

Formation MicroService - Sécurité

Ihab ABADI / UTOPIOS

SOMMAIRE

1. Les principes de base en design d'architecture micro-service robuste
2. Utilisation de OAuth 2.0 et OpenID
3. Sécurisation communication entre service avec JWT et mTLS
4. Sécurisation communication entre service en gRPC
5. Configuration sécurisée des containers Dockers
6. Sécurisation des micro-services en Kubernetes
7. Mise en place d'authentification et autorisation en React native
8. Problématique du Deep Linking
9. Sécurisation du stockage local

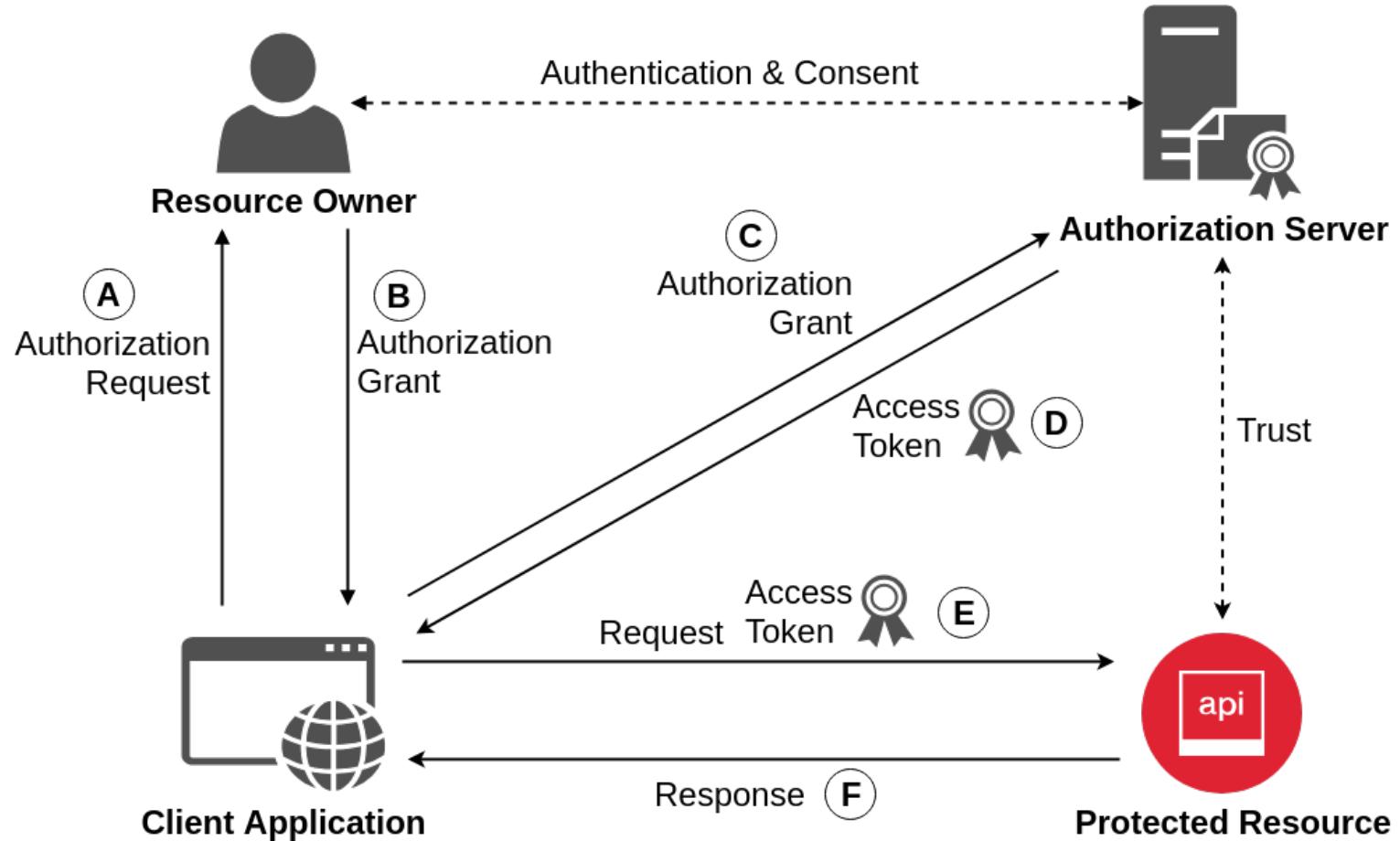
Les principes de base en design d'architecture micro-service robuste

- Sécurité par la conception.
- Scan des dépendances.
- Utilisation des HTTPS.
- Utilisation des Access et Identity Token.
- Crypter et protéger les secrets.
- Protection au niveau du conteneur.

