Oracle® Cloud

Administering Oracle Cloud Identity Management

Release 17.2

E59052-15

May 2017

Documentation for Oracle Cloud account administrators, security administrators, and identity domain administrators that explains how to configure Federation SSO and how to provision OAuth resources and clients using the self-service user interface (UI) and how to protect Oracle Cloud services using two-legged OAuth, service-to-service authorization.

ORACLE

Oracle Cloud Administering Oracle Cloud Identity Management, Release 17.2

E59052-15

Contents

3 Securing Authorizations in Oracle Cloud

Preface

Oracle ® Cloud Administering Oracle Cloud Identity Management explains how to provision Oracle Single Sign-On (SSO)and configure various OAuth resources and clients using the self-service user interface.

Topics:

Audience

This guide is intended for Oracle Cloud account administrators and customers buying Oracle Cloud services, who want to configure SSO and Identity Federation using Security Assertion Markup Language (SAML), and manage various OAuth resources and clients..

Scope of the Guide

The tasks explained in the guide include:

- Single Sign-On (SSO)
- OAuth resource management
- OAuth client management

Shared Identity Management (SIM) uses SAML to function as a SAML service provider to Oracle Fusion Applications SAML identity provider. This is done through Oracle Public Cloud support. In addition, SIM operates as a SAML service provider to federate with a SAML identity provider, such as Oracle Fusion Applications, Oracle Access Management, Microsoft Active Directory Federation Services (ADFS), and Shibboleth.

Related Resources

For additional documentation related to your Oracle Cloud service, visit the Oracle Cloud website at:

`http://cloud.oracle.com`

Open the **Support** menu at the top of the page and select **Documentation** to access the Oracle Cloud Documentation home page. Search or browse the library for documentation specific to your application, infrastructure, or platform cloud service.

Conventions

The following text conventions are used in this guide:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Managing Oracle Single Sign-On

By implementing Oracle Single Sign-On, your users can access multiple Oracle Cloud services using one set of credentials. Also, logging out of one service logs a user out of all other services.

As administrator, you configure SSO because you want to use identity federation between Oracle Cloud as service provider and an external identity provider. This task requires you to configure Oracle Cloud as service provider, prepare your identity provider, test your SSO configuration, and finally, enable SSO.

Topics:

Note: To learn more about the concepts of Oracle Single Sign-On, see About SSO in *Understanding Identity Concepts*.

Overview of SSO Configuration Tasks

As administrator, you enable SSO so your users can use their company credentials to log in to all applications, including Oracle Cloud applications. This requires you to configure SAML 2.0 between Oracle Cloud and the identity provider.

The following table shows you the steps that you must follow when configuring SSO on the SSO Configuration page from My Services in Oracle Cloud:

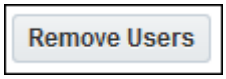
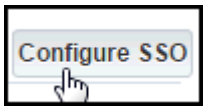
Task	Description	Additional Information
Configure Oracle Cloud as a service provider.	Go to the **Users** page and then click the **SSO Configuration** tab to configure Oracle Cloud as the service provider.	Configuring Oracle Cloud as the Service Provider
Configure an identity provider.	After you configure Oracle Cloud as a service provider, you configure your identity provider.	Configuring an Identity Provider
Test Single Sign-On.	Test your SSO configuration before enabling SSO.	Testing Single Sign-On
Identify problems by testing SSO.	Testing SSO can identify a number of problems that you must fix before you can enable SSO.	Problems Identified by Testing SSO
Enable SSO.	You must enable SSO before you can use it.	Enabling SSO
Enable sign in with identity domain credentials.	If you want users (such as identity domain administrators) to log in using their identity domain credentials, you must enable this option,	Enabling Sign In With Identity Domain Credentials
Remove users.	After you enable SSO, ensure that users do not have credentials in Oracle Cloud.	Removing Users
Update SSO metadata.	At some point, after you've enabled SSO in production, you might need to update the SSO metadata.	Updating SSO Metadata
Troubleshoot SSO.	If you can't resolve a configuration problem by testing SSO, then you must troubleshoot the configuration.	Troubleshooting SSO

Exploring the SSO Configuration Page in My Services

The SSO Configuration page in My Services helps Oracle Cloud account administrators and customers buying Oracle Cloud services to configure SSO between your identity provider and with Oracle Cloud as the service provider.

What You Can Do from the SSO Configuration Page

The following table describes what you can do from the SSO Configuration page:

Tool	Description
	Click **Remove Users** to remove users that you added in Oracle Cloud before enabling SSO. To learn more about why you should remove these users, see Removing Users.
	Click **Configure SSO** to start a set of tasks to configure an identity provider, service provider, and SSO. To learn more about the configuration steps and the tasks that you must perform, see Managing Oracle Single Sign-On. The Configure an Identity Provider with Oracle Cloud - Tutorial Series guides you through the configuration steps for different identity providers.

What You Can See from the SSO Configuration Page

The SSO Configuration page displays the following information:

Field	Description
	You can see the status of the SSO configuration when you access the SSO Configuration page. Before configuring SSO, it shows that SSO is **Not Configured.**
	You can start configuring SSO between Oracle Cloud and your identity provider.

Configuring Oracle Cloud as the Service Provider

To configure SSO, start with configuring Oracle Cloud as service provider.

To configure SAML 2.0 SSO between Oracle Cloud as service provider and the identity provider:

1. Go to the **My Services** dashboard page and click **Users**. Then click the **SSO Configuration** tab.
2. Click **Configure SSO**.

   The **Configure SSO** page is displayed.
3. Select whether to import identity provider metadata or enter provider the metadata manually. Your choice depends on whether your identity provider can export metadata.
4. The next step depends on the selection that you made in Step 3.

   a. If your Identity Provider can export metadata, then you can import the metadata into Oracle Cloud. Select **Import Identity Provider metadata**. Click **Choose File** and upload the identity provider metadata file (such as **idp_metadata.xml**).

b. For the **SSO Protocol** field, **HTTP POST** is recommended and is the default. The **SSO Protocol** field value refers to the SAML binding that's used. SAML bindings define how the SAML protocols map to the type of transport used. Oracle Cloud supports **HTTP POST** and **HTTP Artifact**. The **HTTP POST** binding defines how SAML protocol messages can be transported with the base64-encoded content of a form control within an HTML form. The **HTTP Artifact** binding defines how a reference (or *artifact*) to a SAML request or response is transported over HTTP; the artifact (a small representation of a complete SAML assertion) can be embedded in a URL as a query string parameter, or it can be placed in a hidden form control.

c. Select the **User Identifier** field. The user identifier is the Oracle LDAP directory attribute that's used to map the user information contained in the incoming SSO SAML assertion to an Oracle Cloud user. It's either the user's email address or the user ID. Select **User's Email Address**.

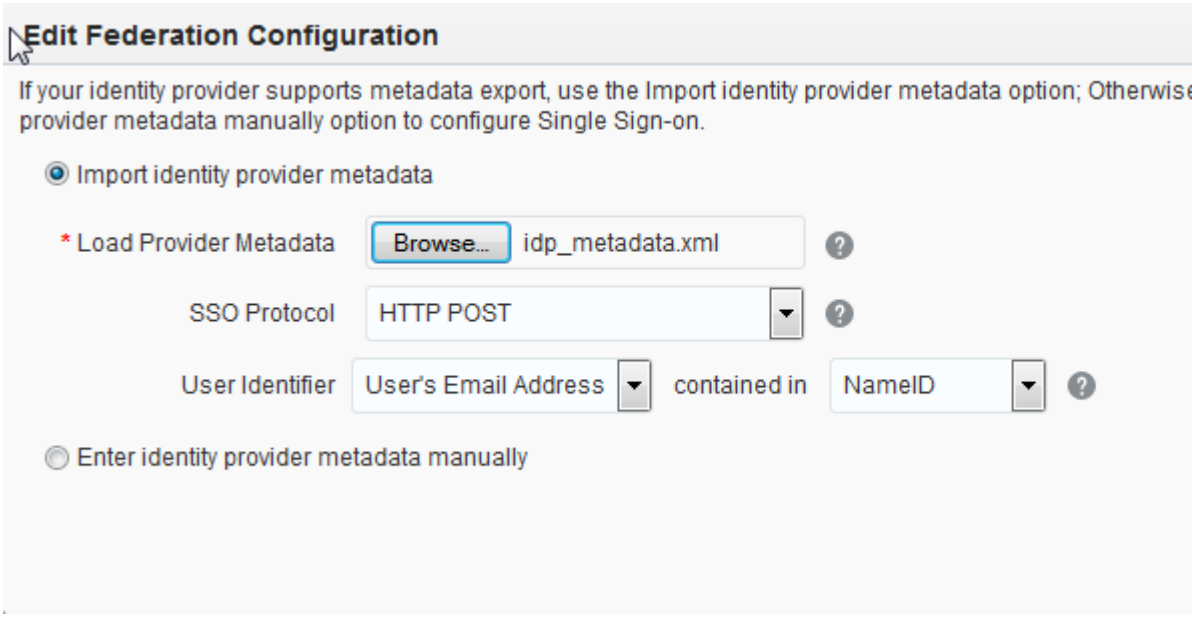
d. Select the **contained in** field. If the **User Identifier** is the user's email address, then the **contained in** field must be **NameID**.

Note:

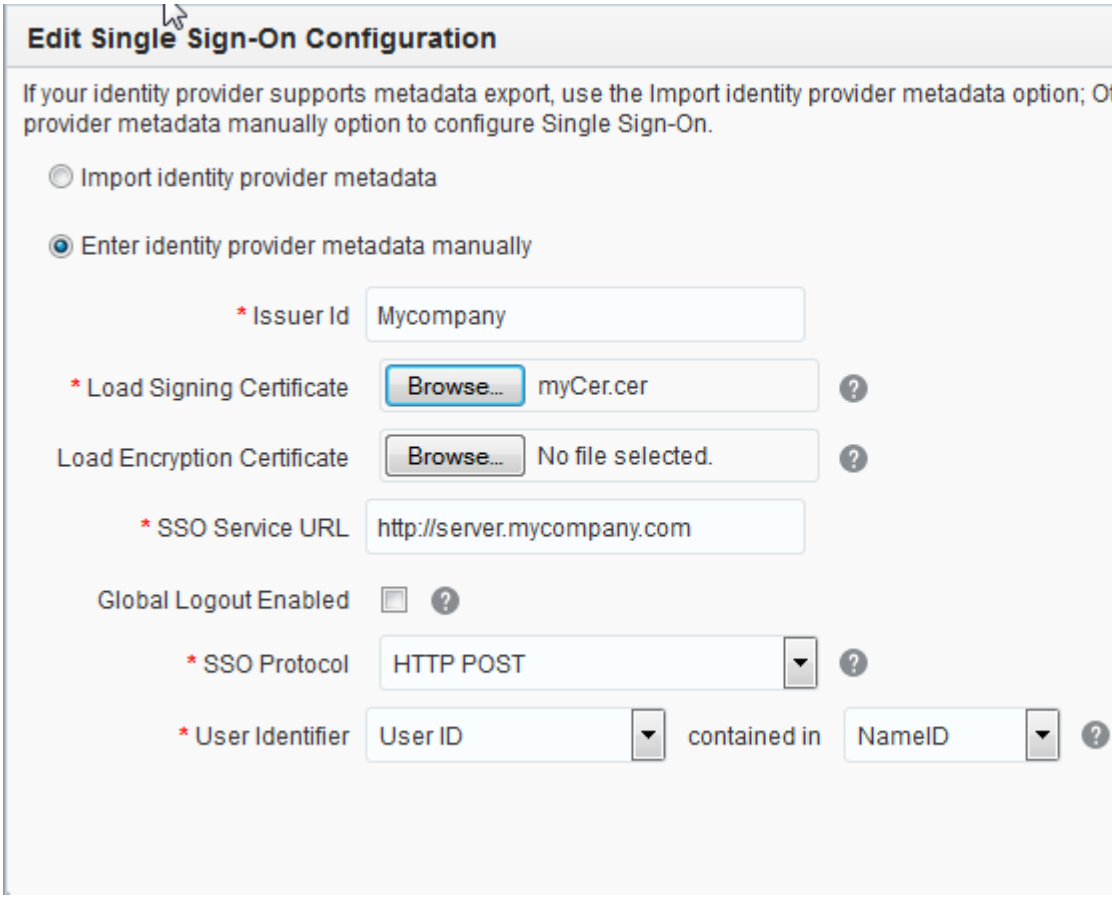
If the **User Identifier** value is the user ID, then the **contained in** field must be the SAML attribute and you must specify the name of the SAML attribute for the **contained in** field such as SamAccountName in the case of Microsoft Active Directory Federation Services.

---

e. Click **Save.**



f. If your identity provider can't export metadata, then you must enter metadata information manually, which means you must also provide the **Issuer ID** and **SSO Service URL** (this is the SAML assertion consumer URL), and indicate whether **Global logout** should be enabled. You must also load your identity provider's **signing certificate** and **encryption certificate.**



Configuring an Identity Provider

After you configure Oracle Cloud as a service provider, configure your identity provider in the **Configure your Identity Provider Information** section of the **SSO Configuration** page.

1. Go to the **Users** page and click the **SSO Configuration** tab. Then scroll down to the **Configure your Identity Provider Information** section.
2. What you need to configure the identity provider depends on one of the following:
   - If your identity provider can import metadata, export the metadata from the Service Provider to import into the Identity Provider by doing the following:
     a. In the **Configure your Identity Provider Information** section click **Export Metadata**, then select **Provider Metadata**.
     b. Save the metadata to a local file as **SP_metadat.xml**.
   - If your identity provider can't import metadata, then copy and paste the provider ID and URLs into a SAML 2.0 file to be used by the identity provider. Download the certificates from the service provider.
3. Configure your identity provider, using its configuration interface. The configuration steps are specific to each identity provider.

Testing SSO

Test SSO to identify any SSO configuration problems.

Go to the **Users** page and then click the **SSO Configuration** tab.

1. On the **SSO Configuration** page in the **Test your SSO** section, click **Test**.

   The **Initiate Federation SSO** page appears.

2. Click **Start SSO**.

   Clicking **Start SSO** triggers a Federation SSO workflow. You're redirected to the identity provider's login page and challenged for authentication.

3. Log in as an administrator. After the Federation SSO is performed, the result is displayed in the **Test SSO** page.

4. The next step depends on whether the test is successful:

   - If the test is successful, then proceed to Enabling SSO.
   - If the test is unsuccessful, then view the test results to determine the cause. See Problems Identified by Testing SSO

Problems Identified by Testing SSO

The Test SSO feature can identify various problems.

The Assertion Couldn't be Mapped to an Oracle Cloud User

This may occur for the following reasons:

- The SIM user corresponding to the identity provider user doesn't exist.
- Oracle Cloud was incorrectly configured to map the incoming SSO assertion.

An Error Occurs When Oracle Cloud Consumes the SAML Assertion

To resolve this problem:

- Ensure that the Oracle Cloud federation server has the latest identity provider metadata and signing certificate.
- If the identity provider encrypts the assertion, ensure that the identity provider has the correct Oracle Cloud encryption certificate.

After Logging Out, the User is Automatically Logged in Again

This typically occurs when Oracle Cloud is wired with the identity provider using HTTP basic authentication or with Microsoft Active Directory Federation Services identity provider using Windows Integrated Authentication as the challenge mechanism. Upon logging out and performing the SAML 2.0 logout protocol, the user is automatically logged in again. The identity provider can't log the user out because:

- The browser caches the HTTP basic authentication credentials and thus the identity provider can't log the browser out.

- The Windows Desktop machine where the user is signed in automatically signs in the browser with Microsoft Active Directory Federation Services identity provider, so the identity provider can't log the browser out.

To resolve this problem, change the authentication mechanism at the identity provider.

