【MicroProfile開発ガイド】
MicroProfile JWT Propagation

2018/9

日本アイ・ビー・エム システムズ・エンジニアリング株式会社

© 2018 IBM Corporation

Disclaimer

- この資料は日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム　システムズ・エンジニアリング株式会社の正式なレビューを受けておりません。
- 当資料は、資料内で説明されている製品の仕様を保証するものではありません。
- 資料の内容には正確を期するよう注意しておりますが、この資料の内容は2018年9月現在の情報であり、製品の新しいリリース、PTFなどによって動作、仕様が変わる可能性があるのでご注意下さい。
- 今後国内で提供されるリリース情報は、対応する発表レターなどでご確認ください。
- IBM、IBMロゴおよびibm.comは、世界の多くの国で登録されたInternational Business Machines Corporationの商標です。他の製品名およびサービス名等は、それぞれIBMまたは各社の商標である場合があります。現時点でのIBMの商標リストについては、www.ibm.com/legal/copytrade.shtmlをご覧ください。
- 当資料をコピー等で複製することは、日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム　システムズ・エンジニアリング株式会社の承諾なしではできません。
- 当資料に記載された製品名または会社名はそれぞれの各社の商標または登録商標です。
- JavaおよびすべてのJava関連の商標およびロゴはOracleやその関連会社の米国およびその他の国における商標または登録商標です。
- Microsoft, WindowsおよびWindowsロゴは、Microsoft Corporationの米国およびその他の国における商標です。
- Linuxは、Linus Torvaldsの米国およびその他の国における登録商標です。
- UNIXはThe Open Groupの米国およびその他の国における登録商標です。
- GoogleおよびGoogleロゴはGoogle LLCの登録商標です。

© 2018 IBM Corporation

目次

- MicroProfile JWT Propagation概要
  - MicroProfile JWT Propagationとは
  - 利用シナリオ
  - OpenID Connect (OIDC)とは
  - JSON Web Token (JWT)とは
  - MP-JWTとは
  - ロールベース・アクセス制御(RBAC)とは
  - フィーチャーの有効化
- MicroProfile JWT Propagationの使用方法
  - サンプル構成
    - サーバ設定
    - アプリケーション
    - JWTの署名に使用するアルゴリズム
      - RS256
        - 鍵ペアを動的生成する場合
        - 事前に生成ずみの鍵ペアを使用する場合
      - HS256
    - JWTクレームの取得
      - SecurityContext
      - インジェクション
    - サービスからサービスへのJWTの伝搬
  - サードパーティーが発行したJWTの使用
- 参考リンク

© 2018 IBM Corporation

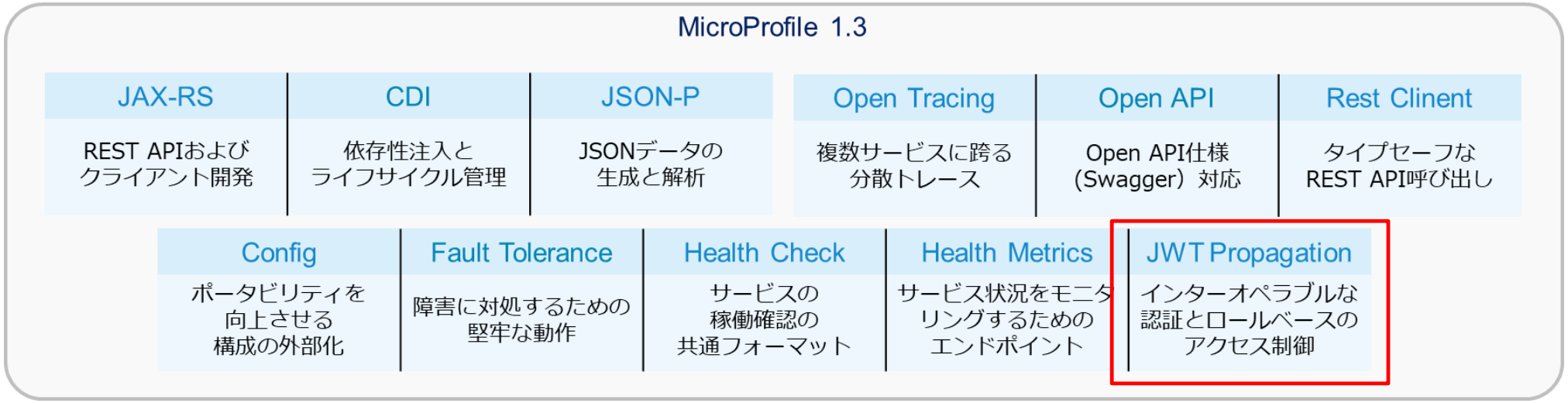
MicroProfile JWT Propagation概要

© 2018 IBM Corporation

MicroProfile JWT Propagationとは

Eclipse MicroProfile JWT Propagation (mpJwt)は、複数のマイクロサービス・エンドポイントからなる環境において一気通貫のロールベース・アクセス制御(RBAC)を行うために、OpenID Connect (OIDC)やJSON Web Token (JWT)のテクノロジーを活用して認証・認可情報を伝搬する機能を提供します。