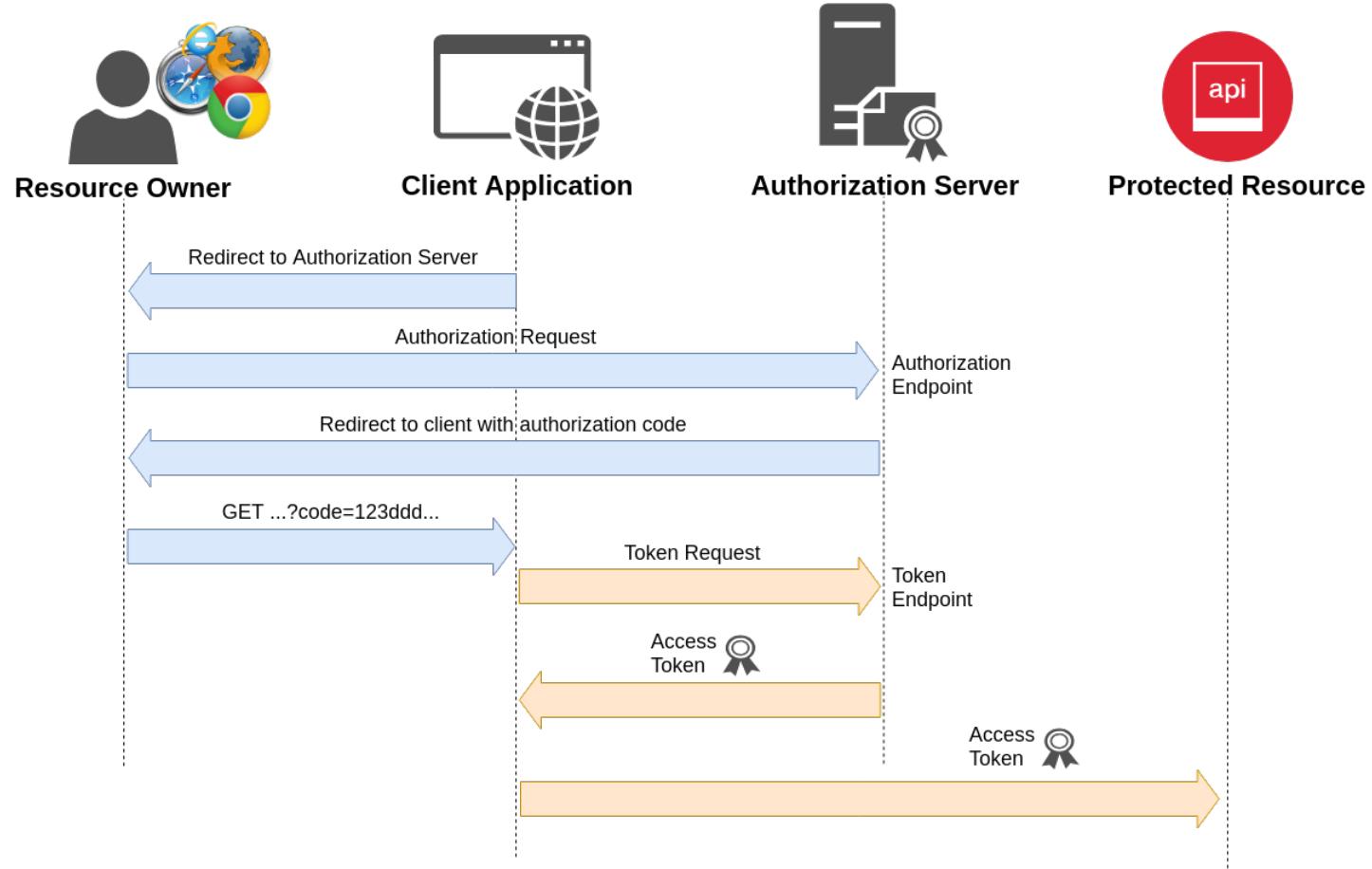
Utilisation de OAuth2

- Oauth2 est un framework de délégation d'autorisation.
- Oauth2 met en place 4 entités :
 - Client Application
 - Resource Owner
 - Autorisation Server
 - Protected Resource

Utilisation de OAuth2



Utilisation de OAuth2 - WorkFlow



Utilisation de OAuth2 - AUTHORIZATION REQUEST

- GET `https://authserver.example.com/authorize`
 - `?response_type=code`
 - `&client_id=abcdefg`
 - `&redirect_uri=https://client.abc.com/callback` &`scope=api.read api.write`
 - `&state=xyz`

Utilisation de OAuth2 - AUTHORIZATION REQUEST

- GET <https://authserver.example.com/authorize>
 - ?response_type=code
 - &client_id=abcdefg
 - &redirect_uri=https://client.abc.com/callback &scope=api.read api.write
 - &state=xyz

Utilisation de OAuth2 - AUTHORIZATION REQUEST

- GET `https://authserver.example.com/authorize`
 - `?response_type=code`
 - `&client_id=abcdefg`
 - `&redirect_uri=https://client.abc.com/callback` &`scope=api.read api.write`
 - `&state=xyz`

