# Securing Your Application with OIDC and JWT

IBM

# Contents

# 1 Introduction

In this lab we will take an existing application and transform it so that the frontend supports login with OpenID Connect (OIDC), while the backends are secured by JSON Web Tokens (JWT). The existing application is composed of a frontend web application secured via JEE security and two unsecured backend REST-ful services.

**Note:** This lab is suitable for both developers and administrators. There is no requirement to use WebSphere Developer Tools (WDT).

For this lab, the various passwords are:

1. OIDC client registration: client ID: **rp**, client secret: **secret**
2. Key store password: **labPassword**
3. User IDs and passwords to login to the application:
    a. user: **user1**, password: **user1**
    b. user: **user2**, password: **user2**

Please refer to the following table for file and resource location references on different operating systems.

| Location Ref. | OS | Absolute Path |
|---|---|---|
| *{LAB_HOME}* | Windows | `C:\WLP_<version>` |
| | Linux | `~/WLP_<version>`<br>`Or your choice` |
| | Mac OSX | `~/WLP_<version>`<br>`Or your choice` |

# 2 Getting to Know the Existing Sample

The existing sample is composed of three different applications:

- Frontend: the frontend web application that calls the other two service.
- Banking: a Jax-RS service for processing bank account.
- Credit: a JAX-RS service for processing credit card account.

The source code for the applications are stored with the .war file of the applications.

## 2.1 Running the Sample

To run the sample:

__1. Copy the following directories to {LAB_HOME}/wlp/usr/servers directory

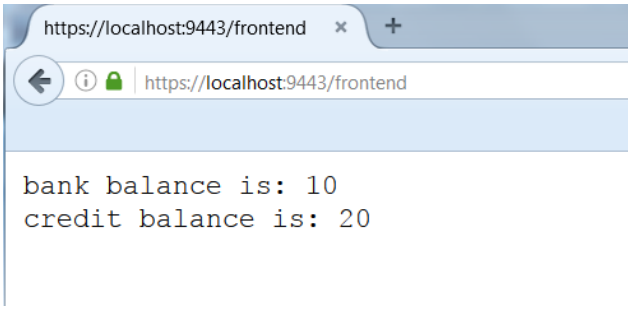
- **Banking**
- **Credit**
- **Frontend**

__2. Change directory to {LAB_HOME}/wlp/bin

__3. Start the servers

**server start Banking**
**server start Credit**
**server start Frontend**

__4. Point your browser to: **http://localhost:9080/frontend/** . Login with user **user1** and password **user1**. After login, you should see the numbers returned from the two back end services:




__5. Stop the servers

```
server stop Banking
server stop Credit
server stop Frontend
```

## 2.2 The Frontend Application

The Frontend is a web application that calls two backend JAX-RS services: Banking and Credit. It gets the URL of the backend services from the environment. Currently the values of the environment variables are stored in server.env:

```
BANKING_SERVICE_LOCATION=http://localhost:9081
CREDIT_SERVICE_LOCATION=http://localhost:9082
```

The Frontend is secured through JEE security. Currently it allows all authenticated users, as shown in its web.xml:

```
        <security-constraint>
                <web-resource-collection>
                        <web-resource-name>all</web-resource-name>
                        <url-pattern>/*</url-pattern>
                </web-resource-collection>
                <auth-constraint>
                        <role-name>**</role-name>
                </auth-constraint>
                <user-data-constraint>
                        <transport-guarantee>CONFIDENTIAL</transport-
guarantee>
                </user-data-constraint>
        </security-constraint>
```

The user registry for the Frontend application is the built-in basic registry, as defined in server.xml:

```
    <basicRegistry>
        <user name="user1" password="{xor}Kiw6LW4="/>
        <user name="user2" password="{xor}Kiw6LW0="/>
    </basicRegistry>
```

