**JWT Bearer Grant**

The JSON Web Token (JWT) is simply a JSON string containing claim values. The JWT Bearer grant handler evaluates and validates the claims in the JWT token and then issues an access token at the Authorization Server end.

WSO2 API Manager, as an OAuth 2.0 Authorization Server with its Key Manager features, can accept JWT assertions from OAuth 2.0 clients as a means of resource owner authentication and authorization. Additionally, it can exchange the JWT token with OAuth 2.0 access tokens in order to access protected resources on behalf of the resource owner.

### The Flow

- Configuring the identity provider and the service provider
- Using the JWT grant
  - Sample
  - The JWT assertion
  - Generating the JWT assertion

#### The Flow

A client can exchange a JWT token for an OAuth 2.0 access token using the JWT Bearer grant type. A service provider (sp) and an identity provider (idp) are required on the WSO2 API Manager Server for the JWT Bearer grant type to work. The service provider gets automatically created on the WSO2 API Manager when you generate keys for an application available on the API Store. The WSO2 APIM server uses the service provider to obtain information about the application created on the API Store. The identity provider configuration must be created explicitly on the API Manager via the management console. This configuration should contain details of the Identity provider (for e.g. WSO2 Identity Server or a third party Identity provider such as Facebook etc.) who creates and signs the JWT assertion. The WSO2 APIM Server uses the Idp configuration to identify the issuer of the JWT and obtain the public certificate of the IDP, in order to validate the JWT.

When a request is made to the token endpoint with the grant type, the JWT assertion, the client key and client secret, the WSO2 APIM Server will read the grant type and trigger the JWT Bearer Grant Handler. This handler will check for the issuer of the JWT token and retrieve the IDP configuration. It will then obtain the public certificate of the IDP stored in the IDP configuration and validate the JWT. Once the JWT is validated, it will create an OAuth2.0 access token for the application holding the provided client key and client secret.

### Configuring the identity provider and the service provider

1. Sign in to the WSO2 API Store.
2. Create a new application if an application is not available already.
3. Generate keys for the application.
4. Sign in to the WSO2 API Manager. Enter your username and password to log in to the Management Console ([https://localhost:9443/carbon](https://localhost:9443/carbon)).
5. Navigate to the **Identity providers** section under the **Main** tab of the management console and click **Add**.
6. Provide the following values to configure the IDP:
   - **Identity Provider Name**: Enter the JWT issuer name as the identity provider name. This is used to generate the JWT assertion.
   - **Identity Provider Public Certificate**: The certificate used to sign the JWT assertion.
Identity provider Public Certificate

The **Identity Provider Public Certificate** is the public certificate belonging to the identity provider. Uploading this is necessary to authenticate the response from the identity provider.

This can be any certificate. If the identity provider is a WSO2 API Manager or WSO2 Identity Server instance, this can be a `ws02.crt` file.

To create the identity provider certificate from the `wso2carbon.jks` file, follow the steps below.

1. In your Command Line interface, go to the `<APIM_HOME>/repository/resources/security/` directory and run the following command.

   ```
   keytool -export -alias wso2carbon -file wso2.crt -keystore wso2carbon.jks -storepass wso2carbon
   ```

2. Once you run this command, the `wso2.crt` file is generated and can be found in the `<APIM_HOME>/repository/resources/security/` directory.

3. In the Identity Provider configuration on the management console, you will find the "Identity Provider Public Certificate" field. Click **Browse** and navigate to `<APIM_HOME>/repository/resources/security/` and upload the certificate you created in step 1.

See [Using Asymmetric Encryption](https://wso2.com/library/asymmetric-encryption) in the WSO2 Product Administration Guide for more information on how public keys work and how to sign these keys by a certification authority.

- **Alias**: Give the name of the alias if the identity provider identifies this token endpoint by an alias (e.g., `https://localhost:9443/oauth2/token`). For more information, see [Adding a new identity provider](#).

Using the JWT grant

The curl commands below can be used to retrieve the access token and refresh the token using a JWT.

1. Go to the main menu and click **Identity Service Providers**.
2. If you have generated the key(s) for the application on the API store as mentioned in step3, you should be able to see a service provider listed as shown in the image below. The service provider name will have the format `<application owner>_<application name>_<generated key type>`.

Using the JWT grant

The curl commands below can be used to retrieve the access token and refresh the token using a JWT.
**Request**

```bash
```

- The `-u` flag should specify the "Client Id":"Client Secret" value.
- The assertion parameter value is the signed base64 encoded JWT. The value of the assertion parameter **MUST** contain a single JWT. See JWT Assertion for more information about assertion.

