oAuth & JWT

Hybrid Integration Enablement

October 2017

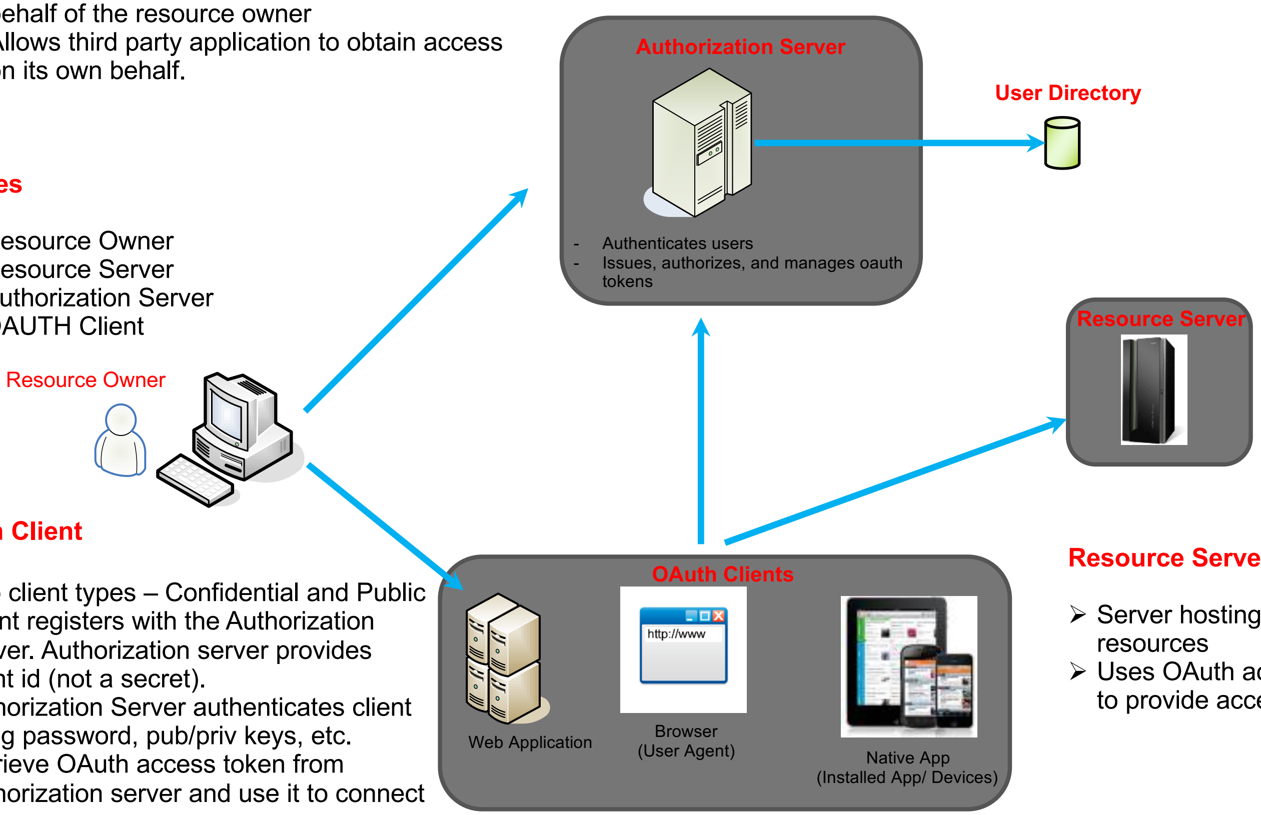
IBM

What is OAuth

- An authorization protocol
- Enables third party to obtain limited access on behalf of the resource owner
- Allows third party application to obtain access on its own behalf.

Roles

- Resource Owner
- Resource Server
- Authorization Server
- OAUTH Client



Resource Owner

Authorization Server

- Authenticates users
- Issues, authorizes, and manages oauth tokens

User Directory

Resource Server

OAuth Clients

Web Application

Browser
(User Agent)

Native App
(Installed App/ Devices)

OAuth Client

- Two client types – Confidential and Public
- Client registers with the Authorization Server. Authorization server provides client id (not a secret).
- Authorization Server authenticates client using password, pub/priv keys, etc.
- Retrieve OAuth access token from Authorization server and use it to connect to resource server.

Resource Server

- Server hosting protected resources
- Uses OAuth access tokens to provide access.

OAuth Terminology

- **Resource** - The information or object of interest.
- **Resource Owner** - The entity that maintains ownership of the resource.
- **Client** - The entity requesting access to the resource. Clients must register with the Authorization Server prior to requesting resources.
- **Resource Server** - Where the resource resides.
- **Authorization Server** - Where the Resource Owners authenticate to allow or deny access to the resources. The Resource Owners never have to share their credentials with the client. Also where Clients authenticate to obtain an Access Token for the resource.
- **Authorization Code** - A short-lived token provided directly to the Client by the Authorization Server. Implies that the client has permission to access the resource on the resource owner's behalf. Used to obtain an Access Token.
- **Access Token** - Used by the client to access protected resources. Contains information that defines the scope and duration of access which is enforced by the Resource Server.

