OAuth/OIDC - One Login to Rule them All

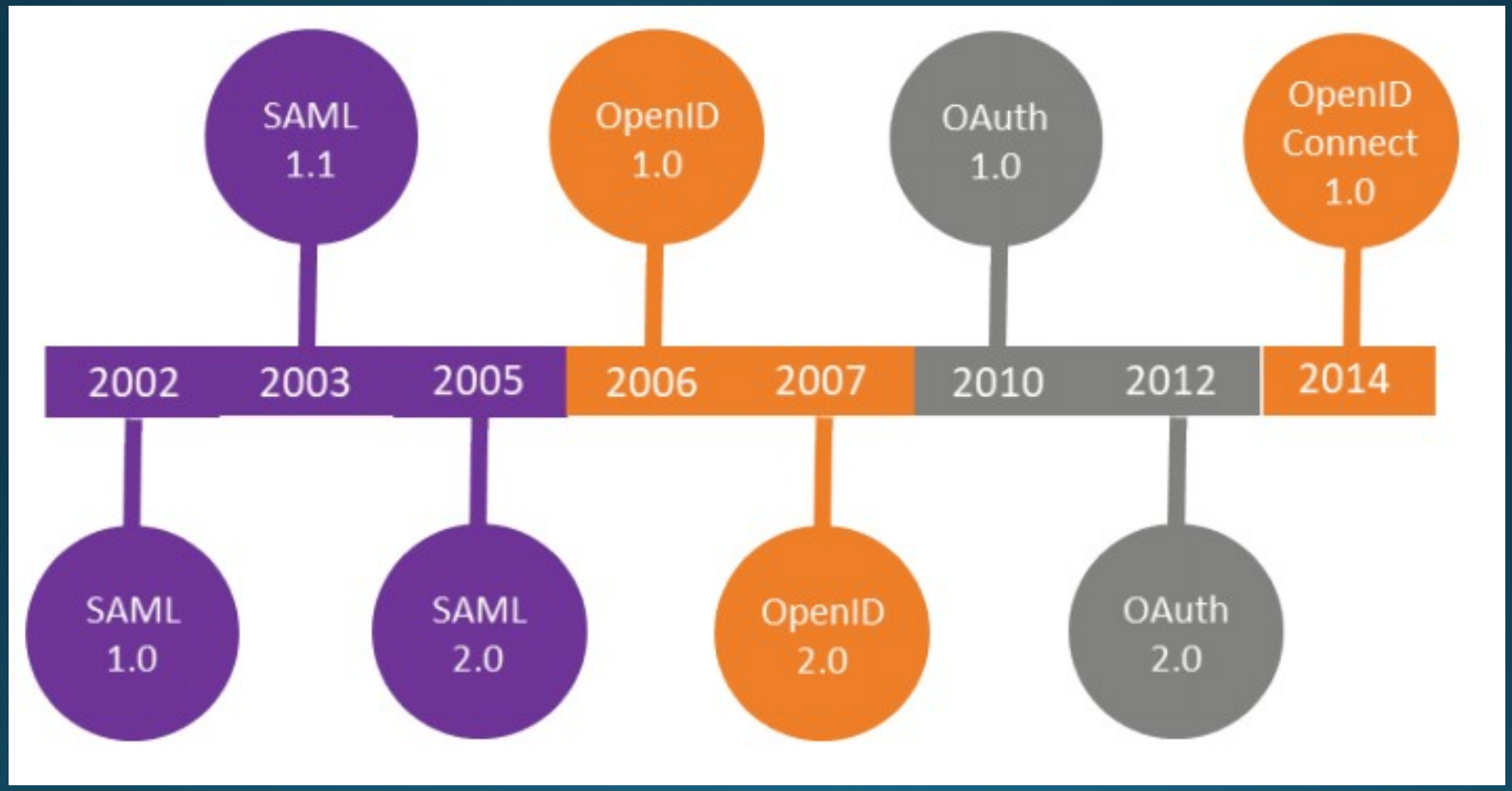
Agenda

- History and Definition of OAuth 2.0
- OAuth 2.0 Roles
- OIDC (OpenID Connect)
- JWT Tokens (ID, Access, Refresh)
- Grant Types (Flows)
- OIDC Federation
- Token Renewal

Acronyms

- SAML – Security Assertion Markup Language
- OIDC – Open ID Connect
- JSON – Java Script Object Notation
- JWT – JSON Web Tokens
- REST – Representational State Transfer
- PKCE – Proof Key For Code Exchange
- JIT – Just In Time (Provisioning)

History of OAuth 2.0



What is OAuth?

- OAuth is an open standard for access delegation, commonly used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords.

