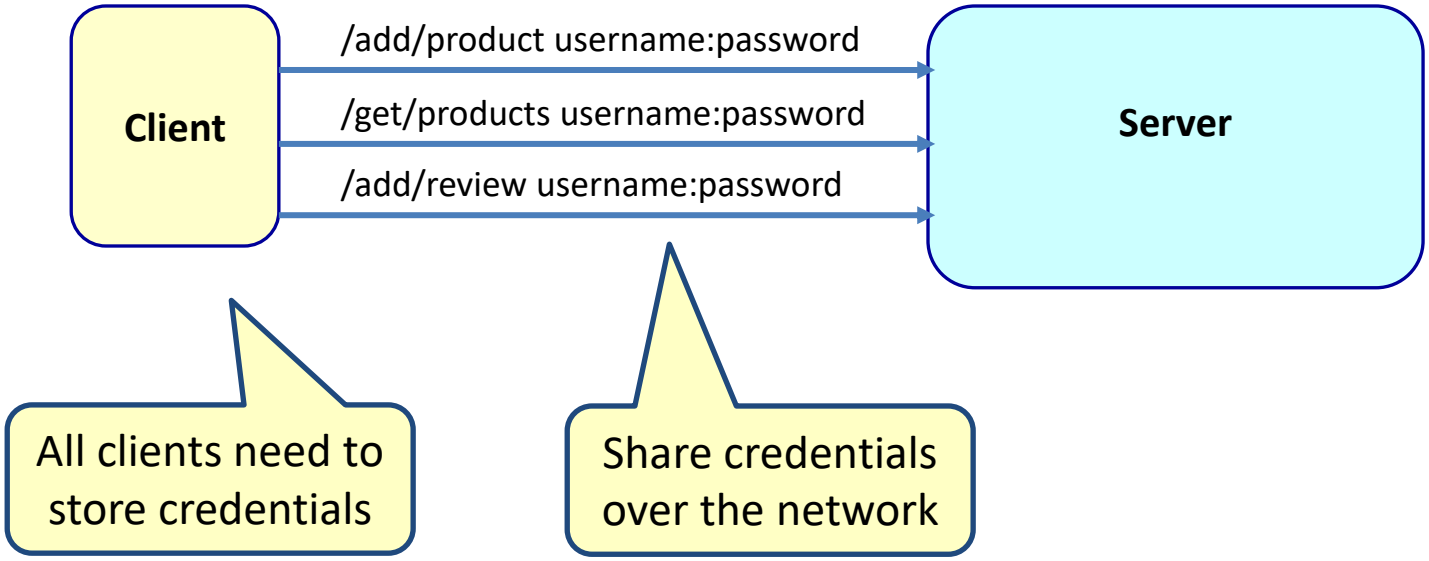
Lesson 11

MICROSERVICES SECURITY

© 2025 MIU

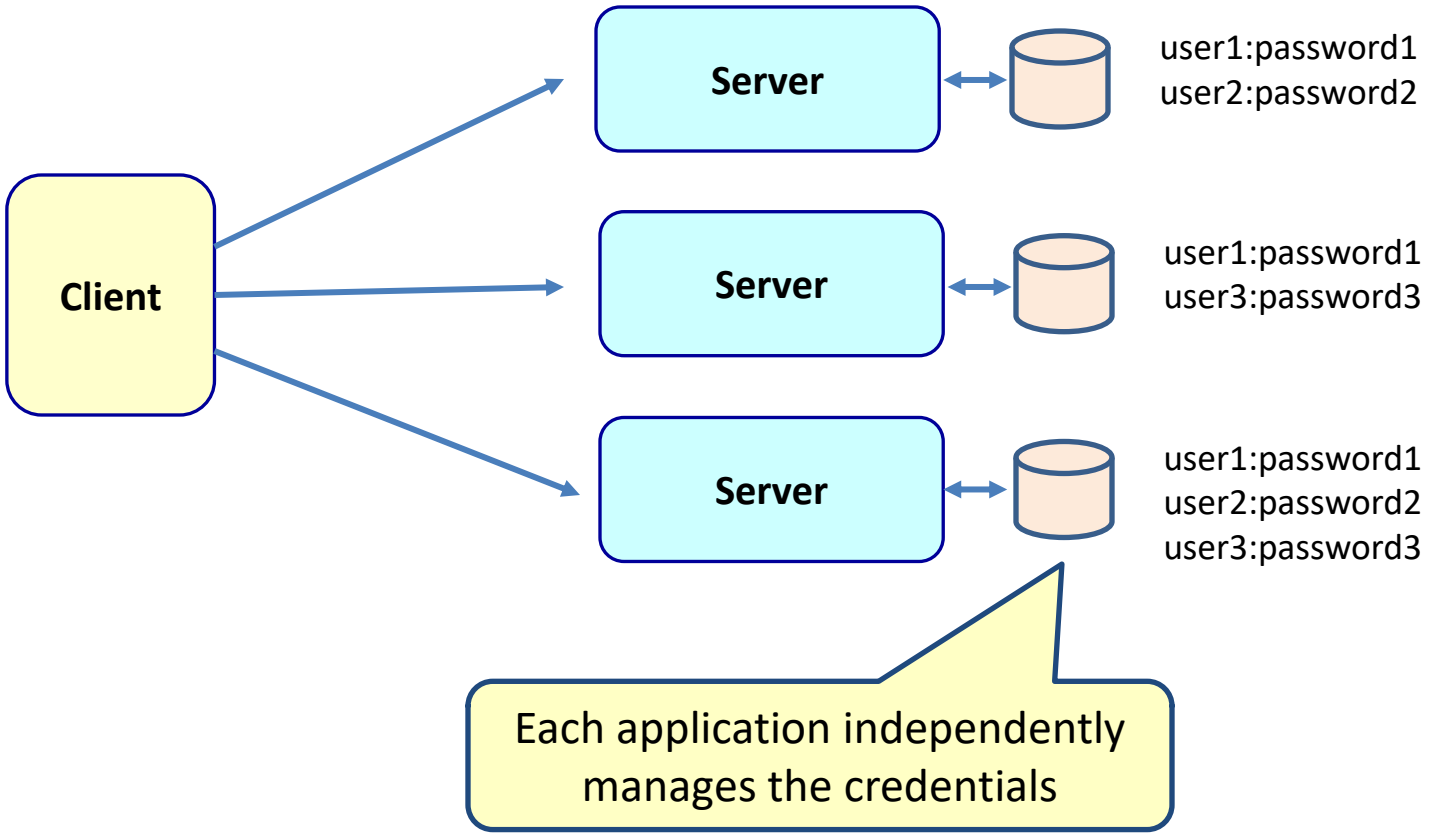
Problems with HTTP basic authentication

- We have to send credentials for every request



Problems with HTTP basic authentication

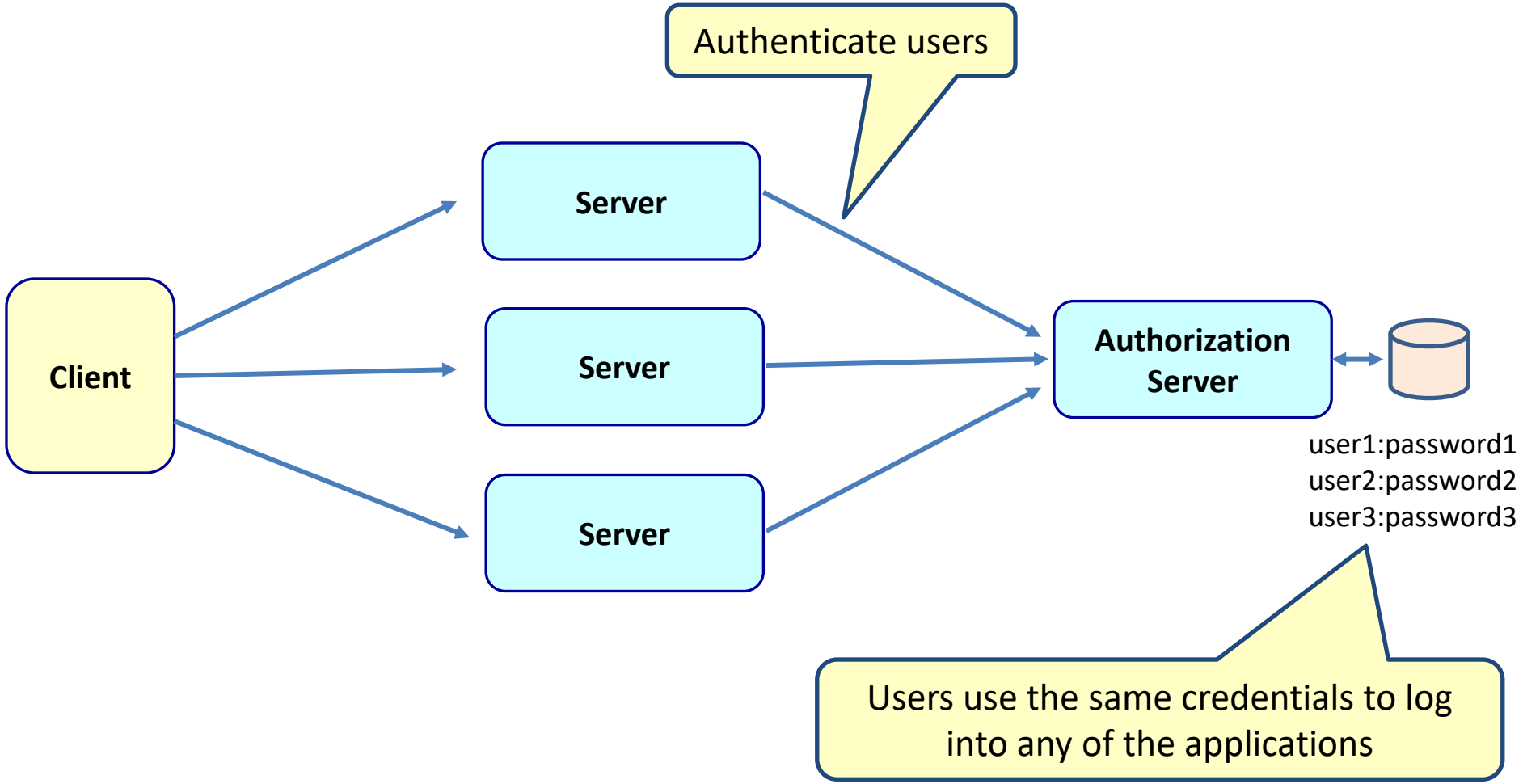
- Every system needs to manage these credentials