Utilisation de OAuth2 - AUTHORIZATION REQUEST

- GET `https://authserver.example.com/authorize`
 - `?response_type=code`
 - `&client_id=abcdefg`
 - `&redirect_uri=https://client.abc.com/callback` &`scope=api.read api.write`
 - `&state=xyz`

Utilisation de OAuth2 - AUTHORIZATION REQUEST

- GET `https://authserver.example.com/authorize`
 - `?response_type=code`
 - `&client_id=abcdefg`
 - `&redirect_uri=https://client.abc.com/callback &scope=api.read api.write`
 - `&state=xyz`

Utilisation de OAuth2 - AUTHORIZATION REQUEST

- GET `https://authserver.example.com/authorize`
 - `?response_type=code`
 - `&client_id=abcdefg`
 - `&redirect_uri=https://client.abc.com/callback &scope=api.read api.write`
 - `&state=xyz`

Utilisation de OAuth2 - AUTHORIZATION REQUEST

- GET `https://authserver.example.com/authorize`
 - `?response_type=code`
 - `&client_id=abcdefg`
 - `&redirect_uri=https://client.abc.com/callback &scope=api.read api.write`
 - `&state=xyz`

Utilisation de OAuth2 - AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: <https://client.abc.com/callback>

?code=ab23bhW56Xb

&state=xyz

Utilisation de OAuth2 - AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: <https://client.abc.com/callback>

?code=ab23bhW56Xb

&state=xyz

Utilisation de OAuth2 - AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: <https://client.abc.com/callback>

?code=ab23bhW56Xb

&state=xyz

Utilisation de OAuth2 TOKEN REQUEST(BASIC AUTH)

Client-Id=123, Client-Secret=456, Base64(123:456)="MTIzOjQ1Ng=="
POST <https://authserver.example.com/token>

Content-Type:

application/x-www-form-urlencoded

Authorization: Basic MTIzOjQ1Ng==

grant_type=authorization_code&code=ab23bhW56X
&redirect_uri=https://client.abc.com/callback

Utilisation de OAuth2 TOKEN REQUEST(BASIC AUTH)

Client-Id=123, Client-Secret=456, Base64(123:456)="MTIzOjQ1Ng=="

POST <https://authserver.example.com/token>

Content-Type:

application/x-www-form-urlencoded

Authorization: Basic MTIzOjQ1Ng==

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

Utilisation de OAuth2 TOKEN REQUEST (BASIC AUTH)

Client-Id=123, Client-Secret=456, Base64(123:456)="MTIzOjQ1Ng=="

POST <https://authserver.example.com/token>

Content-Type:

application/x-www-form-urlencoded

Authorization: Basic MTIzOjQ1Ng==

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

Utilisation de OAuth2 TOKEN REQUEST (BASIC AUTH)

Client-Id=123, Client-Secret=456, Base64(123:456)="MTIzOjQ1Ng=="

POST <https://authserver.example.com/token>

Content-Type:

application/x-www-form-urlencoded

Authorization: Basic MTIzOjQ1Ng==

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

Utilisation de OAuth2 TOKEN REQUEST (BASIC AUTH)

Client-Id=123, Client-Secret=456, Base64(123:456)="MTIzOjQ1Ng=="

POST <https://authserver.example.com/token>

Content-Type:

application/x-www-form-urlencoded

Authorization: Basic MTIzOjQ1Ng==

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

Utilisation de OAuth2 TOKEN REQUEST (BASIC AUTH)

Client-Id=123, Client-Secret=456, Base64(123:456)="MTIzOjQ1Ng=="

POST <https://authserver.example.com/token>

Content-Type:

application/x-www-form-urlencoded

Authorization: Basic MTIzOjQ1Ng==

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=<https://client.abc.com/callback>

Utilisation de OAuth2 TOKEN REQUEST(BODY)

Client-Id=123, Client-Secret=456

POST <https://authserver.example.com/token>

Content-Type:

application/x-www-form-urlencoded

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

&client_id=123&client_secret=456

Utilisation de OAuth2 TOKEN REQUEST(BODY)

Client-Id=123, Client-Secret=456

POST <https://authserver.example.com/token>

Content-Type:

application/x-www-form-urlencoded

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

&client_id=123&client_secret=456

Utilisation de OAuth2 TOKEN REQUEST(BODY)

Client-Id=123, Client-Secret=456

POST <https://authserver.example.com/token>

Content-Type:

application/x-www-form-urlencoded

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

&client_id=123&client_secret=456

Utilisation de OAuth2 TOKEN REQUEST(BODY)

Client-Id=123, Client-Secret=456

POST <https://authserver.example.com/token>

Content-Type:

application/x-www-form-urlencoded

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

&client_id=123&client_secret=456

Utilisation de OAuth2 TOKEN REQUEST(BODY)

Client-Id=123, Client-Secret=456

POST <https://authserver.example.com/token>

Content-Type:

application/x-www-form-urlencoded

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=<https://client.abc.com/callback>