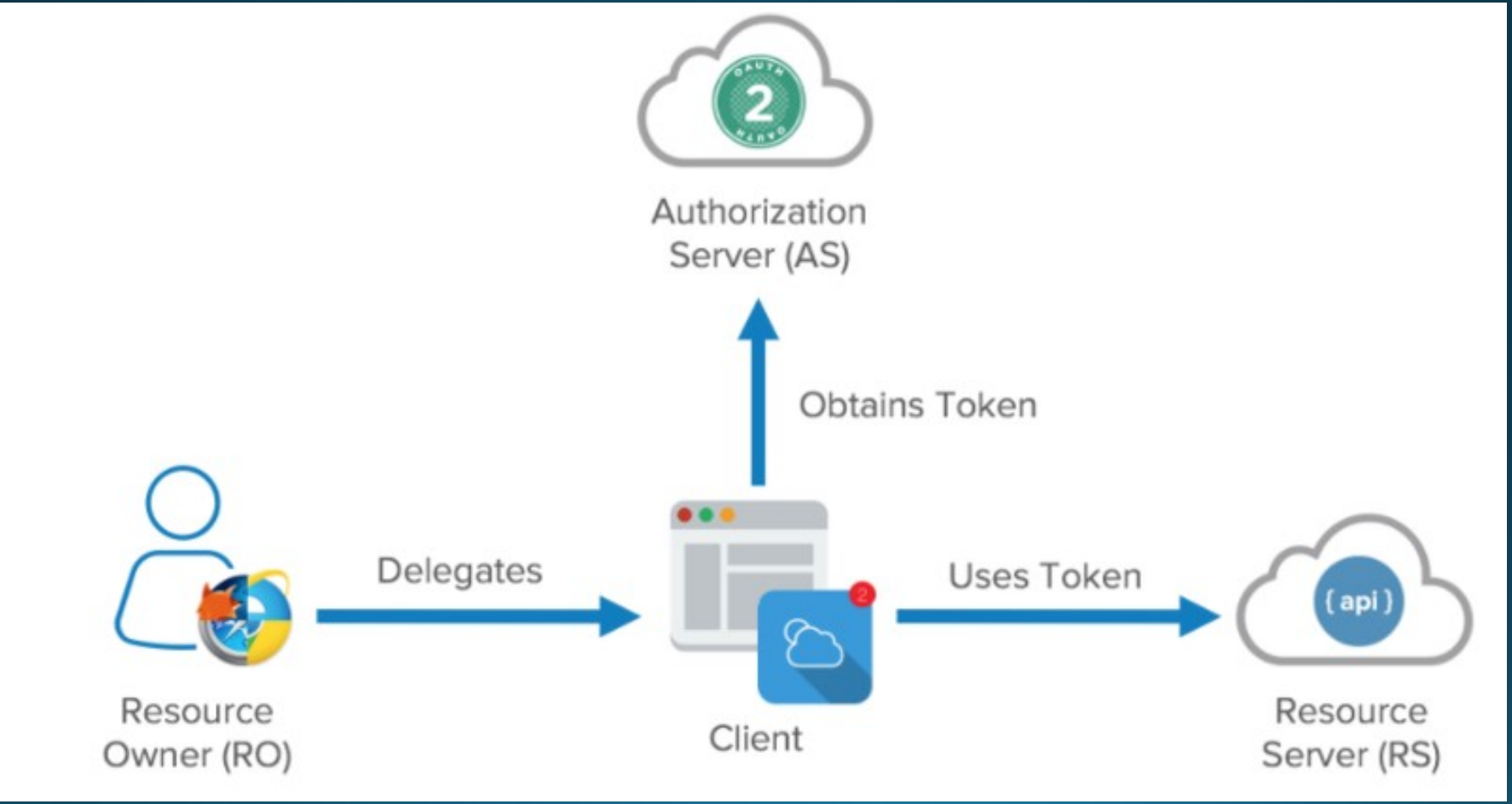
- This mechanism is used by companies such as Amazon, Google, Facebook, Microsoft and Twitter to permit the users to share information about their accounts with third party applications or websites

- OAuth 2.0 is a complete redesign from OAuth 1.0, and the two are not compatible. If you create a new application today, use OAuth 2.0.

OAuth 2.0 Roles

- **Authorization Server** (Identity Provider)— The server that issues the access token. For example, Okta.
- **Resource Owner** — Normally your application's end user that grants permission to access the resource server with an access token.
- **Client** — The application that requests the access token from the authorization server and then passes it to the resource server.
- **Resource Server** — Accepts the access token and must verify that it's valid. In this case this is your application.

OAuth 2.0 Roles



What is OIDC (OpenID Connect)?

- Authentication standard built on top of OAuth 2.0.
- (Identity, Authentication) + OAuth 2.0 = OpenID Connect
- Adds ID token
- Standardizes scopes (restricts authority of access token), endpoint discovery and dynamic registration for clients.

OAuth/OIDC Tokens

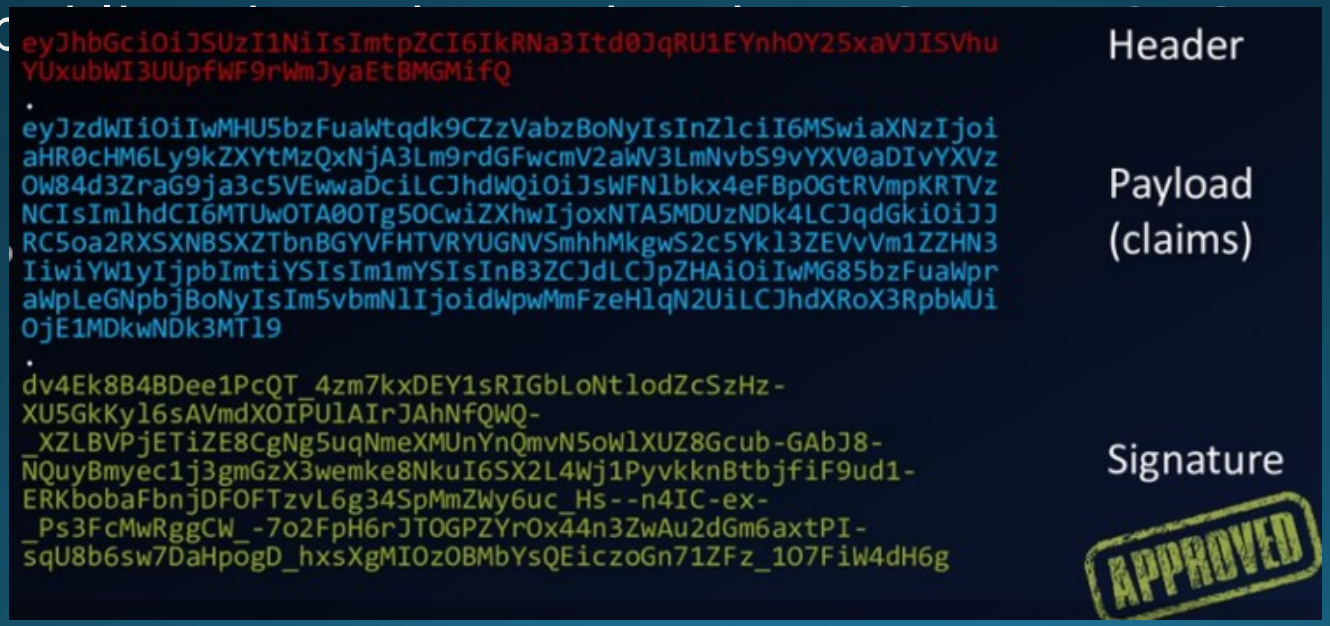
- ID Token
  - Contains information about the end u

