OpenID Connect as SSO solution: strengths and weaknesses

Álvaro Iradier Muro

Agenda

OpenID Connect as SSO solution: strengths and weaknesses

- Single Sign-on basics
- What is OpenID Connect (OIDC)
- Typical flow examples
- OIDC comparison
  - vs others (LDAP, Kerberos, cookies, …)
  - vs SAML
  - vs OpenID 2.0
  - vs Oauth2
  - vs JWT
- Why use OpenID Connect?
- Testing and debugging OIDC
- Creating an OIDC client
- Real world example: dealing with complex authentication scenarios
- Other caveats
- The good, the bad and the ugly
- Q&A

Single Sign-on basics

Allow users use a single set of login credentials for multiple applications.

Applications can be related, but independent.

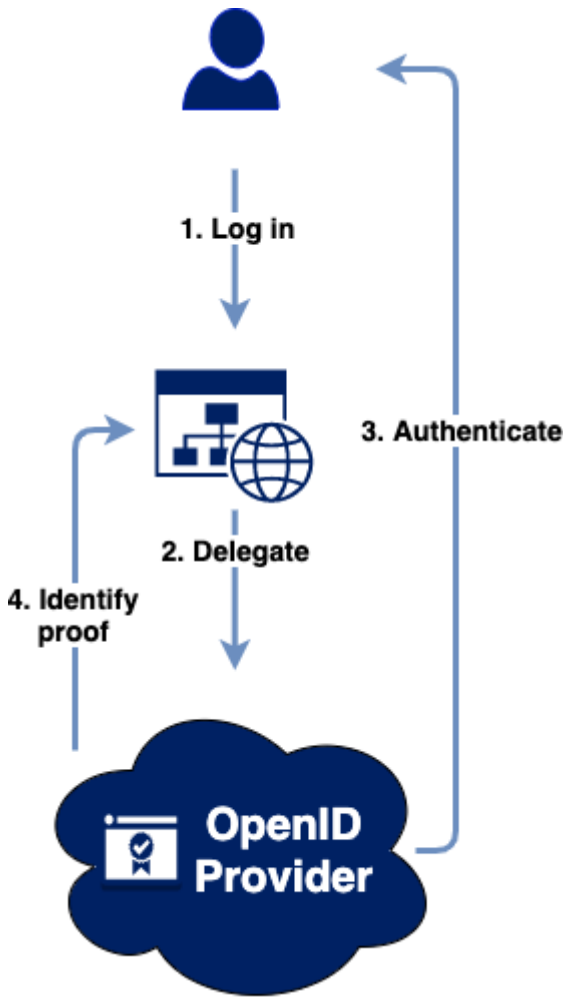
Why?

- Remember less passwords, and no need to reenter them on every app.
- Security: lessen chances of phising, reduce password fatigue.
- Reduce password issues for IT help desks.

Why not?

- Higher risk for exposed credentials. Increase focus on protection (i.e. MFA).
- Criticality – single point of failure - of authentication system.

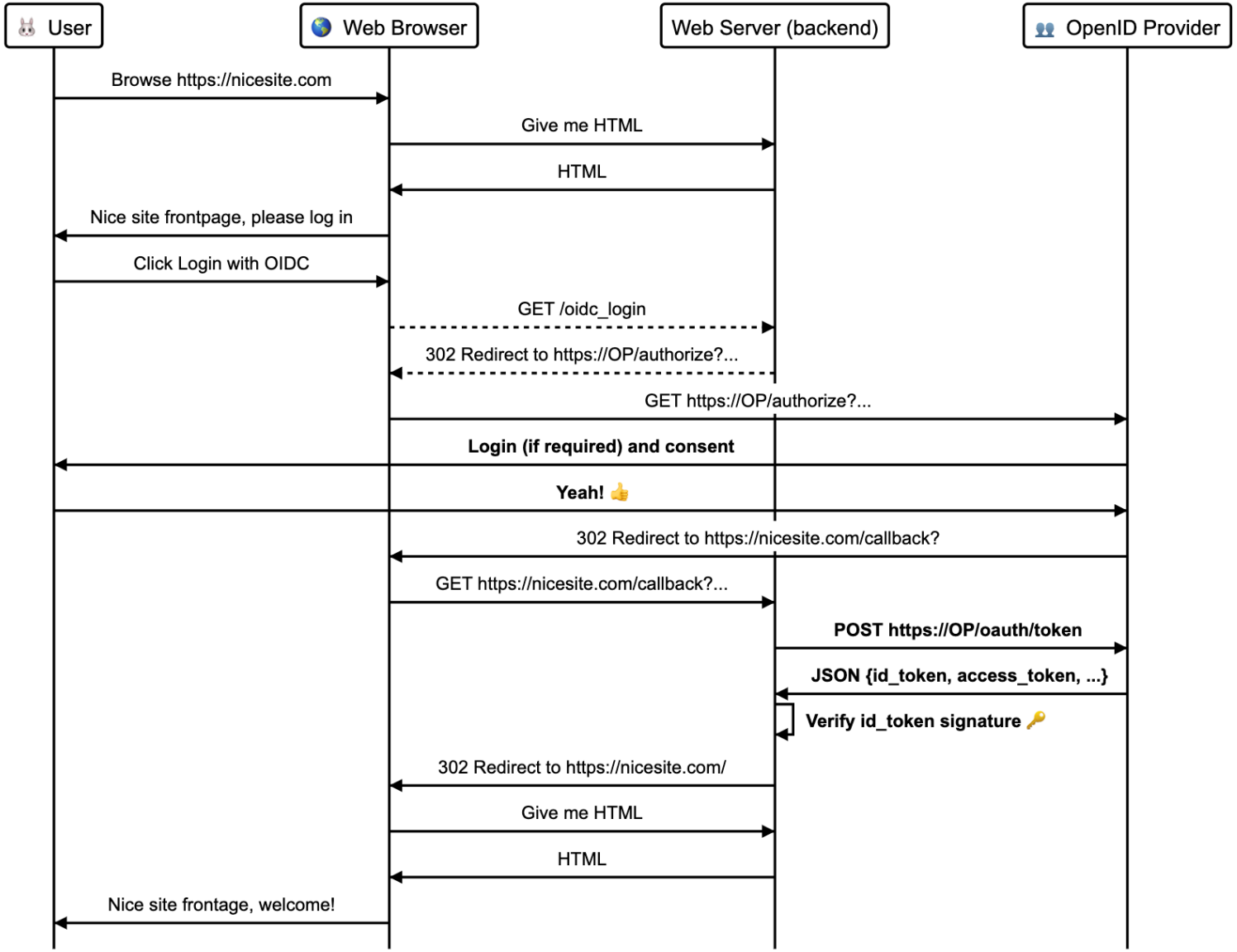
What is OpenID Connect (OIDC)



- Identity layer on top of OAuth 2.0 protocol.
- Verify identity, delegated to an Authorization Server.
- Obtain End-User basic profile information.
- Interoperable REST/JSON manner.
- Web, Javascript, mobile apps, etc.
- https://openid.net/connect/

