Memo

Aan
Bob van der Staak, Geri Wolters, Remco Gehlig

	Kenmerk	Aantal pagina's
Datum	11202186-000-DSC-0014-def	11
3 mei 2018		

	Doorkiesnummer	E-mail
Van	+31(0)6 1019 8112	Stef.Hummel@deltares.nl
Erik de Rooij (Stef Hummel)		

Onderwerp
Uitwerking Digitale Delta authenticatie-flows, geanonimiseerde versie

---

Introduction

The purpose of this document is to describe what possibilities there are for implementing a Digitale Delta (DD) network that uses OpenID connect for authenticating (human) users and external applications. A DD network can be setup in different ways. Here we foresee the following possible configurations:

1. Separate DD-Nodes: Each DD-Node acts as an individual open endpoint to which users connect directly for retrieving data.

2. Multiple DD-Nodes with a browser front-end: A number of DD-Nodes are queried from a web browser that runs a JavaScript application.

3. Multiple DD-Nodes with a DD Web application: End Users run queries on a DD Web application. The web application runs on a server and has the ability to securely store configuration and make connections to DD-Nodes.

4. Multiple DD-Nodes are queried without user input required: An automated process runs queries on DD-Nodes to extract data.

Test Setup

For all tests we make use of the Google Identity provider.
(google configuration page: https://console.developers.google.com/apis)

Client IDs:
email : <client1_e-mail>
pw: <client1_pw>
clientId: <client1_id>
client secret: <client1_secret>

email : <client2_e-mail>

pw: <some_pw>
clientId: <client2_id>
client secret: <client2_secret>

Redirect URI's:
These endpoints represent DD-Nodes or DD- Web application. They have been registered as valid endpoints for the above client ids using the Google configuration page.
http://localhost:9000/Callback
http://localhost:9001/Callback

Google Authorization URL
https://accounts.google.com/o/oauth2/auth

Google Token URL
https://accounts.google.com/o/oauth2/token

Google Toke INFO URL
https://www.googleapis.com/oauth2/v3/tokeninfo

Use Case 1: Multiple DD-Nodes with browser front-end

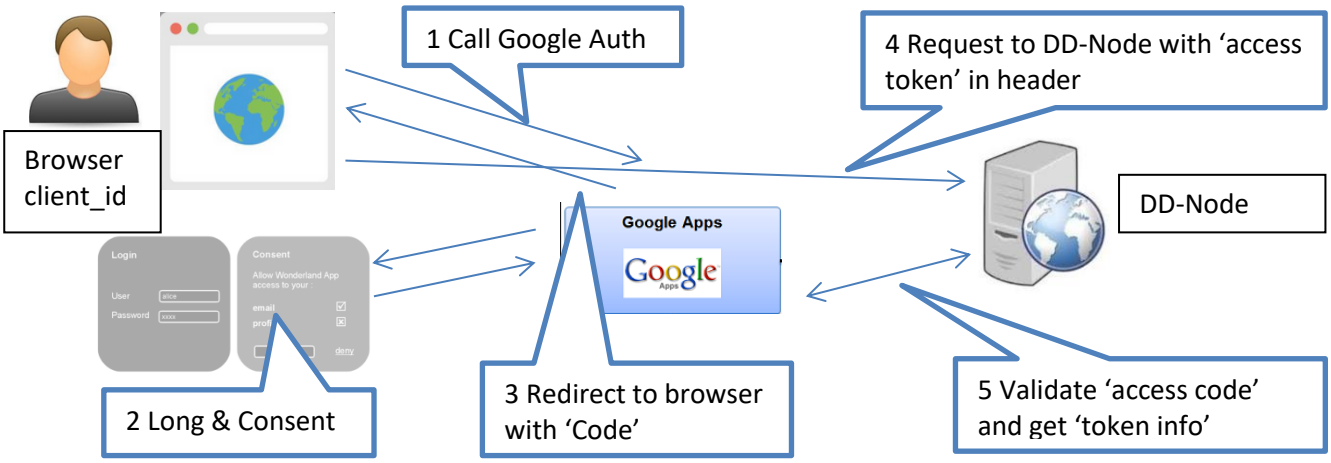
In this use case an actual person is querying multiple DD-Nodes through a web browser. In this case the web browser takes care of the authentication steps. The browser queries the authentication endpoint and retrieves the retrieved 'access_token'. This token is validated and added as an Authorization header in the requests to the DD-Nodes as follows:

Authorization = Bearer <access_token>

This allows each DD-Node to retrieve information about the user without the user having to login to each DD-Node separately. This form of authentication is supported by OpenId but is considered the least secure. Each DD-Node will validate the 'access token' and use it to obtain the 'user info'.

Authentication Steps:

1. User calls the Google Auth endpoint with response_type=id_token token
2. From the Google Auth endpoint the user is directed to the Login & consent pages
3. After successful login Google redirects user to the redirect url of the browser application.
4. The browser application retrieves the 'access code' and inserts it into the header of the request to the DD-Node.
5. The DD-Node validates the 'access token' and uses it to retrieve the user information from the Google Token Info endpoint.



Example Requests

1. Call Google Auth endpoint:

   https://accounts.google.com/o/oauth2/auth?client_id=<client1_id>&redirect_uri=http%3A%2F%2Flocalhost%3A9000%2FCallback&response_type=id_token%20token&scope=openid%20email&nonce=<someNonce>

2. Redirect to Login & Consent page.
3. Redirect to web browser with 'access_token' and 'id_token':

   id_token:
   JWT Header :
   {
   "alg":"RS256",
   "kid":"<someKidId>"
   }
   JWT Body :
   {
   "azp":"<client1_id>",
   "aud":"<client1_id>",
   "sub":"<someSubId>",
   "email":"<someone@gmail.com>",
   "email_verified":true,
   "at_hash":"<someHashCode>",
   "nonce":"<someNonce>",
   "exp":<someExp>,
   "iss":"accounts.google.com",
   "jti":"<someJti>",
   "iat":<someIat>
   }
   JWT Signature:
   {
   <left out>
   }

   access_token:

<someAccessToken>

4. Validate the ‘id_token’ for correctness
5. Web browser inserts ‘access_token’ into header of DD-Node request
6. DD-Node uses ‘access_token’ to call Google Token Info endpoint and retrieves the user email address.

Use Case 2: Multiple DD-Nodes with a DD Web application

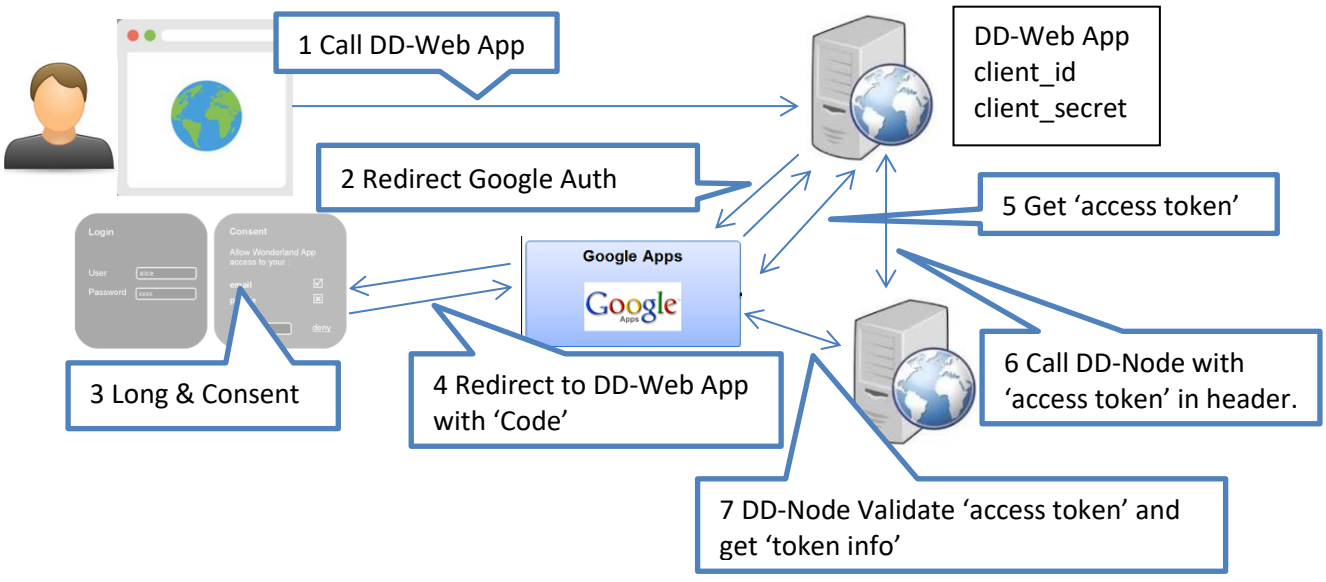
In this use case an actual person is querying multiple DD-Nodes through the DD Web Application. In this case the web application takes care of the authentication steps. The web application calls the Google endpoints; auth and token. The retrieved ‘access_token’ is inserted as an Authorization header in the requests to the DD-Nodes as follows:

Authorization = Bearer <access_token>

This allows each DD-Node to retrieve information about the user without the user having to login to each DD-Node separately. This form of authentication is more secure then the example in use case 2 as the web application is able to retrieve the ‘access_token’ using the ‘client1_id’ and ‘client_secret’. The Web Application can store these credentials securely opposed to the Web browser of use case 2. Each DD-Node will validate the ‘access token’ and use it to obtain the ‘user info’.

Authentication Steps:

1. User calls the DD-Web Application endpoint with response_type=code.
2. DD-Web Application redirects call to Google Auth endpoint with response_type=code
3. From the Google Auth endpoint the user is directed to the Login & consent pages
4. After successful login Google redirects user to the redirect url of the DD-Web Application
5. The DD-Web Application retrieves the ‘code’ from from the response and uses this to obtain an ‘access token’ from Googles Token endpoint.
6. The DD-Web Application inserts the ‘access token’ into the request header to the DD-Nodes.
7. The DD-Node validates the ‘access token’ and uses it to retrieve the user information from the Google Token Info endpoint.



Example Requests

1. Call DD-Web Application: http://localhost:9000/ddwebapp/query
2. Call Google Auth endpoint:

   https://accounts.google.com/o/oauth2/auth?client_id=<client1_id>&redirect_uri=http%3A%2F%2Flocalhost%3A9000%2FCallback&response_type=code&scope=openid%20email&nonce=<someNonce>

3. Redirect to Login & Consent page.
4. Redirect to DD-Web Application with 'code':
   http://localhost:9000/Callback?code=<someCode>
5. Call Google Token endpoint:
   POST https://accounts.google.com/o/oauth2/token
   Content-Type: application/x-www-form-urlencoded
   code=<someCode>&client1_id=<client1_id>&client_secret=<client1_secret>&redirect_uri=http%3A%2F%2Flocalhost%3A9000%2FCallback&grant_type=authorization_code

   Response:
   id_token:
   JWT Header :
   {
   "alg":"RS256",
   "kid":"<someKidId>"
   }
   JWT Body :
   {
   "azp":"<client1_id>",
   "aud":"<client1_id>",
   "sub":"<someSubId>",
   "email":"<someone@gmail.com>",
   "email_verified":true,
   "at_hash":" <someHashCode>",
   "nonce":"<someNonce>",
   "exp": <someExp>,
   "iss":"accounts.google.com",
   "iat": <someIat>
   }
   JWT Signature:
   {
   <left out>
   }

   access_token:
   <someAccessToken>

6. Validate the 'id_token' for correctness
7. DD Web Application inserts 'access_token' into header of DD-Node request
8. DD-Node uses 'access_token' to call Google Token Info endpoint and retrieves the user email address.