The Identity Metadata Fails to Be Uploaded from the Console.

To resolve this problem:

- Ensure that the metadata wasn't modified.
- When downloading the metadata from the identity provider, save it using the **File —> Save As** command. That is, don't copy and paste the contents of the browser, because this action modifies the contents of the metadata.

SSO Fails Because the Assertion Isn't Signed.

The Oracle Cloud federation server requires the SAML assertion to be signed. Ensure that the assertion is signed and contains a digital signature element, even if the SSO response is signed.

Problems that Can't Be Resolved

If you can't resolve the problem using the Test feature, proceed to Troubleshooting SSO.

Enabling SSO

Until you specifically enable SSO, you can't use it. After SSO is enabled, you should be able to authenticate through the identity provider, after selecting **Sign in using your company ID** on the **Sign In to Oracle Cloud** page.

Go to the **Users** page and then click the **SSO Configuration** tab. If the status in the **Enable SSO** section is **SSO is Not Enabled**, and you tested SSO successfully, and you want to enable SSO, then click **Enable SSO** to enable SSO for all Oracle Cloud services. Until you do this, SSO isn't enabled.

After you enabled SSO, you can disable it from the **Enable SSO** section of the **SSO Configuration page**.

Enabling Sign In With Identity Domain Credentials

After SSO is enabled, users typically sign in using their identity provider credentials. If you want your users to be able to sign in with their identity domain (Oracle Cloud) credentials, you need to enable this option.

After you enable SSO, you have the option to allow users to sign in with their identity domain credentials as well. This option is disabled by default because typically, as administrator you want to force users to log in using their identity provider credentials.

To enable the option for users to sign in with their identity domain credentials:

1. Go to the **Users** page and then click the **SSO Configuration** tab.
2. Go to the **Enable Sign In to Oracle Cloud Services with Identity Domain credentials** section. Click **Enable**.

3. A confirmation window appears informing you that after enabling, users that do have credentials in their identity domains (for example identity domain administrators), will be able to sign in to Oracle Cloud services using either their identity provider or identity domain credentials.

> **Note:** You can't enable signing in with identity domain credentials, if SSO was auto-configured for your system. The **Enable Sign In to Oracle Cloud Services with Identity Domain credentials** button is disabled in this case.

After you enabled sign in to Oracle Cloud with identity domain credentials, you can disable it from the **Enable Sign In to Oracle Cloud Services with Identity Domain credentials** section of the page. This is necessary, if you want to force users to sign in only with their identity provider credentials.

Removing Users

Remove all users without the identity domain administrator role after you enable SSO.

After you enable SSO, only users that have the identity domain administrator role or were created before SSO was enabled, have credentials in Oracle Cloud. To avoid maintaining credentials in two places after enabling SSO, you typically delete the existing users and then reimport them. This step ensures that the users don't have credentials in Oracle Cloud and can access Oracle Cloud applications only with their company credentials.

To delete all users that don't have the identity domain administrator role assigned:

1. Go to the **Users** page and then click the **SSO Configuration** tab.
2. Click **Remove Users**.
3. A window appears confirming that all users without the identity domain administrator role will be removed, and that this operation can't be undone.
4. Click **Remove Users** to remove all users who don't have the identity domain administrator role assigned.
5. A window displays the progress of the removal process, and then the number of users removed.

Updating SSO Metadata

After you've enabled SSO in production, you might want to update the SSO metadata.

Reasons for updating the metadata include:

- The identity provider or service provider certificate has expired.
- The identity provider or provider key has been compromised.
- The identity provider URL endpoints need to be updated.

If any of these reasons applies, then:

1. Schedule an update of the SSO metadata in advance, because it requires an outage.
2. Disable SSO using **Disable SSO**.

3. Update the identity provider or service provider metadata as needed.
4. Test the configuration, as described in Testing SSO.
5. After testing shows that SSO is working correctly, reenable SSO by clicking **Enable SSO** as described in Enabling SSO.

Troubleshooting SSO

If you can't resolve a configuration problem by using the Test feature, then troubleshoot the configuration by following these steps.

1. Review the Known Issues guide for any similar problem.
2. Review any changes made on the identity provider and Oracle Cloud service provider before the problem in the SSO workflow.
3. Capture an HTTP trace of the SSO workflow, using a tool such as Fiddler Web Debugging Tool.
4. Review the workflow to determine the point where the SSO workflow terminated and which identity-related components are involved: identity provider, service provider, web tier, gateways, proxies, and firewalls.
5. Review the protocol messages and component logs to identify exceptions.
6. Go to MyOracle Support to review known issues and find out if your problem exists there.
7. If you've performed all troubleshooting steps and you're confident that the problem is due to Oracle Cloud, then contact Oracle Support Services. Be ready to provide all your information, including a Fiddler trace, identity provider metadata, and identity provider logs.

2

Managing OAuth Resources and Clients

OAuth 2.0 is an authorization framework that enables an application or service to obtain limited access to a protected HTTP resource. In OAuth, the applications are called clients; they access protected resources by presenting an access token to the HTTP resource. As an administrator, you configure OAuth resources and clients and administer them.

Topics:

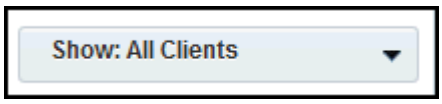
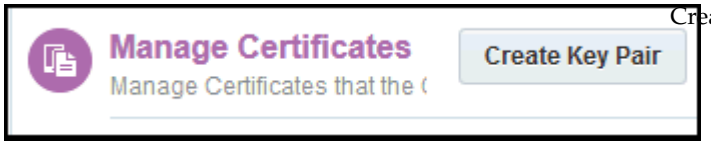
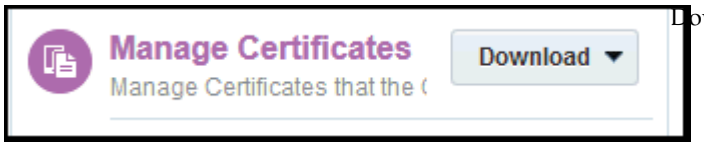
- Managing Client Certificates
- Troubleshooting OAuth

Exploring the OAuth Administration Page in My Services

The OAuth Administration page in My Services helps Oracle Cloud account administrators and customers buying Oracle Cloud services to configure OAuth clients and resources. You can register new OAuth clients and resources, grant/revoke API access, and manage the settings of resources and clients

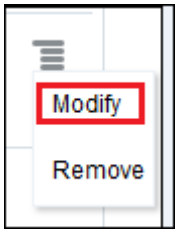
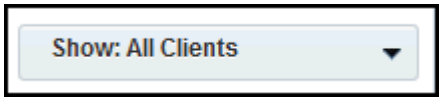
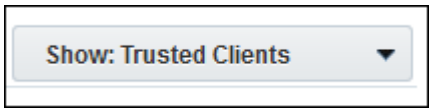
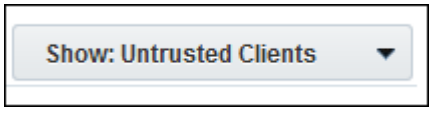
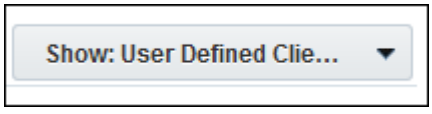
What You Can Do from the OAuth Administration Page

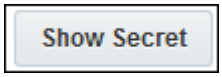
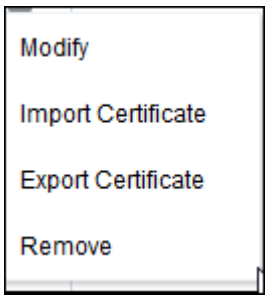
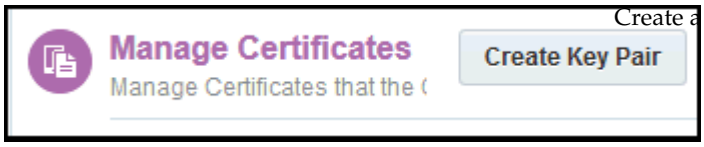
The following table describes what you can do from the **OAuth Administration** page.

Tool	Description
	Register a new OAuth resource.
	Register resources by importing data from a file.
	Enter the name of a resource (or a part of the name), and then search for OAuth resources.
	Register a new OAuth client.
	Show all registered OAuth clients.
	Enter the name of a client (or a part of the name), and then search for OAuth clients.
	Create a key pair for the OAuth client certificate.
	Download OAuth client certificates.

What You Can See from the OAuth Administration Page

The OAuth Administration page displays the following information:

Field	Description
	Register a new OAuth resource.
	Register resources by importing data from a file.
	View the various options to edit an existing OAuth resource: • Select **Modify** to change the properties of the resource. • Select **Remove** to delete the OAuth resource.
	Enter the name of a resource (or a part of the name), and then search for OAuth resources.
	Register a new OAuth client.
	Show all registered OAuth clients.
	Show all OAuth clients that are trusted.
	Show all OAuth clients that are untrusted.
	Show all OAuth clients that are user-defined
	Show all OAuth Clients that are created by the infrastructure.

Field	Description
	Enter the name of a client (or a part of the name), and then search for OAuth clients.
	Shows the secret of the OAuth client.
	View the various options to edit an existing OAuth client: • Select **Modify** to change the client properties. • Select **Import Certificate** to import a new certificate for the OAuth client. • Select **Export Certificate** to export the existing client certificate. • Select **Remove** to delete the client.
	Create a key pair for the OAuth client certificate.
	View a list of the various client certificate that you can download. Select the appropriate option to save the certificate locally.

How Do I Set Up OAuth in Oracle Cloud?

You're an administrator at a company that has purchased some Oracle Cloud services. You want to configure OAuth to secure access to those services. As an administrator, you are responsible for setting up OAuth. During set up, you need to configure OAuth clients and resources in Oracle Cloud.

You have the following responsibilities as an administrator:

- Configure and manage OAuth resources.
- Configure and manage OAuth clients.
- Ensure that the communication between different services (on-premises and cloud) is secure.

The following table describes the steps to follow when setting up OAuth using the OAuth Administration page from My Services in Oracle Cloud:

Task	Description	Additional Information
Register an OAuth resource.	Go to the **OAuth Administration** page to register a new OAuth resource.	Registering New Resources in Oracle Cloud
Register a trusted OAuth client.	After you register an OAuth resource, configure and register an OAuth client. Decide whether you want the OAuth client to be trusted or untrusted. To register a new trusted OAuth client, go to the **OAuth Administration** page.	Registering a Trusted OAuth Client
Register an untrusted OAuth client.	After you register an OAuth resource, you can configure and register either a trusted or an untrusted OAuth client. To register a new untrusted OAuth client, go to the **OAuth Administration** page.	Registering an Untrusted OAuth Client

Note: To learn the difference between trusted and untrusted OAuth clients, see OAuth Client Types and Digital Signatures in *Understanding Identity Concepts.*

How Do I Administer OAuth in Oracle Cloud?

As an administrator, you're responsible for managing OAuth clients and resources in Oracle Cloud.

Administering OAuth in Oracle Cloud: Task Flow

An OAuth client and an OAuth resource can be managed in different ways. You may need to get information about an existing client or resource. You can search for an OAuth client or a resource to get this information. As an administrator, you're required to modify the configuration of an OAuth client or a resource. If security is compromised, then you may even need to remove the OAuth client. Similarly, a protected OAuth resource can also be removed from Oracle Cloud.

This section describes the high-level tasks for administering OAuth in Oracle Cloud. Both OAuth clients and resources can be administered in Oracle Cloud.

OAuth clients and resources can be managed by using the OAuth Administration page.

To manage existing OAuth resources:

- Search: See Viewing OAuth Resources.
- Update: See Updating OAuth Resources.
- Delete (Remove): See Deleting OAuth Resources

To manage existing OAuth clients (both trusted and untrusted):

- Search: See Viewing OAuth Clients.
- Update: See Updating OAuth Clients
- Delete (Remove): See Deleting OAuth Clients

To modify the certificates of both trusted and untrusted OAuth clients, see Managing Client Certificates

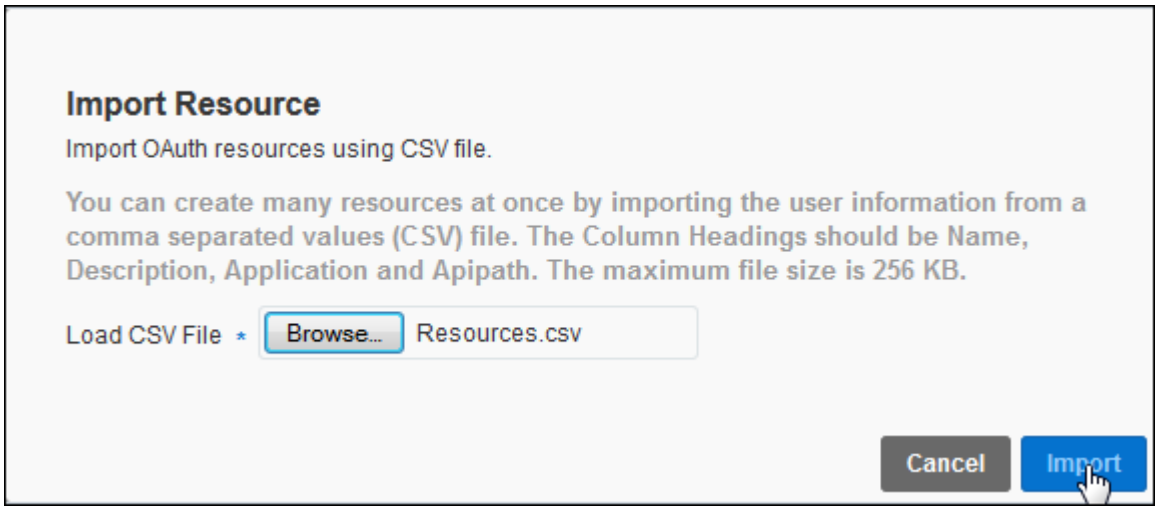
Registering New Resources in Oracle Cloud

From the OAuth Administration page, you can register a new resource in Oracle Cloud. A resource is a protected service in Oracle Cloud. When you register a new resource, you define some parameters and these parameters are used in authorizing the client request to those services,

To register new resources in Oracle Cloud using the UI:

1. Log in to the user's identity domain in Oracle Cloud and click **Users**.
2. Click the **OAuth Administration** tab.

   You can see the list of existing resources.
3. To register multiple resources at one time by using the data from a comma-separated values (CSV) file, click **Import**.



4. To register a new resource, click **Register**. To register the resource (the * indicates mandatory fields), enter the following information in these fields :

   a. **Name**: Enter the name of the OAuth resource. If another resource in the same application already uses this name, then an error is thrown.

   b. **Application**: Enter the name of the application, which is the service name. The resource is part of the application.

   c. **Description**: Enter a description of the OAuth resource. A description is optional. If this field is left blank, the description value defaults to the name of the resource.

d. **API Path**: Enter a valid path for the API. The API path should be an absolute path that points to the URL of the resource. When registering a resource from the UI, you can provide only one API path for the resource.

5. Click **Register**.

Populating the Audience Attribute for the New OAuth Resource:

When a resource is registered with OAuth, information about the API path is stored in the `Audience` attribute for the resource. When a client application requests an access token to access a protected service, OAuth uses this audience information to validate the API information provided in the access token request.

Overview of Managing OAuth Resources

You can search for an existing OAuth resource, modify its properties, and remove an OAuth resource, if necessary.

Task	Description	Additional Information
View OAuth resources.	View an existing OAuth resource from the **OAuth Administration** page.	Viewing OAuth Resources
Modify OAuth resources.	Change the properties of an OAuth resource by using the **OAuth Administration** page.	Updating OAuth Resources
Delete OAuth Resources.	Remove OAuth resources by using the **OAuth Administration** page.	Deleting OAuth Resources

Viewing OAuth Resources

An existing OAuth resource can be viewed at any time from the **OAuth Administration** page.