OAuth interactions, Grant and Client Types

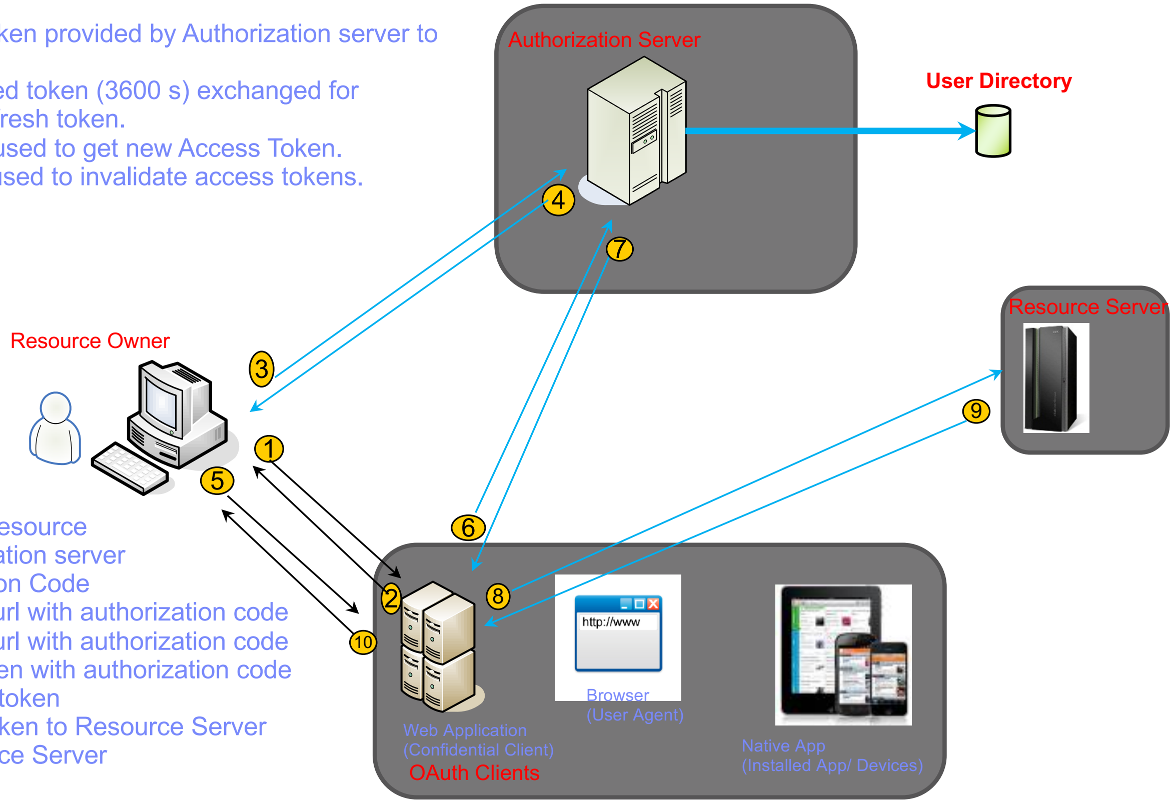
- An OAuth client can be considered:
  - **Public**: an application is incapable of maintaining the secrecy of the client secret. This is usually the case when the application is native on a computer or mobile where the secret would have to be stored on the user's device, likely inside the source code of the application.
  - **Confidential**: an application is capable of maintaining the secrecy of the client secret. This is usually the case when an application runs in a browser and accesses its own server when obtaining OAuth access tokens.
- There are 4 types of OAuth Flow depending on the level of security. Each have a corresponding Grant Type associated:

OAuth Flow	OAuth Grant Type	Clients Types
Access Code	Authorization Code	Both
Implicit	Implicit	Public
Password	Resource Owner Password Credentials	Both
Application	Client Credentials	Confidential

Access Code / Authorization Code

Types of Tokens

- Authorization Code – Token provided by Authorization server to resource owner
- Access Token – shortlived token (3600 s) exchanged for authorization code or refresh token.
- Refresh Token – Token used to get new Access Token.
- Revoke Token – Token used to invalidate access tokens.