- Access Token
  - For API Access
  - Short Lived

- Refresh Token
  - For renewing Access Token
  - Long Lived

JWT (JSON Web Token)

JSON Web Token (JWT) is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the **HMAC** algorithm) or a p



OAuth 2.0 Grant Types/Flows

In **OAuth** 2.0, the term "**grant type**" refers to the way an application gets an access token

1) **Authorization Code Grant Type**

- Most common OAuth 2.0 grant type.
- Used by both web apps and native apps to get an access token after a user authorizes an app.

2) **Client Credentials Grant Type**

- Server-side (confidential) client applications with no end user.
- It involves a single, authenticated request to the /token endpoint, which returns an access token.

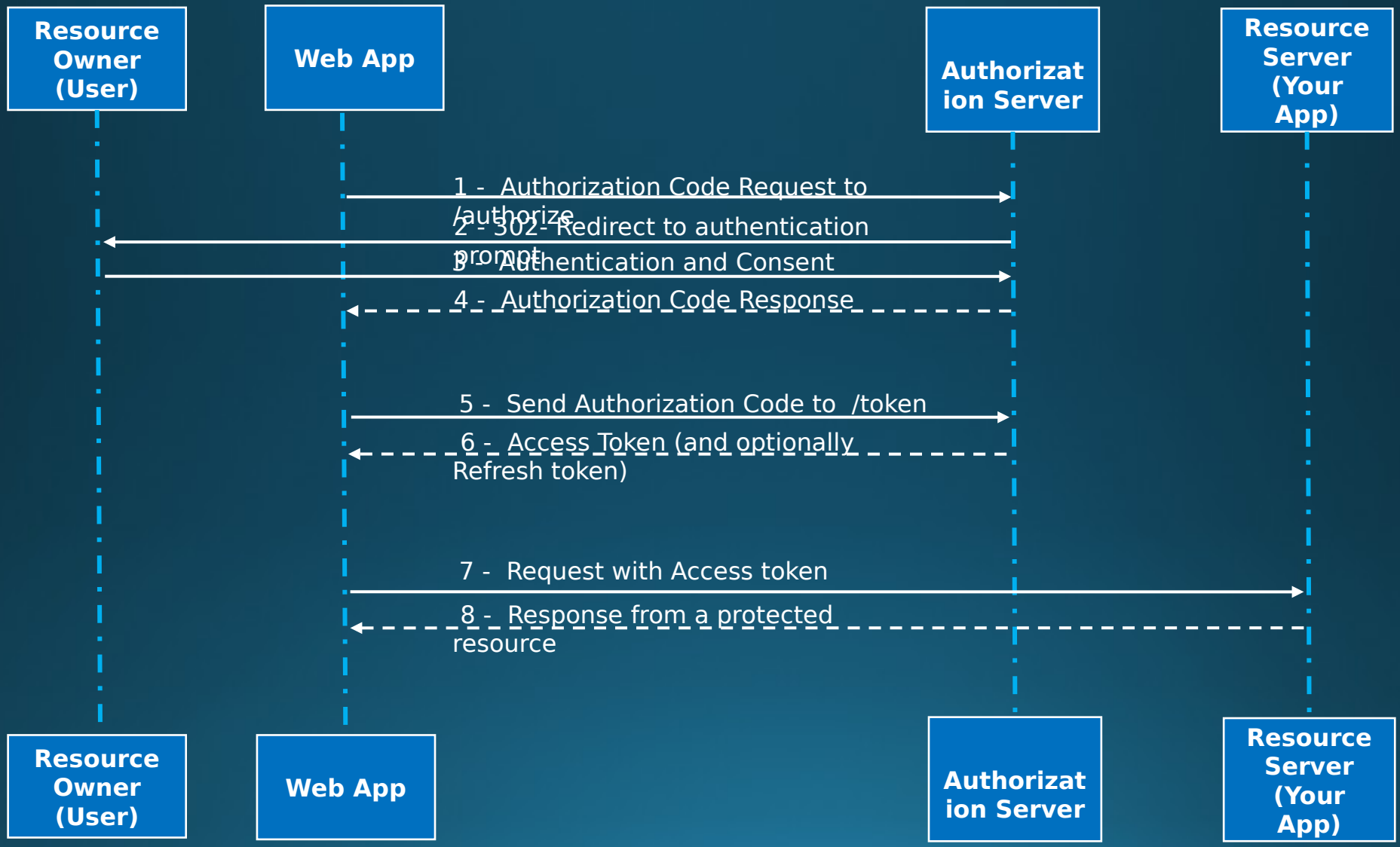
3) **Resource Owner Password Grant Type**

- Is a way to get an access token given a username and password.
- Used only by a service's own mobile apps and is not made available to third party developers.