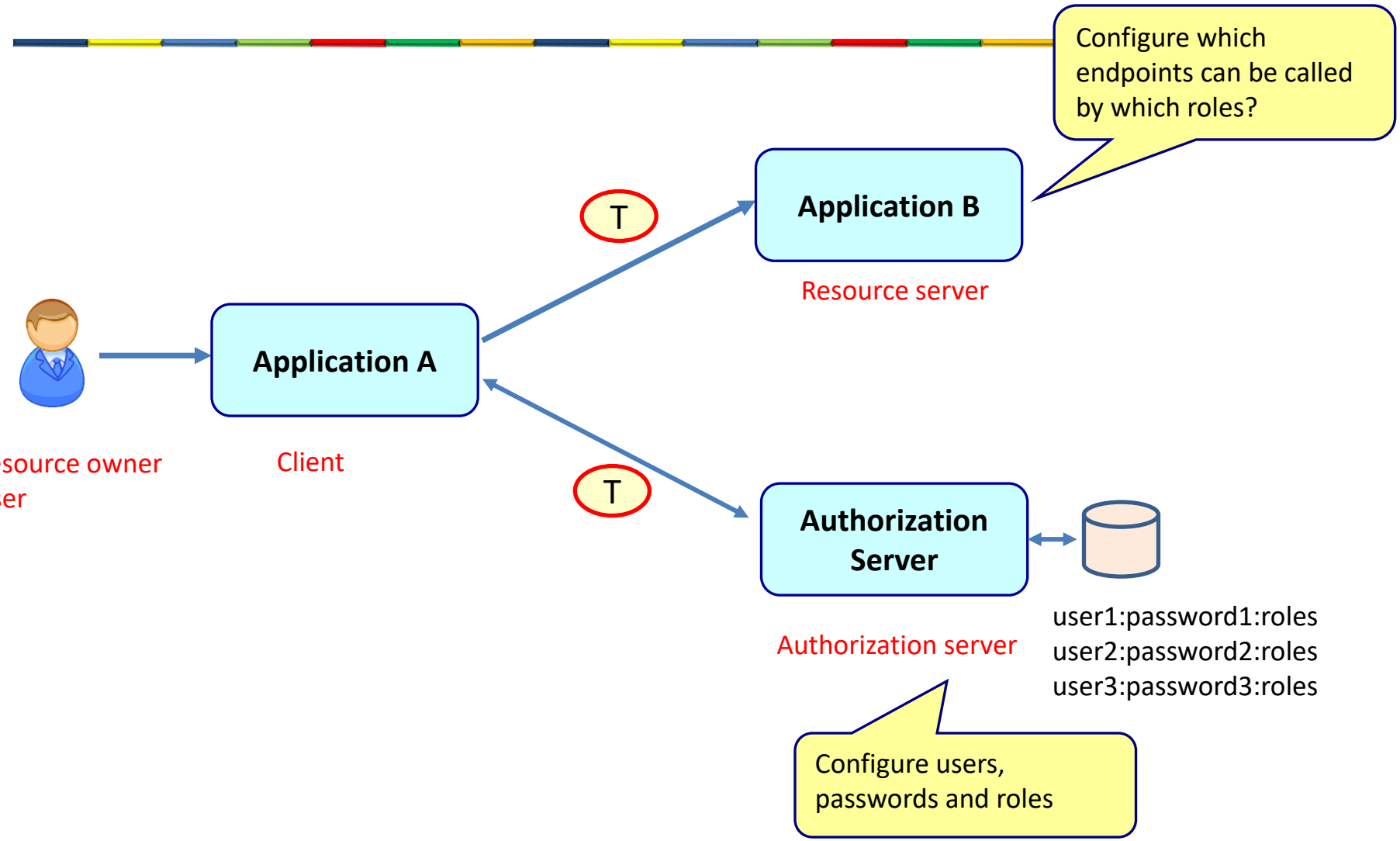
© 2025 MIU

Better solution

- Authorization server



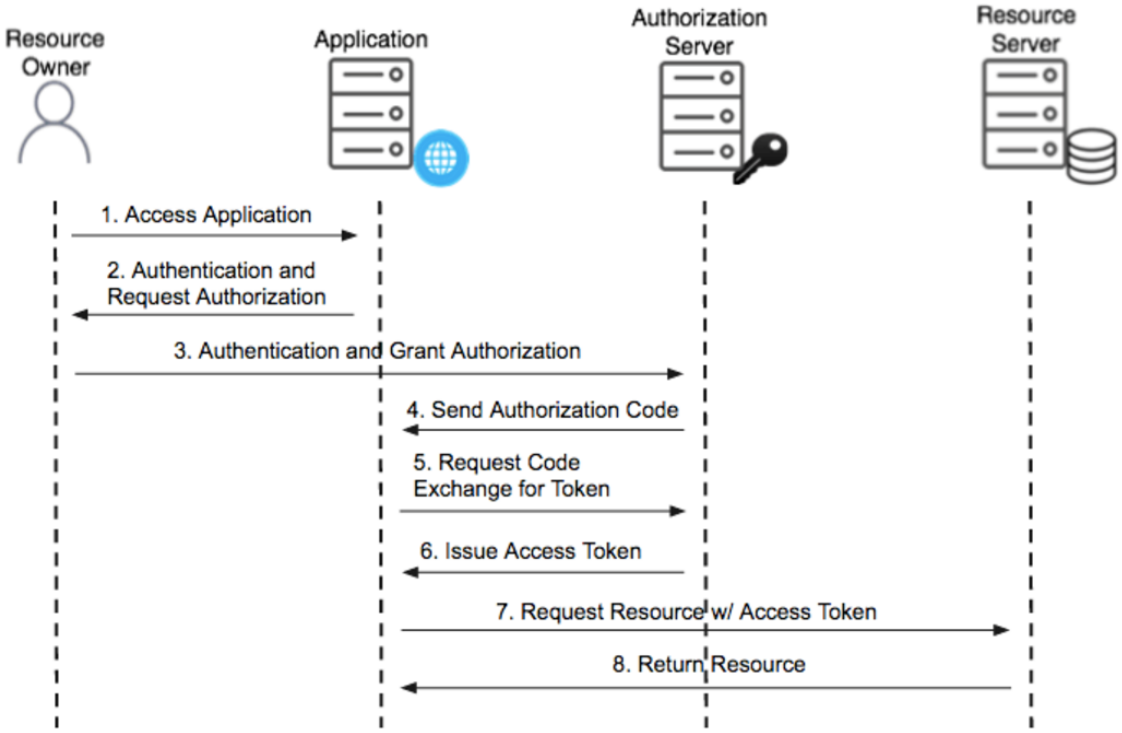
Parts of OAuth 2



OAuth 2 grants

- Authorization code
- Password
- Client credentials
- Refresh token

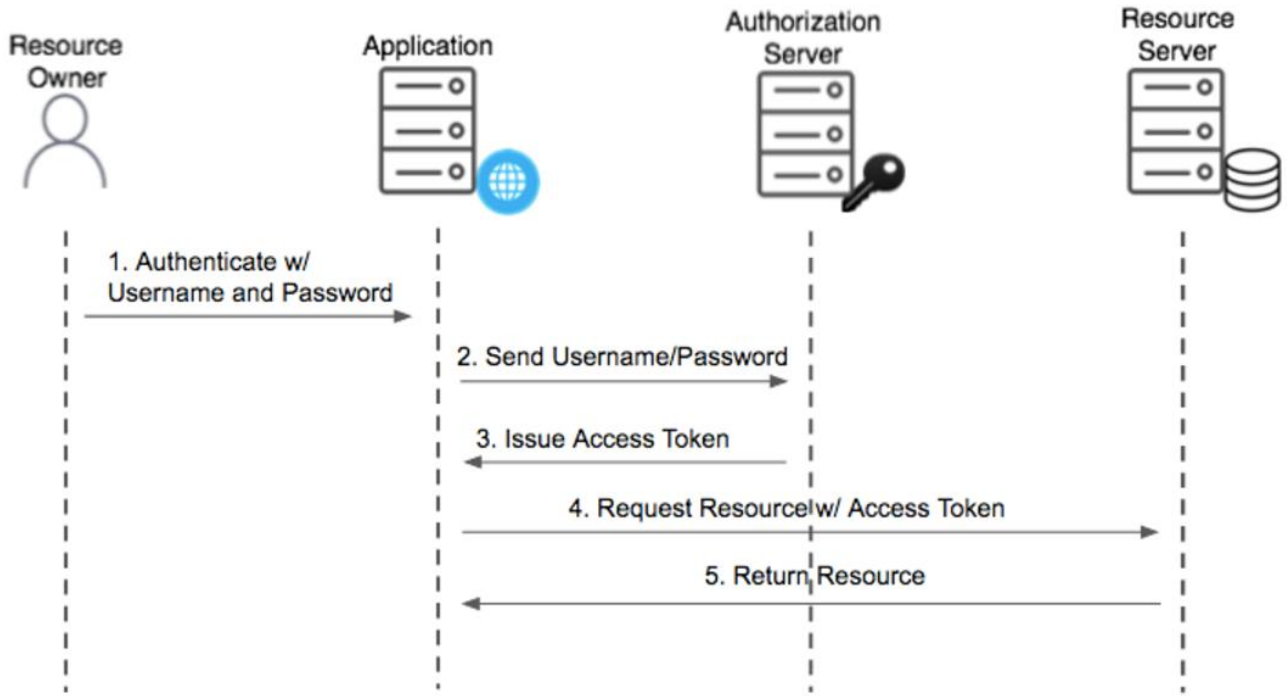
Authorization code grant



- 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

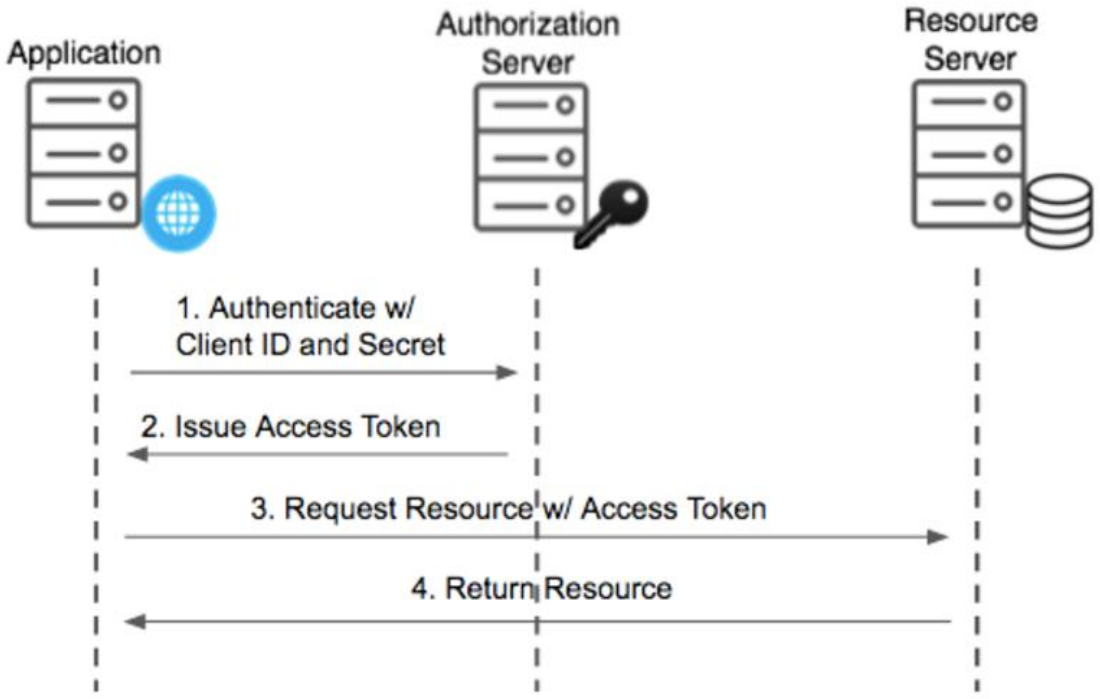
© 2025 MIU

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.

