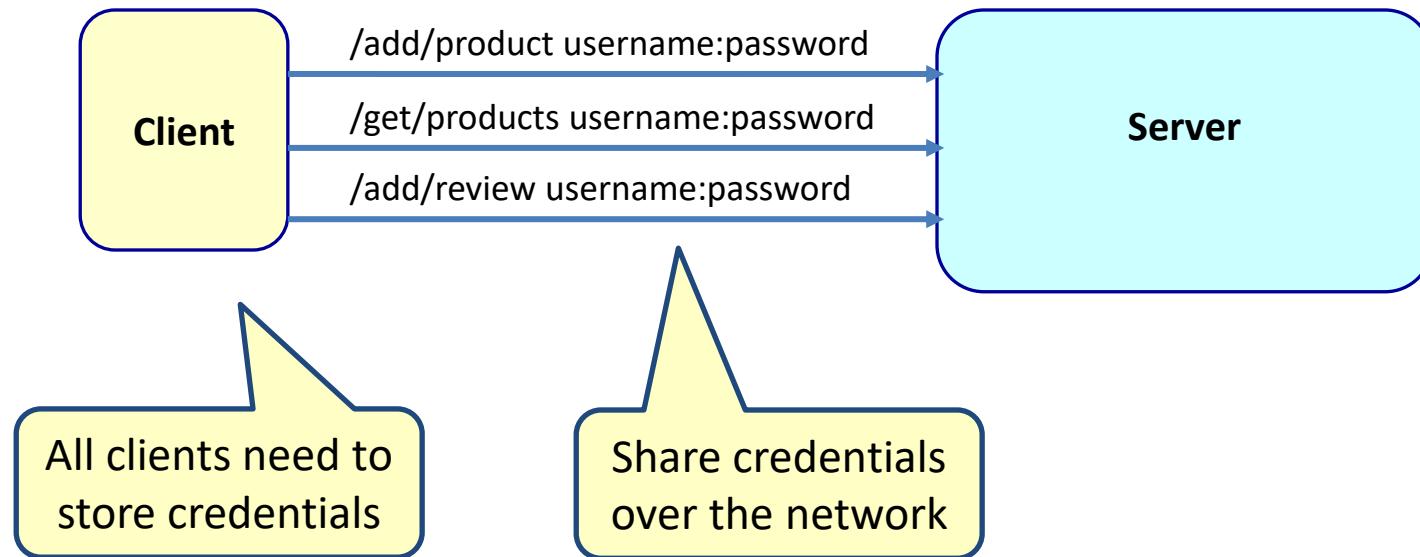


Lesson 11

MICROSERVICES SECURITY

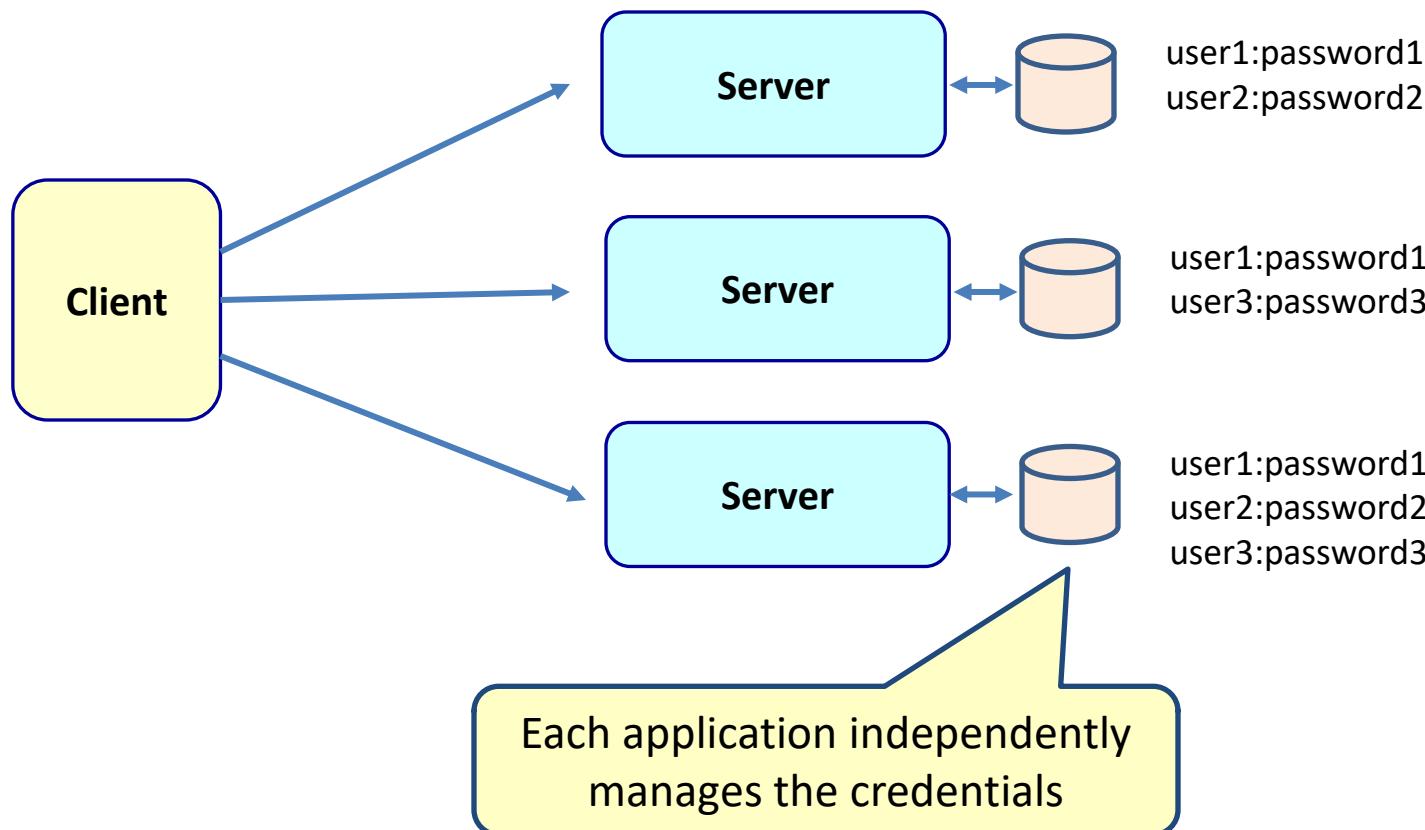
Problems with HTTP basic authentication

- We have to send credentials for every request



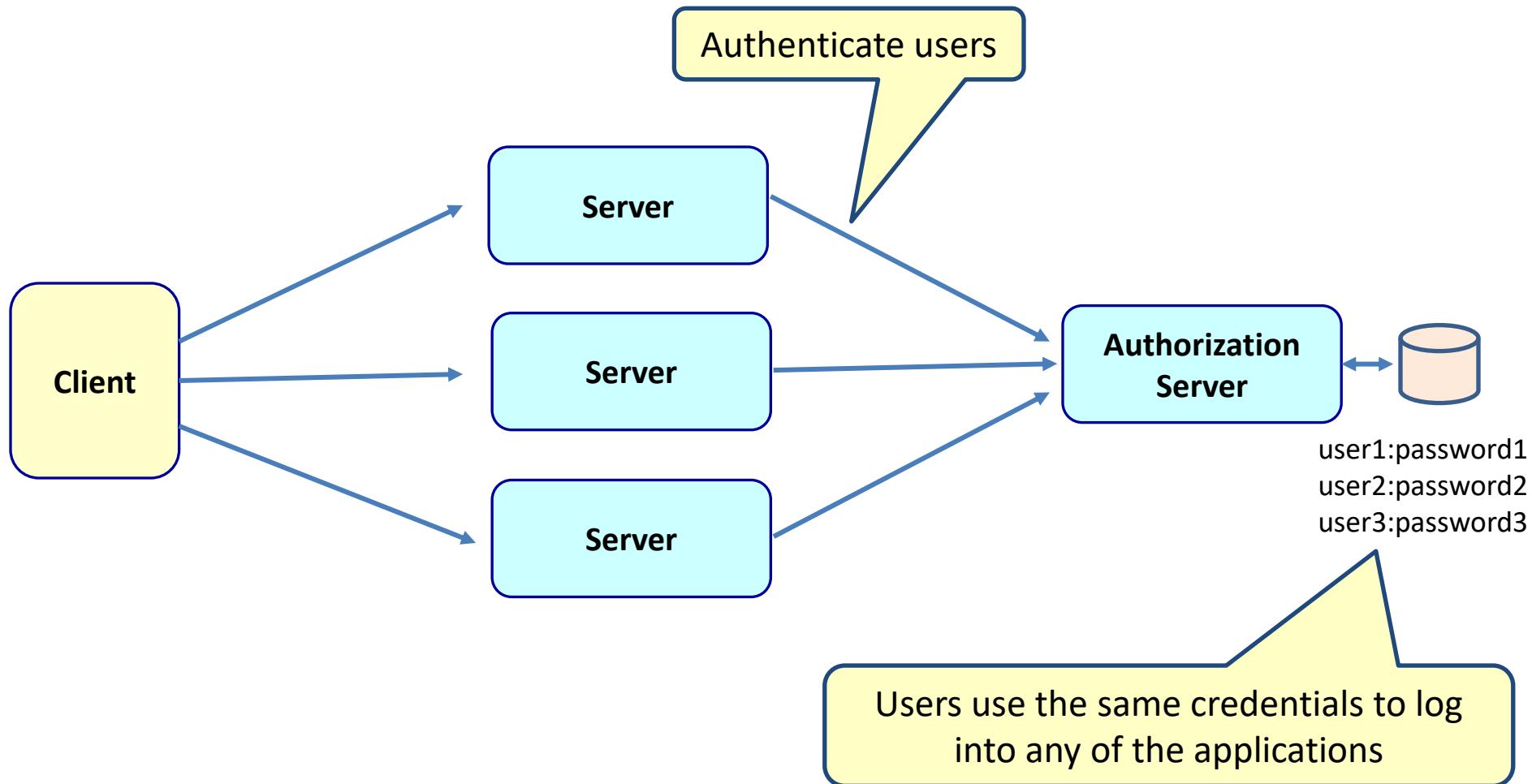
Problems with HTTP basic authentication

- Every system needs to manage these credentials

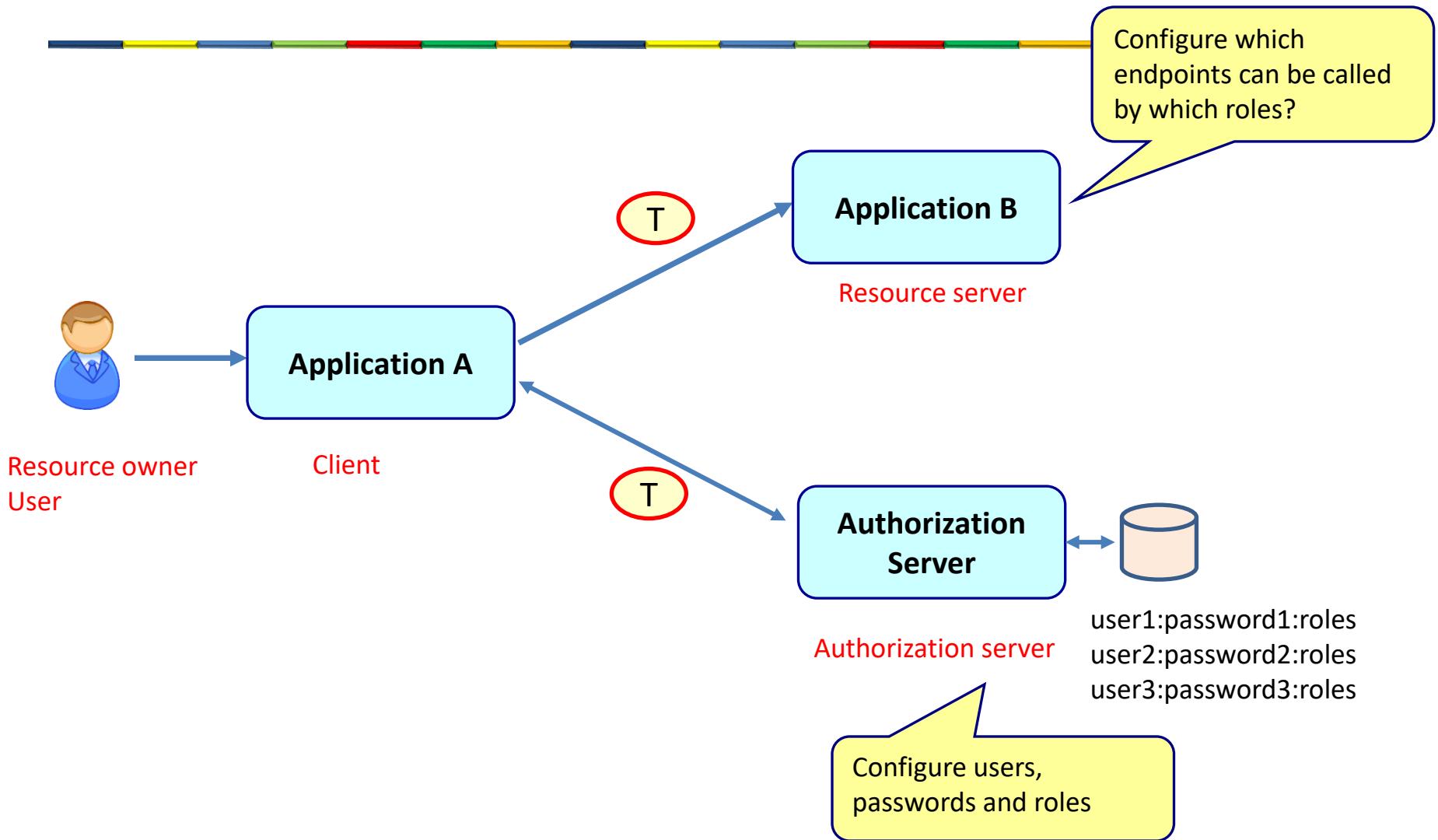


Better solution

- Authorization server



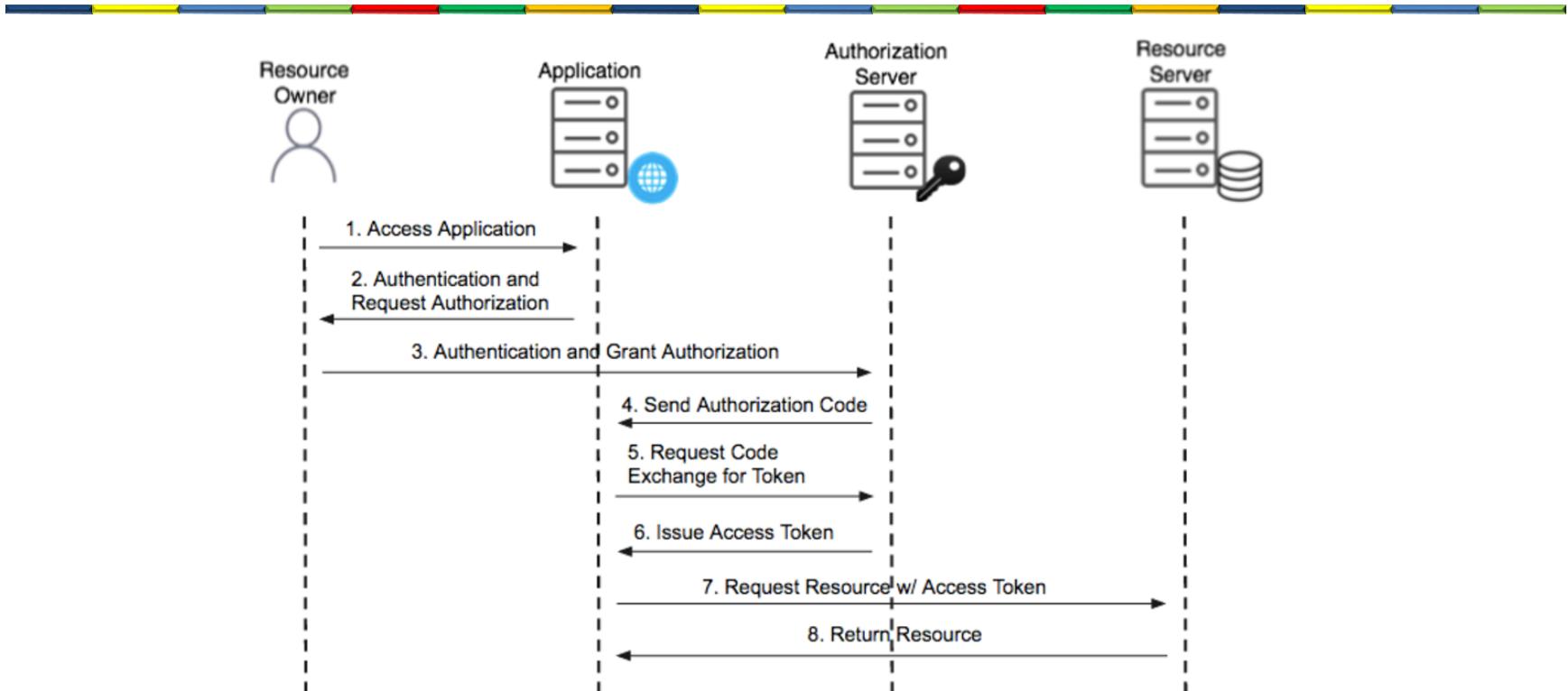
Parts of OAuth 2



OAuth 2 grants

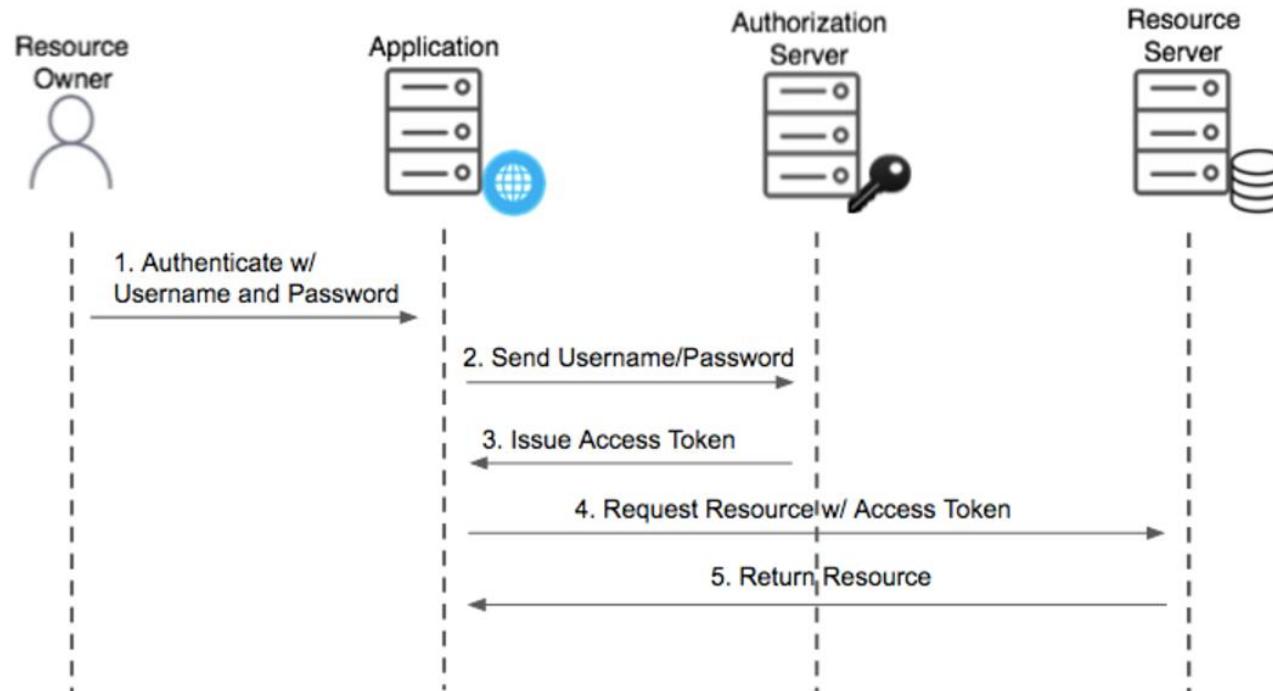
- Authorization code
- Password
- Client credentials
- Refresh token

Authorization code grant