4) **Implicit Grant Type**

- Deprecated in favor of Authorization Code Grant Type with PKCE (extension for public clients)

Authorization Code Flow



Authorization Code Flow Sample

```
var redirect_uri = "https://www.sample.com/";
var authorization_endpoint =
"https://sample.okta.com/oauth2/v1/authorize";
var token_endpoint = "https://sample.okta.com/oauth2/v1/token";
var requested_scopes = "openid email profile";
var state = generateRandomString();
var code_verifier = generateRandomString();
var code_challenge = sha256(code_verifier);

localStorage.setItem("pkce_state", state);
localStorage.setItem("pkce_code_verifier", code_verifier);

window.location = authorization_endpoint
      + "?response_type=code"
      + "&response_mode=fragment"
      + "&client_id=" + encodeURIComponent(client_id))
      + "&state=" + encodeURIComponent(state)
      + "&scope=" + encodeURIComponent(requested_scopes)
      + "&redirect_uri=" + encodeURIComponent(redirect_uri)
      + "&code_challenge=" + encodeURIComponent(code_challenge)
      + "&code_challenge_method=S256";
```

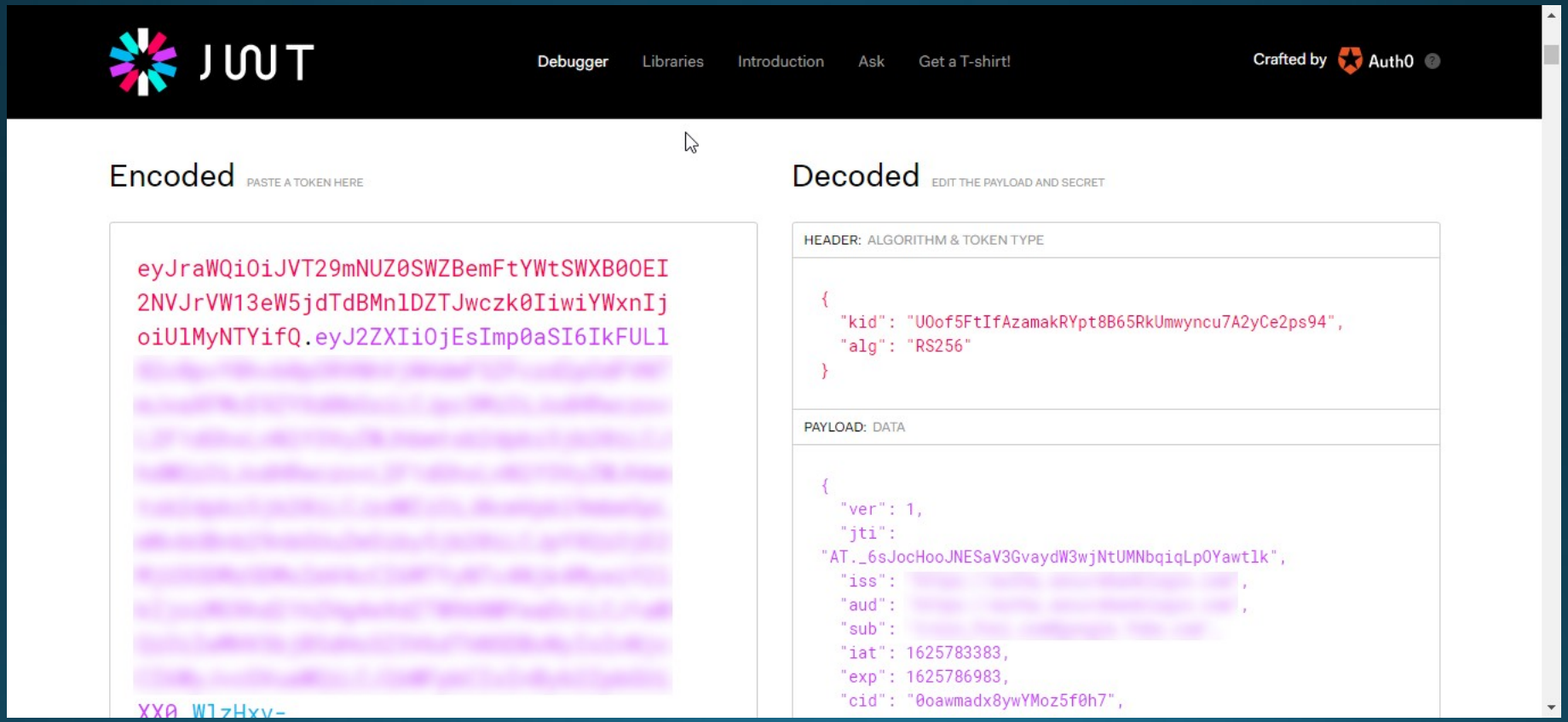
Authorization Server Callback Sample

```
var q = parseQueryString(window.location.hash.substring(1));
if (q.error) {
    alert(q.error + ": " + q.error_description);
}

if (q.code) {
    if (localStorage.getItem("pkce_state") !== q.state) {
        alert("Invalid state");
    }
    sendPostRequest(token_endpoint, {
        grant_type: "authorization_code",
        code: q.code,
        client_id: client_id,
        redirect_uri: redirect_uri,
        code_verifier: localStorage.getItem("pkce_code_verifier")
    }, function(request, body) {
        alert("Access token: " + body.access_token);
    });

    localStorage.removeItem("pkce_state");
    localStorage.removeItem("pkce_code_verifier");
}
```