1. Log in to the user's identity domain and click **Users**.

2. Click the **OAuth Administration** tab.

   A list of all resources appears, including the resource name, application name, resource description, resource identifier, and API path.

3. To search for a resource, entering the name (partial name or full name) in the **Search** field.

   All resources whose names match the pattern text that you enter in the **Search** field appear. The following example uses, **res2** as search pattern in the **Search** field. As a result, only one resource appears (**test_res2**) because only one resource has **res2** in its name.

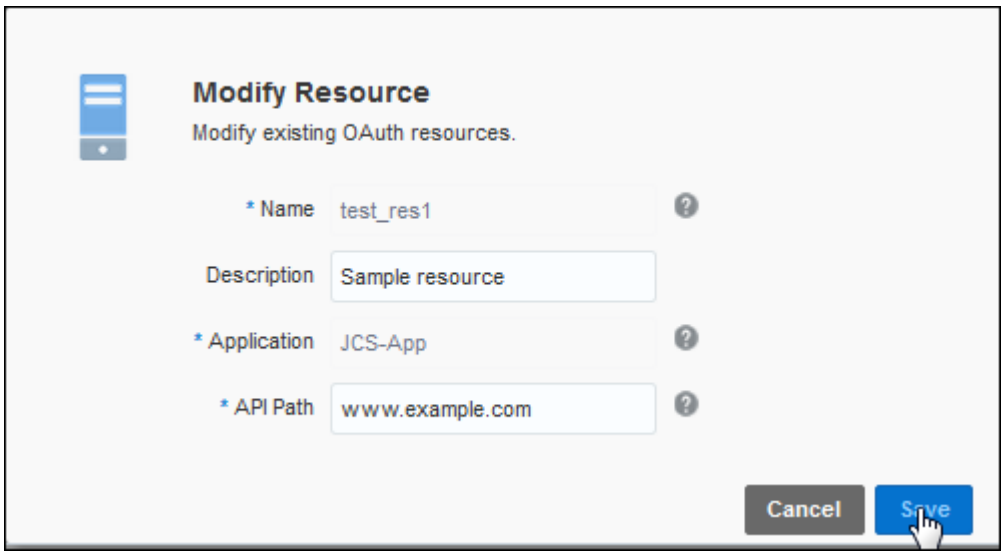
Updating OAuth Resources

The properties of an existing OAuth resource can be modified.

To change properties of an existing OAuth resource using the UI:

1. Log in to the user's identity domain in Oracle Cloud and click **Users**.
2. Click the **OAuth Administration** tab. The list of resources appears.
3. Click the **Action** menu icon to the right of the resource that you want to update. A menu is displayed with two options: **Modify** and **Remove**.

4. To change the properties of the resource such as the resource's description and the API path, click **Modify**. The name of the resource and the name of the application can't be modified.



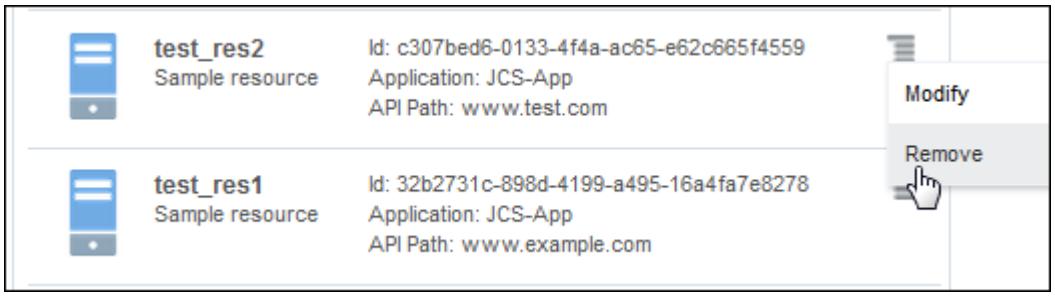
5. Click **Save**.

Deleting OAuth Resources

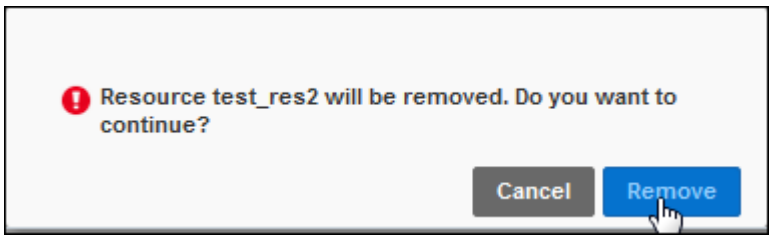
An existing OAuth resource can be removed at any time.

To delete a resource using the **OAuth Administration** page:

1. Log in to the user's identity domain in Oracle Cloud and click **Users**.
2. Click the **OAuth Administration** tab. The list of resources appears.
3. Click the **Action** menu icon to the right of the resource that you want to delete. Select **Remove**.



4. The **Remove Resource** confirmation window appears. Click **Remove** to delete the resource.



Overview of OAuth Client Configuration Tasks

As an administrator, you configure and administer OAuth clients for Oracle Cloud.

These configuration and administration tasks consist of the following:

1. Registering trusted and untrusted OAuth clients using the OAuth administration UI.
2. Importing an OAuth certificate for an OAuth client.
3. Testing the OAuth client registration to ensure that the client is registered successfully.
4. Viewing OAuth clients from the OAuth administration UI in My Services.
5. Updating OAuth configuration settings and properties by using the OAuth administration UI.
6. Managing client certificates by using the OAuth Administration UI.
7. Testing the OAuth configuration.
8. Troubleshooting problems that may occur if OAuth isn't working properly.

Overview of Registering OAuth Clients

An OAuth client can be confidential or public. A confidential client is an application that's capable of keeping a client password confidential to the world, whereas a public client doesn't keep a client password confidential. A confidential client can be trusted or untrusted. OAuth clients, either trusted or untrusted, can be registered using the OAuth Administration page.

Task	Description	Additional Information
Register an untrusted OAuth client.	Register a new untrusted client in the **Register Client** section of the **OAuth Administration** page in **My Services**.	Registering an Untrusted OAuth Client
Register a trusted OAuth client.	Register a new trusted client in the **Register Client** section of the **OAuth Administration** page in **My Services**.	Registering a Trusted OAuth Client

Task	Description	Additional Information
Import an OAuth certificate from a key pair.	Associate a certificate with an OAuth client. The **OAuth Administration** page provides a helper function to generate and download a key pair.	Importing an OAuth Certificate from a Key Pair
Extract a certificate by using `openssl`.	On a Linux or UNIX system, use the `openssl` command to extract the certificate from the key pair that you downloaded from the **OAuth Administration** page.	Extracting a Certificate by Using openssl
Extract a certificate by using the Certificate Import and Certificate Export wizards.	Use the Windows Certificate Import and Certificate Export wizards to extract a certificate from the generated key pair that you downloaded from the **OAuth Administration** page.	Extracting a Certificate by Using the Certificate Import and Certificate Export Wizards

Registering Client Information in OAuth

When an OAuth client is registered in OAuth, the client information is stored in the client profile.

The client information includes the client name and its attributes. An OAuth client created by an identity domain administrator is a user-managed OAuth client. If Oracle Cloud (Service Deployment Integration team) creates the OAuth client using a script, then it is a tenant-managed OAuth client. The attributes of the client stored in the client profile include whether it is tenant-managed or user-managed. The client profile also includes whether the OAuth client is trusted or untrusted. In addition to these attributes, the API path information of the client is stored in the `Audience` attribute.

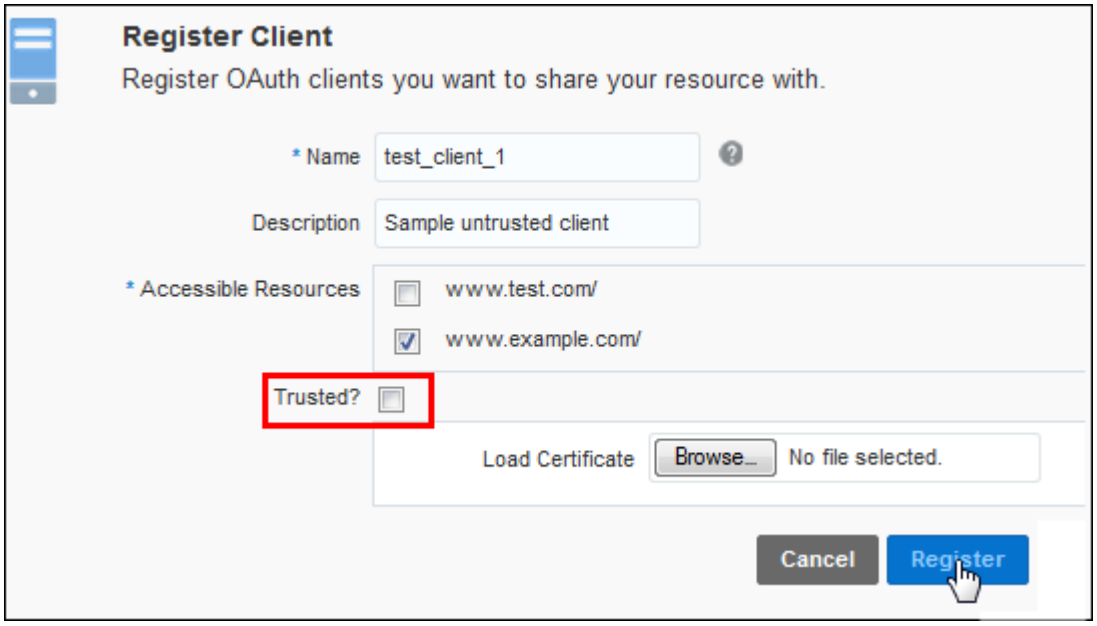
Registering an Untrusted OAuth Client

To register a new client, from **My Service**, go to the **OAuth Administration** page, and then to the **Register Client** section.

An OAuth client can be trusted or untrusted. By default, any new OAuth client created in Oracle Cloud is a confidential client (that is, the OAuth client's credentials are never exposed directly). To create an untrusted client:

1. Click **Register**. The Register Client dialog box appears. The * indicates mandatory fields.
2. In the **Name** field, specify the name of the client.
3. In the **Description** field, provide explanatory information about the OAuth client.
4. In the **Accessible Resources** field, select the API resources to which this client should have access. This corresponds with the registered resources at the bottom of the OAuth Configuration page. This is a mandatory field. Select at least one of the listed APIs. If a single API path has more than one scope defined, the scopes are listed below the API path. One or more scopes can be selected for a given API path.
5. Leave the **Trusted** check box empty because this client is an untrusted client.

6. To upload a client certificate for an untrusted client, click **Browse** next to **Load Certificate**. This isn't mandatory.
7. To register the untrusted OAuth client, click **Register**.



To learn more about the different fields in a client profile for an OAuth client, see Registering Client Information in OAuth.

A portion of the client profile for the untrusted client previously registered is in the following example. The `isTrusted` flag has a value of `FALSE`.

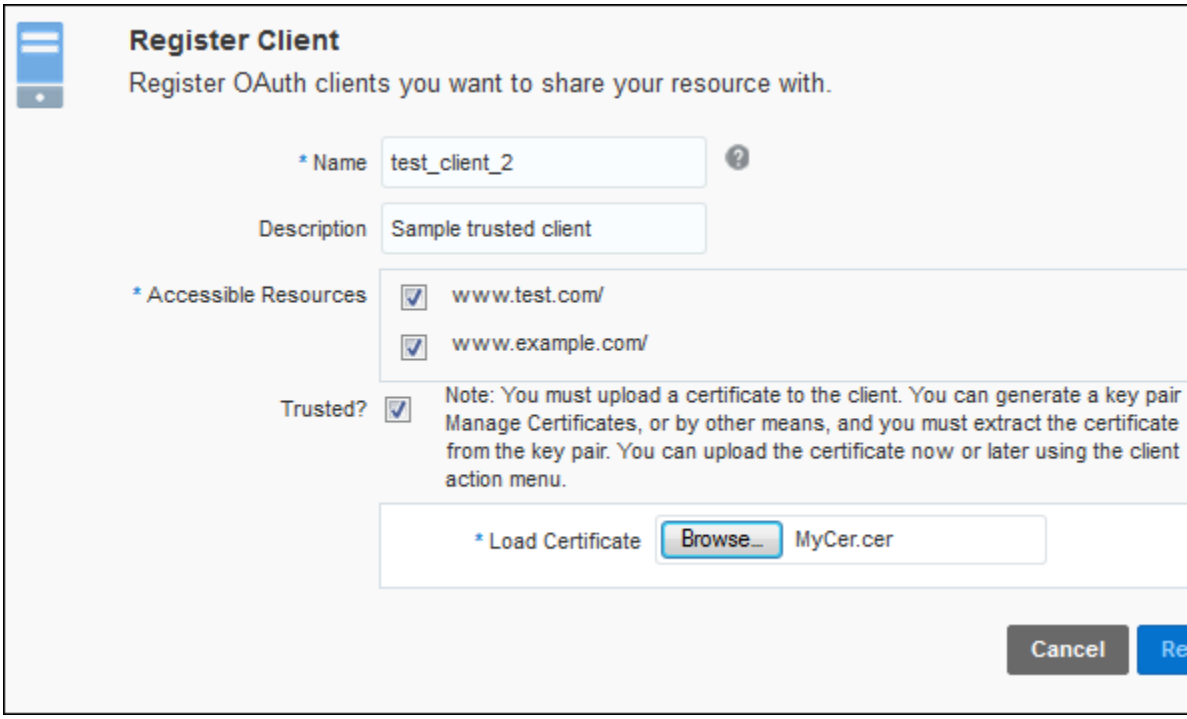
```
<tenant>dom1</tenant>
<name>test_client_1</name>
<description>Sample untrusted client</description>
<isDisabled>false</isDisabled>
<appId>948bb730-a101-43b9-9497-d3ca33eb8d7f</appId>
<appSecret>948bb730-a101-43b9-9497-d3ca33eb8d7f</appSecret>
<clientType>CONFIDENTIAL_CLIENT</clientType>
<clientMetadata>
<isTrusted>FALSE</isTrusted>
<isTenantManaged>true</isTenantManaged>
</clientMetadata>
<activityData> <createdOn>06/05/2015 02:23:18 </createdOn> </
activityData>
<audiences>http://www.example.com</audiences>
```

Registering a Trusted OAuth Client

To register a new trusted client from **My Services**, go to the **OAuth Administration** page, and then to the **Register Client** section.

An OAuth client can be trusted or untrusted. To create a trusted OAuth client:

1. Click **Register**. The Register Client dialog box appears. The * indicates mandatory fields.
2. In the **Name** field, specify the name of the client.
3. In the **Description** field, provide explanatory information about the OAuth client.
4. In the **Accessible Resources** field, select the API resources to which this client should have access. This corresponds with the registered resources at the bottom of the OAuth Configuration page. This is a mandatory field. Select at least one of the listed APIs. If a single API path has more than one scope defined, then the scopes are listed below the API path. You can select one or more scopes for a given API path.
5. To indicate that the client is trusted, select the **Trusted** check box. For a trusted client, you must generate and upload a client certificate, as described in Importing an OAuth Certificate from a Key Pair. The signing algorithm must be `RS256`: `RSASSA-PKCS-v1_5` using the SHA-256 hash algorithm.
6. Click **Register**.



To learn more about the different fields in a client profile for an OAuth client, see Registering Client Information in OAuth.