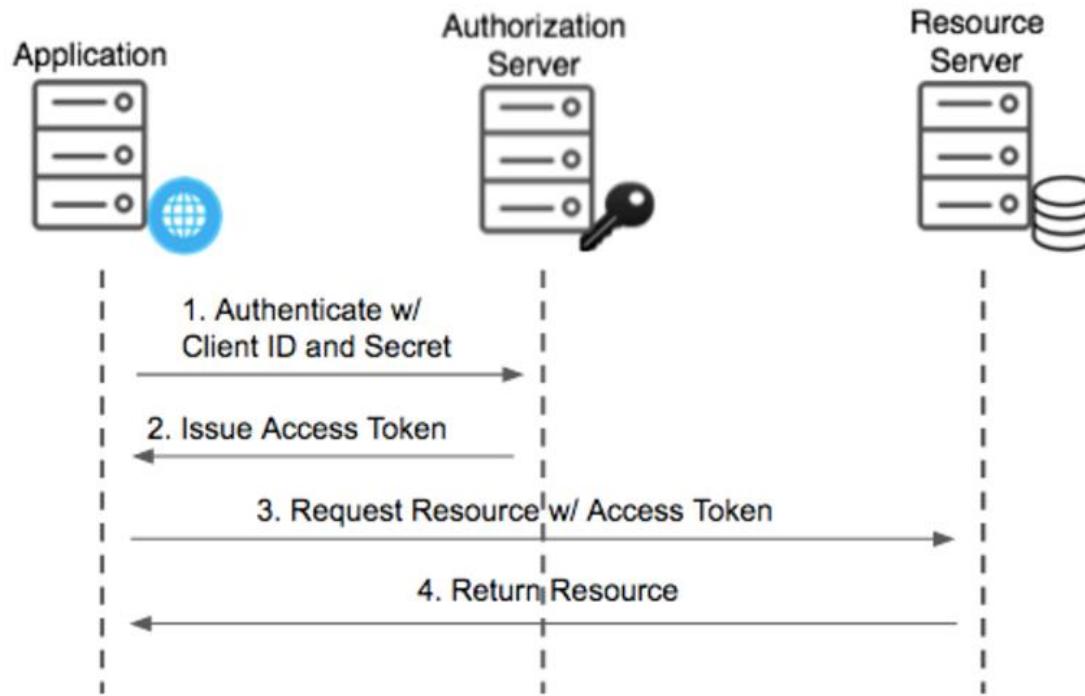
- This flow is widely used for its robust security.
- **Pros:** High security as authorization codes and access tokens are separate; ideal for server-side applications that can securely store client secrets.
- **Cons:** Involves multiple redirects between the client, authorization server, and resource server

Password grant



- This method directly uses user credentials (username and password).
- **Pros:** Simple implementation, suitable for trusted first-party applications.
- **Cons:** Lower security due to direct exposure of user credentials; not recommended for public clients.

