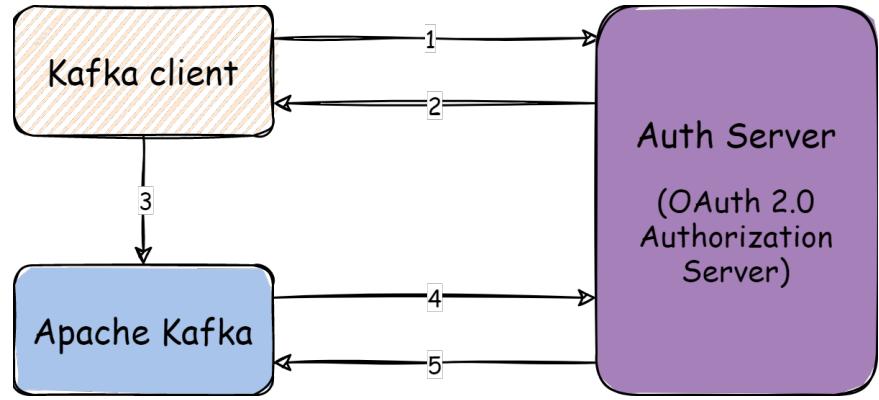
- [KIP-255 OAuth Authentication via SASL/OAUTHBEARER](#)
- Adopted in 2.0

## KIP-368

- [KIP-368: Allow SASL Connections to Periodically Re-Authenticate](#)
- Adopted in 2.2
- Kafka connections are long-lived, so there is no ability to change the bearer token associated with a particular connection. Allowing SASL connections to periodically re-authenticate would resolve this.
- Additional benefits:
  - eliminate access to Kafka for connected clients, the current requirement is to remove all authorizations
  - the use of short-lived tokens is a common OAuth security recommendation

# Kafka Authorizer for OAuth2

- Strimzi Kafka OAuth
  - Details:
    - <https://github.com/stimzi/stimzi-kafka-oauth>
  - Provides server-side and client-side parts:
    - custom server callback that will provide server-side token validation
    - custom client callback that will automatically perform re-authentication if access token expires



# SASL/OAUTHBEARER – Client library support

---

## JAVA

- Apache Kafka client
- Strimzi client lib (callback handler)

## NON-JAVA

- librdkafka
  - Is the Apache Kafka C/C++ client library
  - *Supports KIP-255 only*
  - Has language bindings
    - C#/.NET: confluent-kafka-dotnet (based on rdkafka-dotnet)
    - Go: confluent-kafka-go
    - Lua: luardkafka
    - Node.js: node-rdkafka
    - Python: confluent-kafka-python
    - Python: PyKafka
    - Rust: rust-rdkafka
    - Shell: kafkacat - Apache Kafka command line tool
    - Etc. <https://github.com/edenhill/librdkafka#language-bindings>

# SASL/OAUTHBEARER - KIP-368 in librdkafka

KIP-368 issue: <https://github.com/edenhill/librdkafka/issues/3304>

The screenshot shows a GitHub issue page for KIP-368: Allow SASL Connections to Periodically Re-Authenticate. The issue was opened by vctoriawu on March 18, 2022, and has 1 comment. The user edenhill commented on April 6, 2022, with the following text:

Thanks for looking into this feature!