&client_id=123&client_secret=456

Utilisation de OAuth2 TOKEN REQUEST(BODY)

Client-Id=123, Client-Secret=456

POST https://authserver.example.com/token

Content-Type:

application/x-www-form-urlencoded

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

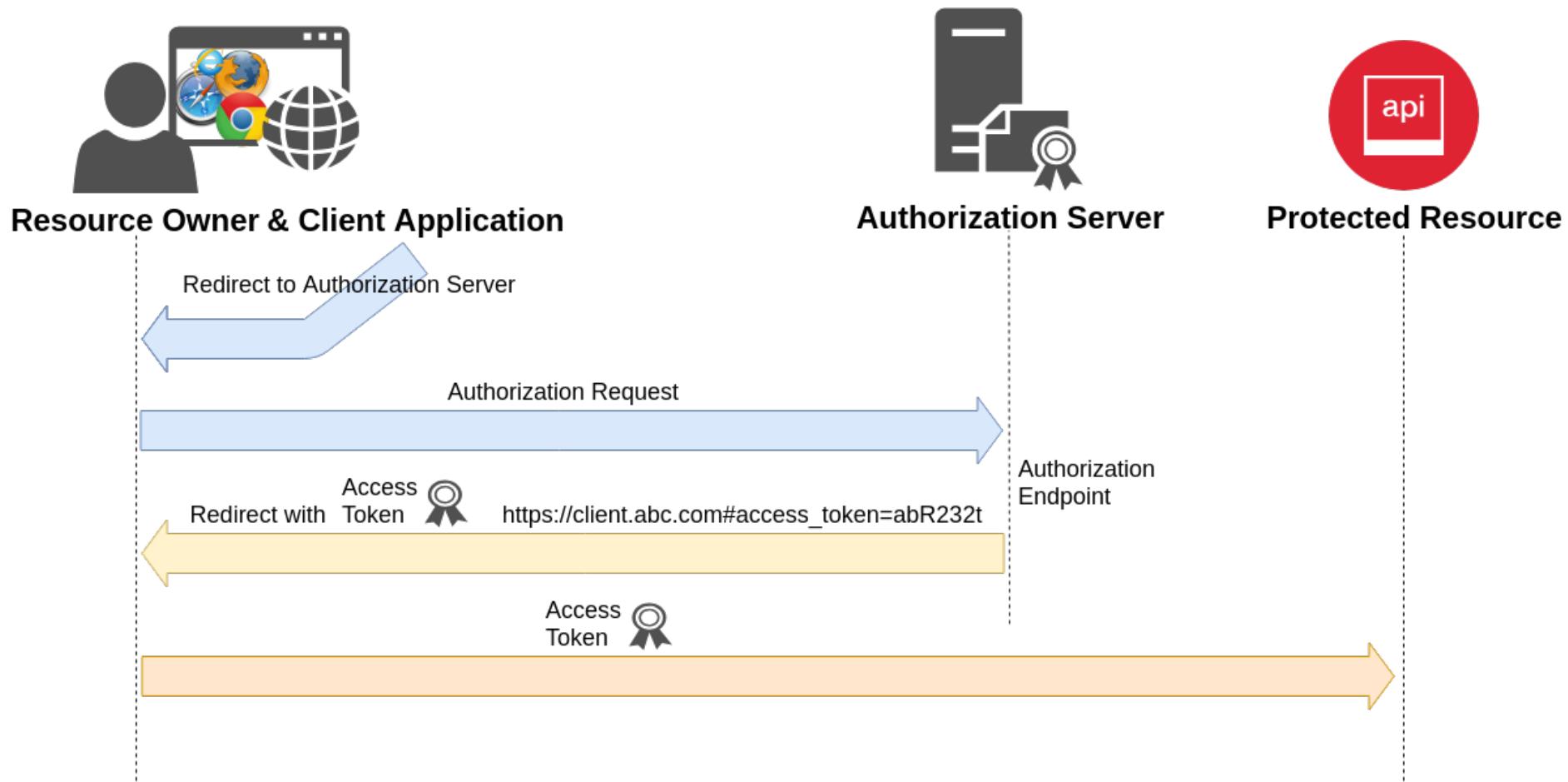
&client_id=123&client_secret=456

Utilisation de OAuth2 TOKEN RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{
    "access_token": "2YotnFZFEjr1zCsicMWpAA",
    "token_type": "bearer",
    "expires_in": 3600,
    "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA"
}
```

Utilisation de OAuth2 IMPLICIT GRANT FLOW



Utilisation de OAuth2 -AUTHORIZATION REQUEST

- GET `https://authserver.example.com/authorize`
 - `?response_type=token`
 - `&client_id=abcdefg`
 - `&redirect_uri=https://client.abc.com/callback &scope=api.read api.write`
 - `&state=xyz`

Utilisation de OAuth2 - AUTHORIZATION REQUEST

- GET <https://authserver.example.com/authorize>
 - ?response_type=token
&client_id=abcdefg
 - &redirect_uri=https://client.abc.com/callback &scope=api.read api.write
 - &state=xyz

Utilisation de OAuth2 - AUTHORIZATION REQUEST

- GET `https://authserver.example.com/authorize`
 - `?response_type=token`
 - `&client_id=abcdefg`
 - `&redirect_uri=https://client.abc.com/callback &scope=api.read api.write`
 - `&state=xyz`

Utilisation de OAuth2 - AUTHORIZATION REQUEST

- GET `https://authserver.example.com/authorize`
 - `?response_type=token`
 - `&client_id=abcdefg`
 - `&redirect_uri=https://client.abc.com/callback &scope=api.read api.write`
 - `&state=xyz`

Utilisation de OAuth2 - AUTHORIZATION REQUEST

- GET `https://authserver.example.com/authorize`
 - `?response_type=token`
 - `&client_id=abcdefg`
 - `&redirect_uri=https://client.abc.com/callback &scope=api.read api.write`
 - `&state=xyz`

Utilisation de OAuth2 - AUTHORIZATION REQUEST

- GET `https://authserver.example.com/authorize`
 - `?response_type=token`
 - `&client_id=abcdefg`
 - `&redirect_uri=https://client.abc.com/callback` `&scope=api.read api.write`
 - `&state=xyz`

Utilisation de OAuth2 - AUTHORIZATION REQUEST

- GET `https://authserver.example.com/authorize`