### 2.2.1 (Optional) Review the Frontend Source Code

The relevant source code for the frontend application is:

```
  32          protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException,
IOException {
    33                  // TODO Auto-generated method stub
    34                  try {
    35                          Client client = ClientBuilder.newClient();
    36                          String target =
System.getenv("BANKING_SERVICE_LOCATION");
    37                          String userName =
request.getUserPrincipal().getName();
    38                       target += "/banking/account/balance/"+userName;
    39                          WebTarget myResource = client.target(target);
    40                          String ret =
myResource.request(MediaType.TEXT_PLAIN).get(String.class);
```

```
    41                      response.getWriter().append("bank balance is: " + ret +
"\n");
    42
    43                          Client client2 = ClientBuilder.newClient();
    44                      String target2 =
System.getenv("CREDIT_SERVICE_LOCATION");
    45                      target2 += "/credit/account/balance/"+userName;
    46                          WebTarget myResource2 = client2.target(target2);
    47                          String ret2 =
myResource2.request(MediaType.TEXT_PLAIN).get(String.class);
    48                      response.getWriter().append("credit balance is: " +
ret2);
    49                  } catch (Exception ex) {
    50                          response.getWriter().append("unable to call service:
" + ex);
    51                      ex.printStackTrace();
    52                  }
    53
    54                  // response.getWriter().append("Served at:
").append(request.getContextPath());
    55          }
```

Note that:

- Line 36: The application retrieves the URL for the banking service from the environment
- Line 37: The application gets the logged in user's name from the Principal.
- Line 38: The application builds the URL for the REST call
- Line 39: The application calls the Banking service
- Line 40: The application appends the result to the response.
- Lines 41-49: The same repeats for the Credit service

## 2.3 Banking Application

The Banking application is just a simple JAX-RS application that returns the number 10. It is currently not secure.

### 2.3.1 (Optional) Review the Banking Service Source Code

```
@Path("account")
public class BankingServiceResource<E> {
    @GET
    @Path("balance/{id}")
    @Produces(MediaType.TEXT_PLAIN)
    public String balance(@PathParam("id") String id){
        return "10";
    }
}
```

## 2.4 Credit Application

The Credit application is just a simple JAX-RS application that returns the number 20. It is currently not secure.

### 2.4.1 (Optional) Review the Credit Service Source Code

```
@Path("account")
public class CreditServiceResource<E> {
    @GET
    @Path("balance/{id}")
    @Produces(MediaType.TEXT_PLAIN)
    public String balance(@PathParam("id") String id){
        return "20";
    }
}
```

# 3 Adding OIDC and JWT

In this section we will show how to add OIDC support to the frontend application, and JWT to secure the backend services. The advantages of adding OIDC support include:

- Allowing users to login with their social media account, such as Google, or Facebook.
- Single sign-on across multiple applications
- Delegating user and password management and multi-factor authentication to the OIDC provider.

The advantages of adding JWT support include:

- No session or cookies required. Can be called directly even by front end or mobile applications or javascript in browser
- Portable across multiple applications
- Self-contained with all authentication information and expiration.

To support OIDC, no code change is required to the frontend application. Only server configuration changes are required.

To secure the backend via JWT, in addition to server configuration changes, two application changes are made:

- The frontend application is changed to add a JWT header when making a REST call. This code change is not required if the application only wants to pass along the JWT token it received from the OpenID Provider in a JAX-RS call. This sample demonstrates the scenario where the application self-issues a new JWT, which requires a code change.
- The backend application's deployment descriptor is changed to enable security.

Let's first run the modified applications before examining the changes in detail.

## 3.1 Running the Modified Sample

To run the modified sample:

__1. Ensure the existing samples are stopped, as they use the same port numbers as the modified samples.

```
server stop Banking
server stop Credit
server stop Frontend
```

__2. Copy the following directories to {LAB_HOME}/wlp/usr/servers directory

- **BankingJWT**
- **CreditJWT**
- **FrontendOIDC**
- **OP**

Note that the new OP server hosts Liberty's built-in OpenID Connect Provider. You can test applications secured by OIDC or JWT using the built-in OP, rather than having to create a test account in an external provider, such as your corporate OP, or a social medial OP such as Google or Facebook.

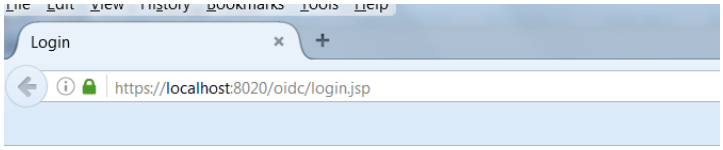
__3. Change directory to {LAB_HOME}/wlp/bin.