- Specify a callback function in the config that gets/refreshes token info

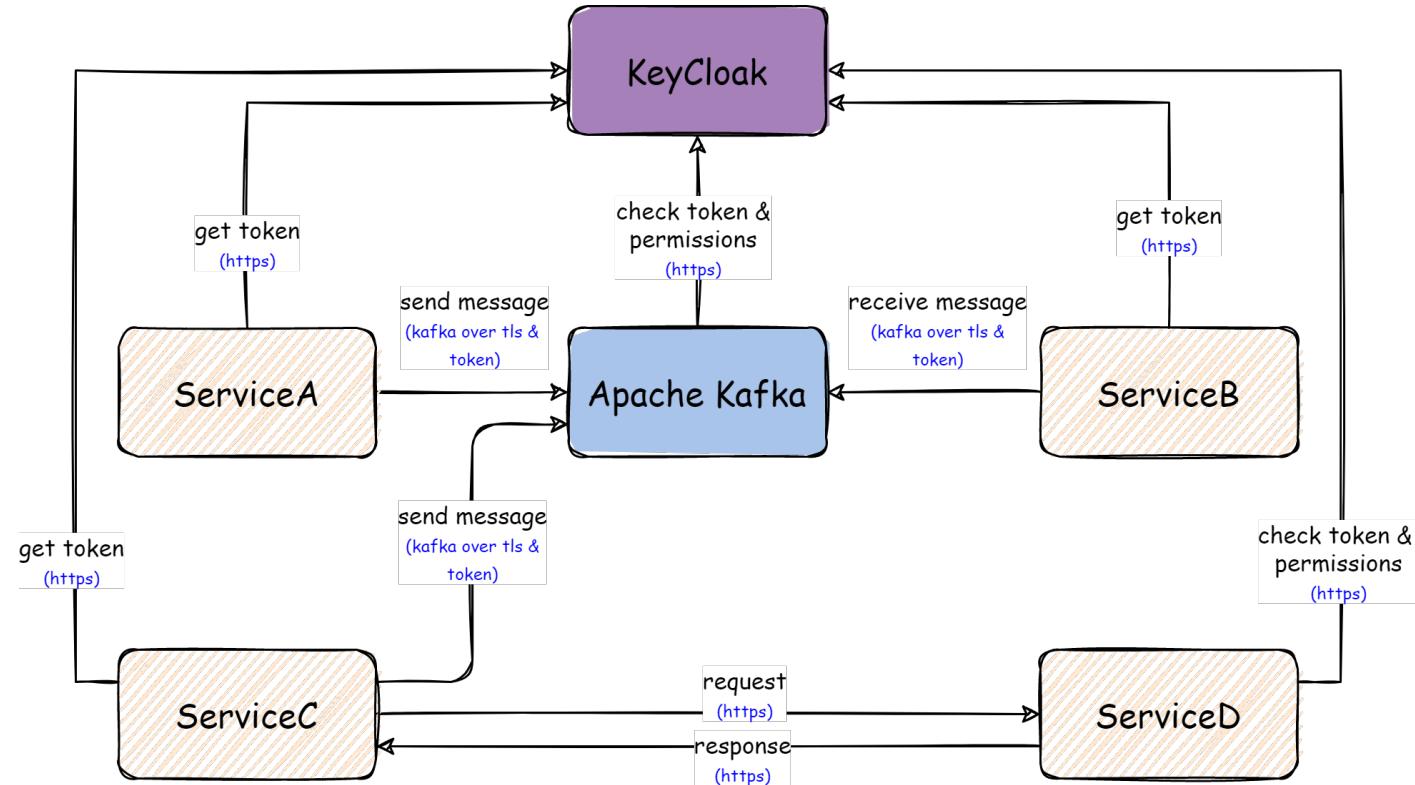
What token information is that? OAUTHBEARER? If so, there's already a refresh callback for that.

- Store the expiration time retrieved into rk\_trans

Authentication is not a transport-level thing in librdkafka (as opposed to the Java clients), so this state should be kept on the rkb (broker\_t).

I think it might be cleaner to add a reauth timer on the rkb and once that timer hits transition to a new BROKER\_STATE\_REAUTH\_DRAIN state that holds back any new sends and waits for all outstanding requests to finish. And when that is done transitions to the AUTH state to re-authenticate.

# Unified solution



## **DEEP DIVE INTO KAFKA SASL/OAUTHBEARER**

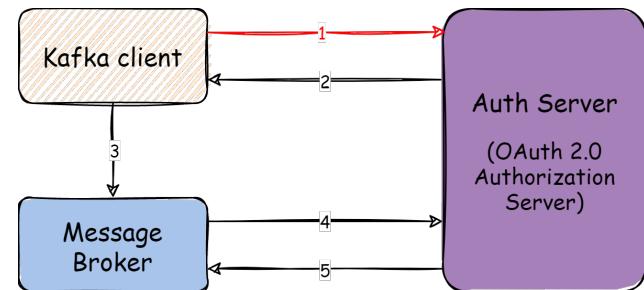
# How it works? #1

POST <http://keycloak:8080/auth/realms/kafka-authz/protocol/openid-connect/token>