A portion of the prior client profile for the trusted client registered follows. The `isTrusted` flag has a value of `TRUE`.

```
<tenant>dom1</tenant>
<name>test_client_2</name>
<description>Sample trusted client</description>
<isDisabled>false</isDisabled>
```

```
<appId>948bb730-c201-43b9-9497-r54a33eb8d7f</appId>
<appSecret>948bb730-a101-43b9-9497-d3ca33eb8d7f</appSecret>
<clientType>CONFIDENTIAL_CLIENT</clientType>
<clientMetadata>
<isTrusted>TRUE</isTrusted>
<isTenantManaged>true</isTenantManaged>
</clientMetadata>
<activityData> <createdOn>06/05/2015 02:23:18 </createdOn> </
activityData>
<audiences>http://www.test.com/, http://www.example.com/</
audiences>
```

Importing an OAuth Certificate from a Key Pair

Import and associate an OAuth certificate with an OAuth client. This is mandatory for trusted clients and optional for untrusted clients.

The OAuth Administration page provides a helper function to generate and download a key pair, which contains a private key and the corresponding certificate. The key pair file is in the PKCS#12 format. PKCS #12 is one of the standards called Public-Key Cryptography Standards (PKCS) published by RSA Laboratories. The file name extension is usually `.p12`, but may have the older `.pfx` extension. You don't have to use the helper function. You can generate the key pair by other means. (Some applications and operating systems include key pair generators.) If you've a certificate from another signing authority, there is no need to generate a key pair.

1. From the **OAuth Administration** page, go to the **Manage Certificates** section, and then click **Create Key Pair**.

   The Generate Key Pair dialog box appears.

2. Enter the appropriate information in the **Subject DN** and **Key Store Password** fields.

3. Click **Generate**.

4. After downloading the generated key pair, extract the private key and the corresponding certificate by using a tool such as the `openssl` command-line tool on Linux or UNIX, or the Certificate Import and Certificate Export wizards on Windows. For more information, see Extracting a Certificate by Using openssl and Extracting a Certificate by Using the Certificate Import and Certificate Export Wizards. The extracted file is a DER-encoded certificate. Distinguished Encoding Rules (DER) define a set of rules for encoding. The certificate file has the extension `.cer`.

5. Extract the certificate from the key pair.

   The Import Certificate dialog box appears.

6. To associate the certificate with a specific client, see Associating a Certificate with an OAuth Client.

Store the PKCS#12 format key pair securely and don't share it. The OAuth client uses this key pair to sign OAuth protocol messages sent to the OAuth service in Oracle Cloud.

Extracting a Certificate by Using openssl

On a Linux or UNIX system, you can use the `openssl` command to extract the certificate from a key pair that you downloaded from the OAuth Configuration page.

To extract the certificate, use these commands, where *`cer`* is the file name that you want to use:

1. `openssl pkcs12 -in store.p12 -out cer.pem`

   This extracts the certificate in a `.pem` format.

2. `openssl x509 -outform der -in cer.pem -out cer.der`

   This formats the certificate in a `.der` format.

You can then associate *`cer`*`.der` with a client.

You can also extract the private key by using the command:

`openssl pkcs12 -in store.p12 -out pKey.pem -nodes -nocerts`

For more information, see the OpenSSL documentation.

Extracting a Certificate by Using the Certificate Import and Certificate Export Wizards

You can use the Windows Certificate Import and Certificate Export wizards to extract a certificate from the generated key pair that you downloaded from the OAuth Administration page.

To import the key pair and export the certificate, follow these steps:

1. Open the Certificate Import Wizard on your computer. You might be able to open it by double-clicking the key pair file. If not, follow these steps:
   a. In Internet Explorer, click **Tools**, and then select **Internet Options**.
   b. Click the **Content** tab. Click **Certificates**.
   c. Click **Import**. The Certificate Import Wizard opens.
2. Enter or browse to the file path and click **Next**. When prompted, enter the password for the private key. This was specified while generating the key pair earlier.
3. Select the **Automatically select the certificate store based on the type of certificate** check box.
4. Click **Next**.
5. Click **Finish**. A dialog box appears indicating that the key pair was imported into Internet Explorer successfully.
6. In Internet Explorer, click **Tools**, and then select **Internet Options**.

7. Click the **Content** tab, and then click **Certificates**.
8. Click the **Personal** tab, and then select the certificate that you just imported.
9. Click **Export**. The Certificate Export Wizard opens.
10. Click **Next**.
11. Select the **No, do not export the private key** check box.
12. Click **Next**.
13. Select the **DER encoded binary X.509 (CER)** check box.
14. Click **Next**.
15. Specify a file path, and then click **Save**.
16. Click **Next**.
17. To export the file to your local file system, click **Finish**. A dialog box appears indicating that the export was successful.

After completing all steps, you can associate the DER-encoded file with an OAuth client. For more information, see Associating a Certificate with an OAuth Client.

Associating a Certificate with an OAuth Client

Associate an OAuth certificate with an OAuth client. This is mandatory for trusted clients and optional for untrusted clients

To associate an OAuth certificate with a specific OAuth client:

1. From the **My Services** page, go to the **OAuth Administration** page, and then to the **Register Client** section.
2. To associate the certificate with a specific client, click the **Action** menu for that client, and then select **Import Certificate**.
3. To select the certificate file, click **Browse**, and then click **Import**.

A success message is displayed.

Overview of Managing OAuth Clients

An existing OAuth client (trusted or untrusted) can be viewed, the properties of the client can be modified, certificates associated with the client can be managed, and the client can be removed.

Task	Description	Additional Information
View OAuth clients.	An existing OAuth client can be viewed from the OAuth Administration page.	Viewing OAuth Clients
Modify OAuth clients.	The properties of an OAuth client can be changed using the OAuth Administration page.	Updating OAuth Clients

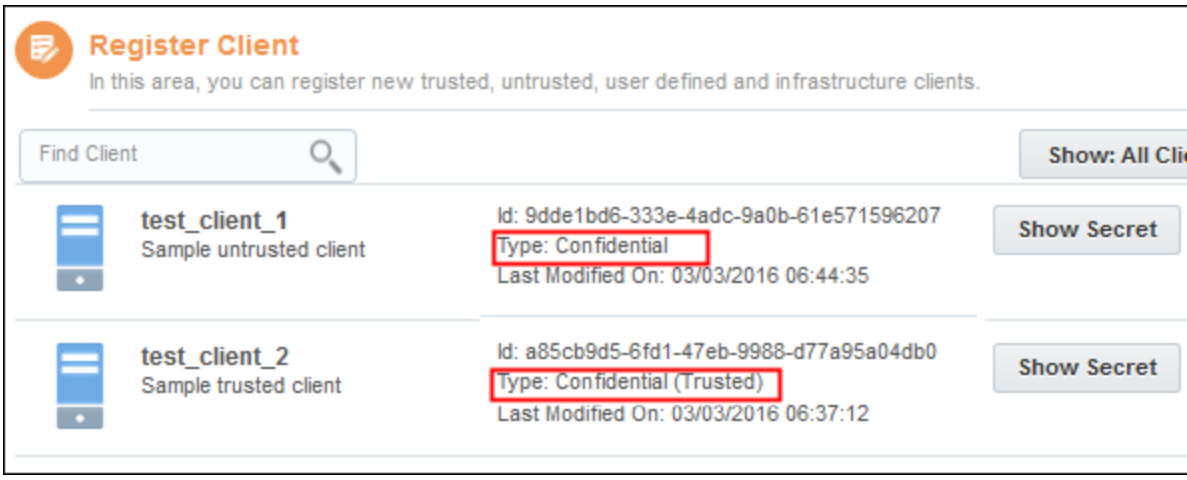
Task	Description	Additional Information
Manage client certificates.	Client certificates can be managed from the OAuth Administration page. You can access the root certificate and the OAuth certificate.	Managing Client Certificates
Delete an OAuth client.	An OAuth client can be removed by using the OAuth Administration page.	Deleting OAuth Clients

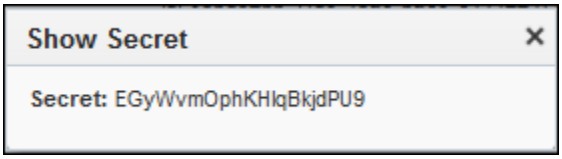
Viewing OAuth Clients

View OAuth clients on the OAuth Administration page in My Services.

The OAuth Administration page in My Services lists registered clients. You can view information about clients in the Register Client section of the page.

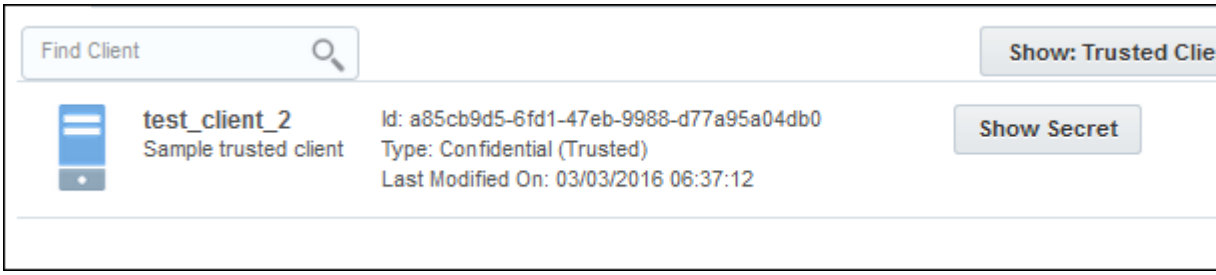
- Trusted clients are indicated with the label **Trusted.**
- The first column of the list indicates the client name.
- The second column lists the ID, type (confidential trusted or untrusted), and modification information.
- Clicking **Show Secret** displays the client secret string. The client secret is used to authenticate the identity of the client to the service API when the client requests to access a user's account, and must be kept private between the client and the API. Think of the secret string as a passphrase that proves to the authentication server that the client is authorized to make a request on behalf of the user.



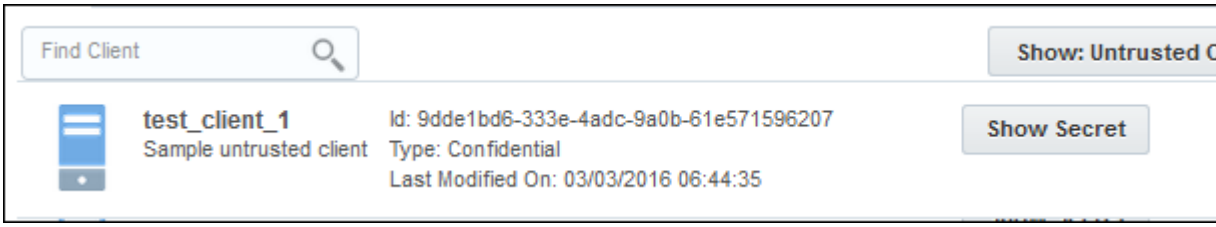


Search for a Specific OAuth Client Type

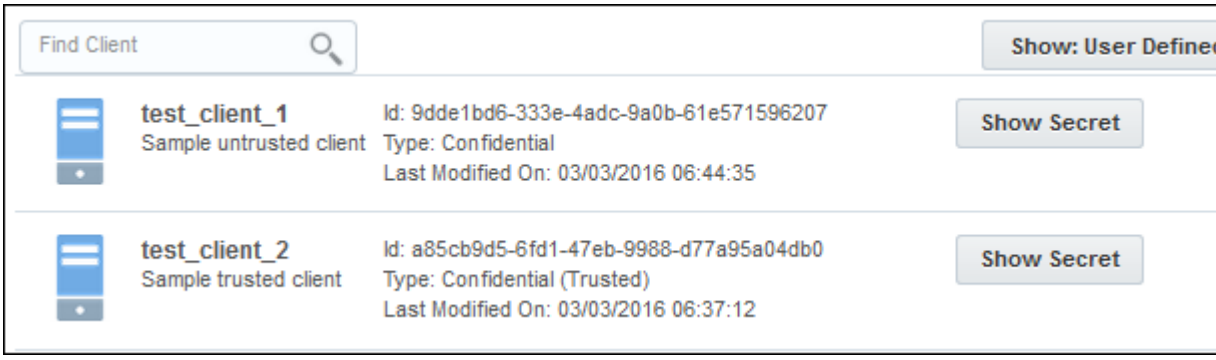
- **View All Trusted Clients:** From the drop-down menu, select **Show: Trusted Clients**. All trusted clients (which are confidential by default) are listed. A trusted client has the `Confidential (Trusted)` client type.



- **View All Untrusted Clients:** From the drop-down menu, select **Show: Untrusted Clients**. All untrusted clients (which are confidential by default) are listed below An untrusted client has the `Confidential` client type.



- **View All User-Defined Clients:** From the drop-down menu, select **Show: User Defined Clients**. A user-defined client is one that's created by an identity domain administrator. All user-defined clients, trusted and untrusted, appear.



- **View All Infrastructure Clients:** From the drop-down menu, select **Show: Infrastructure Clients**. An infrastructure client is one that's automatically created by the Service Deployment Integration (SDI) team (using a script). No infrastructure clients are available in the example.

Updating OAuth Clients

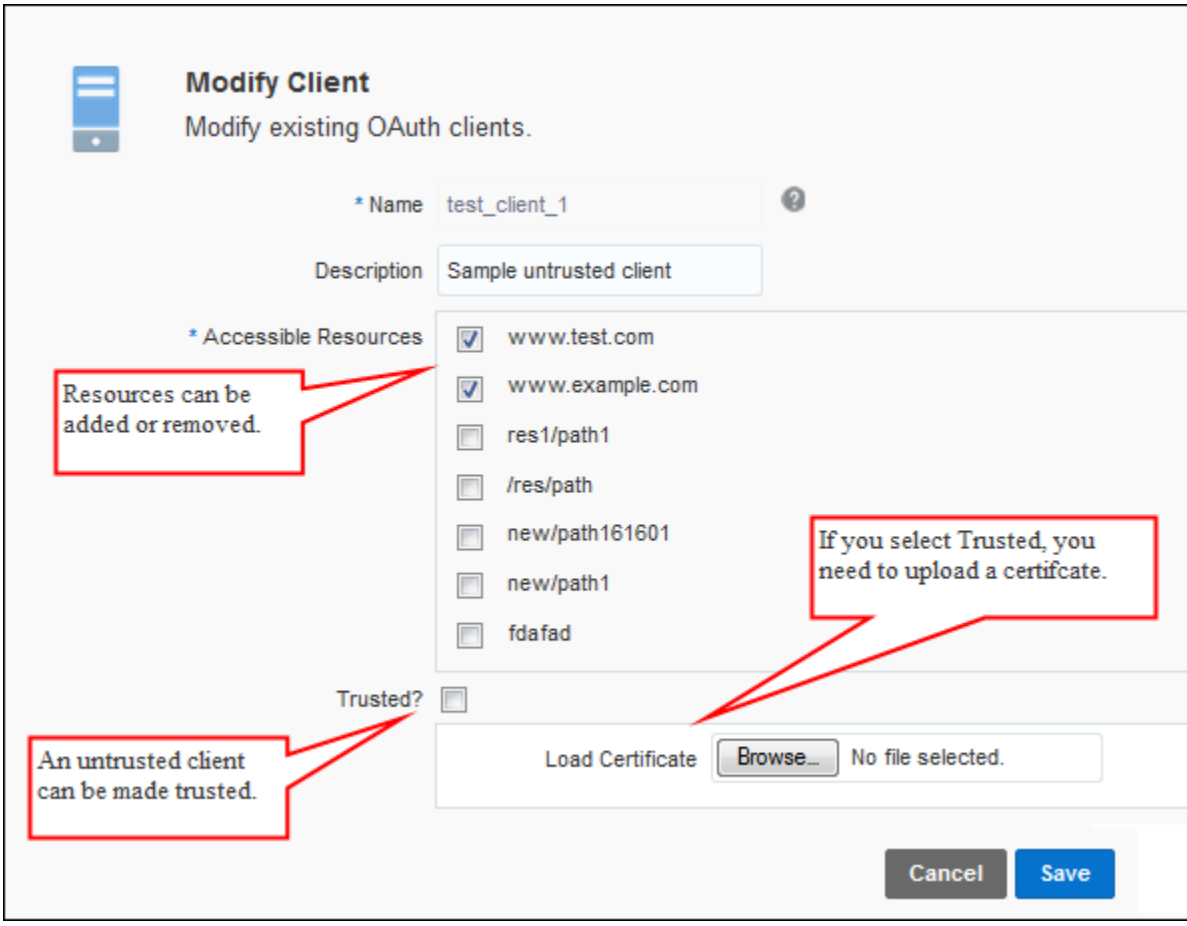
Modify the properties of an OAuth client on the Modify Clients page.