__4. Start the servers

**server start OP**
**server start BankingJWT**
**server start CreditJWT**
**server start FrontendOIDC**

__5. Point your browser to: **http://localhost:9080/frontend/** .

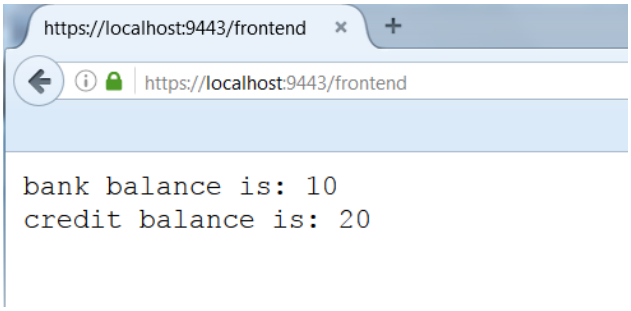
__a. Look at the browser's URL and note that you have been redirected to the OP's login page at: https://localhost:8020/oidc/login.jsp. Login with user **user1** and password **user1**.





__b. At the next page, the OP prompts you to select what claims you want to make available to the application. The claims, such as openid, profile, or email, are attributes stored by the OP, and not released to the calling application without your permission. For a real application, the claims will include personal information such as your name, address, and email. For this sample, there is no real data associated with these claims. So just click "Allow, remember my decision" to proceed.


__c. You should be redirected back to the application, where the same results from the two backend services are displayed:




__6. Stop the servers

```
server stop BankingJWT
server stop CreditJWT
server stop FrontendOIDC
server stop OP
```

## 3.2 Changes to Enable OIDC

Two steps are required to enable OIDC for the Frontend Application

- Register the application as a relying party (RP) on the OIDC provider (OP)
- Configuring the application as a OIDC client in the application's server.xml

### 3.2.1 Configuring the OP

You need to register your application to your OP of choice. For the Liberty built-in OP, you may register in two ways:

- By adding entries to server.xml, as shown below in this lab.
- By calling a REST API to perform the registration, and by configuring a database to persist the changes. This is beyond the scope of this lab.

Open the OP's server.xml in an editor, and note its configuration:

```
 1  <server description="new server">
 2
 3      <!-- Enable features -->
 4        <featureManager>
 5         <feature>openidConnectServer-1.0</feature>
 6         <feature>appSecurity-2.0</feature>
 7         <feature>servlet-3.0</feature>
 8         <feature>localConnector-1.0</feature>
 9      </featureManager>
10
11     <basicRegistry id="basic" realm="OpBasicRealm">
12         <user name="user1" password="user1"/>
13         <user name="user2" password="user2"/>
14     </basicRegistry>
15
16     <openidConnectProvider id="OP" jwkEnabled="true" oauthProviderRef="Oauth"
signatureAlgorithm="RS256"/>
17
18     <keyStore id="defaultKeyStore" password="labPassword"/>
19
20     <httpEndpoint httpPort="8010" httpsPort="8020" id="defaultHttpEndpoint" host="*"/>
21
22      <oauth-roles>
23          <authenticated>
24              <special-subject type="ALL_AUTHENTICATED_USERS" />
25          </authenticated>
26      </oauth-roles>
27
28     <oauthProvider httpsRequired="true" id="Oauth" tokenFormat="mpjwt" >
29         <localStore>
30              <client displayname="rp" name="rp" secret="{xor}LDo8LTor"
31                  scope="openid profile scope1 email phone address" enabled="true" >
32                  <redirect>https://localhost:9443/oidcclient/redirect/RP</redirect>
33              </client>
34             </localStore>
35     </oauthProvider>
36
37  </server>
```

Note that:

- Line 5: uses the openidConnectServer feature
- Lines 11-14: defines the basic user registry. The user registry no longer resides in the front-end application, but in the OP. The OP may also be configured to use other user registries, such as LDAP.
- Line 16 defines this server as an OIDC provider. The jwkEnabled attribute, if set to true, enables the provider to generate encryption keys following the JSON Web Key (JWK) Protocol. The OauthProviderRef refers to client registration defined at line 28.