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL

KEY	VALUE
<input checked="" type="checkbox"/> grant_type	client_credentials
<input checked="" type="checkbox"/> client_id	team-b-client
<input checked="" type="checkbox"/> client_secret	team-b-client-secret
Key	Value



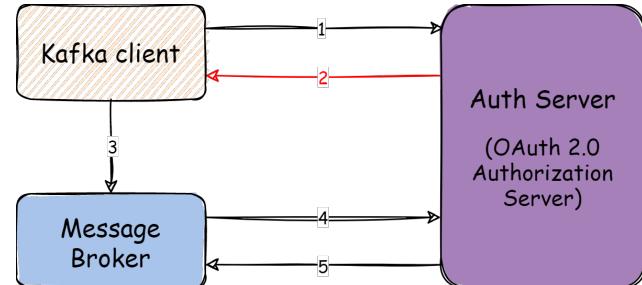
# How it works? #2

Body Cookies Headers (12) Test Results

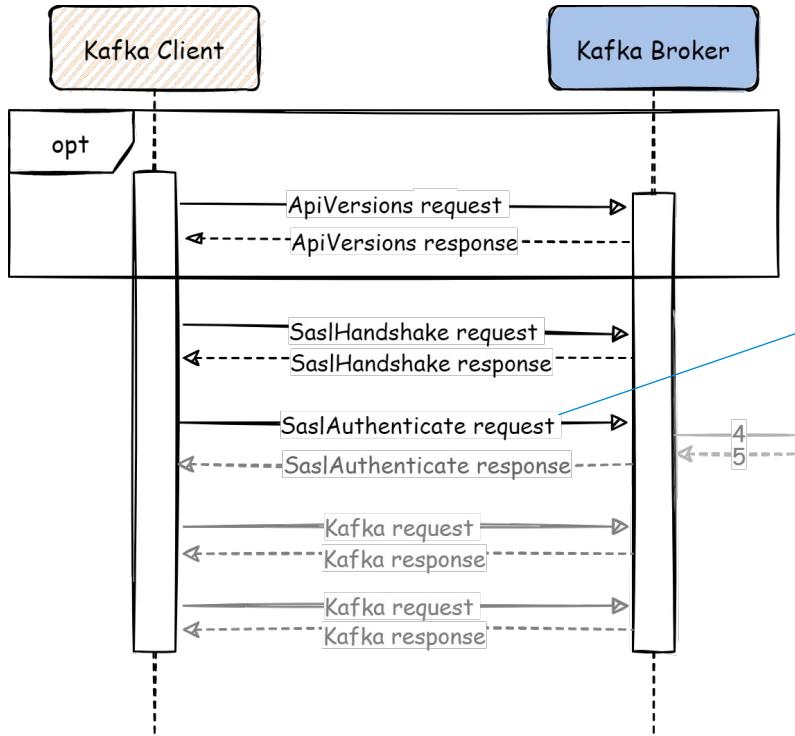
Status: 200 OK Time: 45 ms Size: 1.8 KB Save Response