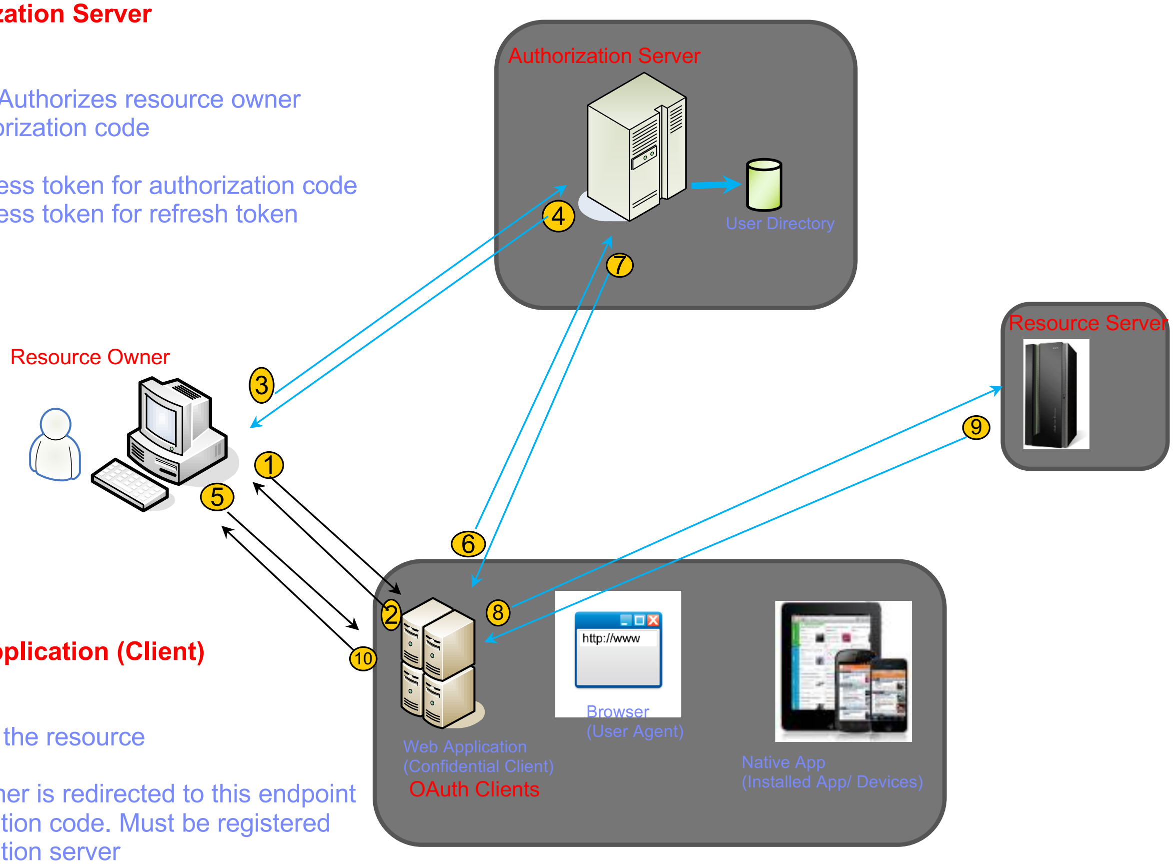
Authorization Code Flow

1. Request for protected resource
2. Redirection to Authorization server
3. Request for Authorization Code
4. Redirection to redirect url with authorization code
5. Request to redirection url with authorization code
6. Request for access token with authorization code
7. Response with Access token
8. Request with access token to Resource Server
9. Response from Resource Server
10. Response to client

Access Code / Authorization Code

Endpoints on Authorization Server

- Authorization
  - Authenticates/Authorizes resource owner
  - Provides authorization code
- Token
  - Exchange access token for authorization code
  - Exchange access token for refresh token



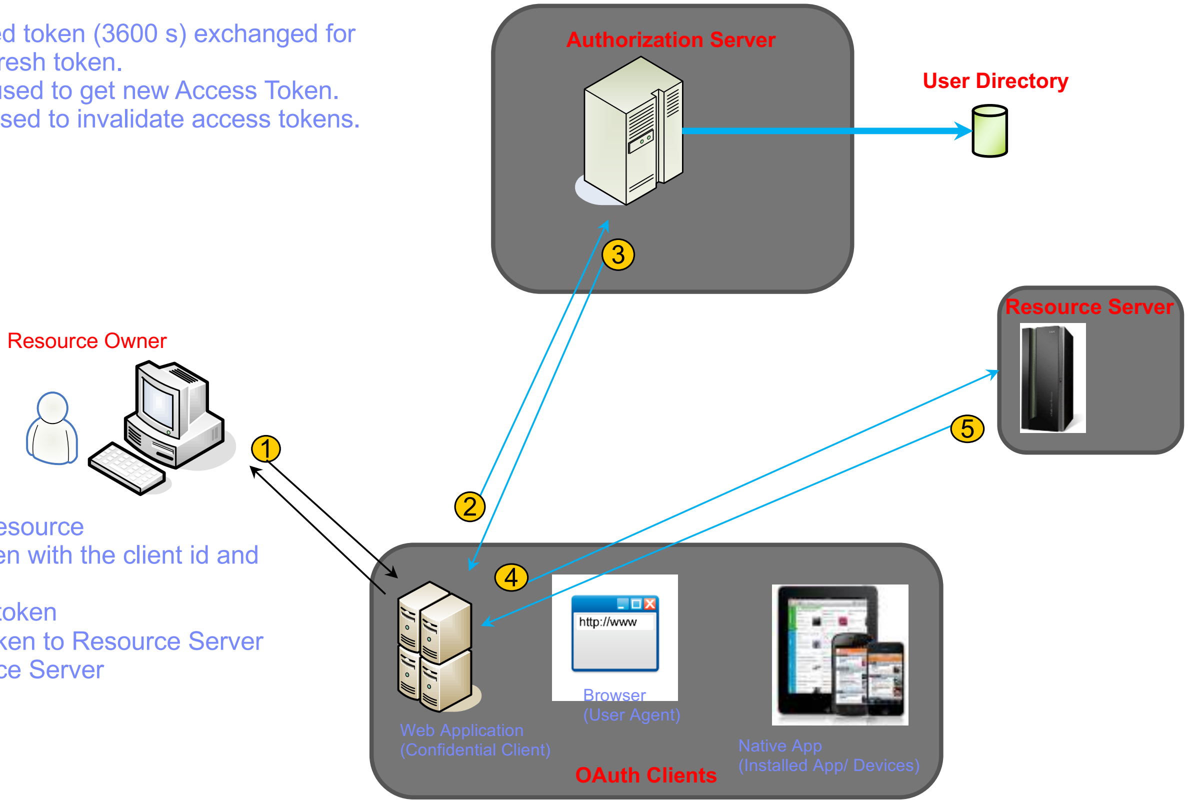
Endpoints on Web Application (Client)

- Resource
  - URL to acces the resource
- Redirect URI
  - Resource owner is redirected to this endpoint with authorization code. Must be registered with authorization server

Application / Client Credentials

Types of Tokens

- Access Token – shortlived token (3600 s) exchanged for authorization code or refresh token.
- Refresh Token – Token used to get new Access Token.
- Revoke Token – Token used to invalidate access tokens.