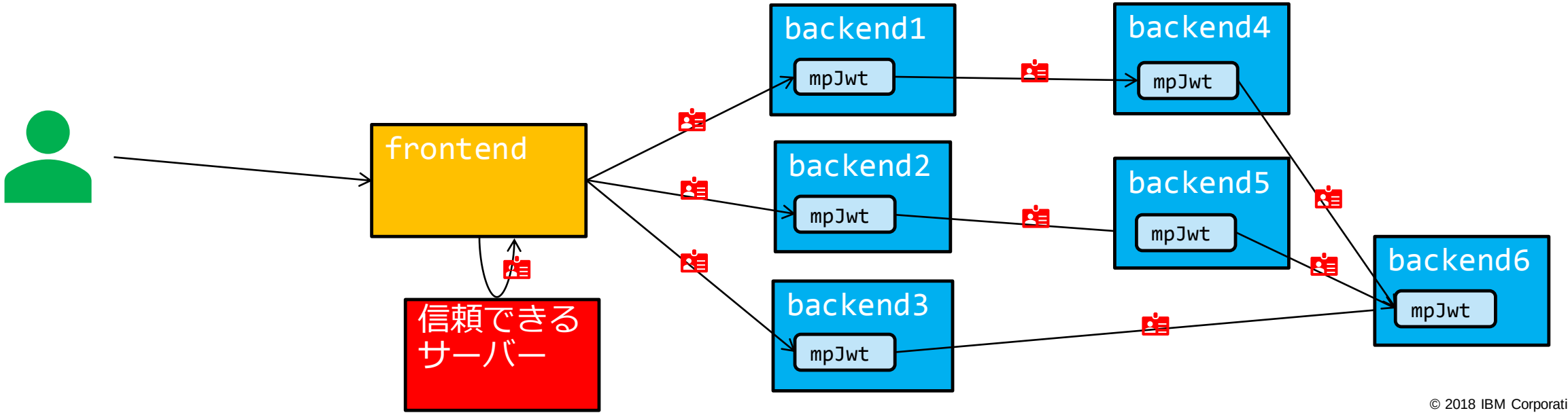
WAS Libertyでは、MicroProfile JWT Authentication 1.0仕様に準拠した、mpJwt-1.0フィーチャーが提供されています。



© 2018 IBM Corporation

利用シナリオ

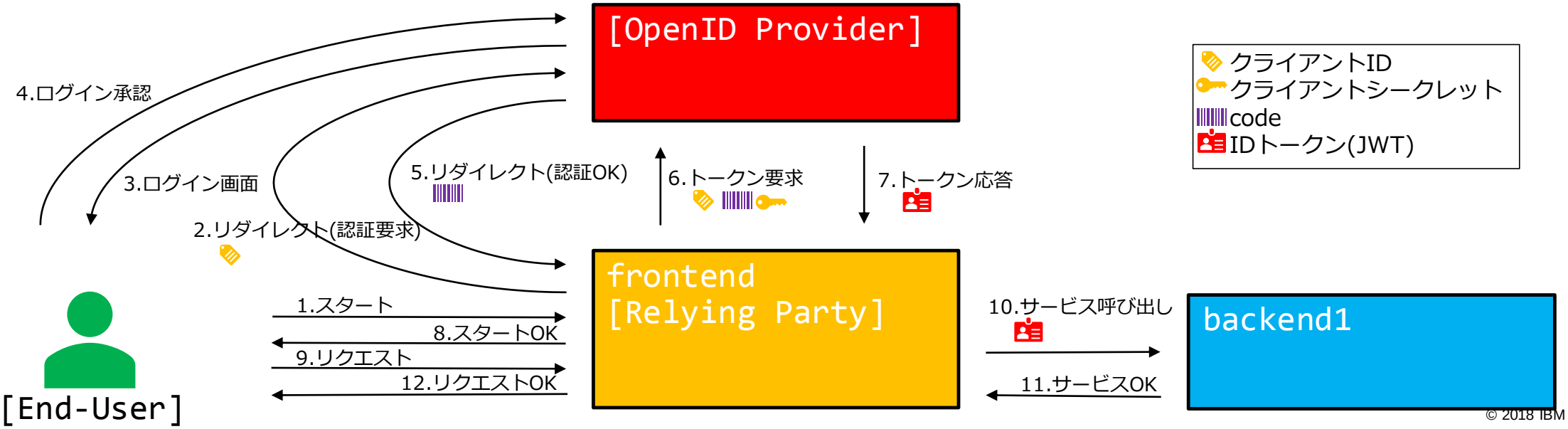
- マイクロサービスにおいて、アクセス元のエンドユーザーの権限に応じて提供するサービスを制御したい場合に使用します。
  - サービスは通常ステートレスであり、呼び出し毎に認証・認可のチェックが必要です。各サービスで認証・認可を行うとサービス数が多くなればなるほど大きな負荷がかかります。
  - そこでmpJwtでは、信頼できるトークンにユーザー情報を載せて伝搬し、各サービスではトークン上の情報を信用して利用することで認証・認可の負荷を軽減します。
    - 信頼できるサーバーで認証を実施して認証結果と認可に必要な情報を含むトークンを作成し、このトークンをリクエスト毎に添付してサービスを呼び出します。
    - mpJwt-1.0フィーチャーは、受け取ったトークンが信頼できるかどうかの検証、トークン上のユーザー情報に基づくアプリケーション実行の認可、アプリケーションからトークン上の情報へのアクセス、次のサービスにトークンを伝搬する機能を提供します。



© 2018 IBM Corporation

OpenID Connect (OIDC)とは

- 複数のアプリケーション間でのシングルサインオンやサードパーティー認証を実現する認可と認証のプロトコルです。
  - 認可の仕組みであるOAuth 2.0を拡張し、エンドユーザー情報を伝達する仕組みを提供します。
    - エンドユーザーの本人証明書としてIDトークン(JWT)を発行します。
    - トークンを安全にやり取りするための規約を規定します。
  - システム構成に応じて使い分けできる以下のFlowが定義されています。
    - Authorization Code Flow　：サーバーサイドアプリ向け（下図の例）
    - Implicit Flow　：端末サイド（モバイル）アプリ向け
    - Hybrid Flow　：端末 ＋ サーバー構成のアプリ向け



© 2018 IBM Corporation

JSON Web Token (JWT)とは

- JSON Web Token (JWT)は、属性情報(Claim)をJSONデータ構造で表現したトークンの標準仕様です。
  - 下図の例は、JSON Web Signature (JWS)仕様に従いJWTに電子署名を施したもので、一般的にJWTと言うとこの形式のものを指します。JWSで電子署名されたJWTは、ピリオドで区切られた3つのパートから構成されます。
    - ヘッダー　：署名アルゴリズムやメタ情報
    - ペイロード　：属性情報(Claim)のセット
    - 電子署名　：ヘッダー+ペイロードに対する電子署名
  - 電子署名の検証に合格すれば、ペイロードに記載された属性情報(Claim)を改ざんのないものとして信用できます。