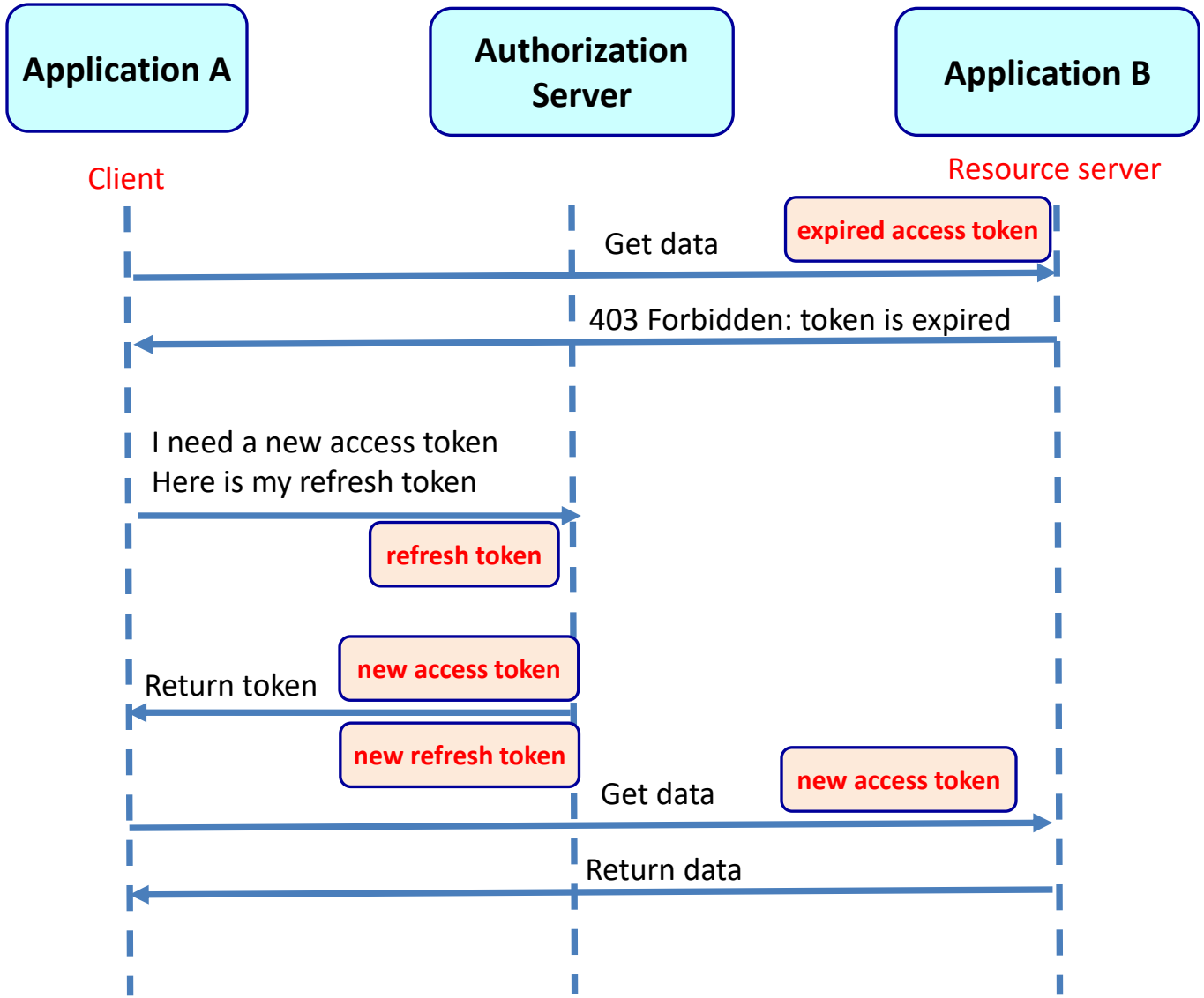
© 2025 MIU

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

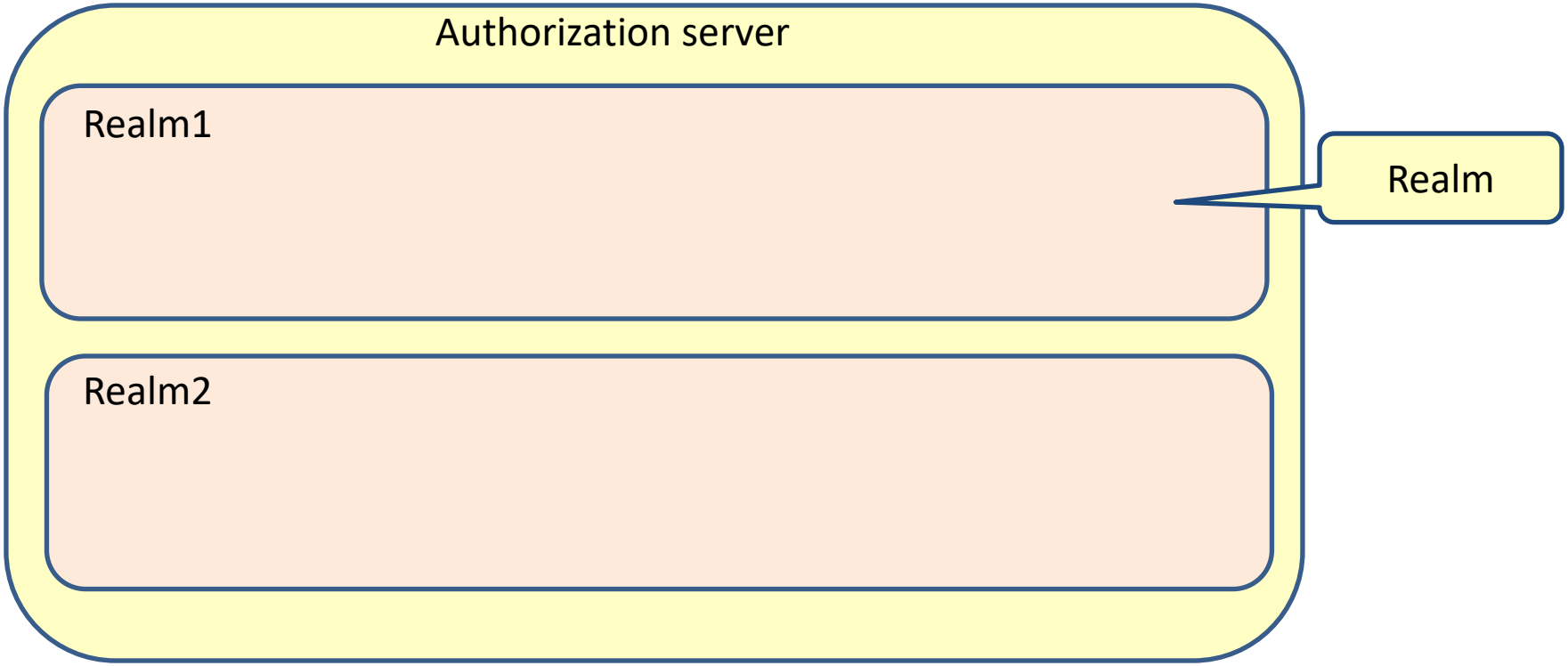
© 2025 MIU

Refresh token grant



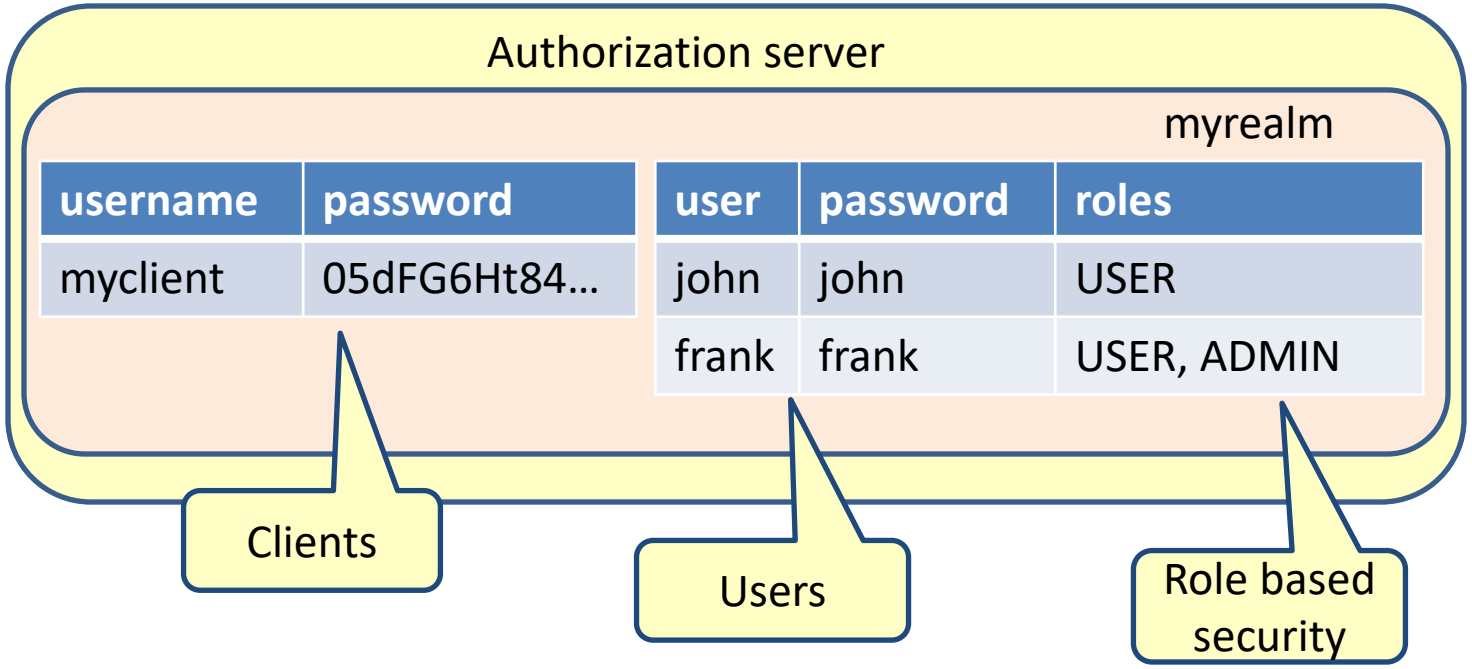
OAUTH2 AUTHORIZATION SERVICE

OAuth2 authorization server



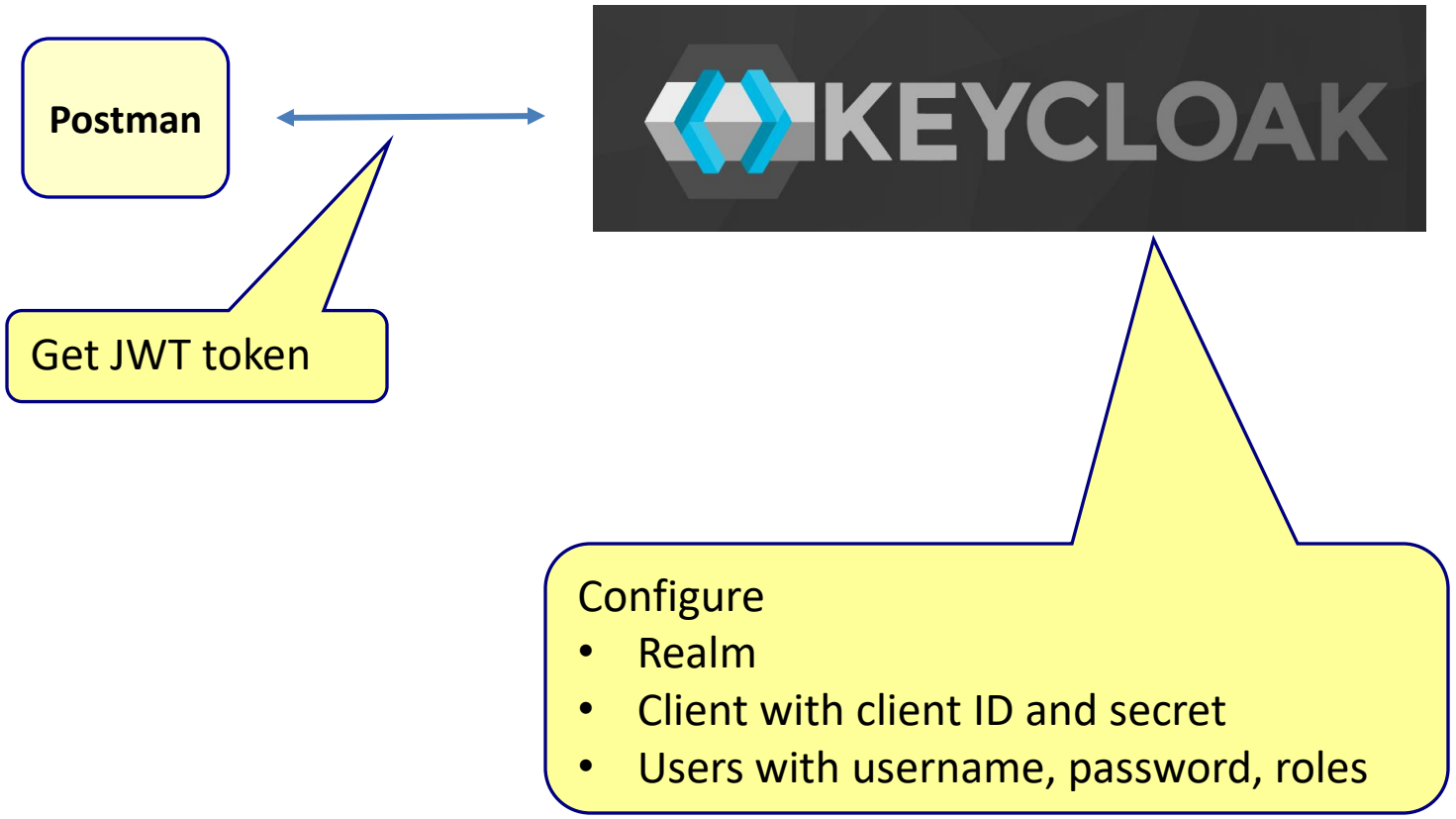
- A **realm** is a **logical isolation boundary** — basically, a **container** that holds **users, clients, roles, and configurations**.