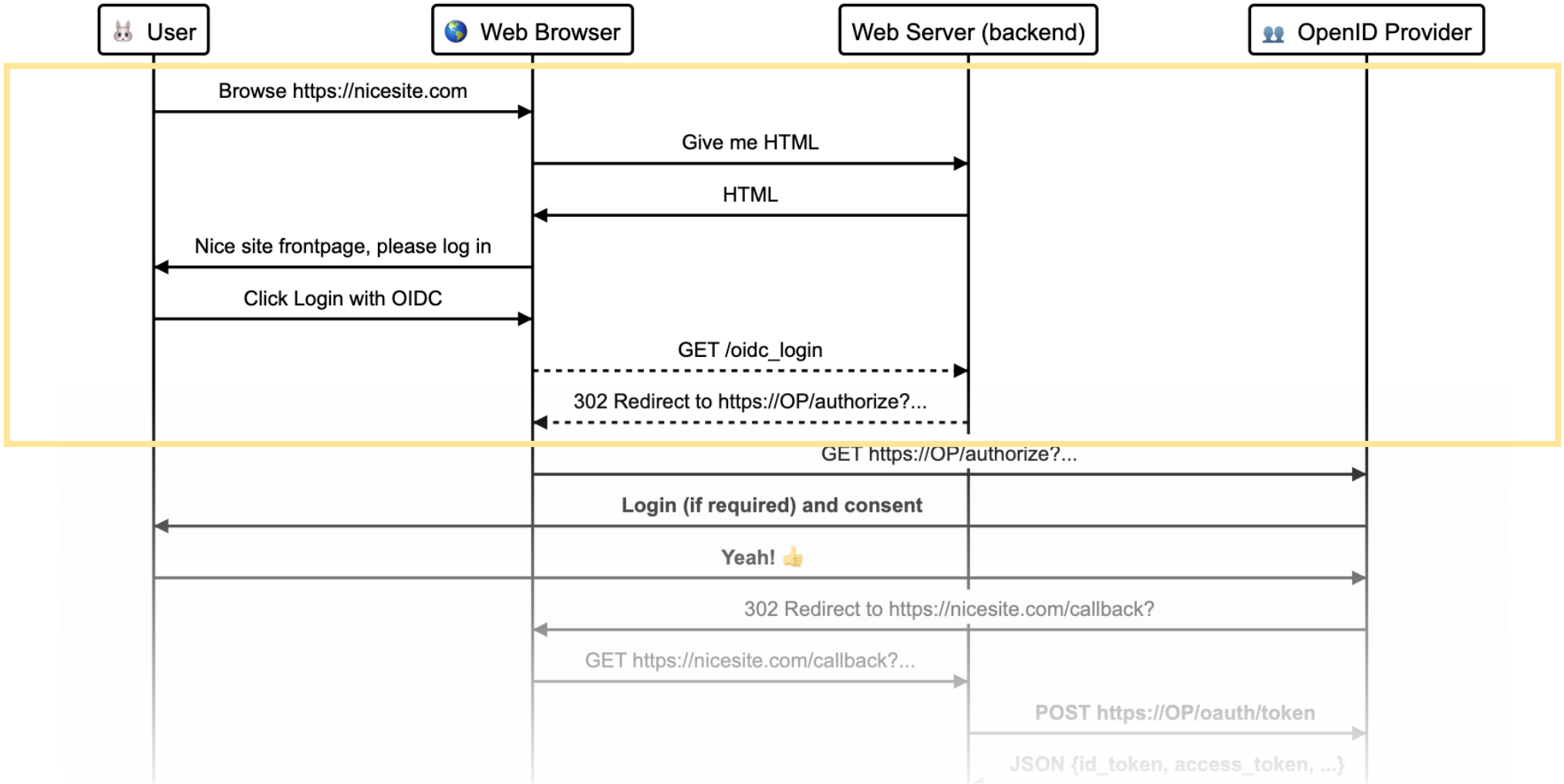
- **OpenID Provider (OP)**: auth as a service.
  - Or Identity Provider (IDP)
- **Relying Party**: app that outsources user authentication to an OP.