Client credentials grant



- This grant is specifically for scenarios where applications act on their behalf rather than on behalf of a user.
- **Pros:** Effective for machine-to-machine interactions that do not require user involvement.
- **Cons:** Access permissions are granted to the application itself, requiring careful consideration of the permissions allowed.

Token

- Tokens can expire
 - Avoid tokens that don't expire
- Why refresh token?
 - User does not have to login again after token is expired
 - Client does not need to store user credentials

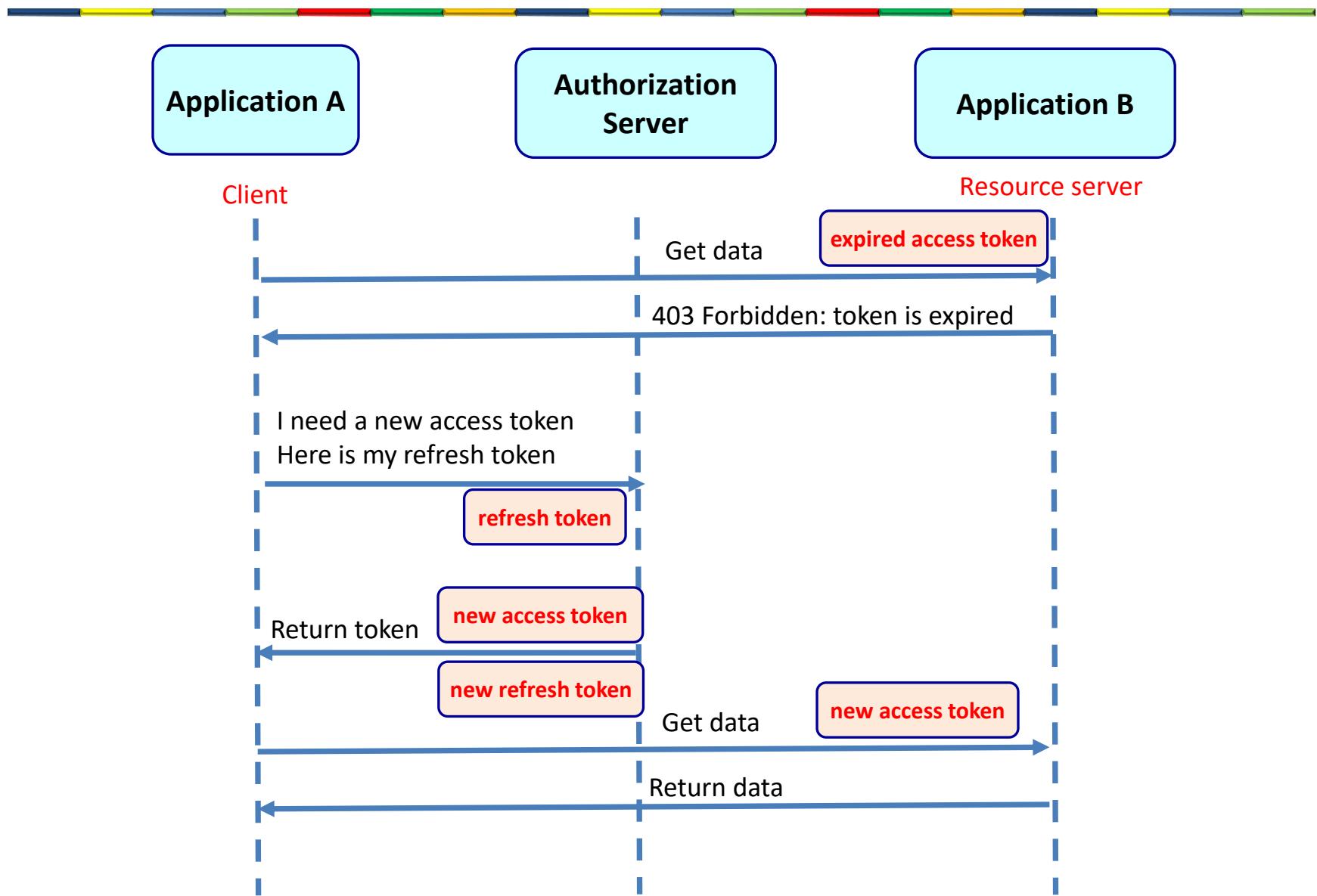
The screenshot shows a REST API response in JSON format. The response body is:

```
1 {  
2   "access_token": "b56f8dc4-4a19-411c-86f9-96c1c502f9a7",  
3   "token_type": "bearer",  
4   "refresh_token": "9618262c-0a5a-458d-afce-c45a544b5aad",  
5   "expires_in": 43199,  
6   "scope": "webclient"  
7 }
```

Annotations explain the fields:

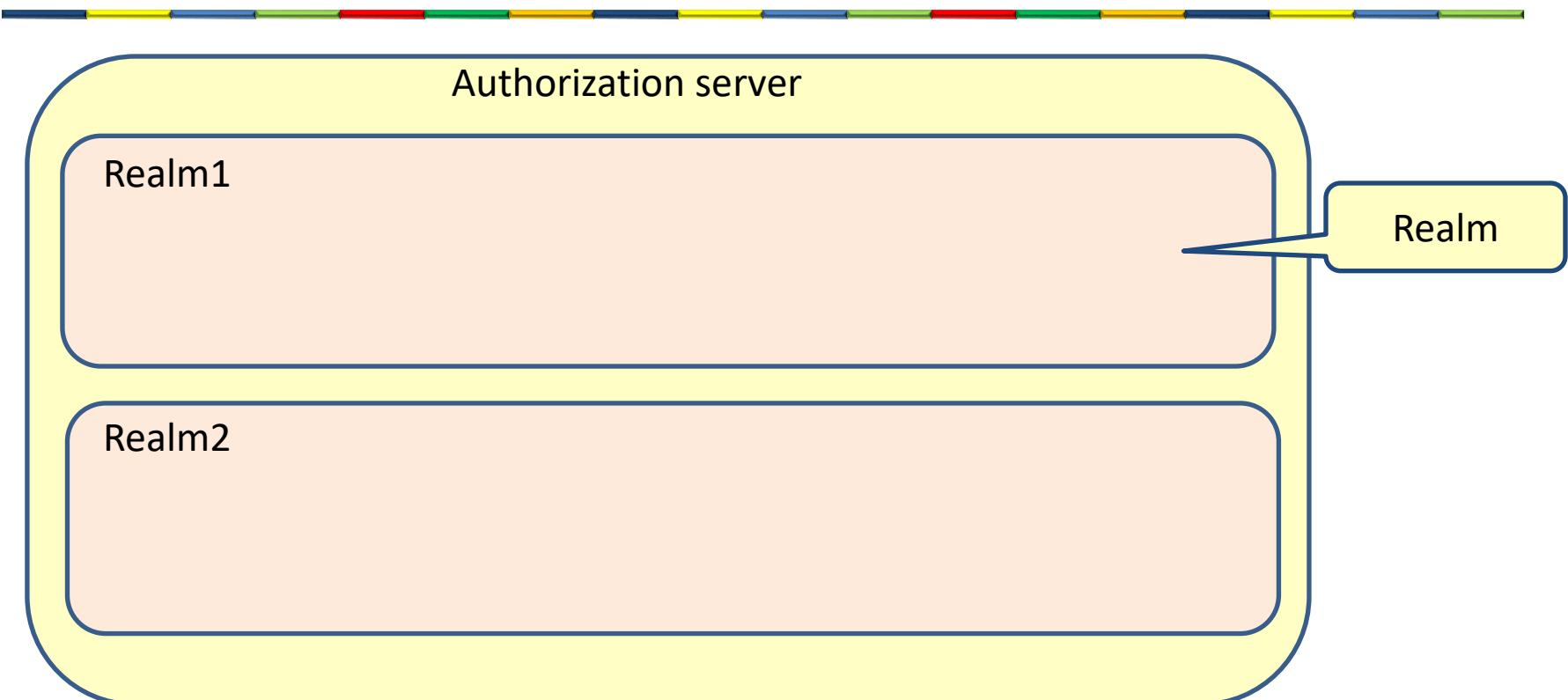
- Access token is presented with each call (points to the access_token field)
- Refresh token (points to the refresh_token field)
- Number of seconds before the token expires in seconds (points to the expires_in field)