- Line 22 defines the roles of those users who logged in that are also allowed to be authenticated via OIDC. Normally you want to allow all authenticated users to be authenticated via this OP.
- Line 28 starts the client registrations.
    - Note that the token format is “mpjwt”. The microprofile JWT format defines required signature algorithm and a set of standard claims. It can be used as the format for the JWT token, even if you don’t use the microprofile JWT feature in your code.
    - The use of “localStore” means the client registration is stored locally in server.xml, instead of being persisted to a database.
- Line 30 starts the actual client registration. Note the name and secret attributes must match the ClientID and secret attributes in the server.xml of the frontend application. We will point this out when reviewing the frontend.
- Line 31: The scope declares those claims that may be made available to the users of this application. The user is still free to choose which claims to give to the application, as we have seen when running the application.
- Lines 32 specifies the valid redirect URLs. The OP will only redirect the results of the login to valid redirect URLs. More than one may be specified to allow more than one application to share the same registration.

### 3.2.2 Configuring the Frontend

The Frontend application serves as the Relying Party (RP) to the OP. Only changes to the Frontend’s server.xml is required to enable OIDC. Use an editor to open the server.xml in the FrontendOIDC directory.

```
 1  <server description="new server">
 2
 3      <!-- Enable features -->
 4      <featureManager>
 5          <feature>jsp-2.3</feature>
 6          <feature>localConnector-1.0</feature>
 7          <feature>jaxrs-2.0</feature>
 8          <feature>appSecurity-2.0</feature>
 9          <feature>jwt-1.0</feature>
10          <feature>openidConnectClient-1.0</feature>
11      </featureManager>
12
13      <!-- To access this server from a remote client add a host attribute to the
following element, e.g. host="*"
 -->
14      <httpEndpoint httpPort="9080" httpsPort="9443" id="defaultHttpEndpoint"
host="localhost"/>
15
16
17      <keyStore password="{xor}Mz49Dz4sLCgwLTs="></keyStore>
18
19      <!-- Automatically expand WAR files and EAR files -->
20      <applicationManager autoExpand="true"/>
21
22
23      <applicationMonitor updateTrigger="mbean"/>
24
25      <webApplication contextRoot="/frontend" id="BankingFrontendOIDC"
location="BankingFrontendOIDC.war" name="Ba
```

```
nkingFrontendOIDC"/>
    26
    27      <webAppSecurity  logoutOnHttpSessionExpire="false"  ssoCookieName="RP" />
    28
    29      <openidConnectClient id="RP" signatureAlgorithm="RS256"
    30             scope="openid profile email"
    31             isClientSideRedirectSupported="false"
    32             clientId="rp"
    33             clientSecret="secret"  sharedKey="secret"
    34             discoveryEndpointUrl="https://localhost:8020/oidc/endpoint/OP/.well-
known/openid-configuration"
    35             />
    36
    37      <jwtBuilder id="defaultJWT" keyAlias="default"  />
    38
    39  </server>
```

Note that:

- The basic registry is no longer defined. The application now relies on the registry of the OP.
- Line 10 enables the openidConnectClient feature
- Line 29 configures the openidConnectClient. This enables using OIDC for login.
- Line 30 defines those claims that this application will ask the OP to return. Note that the OP will only return those claims that are configured to be available in the registration defined in the OP, and only if approved by the user during login.
- Line 31 specifies that this client does not rely on client side redirect. We will only rely on server side redirect for our sample. You will need client side redirect if you are running a single page application with javascript on the browser. Instead of the RP redirecting you to the OP, a Javascript is returned to the browser to perform the redirect.
- Lines 32 and 33 define the client ID and client secret. These must match the registration in the OP.
- Line 34 defines the discovery endpoint in the OP. The RP calls this endpoint to discover additional URLs required to support OIDC login.

## 3.3 Changes to Enable JWT To Secure the Backend

The changes made to enable backend service include:

- Configuring Frontend's server.xml and server.env
- Changing Frontend's application to issue a new JWT
- Configuring backend services' deployment descriptors to enable security
- Configuring backend services' server.xml to support JWT

### 3.3.1 Changing the Frontend configuration and server.env

Refer the server.xml in section 3.2.2 and note that:

- Line 9 enables the jwt-1.0 feature for the Frontend to build and issue JWT.

- Line 37 specifies which key in the key store is to be used to sign the JWT. There are multiple options to get a JWT. For our sample, the Frontend creates and issues its own JWT, which gives it more flexibility to add more claims if needed. As an issuer, the front needs to sign the JWT with the configured key. Another option, which does not require code change, is to automatically pass along the JWT received from the OP. This scenario is not covered this lab. A third scenario, also not covered by this lab, is to configure the frontend application to use the JWK protocol to generate signing keys dynamically.