Pretty Raw Preview Visualize JSON

```
{  
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldeUiwia2lKIa6ICJzbkUrRX1YszVHTVEycVRyb240WFRVY1pNUXdhcXVjZGRGd3JpM1hDSHhrIn0...  
  eyJleHAiOjE2MjU0MDExNzMsImlhdC16MTYyNTQwMTAyMjwianRpIjo1MDM2N2JjMWQtYjk0Zc00Dg2L1jMtgtNGY0OTkzMGEyM2ISIiwiaXNzIjoiaHR0cDovL2tleWNs  
  b2FrOjgwODAvYXV0aC9yZWFsbXMva2Fma2EtYXV0aHoILCJhdWQioiJhY2NvdW50IwiC3ViJoinJ1MTRhY2YtYzUwZi00NDM0LTl1ZTmTYwVizDEwODU2N2VlIwidHlw  
  IjoiQmVhcmVyiwiYXpwIjoidGVhb51iLNsakVudCisImFjciI6IjeilCjyZWfSb9hy2Nlc3MiOnsicm9sZXMiolsibZmbGluzVvh2Nlc3MiLCjEZXYgVGvhsBC1l19  
  LCjyZXNvdXjzV9hY2Nlc3MiOnsiYWjb3VudC16eyJyb2xlcI6WyJtW5hZ2UtYWNjb3VudCisIm1hbmfns1hY2Nvdw50LwxpbtzIiwidmldy1wcm0maWxlii19f5wi  
  c2NvcGUioi1lbWFpbBvcm9maWxliwiZm1habxfdmVyaWZpZQ10zhhmN1LCJjbG1bnR1b3N0IjoiMTcyLjE4LjAuMSIsImNsakVudCisInByZW1cn1lZf91c2VybmfTzS16In1nCnZpY2UtYWNjb3VudC10ZwFtLWtY2xpZw50IiwiY2xpZw50QRkcmVzyI6IjE3M14x0C4wLjEifq  
  f50pP9FwMJoVzhaKaJDvJq4f6s9KeVRShpIfJMuc2hehx0ohVMAME9pnUcbH1Yr7Hcr7zgSDjxHQ8iUh9pZ_XvPbiJzYl8sk5GqARGaOnGbHf7jcU_fA1HUY5LyAm8I1wxL  
  F7m1CHDTSa5jknfLsvxlg-9sfk__s7wXcovyyHqXFUMUmtdhbkagZkzfK1MZXTujtxBvkOf_yLftfulyptXkw3X-bkTu4cm_ax20xQMuKH_B41pgjjpy9eUfk075MOV  
  3RQ8GmbsNftTZV1n8K6zPiJ8sH5YrG1rTs2KFC_-crSJMSe4-6Bmme22_nTyP98JqsBodQc9RrKTg",  
  "expires_in": 120,  
  "refresh_expires_in": 0,  
  "token_type": "Bearer",  
  "not-before-policy": 0,  
  "scope": "email profile"}  
}
```



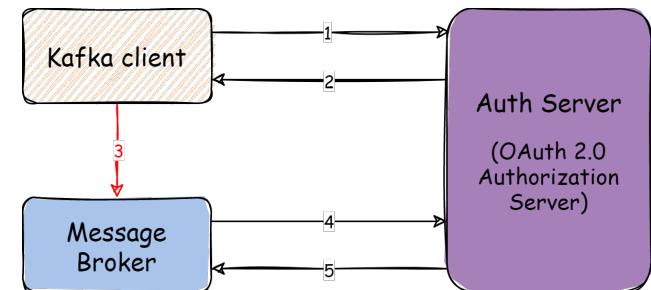
# How it works? #3



More details:

- [https://kafka.apache.org/protocol#sasl\\_handshake](https://kafka.apache.org/protocol#sasl_handshake)
- <https://www.novatec-gmbh.de/en/blog/kafka-security-behind-the-scenes/>
- <https://datatracker.ietf.org/doc/html/rfc4422>

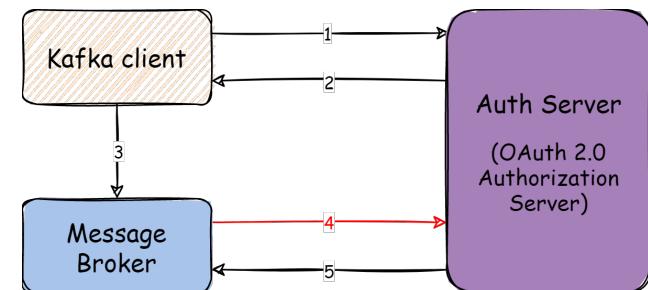
*including assess token from #2*



# How it works? #4

Screenshot of Postman Headers tab showing OAuth 2.0 parameters:

KEY	VALUE
Auth	Params Authorization Headers (11) Body Pre-request Script Tests Settings Cookies
Cache	none form-data x-www-form-urlencoded raw binary GraphQL
Content-Type	application/json
Content-Type	application/x-www-form-urlencoded
Host	kafka
audience	kafka
grant_type	urn:ietf:params:oauth:grant-type:uma-ticket
response_mode	permissions
Key	Value



# How it works? #5

Screenshot of a Postman API response showing OAuth 2.0 token information.