ヘッダー

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

base64 url encoding

属性情報(Claim)

ペイロード

```
{
  "iss": "joe",
  "exp": 1300819380,
  "http://example.com/is_root": true
}
```

base64 url encoding

JWT

eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt cGxlLmNvbS9pc19yb290Ijp0cnVlfQ.dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk

base64 url encoding

電子署名

ヘッダー＋ペイロードを、ヘッダーのalgクレームに指定のアルゴリズムで署名したもの

© 2018 IBM Corporation

MP-JWTとは

- Eclipse MicroProfileでは、認証・認可に利用するために、JWTに以下の2つの属性情報(Claim)を導入したMP-JWTを定義しています。

クレーム	説明
upn	ユーザーを一意に識別する情報。ユーザー・プリンシパル名
groups	ユーザーの所属グループのリスト。Java EEアプリケーションの役割（ロール）にマップされる

- MP-JWTでは上記2つに加えて、以下の属性情報(Claim)をMP-JWT Required Claimsとしています。

クレーム	説明
typ	コンテンツの種類。MP-JWTの場合は、"JWT"に固定
alg	JWTを保護するために使用される暗号化アルゴリズム
kid	JWTを保護するために使用されたキーを示すヒント
iss	MP-JWTの発行者の識別子
sub	一般的にはJWTが示す対象の識別子。MP-JWTでは、upnクレームが無い場合に代替で使用される
exp	JWTの有効期限。これ以降はこのJWTを処理してはならない
iat	JWTを発行した時刻
jti	JWTのための一意な識別子。JWTが再利用されることを防ぐために利用できる

© 2018 IBM Corporation

ロールベース・アクセス制御(RBAC)とは

- ロールベースアクセス制御とは、ユーザーに特定の役割（ロール）を与え、そのロールに応じて操作を実行する許可（パーミッション）を割り当てる仕組みです。
  - 以下の例では、ユーザー"bob"とユーザー"carol"は、"admin"としての役割を持たないため、"admin"のみを許可するアプリケーション"service1"へのアクセスを許可しません。

ユーザー	アプリケーション / 許可 / 役割	service1	service2
		admin	admin, user
alice	admin, user	○	○
bob	user	×	○
carol	user, other	×	○

- mpJwtでは、
  - MP-JWTのgroupsクレームが、ユーザーのロールを表します。
  - アプリケーションの"@RolesAllowed"アノテーション指定や、ロール・マップ・バインディング設定で、ロールに対してパーミッションを与えます。
  - ロールを持たないユーザーによるサービスの利用は、"HTTP 403 Forbidden"で拒否します。

© 2018 IBM Corporation

フィーチャーの有効化

WAS Liberty上でMicroProfile JWT Propagationの機能を有効化するためには、該当サーバーのserver.xmlにmpJwt-1.0フィーチャーを設定します。

当該フィーチャーが含まれるmicroProfile-1.2やmicroProfile-1.3を設定することも可能です。

```
<featureManager>
        <feature>mpJwt-1.0</feature>
</featureManager>
```

mpJwt-1.0フィーチャーを有効化することにより、以下のフィーチャーが暗黙的に有効化されます。

- cdi-1.2
- distributedMap-1.0
- jndi-1.0
- jsonp-1.0
- jwt-1.0
- servlet-3.1
- ssl-1.0

通常同時に使用するjaxrs-2.0フィーチャーを共に有効化すると、以下のフィーチャーも有効化されます。

- appSecurity-2.0

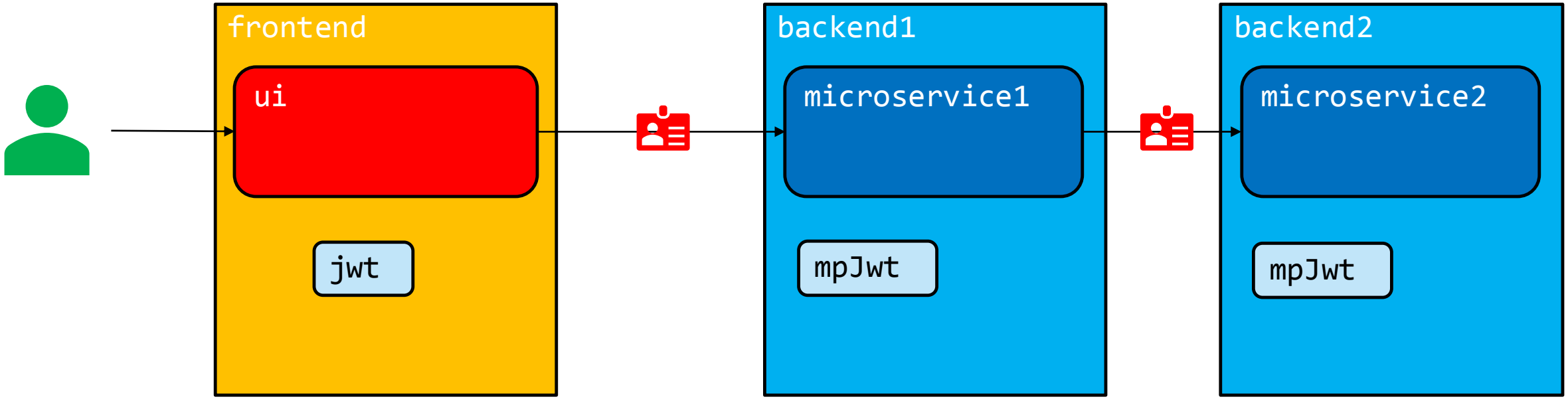
© 2018 IBM Corporation

MicroProfile JWT Propagationの使用方法

© 2018 IBM Corporation

サンプル構成

- このページ以降では、MP-JWTを発行するfrontendのLibertyサーバー、MP-JWTを受け取ってサービスを提供するbackendのLibertyサーバーを構成する場合のサンプルを例示します。