The properties of an OAuth client such as the description, can be modified. A previously untrusted OAuth client can be changed to be a trusted OAuth client, so that the client can now obtain a user token and propagate an end-user identity. APIs can be granted to or revoked from existing clients. The client certificates can be reloaded.

Modifying Properties of an Untrusted OAuth Client

You can modify the following client properties of an existing untrusted client:

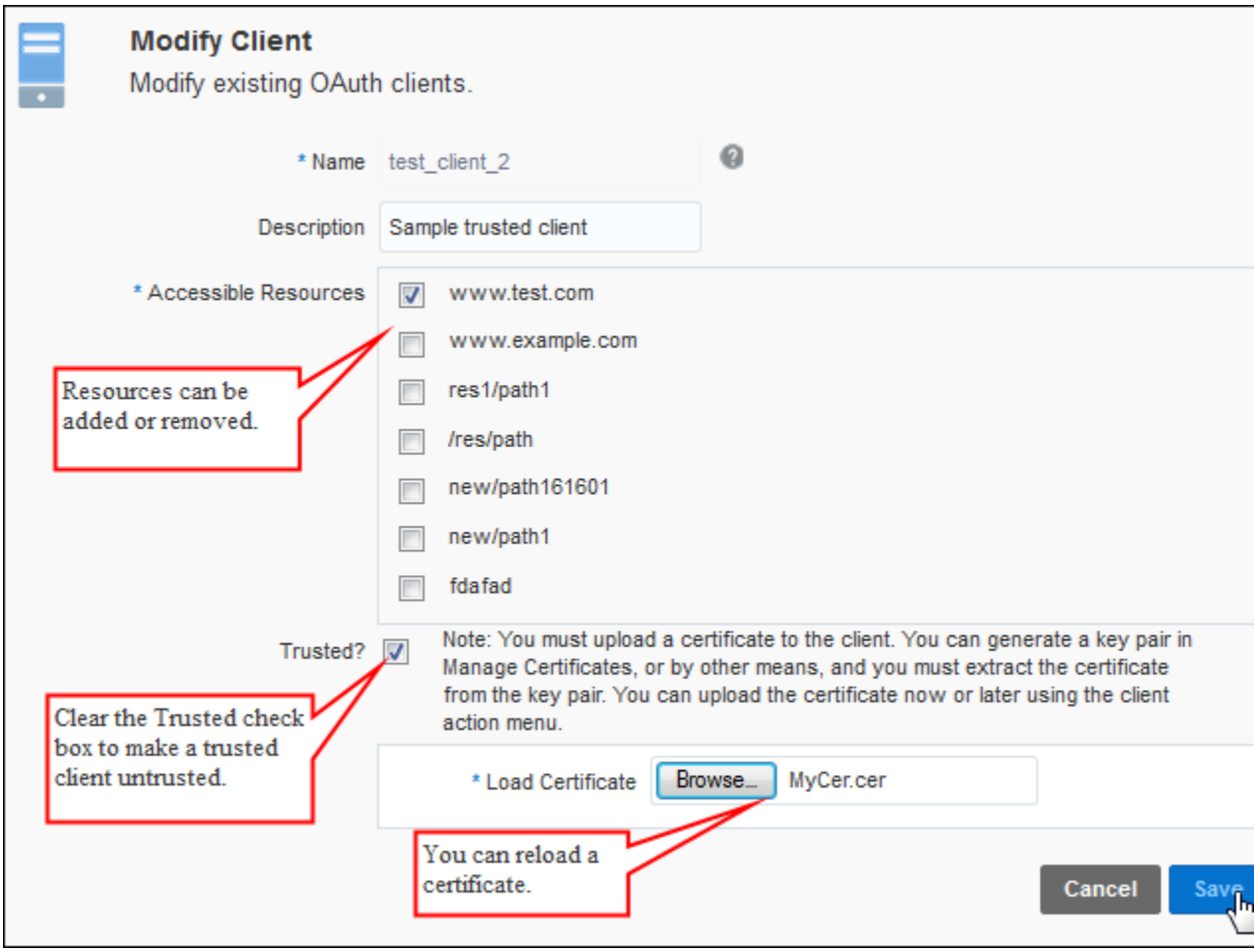
- Edit the description. The client name can't be changed.
- Add new resources or remove existing resources.
- Change the untrusted client to a trusted client by selecting the **Trusted** check box. To change the client from untrusted to trusted, you must upload a certificate.
- Upload a new certificate or use an existing certificate.



Modifying Properties of a Trusted OAuth Client

You can modify the following properties of an existing trusted client:

- Edit the description. The client name cannot be changed.
- Add new resources or remove existing resources.
- Update the client's public key (reload the client certificate).
- Change the trusted client to an untrusted client by clearing the **Trusted** check box.
- Upload a new certificate.



Overwriting the Client Profile:

The existing client profile is overwritten based on the changes made to the client. If the `api_path` is changed (added or removed), then the corresponding `Audience` attribute of the client changes.

Managing Client Certificates

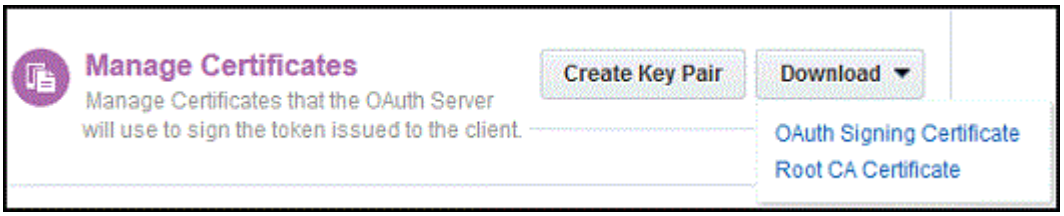
Trusted and untrusted OAuth clients can use certificates. Client certificates can be managed from the UI console.

You can fetch a root certificate authority or an OAuth signing certificate from the OAuth Administration page.

Download Existing Certificates

To download an OAuth signing certificate or a root certificate authority:

1. Log in to the user's identity domain in Oracle Cloud, and then click **Users**.
2. Go to the **OAuth Administration** page.
3. In the **Manage Certificates** section, click **Download**.



4. To save the OAuth signing certificate to a local folder, click **OAuth Signing Certificate**.
5. To save the root certificate authority to a local folder, click **Root CA Certificate**.

Deleting OAuth Clients

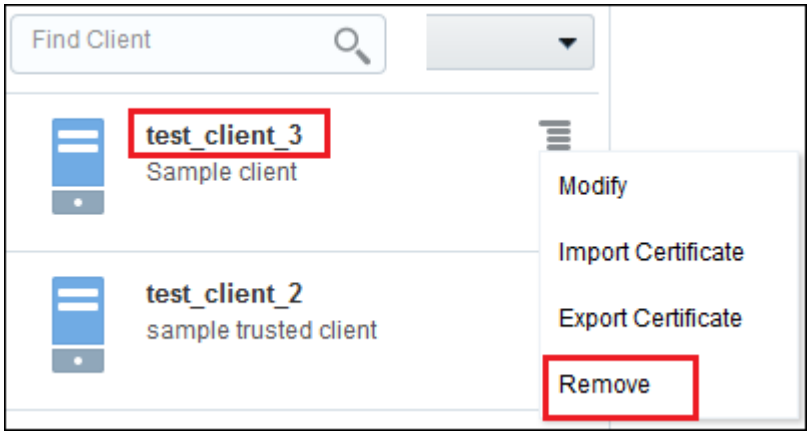
An OAuth client can be deleted at any time.

If either of the following is true, then you must delete an OAuth client and register a new one :

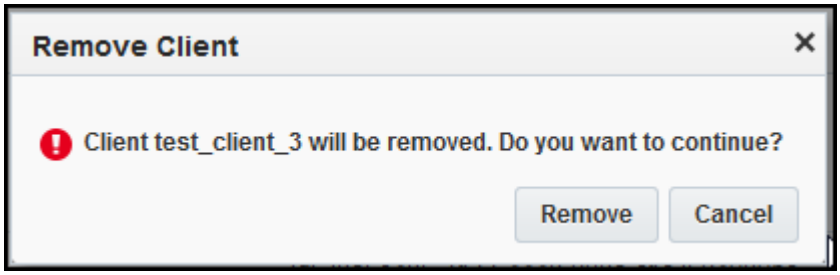
- The client's credentials (password) have been compromised.
- You want to change certain attributes of the client that can't be edited through the UI. Examples of such attributes include the client identifier and the client password.

When an OAuth client is removed, the tokens supplied earlier for the client can no longer be validated. Also, no new access tokens are provided for the client. Persistent expired tokens are removed from the database periodically.

1. To delete a client, click the **Action** menu icon to the right of the client that you want to remove. Select **Remove.**



2. The Remove Client confirmation dialog box appears. To delete the client, click **Remove**.



Troubleshooting OAuth

If OAuth isn't working properly, you must troubleshoot it.

Before contacting Oracle Support:

1. Capture the protocol message requests and responses by using a proxy or logging the messages.
2. To identify exceptions or deviations, review the protocol messages .
3. Ensure that the credentials being used are valid and that the OAuth client hasn't been disabled.
4. Access the MyOracle Support knowledge base to review known problems and determine whether the problem that you're having has been experienced by others.
5. If you've performed all these troubleshooting steps and you believe that the problem is because of Oracle Cloud, then contact Oracle Support Services. Be ready to provide all your information, including diagnostic data such as protocol message logs.

3

Securing Authorizations in Oracle Cloud

This chapter describes when and how to use authorization grants. An authorization grant is a credential representing the resource owner's authorization (to access its protected resources) used by the client to obtain an access token. The OAuth 2.0 core specification describes different authorization grants. Oracle Cloud supports the following grant types: resource owner password credentials, client credentials, and user assertion.

OAuth Endpoints

OAuth endpoints are the URLs you use to make OAuth authentication requests. The OAuth server exposes a token endpoint that you use to obtain an access token. The format of the OAuth token endpoint URL is https://<idm-domain>.identity.<data-center>.oraclecloud.com/oauth/tokens. For example, if the tenant is tenant1, the data center is Chicago, and the data center code is us2, then the OAuth token endpoint URL is: https://tenant1.identity.us2.oraclecloud.com/oauth/tokens.

Topics:

How Do I Use Authorization Grants?

In Oracle Cloud, an OAuth client makes a Representational State Transfer (REST) API call to access a protected service. As an administrator, when you secure cloud services, follow the guidelines to decide which type of authorization grant is suitable. An authorization grant is a credential representing the resource owner's authorization to

access its protected resource. The authorization grant is used by the OAuth client to obtain an access token.

Oracle Cloud supports the following grant types:

- Resource owner password credentials grant
- Client credentials grant
- User assertion grant

Guidelines to Choose an OAuth Workflow

Use the following guidelines to determine which workflow or grant type to use:

Use the resource owner password credentials workflow when:

- The OAuth clients are confidential clients.
- The resource owner has a trust relationship with the client.
- The client application doesn't need to store the credentials of the resource owner within the application or on the device.

Using the resource owner password credentials workflow, there are two ways to request an access token:

- By sending a simple client header in the token request in addition to the user's credentials. If you don't want to use a client assertion, but just the user's credentials with a basic client header, then see Obtain an Access Token by Using the User Credentials Without a Client Assertion.
- By using a client assertion in addition to the user's credentials. To use the client token and the user's credentials to request an access token, see Obtain an Access Token by Using the User Credentials and a JWT Client Assertion.

Using the client credential workflow, there are two ways to request an access token:

- By using a simple client header. If you want to use a simple client header, then see Obtain an Access Token by Using the Client Authorization Header.
- By using a client assertion. After you've a self-assigned client assertion, see Obtain an Access Token by Using a Self-Signed Client Assertion to request an access token.

Use the user assertion workflow when:

- The OAuth clients are confidential clients.
- The user's credentials should never be accessible to the client application.
- The OAuth clients are trusted to assert a user identity on behalf of the user.

Using the user assertion workflow, there are two ways to request an access token:

- By using a user assertion with a simple client header. If you want to use a simple client header with a self-signed user assertion, then see Obtain an Access Token by Using a Self-Signed User Assertion and the Client Credentials to request an access token.
- By using a user assertion with a client assertion. If you do not have a user token, you first need to build one. If you want to use a client assertion, but don't have a

client token yet, then build your own assertion. After you have a client token and a self-assigned user assertion, see Obtain an Access Token by Using a Self-Signed User Assertion and a Client Assertion to request an access token.

Resource Owner Password Credentials Workflow

When using the resource owner password credentials grant, the user provides the credentials (user name and password) directly to the application. The application then uses the credentials to obtain an access token from the OAuth token service.

The resource owner password credentials grant is a grant workflow where the client application, together with its client identifier and secret, sends the user name and password in exchange for an access token. Instead of the user having to log in and approve the authorization request in a web interface, the user can enter the user name and password in the client application UI directly. This workflow has different security properties than other OAuth workflows. The primary difference is that the user's password is accessible to the application. This requires a strong trust of the application by the user.

Security Properties

If the resource owner password credentials workflow is used, the application needs access to the user's credentials only once, on first use, when the credentials are exchanged for an access token. This means that there is no requirement for the application to store these credentials within the application or on the device, and revoking access is easy as well.

Key Characteristics of the Resource Owner Password Credentials Grant

The resource owner password credentials grant:

- Is used with confidential clients.
- Uses the user name and password of the resource owner.
- Isn't redirection-based; it takes a request only from the client application to the authorization server, and the user isn't redirected between interfaces to authorize the request.

Step-by-Step Workflow of the Resource Owner Password Credentials Grant

The resource owner password credentials grant workflow allows for the exchanging of the user name and password of a user for an access token.

When using the resource owner password credentials grant, the user provides the credentials (user name and password) directly to the application. The application then uses the credentials to obtain an access token from the service.

Workflow of Resource Owner Password Credentials Grant

1. **Obtain user credentials:** The user provides the credentials to the application. The user credentials are the resource owner's user name and password.
2. **Request an access token:** The user credentials are exchanged for an access token. The client application makes a request to the authorization server and includes the user's credentials and either the client credentials or a client assertion. The client application can use an already-generated client assertion or build a new assertion.

Obtain an access token by using different scenarios in the resource owner password credentials workflow:

- Obtaining an Access Token by Using the User Credentials Without a Client Assertion
- Obtaining an Access Token by Using User Credentials and JWT Client Assertion

3. **Receive an access token from the authorization server**: The authorization server authenticates the client based on the client identifier and secret, determines whether it's authorized for making this request, and verifies that the resource owner credentials and other parameters are supplied. If everything is verified successfully, then the authorization server returns an access token in the response. This is described in Successful Authorization.

   If the authorization request fails for any reason, then the authorization server returns a response containing the information about the error. This is described in Authorization Error.

4. **Use the access token to make a service request**: The OAuth client makes a REST API call to the resource server using the access token to access the protected resource.
5. **Send a response:** The resource server sends a response to the service request.
6. **Grant access to the resource:** The enduser or service gets access to the protected resource.

Using REST API Calls for the Resource Owner Password Credentials Grant

Get an access token by using different scenarios in the resource owner password credentials workflow. These scenarios include using the user's credentials with either the client credentials or a client assertion.