Body (Pretty JSON):

```
9
10    },
11    "rsid": "ca24cc58-47e7-4ac0-a843-6954f9063d7f",
12    "rsname": "kafka-cluster:cluster2,Group:*
```

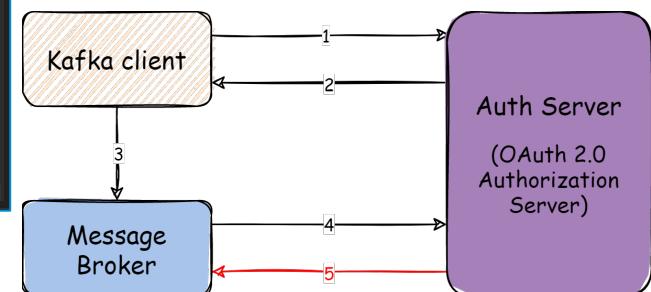
13 },  
14 {  
15 "scopes": [  
16 "ClusterAction",  
17 "DescribeConfigs",  
18 "AlterConfigs"  
19 ],  
20 "rsid": "190e4134-bd7b-4928-9b5e-9da847fff8a7",  
21 "rsname": "kafka-cluster:cluster2,Cluster:\*

22 },  
23 {  
24 "scopes": [  
25 "Alter",  
26 "Delete",  
27 "Describe",  
28 "Write",  
29 "Read",  
30 "IdempotentWrite",  
31 "Create",  
32 "DescribeConfigs",  
33 "AlterConfigs"  
34 ],  
35 "rsid": "f98d2535-bb30-418f-a51e-5292732b1e73",  
36 "rsname": "kafka-cluster:cluster2,Topic:\*

37 }

Response Headers:

- Status: 200 OK
- Time: 107 ms
- Size: 883 B
- Save Response



**HOW COULD IT BE CONSISTENT FOR HTTP?**

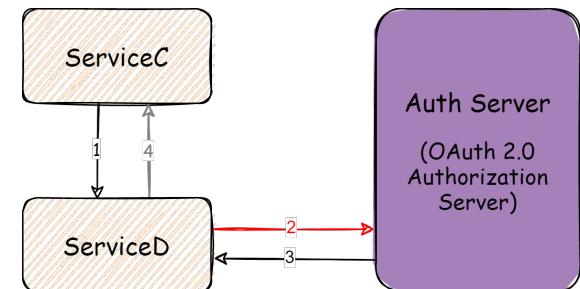
# HTTP interceptor to check permissions #1

The screenshot shows the Postman interface with two requests. The top request's Headers section contains:

KEY	VALUE
Authorization	Bearer eyJhbGciOiJSUzI1NiIsInR5cClg0AiSlldUiwia2lkIA6lCJLY1JWY1gxZHvndVc0bnBYT3VQRzg1emdaeHvremx6Z2VSTDg3SHdvVWEwIn0.e...
Cache-Control	
Content-Type	
Content-Range	
Host	

The bottom request's Body section (using x-www-form-urlencoded) contains:

KEY	VALUE
audience	service-d
grant_type	urn:ietf:params:oauth:grant-type:uma-ticket
response_mode	permissions



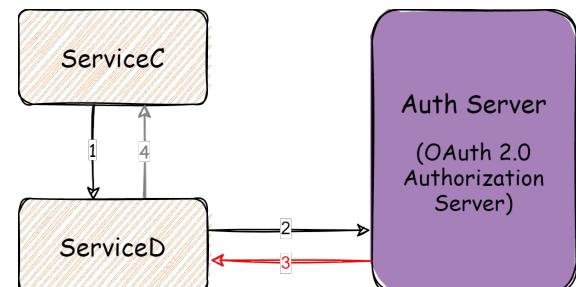
# HTTP interceptor to check permissions #2