Use Case 3: Query DD-Node without user input

In this use case there is no actual person. An automated process is trying the query the DD-Nodes directly or through the DD-Web application. In the above OpenID calls a user is always required to perform a login action. For this use case the only OpenID option available is to connect using a refresh_token. Other authentication options for connecting without a user are available in the OAuth2 protocol.

OpenID Refresh Token

To retrieve a refresh token a one-time manual authentication is required. Afterwards the access_token can be retrieved indefinitely with the refresh_token. The newly obtained access_token can be passed in the header of the data request to the DD-Nodes. This approach works for automated clients accessing a DD-Node directly or through a DD –Web Application.

OAuth2 Authorization

If you do not require OpenID then another option would be to use the OAuth2 protocol with grant types 'Password Grant' or 'Client Credential' instead.

Password Grant: Here a 'username' and 'password' are passed in the request to retrieve a token. If the client was issued a secret, the following parameters are also required a 'client1_id' and 'client_secret'.

Client Credential: Here a 'client1_id' and 'client_secret' are passed in the request to retrieve a token.

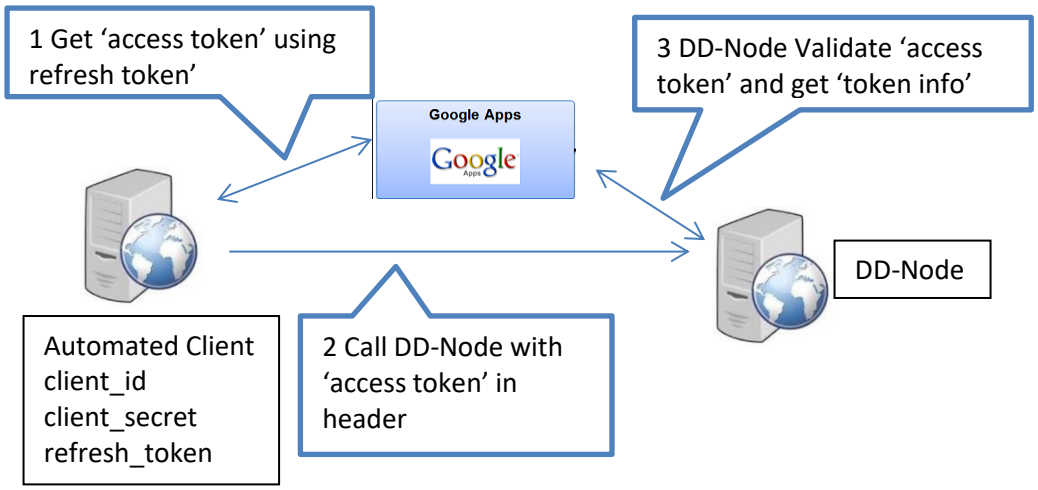
Authentication Steps (OpenID):

One-time steps

1. Call to Google Auth endpoint with response_type=code and access_type=offline.
2. From the Google Auth endpoint the user is directed to the Login & consent pages
3. Call the Google Token endpoint with the code value and grant_type=authorization_code
4. From the response extract and store the refresh_token.

Automated login steps

1. Call the Google Token endpoint using the 'refresh token', 'client1_id' and 'client_secret'. Redirect the response back to the caller.
2. From the response retrieve the 'access token'. Insert 'access token' into header of to the DD-Node.
3. The DD-Node validates the 'access token' and uses it to retrieve the user information from the Google Token Info endpoint.



Example Requests

One-time steps

1. Call Google Auth endpoint:
   https://accounts.google.com/o/oauth2/auth?client_id=<client1_id>&redirect_uri=http%3A%2F%2Flocalhost%3A9000%2FCallback&response_type=code&scope=openid%20email&access_type=offline&nonce=<someNonce>

2. Redirect to Login & Consent page. Login with user **<client1_e-mail>**
3. Redirect to caller application with 'code':
   http://localhost:9000/Callback?code=<someCode>