© 2018 IBM Corporation

サーバー設定

- frontendサーバーでJWTを発行し、backendサーバーで伝搬されたJWTを受信する場合のserver.xml設定を以下に例示します。ここで注意すべき点は以下の2点です。
  - JWTの発行者(issuer)が一致していること
  - JWTの署名に使用するアルゴリズム(signatureAlgorithm)が一致していること
    - mpJwt-1.0フィーチャーではRS256とHS256の二つのアルゴリズムが使用できます。
      - RS256 ：RSASSA-PKCS1-v1_5 using SHA-256 ：公開鍵/秘密鍵のペアによって署名と検証を行います
      - HS256 ：HMAC using SHA-256 ：発行者と利用者で共有する秘密鍵によって署名と検証を行います
    - アルゴリズム毎の追加設定については後続の「JWTの署名に使用するアルゴリズム」を参照してください。

frontend server.xml

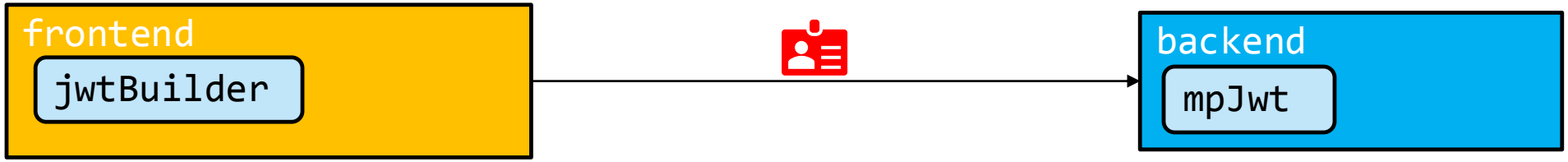
```
<featureManager>
~
    <feature>jwt-1.0</feature>
    <feature>jaxrsClient-2.0</feature>
</featureManager>
~
<jwtBuilder
    id="jwtFrontEndBuilder"
    issuer="frontend.example.com"
~
    signatureAlgorithm="RS256"/>
```

JWT

```
{
  "typ": "JWT",
  "alg": "RS256"
}
{
  ~
  "iss": "frontend.example.com",
  ~
}
~
```

backend server.xml

```
< featureManager>
~
    <feature>mpJwt-1.0</feature>
    <feature>jaxrs-2.0</feature>
</featureManager>
~
<mpJwt
    id="jwtUserConsumer"
    issuer="frontend.example.com"
~
    signatureAlgorithm="RS256"/>
```



© 2018 IBM Corporation

アプリケーション

- frontendサーバー上のアプリケーションでは以下により、backendサーバー上のmpJwt-1.0フィーチャーでJWTを受け取れるようリクエストを送信できます。
  - com.ibm.websphere.security.jwt.JwtBuilder APIを使用し、server.xmlで構成したjwtBuilderでJWTを発行
  - JAX-RSクライアントAPIで"Authorization: Bearer"ヘッダーにトークンを付けてリクエストを実行
- backendサーバー上でJWTを受け取るJAX-RSアプリケーションを保護するには以下のようにします。
  - @RollesAllowedアノテーションで許可するロールを指定

frontend server.xml

```
<jwtBuilder
    id="jwtFrontEndBuilder"
~
/>
```

frontend application

```
String jwt = JwtBuilder.create("jwtFrontEndBuilder")
        .claim("upn", "alice")
        .claim("groups", new String[]{"admin", "user"})
        .buildJwt()
        .compact();

Client client = ClientBuilder.newClient();
WebTarget target =
client.target("https://backend1.example.com:19443/service1");
Response res = target.request(MediaType.APPLICATION_JSON)
        .header(HttpHeaders.AUTHORIZATION, "Bearer " + jwt)
        .get();
```

JWT

```
~
{
  ~
  "upn": "alice",
  "groups": [
    "admin",
    "user"
  ],
  ~
}
~
```

backend application

```
@RequestScoped
@Path("service1")
public class Service1 {
    @GET
    @RolesAllowed({ "admin", "user" })
    @Produces(MediaType.APPLICATION_JSON)
    public Properties getProperties() {
        ~
    }
}
```

送信されるHTTPリクエスト

```
GET /service1 HTTP/1.1
Accept: application/json
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiQmVhcmVyIiwidXBuIjoiYWxpY2UiLCJncm91c
HMiOlsiYWRtaW4iLCJ1c2VyIl0sImlzcyI6ImZyb250ZW5kLmV4YW1wbGUuY29tIiwiZXhwIjoxNTMzNzk0ODMzLCJpYXQiOjE1MzM3ODc2MzN9.N5ge
CEpUL6CTdcd4ZAM_zgrRY4T2TmPo8i9Jz83E6yQ0kXU35ifl0SIdLbTKzRKpqw-AimPh9smIknsWiKIggeXr1YYBkG86WJUcqzxBS0dGkBVN7J4QwAmz
LybI4V9VF1sVyKbfjHVe50w_nubQx7ZB2Oj6UViiZa_YMAKQs7_-PD3iGefN04wT63VjbR8OSAn7jsv0W578MQn4ry-7oM_vuYMBrVTvEOP_Qnc6rxkS
43dPJmmo5R01qOnWY4jKwZaLCAxVQFH3yH-xz0s4Vybl33wowVsJHoDsOsG7iB1HJns1cEpLm3zI1pp6Qe9Uz5PheLlGiCQ2xZ1PtR1B9Q
User-Agent: Apache-CXF/3.1.14
Cache-Control: no-cache
Pragma: no-cache
Host: backend1.example.com:19443
Connection: keep-alive
```