 - `?response_type=token`
 - `&client_id=abcdefg`
 - `&redirect_uri=https://client.abc.com/callback` &`scope=api.read api.write`
 - `&state=xyz`

Utilisation de OAuth2 - AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: <https://client.abc.com/callback>

#access_token=2YotnFZFEjr1zCsicMWpAA

&token_type=bearer

&expires_in=3600

&scope=api.read api.write

&state=xyz

Utilisation de OAuth2 - AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: <https://client.abc.com/callback>

#access_token=2YotnFZFEjr1zCsicMWpAA

&token_type=bearer

&expires_in=3600

&scope=api.read api.write

&state=xyz

Utilisation de OAuth2 - AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: <https://client.abc.com/callback>

#access_token=2YotnFZFEjr1zCsicMWpAA

&token_type=bearer

&expires_in=3600

&scope=api.read api.write

&state=xyz

Utilisation de OAuth2 - AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: <https://client.abc.com/callback>

#access_token=2YotnFZFEjr1zCsicMWpAA

&token_type=bearer

&expires_in=3600

&scope=api.read api.write

&state=xyz

Utilisation de OAuth2 - AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: <https://client.abc.com/callback>

#access_token=2YotnFZFEjr1zCsicMWpAA

&token_type=bearer

&expires_in=3600

&scope=api.read api.write

&state=xyz

Utilisation de OAuth2 - AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: <https://client.abc.com/callback>

#access_token=2YotnFZFEjr1zCsicMWpAA

&token_type=bearer

&expires_in=3600

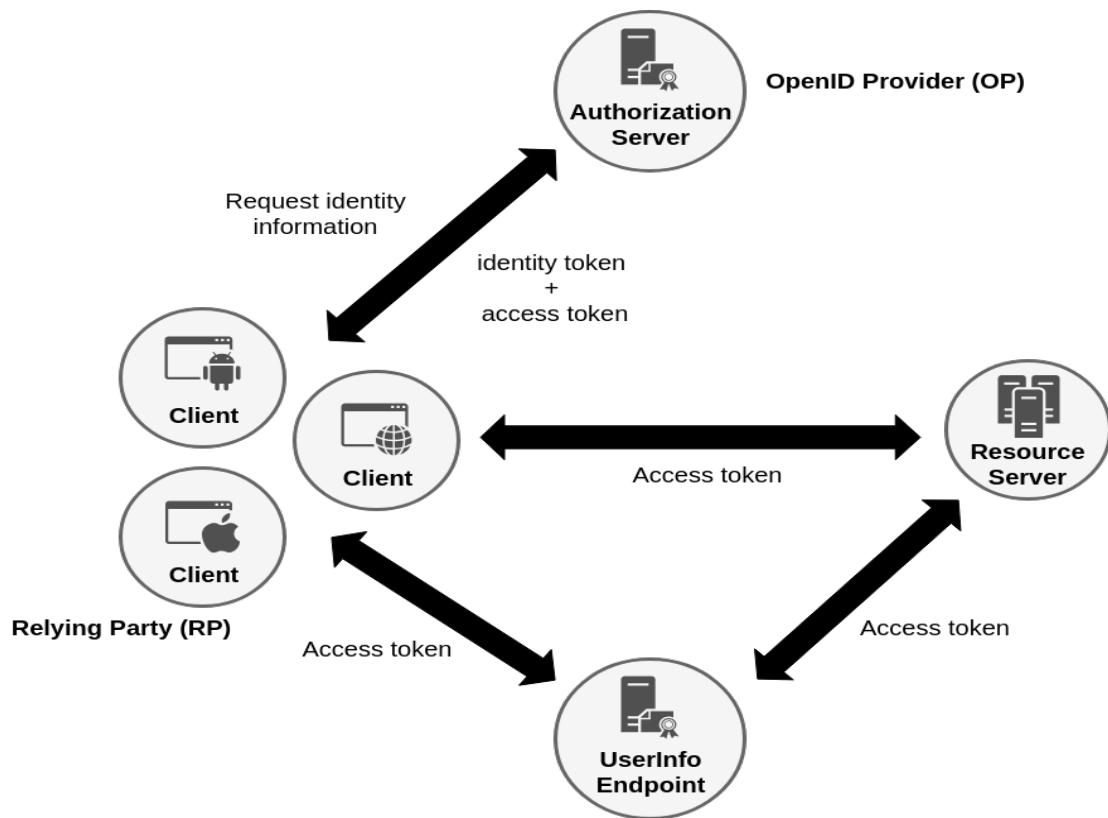
&scope=api.read api.write

&state=xyz

OpenId Connect

- OAuth2 est un framework qui permet de gérer uniquement les autorisations.
- OAuth2 ne permet pas de récupérer des informations d'identification.
- OpenId Connect est une couche qui permet d'ajouter la partie identification à notre framework OAuth2

OpenId Connect



OpenId Connect

Id Token (JWT format)

User Info Endpoint

Standard Scopes

Hybrid Grant Flow

OpenID Provider Configuration Information

JSON WEB TOKEN (JWT)

JSON WEB TOKEN (JWT)

Base 64 Encoded JSON Formatted Value of...

...Header

...Payload

...Signature