Refresh token grant



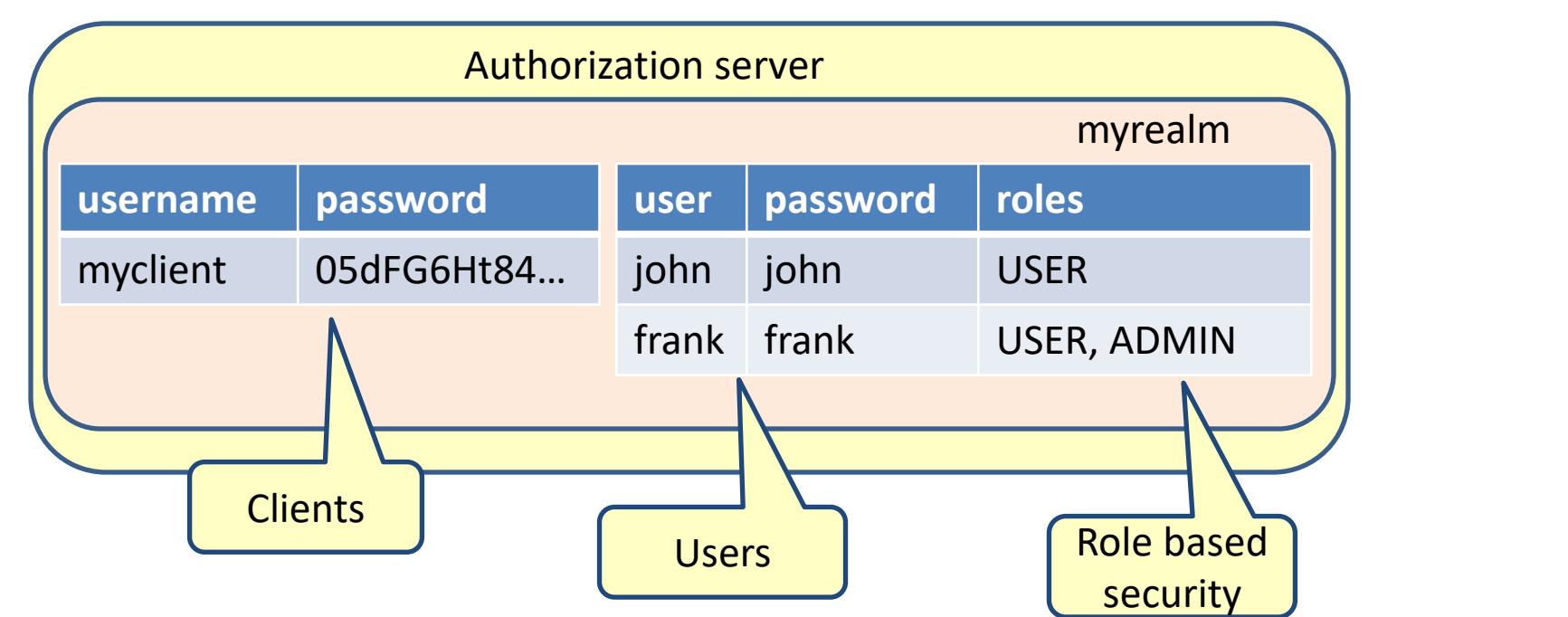
OAUTH2 AUTHORIZATION SERVICE

OAuth2 authorization server



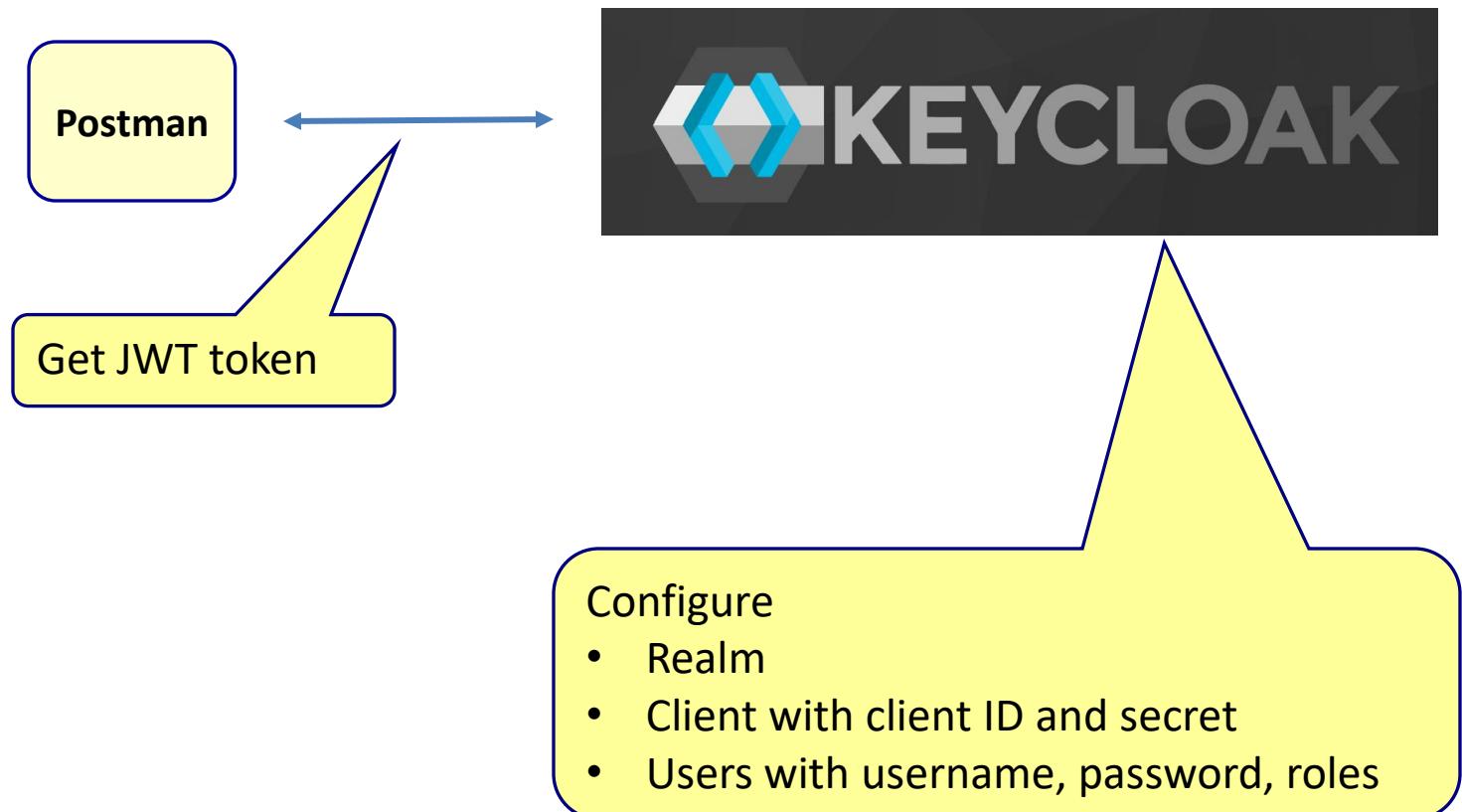
- A **realm** is a **logical isolation boundary** — basically, a **container** that holds **users, clients, roles, and configurations**.

OAuth2 authorization server



- A **client** is an application that wants security information
- A **user** is a human that want to access a secure application (client)

Keycloak authorization server



Keycloak URL's



localhost:8090/realms/myrealm/.well-known/openid-configuration

Pretty-print

```
{  
  "issuer": "http://localhost:8090/realms/myrealm",  
  "authorization_endpoint": "http://localhost:8090/realms/myrealm/protocol/openid-connect/auth",  
  "token_endpoint": "http://localhost:8090/realms/myrealm/protocol/openid-connect/token",  
  "introspection_endpoint": "http://localhost:8090/realms/myrealm/protocol/openid-connect/token/introspect",  
  "userinfo_endpoint": "http://localhost:8090/realms/myrealm/protocol/openid-connect/userinfo",  
  "end_session_endpoint": "http://localhost:8090/realms/myrealm/protocol/openid-connect/logout",  
  "frontchannel_logout_session_supported": true,  
  "frontchannel_logout_supported": true,  
  "jwks_uri": "http://localhost:8090/realms/myrealm/protocol/openid-connect/certs",  
  "check_session_iframe": "http://localhost:8090/realms/myrealm/protocol/openid-connect/login-status-iframe.html",  
  "grant_types_supported": [  
    "authorization_code",  
    "client_credentials",  
    "implicit",  
    "password",  
    "refresh_token",  
    "urn:ietf:params:oauth:grant-type:device_code",  
    "urn:ietf:params:oauth:grant-type:token-exchange",  
    "urn:ietf:params:oauth:grant-type:uma-ticket",  
    "urn:openid:params:grant-type:ciba"  
,  
  "acr_values_supported": [  
    "0",  
    "1"  
]
```

Retrieve a token

The screenshot shows the Postman interface with a POST request to `http://localhost:8090/realm/myrealm/protocol/openid-connect/token`. The 'Auth' tab is selected, showing 'Basic Auth' chosen. The 'Username' field contains 'myclient' and the 'Password' field contains a redacted password. A note on the left explains that the authorization header will be generated automatically.

The screenshot shows the Postman interface with a POST request to `http://localhost:8090/realm/myrealm/protocol/openid-connect/token`. The 'Body' tab is selected, showing 'x-www-form-urlencoded' chosen. The table below lists the form-data fields:

Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/> grant_type	password		
<input checked="" type="checkbox"/> username	john		
<input checked="" type="checkbox"/> password	john		
<input checked="" type="checkbox"/> scope	openid roles		
<input type="checkbox"/>			

Returned payload

The diagram illustrates a JSON object structure with five numbered callouts pointing to specific fields:

- Access_token** (Callout 1) points to the "access_token" field.
- Number of seconds before the token expires** (Callout 2) points to the "expires_in" field.
- Refresh token** (Callout 3) points to the "refresh_token" field.
- Number of seconds before the token expires** (Callout 4) points to the "refresh_expires_in" field.
- Number of seconds before the token expires** (Callout 5) points to the "refresh_token" field.

```
"access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJyeHBmMDJqMWF0UlgsS012SHdLTnZXVFoxcmdzOHdDYlh5RENka11DamtFIn0.eyJleHAiOjE3Njc3NjYsImlhdCI6MTc2MjU2NDc2NiwianRpIjoib25ydHJvOjE40WE5MTk2LThkYWEtMzg3Zi0yZmQ2LTAYzE2ZWNiMTI4MiIsImlzcyI6Imh0dHA6Ly9sb2Nhbgvhc3Q6ODA5MC9yZWFBsXNbXlyZWFBsSIsmF1ZCI6ImFjY291bnQiLCJzdWIiOjJlMTE1NGFkYy0wYjBiLTQwNTQtYWJ1Mi05ZjczNjNkM2FiNjUiLCJ0eXAiOjJCZWfyzXiiLCJhenAiOjJteWNsaWVu dCIsInNpZCI6Ijg4NmFhOTAyLTZhMTEtM2Y4Ny0wNWExLTQ2MTFiMzU1MjU0ZCIsImFjciI6IjEiLCJhbGxvd2VklW9yaWdpbnMi0lsilYoiXSwicmVhbG1fYWNgZXNzIjp7InJvbGVzIjpbImR1ZmF1bhQtc9sZXMtBxlyZWFBsSIsm9mZmxpbmVfYWNgZXNzIwidW1hX2F1dGhvcm16YXRpb24iXX0sInJlc291cmN1X2FjY2VzcyI6eyJteWNsaWvudCI6eyJyb2xlcyI6WyJVU0VSI119LCJhY2NvdW50Ijp7InJvbGVzIjpbIm1hbmfnsZs1hY2NvdW50IiwibWFyWd1LWFjY291bnQtbGlua3MiLCJ2aWV3LXByb2ZpbGuIXX19LCJzY29wZSI6Im9wZW5pZCB1bWFpbCBwcm9maWxlIwiZW1haWxfdmVyaWZpZWQjOnRydWUsIm5hbWUiOjqb2huIGRvZSIisInByZWZlcnJlZF91c2VybmrTzS16ImpvaG4iLCJnaXZ1b19uYW11Ijoiam9obiIsImZhbWlseV9uYW11IjoizG91IiwiZW1haWwiOjqb2huQGdtYWlsLmNvbsJ9. nkt1Xac0sCziWLWdgV2PzSFaW-aTBjkFvFvixrynbisogmUyKD1AljoWUtLqd_hLx6kWtTaMzc o-MCXZj3U5QyWWtXEsxCA-d4eDUTJE1jEG06yQIN2d3NiZcbLrYyP6B71Nd33oh7UEchBeXW0e F-b1HUYs78IxysYwFJE0cxXhD5fu6wtrDWB5DKY91W3z5gJXujsNhi9VQ6g0evEaYWBD_iKWmZ0 sefj_IuxiWQj5jNCnFFVj6nrhuF0VoqXAonf5a3PsUQtqy5quk2yg60MUssSeGB_0FUb9leV1sNzXNmuWd1SAoFo--9gizhG4anrFAAA2PsYt0zSgRBW8wqfA", "expires_in": 3000, "refresh_expires_in": 1800, "refresh_token": "eyJhbGciOiJIUzUxMiIsInR5cCIgOiAiSldUIiwia2lkIiA6IC12YWU4ZjNkOS1hYmN1LTQ0YzUtYmQxYy0yMzdjMzMwY2MxY2IifQ.eyJleHAiOjE3Njc1NjY1NjYsImlhdCI6MTc2MjU2NDc2NiwianRpIjoizGM0ZjBiMTQtOGUwMy04NjI1LTU1MmEtNDQ1ZWZ1YzcxZDdiIiwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo4MDkwL3J1YWxtcy9teXJ1YWxtIiwiC3ViIjoizTExNTRhZGMtMGIwYi00MDU0LWFfZTIt0WY3MzYzZDNhYjY1IiwidHlwIjoiUmVmcmVzaCIsImF6cCI6Im15Y2xpZW50IiwiC2lkIioi0Dg2YWE5MDItNmExMS0zZig3LTA1YT
```