Task	Description	Additional Information
Obtain an access token by using the user's credentials and the client credentials.	Get the access token by providing the resource owner's user name and password and the client credentials.	Obtaining an Access Token by Using the User Credentials Without a Client Assertion
Obtain an access token by using the user's credentials and a client assertion.	Get the access token by providing the resource owner's user name and password and a client assertion.	Obtaining an Access Token by Using the User Credentials and a JWT Client Assertion

Obtaining an Access Token by Using the User Credentials Without a Client Assertion

Using the resource owner password credentials workflow, the OAuth client can obtain an access token by providing the user's credentials (that is the user name and password).

This workflow has a resource owner request that uses the user identifier and password of the resource owner.

Obtaining an Access Token by Using the User Credentials

The resource owner password credentials grant workflow, allows you to obtain an access token by using the user's credentials.

Parameters used in the access token request:

- **X-USER-IDENTITY-DOMAIN-NAME**: The name of the identity domain.
- **Authorization: Basic:** The basic authorization header. The client identifier and client secret of the client application is base64–encoded and sent in the header. For example, the Authorization header has the value of `base64encoded(client_id:client_secret)`.
- **Content-Type:** The type of content that's sent in the request. It's a URL-encoded application.
- **Request:** The type of request that's sent. Here, it's a `POST` request to obtain an access token. This is followed by the authorization server URL which provides tokens.
- **grant_type:** The grant type used to obtain the token. In the example that follows, it's resource owner password credentials grant. The value of `password` is given for this grant type.
- **username:** The name of the user.
- **password:** The name of the password.
- **scope:** The limit of a particular scope for an access token.

The client identifier and client secret of the client application is base64–encoded and sent in the header. This is sent along with the user's credentials to obtain an access token.

To obtain an access token that contains the user and client credentials, use the following `cURL` command:

```
curl -i -H 'X-USER-IDENTITY-DOMAIN-NAME: OAuthTestTenant125'
-H 'Authorization: Basic
MzAzYTI0OTItZDY0Zi00ZTA0LWI3OGYtYjQzMzAwNDczMTJiOll5Sk5NSkdFc0ZqUkxWZVZsdVMz'
-H 'Content-Type: application/x-www-form-urlencoded;charset=UTF-8'
--request POST https://<idm-domain>.identity.<data-center>.oraclecloud.com/oauth/
tokens
-d 'grant_type=password
&username=tenantAdminUser
&password=Fusionapps1
&scope=http://www.example.com'
```

The output of the `cURL` command is:

```
{
"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsIng1dCI6Ild3cmVwdTJkYXNhSXBHUi1Bb
FZwSGtVQjZKZyIsIm
tpZCI6Ik9BdXRoVGVzdFRlbmFudDEyNS5jZXJ0In0.eyJzdWIiOiJ0ZW5hbnRBZG1pblVzZXIiLCJvcmFjbGU
ub2F1dGgudXNlcl9v
cmlnaW5faWRfdHlwZSI6IkxEQVBfVUlEIiwib3JhY2xlLm9hdXRoLnVzZXJfb3JpZ2luX2lkIjoidGVuYW50Q
WRtaW5Vc2VyIiwiaX
NzIjoiT0F1dGhUZXN0VGVuYW50MTI1Iiwib3JhY2xlLm9hdXRoLnN2Y19wX24iOiJPQXV0aFRlc3RUZW5hbnQ
```

xMjVTZXJ2aWNlUHJv
ZmlsZSIsImlhdCI6MTQyNTQyMzYxOTAwMCwib3JhY2xlLm9hdXRoLnBybi5pZF90eXBlIjoiTERBUF9VSUQiL
CJvcmFjbGUub2F1dG
gudGtfY29udGV4dCI6InJlc291cmNlX2FjY2Vzc190ayIsImV4cCI6MTQyNTQyNzIxOTAwMCwiYXVkIjpbImh
0dHA6Ly93d3cuZXhh
bXBsZS5jb20iXSwicHJuIjoidGVuYW50QWRtaW5Vc2VyIiwianRpIjoiZmE0NmNjNzktOTFhYi00NTUzLTk2Z
DQtYTRhZWYxYzhmZm
M3Iiwib3JhY2xlLm9hdXRoLmNsaWVudF9vcmlnaW5faWQiOiIzMDNhMjQ5Mi1kNjRmLTRlMDQtYjc4Zi1iNDM
zMDA0NzMxMmIiLCJv
cmFjbGUub2F1dGguc2NvcGUiOiJodHRwOi8vd3d3LmV4YW1wbGUuY29tIiwidXNlci50ZW5hbnQubmFtZSI6I
k9BdXRoVGVzdFRlbm
FudDEyNSIsIm9yYWNsZS5vYXV0aC5pZF9kX2lkIjoiMTM0NjM2NzUxMzgzMDI1NjYifQ.XP4cprA98Zx_k2Ua
Gx5bYn_GDVJkzfvYo
BnXLZVWYdMB_Sgkf8sS2i-lOHFuMqqwFvpYu4EccppQ8nWZYoMIJ_ZBnA3kvwosn1j-
_DGYr2aukuBrlM53mfkQlQ0kwd_zrVfJNVH
jaaw3grs0rX52NCRwWhxgE30OOahjj4ErWE64L9tDE0kz7YyKFAcPp6tPgQFO8eikVnlhgJAgexq5vABXa_QS
FyRhcsmk0kyNi9FtX
AKu4GEh_e0waQ9KGjeqzgwCSfRzriJ9otZjofbXca6dZpCxEWhfL06VHFiW-33GQpenpYCA-
x8XWmQtUR6c_jkE6TlMPp7AzR32Qud
nAA"}

The JavaScript Object Notation web token (JWT) obtained can be decoded and the claims in the access token can be viewed as follows:

Access Token:

```
{
 alg: "RS256",
 typ: "JWT",
 x5t: "Wwrepu2dasaIpGR-AlVpHkUB6Jg",
 kid: "OAuthTestTenant125.cert"
}.
{
 sub: "tenantAdminUser",
 oracle.oauth.user_origin_id_type: "LDAP_UID",
 oracle.oauth.user_origin_id: "tenantAdminUser",
 iss: "OAuthTestTenant125",
 oracle.oauth.svc_p_n: "OAuthTestTenant125ServiceProfile",
 iat: 1425423619000,
 oracle.oauth.prn.id_type: "LDAP_UID",
 oracle.oauth.tk_context: "resource_access_tk",
 exp: 1425427219000,
 aud: [
  "http://www.example.com"
 ],
 prn: "tenantAdminUser",
 jti: "fa46cc79-91ab-4553-96d4-a4aef1c8ffc7",
 oracle.oauth.client_origin_id: "303a2492-d64f-4e04-b78f-b4330047312b",
 oracle.oauth.scope: "http://www.example.com",
 user.tenant.name: "OAuthTestTenant125",
 oracle.oauth.id_d_id: "13463675138302566"
}.
[signature]
```

Audience and scope claims in the output:

The audience claim in an access token always contains the API path of the resource. The `oracle.oauth.scope` claim contains the valid API path with the scope in the response. In the example above, the incoming request has a scope of `http://www.example.com`. The client audience configuration also has a value of `http://www.example.com`. The OAuth token service validates the incoming request scope

with the value in the client audience configuration values. If this is a valid request, then the OAuth token service sends a valid access token response. In this case, both the audience claim and scope have the same value of `http://www.example.com`.

Obtaining an Access Token by Using the User Credentials and a JWT Client Assertion

The OAuth client can request an access token by providing the user's credentials (that is, the user name and password) and a JSON web token (JWT) client assertion.

This workflow has a resource owner request that uses the user identifier and password of the resource owner, and a JWT client assertion generated by a third party. When using the resource owner password credentials grant workflow, you can obtain an access token by providing the user's credentials and a client assertion. See Step-by-Step Workflow of the Client Credentials Grant to identify the claims that need to be part of the client assertion.

Parameters used in the access token request:

- **X-USER-IDENTITY-DOMAIN-NAME**: The name of the identity domain.
- **Content-Type:** The type of content that's sent in the request. It's a URL-encoded application.
- **Request:** The type of request that's sent. The example uses a `POST` request to obtain an access token. This is followed by the authorization server URL, which provides tokens.
- **grant_type:** The grant type used to obtain the token. In the example that follows, the grant type is resource owner password credentials grant. The value of `password` is given for this grant type.
- **username:** The name of the user.
- **password:** The name of the password.
- **client_assertion_type:** The type of client assertion. In Oracle Cloud, it is `jwt_bearer`.
- **client_assertion:** The value of the client token obtained.
- **scope:** The limit of a particular scope for an access token.

The client credentials are available in the form of an already-generated JWT client assertion. This is sent along with the user's credentials to obtain an access token.

To obtain an access token, use the following `cURL` command:

```
curl -i -H 'X-USER-IDENTITY-DOMAIN-NAME: OAuthTestTenant125'
-H 'Authorization: Basic
MzAzYTI0OTItZDY0Zi00ZTA0LWI3OGYtYjQzMzAwNDczMTJiOll5Sk5NSkdFc0ZqUkxWZVZsdVMz'
-H 'Content-Type: application/x-www-form-urlencoded;charset=UTF-8'
--request POST https://<idm-domain>.identity.<data-center>.oraclecloud.com/oauth/
tokens
-d 'grant_type=password
&username=tenantAdminUser
&password=Fusionapps1
&client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-
bearer
&client_assertion=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsIng1dCI6Ild3cmVwdTJkYXNhSXBHUi1
BbFZwSGtVQjZK
ZyIsImtpZCI6Ik9BdXRoVGVzdFRlbmFudDEyNS5jZXJ0In0.eyJvcmFjbGUub2F1dGgudGtfY29udGV4dCI6I
```

```
mNsaWVudF9hc3Nlc
nRpb24iLCJleHAiOjE0MjYwMzI4MzgwMDAsInN1YiI6IjMwM2EyNDkyLWQ2NGYtNGUwNC1iNzhmLWI0MzMwMD
Q3MzEyYiIsImlzcy
I6Ik9BdXRoVGVzdFRlbmFudDEyNSIsInBybiI6IjMwM2EyNDkyLWQ2NGYtNGUwNC1iNzhmLWI0MzMwMDQ3MzE
yYiIsImp0aSI6IjY
yNzZhYTI0LTUxNjQtNGEwZC1iYzQxLTlmMzVjMGU1ZjgxZiIsIm9yYWNsZS5vYXV0aC5zdmNfcF9uIjoiT0F1
dGhUZXN0VGVuYW50
MTI1U2VydmljZVByb2ZpbGUiLCJpYXQiOjE0MjU0MjgwMzgwMDAsIm9yYWNsZS5vYXV0aC5pZF9kX2lkIjoiM
TM0NjM2NzUxMzgzM
DI1NjYiLCJ1c2VyLnRlbmFudC5uYW1lIjoiT0F1dGhUZXN0VGVuYW50MTI1Iiwib3JhY2xlLm9hdXRoLnBybi
5pZF90eXBlIjoiQ2
xpZW50SUQifQ.OCHS9FhKJEXpIg3IvE6qWdTz3tRY449LZoBAcc3yDoaMbjS4CZxDDuKx6MUBpHmkmVoHRZSm
krILOzel51sT_kjE
HfNtzwMCIs2re_JcSfGkvnzv0aCV1r_V5dvmmZulhGaOUTu9nkEFzCq-JNa23eO_dEq8jfP7-
Y7H2KGMvuC5lHGGQViw1ega-4mFu
ZBJlSvzEqDcYIPde0m8gSUF--IFuiovgGTKCe97-0MF34za6SZ0HJv9p3WesvCS8YV1bcWVwTGEXCZ3qA1mA-
IOKvaMZNOxM_D9tT
5KVCub-i-H6r0uHpkovOCzunffcuL4cOg5ptrFv-abn-JP47eNag
&scope=http://www.example.com'
```

The output of the cURL command is:

```
{
"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsIng1dCI6Ild3cmVwdTJkYXNhSXBHUi1Bb
FZwSGtVQjZKZyIs
ImtpZCI6Ik9BdXRoVGVzdFRlbmFudDEyNS5jZXJ0In0.eyJzdWIiOiJ0ZW5hbnRBZG1pblVzZXIiLCJvcmFjb
GUub2F1dGgudXNl
cl9vcmlnaW5faWRfdHlwZSI6IkxEQVBfVUlEIiwib3JhY2xlLm9hdXRoLnVzZXJfb3JpZ2luX2lkIjoidGVuY
W50QWRtaW5Vc2Vy
IiwiaXNzIjoiT0F1dGhUZXN0VGVuYW50MTI1Iiwib3JhY2xlLm9hdXRoLnN2Y19wX24iOiJPQXV0aFRlc3RUZ
W5hbnQxMjVTZXJ2
aWNlUHJvZmlsZSIsImlhdCI6MTQyNTQyODA2ODAwMCwib3JhY2xlLm9hdXRoLnBybi5pZF90eXBlIjoiTERBU
F9VSUQiLCJvcmFj
bGUub2F1dGgudGtfY29udGV4dCI6InJlc291cmNlX2FjY2Vzc190ayIsImV4cCI6MTQyNTQzMTY2ODAwMCwiY
XVkIjpbImh0dHA6
Ly93d3cuZXhhbXBsZS5jb20iXSwicHJuIjoidGVuYW50QWRtaW5Vc2VyIiwianRpIjoiMGZmNTM4NzQtZTJhY
S00OTYwLTg0NTU
tZWUzZjQ2N2ZjNDEzIiwib3JhY2xlLm9hdXRoLmNsaWVudF9vcmlnaW5faWQiOiIzMDNhMjQ5Mi1kNjRmLTRl
MDQtYjc4Zi1iNDM
zMDA0NzMxMmIiLCJvcmFjbGUub2F1dGguc2NvcGUiOiJodHRwOi8vd3d3LmV4YW1wbGUuY29tIiwidXNlci50
ZW5hbnQubmFtZSI
6Ik9BdXRoVGVzdFRlbmFudDEyNSIsIm9yYWNsZS5vYXV0aC5pZF9kX2lkIjoiMTM0NjM2NzUxMzgzMDI1NjYi
fQ.q9yLvZ9RTX1d
fJ-B8bcR6sQHKylSuXol02tOTbCmuRKgb9Ej5QxjPea35_Y0bMlOiRaUPk-
elE0tpx7m4b7tLsCHDYo2YtRWkOrKbSPWyVulPsXA
rTvtiy3qz5UX4mLhXXbWRwxUfNuUmUTen7hhqigJzxk_V3_BO85OT57aQBCp4QYBJ7HOmCeVjR4McyufTZEsR
L8v_D9CP85IxF9G
MrxZzLD8-
VTprmcirgAOlwdDUlWRMUtBwAmui5jfMJK0K0tkC8ABaUqPA58Af_HjUCT8wn0qY_b1mJH0tq0Is5i_n9tJ3f
DjxcQB
5yoAGMBp2CQgb4b1Hh7BWo5TY8HneA"
}
```

The JSON web token (JWT) obtained can be decoded and the claims in the access token can be viewed as follows:

Access token:

```
{
 alg: "RS256",
 typ: "JWT",
 x5t: "Wwrepu2dasaIpGR-AlVpHkUB6Jg",
 kid: "OAuthTestTenant125.cert"
}.
{
 sub: "tenantAdminUser",
 oracle.oauth.user_origin_id_type: "LDAP_UID",
 oracle.oauth.user_origin_id: "tenantAdminUser",
 iss: "OAuthTestTenant125",
 oracle.oauth.svc_p_n: "OAuthTestTenant125ServiceProfile",
 iat: 1425423619000,
 oracle.oauth.prn.id_type: "LDAP_UID",
 oracle.oauth.tk_context: "resource_access_tk",
 exp: 1425427219000,
 aud: [
  "http://www.example.com"
 ],
 prn: "tenantAdminUser",
 jti: "fa46cc79-91ab-4553-96d4-a4aef1c8ffc7",
 oracle.oauth.client_origin_id: "303a2492-d64f-4e04-b78f-b4330047312b",
 oracle.oauth.scope: "http://www.example.com",
 user.tenant.name: "OAuthTestTenant125",
 oracle.oauth.id_d_id: "13463675138302566"
}.
[signature]
```

Audience and scope claims in the output:

The audience claim in an access token contains the API path of the resource. The `oracle.oauth.scope` claim contains the valid API path with the scope in the response. In the prior example, the incoming request has a scope of `http://www.example.com`. The client audience configuration also has a value of `http://www.example.com::*`. The OAuth token service validates the incoming request scope with the value in the client audience configuration values. Because this is a valid request, the OAuth token service sends a valid access token in the response. In this case, the audience claim has a value of `http://www.example.com` and the scope has a value of `http://www.example.com`.