Authorization code grant flow



Authorization code grant flow

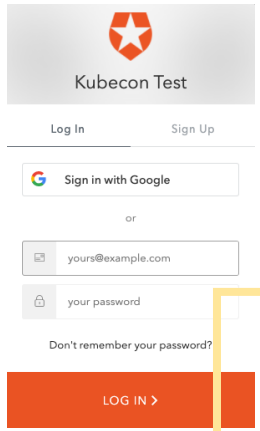
User requests login with OIDC

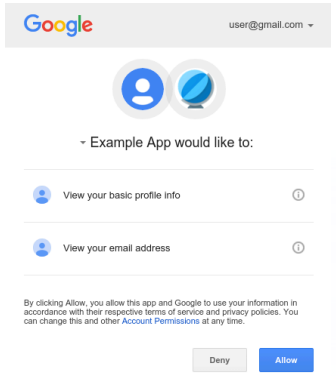


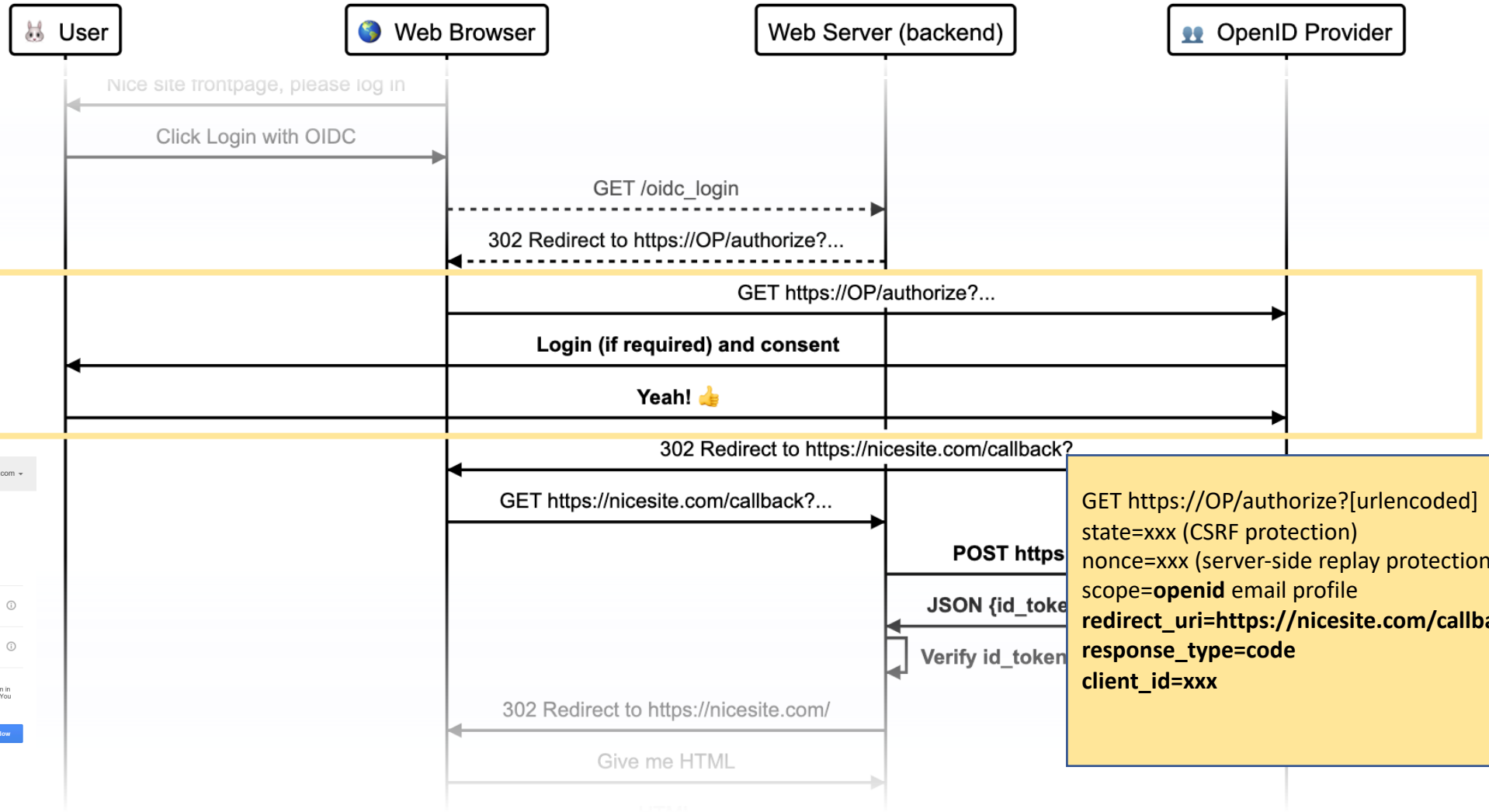
Authorization code grant flow

KubeCon CloudNativeCon Europe 2020 Virtual

OpenID provider authenticates user



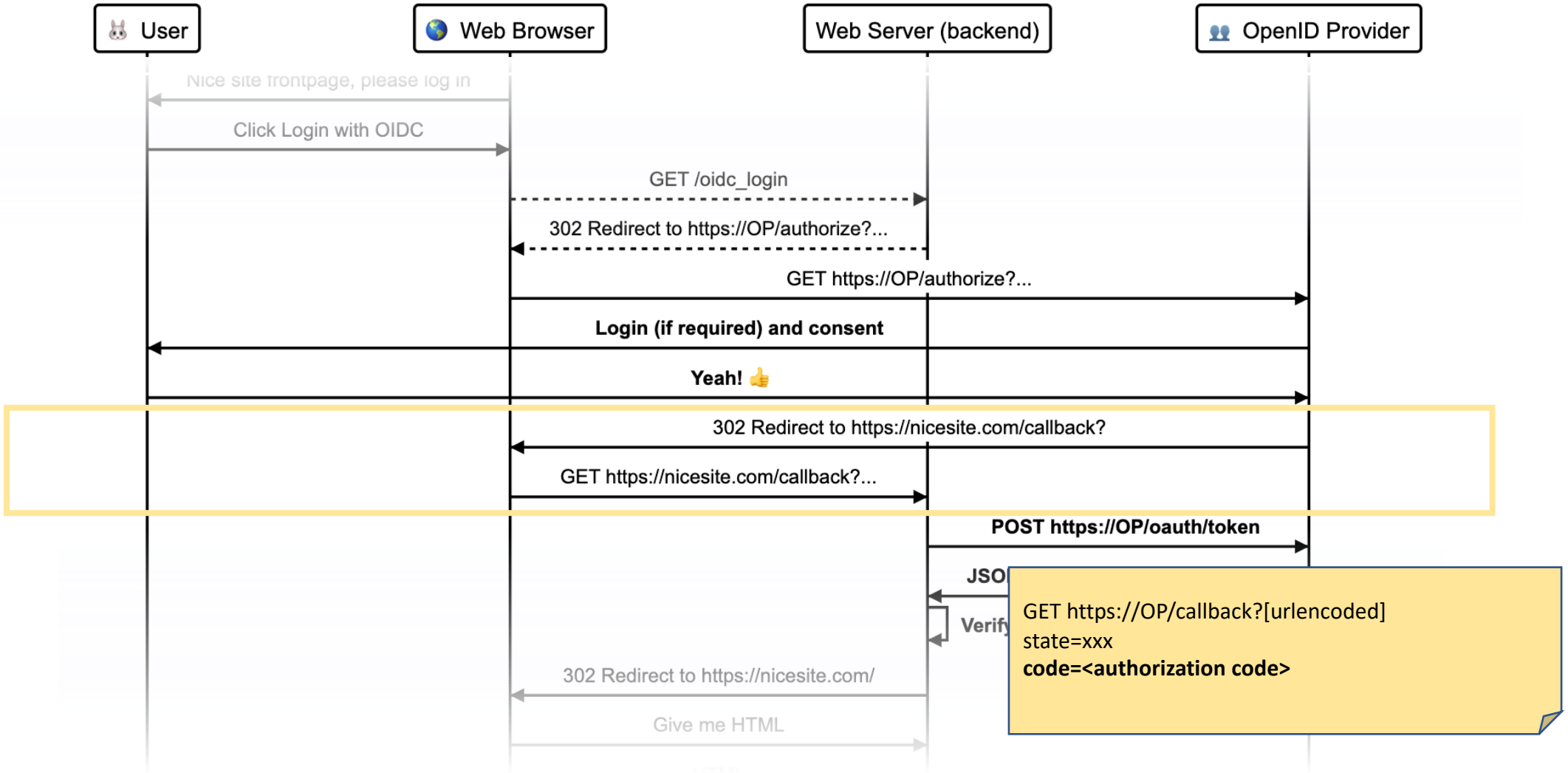




GET https://OP/authorize?[urlencoded]
state=xxx (CSRF protection)
nonce=xxx (server-side replay protection)
scope=**openid** email profile
redirect_uri=https://nicesite.com/callback
response_type=code
client_id=xxx