Since the backend services are now secured, their URLs have also changed to use https. Note the changes in server.env:

```
BANKING_SERVICE_LOCATION=https://localhost:9444
CREDIT_SERVICE_LOCATION=https://localhost:9445
```

### 3.3.2 Changing the Frontend application

The change to the frontend application to call Banking service looks like:

```
    …
    String headerValue = "Bearer " + createJwtFromSubject(userName, "BankingService");
    …
    String ret = myResource.request(MediaType.TEXT_PLAIN).header("Authorization",
headerValue).get(String.class);
```

Before calling the Banking service, it first calls createJwtFromSubject method to create a JWT token. It then adds the JWT token to the request as an “Authorization” header. The same code change is used to call Credit service.

The createJwtFromSubject method looks like:

```
    73              private static String createJwtFromSubject(String userName, String
audience){
    74                  String jwt = null;
    75                  try {
    76                      JwtToken token = JwtBuilder.create().subject(userName).
    77                              claim("email", userName + "@client.com").
    78                              claim("role", "impersonator").
    79                              claim("scope", "server client").
    80                              claim("aud", audience)
    81                              .buildJwt();
    82                      jwt = token.compact();
    83                  } catch (Exception e){
    84
    85                  }
    86
    87                  return jwt;
    88
    89              }
```

Note that the JWTBuilder helper class is used to build the JWT. The email claim is auto-generated as an example. And the audience claim is set to the name of the service. The backend server can be configured to only accept JWT containing specific audience, which we will point out later.

### 3.3.3 Changing the Banking Service

There is one code change to the Banking service as an example to show how to fetch the JWT token. Applications can use the claims in the JWT token for further processing. Note the use of PropagationHelper class to fetch the JWT. Though not shown here, you can also use the microprofile JWT feature to inject the token automatically through annotation.

```
@Path("account")
public class BankingServiceResource<E> {
    @GET
    @Path("balance/{id}")
    @Produces(MediaType.TEXT_PLAIN)
    public String balance(@PathParam("id") String id){
        // sample code to get the id token. From here, you can get all the claims
        // if you want to to do application specific authorization
        IdToken idToken = PropagationHelper.getIdToken();
        System.out.println("id toke: " + idToken);

        return "10";
    }
}
```

The web.xml deployment descriptor is modified to add security constraints:

```
        <security-constraint>
                <web-resource-collection>
                        <web-resource-name>all</web-resource-name>
                        <url-pattern>/*</url-pattern>
                </web-resource-collection>
                <auth-constraint>
                        <role-name>**</role-name>
                </auth-constraint>
                <user-data-constraint>
                        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
                </user-data-constraint>
        </security-constraint>
```

### 3.3.4 Configuring the Banking service

The server.xml for the Banking service now looks like:

```
1  <server description="new server">
2
3      <!-- Enable features -->
4      <featureManager>
5          <feature>jsp-2.3</feature>
6          <feature>localConnector-1.0</feature>
7          <feature>jaxrs-1.1</feature>
8          <feature>appSecurity-2.0</feature>
9          <feature>openidConnectClient-1.0</feature>
```

```
    10          <feature>jwt-1.0</feature>
    11      </featureManager>
    12
    13      <!-- To access this server from a remote client add a host attribute to the
following element, e.g. host="*"
 -->
    14      <httpEndpoint httpPort="9081" httpsPort="9444" id="defaultHttpEndpoint"
host="localhost"/>
    15
    16      <keyStore password="{xor}Mz49Dz4sLCgwLTs="></keyStore>
    17
    18      <!-- Automatically expand WAR files and EAR files -->
    19      <applicationManager autoExpand="true"/>
    20
    21
    22      <applicationMonitor updateTrigger="mbean"/>
    23
    24      <webApplication contextRoot="/banking" id="BankingServiceJWT"
    25          location="BankingServiceJWT.war" name="BankingServiceJWT">
    26      </webApplication>
    27
    28      <webAppSecurity  logoutOnHttpSessionExpire="false"  ssoCookieName="RS" />
    29
    30       <openidConnectClient id="RP" signatureAlgorithm="RS256"
    31            inboundPropagation="required"
    32            audiences="BankingService"
    33            issuerIdentifier="https://localhost:9443/jwt/defaultJWT"
    34            trustStoreRef="defaultKeyStore"
    35            trustAliasName="frontend" >
    36        </openidConnectClient>
    37  </server>
```