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)

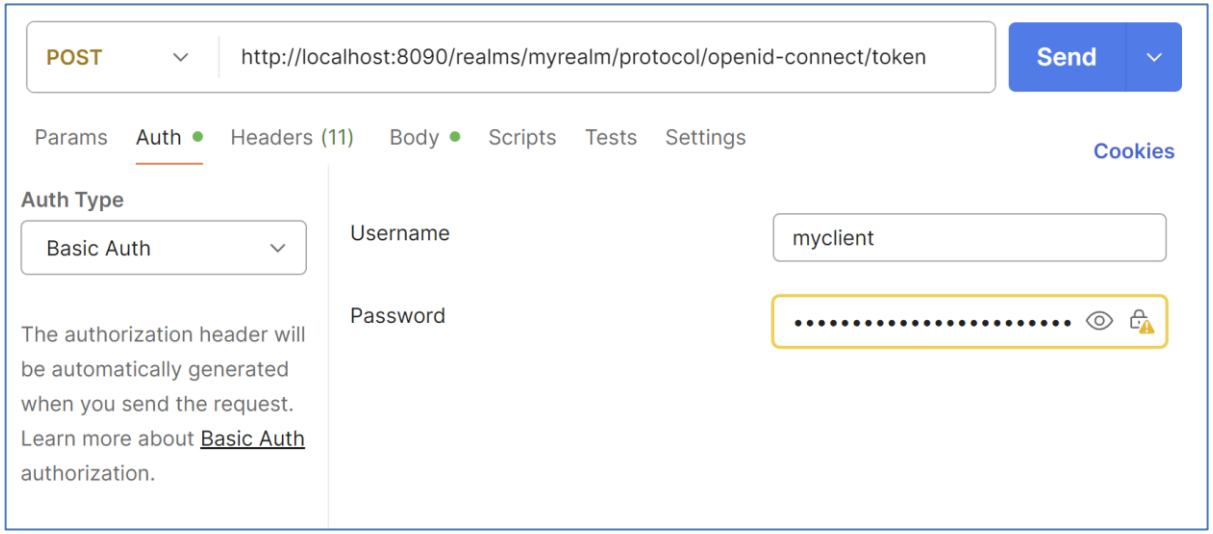
Keycloak authorization server

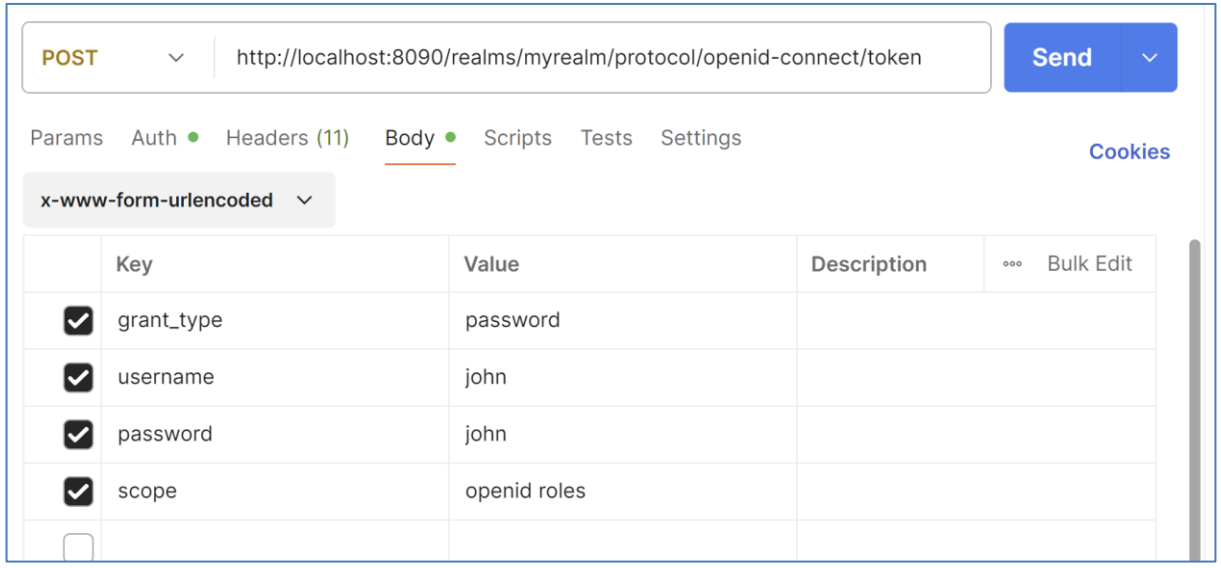


KeyCloak URL’s

© 2025 MIU

Retrieve a token





© 2025 MIU

Returned payload

Access_token

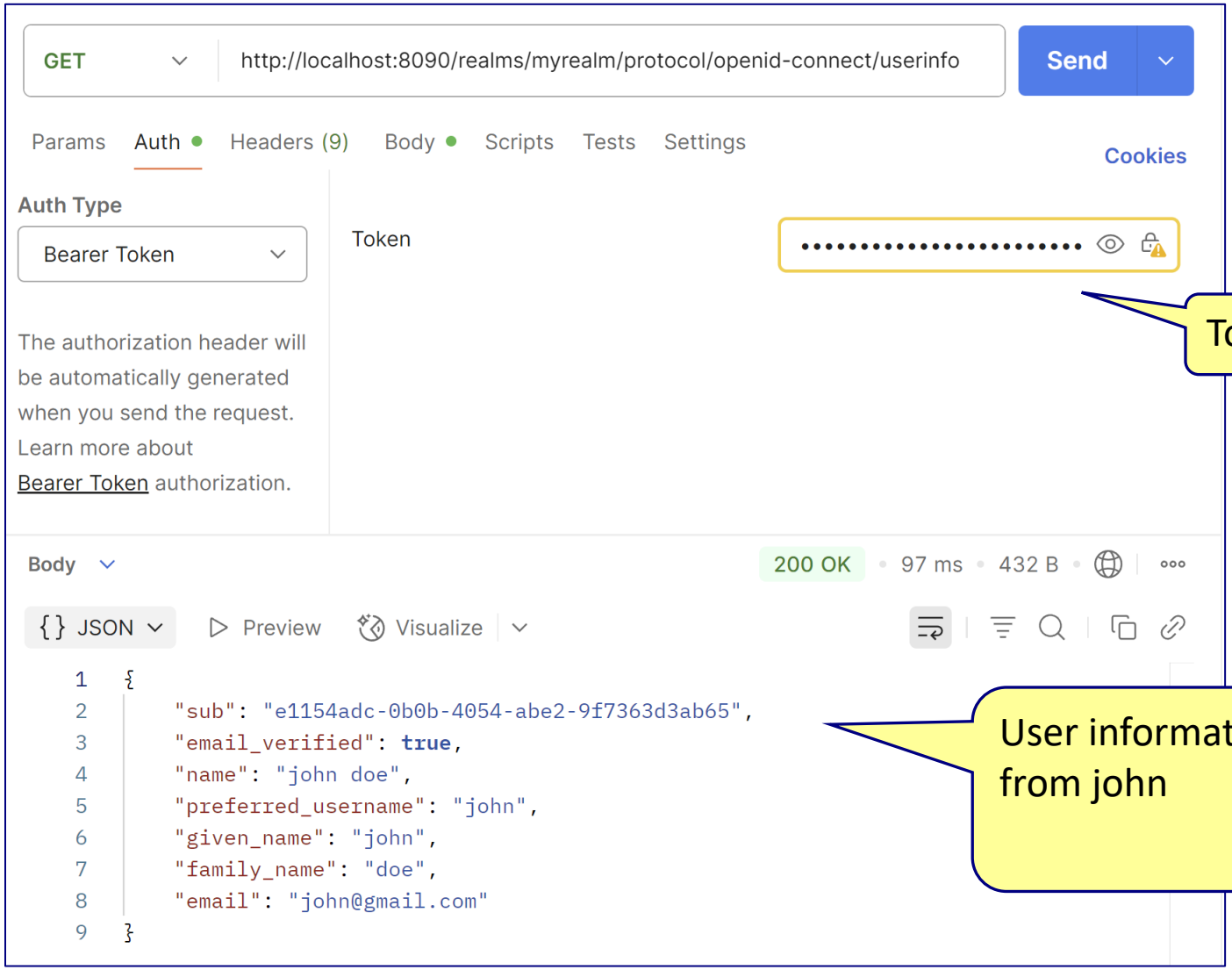
Number of seconds before the token expires