Get user information

The screenshot shows a Postman API request to retrieve user information. The URL is `http://localhost:8090/realm/myrealm/protocol/openid-connect/userinfo`. The request method is GET. The 'Auth' tab is selected, showing 'Bearer Token' as the type. A yellow callout points to the token field, which contains a redacted string and icons for copy, eye, and lock. The 'Body' tab shows a JSON response with user details. A yellow callout points to the response body, which is labeled 'User information from john'. The response status is 200 OK with 97 ms latency and 432 B size.

GET http://localhost:8090/realm/myrealm/protocol/openid-connect/userinfo Send

Params Auth Headers (9) Body Scripts Tests Settings Cookies

Auth Type

Bearer Token

The authorization header will be automatically generated when you send the request.

Learn more about [Bearer Token](#) authorization.

Token

.....

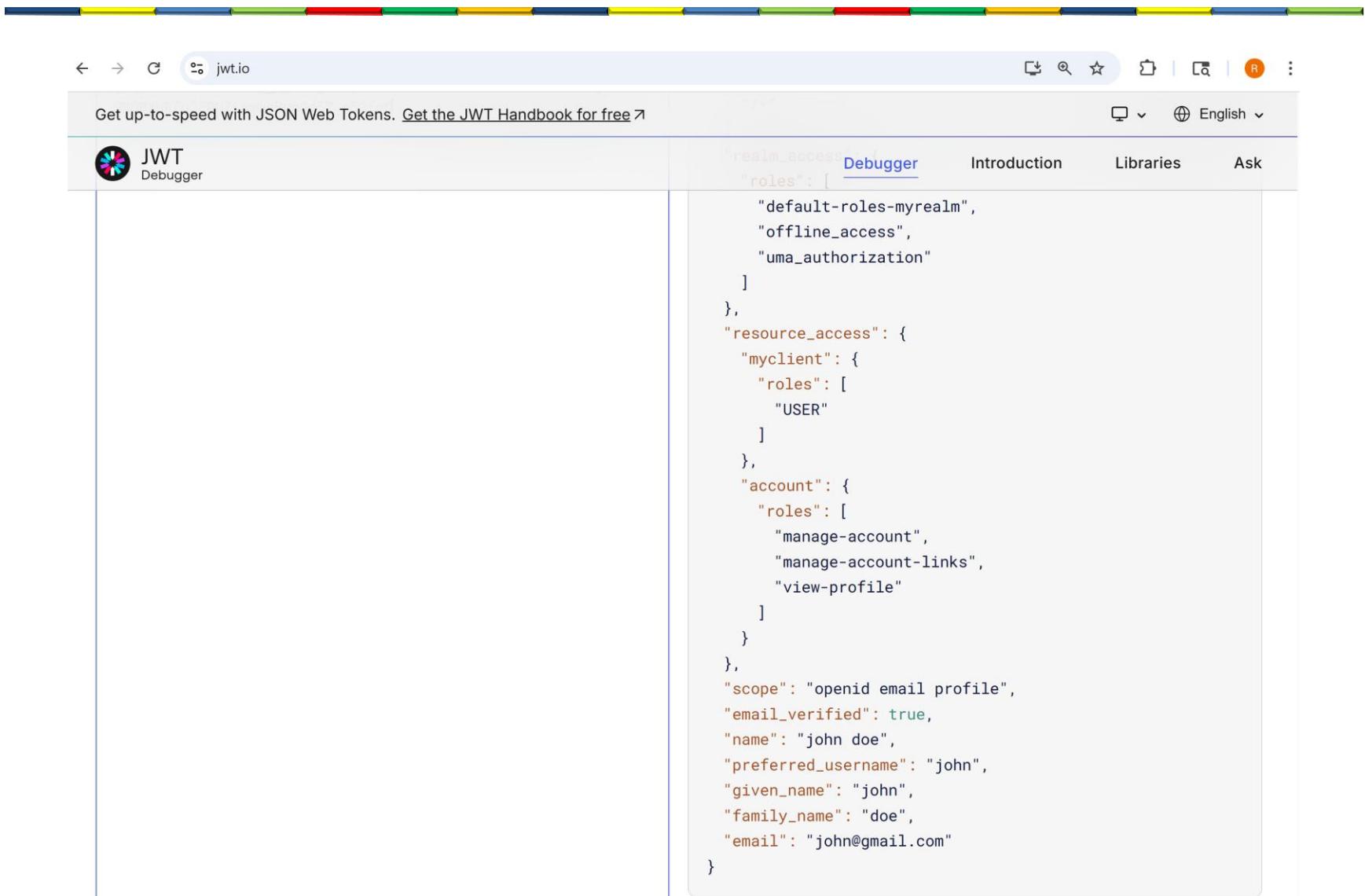
200 OK • 97 ms • 432 B •

{ } JSON Preview Visualize

1 {
2 "sub": "e1154adc-0b0b-4054-abe2-9f7363d3ab65",
3 "email_verified": true,
4 "name": "john doe",
5 "preferred_username": "john",
6 "given_name": "john",
7 "family_name": "doe",
8 "email": "john@gmail.com"
9 }

User information from john

See token content

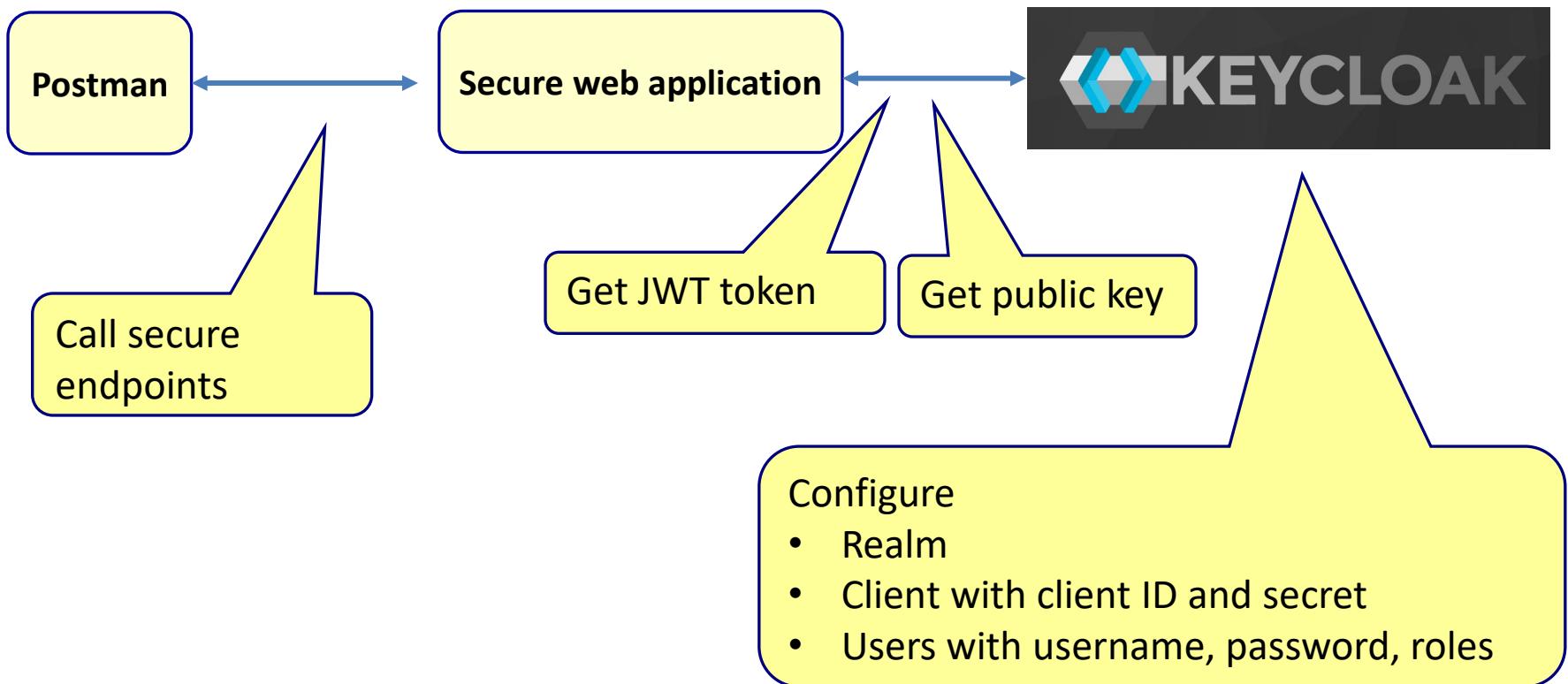


The screenshot shows a browser window for jwt.io. The URL bar has "jwt.io". The page title is "Get up-to-speed with JSON Web Tokens. [Get the JWT Handbook for free ↗](#)". The top navigation bar includes a "JWT Debugger" icon, "Introduction", "Libraries", and "Ask". The main content area displays a JSON object representing a JWT payload:

```
  "realm_access": {
    "roles": [
      "default-roles-myrealm",
      "offline_access",
      "uma_authorization"
    ]
  },
  "resource_access": {
    "myclient": {
      "roles": [
        "USER"
      ]
    },
    "account": {
      "roles": [
        "manage-account",
        "manage-account-links",
        "view-profile"
      ]
    }
  },
  "scope": "openid email profile",
  "email_verified": true,
  "name": "john doe",
  "preferred_username": "john",
  "given_name": "john",
  "family_name": "doe",
  "email": "john@gmail.com"
}
```