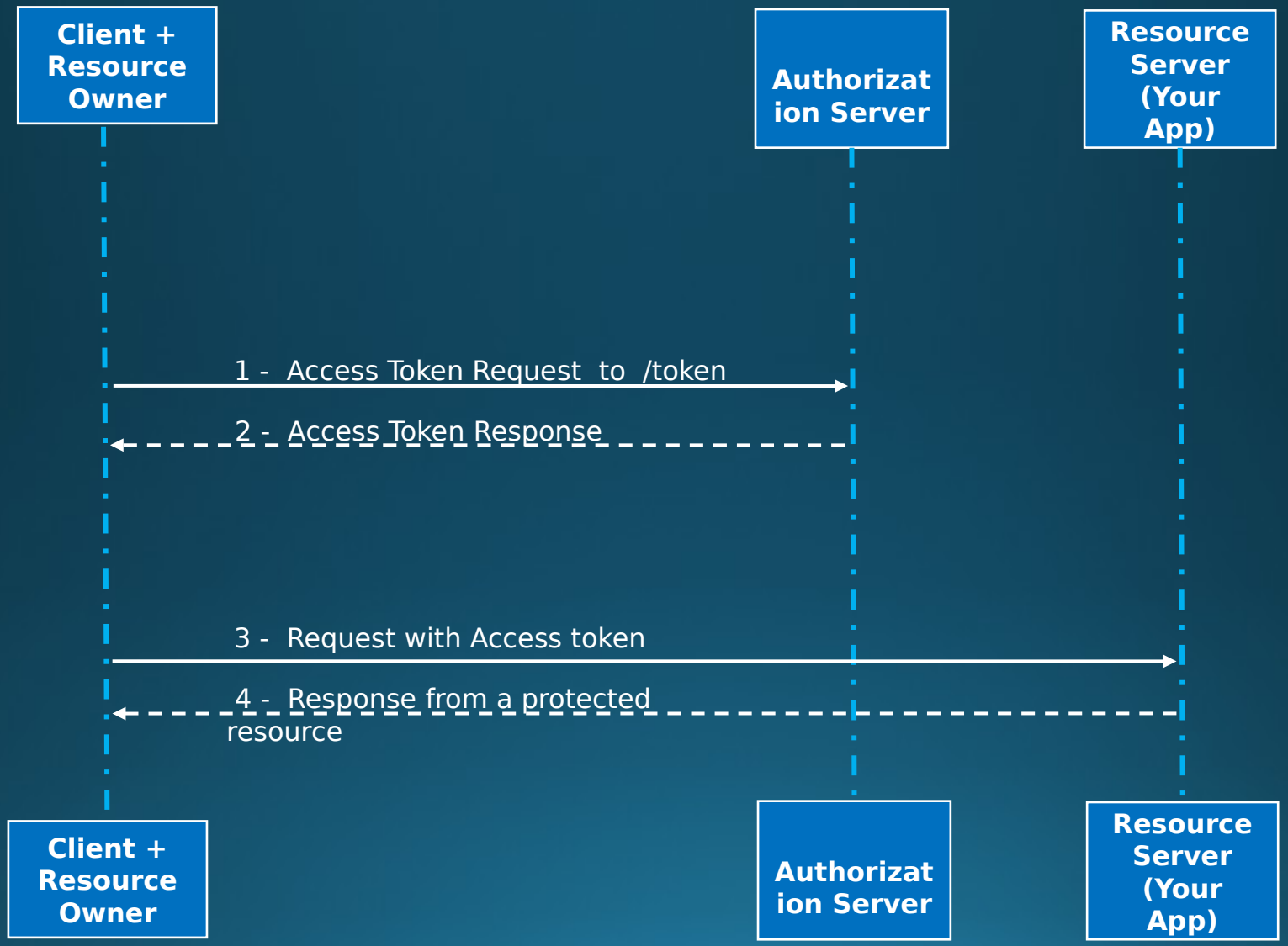
API Request with Bearer Access Token

```
POST /sample-service/1.0/items HTTP/1.1
Authorization: Bearer eyJraWQiOiJnQ3h ... T5p4WIO5asxDiwSQ==
Content-Type: application/json
User-Agent: PostmanRuntime/7.28.1
Accept: */*
Cache-Control: no-cache
Host: www.sample.com
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 40

{"itemId":"9999999999","name":"My Item"}
```