```
GET / HTTP/1.1
Host: localhost:8080
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1N...
```

- [RFC 7519: JSON Web Token \(JWT\)](#)
- [JSON Web Token Best Current Practices](#)
- [Proof-of-Possession Key Semantics for JSON Web Tokens \(JWTs\)](#)

JSON WEB TOKEN (JWT)

Header

```
{  
  typ: "JWT",  
  alg: "RS256"  
}
```

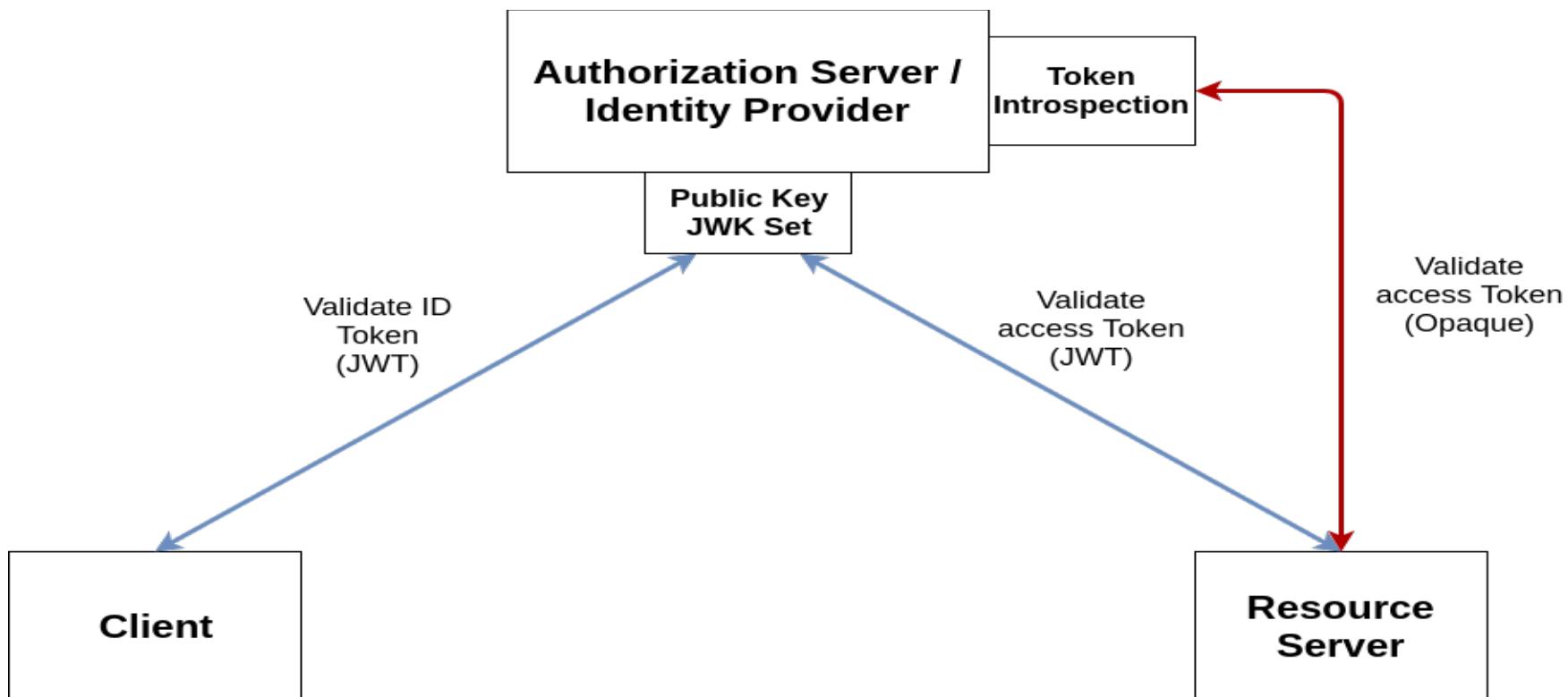
Payload