4. Call Google Token endpoint:
   POST https://accounts.google.com/o/oauth2/token
   Content-Type: application/x-www-form-urlencoded
   code=<someCode>&client1_id=<client1_id>&client_secret=<client1_secret>&redirect_uri=http%3A%2F%2Flocalhost%3A9000%2FCallback&grant_type=authorization_code

   Response:
   id_token:
   JWT Header :

```
{
   "alg":"RS256",
   "kid":"<someKidId>"
}
```

   JWT Body :

```
 {
   "azp":"<client1_id>",
   "aud":"<client1_id>",
   "sub":" <someSubId>",
   "email":"<client1_e-mail>",
   "email_verified":true,
   "at_hash":"<someHashCode>",
   "nonce":"<someNonce>",
   "exp": <someExp>,
   "iss":"accounts.google.com",
   "iat": <someIat>
```

}
JWT Signature:
{
<left out>
}

access_token:
<someAccessToken>

refresh_token:
<someRefreshToken>

Now we have the refresh token. With this token we can make repeated calls to the token endpoint without having to re-authenticate ourselves.

Automated login steps

1. Call Google Token endpoint with 'refresh token':
   POST https://accounts.google.com/o/oauth2/token
   Content-Type: application/x-www-form-urlencoded
   refresh_token=<someCode&client1_id=<client1_id>&client_secret=<client1_secret>&redirect_uri=http%3A%2F%2Flocalhost%3A9000%2FCallback&grant_type=refresh_token

   Reponse:
   id_token:
   JWT Header :
   {
   "alg":"RS256",
   "kid":"<someKidId>"
   }
   JWT Body :
   {
   "azp":"<client1_id>",
   "aud":"<client1_id>",
   "sub":" <someSubId>",
   "email":"<client1_e-mail>",
   "email_verified":true,
   "at_hash":"<someHashCode>",
   "nonce":"<someNonce>",
   "exp": <someExp>,
   "iss":"accounts.google.com",
   "iat": <someIat>
   }
   JWT Signature:
   {
   <left out>
   }

   access_token:
   <someAccessToken>

2. Caller application inserts ‘access_token’ into header of DD-Node request
3. DD-Node uses ‘access_token’ to call Google Token Info endpoint and retrieves the user email address.

Authentication Steps (OAuth2):

1. Call the Authentication Token endpoint using ‘grant_type’ with values ‘password’ or ‘client_credentials’. Further parameters are:
   ‘username’ and ‘password’ for grant _type=password, ‘client1_id’ and ‘client_secret’.
2. From the response retrieve the ‘access token’. Insert ‘access token’ into header of to the DD-Node.

Validating ID_TOKEN

Each time an ID_Token is returned, it must be checked for validity.

An ID_TOKEN contains the following information:
Header:

- alg : Algorithm
- kid :

Body:

- azp : Audience this ID_Token is intended for. This must contain the client1_id of <client1_e-mail>
- aud : Audience this ID_Token is intended for. This must contain the client1_id of <client1_e-mail>
- sub : A unique identifier for the end user issued by issuer
- email :Email of authenticated user
- email_verified:
- at_hash: Access Token hash value
- nonce : String value used to associate a Client session with an ID Token, and to mitigate replay attacks.
- exp : After what time the ID_Token should not accepted
- iss : Issuing authority
- iat : Time that token is issued

1. Required parameters check: iss, sub, aud, exp and iat are required parameters in the id_token

   Validation steps:
2. Check that the ID token’s crypto algorithm matches the one which the client has registered with the OpenID provider;
3. Validate the ID token signature; Public key can be found under endpoint:

   https://www.googleapis.com/oauth2/v1/certs

   Use the public key and ‘alg’ to encode token ‘header’.’body’. The output should be equal to the signature provided in the token.

4. Validate the ID token claims:
    - expiration: The current date/time must be before the expiration date/time listed in the exp claim (which is a Unix timestamp). If not, the request must be rejected.
    - issuer — does the token originate from the expected IdP (accounts.google.com)?
    - audience — is the token intended for me?
    - timestamps — is the token within its validity window?
    - nonce — if set, is it the same as the one in my request?