© 2018 IBM Corporation

JWTの署名に使用するアルゴリズム：RS256：鍵ペアを動的生成する場合

- frontendサーバーでJSON Web Key(JWK)を使用するよう構成(jwkEnabled=true)することで、jwtBuilderが動的に生成した鍵ペアをJWTの署名に使用できます。
- backendサーバーでは、frontendサーバーでjwtBuilderを構成すると公開されるJWK APIを指定(jwksUrl)しておくと、mpJwtフィーチャーはこのAPIを使用して署名の検証用の公開鍵を入手します。

frontend server.xml

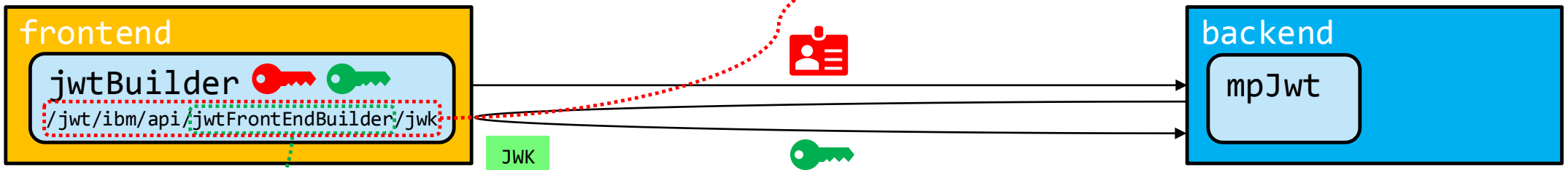
```
<jwtBuilder
    id="jwtFrontEndBuilder"
    issuer="frontend.example.com"
    signatureAlgorithm="RS256"
    jwkEnabled="true"/>
```

JWT

```
{
  "kid": "onS3GLc7XdnGqDLGTdvs",
  "typ": "JWT",
  "alg": "RS256"
}
~
署名
```

backend server.xml

```
<mpJwt
    id="jwtUserConsumer"
    issuer="frontend.example.com"
    jwksUri="https://frontend.example.com:9443/jwt/ibm/api/jwtFrontEndBuilder/jwk"
    signatureAlgorithm="RS256"/>
```



JWK


```
{
  "keys":[
    {
      "kty":"RSA",
      "e":"AQAB",
      "use":"sig",
      "kid":"onS3GLc7XdnGqDLGTdvs",
      "alg":"RS256",
      "n":"nzEKIq4P1lbmLnTsGXSoVxsCB7QjDxPKSATa5yhAiwcZiLLTiUNdLuGrgMG_JzNyVbOpiTtGJ3p46A
05tsJHwEh1zrHu6TRpqjg59bRwaG2H0mbrBikHJZKXtIGfz0P7qh_dU4djz5LrUtdaN4XC81h0RE6wrQDYmgPVk5n
f_cYm9sw2DwAhl31l2_BxY1I5R7m0E6EEQiBzUDY4OaFQLY3QV0y6Zrb20_z22OilauelSLpjdWGduqsd8WAS48V5
7qYiMgca_ySw1zcAii-eTeyw0kBUaaKqi4-b0CknZbkT6hNGPeH0xd1_0WGy2KfVGSVkloQQIlmrDZk4VEdAnQ"
    }
  ]
}
```


© 2018 IBM Corporation

JWTの署名に使用するアルゴリズム：RS256：事前に生成ずみの鍵ペアを使用する場合

- frontendサーバーでJWTの署名に使用する鍵ペアを指定(keyStoreRef, keyAlias)し、backendサーバーのSSL構成で証明書を格納するファイル(trustStore)に公開鍵を追加することで、事前に生成ずみの鍵を使用してJWTの生成・検証ができます。

frontend server.xml

```
<keyStore
    id="jwtKeyStore"
    location="jwt.keystore.p12"
    password="${ssl.keystore.password}"/>

<jwtBuilder
    id="jwtFrontEndBuilder"
    issuer="frontend.example.com"
    keyStoreRef="jwtKeyStore"
    keyAlias="jwt"
    signatureAlgorithm="RS256"/>
```

JWT

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

backend server.xml

```
<ssl
    id="defaultSSLConfig"
    keyStoreRef="defaultKeyStore"
    trustStoreRef="trustStore" />
<keyStore
    id="defaultKeyStore"
    location="ssl.keystore.p12"
    password="${ssl.keystore.password}"/>
<keyStore
    id="trustStore"
    location="ssl.truststore.p12"
    password="${ssl.keystore.password}"/>

<mpJwt
    id="jwtUserConsumer"
    issuer="frontend.example.com"
    keyName="jwt"
    signatureAlgorithm="RS256"/>
```

```
keytool -genkeypair -alias jwt -validity 365 -keyalg RSA -keysize 2048
-keypass secret -storepass secret -storetype pkcs12 -dname
cn=frontend.example.com -keystore jwt.keystore.p12
```

```
keytool -exportcert -alias jwt -storepass secret -storetype pkcs12
-keystore jwt.keystore.p12 -file jwt.crt
```

```
keytool -importcert -alias jwt -storepass secret -storetype pkcs12
-keystore ssl.truststore.p12 -file jwt.crt -noprompt
```


© 2018 IBM Corporation

JWTの署名に使用するアルゴリズム：HS256

- frontendサーバーとbackendサーバーで共有する共通鍵を指定(sharedKey)することで、JWTの署名と検証ができます。

frontend server.xml

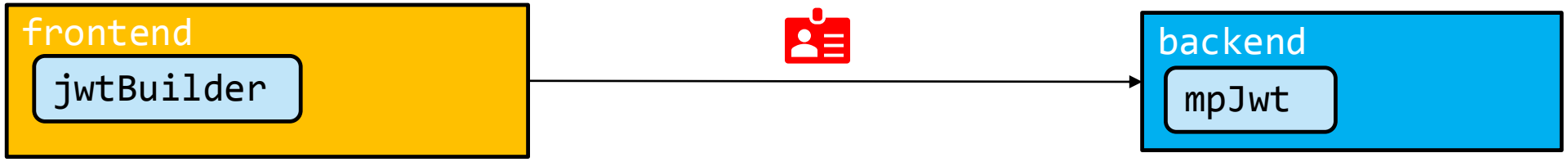
```
<jwtBuilder
    id="jwtFrontEndBuilder"
    issuer="frontend.example.com"
    signatureAlgorithm="HS256"
    sharedKey="{xor}Lz4sLCgwLTs="/>
```