Note that:

- Lines 8-10 enable the application security, OpenID Connect client, and jwt features.
- Line 28 allows the client to stay signed in even after http session expires, since a JWT token is not tied to a http session. The token will still expire upon token expiration. It also changes the name of the cookie from the default “ltpatoken2”.
- Line 30 starts the configuration for OpenID Connect client
- Line 31 requires inbound requests to contain a JWT token
- Line 32 restricts acceptable JWTs only to those whose audience is “BankingService”
- Line 33 restricts acceptable JWTs only to those whose issuing identifier matches.
- Lines 34 and 35 identifies the reference to the trust store and the name of the certificate used to verify the JWT token. In this case, it’s the signer of the Frontend’s certificate.

### 3.3.5 Changing the Credit Service

There is no code change to the Credit service. There is one change to the web.xml deployment descriptor to add security constraints:

```
        <security-constraint>
                <web-resource-collection>
                        <web-resource-name>all</web-resource-name>
                        <url-pattern>/*</url-pattern>
                </web-resource-collection>
                <auth-constraint>
                        <role-name>**</role-name>
```

```
            </auth-constraint>
            <user-data-constraint>
                    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
            </user-data-constraint>
    </security-constraint>
```

### 3.3.6 Configuring the Credit Service

The server.xml for the Credit service looks like:

```
 1  <server description="new server">
 2
 3      <!-- Enable features -->
 4      <featureManager>
 5          <feature>jsp-2.3</feature>
 6          <feature>localConnector-1.0</feature>
 7          <feature>jaxrs-1.1</feature>
 8          <feature>appSecurity-2.0</feature>
 9          <feature>openidConnectClient-1.0</feature>
10          <feature>jwt-1.0</feature>
11      </featureManager>
12
13      <!-- To access this server from a remote client add a host attribute to the
following element, e.g. host="*"
 -->
14      <httpEndpoint httpPort="9082" httpsPort="9445" id="defaultHttpEndpoint"/>
15
16      <!-- Automatically expand WAR files and EAR files -->
17      <applicationManager autoExpand="true"/>
18
19      <keyStore password="{xor}Mz49Dz4sLCgwLTs="></keyStore>
20
21
22      <applicationMonitor updateTrigger="mbean"/>
23
24      <webApplication contextRoot="/credit" id="CreditServiceJWT"
location="CreditServiceJWT.war" name="CreditServ
iceJWT"/>
25
26   <openidConnectClient id="RP" signatureAlgorithm="RS256"
27             inboundPropagation="required"
28             audiences="CreditService"
29             issuerIdentifier="https://localhost:9443/jwt/defaultJWT"
30             jwkEndpointUrl="https://localhost:9443/jwt/ibm/api/defaultJWT/jwk" >
31        </openidConnectClient>
32
33  </server>
```

Note the similarity to the server.xml of the Banking service. The main differences are:

- Line 28: requires inbound JWT to have an audience of “CreditService” instead of “BankingService”
- Line 30 defines a JWK endpoint, rather than reference to a certificate to verify the JWT. Even though the Frontend server issues the JWT signed using its configured certificate, its JWK endpoint is still available for the backend service to fetch the key required to verify the

signature. Although not shown in this lab, the frontend can also be configured to only use the JWK protocol to rotate signing keys, in which case all backend services will be required to configure a jwkEndpoint. Also not shown here, if the frontend only psses the JWT issued by the OP, and the OP is JWK enabled, the backend's server.xml will need to configure a jwkEndpoint to point to the OP's JWK endpoint.

## 3.4 TLS Certificate Configuration

We are using self-signed certificates for this lab. The following certificates are pre-configured:

- For the OP:
  - A personal certificate to identify the OP and sign JWT tokens
- For the Frontend, which serves as a RP
  - A personal certificate to identify the RP and sign the JWT tokens it issues
  - A signer certificate from the OP so it can go to the RP to validate OIDC tokens
  - A signer certificate from Banking and Credit services to make outbound TLS calls
- For the Banking service:
  - A personal certificate to identify itself
  - A signer certificate from the Frontend to verify its JWT token
- For the Credit service:
  - A personal certificate to identify itself
  - A signer certificate from the Frontend to call the Frontend's jwk endpoint to fetch the key to verify the JWT.

