OpenID-Connect: a quick intro.

Paul Millar
(on behalf of the dCache team)

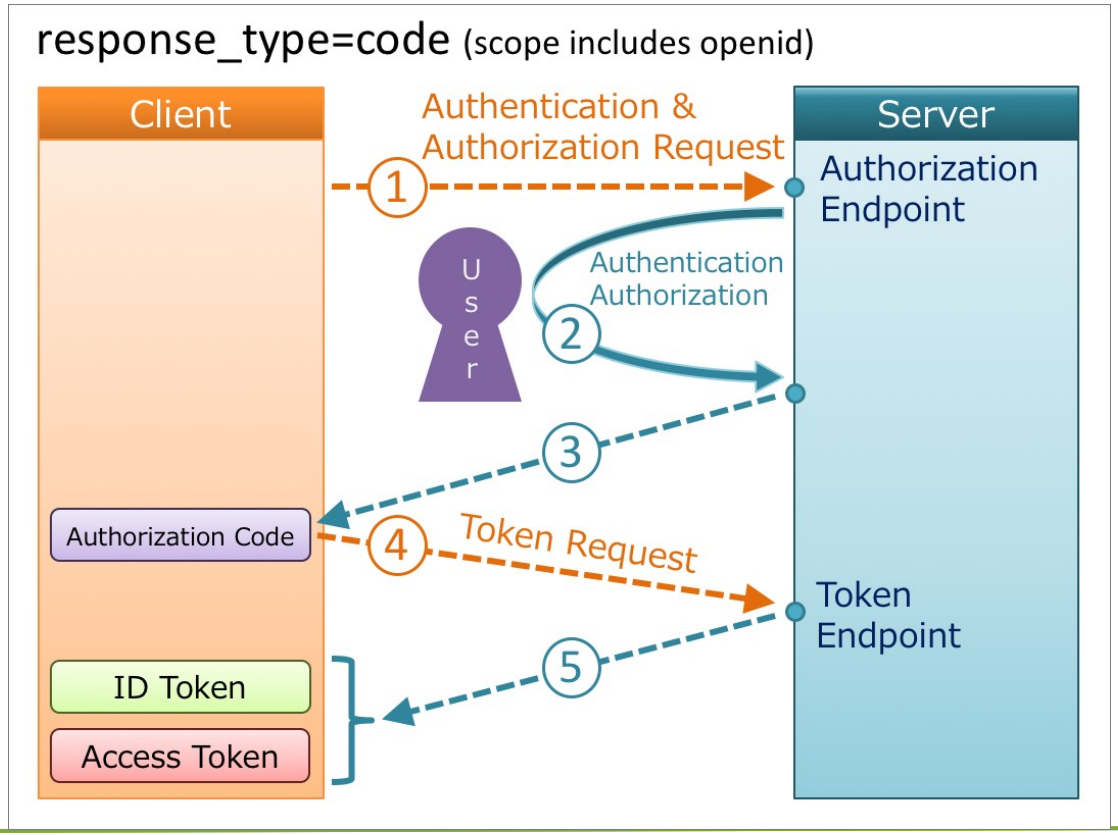
ESCAPE oidc workshop 2020; 2020-01-13
https://indico.in2p3.fr/event/20331/

What is OpenID-Connect

- **Single sign-on**: use the same authentication for multiple services.
- Build on top of **OAuth2** (an authorisation framework).
- Somewhat similar to SAML, but with **JSON** instead of XML.
  - OIDC is more flexible than SAML.
  - Like SAML, is web-browser heavy.
  - Unlike SAML, OIDC also works for non-web clients.

How it works: code flow

For example:
a scientific
web-portal



OIDC Provider (OP)