JWT

```
{
  "typ": "JWT",
  "alg": "HS256"
}
~
署名
```

backend server.xml

```
<mpJwt
    id="jwtUserConsumer"
    issuer="frontend.example.com"
    signatureAlgorithm="HS256"
    sharedKey="{xor}Lz4sLCgwLTs="/>
```



© 2018 IBM Corporation

JWTクレームの取得：SecurityContext

- JAX-RSのコンテナーが注入するjavax.ws.rs.core.SecurityContextのgetUserPrincipal()で取得するユーザー・プリンシパルがorg.eclipse.microprofile.jwt.JsonWebTokenのインスタンスになります。
- アプリケーションは JsonWebTokenのgetterで属性値(Claim) にアクセスできます。

```
@RequestScoped
@Path("service1")
public class Service1 {

    @GET
    @RolesAllowed({ "admin", "user" })
    @Produces(MediaType.APPLICATION_JSON)
    public Properties getProperties(@Context SecurityContext sec) {

        Principal user =sec.getUserPrincipal();
        if (user instanceof JsonWebToken) {
            JsonWebToken jwt = (JsonWebToken) user;
            String name = jwt.getName();
            String rawToken = jwt.getRawToken();
            String issuer = jwt.getIssuer();
            Set<String> audience = jwt.getAudience();
            String subject = jwt.getSubject();
            String tokenID = jwt.getTokenID();
            long expirationTime = jwt.getExpirationTime();
            long issuedAtTime = jwt.getIssuedAtTime();
            Set<String> groups = jwt.getGroups();
            Set<String> claimNames = jwt.getClaimNames();
        }
        ~
    }
}
```


© 2018 IBM Corporation

JWTクレームの取得：インジェクション

- CDIインジェクションを使用して、フィールドにorg.eclipse.microprofile.jwt.JsonWebTokenを注入し、属性値(Claim)にアクセスできます。
- もしくは、org.eclipse.microprofile.jwt.Claim限定子を使って属性値(Claim)を注入することもできます。
  - Claim名の指定はorg.eclipse.microprofile.jwt.Claims列挙型の値を使うか、文字列で指定できます。
  - 注入先の型は、String, Set<String>, Long, Boolean, JsonValueなどのJSON-P型、これらをラップするOptional, org.eclipse.microprofile.jwt.ClaimValueが使用できます。

```java
@RequestScoped
@Path("service2")
public class Service2 {

    @Inject
    private JsonWebToken jwt;

    @GET
    @RolesAllowed({ "admin", "user" })
    @Produces(MediaType.APPLICATION_JSON)
    public Properties getProperties() {

        String name = jwt.getName();
        String rawToken = jwt.getRawToken();
        String issuer = jwt.getIssuer();
        Set<String> audience = jwt.getAudience();
        String subject = jwt.getSubject();
        String tokenID = jwt.getTokenID();
        long expirationTime = jwt.getExpirationTime();
        long issuedAtTime = jwt.getIssuedAtTime();
        Set<String> groups = jwt.getGroups();
        Set<String> claimNames = jwt.getClaimNames();
        ~
    }
}
```

```java
@Inject
@Claim(standard = Claims.iat)
private Long issuedAt;

@Inject
@Claim(standard = Claims.raw_token)
private ClaimValue<String> rawTokenCV;

@Inject
@Claim("jti")
private ClaimValue<Optional<String>> optJTI;

@Inject
@Claim(standard = Claims.iat)
private ClaimValue<Long> issuedAtCV;

@Inject
@Claim("custom-missing")
private ClaimValue<Optional<Long>> custom;

@Inject
@Claim(standard = Claims.iat)
private Instance<Long> providerIAT;

@Inject
@Claim("roles")
private JsonArray jsonRoles;
```


© 2018 IBM Corporation

サービスからサービスへのJWTの伝搬

- backend1サーバーのservice1から、backend2サーバーのservice2を呼び出すときに、service1に伝搬されたJWTをservice2にも自動的に伝搬できます。
  - backend1サーバーのserver.xmlに、authnToken=mpjwtを指定してservice2をwebTargetとして登録
  - service1からservice2の呼び出しを通常通りJAX-RS 2.0クライアントAPIを使用して実装
- これにより、service1が受け取ったJWTが、service2の呼び出し時にもHTTPリクエストの許可ヘッダーに追加されます。

backend1 server.xml

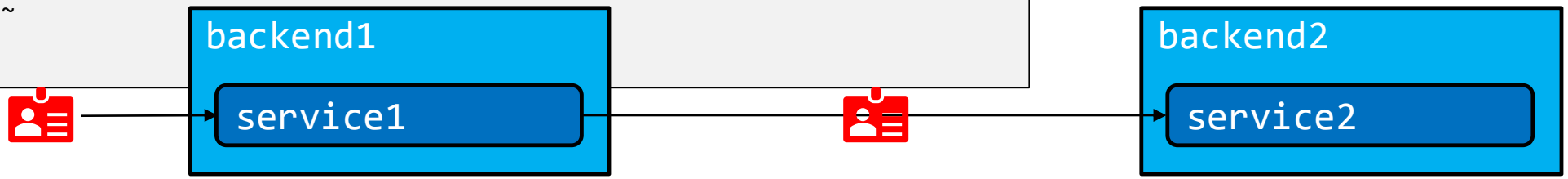
```
<webTarget uri="https://backend2.example.com:29443/service2" authnToken="mpjwt" />
```

backend1 application

```
@RequestScoped
@Path("service1")
public class Service1 {
    @GET
    @RolesAllowed({ "admin", "user" })
    @Produces(MediaType.APPLICATION_JSON)
    public Properties getProperties() {
        ~
        Client client = ClientBuilder.newClient();
        WebTarget target = client.target("https://backend2.example.com:29443/service2");
        Response res = target.request(MediaType.APPLICATION_JSON).get();
        ~
    }
}
```