Let us check the certificates in each application's key store. Note that we have chosen to store both personal certificates and signer certificates in the same keystore.jks. This can be reconfigured into separate key and trust stores.

__1. Verify the "keytool" command is on your path. If not add {LAB_HOME}/wlp/java/bin to your path.

__2. Verify OP's certificate:

__a. Change directory to {LABHOME}/wlp/usr/servers/OP/resources/security

__b. Run the keytool command

**keytool -list -keystore key.jks -storetype jks -storepass labPassword**

__c. Ensure the output contains only one certificate. It looks like:

```
Keystore type: jks
Keystore provider: IBMJCE

Your keystore contains 1 entry

default, Apr 18, 2019, keyEntry,
Certificate fingerprint (SHA1):
BD:AF:23:D3:5D:7D:E4:CD:41:54:43:49:62:3D:6D:40:F0:58:29:6D
```

__3. Repeat for the Frontend's keystore. It should contain 1 personal certificate, and 3 signer certificates

```
Keystore type: jks
```

```
     Keystore provider: IBMJCE

     Your keystore contains 4 entries

     default, Apr 18, 2019, keyEntry,
     Certificate fingerprint (SHA1):
05:47:6B:95:B4:58:24:99:5C:AF:CC:88:88:6D:F8:AC:8A:74:5F:75
     banking, Apr 18, 2019, trustedCertEntry,
     Certificate fingerprint (SHA1):
C5:13:76:B7:7D:84:17:35:14:70:7E:9E:63:A6:CA:76:EB:46:D4:43
     op, Apr 18, 2019, trustedCertEntry,
Certificate fingerprint (SHA1): BD:AF:23:D3:5D:7D:E4:CD:41:54:43:49:62:3D:6D:40:F0:58:29:6D
     credit, Apr 18, 2019, trustedCertEntry,
Certificate fingerprint (SHA1): FA:E6:DE:8F:76:FC:43:EF:45:43:80:17:7D:42:05:E5:E7:3F:5B:21
```

__4. Repeat for banking service and check it has only 2 certificates

```
Keystore type: jks
Keystore provider: IBMJCE

Your keystore contains 2 entries

default, Apr 18, 2019, keyEntry,
Certificate fingerprint (SHA1):
C5:13:76:B7:7D:84:17:35:14:70:7E:9E:63:A6:CA:76:EB:46:D4:43
frontend, Apr 18, 2019, trustedCertEntry,
Certificate fingerprint (SHA1):
05:47:6B:95:B4:58:24:99:5C:AF:CC:88:88:6D:F8:AC:8A:74:5F:75
```

__5. Repeat for Credit service and check it has only 2 certificates

```
Keystore type: jks
Keystore provider: IBMJCE

Your keystore contains 2 entries

default, Apr 18, 2019, keyEntry,
Certificate fingerprint (SHA1):
FA:E6:DE:8F:76:FC:43:EF:45:43:80:17:7D:42:05:E5:E7:3F:5B:21
frontend, Apr 18, 2019, trustedCertEntry,
Certificate fingerprint (SHA1):
05:47:6B:95:B4:58:24:99:5C:AF:CC:88:88:6D:F8:AC:8A:74:5F:75
```

## 3.5 What To Try Next (Optional)

Try the microprofile JWT guide at: https://openliberty.io/guides/microprofile-jwt.html

Try open liberty guides at: https://openliberty.io/guides/

# Appendix A. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have

been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. All references to fictitious companies or individuals are used for illustration purposes only.

# Appendix B. Trademarks and copyrights

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

| | | | | | |
|---|---|---|---|---|---|
| IBM | AIX | CICS | ClearCase | ClearQuest | Cloudscape |
| Cube Views | DB2 | developerWorks | DRDA | IMS | IMS/ESA |
| Informix | Lotus | Lotus Workflow | MQSeries | OmniFind | |
| Rational | Redbooks | Red Brick | RequisitePro | System i | |
| *System z* | *Tivoli* | *WebSphere* | *Workplace* | *System p* | |

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of The Minister for the Cabinet Office, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

# NOTES

# NOTES



The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.




Please Recycle