Refresh token


```
{
  "access_token":
      "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJyeHBmMDJqMWFOUlgxS0l2S
      HdLTnZXVFoxcmdzOHdDYlh5RENka1lDamtFIn0.
      eyJleHAiOjE3NjI1Njc3NjYsImlhdCI6MTc2MjU2NDc2NiwianRpIjoib25ydHJvOjE4OWE5MT
      k2LThkYWEtMzg3Zi0yZmQ2LTAyYzE2ZWNiMTI4MiIsImlzcyI6Imh0dHA6Ly9sb2NhbGhvc3Q6
      ODA5MC9yZWFsbXMvbXlyZWFsbSIsImF1ZCI6ImFjY291bnQiLCJzdWIiOiJlMTE1NGFkYy0wYj
      BiLTQwNTQtYWJlMi05ZjczNjNkM2FiNjUiLCJ0eXAiOiJCZWFyZXIiLCJhenAiOiJteWNsaWVu
      dCIsInNpZCI6Ijg4NmFhOTAyLTZhMTEtM2Y4Ny0wNWExLTQ2MTFiMzU1MjU0ZCIsImFjciI6Ij
      EiLCJhbGxvd2VkLW9yaWdpbnMiOlsiLyoiXSwicmVhbG1fYWNjZXNzIjp7InJvbGVzIjpbImRl
      ZmF1bHQtcm9sZXMtbXlyZWFsbSIsIm9mZmxpbmVfYWNjZXNzIiwidW1hX2F1dGhvcml6YXRpb2
      4iXX0sInJlc291cmNlX2FjY2VzcyI6eyJteWNsaWVudCI6eyJyb2xlcyI6WyJVU0VSIl19LCJh
      Y2NvdW50Ijp7InJvbGVzIjpbIm1hbmFnZS1hY2NvdW50IiwibWFuYWdlLWFjY291bnQtbGlua3
      MiLCJ2aWV3LXByb2ZpbGUiXX19LCJzY29wZSI6Im9wZW5pZCBlbWFpbCBwcm9maWxlIiwiZW1h
      aWxfdmVyaWZpZWQiOnRydWUsIm5hbWUiOiJqb2huIGRvZSIsInByZWZlcnJlZF91c2VybmFtZS
      I6ImpvaG4iLCJnaXZlbl9uYW1lIjoiam9obiIsImZhbWlseV9uYW1lIjoiZG9lIiwiZW1haWwi
      OiJqb2huQGdtYWlsLmNvbSJ9.
      nktlXac0sCzrWLWdgV2PzSFaW-aTBjkFvFvrXrynbiSoGmUyKDlAljoWUtLqd_hLx6kWtTaMzc
      o-MCXZj3U5QyWWtXEsXcA-d4eDUtJE1jEG06yQIN2d3NiZcbLrYyP6B71Nd33oh7UEchBeXWOe
      F-b1HUYs78IxsYwFJEOcxXhD5fu6wtrDWB5DKY9lW3z5gJXujsNhi9VQ6g0evEaYWBD_iKWmZ0
      sefj_IuxiWQj5jNCnFFVj6nrhuF0VoqXAonf5a3PsUQtqy5quk2yg60MUsSeGB_0FUb9leV1sN
      zXNmuWd1SAoFo--9gizhG4anrFAAA2PsYt0zSgRBW8wqFA",
  "expires_in": 3000,
  "refresh_expires_in": 1800,
  "refresh_token":
      "eyJhbGciOiJIUzUxMiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI2YWU4ZjNkOS1hYmNlLTQ0Y
      zUtYmQxYy0yMzdjMzMwY2MxY2IifQ.
      eyJleHAiOjE3NjI1NjY1NjYsImlhdCI6MTc2MjU2NDc2NiwianRpIjoiZGM0ZjBiMTQtOGUwMy
      04NjI1LTU1MmEtNDQ1ZWZlYzcxZDdiIiwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo4MDkwL3Jl
      YWxtcy9teXJlYWxtIiwiYXVkIjoiaHR0cDovL2xvY2FsaG9zdDo4MDkwL3JlYWxtcy9teXJlYW
      xtIiwic3ViIjoiZTExNTRhZGMtMGIwYi00MDU0LWFiZTItOWY3MzYzZDNhYjY1IiwidHlwIjoi
      UmVmcmVzaCIsImF6cCI6Im15Y2xpZW50Iiwic2lkIjoiODg2YWE5MDItNmExMS0zZjg3LTA1YT
```