---

**Sample**

When you send a request to the token endpoint with the JWT Bearer grant type, it replies with a response that contains the access token, refresh token, expiry time, and token type. Following is a sample request and response.

**Sample Request**

```bash
curl -i -X POST -H 'Content-Type: application/x-www-form-urlencoded' -u bBhEoE2wIpU1zB8HA3Gfv2z8xxAa:RKgXUC3pTRQq9xPnPwutTFGtnSQA -k -d 'grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer&assertion=eyJhbGciOiJSUzI1NiJ9.eyJleHAiOjE0NTgxNjY5MDUsInN1YiI6ImFkbWluIiwibmJmIjoxNDU4MDA2OTg1LCJhdWQiOiJ1c2VyIiwiaWF0IjoxNDU4MDA2OTg1fQ.eyJpc3MiOiJqd3RJRFAiLCJqdGkiOiJUb2tlbjU2NzU2IiwiaWF0IjoxNDU4MDA2OTg1fQ.ZcxdoTVEsWoil80ne42QzmsfelMWyjRZJEjUK1c2vMZJjhrZnsWExyCA5tNEIXYPAXC_7rkF

```

**Sample Response**

```json
{"token_type":"Bearer","expires_in":3600,"refresh_token":"bb4b78e2b0e4f956ac89f0f2e38a8833","access_token":"615ebbcc943be052cf6dc27c6ec578816"}
```

---

**The JWT assertion**

A JWT assertion contains a header, payload, and signature, each separated by a dot (.).

The header identifies the algorithm used to generate the signature. For example, see the following code block.
Sample header

```
{
   "alg":"RS256"
}
```

The payload contains the claims mentioned below:

- **iss (issuer)** - The JWT must contain an **iss (issuer)** claim containing a unique identifier that identifies the identity provider that issued the JWT.
- **sub (subject)** - The JWT must contain a **sub (subject)** claim identifying the entity that the identity provider or the entity that issued the JWT vouches for.
- **aud (audience)** - The JWT must contain an **aud (audience)** claim containing a value that identifies the authorization server as the intended audience. This value should be registered as a token endpoint alias in the identity provider.
- **exp (expiration time)** - The JWT must contain an **exp (expiration) claim** that limits the time window during which the JWT can be used.
- **nbf (not before)** - The JWT may contain a **nbf (not before time)** claim that forces a JWT to be used only after a specified time.
- **iat (issued at)** - The JWT may contain an **iat (issued at)** claim that identifies the time at which the JWT was issued.
- **jti (json web token ID)** - The JWT may contain a **jti (JWT ID)** claim that provides a unique identifier for the token.
- **Other custom claims** - The JWT may contain claims other claims not mentioned above. This is the extension point of the JWT specification.

For example, see the following code block.

Sample payload

```
{
   "sub":"admin",
   "aud": [
      "https://localhost:9443/oauth2/token"
   ],
   "nbf":1507546100,
   "iss":"jwtIDP",
   "exp":1507606100,
   "iat":1507546100,
   "jti":"Token56756"
}
```

Generating the JWT assertion

1. Deriving the signature.
   a. Encode the header and the payload separately using a base64 URL.
   b. Concatenate the encoded header and payload with a period and sign it to generate the signature.

   \[
   \text{Signature} = \text{sign} (\text{encodeBase64(header)} + \cdot + \text{encodeBase64(payload)})
   \]

2. Encode the signature using base64 URL encoding.
3. Generate the JWT assertion by concatenating the values of the base64 URL encoded header, payload, and signature using a dot "." as the separator.

   \[
   \text{assertion} = \text{encodeBase64(header)} + \cdot + \text{encodeBase64(payload)} + \cdot + \text{encodeBase64(signature)}
   \]

The result is as follows:
Sample assertion

| eyJhbGciOiJIUzI1NiJ9. eyJzdWIiOiJhZGl1bSIsImF1dF9tZXRhcmVzc1wvbG9jYWxob3N0OiI1N0NDNcL29hdXRoMlwvdG9rZW4iXSwiZXhwIjoiMjI0MDdjZjIiLCJleHAiOjE1MDc2MDYxNjAsImlhdCI6MTUwNzU0NjEwMCwiZXhwIjoxNTA3NTQ2MTAwLCJpYyI6e1wvcmVxdWlyb3NvZGluZ3M6NzU6fQ== |
| iGMbjjibB0W2Q7Q1M27gnHps647Eybw8cAHf2O21-xqo284uL_1VppF14CCJXTj4qrvV9vmkJ5fKNAAyiTa6wOMXGL4Xnwywp0AoKXvbozn1EDNRpw3htW34nLvuyu6Pj8bdvAPVjh8kPrwfs7esRr2p-lueCGc21mJjWdhyGzM4hE |