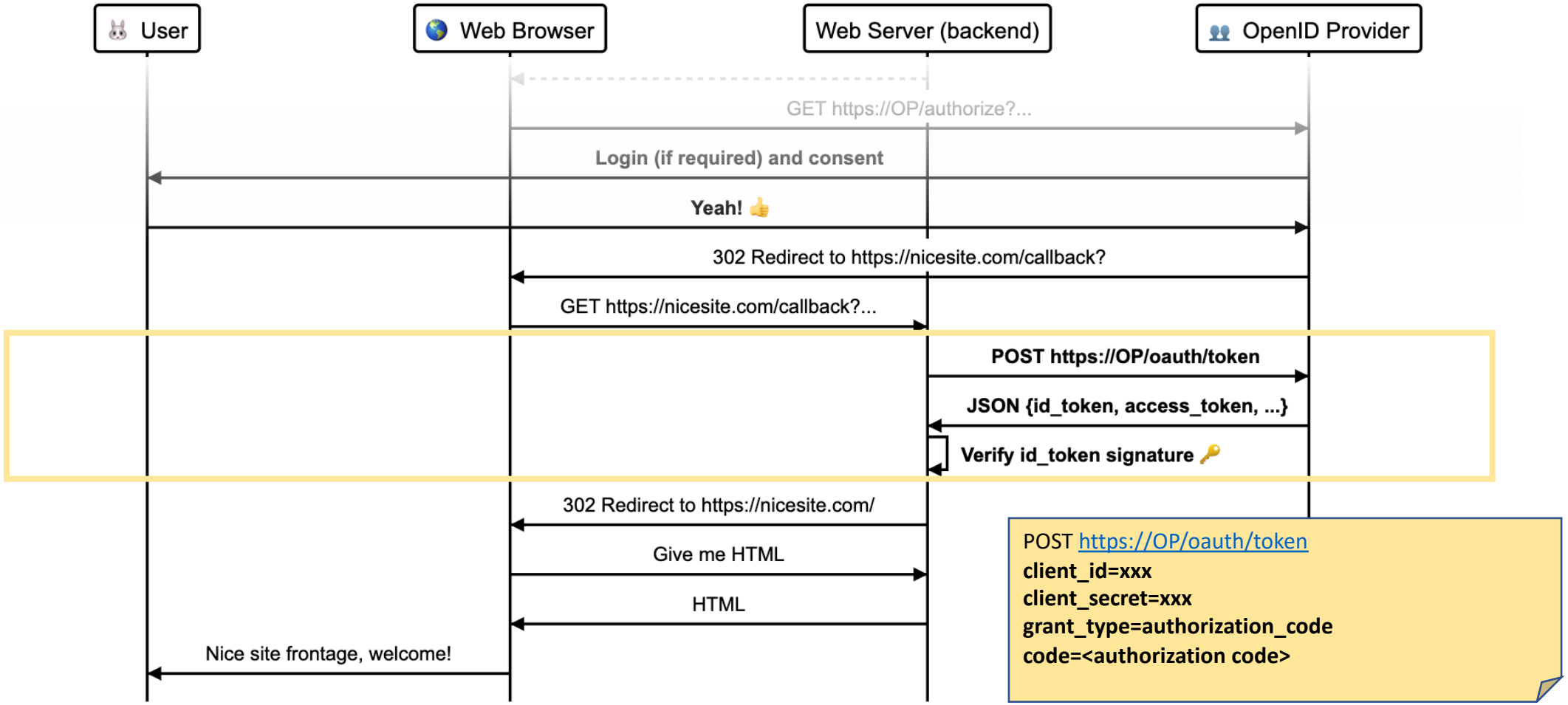
Authorization code grant flow

OpenID provider authenticates user



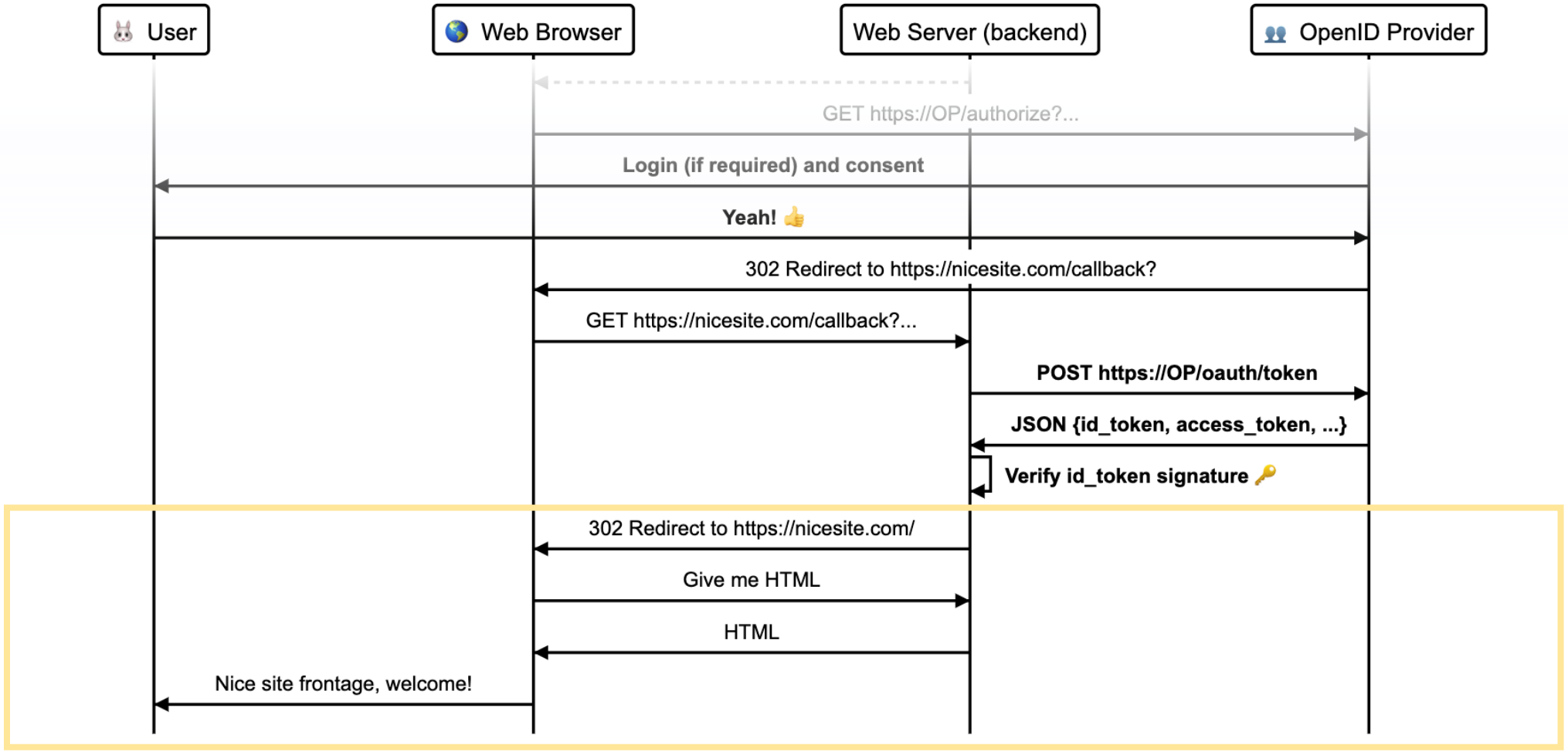
Authorization code grant flow

Backend obtains ID and access tokens

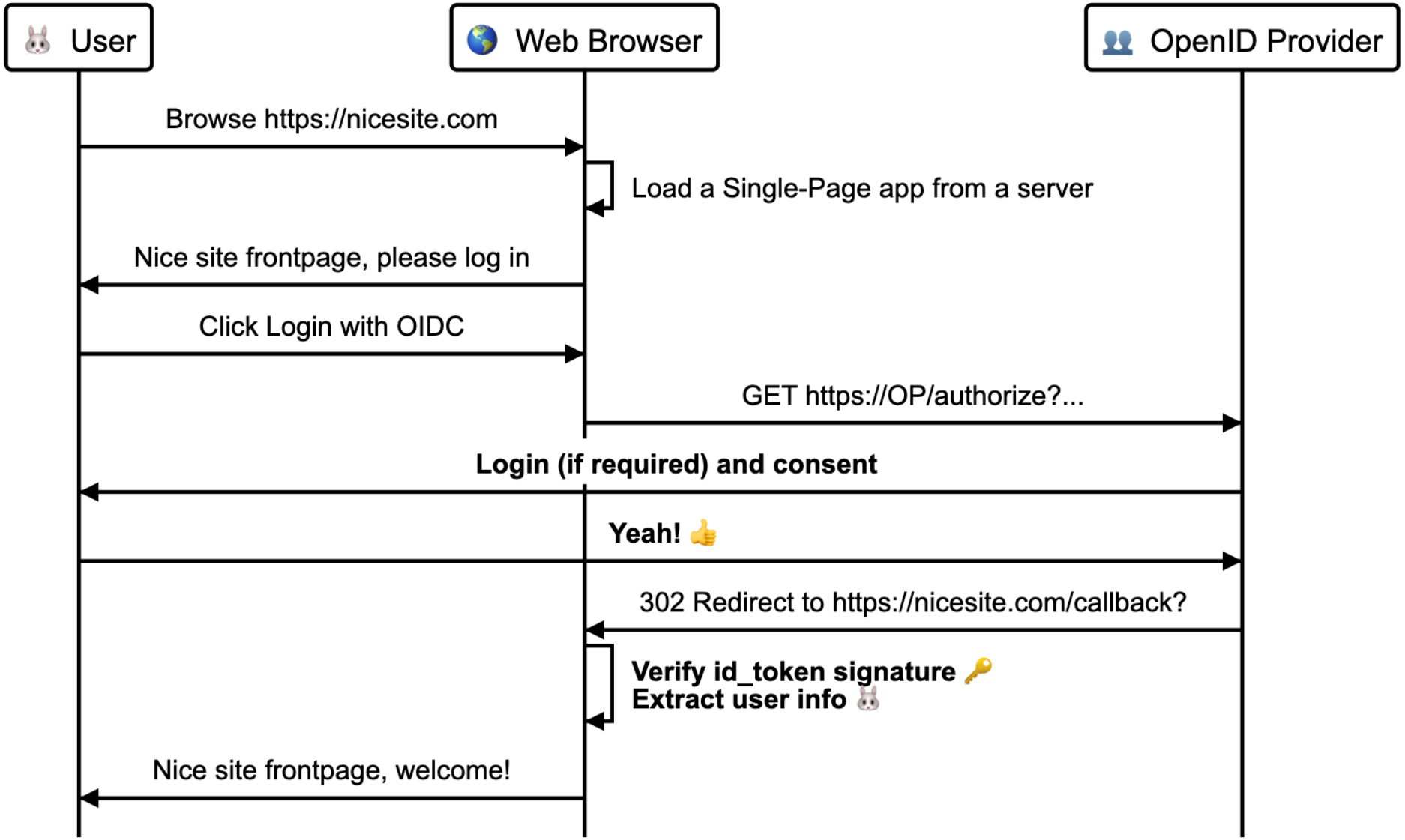


Authorization code grant flow

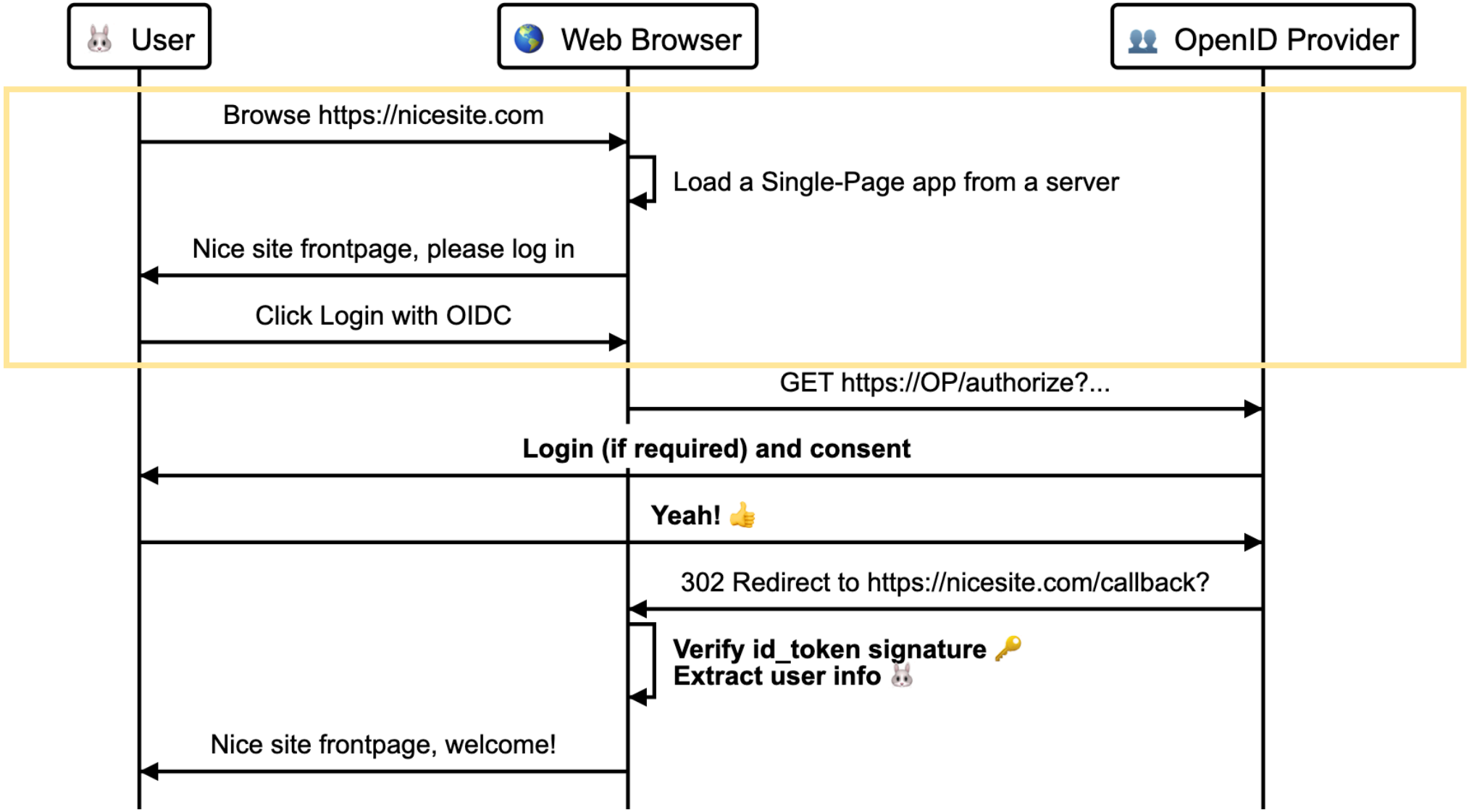
User is logged in



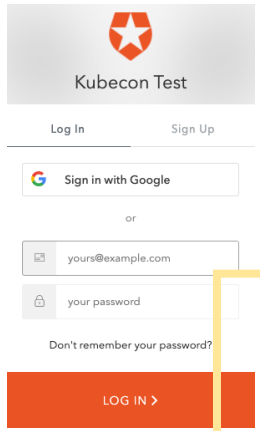
Implicit flow example

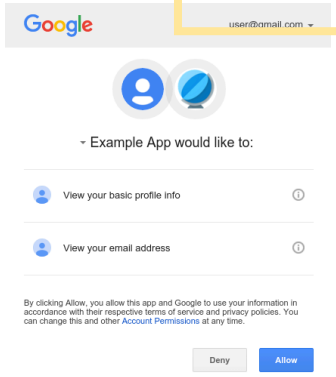


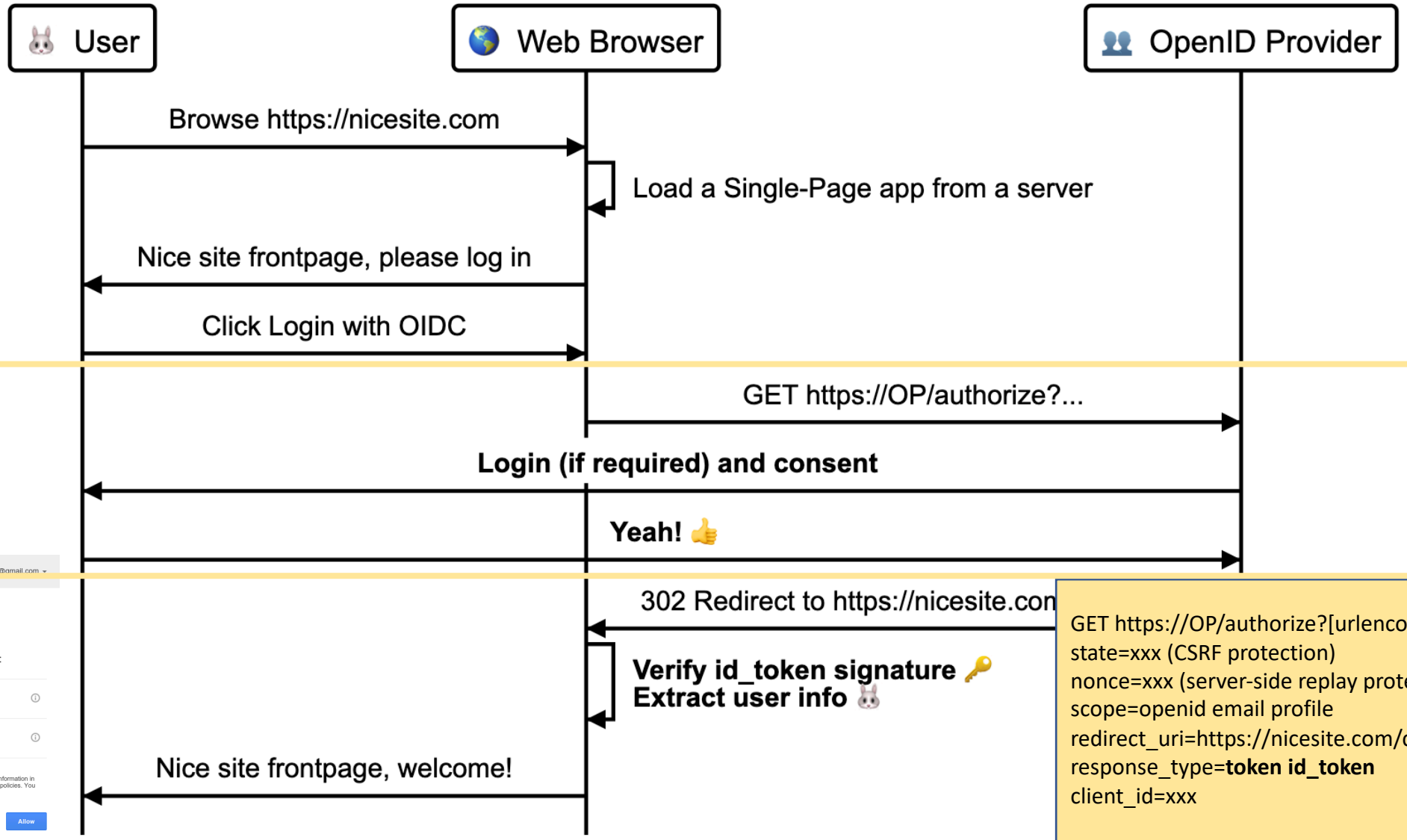
Implicit flow example



Implicit flow example

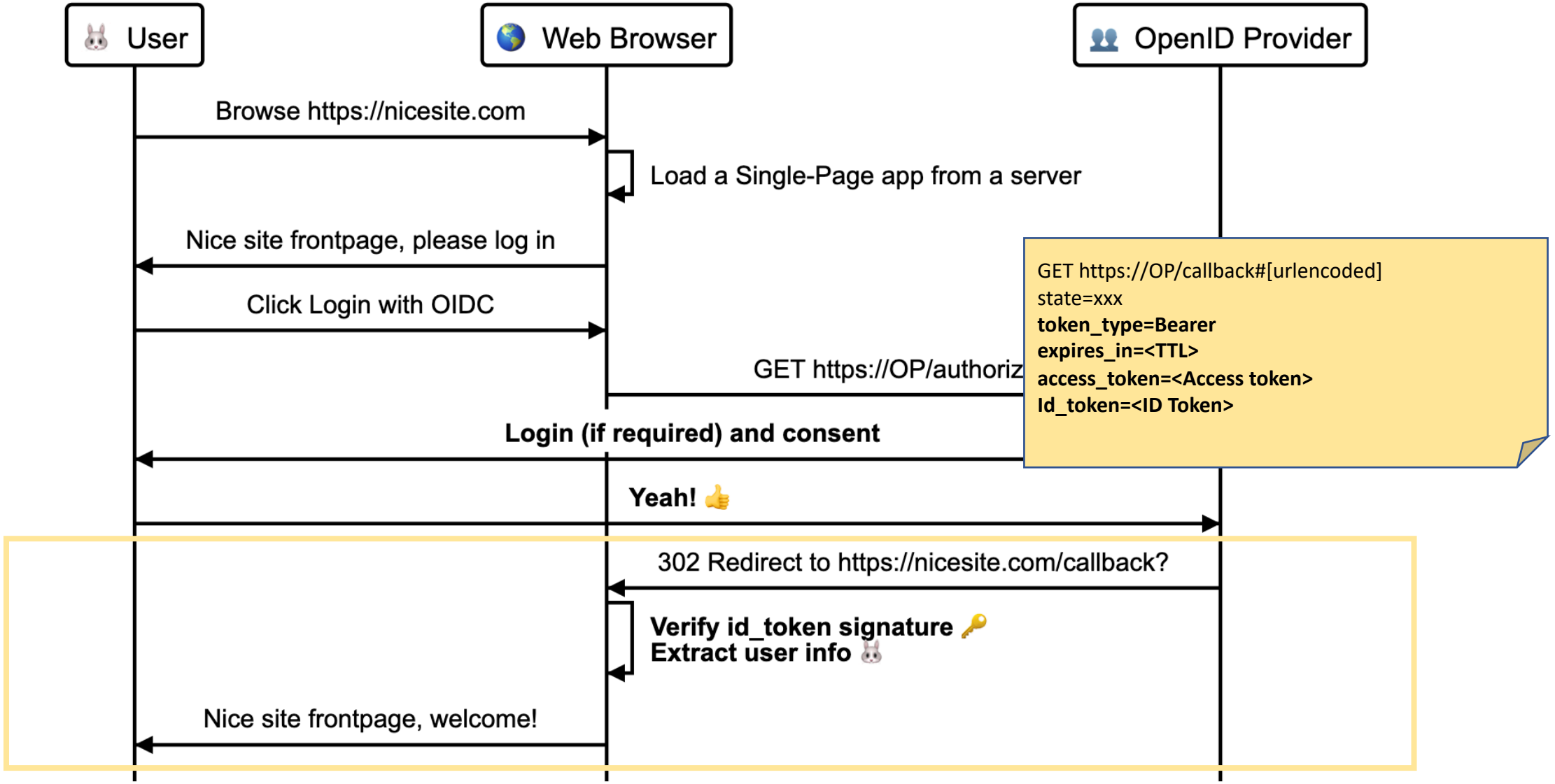






GET https://OP/authorize?[urlencoded]
state=xxx (CSRF protection)
nonce=xxx (server-side replay protection)
scope=openid email profile
redirect_uri=https://nicesite.com/callback
response_type=**token id_token**
client_id=xxx

Implicit flow example



OIDC comparison

vs other "single sign-on" mechanisms

- Reuse credentials: i.e. directory server (LDAP) or shared user DB.