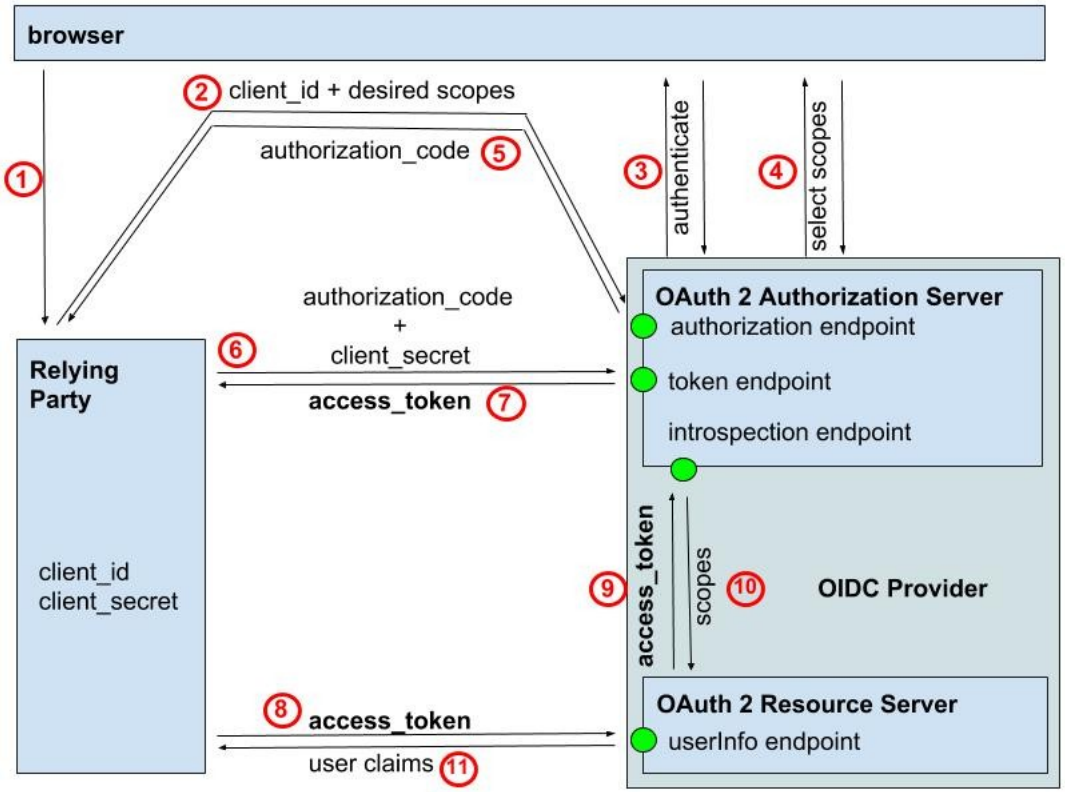
IAM

Google

From https://medium.com/@darutk/diagrams-of-all-the-openid-connect-flows-6968e3990660