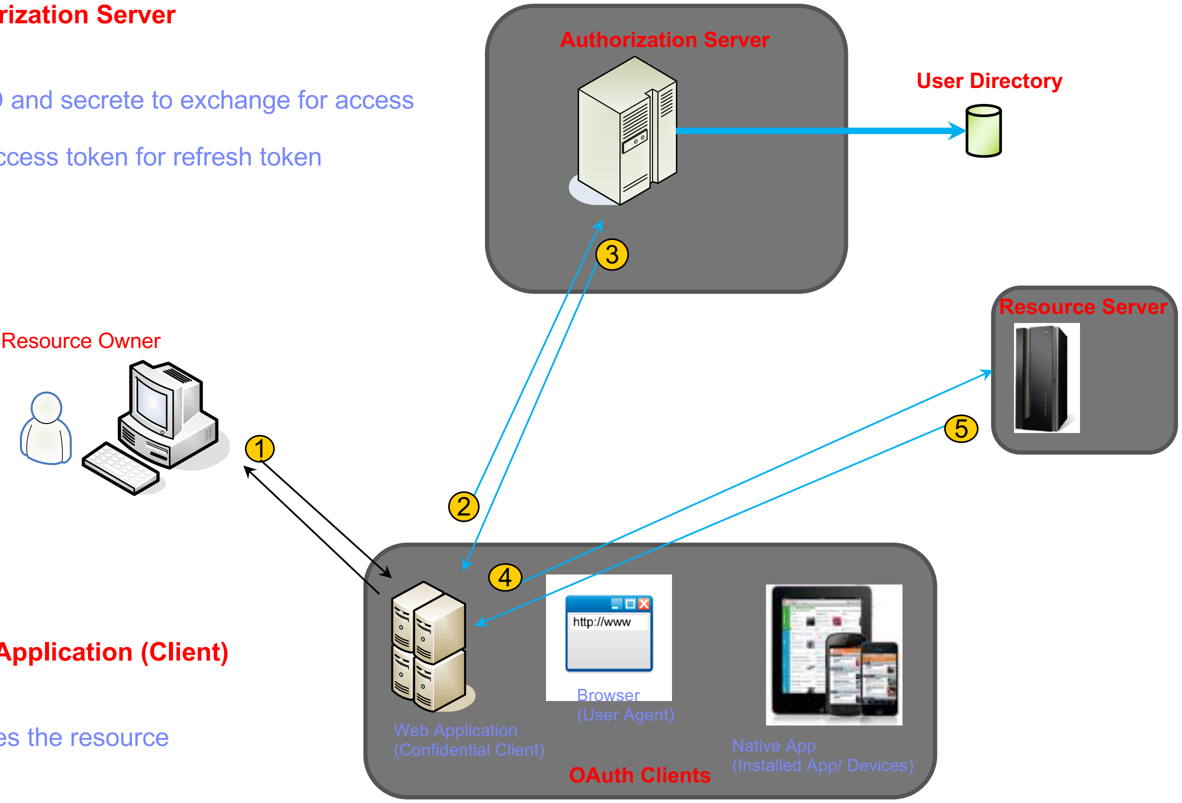
Authorization Code Flow

1. Request for protected resource
2. Request for access token with the client id and secrete
3. Response with Access token
4. Request with access token to Resource Server
5. Response from Resource Server
6. Response to client

Application / Client Credentials

Endpoints on Authorization Server

- Token
    - Use client ID and secrete to exchange for access token
    - Exchange access token for refresh token



Endpoints on Web Application (Client)

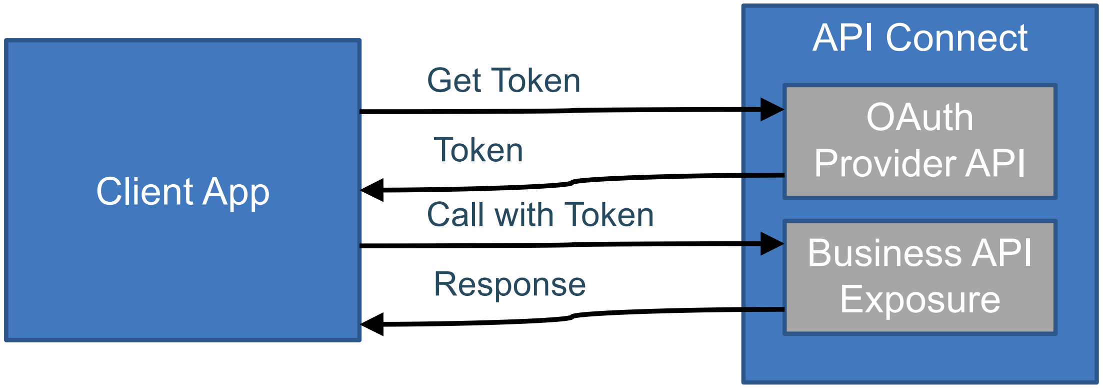
- Resource
    - URL to acces the resource