  - Shared credentials, but need to re-authenticate on every app..
- Smart-card based authentication
  - Credentials are the smart card… applications must support it.
- "Token" based solutions. Obtain token, token is the identity
  - Cookies for applications hosted in the same domain (paths or subdomains).
  - Kerberos or other Ticket Granting Ticket systems, very specific.
- Facebook connect / Log in with Facebook.
  - Identity + provide access to user data.
  - You must trust Facebook. Censorship?
- …

OIDC comparison

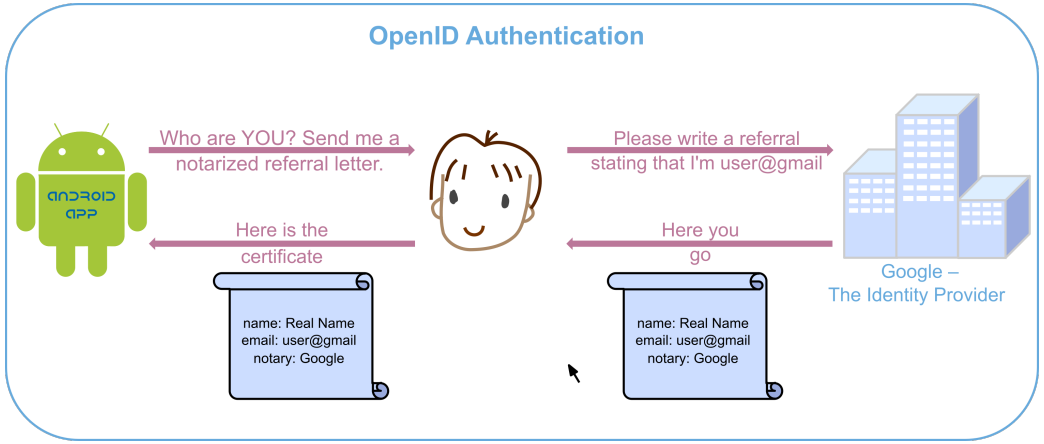
vs SAML (Security Assertion Markup Language)

- SAML is SOAP and XML Format – OIDC is RESTful+JSON.
- SAML: Service Provider (SP) and Identity Provider (IDP).
  - OIDC: Repying Party (RP) and OpenID Provider (OP).
- In SAML, SP is *always* a website.
  - OIDC can be web, mobile or native applications.