How does it work: access token

For example:
a scientific
web-portal



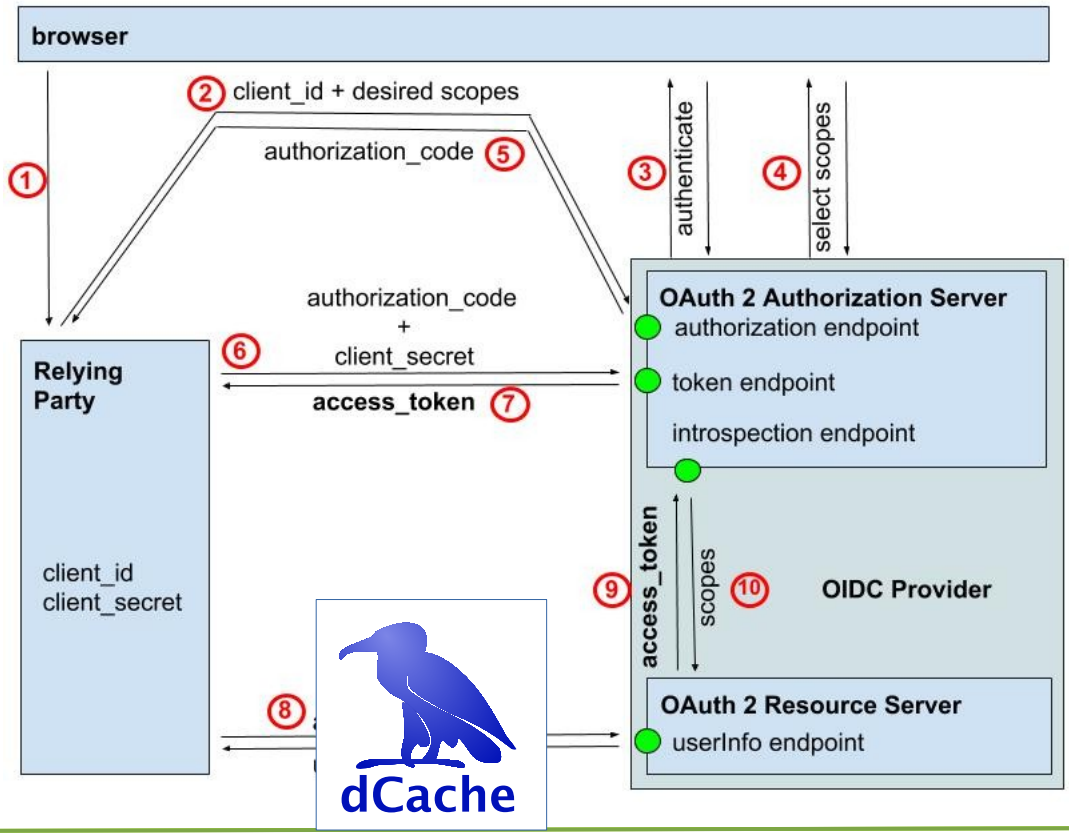
IAM

Google

From: https://gist.github.com/arax/1632db374fc203d726a9895854e26176

How does it work: access token

For example:
a scientific
web-portal



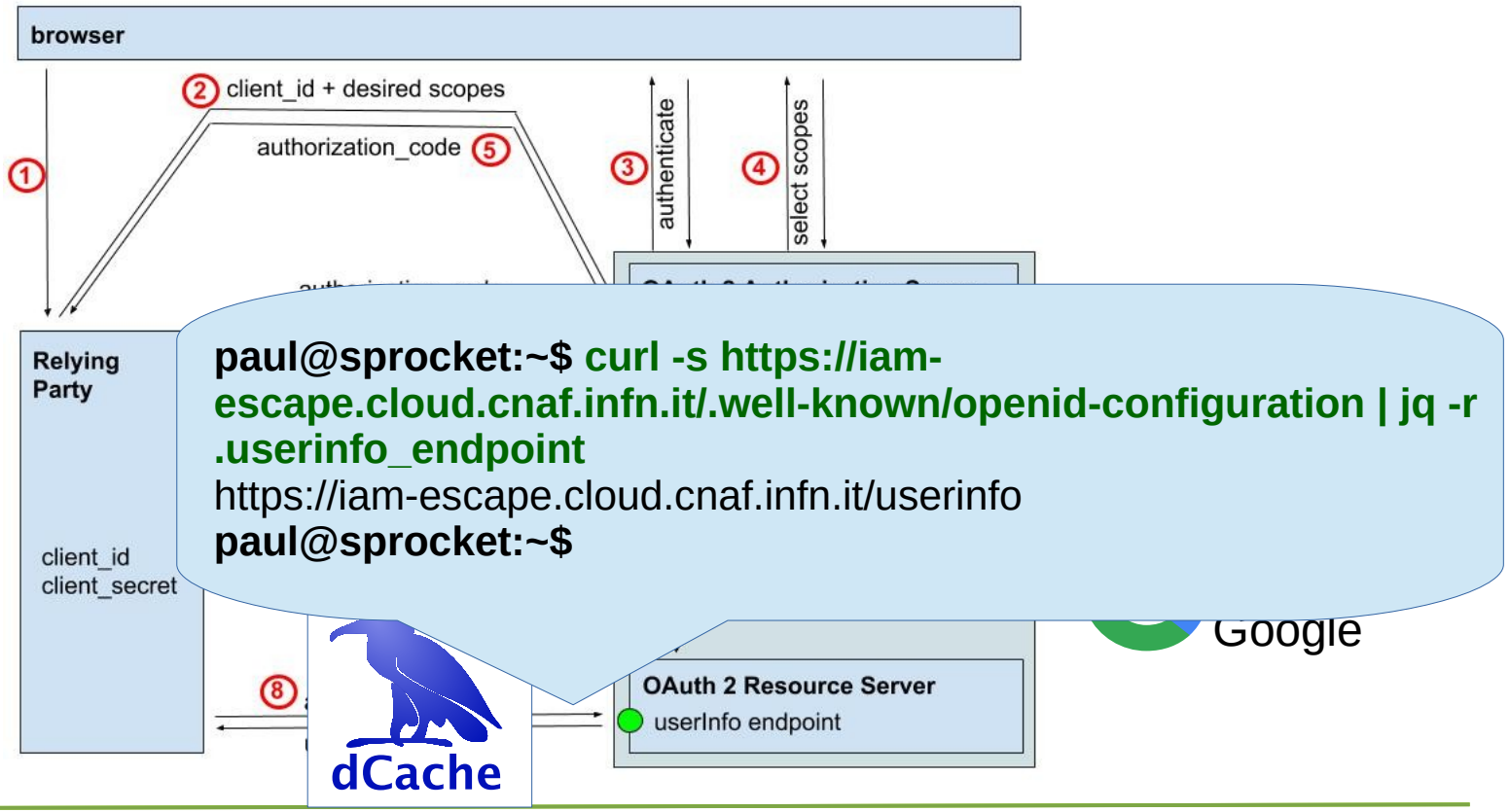
IAM

Google

From: https://gist.github.com/arax/1632db374fc203d726a9895854e26176

How does it work: access token

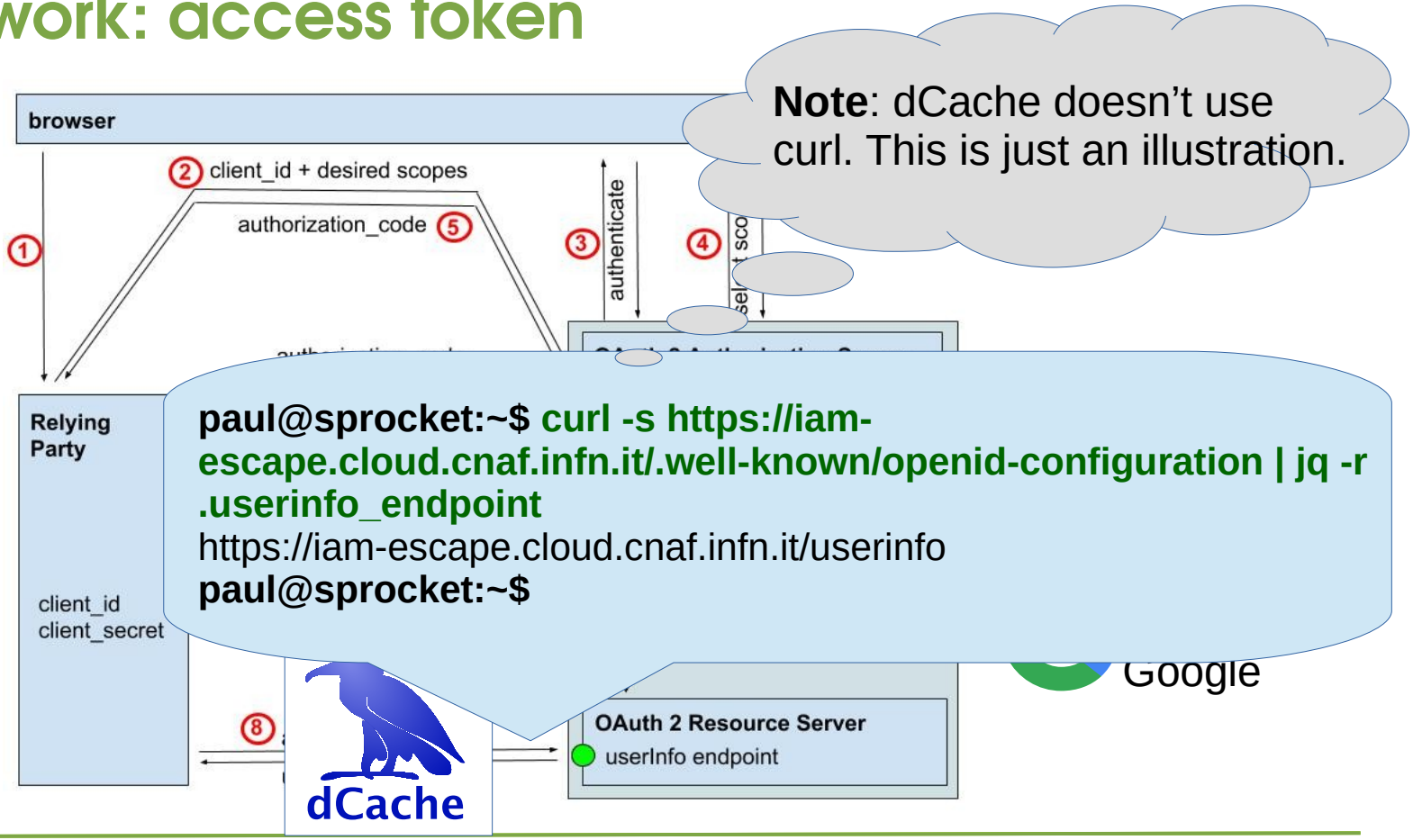
For example:
a scientific
web-portal



From: https://gist.github.com/arax/1632db374fc203d726a9895854e26176

How does it work: access token

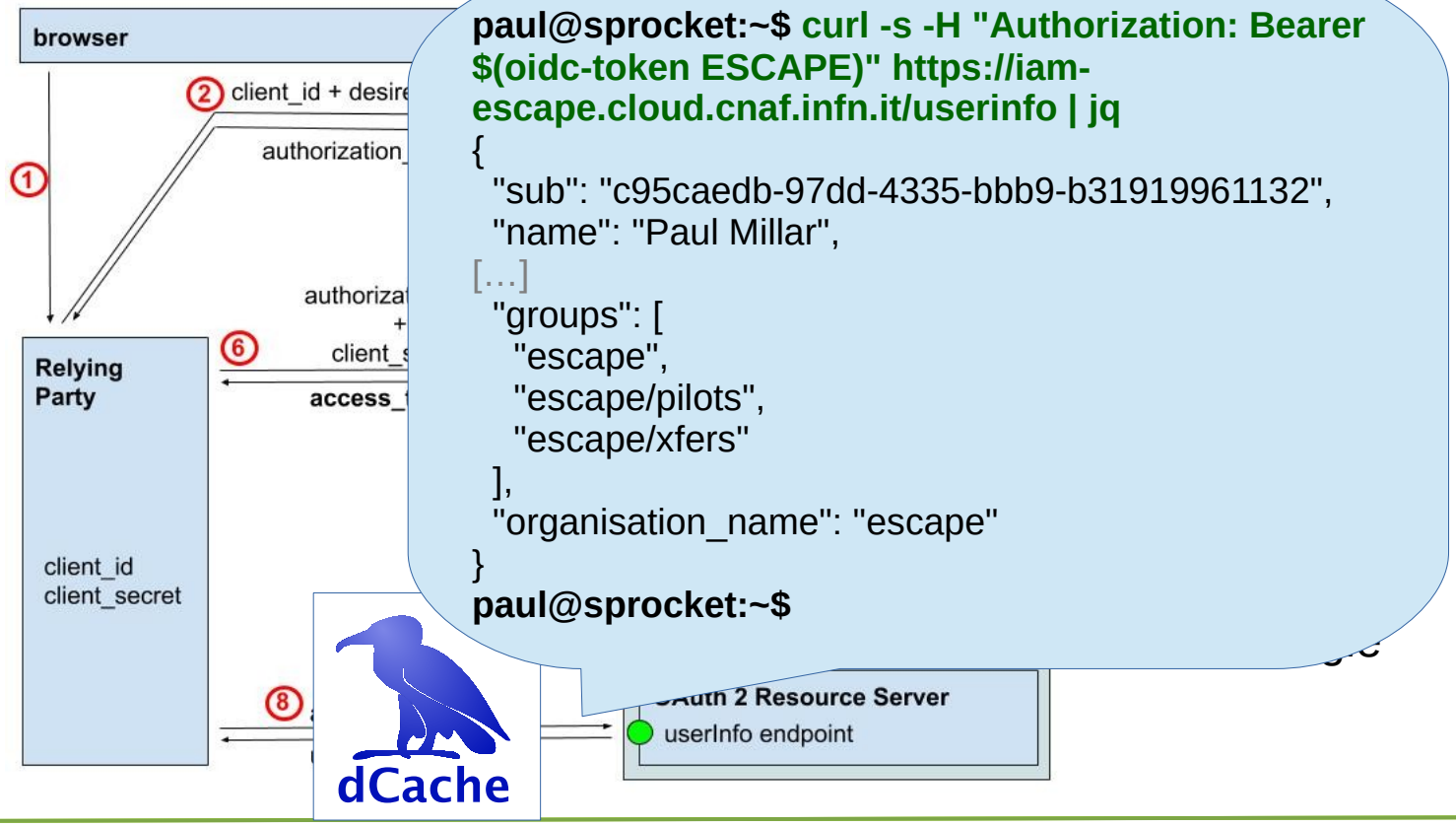
For example: a scientific web-portal



From: https://gist.github.com/arax/1632db374fc203d726a9895854e26176

How does it work: access token

For example:
a scientific
web-portal




```