A SECURE APPLICATION

Secure application



A secure application



Authentication server

username	password
myclient	4H0uT54...

user	password	roles
john	john	USER
frank	frank	USER, ADMIN

Secure application

URL	Roles
/phone	USER
/salary	ADMIN

A secure application

```
@RestController  
public class EmployeeController {  
    @GetMapping("/name")  
    public String getName() {  
        return "Frank Brown";  
    }  
  
    @PreAuthorize("hasRole('ADMIN')")  
    @GetMapping("/salary")  
    public String getGetSalary() {  
        return "95.000";  
    }  
  
    @PreAuthorize("hasRole('USER')")  
    @GetMapping("/phone")  
    public String getPhone() {  
        return "645322899";  
    }  
}
```

Method based security

Security configuration

```
@Configuration  
@EnableWebSecurity  
@EnableMethodSecurity  
public class SecurityConfig {  
    @Bean  
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
        http.httpBasic(Customizer.withDefaults());  
  
        http  
            .oauth2ResourceServer(oauth2 -> oauth2  
                .jwt(jwt -> jwt.jwtAuthenticationConverter(jwtAuthenticationConverter())));  
  
        http  
            .sessionManagement(session -> session  
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)  
            );  
  
        return http.build();  
    }  
}
```

Method based security

Security configuration

```
@Bean
public JwtAuthenticationConverter jwtAuthenticationConverter() {
    JwtAuthenticationConverter converter = new JwtAuthenticationConverter();
    converter.setJwtGrantedAuthoritiesConverter(jwt -> {
        Set<String> roles = new HashSet<>();

        // Realm roles
        Map<String, Object> realmAccess = jwt.getClaimAsMap("realm_access");
        if (realmAccess != null && realmAccess.get("roles") instanceof Collection<?> realmRoles) {
            roles.addAll(realmRoles.stream().map(Object::toString).toList());
        }

        // Client roles (replace "myclient" with your client ID)
        Map<String, Object> resourceAccess = jwt.getClaimAsMap("resource_access");
        if (resourceAccess != null && resourceAccess.get("myclient") instanceof Map<?, ?> clientAccess) {
            Object clientRoles = ((Map<?, ?>) clientAccess).get("roles");
            if (clientRoles instanceof Collection<?> clientRolesList) {
                roles.addAll(clientRolesList.stream().map(Object::toString).toList());
            }
        }

        // Convert to SimpleGrantedAuthority with ROLE_ prefix
        return roles.stream()
            .map(role -> new SimpleGrantedAuthority("ROLE_" + role))
            .collect(Collectors.toSet());
    });
    return converter;
}
```

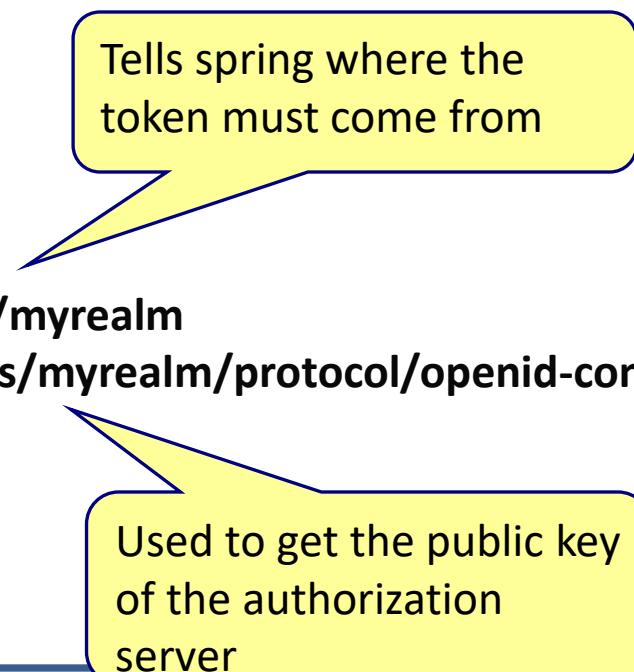
Make sure we get both the realm and the client roles

Spring expects “ROLE_USER” instead of “USER”

The configuration

application.yml

```
spring:  
  security:  
    oauth2:  
      resourceserver:  
        jwt:  
          issuer-uri: http://localhost:8090/realm/myrealm  
          jwk-set-uri: http://localhost:8090/realm/myrealm/protocol/openid-connect/certs  
  
      server:  
        port: 8081
```



Tells spring where the token must come from

Used to get the public key of the authorization server

Get the user name

The screenshot shows the POSTMAN application interface. At the top, there is a header bar with a color gradient. Below it, the main interface has a search bar with "GET" and "http://localhost:8081/name". To the right of the search bar is a "Send" button. Below the search bar, there are tabs for "Params", "Auth", "Headers (8)", "Body", "Scripts", "Tests", and "Settings". The "Auth" tab is currently selected and underlined. A dropdown menu labeled "No Auth" is open. On the right side of the interface, there is a section titled "No Auth" with the subtext "This request does not use any authorization." Below this, there is a "Body" section with a "Raw" tab selected, showing the response "200 OK" and the content "Frank Brown". There are also "Preview" and "Visualize" tabs. At the bottom right, there is a yellow callout bubble with the text "Everyone can get the user name without authorization".

Get the phone number

The screenshot shows a POSTMAN interface with the following details:

- Method:** GET
- URL:** http://localhost:8081/phone
- Auth Type:** No Auth
- Headers (8):** (This section is visible but empty)
- Body:** (This section is visible but empty)
- Tests:** (This section is visible but empty)
- Scripts:** (This section is visible but empty)
- Settings:** (This section is visible but empty)
- Cookies:** (This section is visible but empty)
- No Auth:** A note stating "This request does not use any authorization." with an info icon.
- Response Status:** 401 Unauthorized
- Response Time:** 13 ms
- Response Size:** 342 B
- Global:** (with a globe icon)
- More Options:** (with three dots icon)
- Raw:** (selected view)
- Preview:**
- Visualize:**
- Copy:** (with a clipboard icon)
- Search:** (with a magnifying glass icon)
- Link:** (with a link icon)
- Close:** (with a close icon)

A yellow callout bubble with a black border and blue outline points from the bottom right towards the "No Auth" note, containing the text: "You cannot get the phone number without access token".

Get the phone number

GET http://localhost:8081/phone

Auth Type: Bearer Token

The authorization header will be automatically generated when you send the request.

Body: 1 645322899

Response: 200 OK

John can get the phone number

GET http://localhost:8081/phone

Auth Type: Bearer Token

The authorization header will be automatically generated when you send the request.

Body: 1 645322899

Response: 200 OK

Frank can get the phone number

Get the salary

The authorization header will be automatically generated when you send the request.
Learn more about [Bearer Token](#) authorization.

Frank can get the salary (role=ADMIN)

John cannot get the salary (role = USER)

200 OK • 14 ms • 339 B

1 95.000

403 Forbidden • 9 ms • 509 B

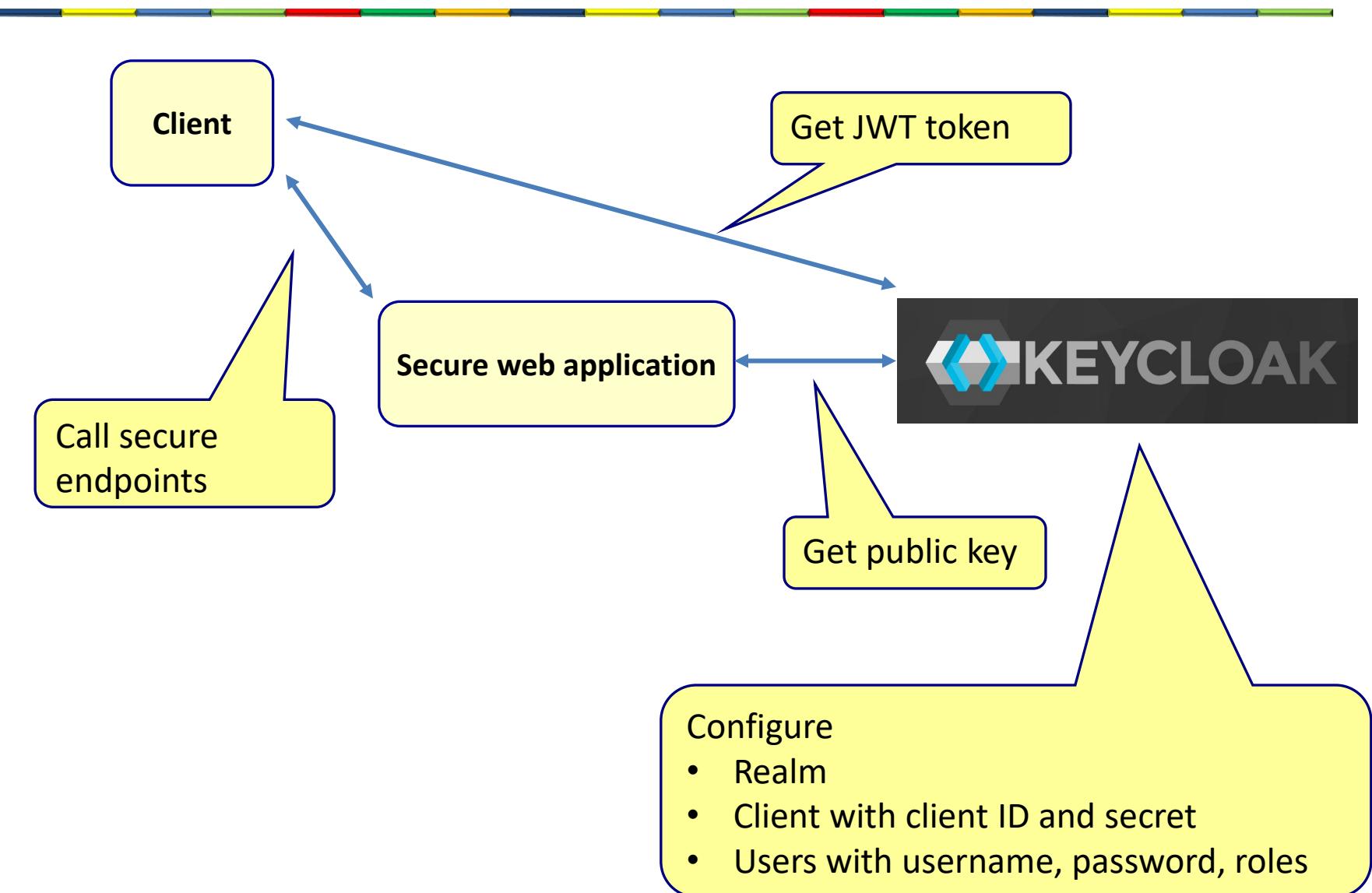
1

Main point

- To implement security in a microservice architecture we use token based security (OAuth2). *The human nervous system is able to access that most basic field of pure consciousness which is the source of all the laws of Nature.*