All grant types

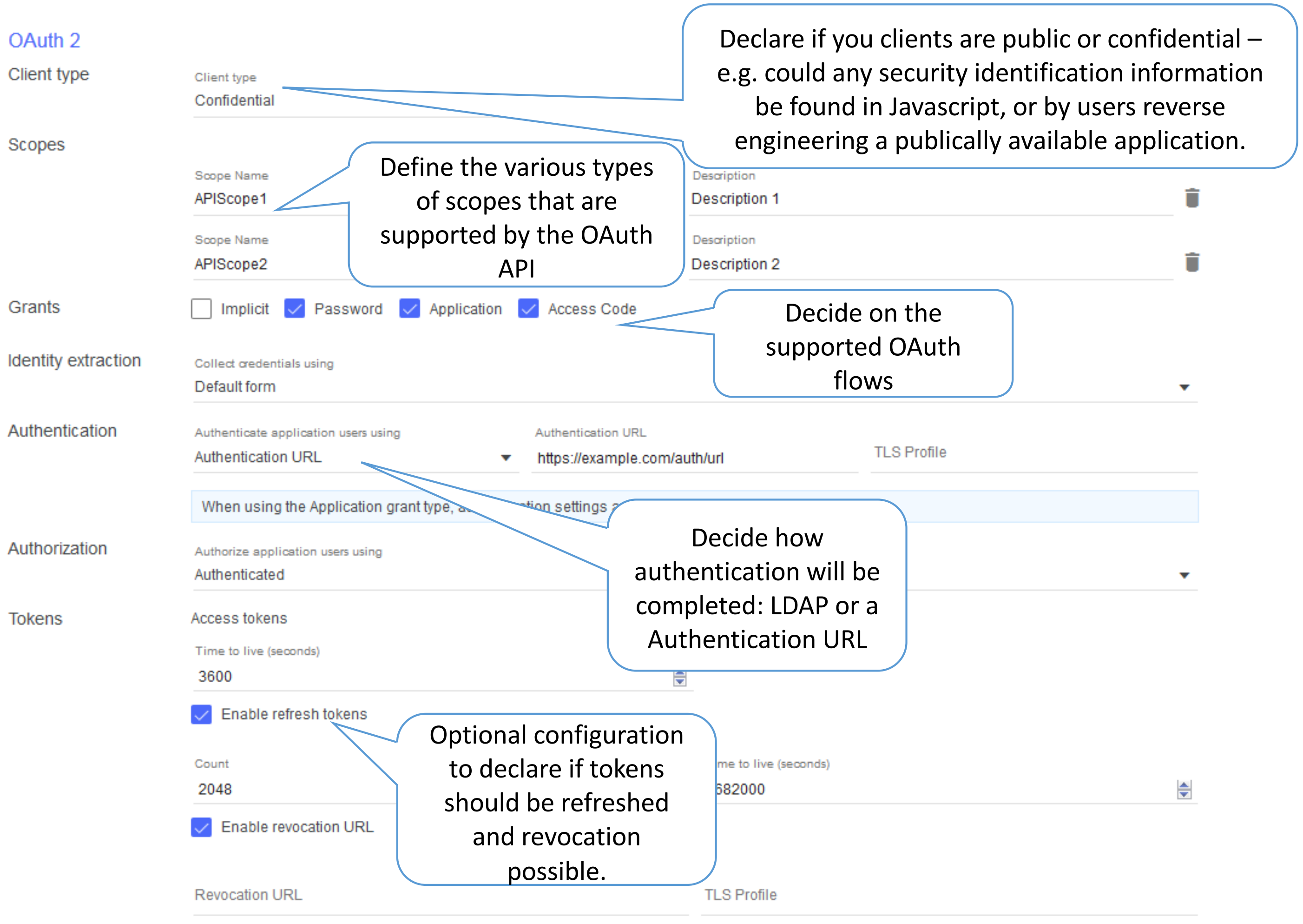
There are four core OAuth grant type:

- **Authorisation code grant** – as described in the previous slides
- **Implicit grant** – similar to the Authorisation code grant described previously, however the user will be redirected in a browser to the authorisation server, sign in, authorise the request but instead of being returned to the client with an ***authentication code*** they are redirected with an ***access token*** straight away. Care needs to be taken as the access token is more freely available in then style of interaction.
- **Resource owner credentials grant** - the client itself will ask the user for their username and password (as opposed to being redirected to the authorisation server to authenticate). The client will send the user credentials to the authorisation server along with the client's own credentials. If the authentication is successful then the client will be issued with an ***access token***. This method is often used for trusted clients, such as Googles own mobile app.
- **Client credentials grant** - This grant is similar to the resource owner credentials grant except only the client's credentials are used to authenticate a request for an ***access token***. This grant should only be allowed to be used by trusted clients. It is a common grant type for machine-to-machine authentication.

Mapping to API Connect



Configuring OAuth API in API Connect



OAuth 2

Client type

Client type
Confidential

Declare if you clients are public or confidential – e.g. could any security identification information be found in Javascript, or by users reverse engineering a publically available application.

Scopes

Scope Name
APIScope1

Description
Description 1

Scope Name
APIScope2

Description
Description 2

Define the various types of scopes that are supported by the OAuth API

Grants

☐ Implicit ☑ Password ☑ Application ☑ Access Code

Decide on the supported OAuth flows

Identity extraction

Collect credentials using
Default form

Authentication

Authenticate application users using
Authentication URL

Authentication URL
https://example.com/auth/url

TLS Profile

When using the Application grant type, a tion settings a

Decide how authentication will be completed: LDAP or a Authentication URL

Authorization

Authorize application users using
Authenticated

Tokens

Access tokens

Time to live (seconds)
3600

☑ Enable refresh tokens

Count
2048

me to live (seconds)
682000

☑ Enable revocation URL

Revocation URL

TLS Profile

Optional configuration to declare if tokens should be refreshed and revocation possible.