- In SAML, the *assertion* is a signed XML document with subject information, issuer and authentication event.
  - OIDC has the equivalent *ID Token*, a signed JSON document.
- SAML *back-channel* is rarely used. SP and IDP don't need connectivity!
  - In OIDC, normally RP uses back channel to retrieve information from OP.
- SAML: No implicit user consent (can be hard-coded by developer).
  - OIDC, built on top of OAuth2, provides built-in authorization layer.

OIDC comparison

vs OpenID 1.0 and OpenID 2.0

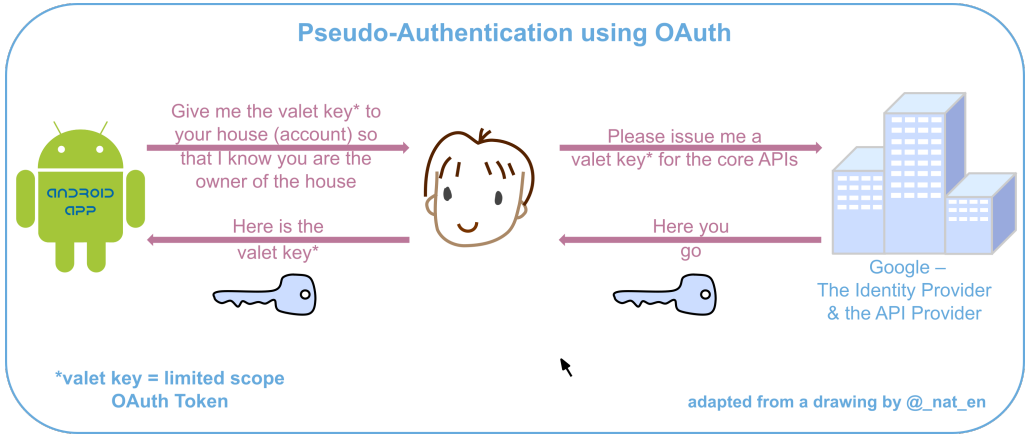
- OpenID Connect != OpenID (different standards)
- OIDC is 3rd generation of OpenID (deprecated)
- In OpenID, identifier is an URL or XRI:
  - i.e. http://alice.openid.example.org
- OpenID works on Oauth 1.0a + extensions
- OpenID provides RP identity “certificate”
- 2005: OpenID 1.0
  - Formerly Yadis (Yey another distributed ID system)
- 2007: OpenID 2.0
  - Google, Microsoft, Paypal, Facebook, MySpace…
- February 2014: OpenID Connect