Body Cookies Headers (11) Test Results

200 OK 18 ms 657 B

Pretty Raw Preview Visualize JSON

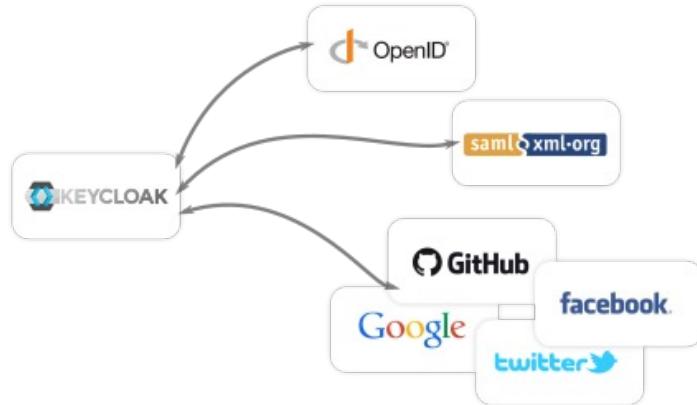
```
1 [  
2 {  
3     "scopes": [  
4         "POST",  
5         "GET"  
6     ],  
7     "rsid": "901de5d1-de6f-40f5-b46e-26b11d3faa65",  
8     "rsname": "/v1/invoices"  
9 },  
10 {  
11     "scopes": [  
12         "DELETE",  
13         "GET",  
14         "PUT"  
15     ],  
16     "rsid": "24e3beea-13fc-4aea-aa19-cc336e6a06bf",  
17     "rsname": "/v1/invoices/{id}"  
18 },  
19 {  
20     "scopes": [  
21         "GET"  
22     ],  
23     "rsid": "2c43f1d4-e6a2-46b9-9d25-6772f194c096",  
24     "rsname": "/v1/users"  
25 }]  
]
```



## AUTHORIZATION SERVER (KEYCLOAK)

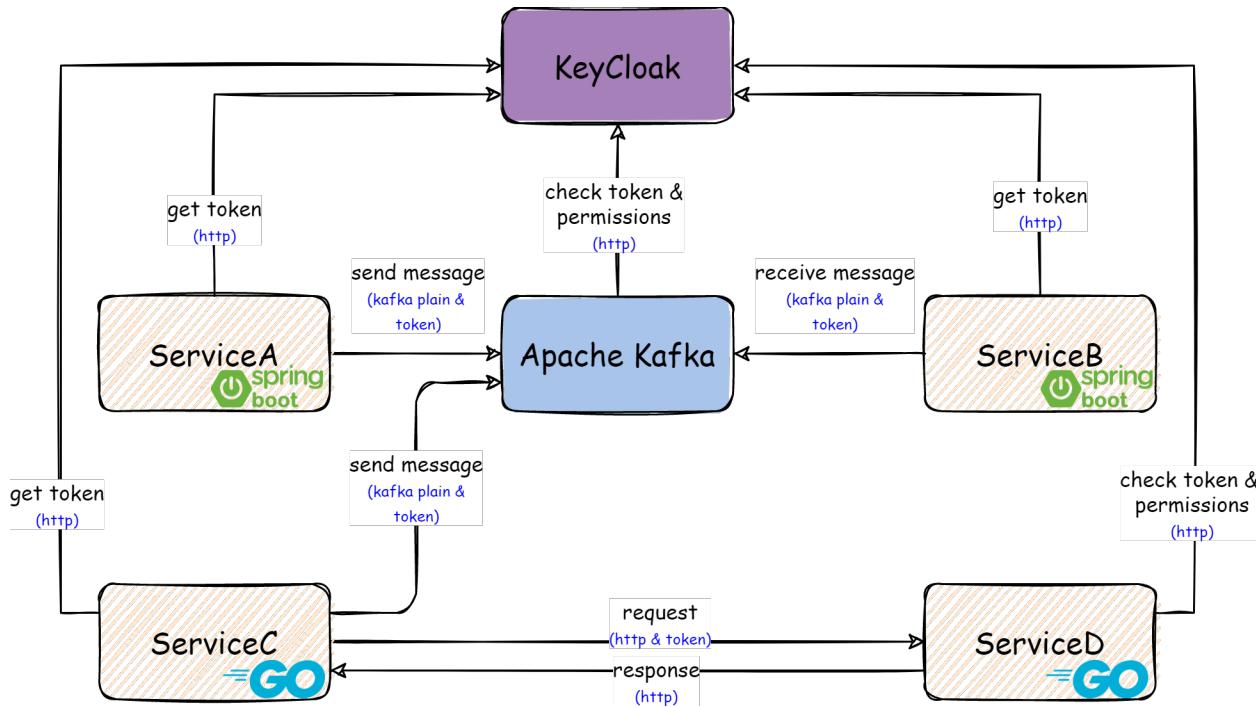
# Keycloak

- Supports protocols:
    - OpenID Connect 1.0
    - OAuth 2.0
    - SAML 2.0
  - Supports access control mechanisms:
    - Attribute-based access control (ABAC)
    - Role-based access control (RBAC)
    - User-based access control (UBAC)
    - Context-based access control (CBAC)
    - Rule-based access control using JavaScript
    - Time-based access control
  - Red Hat Single Sign-On (RH-SSO) is RedHat supported version of Keycloak
- Single Sign-On (SSO):
    - Single Log Out (SLO)
    - Kerberos bridge
  - User Federation:
    - Built-in support to connect to existing LDAP or Active Directory servers
  - Identity Brokering and Social Login