JSON Web Token

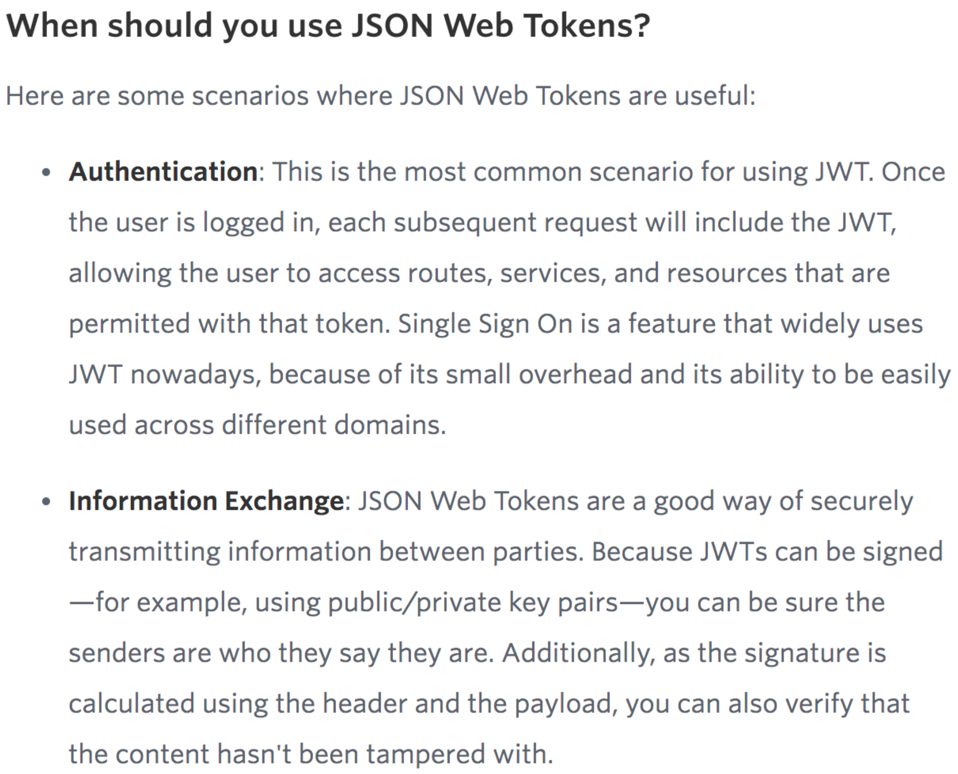
Use Case: Do you have support for JWT

Introduction

JSON Web Token (JWT) is:

- A compact, URL-safe way of representing claims that are to be transferred between two parties.
- Based on RFC 7519 that defines a self-contained way for securely transmitting information between parties as a **JSON** object.
- A policy that can be verified and trusted because it is digitally signed.
- Composed of a header, a payload, and a signature

Good Use Cases for JWT



When should you use JSON Web Tokens?

Here are some scenarios where JSON Web Tokens are useful:

- **Authentication**: This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.
- **Information Exchange**: JSON Web Tokens are a good way of securely transmitting information between parties. Because JWTs can be signed—for example, using public/private key pairs—you can be sure the senders are who they say they are. Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

Implementation

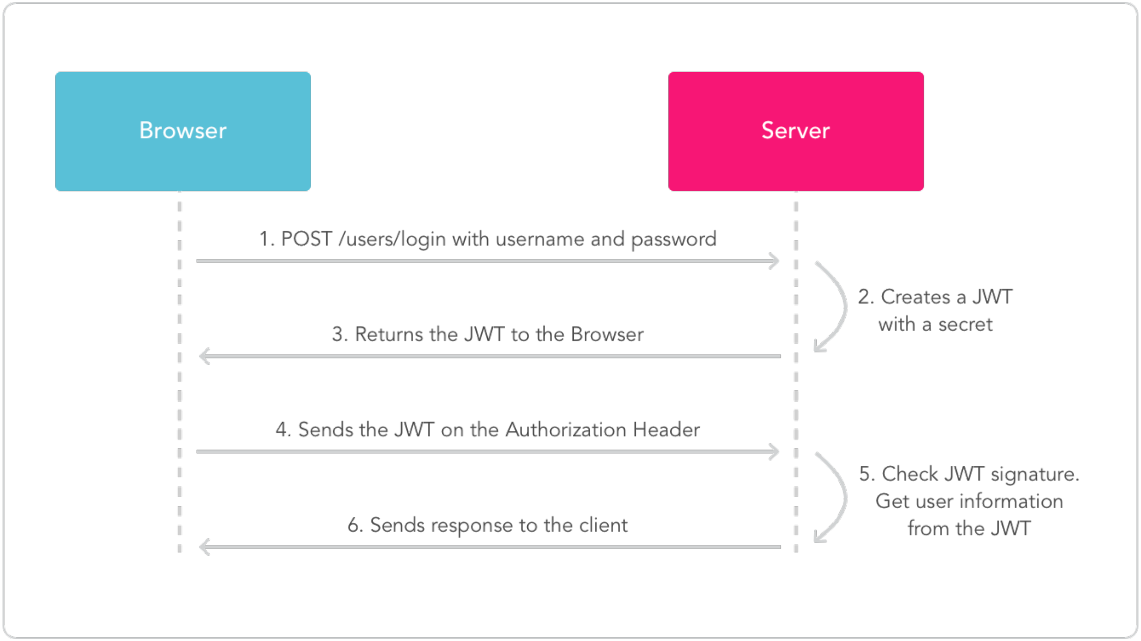
- The Generate JWT policy enables you to generate claims and configure whether they are to be used as either:
  - the payload of a JSON Web Signature (JWS) structure
  - the plain text of a JSON Web Encryption (JWE) structure.
  - specifying the cryptographic material for both the JWS and the JWE produces a nested JWT that is both digitally signed and encrypted.
- The JWT is then assigned to the Authorization header as either:
  - a Bearer token (the default option)
  - the runtime variable in the JSON Web Token (JWT) property, if specified.
- You can attach this policy to the following API flows:
  - REST
  - SOAP

Pre requisites

- IBM API Connect Version 5.0.1 or later.
- IBM DataPower V7.5 with the Application Optimization (AO) option – not supported in Micro Gateway
- If you are using one or more cryptographic objects, they must be located in the IBM API Connect domain on the DataPower appliance. The cryptographic objects must reference the Shared Secret Key or certificate that is needed to encrypt or sign the JWT contents.
- If a JSON Web Key (JWK) is being used, it must be referenced by a runtime variable.