```
{  
  iss: "https://identity.example.com",  
  aud: "my-client-id",  
  exp: 1495782385,  
  nonce: "N0.46824857243233511495739124749",  
  iat: 1495739185,  
  at_hash: "hC1NDSB8WZ9SnjXTid175A",  
  sub: "mysubject",  
  auth_time: 1495739185,  
  email: "test@gmail.com"  
}
```

JWT ID TOKEN CLAIMS

Scope	Required	Description
iss	X	Issuer Identifier
sub	X	Subject Identifier
aud	X	Audience(s) of this ID Token
exp	X	Expiration time
iat	X	Time at which the JWT was issued
auth_time	(X)	Time of End-User authentication
nonce	--	Associate a client with an ID Token

JSON WEB TOKEN (JWT) - Validation



JSON WEB TOKEN (JWT)- USER INFO ENDPOINT

```
GET /userinfo HTTP/1.1
Host:
identityserver.example.com
Authorization: Bearer
SlAV32hkKG
```

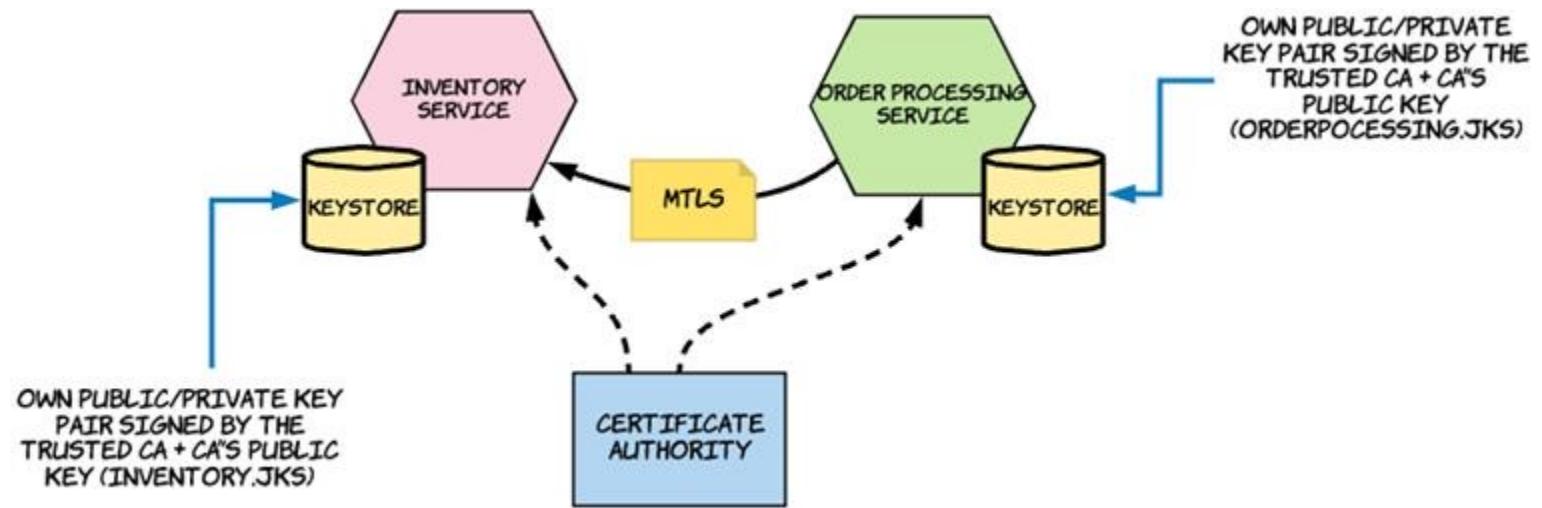
```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "sub": "248289761001",
  "name": "Jane Doe",
  "given_name": "Jane",
  "family_name": "Doe",
  "preferred_username": "j.doe",
  "email":
  "janedoe@example.com",
  "picture": "http://example.com/janedoe/me.jpg"
}
```

Microservices et mTLS

- Mutual TLS (ou mTLS) fait référence à la sécurité de la couche de transport qui utilise un canal crypté bidirectionnel entre le serveur et le client.
- mTLS est le protocole préféré pour sécuriser les communications entre les microservices dans les applications natives du cloud

Microservices et mTLS



DOCKER – UID et GID

- Docker crée les conteneurs avec uid 0
- Nous avons la possibilité de spécifier UID ou GID au moment de créer l'image pour indiquer user à utiliser dans notre conteneur.
- Nous avons également la capacité de surcharger l'utilisateur au moment de créer le conteneur.
- Exemple

DOCKER – Utilisation de namespace remapping et Cgroup

- Dans le cadre d'applications qui doivent être exécuter à l'intérieur d'un conteneur avec des privilèges root, mais avec une isolation du root système, on utilisera le mécanisme de namespace remapping.
- Le namespace remapping consiste à mapper un utilisateur non root du système vers root à l'intérieur du conteneur.
- La configuration peut se faire à l'intérieur du daemon.json
- Exemple

DOCKER – Utilisation de namespace remapping et Cgroup

- Les cGroups fournissent une limitation des ressources et une capacité de création de rapports dans l'espace du conteneur.
- Ils permettent également un contrôle granulaire sur les ressources d'hôte allouées aux conteneurs et à quel moment elles sont allouées.
- Les ressources qu'on peut limiter sont :
 - CPU
 - Memory
 - Network Bandwidth
 - Disk
- Exemple

DOCKER – Attack Surface

- Comme Docker nécessite des privilèges root pour fonctionner, il faut savoir qui a accès à l'hôte docker et au daemon docker.
- Il faut toujours séparer les conteneurs, pour limiter et séparer l'accès au daemon docker.
- Il faut également limiter les capacités de conteneurs strict nécessaire à l'exécution d'application.
- ...

DOCKER – L'utilisation des volumes pour Escalade de privilèges