OIDC comparison

vs OAuth 2.0

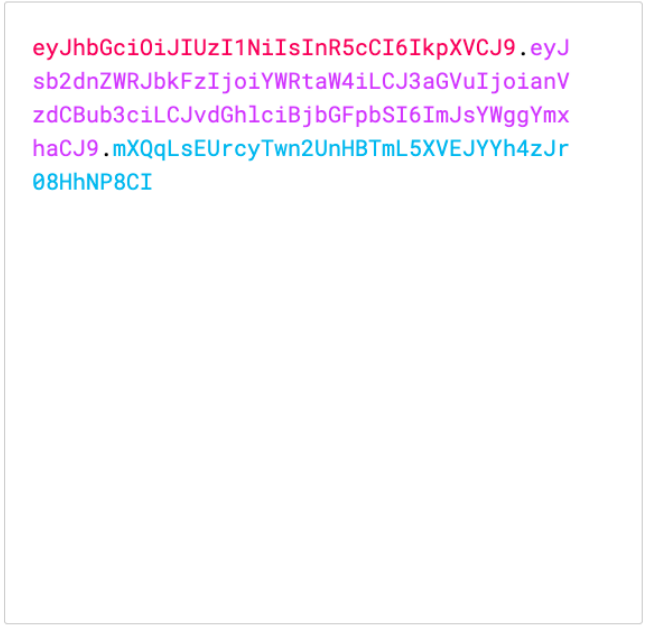
- OAuth is for *authorization*, not *authentication*
- Pseudo-auth using OAuth is possible
  - But dangerous!
  - OAuth provides an access “key”
  - Identity certificate vs your apartment key
  - Does the *key* prove identity?
- In OIDC, the *key* provides access to a *locker* containing the identity information
- “Abusing” standard OAuth2 protocol:
  - plus Identity Token
  - plus UserInfo Endpoint



OIDC comparison

vs JWT (JSON Web Tokens)

- JWT is a standard for signed / encrypted data with JSON payload.
  - Either private secret or public/private key.
- Payload contains *claims* (i.e.: “user logged in as admin”).
- Token provided to a client, then client can use that token as a prove.
- Used in OIDC as *ID Token*



Why use OIDC?

- Interoperability: different implementations get well together.
- Security: reliable, gets you out of the risky business of managing passwords.
  - PKI increases security and delegates responsibility to “expert” service providers.
- Ease of deployment: tons of libraries ready to use.
- Flexibility
- Wide support of devices

Which one should I use?

- Mobile applications → Use OIDC
- Writing a new app? → Use OIDC
- App only supports SAML, and IDP supports SAML? → Use SAML

Testing and debugging OIDC

- Some useful tools to help with debugging:
  - https://openidconnect.net/
  - https://oidcdebugger.com/
  - Browser network console
  - Curl
- Prerequisites
  - OIDC provider
  - Client ID (and secret for code flow) registered in the provider
  - Redirect URIs allowed in the provider
  - Client knows OP endpoint (Autodiscover via /.well-known/openid-configuration)
- Demo: code flow example using https://openidconnect.net/
- Demo: implicit flow using https://oidcdebugger.com/

Creating an OIDC client

- Multiple OIDC libraries: http://openid.net/developers/libraries/
- Certified RP libraries:
  - C Apache mod_auth_openidc
  - C#
  - Erlang
  - JavaScript
  - PHP
  - Python
  - Ruby
  - Typescript

Uncertified RP libraries:

- Elixir
- Erlang
- Go
- Haskell
- Java
- JavaScript
- Python
- …

- Certified Servers and Services, provider libraries, etc.

Python + flask OIDC login

Dependencies and initialization

```
// /requirements.txt

flask
python-dotenv
requests
authlib
six
```

```
# /server.py

from functools import wraps
import json
from os import environ as env
from werkzeug.exceptions import HTTPException

from dotenv import load_dotenv, find_dotenv
from flask import Flask
from flask import jsonify
from flask import redirect
from flask import render_template
from flask import session
from flask import url_for
from authlib.integrations.flask_client import OAuth
from six.moves.urllib.parse import urlencode
```

```
app = Flask(__name__)

oauth = OAuth(app)

auth0 = oauth.register(
    'auth0',
    client_id='TYnGaEIoVYYIaVA6KL3GwxqsGBtYQc3B',
    client_secret='YOUR_CLIENT_SECRET',
    api_base_url='https://airadier.eu.auth0.com',
    access_token_url='https://airadier.eu.auth0.com/oauth/token',
    authorize_url='https://airadier.eu.auth0.com/authorize',
    client_kwargs={
        'scope': 'openid profile email',
    },
)
```

Python + flask OIDC login

Handle the OP callback

```python
# /server.py

# Here we're using the /callback route.
@app.route('/callback')
def callback_handling():
    # Handles response from token endpoint
    auth0.authorize_access_token()
    resp = auth0.get('userinfo')
    userinfo = resp.json()

    # Store the user information in flask session.
    session['jwt_payload'] = userinfo
    session['profile'] = {
        'user_id': userinfo['sub'],
        'name': userinfo['name'],
        'picture': userinfo['picture']
    }
    return redirect('/dashboard')
```

Python + flask OIDC login

Trigger the OIDC authentication

```
<div class="login-box auth0-box before">
    <img src="https://i.cloudup.com/StzWWrY34s.png" />
    <h3>Auth0 Example</h3>
    <p>Zero friction identity infrastructure, built for developers</p>
    <a class="btn btn-primary btn-lg btn-login btn-block" href="/login">Log In</a>
</div>
```

```
# /server.py


@app.route('/login')
def login():
    return auth0.authorize_redirect(redirect_uri='YOUR_CALLBACK_URL')
```

Python + flask OIDC login

Dashboard with user info (requires authentication)

```
# /server.py

@app.route('/dashboard')
@requires_auth
def dashboard():
    return render_template('dashboard.html',
                           userinfo=session['profile'],
                           userinfo_pretty=json.dumps(session['jwt_payload'], indent=4))
```