**DEMO**

# Demo case



Sources: <https://github.com/stn1slv/meetup-authorization>

# Possible disadvantages

---

- [KIP-368: Allow SASL Connections to Periodically Re-Authenticate](#) support is limited for non-JVM platforms
- Infrastructure components don't support SASL/OAUTHBEARER and should be improved. For example:
  - danielqsj/kafka-exporter
  - lightbend/kafka-lag-exporter
  - Confluent Schema Registry (w/o commercial components)
  - Confluent REST proxy (w/o commercial components)
- Depends on authorization service
  - Interaction with Auth service for each HTTP call
  - Interaction with Auth service for each Kafka client authentication (and re-authentication)

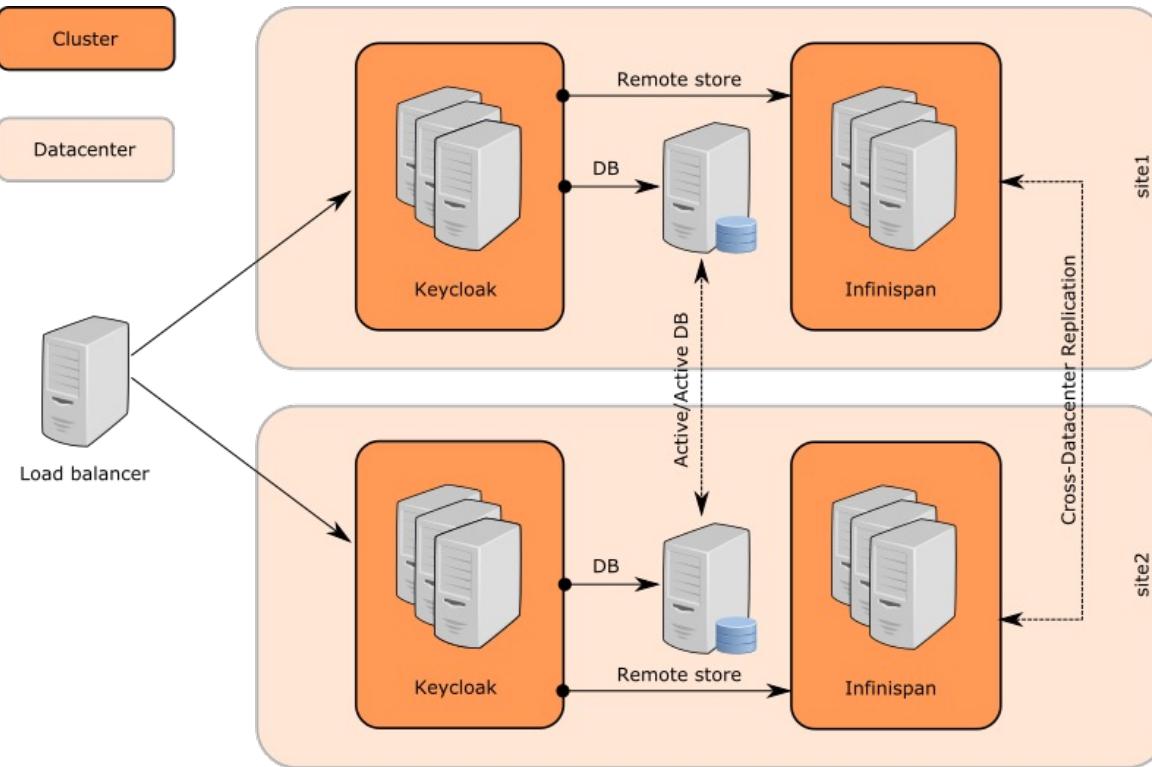


**THANK YOU**

## Q & A

## APPENDIX

# Keycloak clustering



# Keycloak: User-Based Policy

Clients > photoz-restful-api > Authorization > Policies > Add User Policy

## Add User Policy

Name \*

Description

Users \*  ▼

Username	Actions
alice	<button>Remove</button>
jdoe	<button>Remove</button>

Logic  ▼

Save Cancel

# Keycloak: Role-Based Policy

Clients > photoz-restful-api > Authorization > Policies > Add Role Policy

## Add Role Policy

Name \*

Description ?