© 2025 MIU

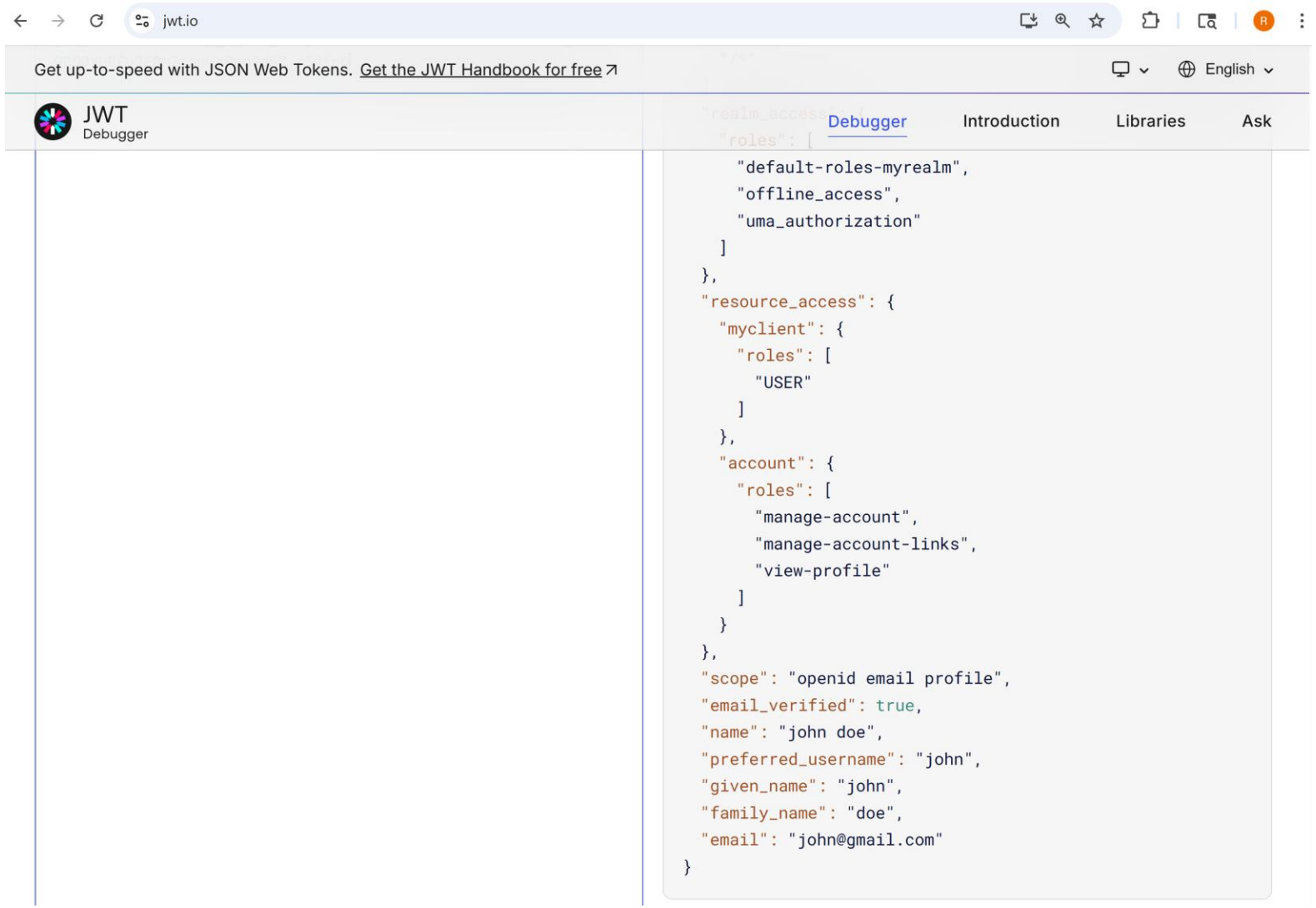
Get user information



Token for john

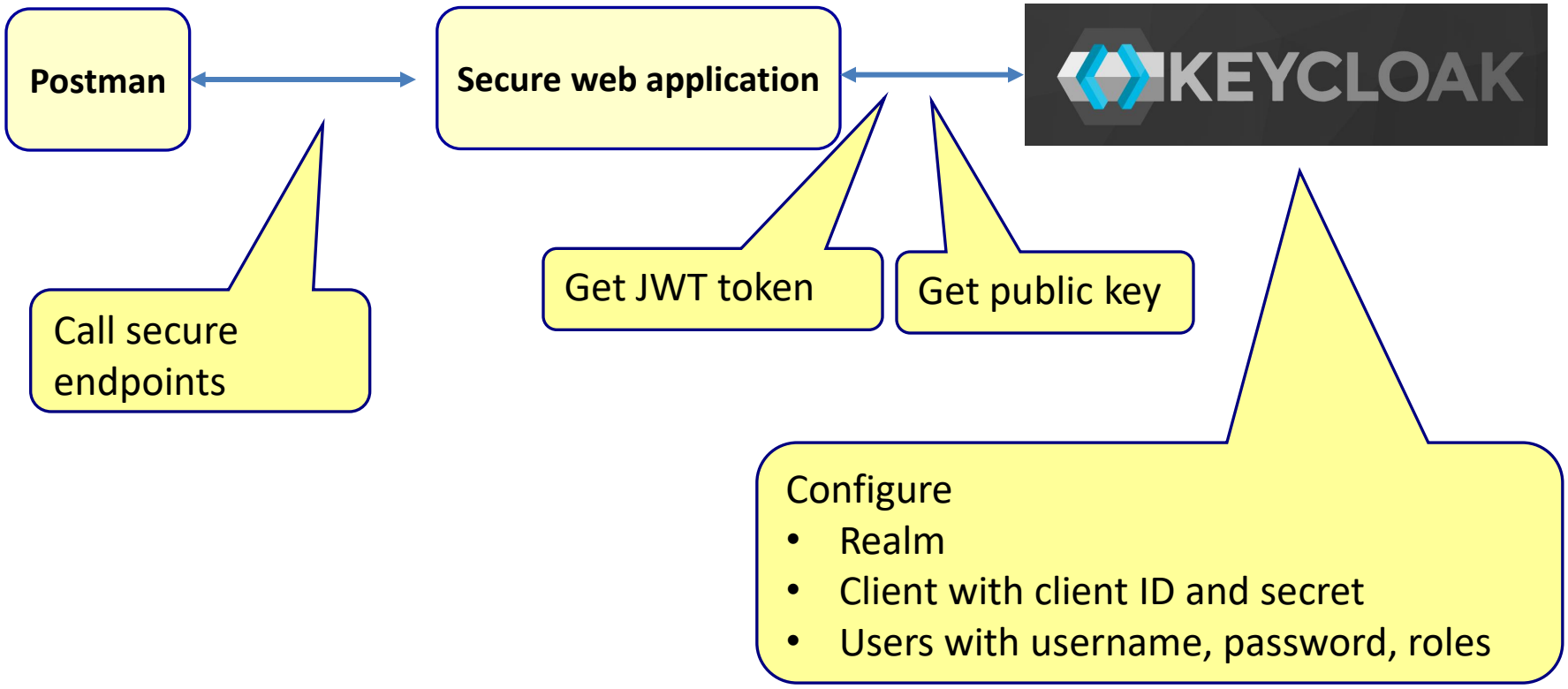
User information from john

See token content



A SECURE APPLICATION

Secure application



© 2025 MIU

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

© 2025 MIU

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

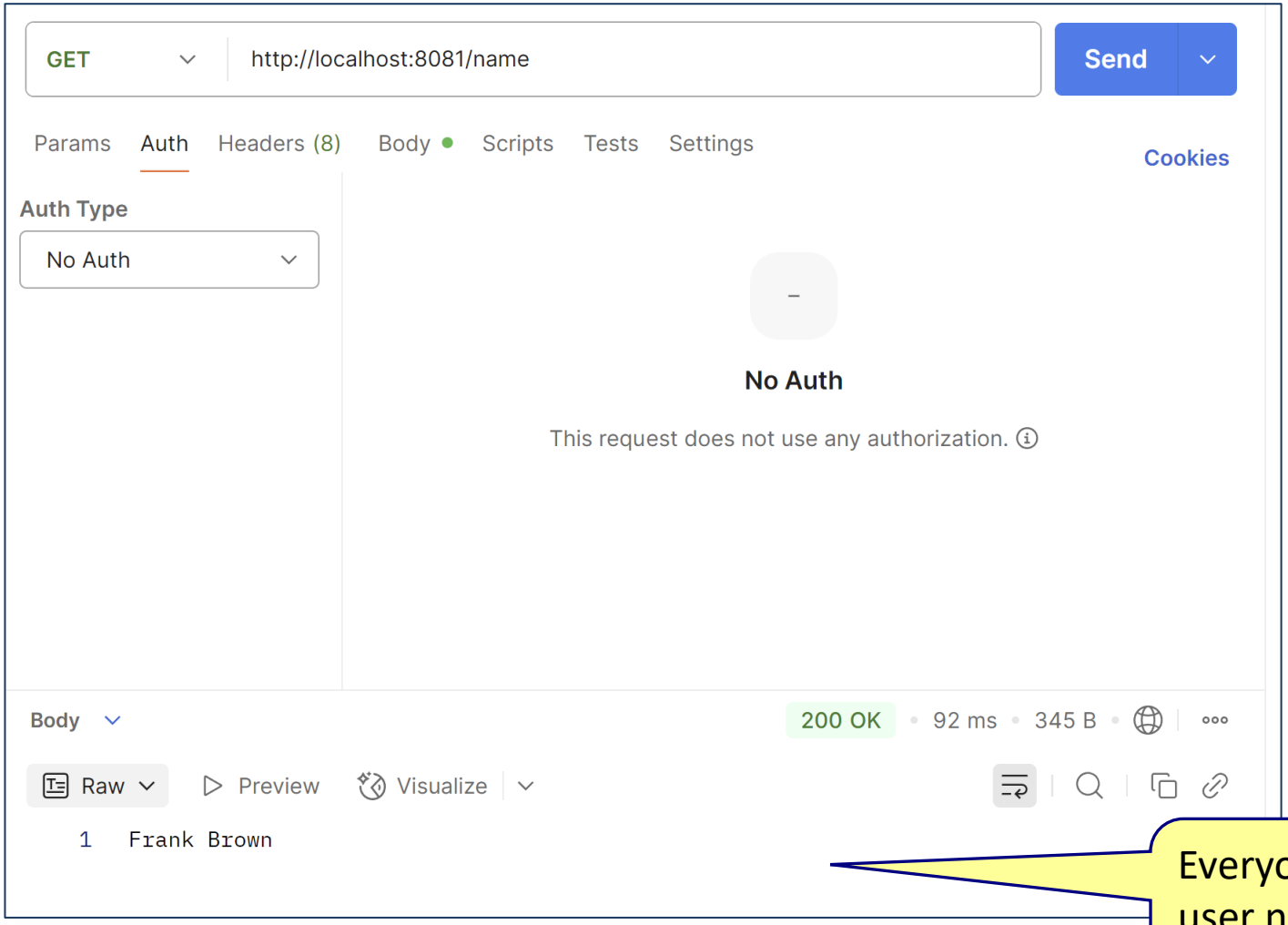
```
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: http://localhost:8090/realms/myrealm
          jwk-set-uri: http://localhost:8090/realms/myrealm/protocol/openid-connect/certs

server:
  port: 8081
```

Tells spring where the token must come from

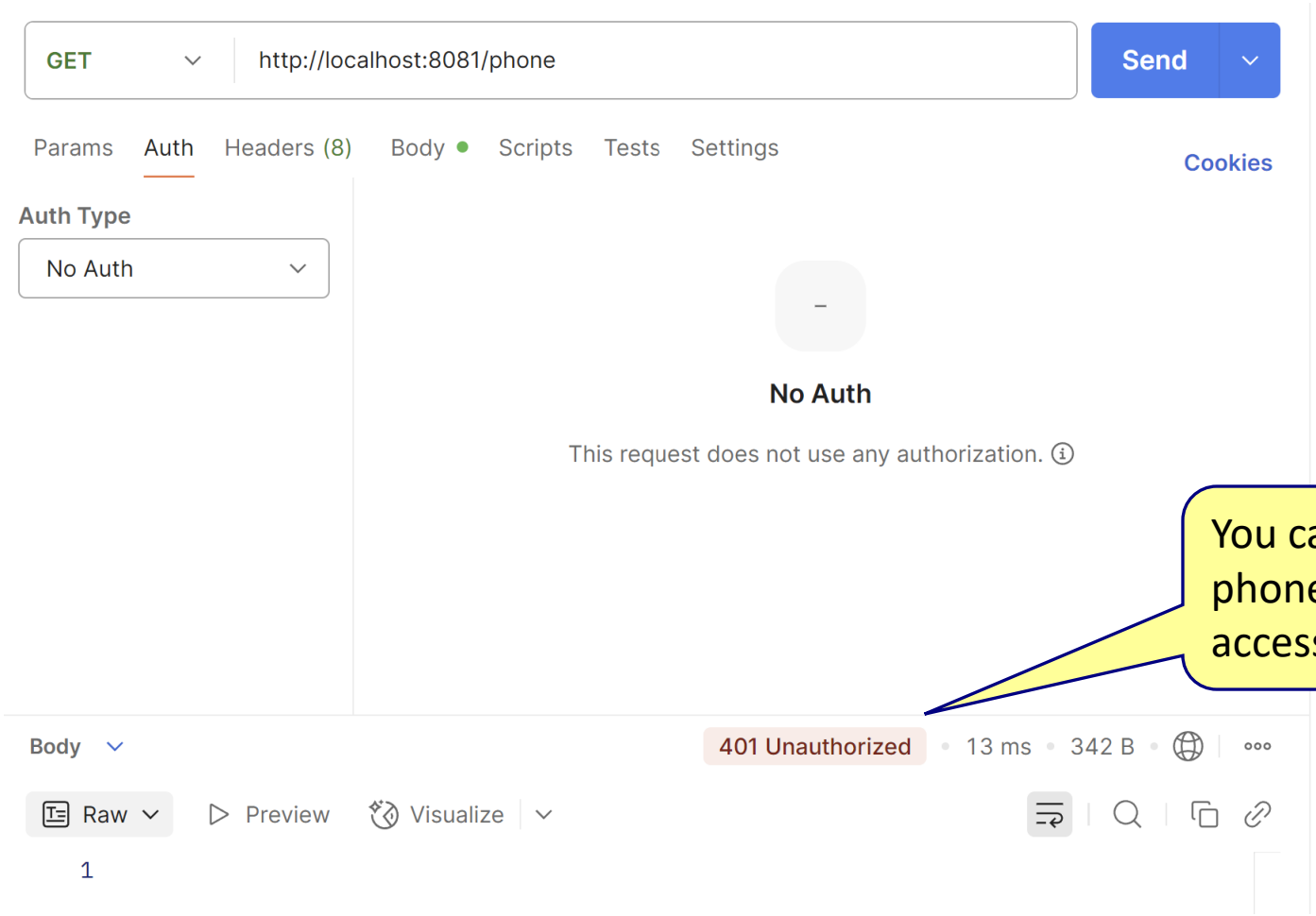
Used to get the public key of the authorization server