Decode Token at jwt.io



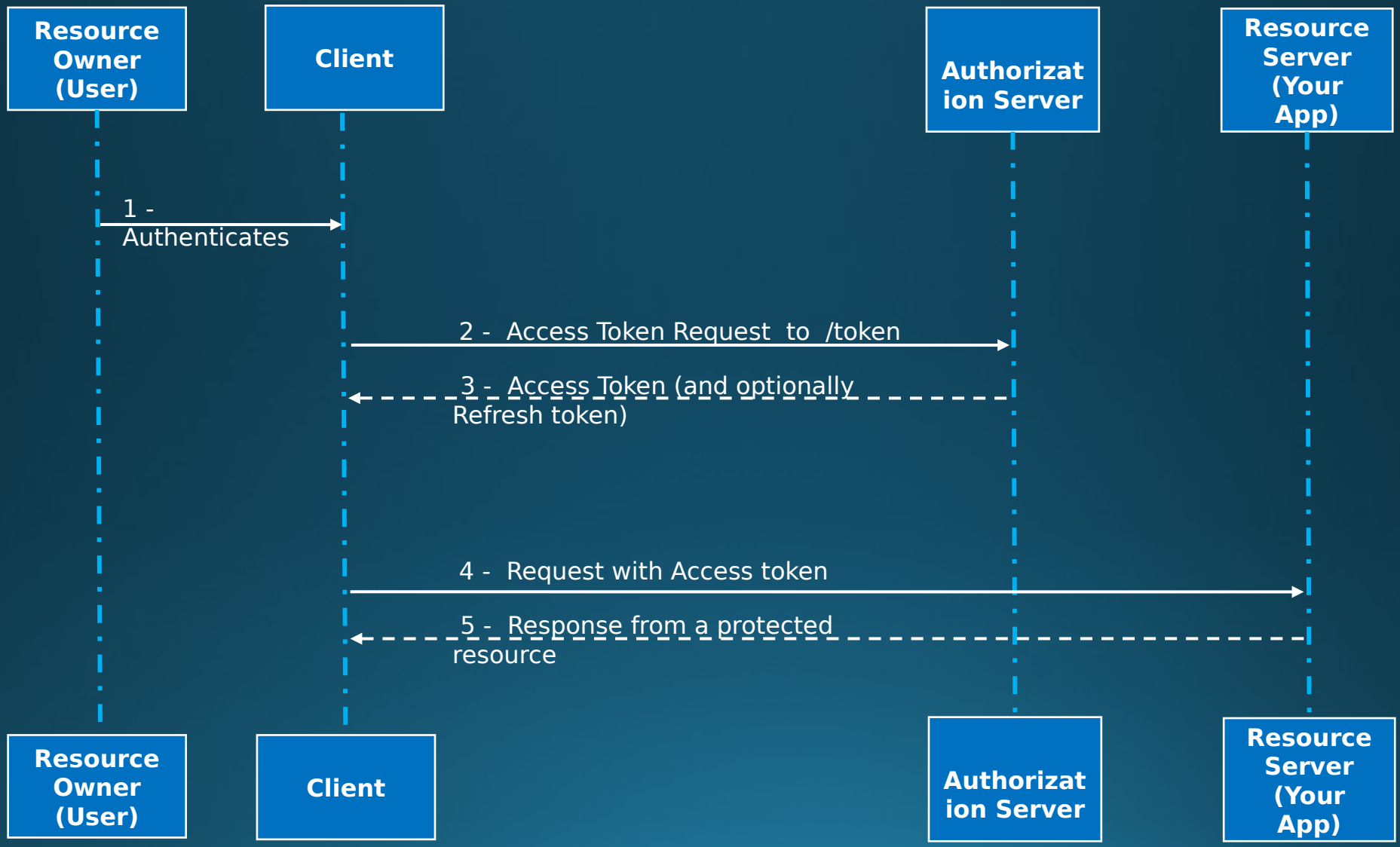
Client Credential Flow



Client Credentials Flow Sample

```
curl --request POST \
  --url https://sample.okta.com/oauth2/default/v1/token \
  --header 'accept: application/json' \
  --header 'authorization: Basic MG9hY...' \
  --header 'cache-control: no-cache' \
  --header 'content-type: application/x-www-form-urlencoded' \
  --data 'grant_type=client_credentials&scope=customScope'
```

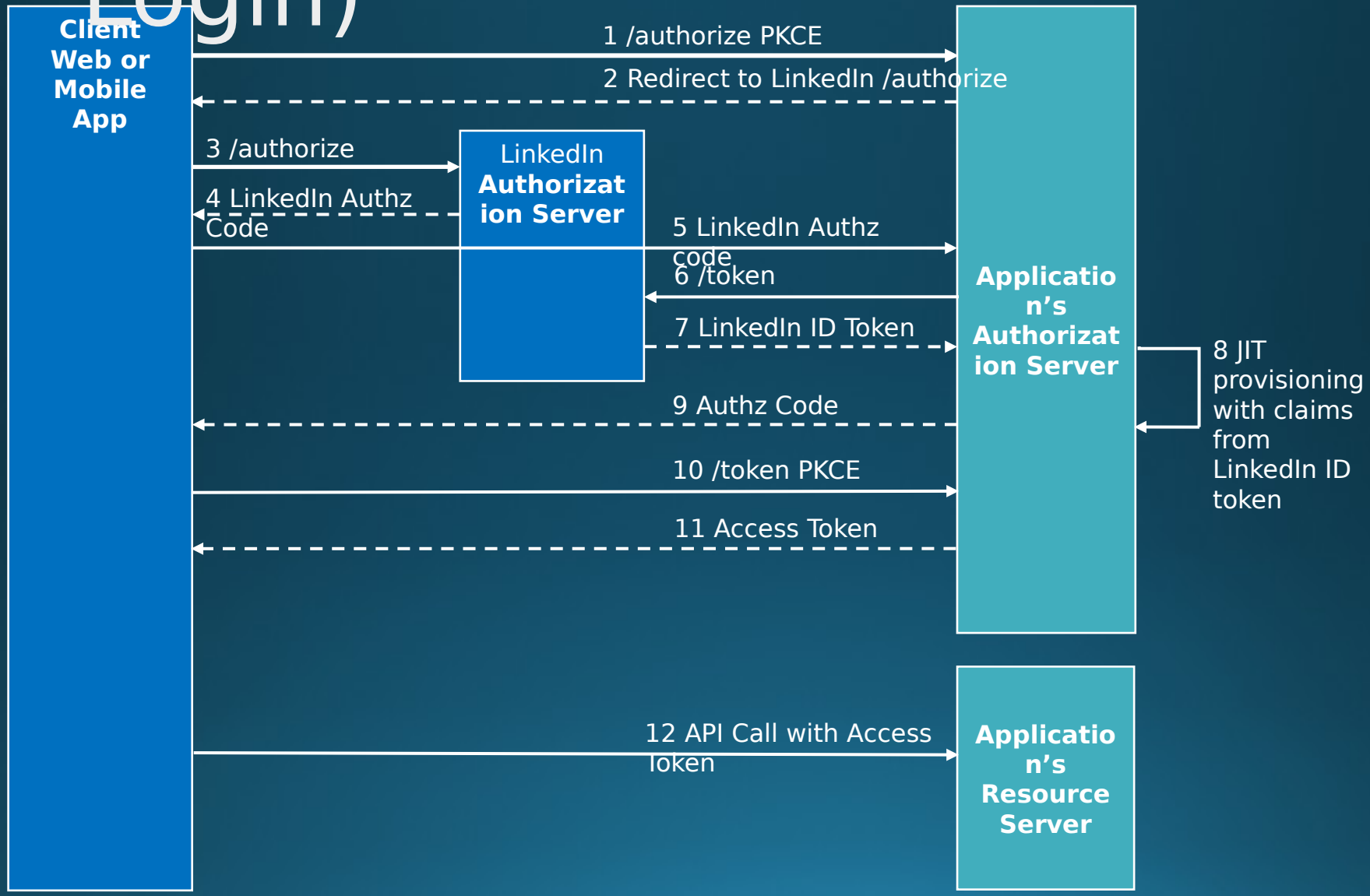
Resource Owner Password Flow



Resource Owner Flow Sample

```
curl --request POST \
  --url https://sample.okta.com/oauth2/default/v1/token \
  --header 'accept: application/json' \
  --header 'authorization: Basic MG9hYn...' \
  --header 'content-type: application/x-www-form-urlencoded' \
  --data
'grant_type=password&username=testuser1%40example.com&password
=%7CmCovrlnU9oZU4qWGrhQSM%3Dyd&scope=openid'
```

OIDC Federation (Social Login)