CLIENT FOR THE SECURE APPLICATION

Client for the secure application



Client for the secure application

```
@SpringBootApplication
public class Client implements CommandLineRunner {

    public static void main(String[] args) {
        SpringApplication.run(Client.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        callRemoteService("http://localhost:8081/name", "nobody", "nobody");
        callRemoteService("http://localhost:8081/phone", "john", "john");
        callRemoteService("http://localhost:8081/phone", "frank", "frank");
        callRemoteService("http://localhost:8081/salary", "john", "john");
        callRemoteService("http://localhost:8081/salary", "frank", "frank");
    }
}
```

No token for user nobody

Call to http://localhost:8081/name for user nobody gave the response Frank Brown

Call to http://localhost:8081/phone for user john gave the response 645322899

Call to http://localhost:8081/phone for user frank gave the response 645322899

Forbidden: insufficient roles http://localhost:8081/salary with user john

Call to http://localhost:8081/salary for user john gave the response null

Call to http://localhost:8081/salary for user frank gave the response 95.000

Client of the secure application

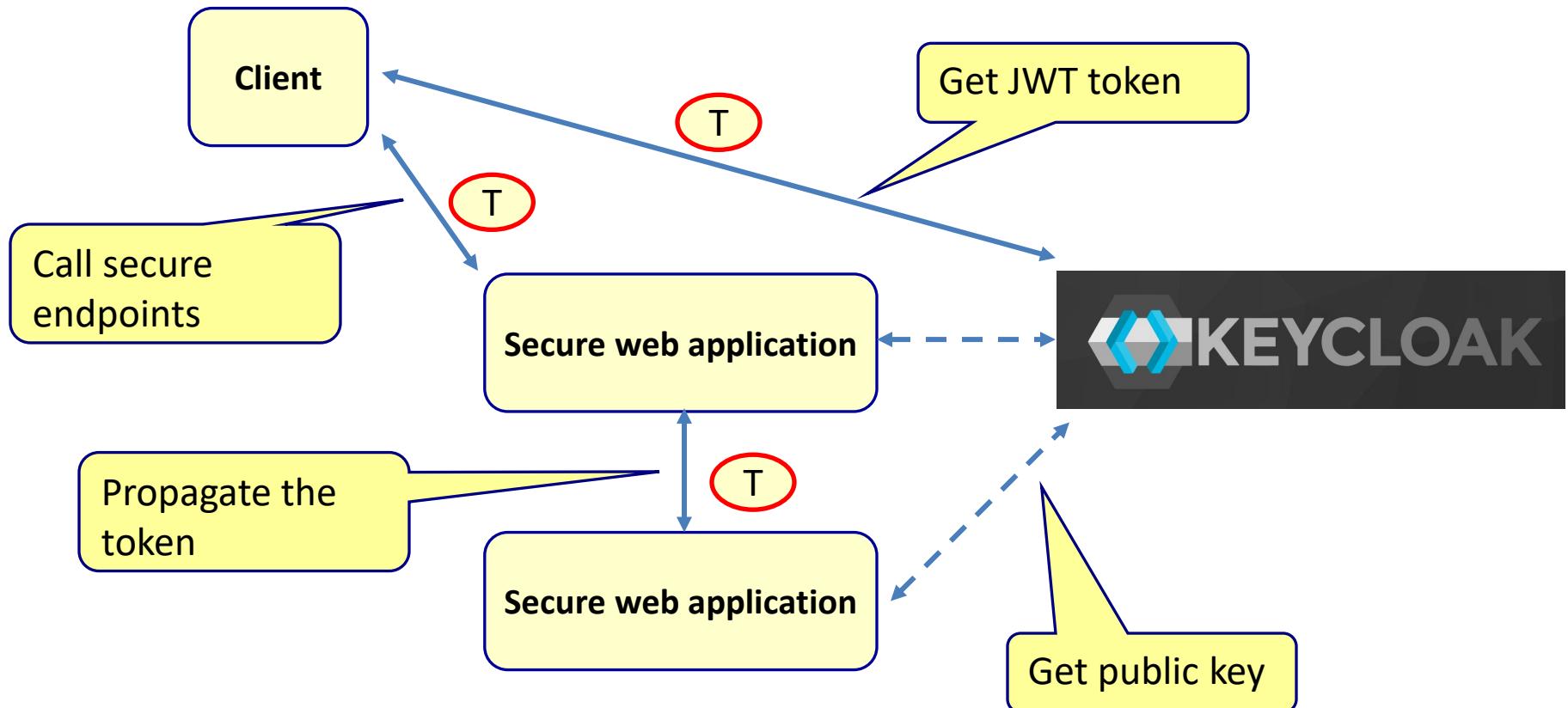
```
public void callRemoteService(String url, String username, String password){  
    String accessToken = getAccessToken(username, password);  
    // Call resource server  
    String response = WebClient.create(url)  
        .get()  
        .headers(h ->{  
            if (accessToken != null)  
                h.setBearerAuth(accessToken);  
        })  
        .retrieve()  
        .onStatus(status -> status.value() == 401, clientResponse -> {  
            System.out.println("Error: Unauthorized - token missing or expired for call "+url+" with user "+username);  
            return Mono.empty(); // Do NOT throw  
        })  
        .onStatus(status -> status.value() == 403, clientResponse -> {  
            System.out.println("Forbidden: insufficient roles "+url+" with user "+username);  
            return Mono.empty(); // Do NOT throw  
        })  
        .bodyToMono(String.class)  
        .block();  
  
    System.out.println("Call to " +url+ " for user "+username+" gave the response "+ response);  
}
```

Client of the secure application

```
public String getAccessToken(String username, String password){  
    WebClient webClient = WebClient.builder()  
        .baseUrl("http://localhost:8090/realm/myrealm/protocol/openid-connect/token")  
        .build();  
    // Get access token  
    Map<String, Object> tokenResponse = webClient.post()  
        .header("Content-Type", "application/x-www-form-urlencoded")  
        .body(BodyInserters.fromFormData("grant_type", "password")  
            .with("username", username)  
            .with("password", password)  
            .with("scope", "openid roles")  
            .with("client_id", "myclient")  
            .with("client_secret", "0maxLeON2d9gwyF1ilv52YBxgB5AoXGL"))  
        .retrieve()  
        .onStatus(status -> status.value() == 401, clientResponse -> {  
            System.out.println("No token for user "+username);  
            return Mono.empty(); // Do NOT throw  
        })  
        .bodyToMono(Map.class)  
        .block();  
  
    String accessToken = (String) tokenResponse.get("access_token");  
    return accessToken;  
}
```

PROPAGATING THE TOKEN

Client for the secure application



Downstream secure application

```
@RestController  
public class PrivateDataController {  
    @PreAuthorize("hasRole('ADMIN')")  
    @GetMapping("/private")  
    public String getPrivateData() {  
        return "This is very private data";  
    }  
}
```

application.yml

```
spring:  
  security:  
    oauth2:  
      resourceserver:  
        jwt:  
          issuer-uri: http://localhost:8090/realm/myrealm  
          jwk-set-uri: http://localhost:8090/realm/myrealm/protocol/openid-connect/certs  
  
  server:  
    port: 8082
```

Upstream secure application

```
@PreAuthorize("hasRole('ADMIN')")
@GetMapping("/sensitive")
public String getPrivateData(@AuthenticationPrincipal Jwt jwt) {
    String token = jwt.getTokenValue();
    WebClient webClient = WebClient.builder()
        .baseUrl("http://localhost:8082")
        .build();
    // Call the downstream application with the same token
    String response = webClient.get()
        .uri("/private")
        .headers(headers -> headers.setBearerAuth(token))
        .retrieve()
        .bodyToMono(String.class)
        .block();

    return response;
}
```

Get JWT token

Propagate the token

Change the Client

```
@Override  
public void run(String... args) throws Exception {  
    callRemoteService("http://localhost:8081/name", "nobody", "nobody");  
    callRemoteService("http://localhost:8081/phone", "john", "john");  
    callRemoteService("http://localhost:8081/phone", "frank", "frank");  
    callRemoteService("http://localhost:8081/salary", "john", "john");  
    callRemoteService("http://localhost:8081/salary", "frank", "frank");  
    callRemoteService("http://localhost:8081/sensitive", "frank", "frank");  
}
```

No token for user nobody

Call to http://localhost:8081/name for user nobody gave the response Frank Brown

Call to http://localhost:8081/phone for user john gave the response 645322899

Call to http://localhost:8081/phone for user frank gave the response 645322899

Forbidden: insufficient roles http://localhost:8081/salary with user john

Call to http://localhost:8081/salary for user john gave the response null

Call to http://localhost:8081/salary for user frank gave the response 95.000

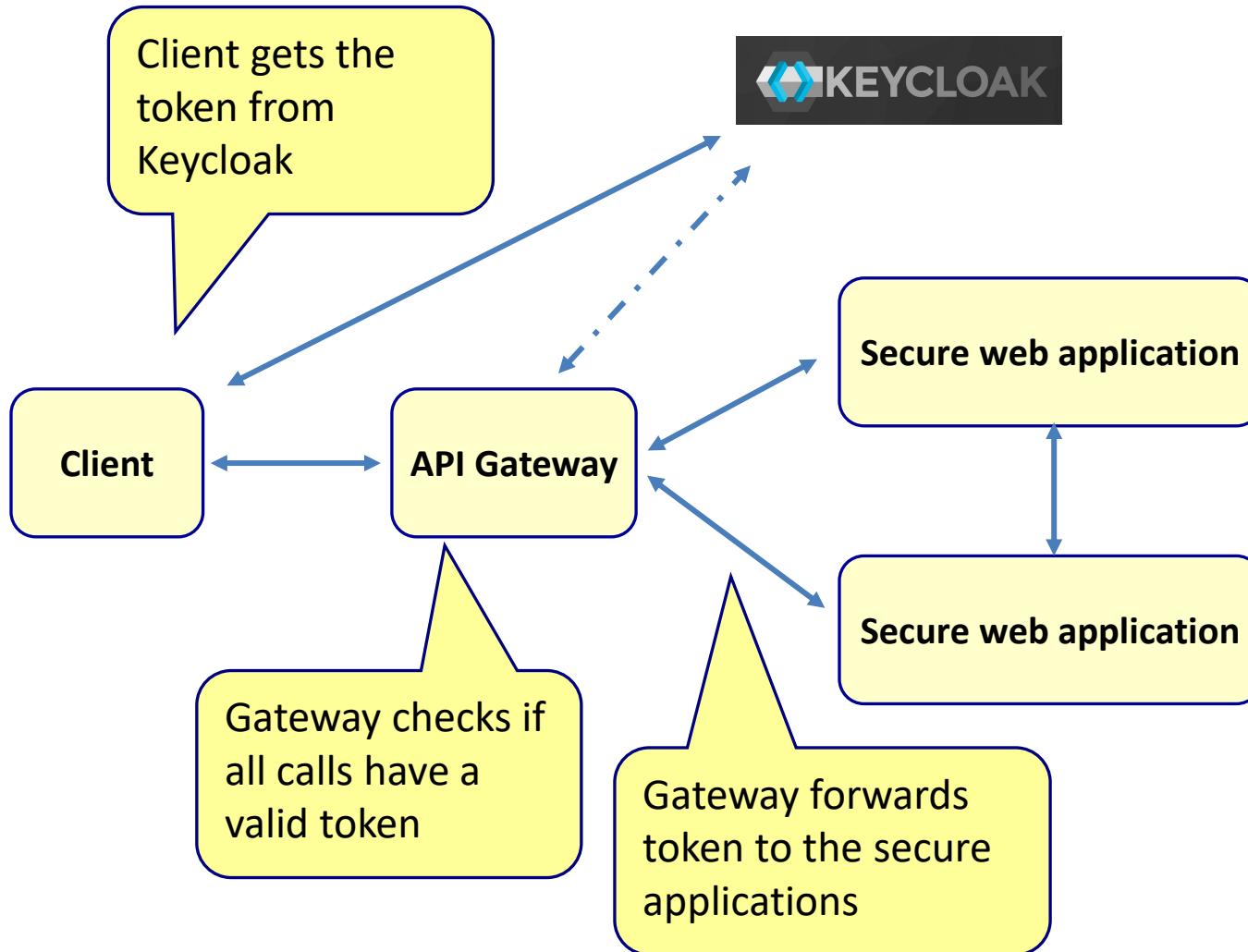
Call to http://localhost:8081/sensitive for user frank gave the response This is very private data