Get the user name

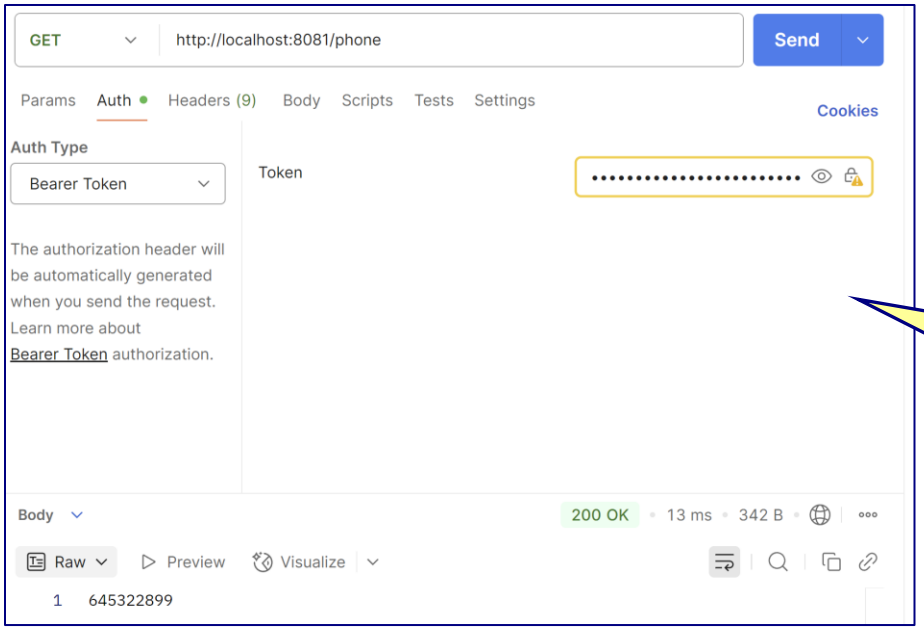


Everyone can get the user name without authorization

Get the phone number

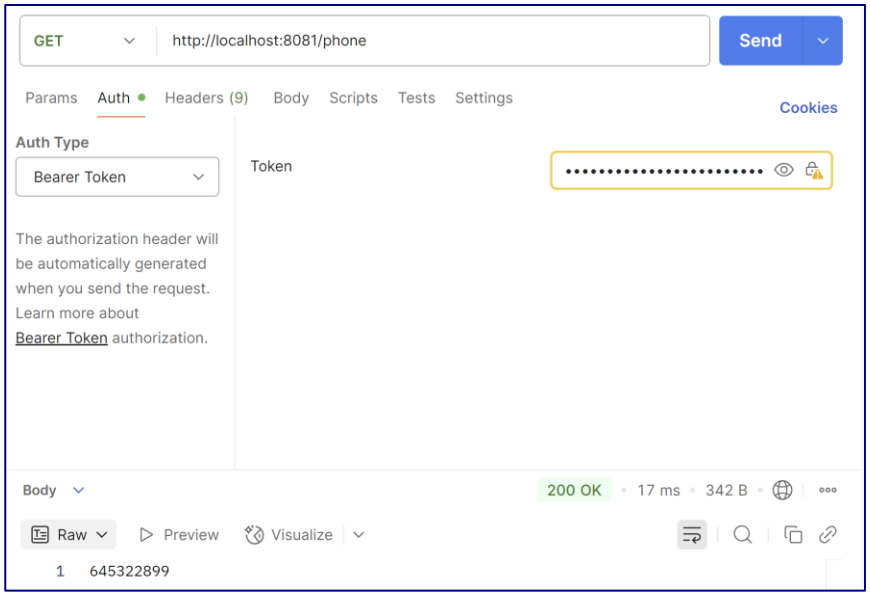


Get the phone number



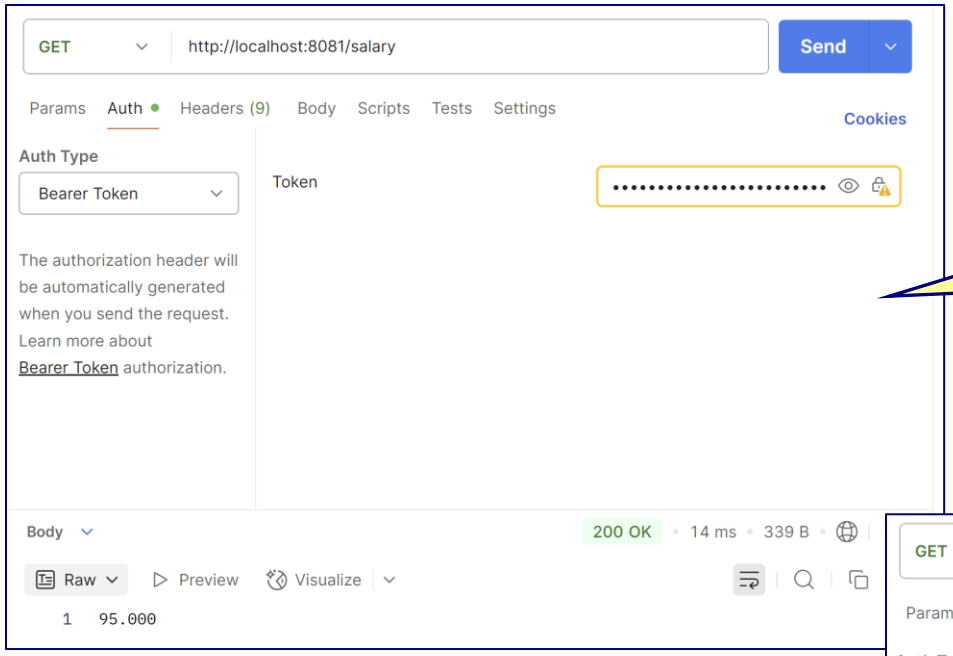
John can get the phone number

Frank can get the phone number



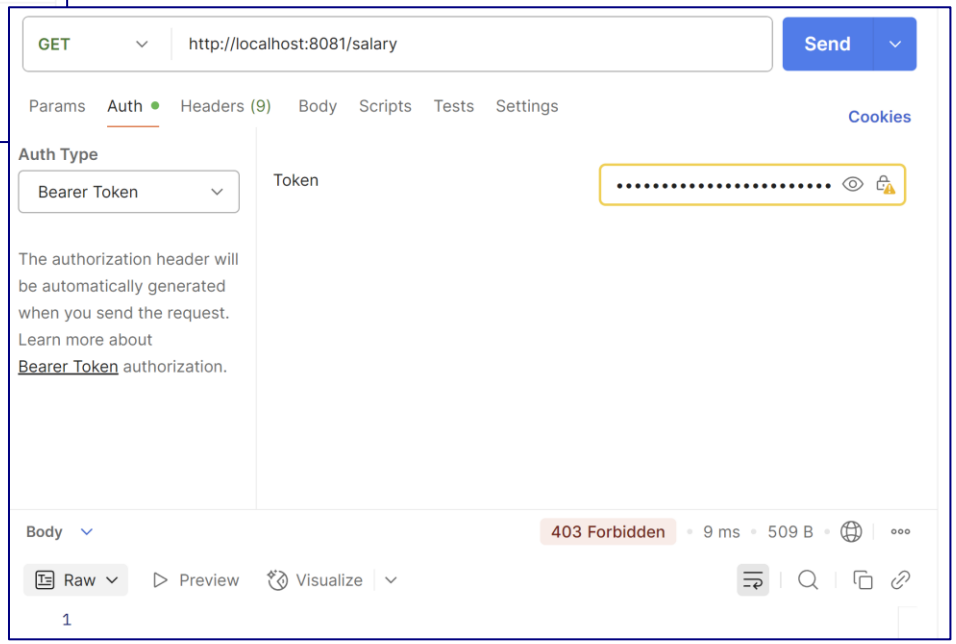
© 2025 MIU

Get the salary



Frank can get the salary (role=ADMIN)

John cannot get the salary (role = USER)

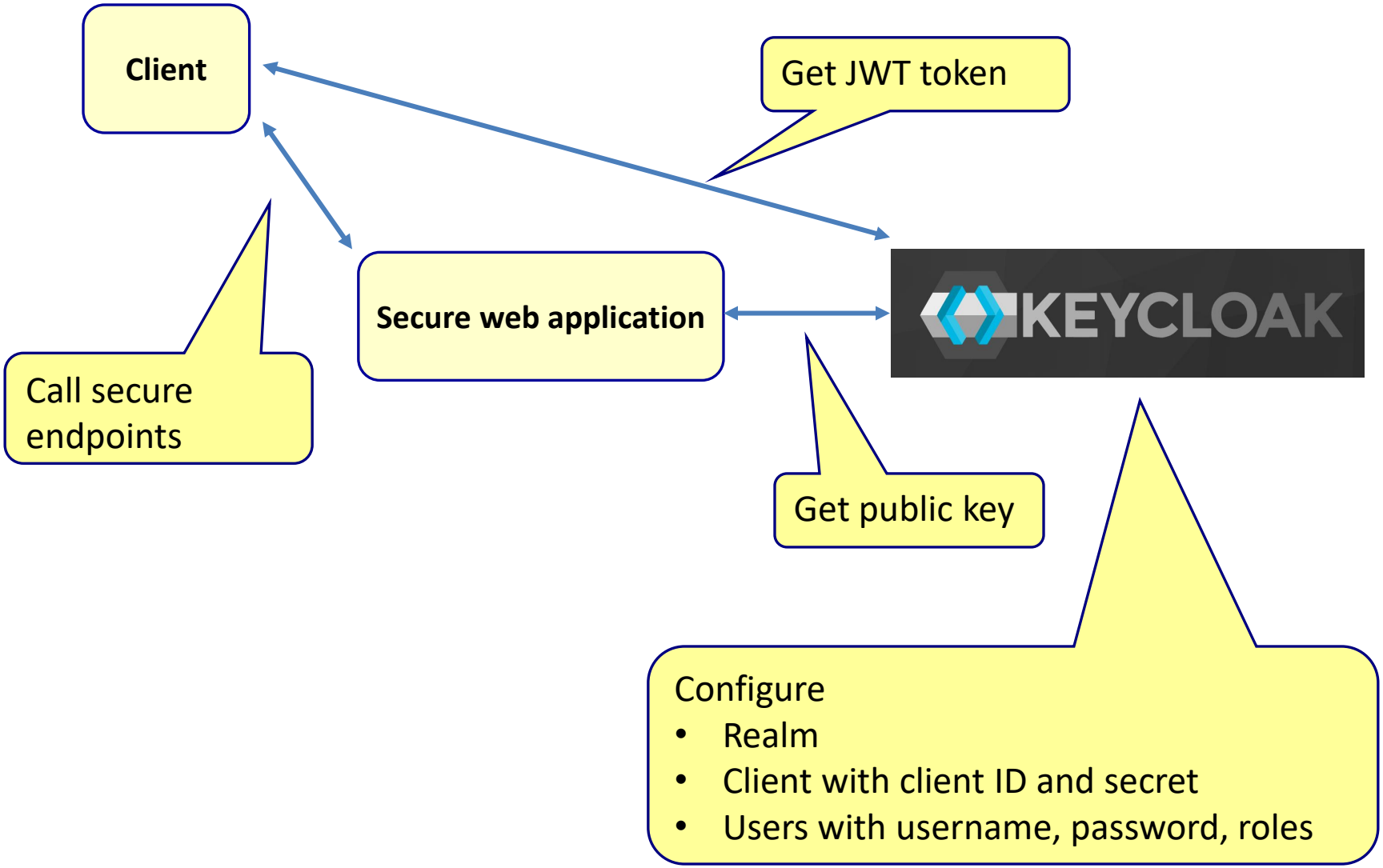


Main point

- 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
```

© 2025 MIU

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

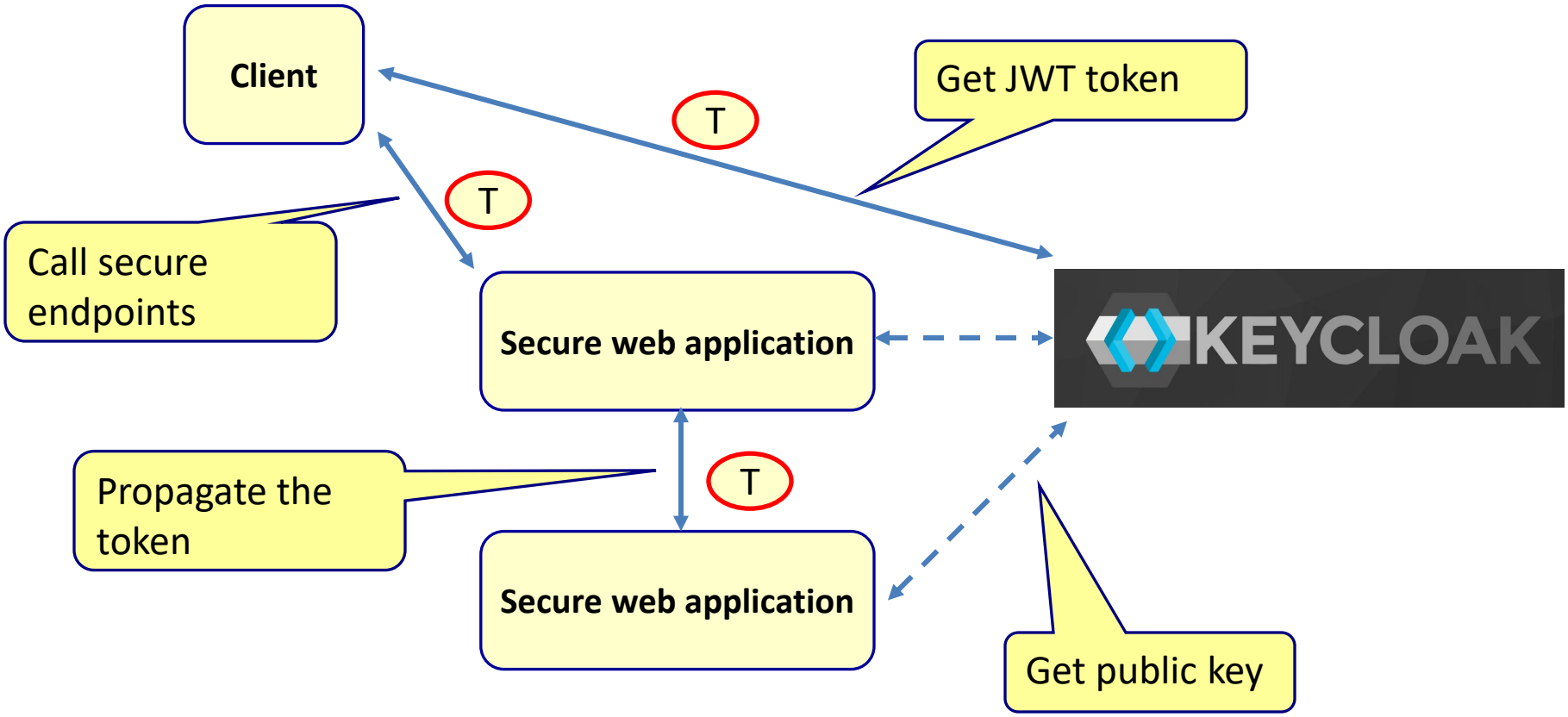
```
public String getAccessToken(String username, String password){
  WebClient webClient = WebClient.builder()
      .baseUrl("http://localhost:8090/realms/myrealm/protocol/openid-connect/token")
      .build();
  // Get access token
  Map<String, Object> tokenResponse = webClient.post()
      .header("Content-Type", "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

© 2025 MIU

Client for the secure application



Downstream secure application

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

application.yml

```yaml
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: http://localhost:8090/realms/myrealm
          jwk-set-uri: http://localhost:8090/realms/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
```