Client Credentials Grant Workflow

Use the client credentials grant when the client itself owns the data and doesn't need delegated access from a resource owner, or the delegated access has already been granted to the application outside of a typical OAuth workflow.

The client credentials grant provides a specific grant flow in which the resource owner (that is, the user) is not involved. When using this grant, the client application requests an access token only with its own credentials (the identifier and secret) or an assertion, and uses the access token on behalf of the client application itself. This grant flow is best-suited when a service provider wants to provide some API methods that are to be used by the client application in general, instead of methods that apply to a certain resource owner, for example, API methods for maintenance. This way of using an API is also referred to as *userless* access.

Security Properties

Depending on the use case for which you want to use the client credentials grant flow, a single set of credentials for a client could provide access to a large amount of data. The more data a single set of credentials has access to, the greater the risk if the

credentials become compromised. It's critical that the credentials used to authenticate the client are kept confidential. Ideally, these credentials would also be rotated regularly.

Key Characteristics of the Client Credentials Grant Type

- It's used by confidential clients.
- The flow is not redirection-based.
- It's useful in cases where the client application communicates with the service provider directly and not on behalf of a resource owner.
- The resource owner isn't part of the flow.

Step-by-Step Workflow of the Client Credentials Grant

The client credentials grant workflow relies on the client being able to properly authenticate with the authorization server and the client's authentication credentials remaining confidential.

When using the client credentials grant workflow, only the client details are used for authentication and there is no resource owner.

Workflow of the Client Credentials Grant

1. **Request an access token:** The client credentials are exchanged for an access token. The client application makes a request to the authorization server, including the HTTP basic authentication header and optionally a client assertion. The client application can use an already-generated client assertion or build a new assertion. The client assertion is a standard JSON web token (JWT), to be signed by a trusted or confidential client using its private key. Verify that the following claims are part of the client assertion:

   Header

   ```
   {
   "alg": "RS256",
   "typ": "JWT",
   "x5t": "<X5t of the certificate>"
   }
   ```

   Body

   ```
   {
   "exp": <Expiry Time in seconds>,
   "sub": "<clientId>",
   "aud": [ "oauth.idm.oracle.com" ],
   "iss": "<clientId>",
   "oracle.oauth.sub.id_type": "ClientID",
   "prn": "<clientId>",
   "jti": "<Globally Unique Id representing the token>",
   "iat": <Issued at Time in seconds>,
   "user.tenant.name": "<tenantname>",
   "oracle.oauth.prn.id_type": "ClientID"
   }
   ```

   Obtain an access token by using different scenarios in the client credentials workflow:

   - Obtaining an Access Token Using the Client Authorization Header

- Obtaining an Access Token Using a Self-Signed Client Assertion

2. **Receive an access token from the authorization server:** The authorization server authenticates the client based on the authorization header or assertion sent and makes a response. If the client is authenticated and the parameters supplied are valid, then the client gets an access token as the response. This is described in Successful Authorization

   If the authorization request fails for any reason, then the authorization server returns a response containing information about the error. This is described in Authorization Error

3. **Use the access token to make a service request:** The OAuth client makes a REST API call to the resource server using the access token to access the protected resource.

4. **Send a response:** The OAuth resource server sends a response to the service request.

5. **Grant access to the resource:** The client (enduser) or service gets access to the protected resource.

Using REST API Calls for the Client Credentials Grant

Get an access token by using the client authorization header, or a self-signed client assertion.

Task	Description	Additional Information
Obtain an access token by using a client authorization header.	The client application sends an authorization basic header to obtain the access token.	Obtaining an Access Token by Using a Client Authorization Header
Obtain an access token by using a self-signed client assertion.	The client application uses a self-signed client assertion as part of the request to obtain the access token.	Obtaining an Access Token by Using a Self-Signed Client Assertion

Obtaining an Access Token by Using a Client Authorization Header

The client credentials workflow allows the client application to obtain an access token by using the basic authorization header.

In Oracle Cloud, all OAuth clients are confidential by default and so their credentials (`client_id` and `password`) are never exposed. The `client_id` and `password` credentials are encoded and sent in the basic authorization header. The format used to obtain the header value is `base64encoded(client_id:password)`.

Parameters used in the access token request:

- **X-USER-IDENTITY-DOMAIN-NAME**: The name of the identity domain.
- **Authorization: Basic:** The basic authorization header. The client identifier and client secret of the client application are base64–encoded and sent in the header. For example, the authorization header has the value of `base64encoded(client_id:password)`.
- **Content-Type:** The type of content that's sent in the request. It is a URL-encoded application.

- **Request:** The type of request that's sent. In the example that follows, a `POST` request is used to obtain an access token. This is followed by the authorization server URL, which provides tokens.
- **grant_type:** The grant type used to obtain the token. In the example that follows, the grant type is client credentials. The value of `client_credentials` is given for this grant type.
- **scope:** The limit of a particular scope for an access token.

The client identifier and password are encoded and sent in the basic authorization header. This is sent to obtain an access token.

To obtain an access token by providing the client credentials, use the following `cURL` command :

```
curl -i -H 'X-USER-IDENTITY-DOMAIN-NAME: OAuthTestTenant125'
-H 'Authorization: Basic
MzAzYTI0OTItZDY0Zi00ZTA0LWI3OGYtYjQzMzAwNDczMTJiOll5Sk5NSkdFc0ZqUkxWZVZsdVMz'
-H 'Content-Type: application/x-www-form-urlencoded;charset=UTF-8'
--request POST https://<idm-domain>.identity.<data-center>.oraclecloud.com/oauth/
tokens
-d 'grant_type=client_credentials
&scope=http://www.example.com'
```

The output of the `cURL` command is:

```
{
"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsIng1dCI6Ild3cmVwdTJkYXNhSXBHUi1Bb
FZwSGtVQjZKZyIsImtp
ZCI6Ik9BdXRoVGVzdFRlbmFudDEyNS5jZXJ0In0.eyJzdWIiOiIzMDNhMjQ5Mi1kNjRmLTRlMDQtYjc4Zi1iN
DMzMDA0NzMxMmIiLCJp
c3MiOiJPQXV0aFRlc3RUZW5hbnQxMjUiLCJvcmFjbGUub2F1dGguc3ZjX3BfbiI6Ik9BdXRoVGVzdFRlbmFud
DEyNVNlcnZpY2VQcm9m
aWxlIiwiaWF0IjoxNDI1NDIyMDk0MDAwLCJvcmFjbGUub2F1dGgucHJuLmlkX3R5cGUiOiJDbGllbnRJRCIsI
mV4cCI6MTQyNTQyNTY5
NDAwMCwib3JhY2xlLm9hdXRoLnRrX2NvbnRleHQiOiJyZXNvdXJjZV9hY2Nlc3NfdGsiLCJhdWQiOlsiaHR0c
DovL3d3dy5leGFtcGxl
LmNvbSJdLCJwcm4iOiIzMDNhMjQ5Mi1kNjRmLTRlMDQtYjc4Zi1iNDMzMDA0NzMxMmIiLCJqdGkiOiJlODE0N
jgyOC1kNWIyLTQxNjkt
ODU3Ny03MTJmODM2YjcyNDEiLCJvcmFjbGUub2F1dGguY2xpZW50X29yaWdpbl9pZCI6IjMwM2EyNDkyLWQ2N
GYtNGUwNC1iNzhmLWI0
MzMwMDQ3MzEyYiIsIm9yYWNsZS5vYXV0aC5zY29wZSI6Imh0dHA6Ly93d3cuZXhhbXBsZS5jb20iLCJ1c2VyL
nRlbmFudC5uYW1lIjoi
T0F1dGhUZXN0VGVuYW50MTI1Iiwib3JhY2xlLm9hdXRoLmlkX2RfaWQiOiIxMzQ2MzY3NTEzODMwMjU2NiJ9.
hM6nwuhj8h7L7vK6ozc
mm-xcyVoFbCkLln8oZZPPlIpf7o-Bahj3J5vcgqTuBl89pVGR-ly6m2AH-
v0yOdWD2Qfo8pnU14B3o01Z7U_vkxu_pc_3qz5P5Jk0rDm
xNT3iVeNK2rht41aXdQBiXJkGOAkUXpI6_MzK5kkY0zHm7PnybkipTfmUy8jEyRhwYvBGFsWvTGp5nIUP6zrm
TfxAb2q-hgFFV1HIGqm
7uGXZA1c7svXttkwEuajV3bWkaaFUe8YGuiokR-
nraoOvs18WpY08eSoiBPu8eTdp_ff6WWpjmMRe4YHvYNWmkbVxP3XD1sRtClLsgb6
eSqVwIK4HFQ"
}
```

The JSON web token (JWT) obtained can be decoded and the claims in the access token can be viewed as follows:

Access token:

```
{
 alg: "RS256",
 typ: "JWT",
 x5t: "Wwrepu2dasaIpGR-AlVpHkUB6Jg",
 kid: "OAuthTestTenant125.cert"
}.
{
 sub: "303a2492-d64f-4e04-b78f-b4330047312b",
 iss: "OAuthTestTenant125",
 oracle.oauth.svc_p_n: "OAuthTestTenant125ServiceProfile",
 iat: 1425422094000,
 oracle.oauth.prn.id_type: "ClientID",
 exp: 1425425694000,
 oracle.oauth.tk_context: "resource_access_tk",
 aud: [
  "http://www.example.com"
 ],
 prn: "303a2492-d64f-4e04-b78f-b4330047312b",
 jti: "e8146828-d5b2-4169-8577-712f836b7241",
 oracle.oauth.client_origin_id: "303a2492-d64f-4e04-b78f-b4330047312b",
 oracle.oauth.scope: "http://www.example.com",
 user.tenant.name: "OAuthTestTenant125",
 oracle.oauth.id_d_id: "13463675138302566"
}.
[signature]
```

Audience and scope claims in the output:

The audience claim in an access token always contains the API path of the resource. The `oracle.oauth.scope` claim contains the valid API path with the scope in the response. In the prior example, the incoming request has a scope of `http://www.example.com`. The client audience configuration also has a value of `http://www.example.com`. The OAuth token service validates the incoming request scope with the value found in the client audience configuration. Because this is a valid request, the OAuth token service sends a valid access token in the response. In this case, the audience claim and the scope have the same value of `http://www.example.com`.

Obtaining an Access Token by Using a Self-Signed Client Assertion

The client application uses a self-signed client assertion as part of the request to obtain the access token.

Instead of sending the client credentials, send the client assertion as part of the request for greater security. In Oracle Cloud, all OAuth clients are confidential by default and so their credentials (`client_id` and `password`) are never exposed directly. A client assertion is generated before requesting an access token. See Step-by-Step Workflow of the Client Credentials Grant to identify the claims that need to be part of the client assertion.

In the client credentials workflow, you obtain an access token by using a client assertion.

Parameters used in the access token request:

- **X-USER-IDENTITY-DOMAIN-NAME**: The name of the identity domain.
- **Content-Type:** The type of content that's sent in the request. It is a URL-encoded application.

- **Request:** The type of request that's sent. In the example that follows, a `POST` request is used to obtain an access token. This is followed by the authorization server URL, which provides tokens.
- **grant_type:** The grant type used to obtain the token. In the example that follows, the grant type is client credentials. The value of `client_credentials` is given for this grant type.
- **scope:** The limit of a particular scope for an access token.
- **client_assertion_type:** This specifies the type of client assertion that's passed. In Oracle Cloud, it's `jwt_bearer`.
- **client_assertion:** The value of the client token obtained.

The client credentials are available in the form of a self-signed JSON web token (JWT) client assertion. This is sent to obtain an access token.

To obtain an access token by using a client assertion, use the following `cURL` command:

```
curl -i -H 'X-USER-IDENTITY-DOMAIN-NAME: OAuthTestTenant150'
-H 'Content-Type: application/x-www-form-urlencoded;charset=UTF-8'
--request POST https://<idm-domain>.identity.<data-center>.oraclecloud.com/oauth/
tokens
-d 'grant_type=client_credentials
&client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-
bearer
&client_assertion=eyJ4NXQiOiJyb2NFQ2NaVDlheG5FdWpQMVVPQVo3ZGNyTmMiLCJraWQiOiJJX0FNX0d
PT0
QiLCJ0eXAiOiJKV1QiLCJhbGciOiJSUzUxMiJ9.eyJzdWIiOiJhNWQ1MzllYy05MDZmLTQzZDYtOGQ3Ny1hOD
YzYjh
kMzdjZTQiLCJpc3MiOiJJX0FNX0dPT0QiLCJvcmFjbGUub2ljLnRva2VuLnR5cGUiOiJDTEllFTlRUT0tFTiIs
ImV4c
CI6NDU3OTI2NjA3NywicHJuIjoiYTVkNTM5ZWMtOTA2Zi00M2Q2LThkNzctYTg2M2I4ZDM3Y2U0IiwiaWF0Ij
oxNDI
1NjY2MDc3LCJvcmFjbGUub2ljLnRva2VuLnVzZXJfZG4iOiJ1aWQ9YTVkNTM5ZWMtOTA2Zi00M2Q2LThkNzct
YTg2M
2I4ZDM3Y2U0LCBjbj10ZXN0ZXIgdGVzdGVyLCBvdT10ZXN0LCBvPW9yYWNsZSwgc3Q9Y2FsaWZvcm5pYSwgYz
11cyJ
9.MHC9Cof6uaZGMbrKAbmdn36b-
nHkI6HWq7A9ygba3VA3hsHRM3_hqZY_qXM1A9H585SVhipmi0RR9TNTINWstj2h
H6Z9WbATX6qJynSbyv8K7vb35dK2-awaGON9oTi2aPdApFkTiaX9r0-lvZSwVMbwx6ZPSIHSuxFMWvrpL58
&scope=http://www.example.com'
```