Main point

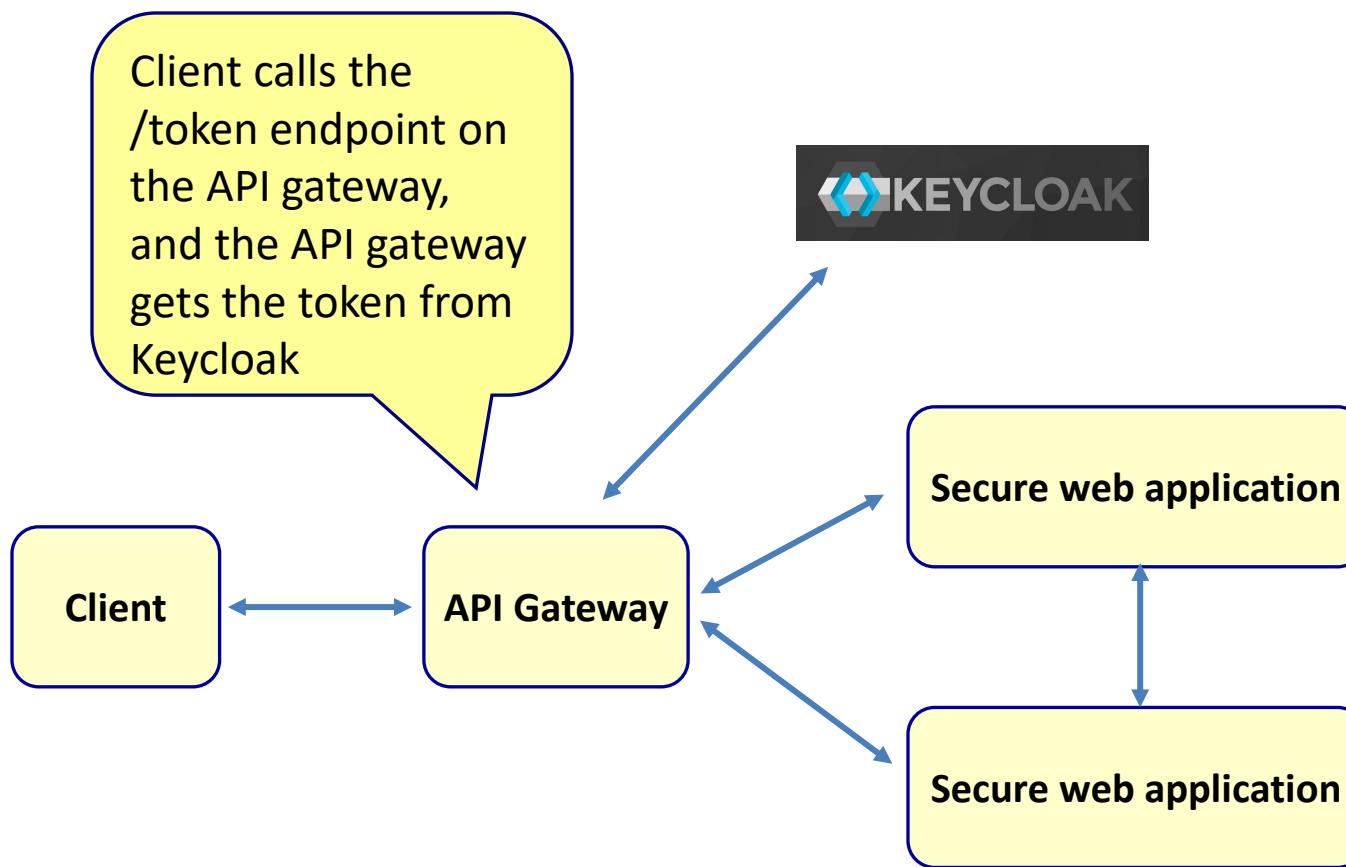
- With JWT a token receiver can verify the correctness of the token using the signature within the token. *The TM technique is the key to transcend and access pure consciousness.*

LETS ADD THE API GATEWAY

API Gateway option 1



API Gateway option 2



API Gateway security

```
spring:  
  application:  
    name: api-gateway  
  security:  
    oauth2:  
      resourceserver:  
        jwt:  
          issuer-uri: http://localhost:8090/realm/myrealm  
          jwk-set-uri: http://localhost:8090/realm/myrealm/protocol/openid-connect/certs  
  cloud:  
    gateway:  
      server:  
        webflux:  
          routes:  
            - id: upstream  
              uri: http://localhost:8081  
              predicates:  
                - Path=/name, /phone, /salary, /sensitive  
  
            - id: downstream  
              uri: http://localhost:8082  
              predicates:  
                - Path=/private  
  server:  
    port: 8085
```

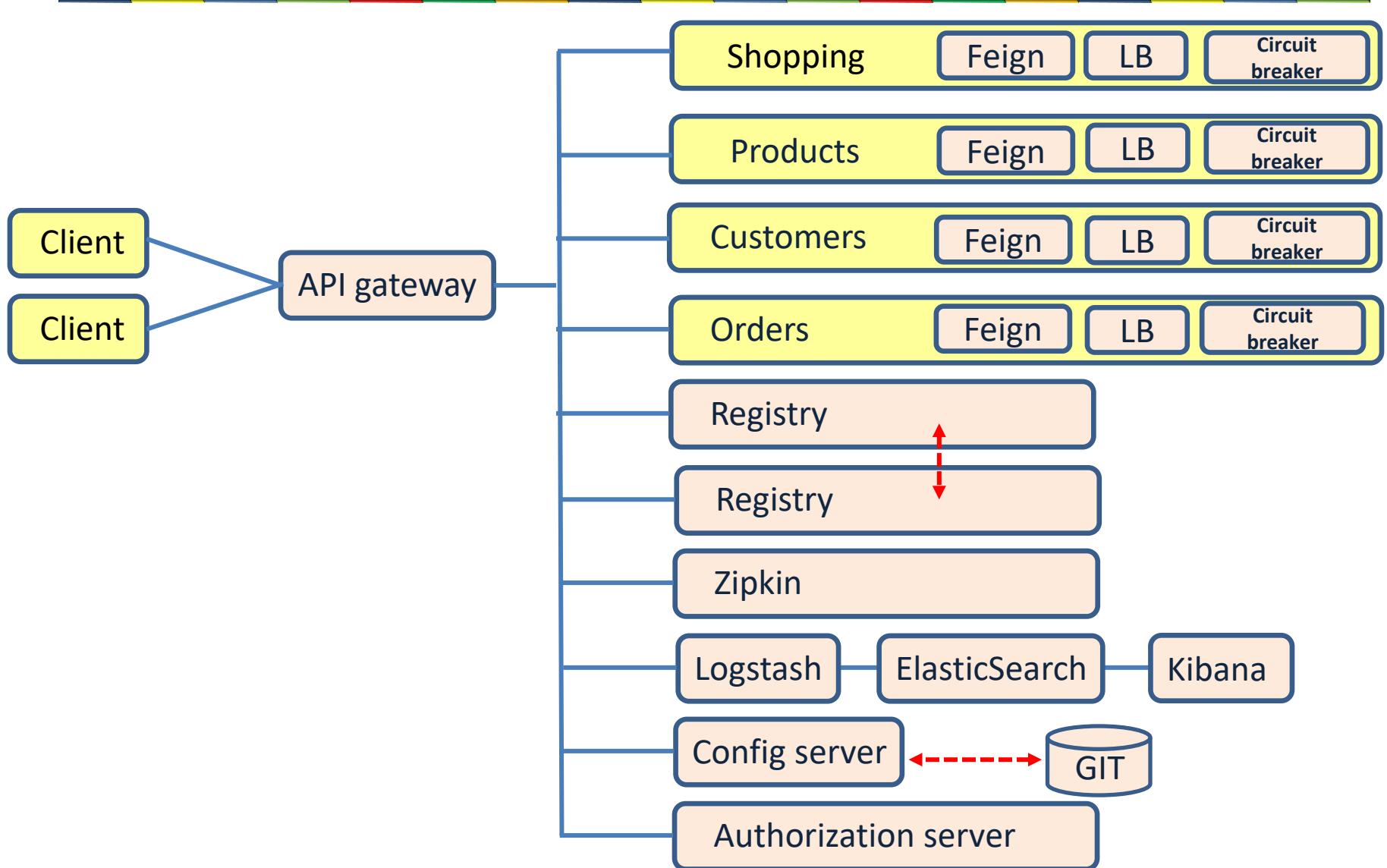
API Gateway security

```
@Configuration
public class SecurityConfig {
    @Bean
    public SecurityWebFilterChain springSecurityFilterChain(ServerHttpSecurity http) {
        http
            .csrf(ServerHttpSecurity.CsrfSpec::disable)

            .authorizeExchange(exchanges -> exchanges
                .pathMatchers("/name").permitAll()      // <- exclude /name from authentication
                .anyExchange().authenticated()         // all other routes require valid JWT
            )
            .oauth2ResourceServer(oauth2 -> oauth2
                .jwt(jwt -> {})
            );

        return http.build();
    }
}
```

Implementing microservices



Challenges of a microservice architecture

Challenge	Solution
Complex communication	Feign Registry API gateway
Performance	CQRS
Resilience	Registry replicas Load balancing between multiple service instances Circuit breaker
Security	Token based security (OAuth2) Digitally signed (JWT) tokens
Transactions	Compensating transactions Eventual consistency
Keep data in sync	Publish-subscribe data change event
Keep interfaces in sync	Spring cloud contract
Keep configuration in sync	Config server
Monitor health of microservices	ELK + beats
Follow/monitor business processes	Zipkin ELK