© 2025 MIU

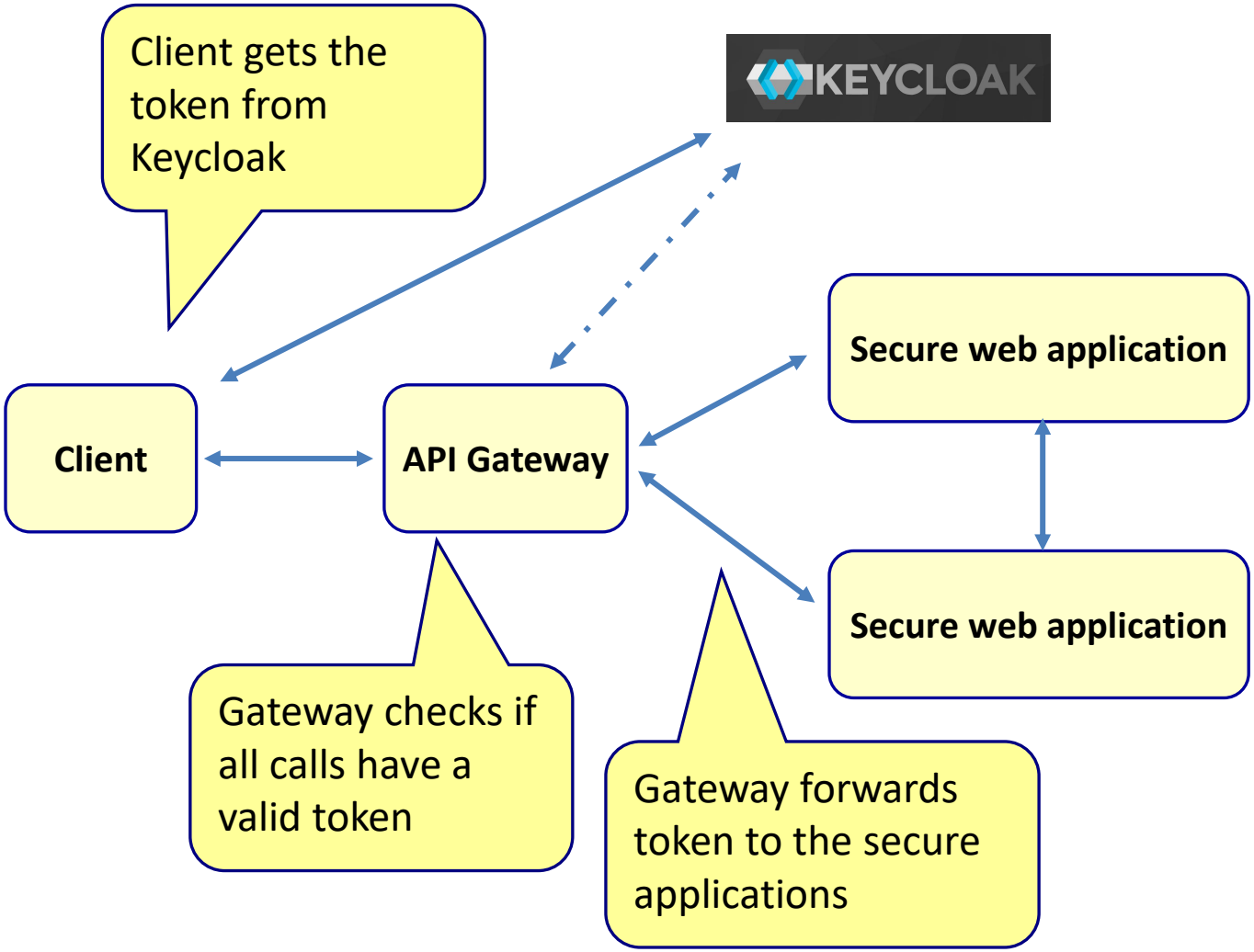
Main point

- 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.*

© 2025 MIU

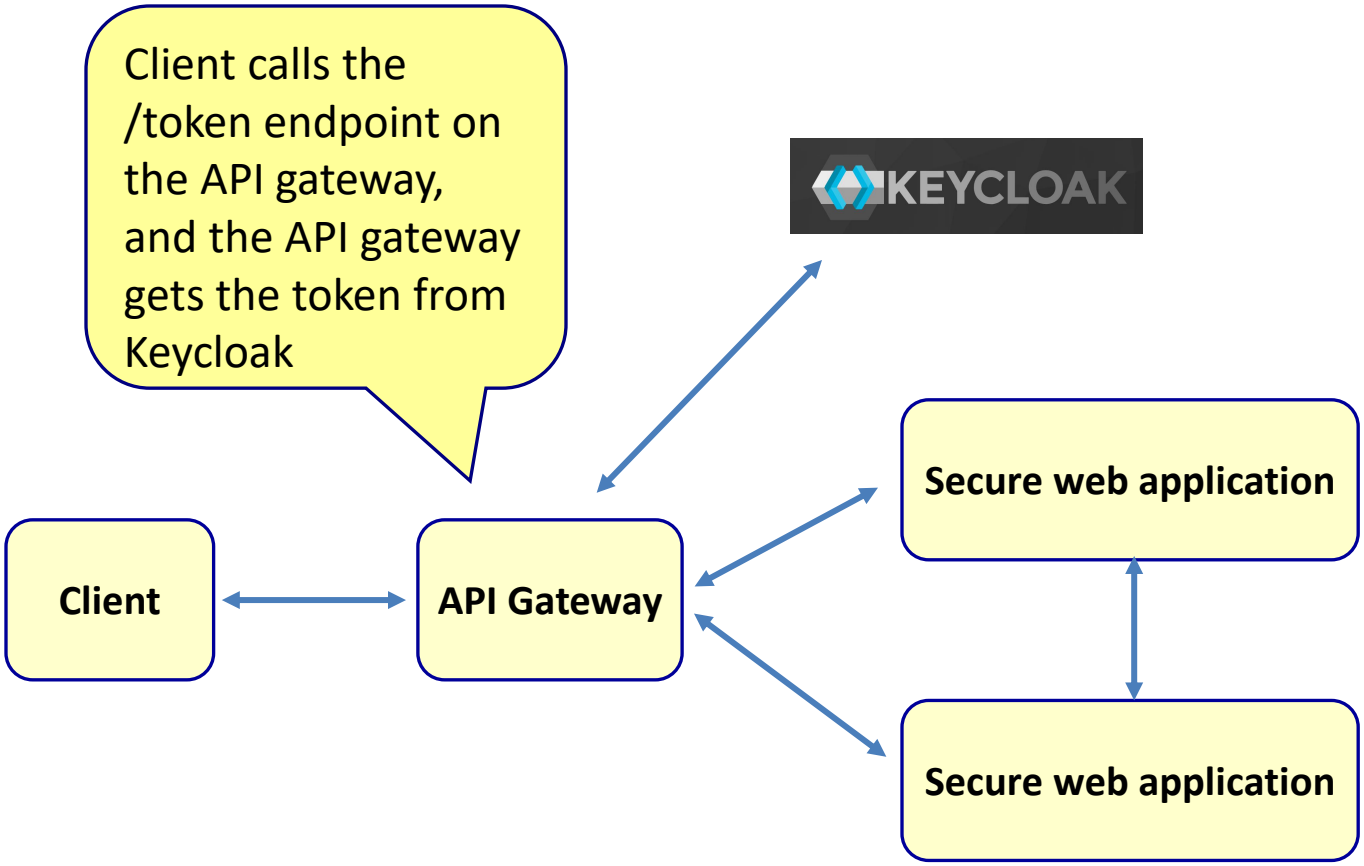
LETS ADD THE API GATEWAY

API Gateway option 1



© 2025 MIU

API Gateway option 2



API Gateway security

```yaml
spring:
  application:
    name: api-gateway
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: http://localhost:8090/realms/myrealm
          jwk-set-uri: http://localhost:8090/realms/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
```

© 2025 MIU

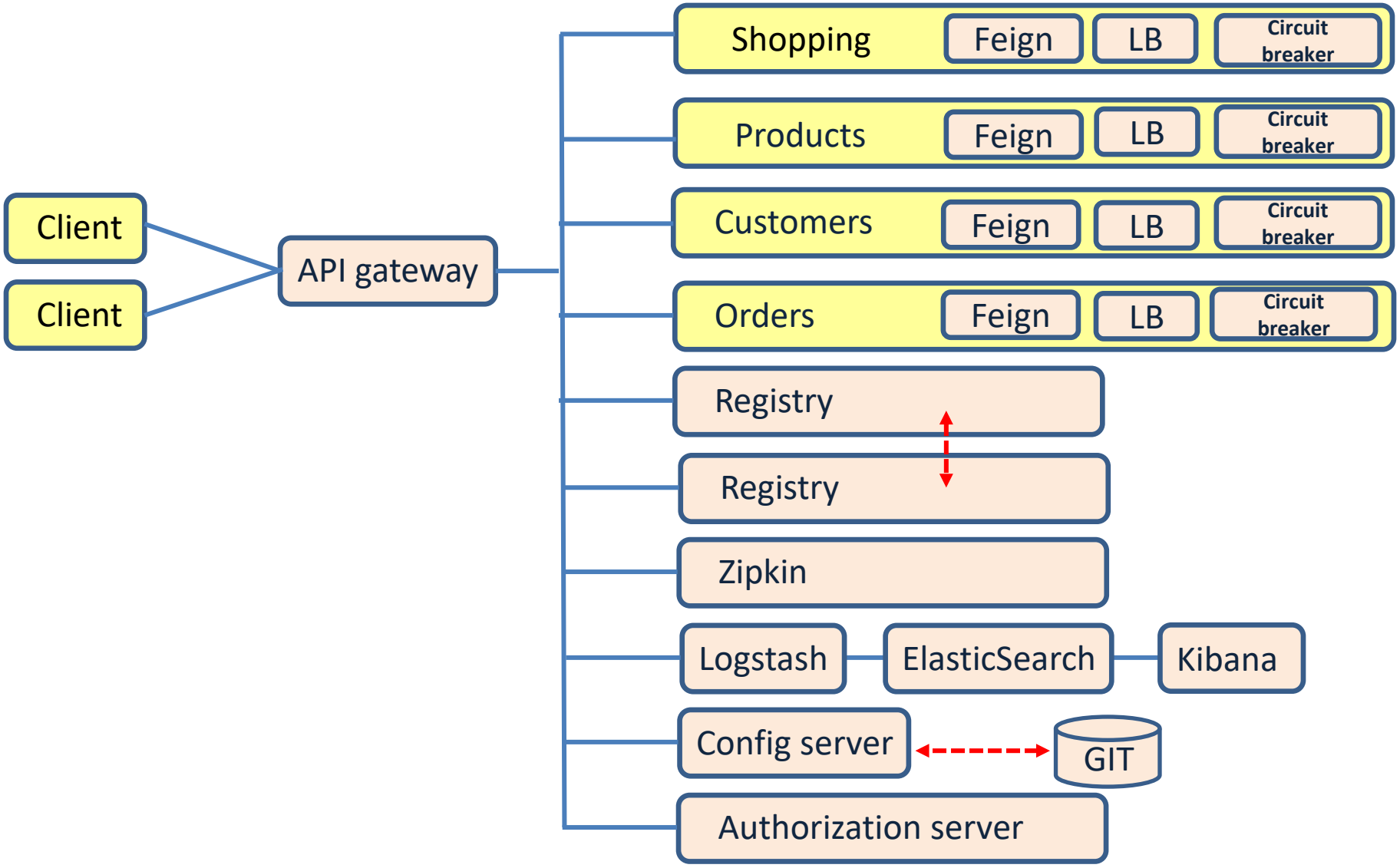
API Gateway security

```
@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



© 2025 MIU

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