- Comme un conteneur est créé par défaut avec les droits root, dans le cadre de montage de volume, le conteneur peut agir et modifier des éléments sur le l'hôte
- Exemple.
- Exercice
- Ajouter un utilisateur dans l'hôte à l'aide d'un conteneur qui partage le volume /etc/

DOCKER – L'utilisation des capacités

- Chaque conteneur crée possède les capacités suivantes:
- CHOWN, DAC_OVERRIDE, FSETID, FOWNER, MKNOD, NET_RAW, SETGID, SETUID, SETFCAP, SETPCAP, NET_BIND_SERVICE, SYS_CHROOT, KILL, AUDIT_WRITE
- Dans certains cas d'utilisations, on peut avoir le besoins de supprimer des capacités ou d'en ajouter.
- C'est possible à l'aide des flags –cap-add ou –cap-drop.
- Exemple

DOCKER – docker.sock

- docker.sock est le socket UNIX que le démon Docker écoute. C'est le principal point d'entrée de l'API Docker.
- Il peut également s'agir d'un socket TCP, défaut, Docker utilise par défaut le socket UNIX.
- Le client cli Docker utilise ce socket pour exécuter les commandes docker par défaut.
- Il peut y avoir différentes raisons pour lesquelles vous devrez peut-être monter le socket Docker à l'intérieur d'un conteneur.
- Exemple docker nginx.

DOCKER – docker.sock – Evasion des conteneurs

- Dans le cadre d'un deamon docker accessible par TCP.
- On peut exécuter des conteneurs avec un volume monté.
- On peut utiliser ce volume pour exécuter des actions sur le l'hôte.
- On peut même prendre le contrôle de l'hôte.
- Exemple.

Sécuriser authentification sur une application Mobile

- Dans le cadre d'une authentification par Oauth2 avec une application mobile, les urls de redirection http ne sont pas sécurisés et garantie.
- Pour garantir les urls, on ajoute une couche supplémentaire qui permet d'identifier les demandeurs de token.
- La couche supplémentaire s'appelle PKCE.

Sécuriser authentification sur une application Mobile Fonctionnement PKCE

- PKCE utilise un mécanisme de vérification basé sur du SHA 256
- Le client commence par générer un
 - Code_verifier : chaîne aléatoire (à stocker en mémoire).
 - Code_challenge: sha 256 du code_verifier
- Pendant la demande d'autorisation le code_challenge est envoyé au serveur.
- Après la réussite de la requête d'autorisation, une deuxième requête est envoyée avec le code_verifier, le serveur recalcule le code_challenge pour s'assurer que c'est le même client.

React native le stockage local

- Pour la persistance sur react-native on peut utiliser :
- Variable d'environnement et de configuration.
- Async Storage

React native le stockage local – Async Storage

- Async storage permet :
 - Stockage des données non sensible.
 - Stockage des données d'état (Redux, GraphQL...).
 - Stockage des variables globales.
- Async storage ne permet pas:
 - Stockage des secrets
 - Stokage des tokens

React native le stockage local – Secure Storage

- React native ne possède pas de stockage sécurisé.
- On peut utiliser une implémentation direct sur chaque plateform
- Keychain Service pour IOS
- Secure Shared Preferences pour Android
- KeyStore pour Android

React native – Deep Linking