backend2 application

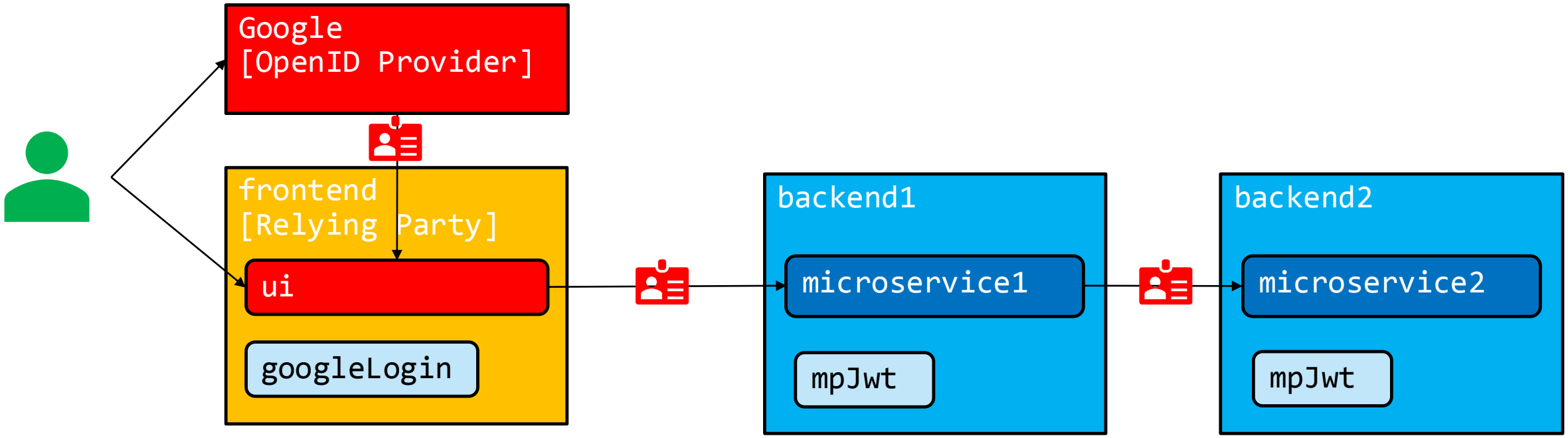
```
@RequestScoped
@Path("service2")
public class Service1 {
    @GET
    @RolesAllowed({ "admin", "user" })
    @Produces(MediaType.APPLICATION_JSON)
    public Properties getProperties() {
        ~
    }
}
```



© 2018 IBM Corporation

サードパーティーが発行したJWTの使用

- このページ以降では、サードパーティーが発行したIDトークン(JWT)を受け取ってサービスを提供するように、backendのLibertyサーバーを構成する場合のサンプルを例示します。
  - ここでは、GoogleのOpenID Connectサービスを利用します。
  - LibertyのsocialLogin-1.0フィーチャーを使用してGoogle IDでログオンできるようにfrontendサーバーを構成します。
  - Googleから発行されたIDトークン(JWT)をbackendサーバーに伝搬してそれをmpJwtで受け取り、認可を行います。



© 2018 IBM Corporation

OpenID Provider (Google)にRelying Party (frontendサーバー)を登録する(1/2)

- 文書末に記載の参考リンクにある「WAS V8.5.5 OpenID Connect 認証構成ガイド」を参考に、Google API Consoleで「クライアントID」と「クライアントシークレット」を用意します。
  - 「4. WASにおけるOpenID Connect の実装および構成方法」>「OpenID Provider (Google)の構成」を参照

1. プロジェクトの作成
   - https://console.developers.google.com/ にアクセスします。
   - 既存のプロジェクトを使用するか、新しいプロジェクトを作成します。

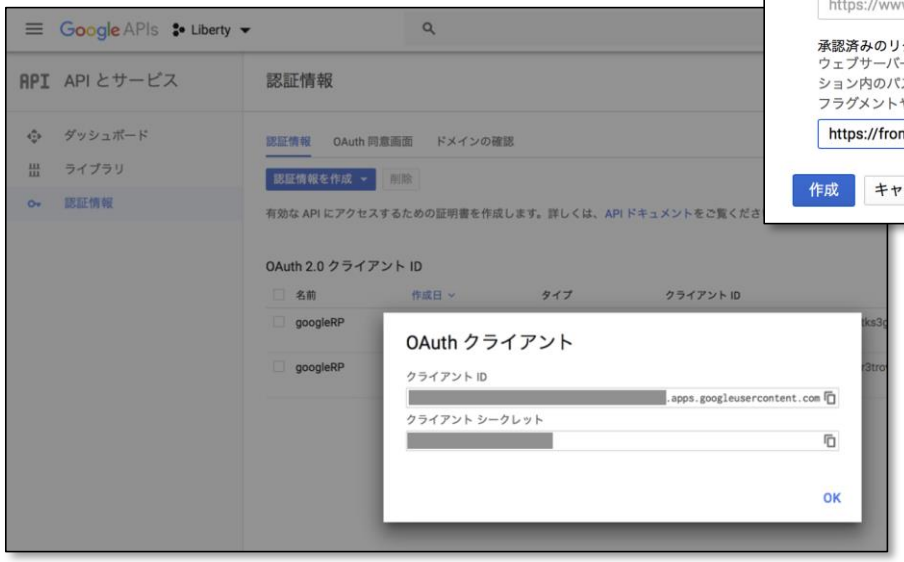
2. クライアントIDの作成
   - 「APIとサービス」>「認証情報」>「認証情報を作成」>「OAuthクライアントID」をクリックします。

© 2018 IBM Corporation

OpenID Provider (Google)にRelying Party (frontendサーバー)を登録する(2/2)

2. クライアントIDの作成（続き）
   - 「ウェブアプリケーション」にチェックを入れ、「認証済みのリダイレクトURI」に以下を入力して「作成」をクリックします。
     - https://*liberty_host:SSL_port*/ibm/api/social-login/redirect/googleLogin