How it works

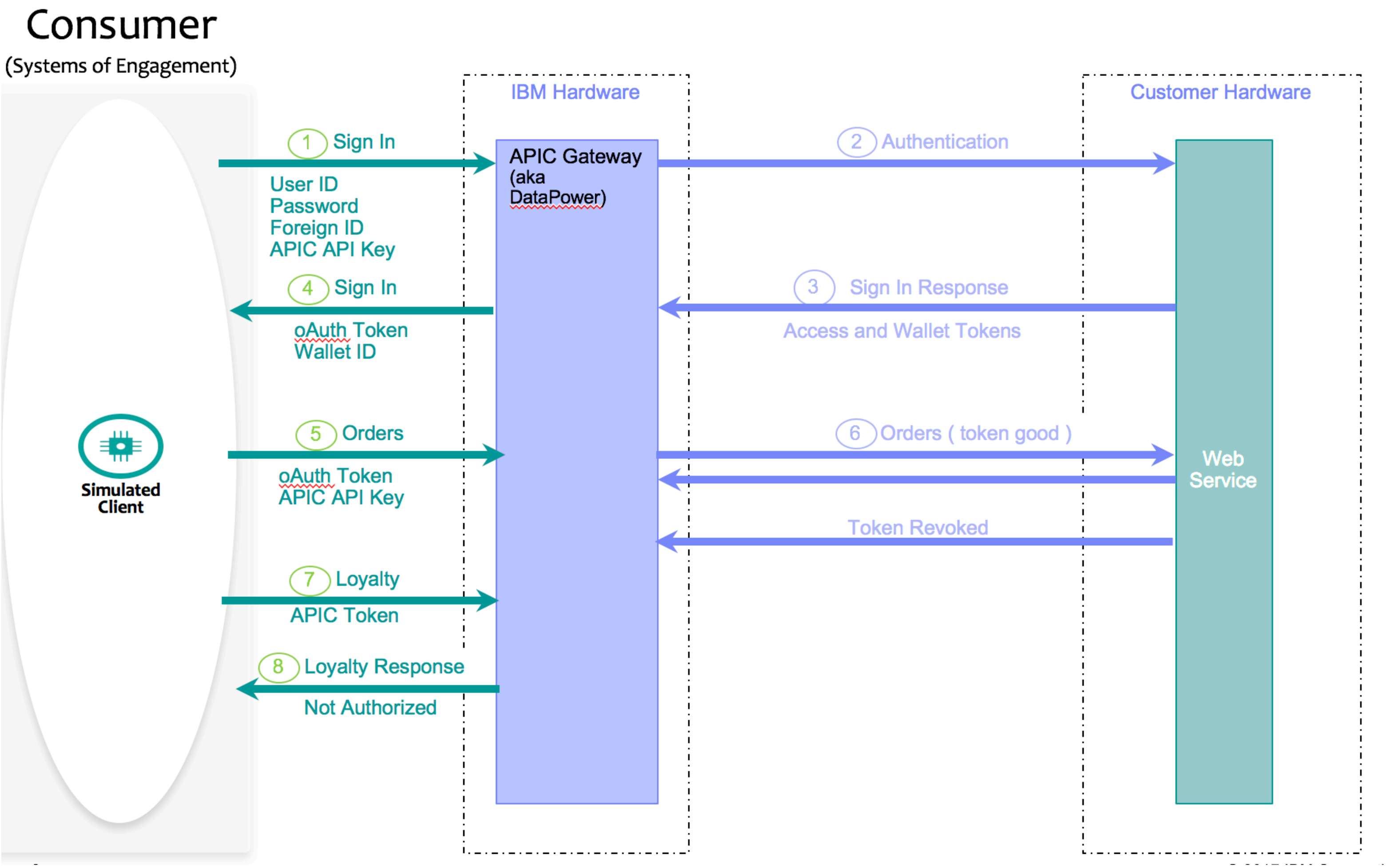
- User signs into an authentication server
- JWT returned to authenticated user
- User passes JWT when making API calls
- Application verifies and API call allowed to proceed


Browser
Server
1. POST /users/login with username and password
2. Creates a JWT with a secret
3. Returns the JWT to the Browser
4. Sends the JWT on the Authorization Header
5. Check JWT signature. Get user information from the JWT
6. Sends response to the client

oAuth 2 Legged

Use Case: I have a mobile app and I want to oauth protect my api so API Gateway can handle session management?

oAuth



Consumer

(Systems of Engagement)

Simulated Client

IBM Hardware

APIC Gateway (aka DataPower)