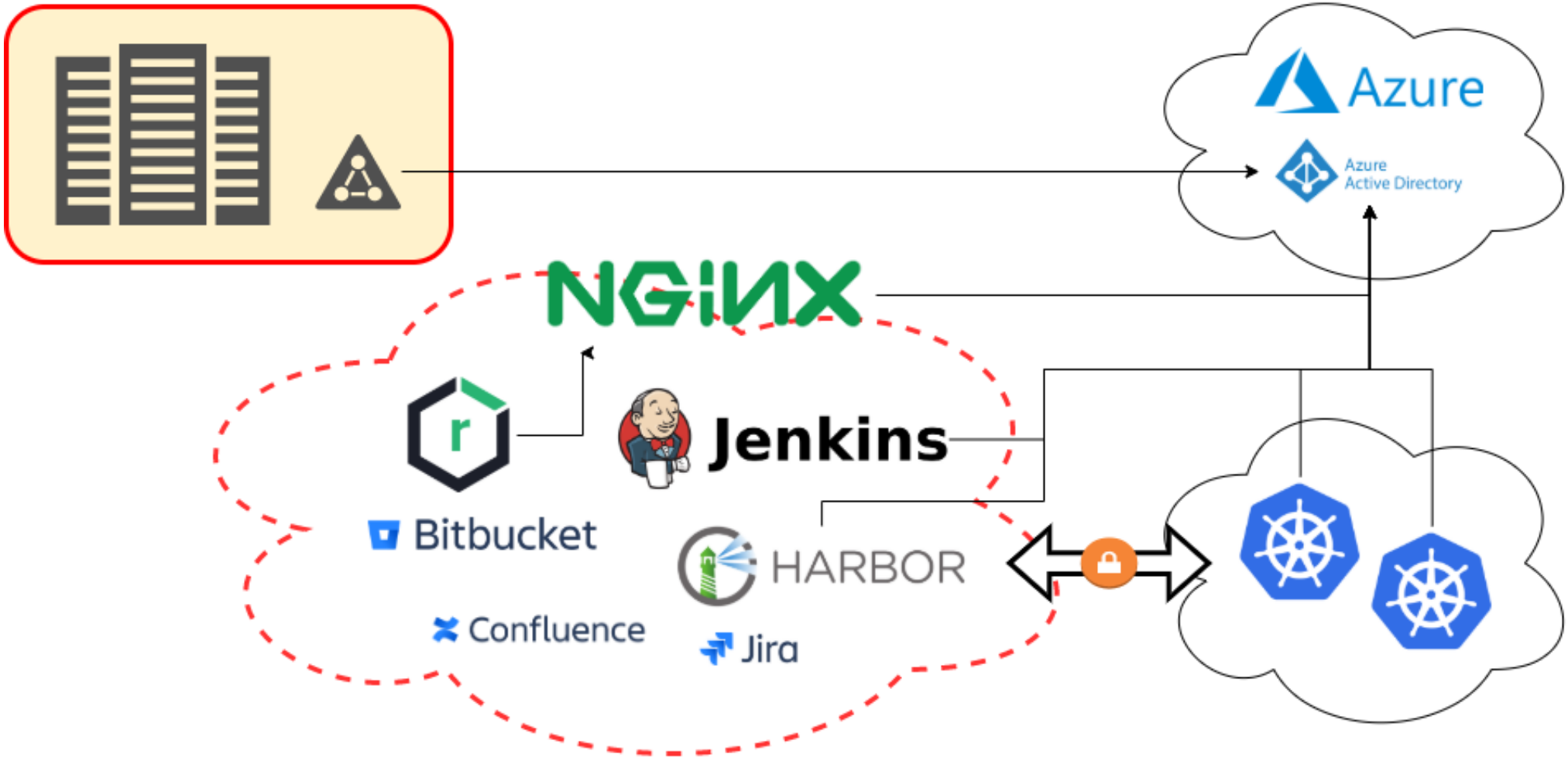
```
def requires_auth(f):
  @wraps(f)
  def decorated(*args, **kwargs):
    if 'profile' not in session:
      # Redirect to Login page here
      return redirect('/')
    return f(*args, **kwargs)

  return decorated
```

```
<div class="logged-in-box auth0-box logged-in">
    <h1 id="logo"><img src="//cdn.auth0.com/samples/auth0_logo_final_blue_RGB.png" /></h1>
    <img class="avatar" src="{{userinfo['picture']}}"/>
    <h2>Welcome {{userinfo['name']}}</h2>
    <pre>{{userinfo_pretty}}</pre>
    <a class="btn btn-primary btn-lg btn-logout btn-block" href="/logout">Logout</a>
</div>
```

Real world example

Dealing with complex authentication scenarios



Real world example

Apps supporting OIDC “out of the box”

Kubernetes

- https://github.com/kubernetes/client-go/tree/master/plugin/pkg/client/auth/azure
- Group based per-namespace permissions.

Harbor

- No groups at the time of implementation. Now available.
- Auto-onboarding issues, user could set its own username → PR merged.
- Configurable username claim → PR merged.

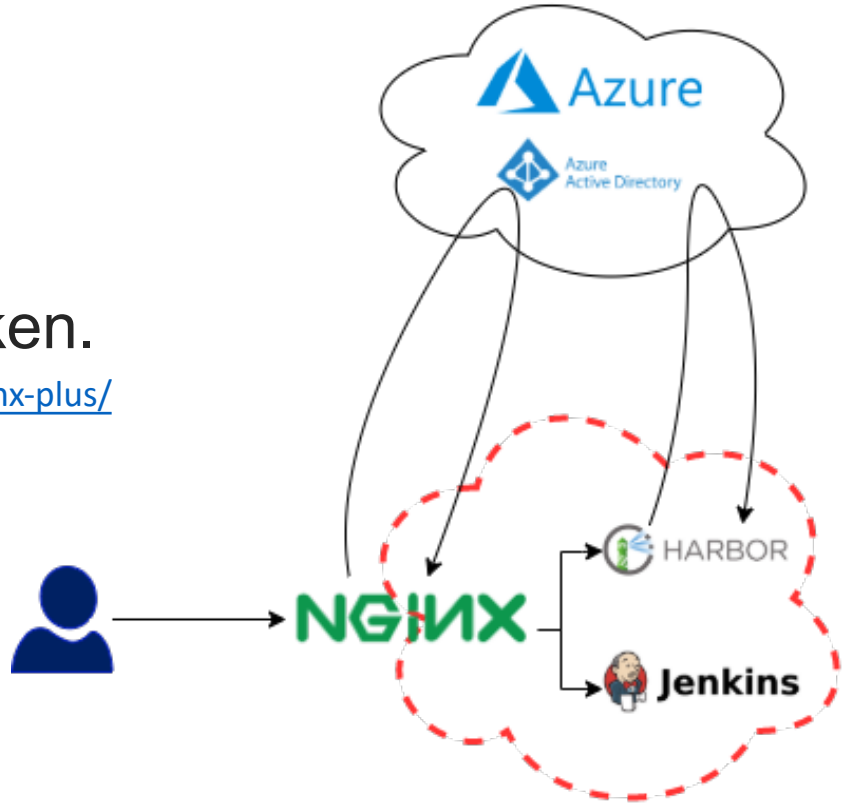
Jenkins (via plugin)

Atlassian Suite (Confluence + Jira + Bitbucket) – SSO plugins

Real world example

Nginx Application Gateway

- Nginx Plus
  - **auth_jwt** module, provides auth via JWT token.
  - Njs module. Some JS for OIDC dance and get token.
  - https://www.nginx.com/blog/authenticating-users-existing-applications-openid-connect-nginx-plus/
  - https://github.com/nginxinc/nginx-openid-connect
- Open-Source version
  - It lacks njs and auth_jwt modules.
  - **Lua** module and some Lua code can do the trick.
  - https://github.com/zmartzone/lua-resty-openidc
- How to handle non-browser sessions (i.e. git ssh clone)?
  - Some *magic* with JS or Lua modules to keep a list of whitelisted Ips
  - OIDC logging code triggers whitelisting. Not perfect… good enough.



Real world example

Non OIDC tools

- Not all apps supported OpenID connect
- Some workarounds were needed.
- Let Nginx authenticate the user and use RUT (Remote User Token) headers to provide username to the application.
  - https://help.sonatype.com/repomanager3/system-configuration/user-authentication/authentication-via-remote-user-token
- Onboarding?
  - Use API to check and onboard if required.

Other caveats

- User migration
- Non-standard claims in ID Token
  - https://openid.net/specs/openid-connect-core-1_0.html#ScopeClaims
  - Email might be required (i.e. Sysdig Monitor / Secure)
- TLS certificates. To trust or not to trust?
- Single Sign Off / Log out
  - Logout in one tool should logout every other tool?
  - https://openid.net/specs/openid-connect-session-1_0.html
- Less secure?
  - Credential leakage can expose multiple applications
  - Use additional security measures: MFA, smartcards, etc.
- Implicit grant flow
  - Not safe and not easy.

The good, the bad and the ugly

- **OpenID Connect is modern, easy to use and to implement, interoperable, flexible, widely supported, and can improve security and make your users happier, while reducing help desk incidents.**

- **Delegating credentials management to a single service can raise trust and availability (single point of failure) issues. Additional measures should be applied to protect credentials, like MFA.**

- Some implementations and standarizations are not yet perfect, and some applications might not yet support OIDC. Some hacks and workarounds might be needed. **But we like challenges, don't we?**

OpenQ&A


KubeCon
CloudNativeCon
Europe 2020
Virtual
HELM
KEEP CLOUD NATIVE
CONNECTED