Realm Roles \*

Clients ?

Client Roles \*

Name	Client	Required	Actions
manage-albums	photoz-restful-api	<input checked="" type="checkbox"/>	<button>Remove</button>

Logic ?

Save Cancel

# Keycloak: JavaScript-Based Policy

Clients > photoz-restful-api > Authorization > Policies > JavaScript > Only from @keycloak.org or Admin

## Only From @keycloak.org Or Admin

Name 	Only from @keycloak.org or Admin
Description 	Only users from @keycloak.org have access
Code 	<pre>1 var context = \$evaluation.getContext(); 2 var identity = context.getIdentity(); 3 var attributes = identity.getAttributes(); 4 var email = attributes.getValue('email').asString(); 5 6 if (identity.hasRole('admin')) email.endsWith('@keycloak.org'); 7   \$evaluation.grant(); 8 }</pre>
Logic 	Posi 

# Keycloak: Time-Based Policy

Clients > photoz-restful-api > Authorization > Policies > Add Time Policy

## Add Time Policy

Name <small>?</small>	Only in Period
Description <small>?</small>	A policy that limits access to a certain time period
Not Before <small>?</small>	2020-03-01 00:00:00
Not On or After <small>?</small>	2020-04-10 12:00:00
Day of Month <small>?</small>	1 <input type="button" value="^"/> to <input type="button" value="^"/> 10 <input type="button" value="v"/>
Month <small>?</small>	3 <input type="button" value="^"/> to <input type="button" value="^"/> 4 <input type="button" value="v"/>
Year <small>?</small>	2020 <input type="button" value="^"/> to <input type="button" value="^"/> 2020 <input type="button" value="v"/>
Hour <small>?</small>	0 <input type="button" value="^"/> to <input type="button" value="^"/> 12 <input type="button" value="v"/>
Minute <small>?</small>	0 <input type="button" value="^"/> to <input type="button" value="^"/> 30 <input type="button" value="v"/>
Logic <small>?</small>	Or <input type="button" value="▼"/>
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	

# Keycloak: Client-Based Policy

Clients > photoz-restful-api > Authorization > Policies > Add Client Policy

## Add Client Policy

Name **\***  My client policy

Description  Only these clients are allowed to access something

Clients **\***  Select an client... 

clientId	Actions
photoz-html5-client	

Logic  Po 

# Keycloak: Group-Based Policy

Clients > photoz-restful-api > Authorization > Policies > Add Group Policy

## Add Group Policy

Name \* ? Only IT Administrators

Description ? Only IT admins can access something

Groups Claim ? groups

Groups \* ? Groups

- People
- Sales
- Marketing
- IT
- Administrators**

Select

path	Extend to Children	Actions
/People/IT/Administrators	<input type="checkbox"/>	<b>Remove</b>

Logic ? Po ▾

**Save** **Cancel**

# Keycloak: Client Scope-Based Policy

Clients > photoz-restful-api > Authorization > Policies > Add Client Scope Policy

## Add Client Scope Policy

Name \*

Description

Client Scopes \*

Name	Required	Actions
album	<input checked="" type="checkbox"/>	<input type="button" value="Remove"/>

Logic

# Keycloak: Aggregated Policy

Clients > photoz-restful-api > Authorization > Policies > Aggregated > Restricted Administration Policy

## Restricted Administration Policy

Name *	Restricted Administration Policy	
Description	Only administrators from a specific network address can do something	
Apply Policy *	<div style="display: flex; align-items: center;"><div style="flex: 1;">Select existing policy...</div><div style="margin-left: 10px;">Create </div></div>	
Name	Description	Actions
Only from a specific client address	Only clients from a specific address can do something	Remove
Only Admin Policy	Only administrators can do something	Remove
Decision Strategy	Unanimous 	
Logic	Po 	
<button>Save</button> <button>Cancel</button>		