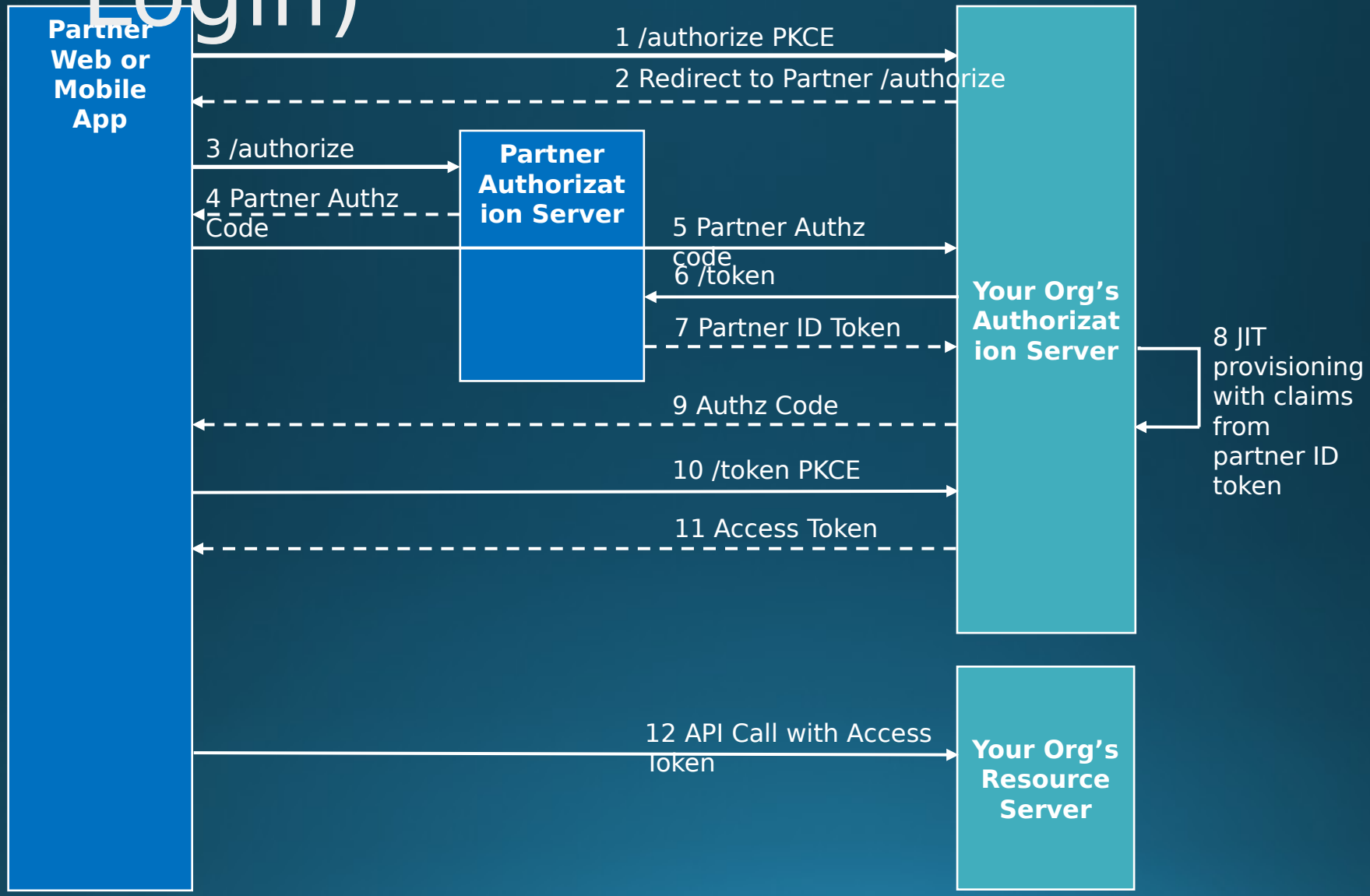


Social Login Statistics

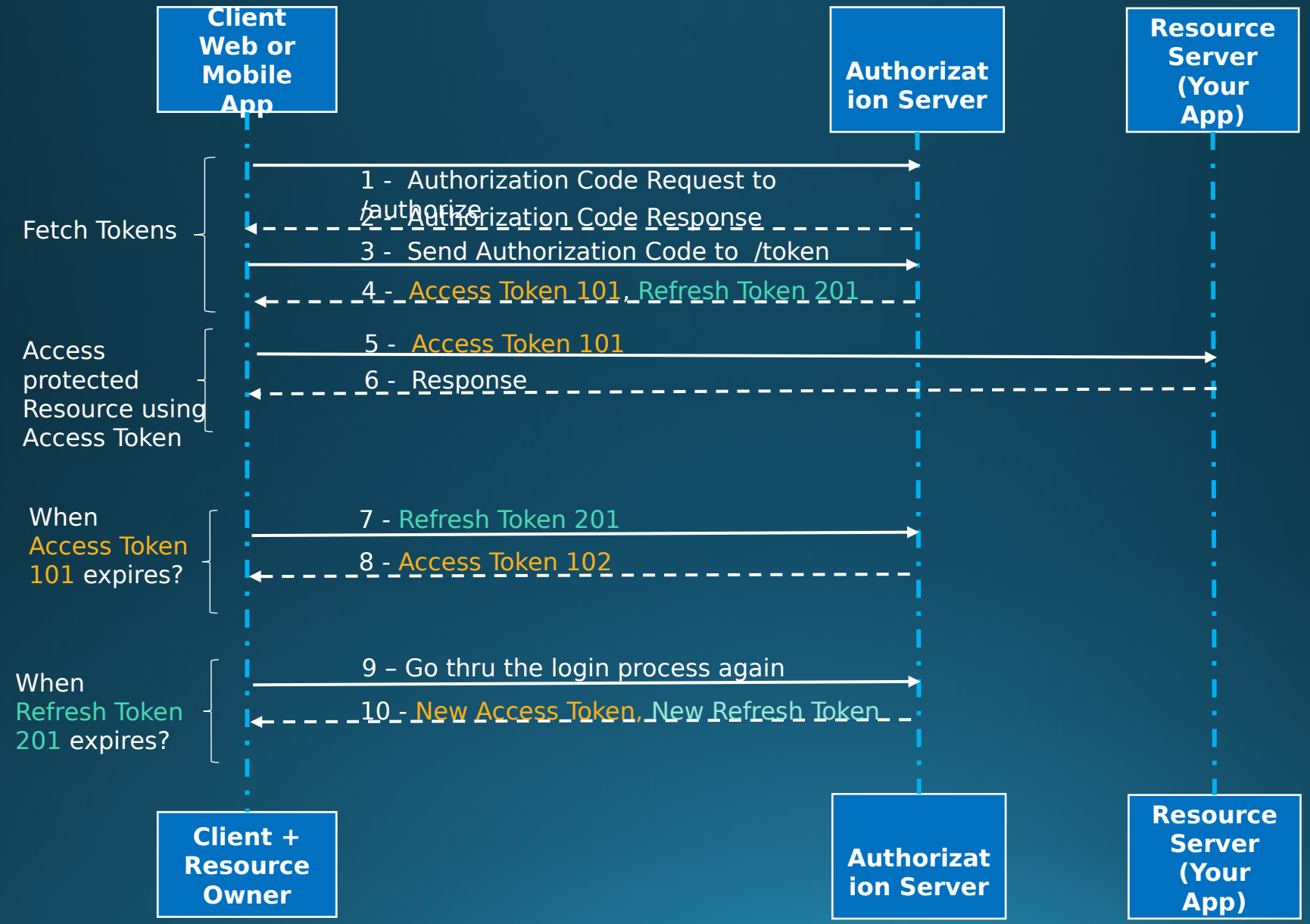
- 86% of users report being bothered by having to create new accounts on websites
- 77% of users believe social login is a good registration solution
- 92% of users will leave a site instead of resetting or recovering login info
- 88% of users admit to entering incomplete or incorrect data on registration forms

For additional details, please refer https://cxl.com/blog/social-login/

OIDC Federation (Partner Login)



Token Renewal

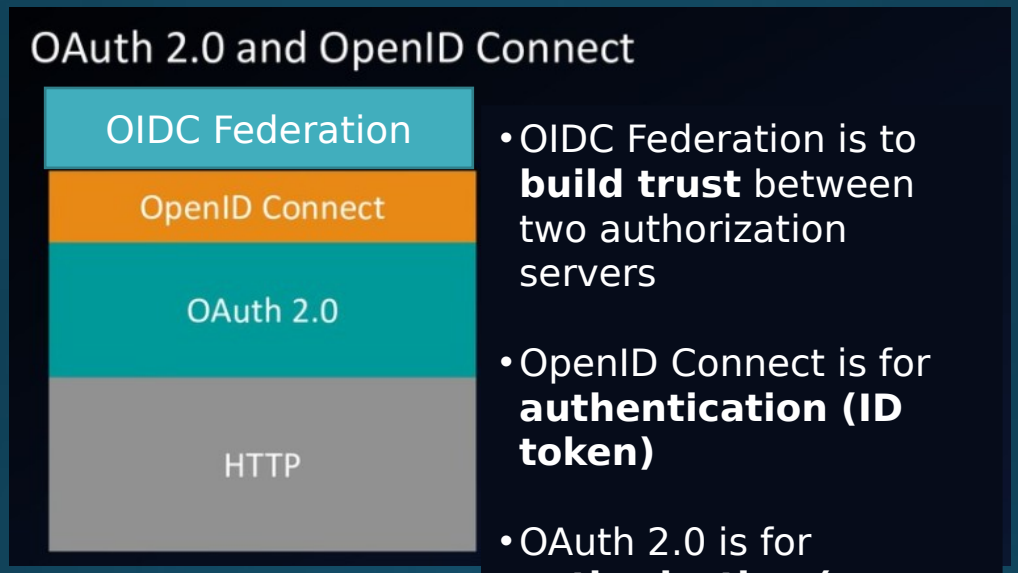


Renew Access Token Using a Refresh Token

```
http --form POST https://sample.okta.com/oauth2/default/v1/token \
  accept:application/json \
  authorization:'Basic MG9hYmg3M...' \
  cache-control:no-cache \
  content-type:application/x-www-form-urlencoded \
  grant_type=refresh_token \
  redirect_uri=http://localhost:8080 \
  scope=offline_access%20openid \
  refresh_token=MIOf-U1zQbyfa3MUfJHhvnUqIut9ClH0xjlDXGJAyqo
```


```
{
    "access_token": "eyJhbGciOiJ[...]K1Sun9bA",
    "token_type": "Bearer",
    "expires_in": 3600,
    "scope": "offline_access%20openid",
    "refresh_token": "MIOf-U1zQbyfa3MUfJHhvnUqIut9ClH0xjlDXGJAyqo",
    "id_token": "eyJraWQiO[...]hMEJQX6WRQ"
}
```

Key Takeaways



- OIDC Federation is to **build trust** between two authorization servers
- OpenID Connect is for **authentication (ID token)**
- OAuth 2.0 is for **authorization (access token)**

Use OAuth 2.0 for:

- **Access without disclosing password**
- **Grant access to your API**
- **Access user data in other systems**

Use OIDC for:

- **Determine the identity of the user**
- **Provide user profile**
- **Log the user in**

Now you have seen how to use OAuth 2.0 and OpenID Connect

Questions?

References

- https://developer.okta.com/blog/2017/06/21/what-the-heck-is-oauth
- https://developer.okta.com/docs/concepts/oauth-openid/
- https://jwt.io/introduction
- https://blog.aspiresys.pl/technology/oauth-vs-oidc-vs-saml-security-battle-royale/
- https://cxl.com/blog/social-login/