paul@sprocket:~$ curl -s -H "Authorization: Bearer
$(oidc-token ESCAPE)" https://iam-
escape.cloud.cnaf.infn.it/userinfo | jq
{
  "sub": "c95caedb-97dd-4335-bbb9-b31919961132",
  "name": "Paul Millar",
[…]
  "groups": [
    "escape",
    "escape/pilots",
    "escape/xfers"
  ],
  "organisation_name": "escape"
}
paul@sprocket:~$
```

From: https://gist.github.com/arax/1632db374fc203d726a9895854e26176

Support in dCache: the oidc plugin

- The oidc plugin is an **auth** phase gPlazma plugin
- When the plugin receiving a bearer token:
  - It reads the **service description** to discover the user-info endpoint.
  - It tries the **user-info endpoint**, using the access token, to identify the user
  - The JSON response contains information about the user
- Supports **multiple OPs**. Sends access token to all OPs, unless access token is a JWT.
- Adds **principals**: OidcSubject, FullName, EmailAddress, OpenIdGroup, LoA

dCache configuration defaults

```
gplazma.oidc.http.slow-threshold = 2
gplazma.oidc.http.slow-threshold.unit = SECONDS
gplazma.oidc.discovery-cache = 1
gplazma.oidc.discovery-cache.unit = HOURS
gplazma.oidc.concurrent-requests = 20
gplazma.oidc.http.total-concurrent-requests = ${gplazma.oidc.concurrent-requests}
gplazma.oidc.http.per-route-concurrent-requests = 10
gplazma.oidc.http.timeout = 30
gplazma.oidc.http.timeout.unit = SECONDS
gplazma.oidc.access-token-cache.size = 1000
gplazma.oidc.access-token-cache.refresh = 100
gplazma.oidc.access-token-cache.refresh.unit = SECONDS
gplazma.oidc.access-token-cache.expire = 120
gplazma.oidc.access-token-cache.expire.unit = SECONDS
```

dCache configuration defaults #2

Map/prefix configuration option:

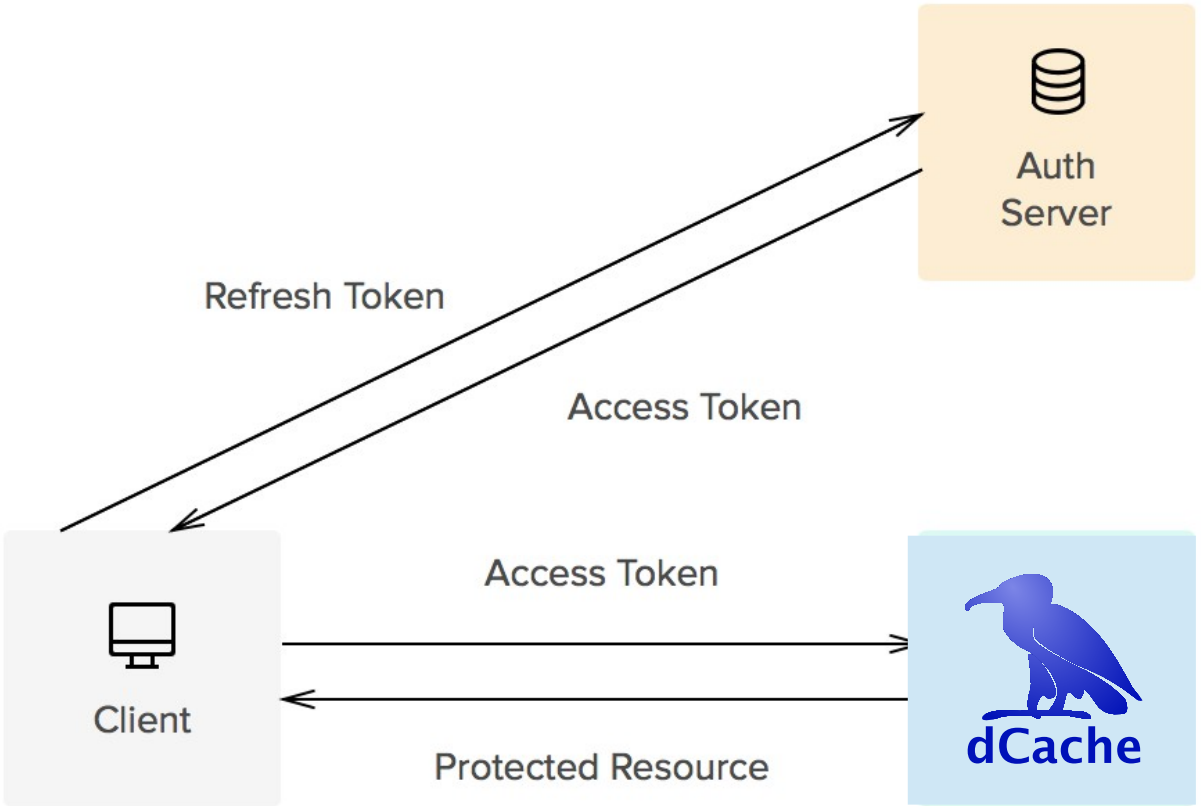
gplazma.oidc.provider!<ALIAS> = <OP>

For example:

gplazma.oidc.provider!ESCAPE = https://iam-escape.cloud.cnaf.infn.it/

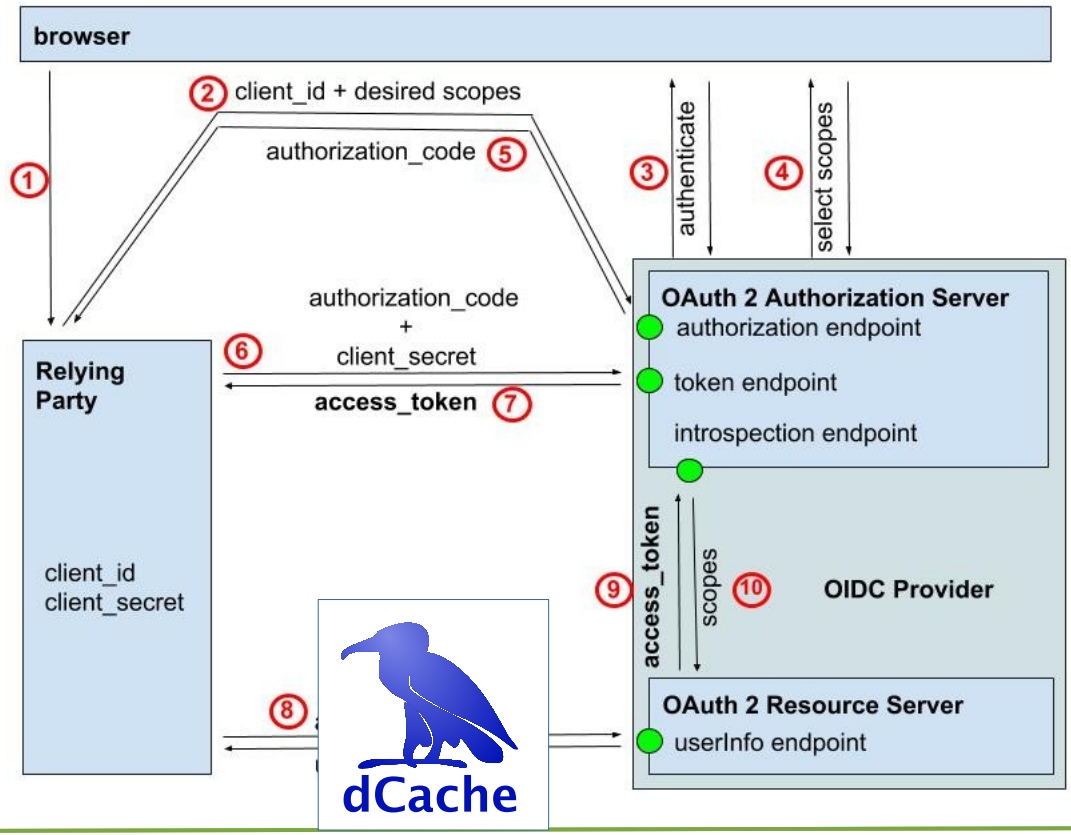
How does it work: refresh token

- Access tokens are (relatively) short-lived: **they expire**!
- Once expired, the client needs to get a **new access token**.
- A **refresh token** lets a client fetch a fresh access token without user re-authenticating.
- This is how **oidc-agent** works