Customer Hardware

Web Service

1 Sign In

User ID
Password
Foreign ID
APIC API Key

2 Authentication

3 Sign In Response

Access and Wallet Tokens

4 Sign In

oAuth Token
Wallet ID

5 Orders

oAuth Token
APIC API Key

6 Orders (token good)

Token Revoked

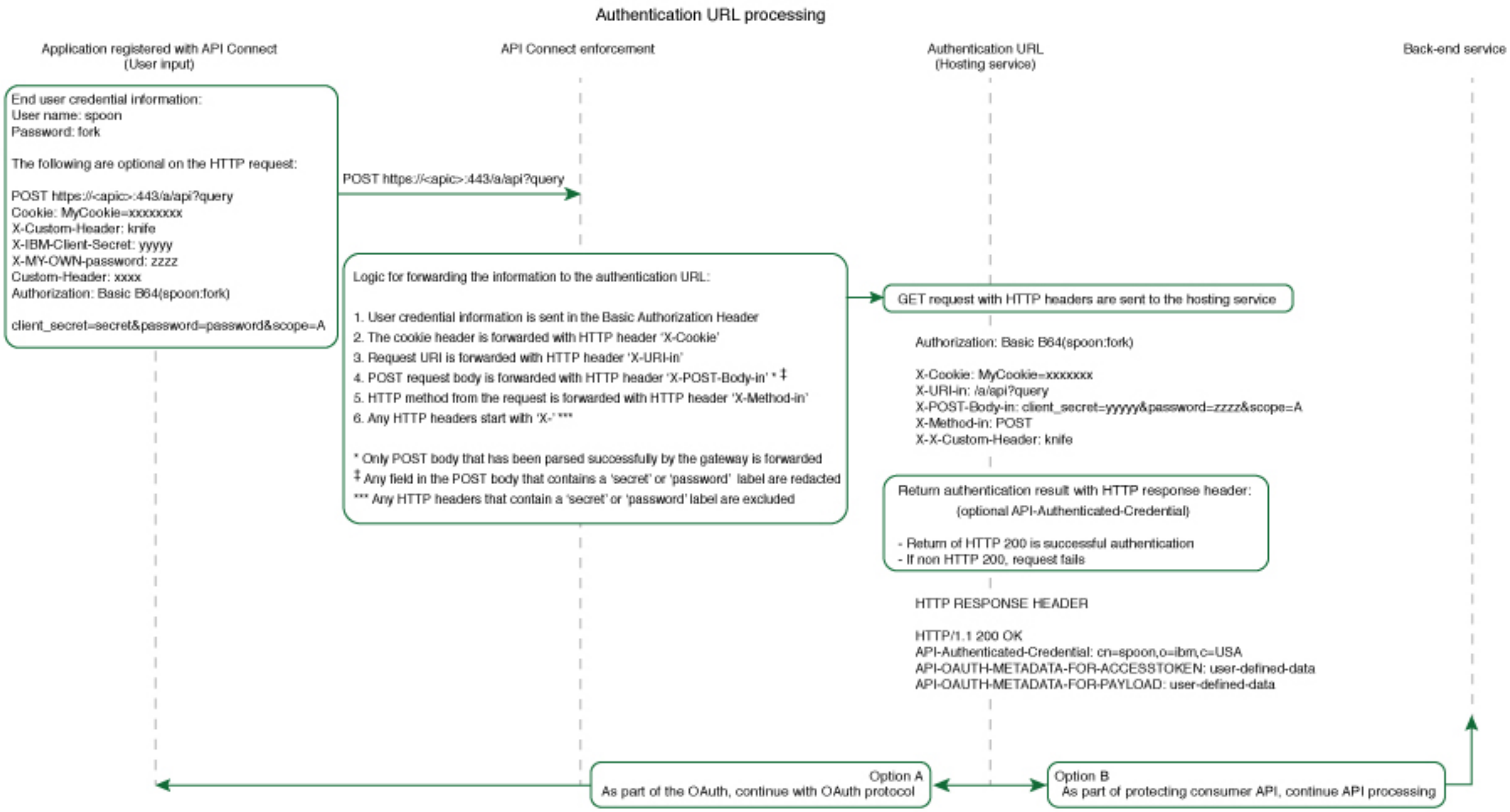
7 Loyalty

APIC Token

8 Loyalty Response

Not Authorized

oAuth



Authentication URL processing

Application registered with API Connect
(User input)

API Connect enforcement

Authentication URL
(Hosting service)

Back-end service

End user credential information:
User name: spoon
Password: fork

The following are optional on the HTTP request:

```
POST https://<apic>:443/a/api?query
Cookie: MyCookie=xxxxxxxx
X-Custom-Header: knife
X-IBM-Client-Secret: yyyyy
X-MY-OWN-password: zzzz
Custom-Header: xxxx
Authorization: Basic B64(spoon:fork)

client_secret=secret&password=password&scope=A
```

POST https://<apic>:443/a/api?query

Logic for forwarding the information to the authentication URL:

1. User credential information is sent in the Basic Authorization Header
2. The cookie header is forwarded with HTTP header 'X-Cookie'
3. Request URI is forwarded with HTTP header 'X-URI-in'
4. POST request body is forwarded with HTTP header 'X-POST-Body-in' * ‡
5. HTTP method from the request is forwarded with HTTP header 'X-Method-in'
6. Any HTTP headers start with 'X-' ***

* Only POST body that has been parsed successfully by the gateway is forwarded
‡ Any field in the POST body that contains a 'secret' or 'password' label are redacted
*** Any HTTP headers that contain a 'secret' or 'password' label are excluded

GET request with HTTP headers are sent to the hosting service

```
Authorization: Basic B64(spoon:fork)

X-Cookie: MyCookie=xxxxxxx
X-URI-in: /a/api?query
X-POST-Body-in: client_secret=yyyyy&password=zzzz&scope=A
X-Method-in: POST
X-X-Custom-Header: knife
```

Return authentication result with HTTP response header:
(optional API-Authenticated-Credential)

- Return of HTTP 200 is successful authentication
- If non HTTP 200, request fails

HTTP RESPONSE HEADER

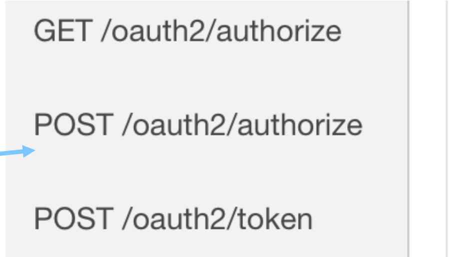
```
HTTP/1.1 200 OK
API-Authenticated-Credential: cn=spoon,o=ibm,c=USA
API-OAUTH-METADATA-FOR-ACCESSTOKEN: user-defined-data
API-OAUTH-METADATA-FOR-PAYLOAD: user-defined-data
```

Option A
As part of the OAuth, continue with OAuth protocol

Option B
As part of protecting consumer API, continue API processing

APIC Conducting Token Management

- Provides endpoints to generate tokens

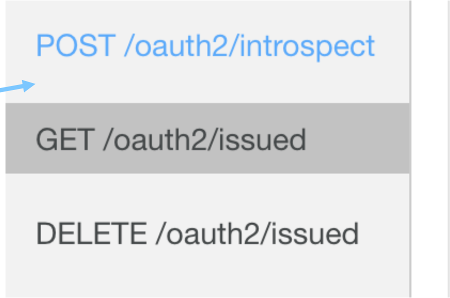


GET /oauth2/authorize

POST /oauth2/authorize

POST /oauth2/token

- Provides endpoints to manage tokens



POST /oauth2/introspect

GET /oauth2/issued

DELETE /oauth2/issued

oAuth/ JWT Questions

- Open Conversation / Questions?

Back to Presentation Topics