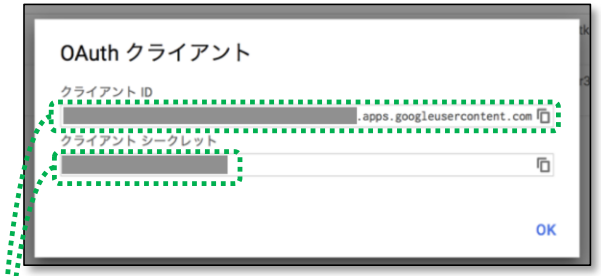
   - 「クライアントID」および「クライアントシークレット」が表示されます。



© 2018 IBM Corporation

Relying Party (frontendサーバー)の設定とアプリケーション

- frontendサーバーにGoogle IDでログオンし、GoogleのOpenID Providerから入手したIDトークンをbackendサーバーに伝搬する場合の、server.xml設定とアプリケーションを以下に例示します。
  - server.xmlにsocialLogin-1.0フィーチャーとjaxrsClient-2.0フィーチャーを追加する
  - server.xmlにgoogleLoginエレメントを追加し、入手したクライアントIDとクライアントシークレットを設定する
  - UserProfileManager、UserProfileのAPIを使用してログオン時に発行されたIDトークンを取得する
  - JAX-RSクライアントAPIで"Authorization: Bearer"ヘッダーにIDトークンを付けてリクエストを実行



frontend server.xml

```
<featureManager>
~
    <feature>jaxrsClient-2.0</feature>
    <feature>socialLogin-1.0</feature>
</featureManager>

<googleLogin
    clientId="**********-**********************************.apps.googleusercontent.com"
    clientSecret="**********************">
</googleLogin>
```

frontend application

```
com.ibm.websphere.security.social.UserProfile profile =
com.ibm.websphere.security.social.UserProfileManager.getUserPr
ofile();
String jwt = profile.getIdToken().compact();

Client client = ClientBuilder.newClient();
WebTarget target =
client.target("https://backend1.example.com:19443/service1");
Response res = target.request(MediaType.APPLICATION_JSON)
        .header(HttpHeaders.AUTHORIZATION, "Bearer " + jwt)
        .get();
```


© 2018 IBM Corporation

backendサーバーの設定：issuer属性とaudiencesエレメント

- GoogleのOpenID Providerが発行したIDトークンをmpJwtで受け取るには、JWKで入手する証明書による署名の検証と、issクレームとaudクレームの検証に合格する必要があります。
  - issクレーム　　：JWTの発行者（ここでは"https:/accounts.google.com"）
  - audクレーム　　：JWTの利用が想定される利用者の識別子一覧（ここでは「クライアントID」）
    - サードパーティー認証の場合、OpenID Providerは別の利用者に対してもIDトークンを発行することがあるため、誰が発行したかだけでなく、誰に対して発行したIDトークンかをaudクレームを見て検証する必要があります。

JWT

```
{
  "alg": "RS256",
  ~
}
{
  ~
  "aud": "■■■■■■■■■■■-■■■■■■■■■■■■■■■■■■■■■.apps.googleusercontent.com",
  ~
  "email": "■■■■■■■■■■■■■■@gmail.com",
  ~
  "iss": "https://accounts.google.com",
  ~
}
```

backend server.xml

```
<featureManager>
<feature>mpJwt-1.0</feature>
<feature>jaxrs-2.0</feature>
</featureManager>

<mpJwt
    id="jwtUserConsumer"
    issuer="https://accounts.google.com"
    jwksUri="https://www.googleapis.com/oauth2/v3/certs">
    <audiences>
        ■■■■■■■■■■■-■■■■■■■■■■■■■■■■■■■■■.apps.googleusercontent.com
    </audiences>
</mpJwt>
```


© 2018 IBM Corporation

backendサーバーの設定：ユーザー・プリンシパル名、グループのマッピング変更

- MP-JWTフォーマットではないGoogleのOpenID Providerが発行したIDトークンを、ユーザー・プリンシパル名やグループにマップします。
  - userNameAttribute属性でユーザー・プリンシパル名用のクレームを、groupAttribute属性でグループ名用のクレームを指定できます。
  - mapToUserRegistry属性に"true"を指定することで、構成されたユーザー・レジストリーからユーザー・プリンシパル名に基づいてセキュリティー・サブジェクトを作成できます。
  - 以下の例では、"email"クレームをユーザー・プリンシパル名にマッピングし、Libertyのベーシック・レジストリーで"■■■■■■■■■■■■■■@gmail.com"ユーザーを"user"グループにマッピングしています。

JWT

```
~
{
  ~
  "email": "■■■■■■■■■■■■■■@gmail.com",
  ~
}
```

backend server.xml

```
<mpJwt
    ~
    userNameAttribute="email"
    mapToUserRegistry="true">
    ~
</mpJwt>

<basicRegistry>
    <user name="■■■■■■■■■■■■■■@gmail.com" password="dummy" />
    <group name="admin">
    </group>
    <group name="user">
        <member name="■■■■■■■■■■■■■■@gmail.com"/>
    </group>
</basicRegistry>
```


© 2018 IBM Corporation

参考リンク

- Eclipse MicroProfile Interoperable JWT RBAC 1.0
  https://github.com/eclipse/microprofile-jwt-auth/files/1305001/microprofile-jwt-auth-spec-1.0.pdf
- IBM Knowledge Center - MicroProfile JSON Web トークンの構成
  https://www.ibm.com/support/knowledgecenter/ja/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/twlp_sec_json.html
- Securing microservices with JSON Web Tokens
  https://openliberty.io/guides/microprofile-jwt.html

- OpenID Connect
  http://openid.net/connect/
- JSON Web Token (JWT)
  https://tools.ietf.org/html/rfc7519
- WASV8.5.5 OpenID Connect 認証構成ガイド
  https://www.ibm.com/developerworks/jp/websphere/library/was/was855_oidcdev_guide/index.html
- IBM Knowledge Center - Liberty でのソーシャル・ログインの構成
  https://www.ibm.com/support/knowledgecenter/ja/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/twlp_sec_sociallogin.html

© 2018 IBM Corporation

End of File

© 2018 IBM Corporation