More on access tokens

For example:
a scientific
web-portal

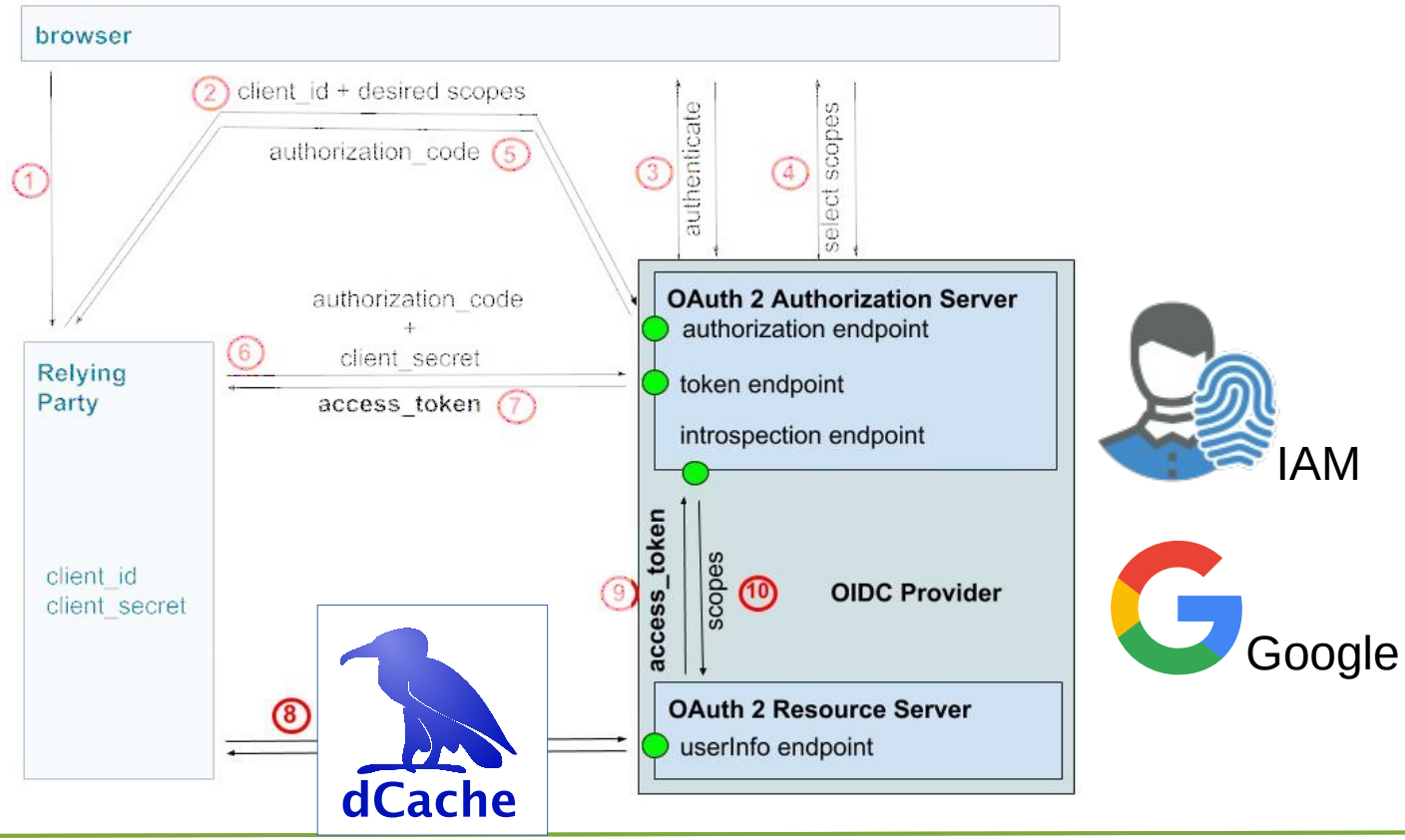


IAM

Google

From: https://gist.github.com/arax/1632db374fc203d726a9895854e26176

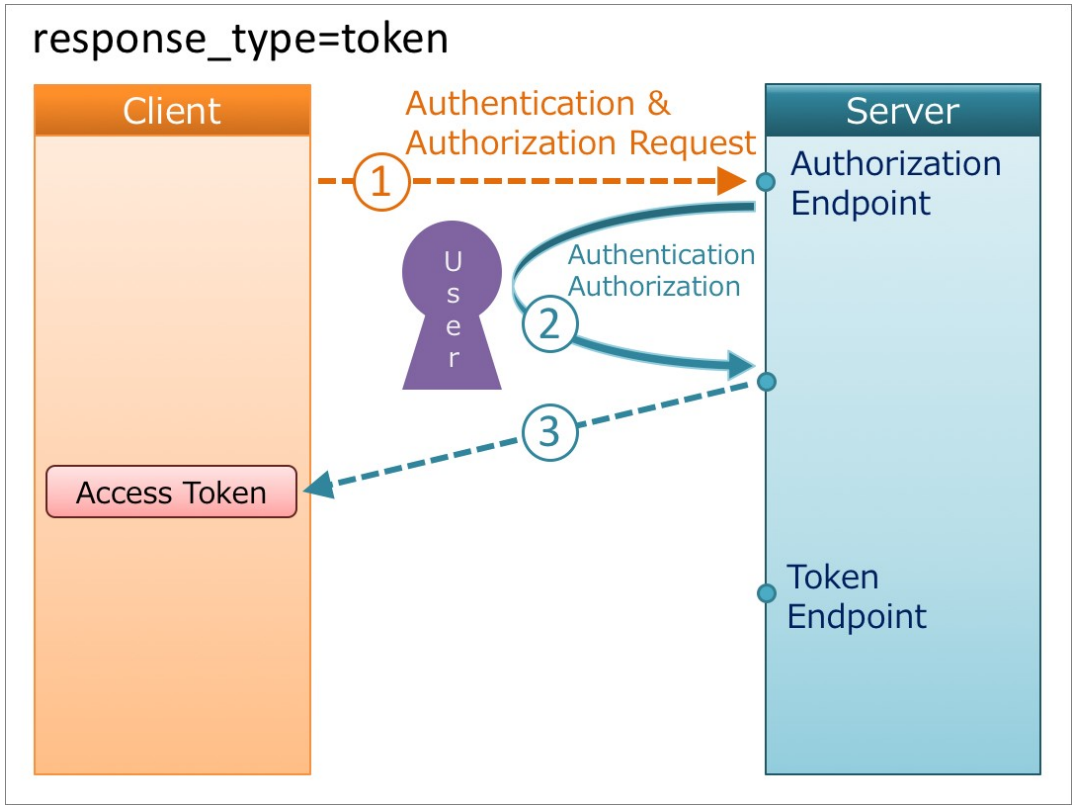
More on access tokens



From: https://gist.github.com/arax/1632db374fc203d726a9895854e26176

dCacheView with implicit flow

dCacheView



OAuth2 Provider (OP)

IAM

Google

From https://medium.com/@darutk/diagrams-of-all-the-openid-connect-flows-6968e3990660

Support in dCache: oidc in dCacheView

- dCacheView uses **implicit flow**.

  The **web-browser** ends up with the access token.

- The token is used to authenticate all **frontend operations**, and **WebDAV transfers**.

- Requires a **client ID**, which dCache/frontend knows and is supplied to dCacheView using the static information:

```
frontend.static!dcache-view.oidc-authz-endpoint-list=https://iam-escape.cloud.cnaf.infn.it/authorize
frontend.static!dcache-view.oidc-client-id-list=<ID>
frontend.static!dcache-view.oidc-provider-name-list=ESCAPE
```

Support in dCache: oidc for WebDAV TPC

- Third-party copy (TPC) may be a long-running activity, especially if queued.
- WebDAV normally has only an **access token**.
- In TPC, the WebDAV door can request a **refresh token**.
- Just before the transfer is due to start, the **pool requests** an access token for the remote machine using the refresh token.
- Experimental feature, not currently used.
- Configuration:

```
webdav.oidc.client.id!ESCAPE = <ID>
webdav.oidc.client.secret!ESCAPE = <SECRET>
```

Conclusions

- OpenID-Connect identifies someone using a **bearer token** called an "access token".

  With the access token, anyone can discover information about the user.

- dCache supports authenticating **WebDAV** and **frontend** requests with an access token.
  - gPlazma calls out to OP when discovering person's identity.
  - Results are cached.

Thanks for listening!