The output of the `cURL` command is:

```
{
"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsIng1dCI6ImtoNlhyVE42V2p6dmhIOExrN
nNLaWVpUD
VodyIsImtpZCI6Ik9BdXRoVGVzdFRlbmFudDE1MC5jZXJ0In0.eyJzdWIiOiJhNWQ1MzllYy05MDZmLTQzZDY
tOGQ3Ny1h
ODYzYjhkMzdjZTQiLCJpc3MiOiJPQXV0aFRlc3RUZW5hbnQxNTAiLCJvcmFjbGUub2F1dGguc3ZjX3BfbiI6I
k9BdXRoVG
VzdFRlbmFudDE1MFNlcnZpY2VQcm9maWxlIiwiaWF0IjoxNDI1NjY2NTk4MDAwLCJvcmFjbGUub2F1dGgucHJ
uLmlkX3R5
cGUiOiJDbGllbnRJRCIsImV4cCI6MTQyNTY3MDE5ODAwMCwib3JhY2xlLm9hdXRoLnRrX2NvbnRleHQiOiJyZ
XNvdXJjZV
9hY2Nlc3NfdGsiLCJhdWQiOlsiaHR0cDovL3d3dy5leGFtcGxlLmNvbSJdLCJwcm4iOiJhNWQ1MzllYy05MDZ
```

```
mLTQzZDYt
OGQ3Ny1hODYzYjhkMzdjZTQiLCJqdGkiOiJmOTA3MTFlOS0xOTgxLTQ4YzItOGMwOS1kOTE1MzRjMGRiY2EiL
CJvcmFjbG
Uub2F1dGguY2xpZW50X29yaWdpbl9pZCI6ImE1ZDUzOWVjLTkwNmYtNDNkNi04ZDc3LWE4NjNiOGQzN2NlNCI
sIm9yYWNs
ZS5vYXV0aC5zY29wZSI6Imh0dHA6Ly93d3cuZXhhbXBsZS5jb20iLCJ1c2VyLnRlbmFudC5uYW1lIjoiT0F1d
GhUZXN0VG
VuYW50MTUwIiwib3JhY2xlLm9hdXRoLmlkX2RfaWQiOiIzMDE2NzQ1NTk1MzQ0NzA4MSJ9.I8u207Vmvt3qiP
I8tBeU2-t
liGgyiXHLzJJ1sY_jvv-8B_irYjkMBjyC12RzLb2-
S0fpcZwuycCEjI4TCfkvfe6qBWdyEJHBxF1ioaKUhOs7oXgNxYcmo
8ZkkcwjdAg9nR4hRZ9lFZcYZOuHcSXxP2qsbnodrTxa6DLihj7cbyTql0i0d3wZjPMq9MPU3OyM6mtPDqltNT
KVU56r-X9
w23-MDguQGLSLFbbxpv6XgEo3eS6j1sfXfZZ9kPLwW-rSITOuqnmdq90IUsSh8Y4_wAg-
IrClftA9iRZ6D7z46t5-koXSY
U_oh6FwHJA0nhU-wt1z4UnD3HB60xKhEX8F8A"
}
```

The JWT obtained can be decoded, and the claims in the access token can be viewed as follows:

Access token:

```
 {
 alg: "RS256",
 typ: "JWT",
 x5t: "kh6XrTN6WjzvhH8Lk6sKieiP5hw",
 kid: "OAuthTestTenant150.cert"
}.
{
 sub: "a5d539ec-906f-43d6-8d77-a863b8d37ce4",
 iss: "OAuthTestTenant150",
 oracle.oauth.svc_p_n: "OAuthTestTenant150ServiceProfile",
 iat: 1425666598000,
 oracle.oauth.prn.id_type: "ClientID",
 exp: 1425670198000,
 oracle.oauth.tk_context: "resource_access_tk",
 aud: [
  "http://www.example.com"
 ],
 prn: "a5d539ec-906f-43d6-8d77-a863b8d37ce4",
 jti: "f90711e9-1981-48c2-8c09-d91534c0dbca",
 oracle.oauth.client_origin_id: "a5d539ec-906f-43d6-8d77-a863b8d37ce4",
 oracle.oauth.scope: "http://www.example.com",
 user.tenant.name: "OAuthTestTenant150",
 oracle.oauth.id_d_id: "30167455953447081"
}.
[signature]
```

Audience and scope claims in the output:

The audience claim in an access token contains the API path of the resource. The `oracle.oauth.scope` claim contains the valid API path with the scope in the response. In the prior example, the incoming request has a scope of `http://www.example.com`. The client audience configuration has a value of `http://www.example.com::*`. The OAuth token service validates the incoming request scope with the value found in the client audience configuration. Because this is a valid request, the OAuth token service sends a valid access token in the response. In this case, the audience claim has a value of `http://www.example.com`, and the scope has a value of `http://www.example.com`.

User Assertion Workflow

In this workflow the user provides the user assertion to obtain an access token from the OAuth token service.

The user assertion grant describes a flow where the client application, together with its client identifier and password, sends the user assertion in exchange for an access token. This flow has different security properties than the other OAuth flows. The primary difference is that the user's credentials are never accessible to the application.

Key Characteristics of the User Assertion Workflow

The user assertion workflow:

- It's used with confidential clients.
- It uses the assertion of the Resource Owner.
- It isn't redirection-based; it takes a request only from the client application to the authorization server, and the user is not redirected between interfaces to authorize the request.

Workflow of the User Assertion Grant

1. **Obtain user assertion:** The user provides the assertion. The client application can use an already-generated user assertion or build a new assertion. The user assertion is a standard JSON web token (JWT), to be signed by a trusted client using its private key. Verify that the following claims are part of the user assertion:

   Header

   ```
   {
   "alg": "RS256",
   "typ": "JWT",
   "x5t": "<X5t of the certificate>"
   }
   ```

   Body

   ```
   {
   "exp": <Expiry Time in seconds>,
   "sub": "<username like john.doe@example.com>",
   "aud": [   "oauth.idm.oracle.com" ],
   "iss": "<clientid>",
   "oracle.oauth.sub.id_type": "LDAP_UID",
   "prn": "<username like john.doe@example.com>",
   "jti": "<Globally Unique Id representing the token>",
   "iat": <Issued at Time in seconds>,
   "user.tenant.name": "<tenantname>",
   "oracle.oauth.prn.id_type": "LDAP_UID"
   }
   ```

2. **Request an access token:** The user assertion is exchanged for an access token. The client application makes a request to the authorization server, including the user's assertion and either the client's credentials or the client assertion. The client application can use an already generated client assertion or build a new assertion.

Note:

In the regular flow the access token's expiry claim is obtained from the configuration and the expiry time of the access token is by default 1 hour. However, in case of using the self-signed user assertion and client credentials flow, the expiry time of the access token can be modified to a value up to 90 days. The OAuth Server looks for the `exp` claim in the user assertion to determine the expiry claim of the resulting access token.

Obtain an access token by using different scenarios in the user assertion workflow:

- Obtaining an Access Token by Using a Self-Signed User Assertion and the Client Credentials
- Obtaining an Access Token by Using a Self-Signed User Assertion and a Client Assertion

3. **Receive an access token from the authorization server:** The authorization server authenticates the client based on the client identifier and secret, determines whether it is authorized for making this request, and verifies the user's assertion and other parameters that are supplied. If everything is verified successfully, then the authorization server returns an access token in response. This is described in Successful Authorization.

   If the authorization request fails for any reason, then the authorization server returns a response containing information about the error. This is described in Authorization Error.

4. **Use the access token to make a service request:** The OAuth client makes a REST API call to the resource server using the access token to access the protected resource.

5. **Send a response:** The OAuth resource server sends a response to the client application that made the request.

6. **Grant access to the resource:** The enduser or service gets access to the protected resource.

Using REST API Calls for the User Assertion Grant

In the user assertion flow, the user provides the user assertion to obtain an access token from the OAuth Service. In addition to the user assertion, the client provides an Authorization header, a self-signed client assertion, or a third-party generated client assertion in the access token request.

The table displays the different options to obtain an access token.

Task	Description	Additional Information
Obtain an access token by using a self-signed user assertion and the client credentials.	The OAuth client can request an access token by providing a self-signed user assertion and the client credentials.	Obtaining an Access Token by Using a Self-Signed User Assertion and the Client Credentials

Task	Description	Additional Information
Obtain an access token by using a self-signed user assertion and a client assertion.	The OAuth client can request an access token by providing a self-signed user assertion and a client assertion.	Obtaining an Access Token by Using a Self-Signed User Assertion and a Client Assertion

Obtaining an Access Token by Using a Self-Signed User Assertion and the Client Credentials

The OAuth client can request an access token by providing the user assertion and client credentials.

This workflow describes an access token request that uses the self-signed user assertion and a basic client authorization header. This is a more secure workflow than when the resource owner's credentials (user name and password) are exposed.

The user assertion grant workflow allows you to obtain an access token by using a user assertion and the client credentials that are supplied in the form of a basic authorization header. See User Assertion Workflow to identify the claims that need to be part of the user assertion.

The client application makes a request to the authorization server that includes the HTTP basic authorization header. The basic authorization header is `base64encoded(client_id:client_password)`.

Parameters used in the access token request:

- **X-USER-IDENTITY-DOMAIN-NAME**: The name of the identity domain.
- **Content-Type:** The type of content that's sent in the request. It's a URL-encoded application.
- **Authorization: Basic:** The basic authorization header. The client id and client secret of the client application are base64–encoded and sent in the header. For example, the authorization header has a value of `base64encoded(client_id:client_password)`.
- **Request:** The type of request that's sent. In the example that follows, a `POST` request is used to obtain an access token. This is followed by the authorization server URL which provides tokens.
- **grant_type:** The grant type used to obtain the token. In the example that follows, the grant type is user assertion grant. The value of `jwt-bearer` is given for this grant type.
- **scope:** The limit of a particular scope for an access token.
- **assertion:** The value of the user token obtained.

The client identifier and password are encoded and sent in the basic authorization header. This is sent along with the self-signed user assertion to obtain an access token.

To obtain an access token by using the user assertion and the client credentials, use the following `cURL` command:

```
curl -i -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
-H 'X-USER-IDENTITY-DOMAIN-NAME: OAuthTestTenant125'
```

```
 -H 'Authorization: Basic
YTVkNTM5ZWMtOTA2Zi00M2Q2LThkNzctYTg2M2I4ZDM3Y2U0OlZkT0dVSGZIVHpFUFFNcHhVbHkx'
--request POST https://<idm-domain>.identity.<data-center>.oraclecloud.com/oauth/
tokens
-d 'grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer
&client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-
bearer
&assertion=eyJraWQiOiJPQXV0aFRlc3RUZW5hbnQxNTAuaml0aGVuLWNsaWVudDEwLmNlcnQiLCJ0eXAiOi
JKV1QiLCJhbGciOiJSUzUxM
iJ9.eyJpc3MiOiJJX0FNX0dPT0QiLCJvcmFjbGUub2ljLnRva2VuLnR5cGUiOiJVU0VSVE9LRU4iLCJleHAiO
jQ1Nzk2MTcwMzQ2MTksInBybi
I6InRlbmFudEFkbWluVXNlciIsImlhdCI6MTQyNjAxNzAzNDYxOSwib3JhY2xlLm9pYy50b2tlbi51c2VyX2R
uIjoidWlkPXRlbmFudEFkbWlu
VXNlciwgY249dGVzdGVyIHRlc3Rlciwgb3U9dGVzdCwgbz1vcmFjbGUsIHN0PWNhbGlmb3JuaWEsIGM9dXMif
Q.elcNdSL6rl7RjmBPnS0UVN8
m7bJP7M7LUGLm6I4LXY3-mPSv1IP-Mn8r4GfMx7qCSgcCV16Lm3kBeXl9j-
YUYg1j2O8Z1AmxzQx_P3OvmRokUOv1SlCvW8Z560vrX3o1bhdfn
iFOJYef5pJrgTvri9WhNSTVjcYjJFRAxr7Ysfw
&scope=http://www.example.com'
```

The output of the cURL command is:

```
{
"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsIng1dCI6Ild3cmVwdTJkYXNhSXBHUi1Bb
FZwSGtVQjZKZyIsImtpZCI6I
k9BdXRoVGVzdFRlbmFudDEyNS5jZXJ0In0.eyJzdWIiOiJ0ZW5hbnRBZG1pblVzZXIiLCJvcmFjbGUub2F1dG
gudXNlcl9vcmlnaW5faWRfdHl
wZSI6IkxEQVBfVUlEIiwib3JhY2xlLm9hdXRoLnVzZXJfb3JpZ2luX2lkIjoidGVuYW50QWRtaW5Vc2VyIiwi
aXNzIjoiT0F1dGhUZXN0VGVuY
W50MTI1Iiwib3JhY2xlLm9hdXRoLnN2Y19wX24iOiJPQXV0aFRlc3RUZW5hbnQxMjVTZXJ2aWNlUHJvZmlsSZS
IsImlhdCI6MTQyNTQyNDMxODA
wMCwib3JhY2xlLm9hdXRoLnBybi5pZF90eXBlIjoiTERBUF9VSUQiLCJvcmFjbGUub2F1dGgudGtfY29udGV4
dCI6InJlc291cmNlX2FjY2Vzc
190ayIsImV4cCI6MTQyNTQyNzkxODAwMCwiYXVkIjpbImh0dHA6Ly93d3cuZXhhbXBsZS5jb20iXSwicHJuIj
oidGVuYW50QWRtaW5Vc2VyIiw
ianRpIjoiZDM4NWNjNzEtOGYxOC00NmE1LTlhZTQtNmFiNmYwODViYWRiIiwib3JhY2xlLm9hdXRoLmNsaWVu
dF9vcmlnaW5faWQiOiIzMDNhM
jQ5Mi1kNjRmLTRlMDQtYjc4Zi1iNDMzMDA0NzMxMmIiLCJvcmFjbGUub2F1dGguc2NvcGUiOiJodHRwOi8vd3
d3LmV4YW1wbGUuY29tIiwidXN
lci50ZW5hbnQubmFtZSI6Ik9BdXRoVGVzdFRlbmFudDEyNSIsIm9yYWNsZS5vYXV0aC5pZF9kX2lkIjoiMTM0
NjM2NzUxMzgzMDI1NjYifQ.DC
2OrybsETGdXJriaVQBhMxobxu6qGL-
r51X6wCUerep9WQgAsCjQrdtFPrFjqDRJEfhgZqPDH5GcCZqIJ9ckFk1WlDVBRsYudRWfgmVKPYwazU1
VUbwtfNkSDWnRJg_pE4ndMo_Ioi_D2LeP4PROgOCRHUUihtgyuKAKZk8f4VxIto4iVuUTxEy-0LU5v54Wnclt
K24LaUwVkqTBa2MgqrMJSdpJ2
91S2-
qeyY0cy9VcaxPyqZAbMRS5OhWyA_y45iqPPoUqAuRcZ9Mu9nhzmY_fewwf2nsJqoLTan4ruB0lLx7DuLs7ZfP
77UCULckkxrfcY8Ahmx_
HjO3LGpuzQ"
}
```

The JSON web token (JWT) obtained can be decoded, and the claims in the access token can be viewed as follows:

Access Token:

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

```
 x5t: "Wwrepu2dasaIpGR-AlVpHkUB6Jg",
 kid: "OAuthTestTenant125.cert"
}.
{
 sub: "tenantAdminUser",
 oracle.oauth.user_origin_id_type: "LDAP_UID",
 oracle.oauth.user_origin_id: "tenantAdminUser",
 iss: "OAuthTestTenant125",
 oracle.oauth.svc_p_n: "OAuthTestTenant125ServiceProfile",
 iat: 1425424318000,
 oracle.oauth.prn.id_type: "LDAP_UID",
 oracle.oauth.tk_context: "resource_access_tk",
 exp: 1425427918000,
 aud: [
  "http://www.example.com"
 ],
 prn: "tenantAdminUser",
 jti: "d385cc71-8f18-46a5-9ae4-6ab6f085badb",
 oracle.oauth.client_origin_id: "303a2492-d64f-4e04-b78f-b4330047312b",
 oracle.oauth.scope: "http://www.example.com",
 user.tenant.name: "OAuthTestTenant125",
 oracle.oauth.id_d_id: "13463675138302566"
}.
[signature]
```

Note:

In the regular flow the access token's expiry claim is obtained from the configuration and the expiry time of the access token is by default 1 hour. However for this use case the expiry time of the access token can be modified to a value up to 90 days. The OAuth Server looks for the `exp` claim in the user assertion to determine the expiry claim of the resulting access token. See User Assertion Workflow to determine the claims a self-signed user assertion should have.

Audience and scope claims in the output:

The audience claim in an access token contains the API path of the resource. The `oracle.oauth.scope` claim contains the valid API path with the scope in the response. In the prior example, the incoming request has a scope of `http://www.example.com`. The client audience configuration has a value of `http://www.example.com::*`. The OAuth token service validates the incoming request scope with the value found in the client audience configuration. Because this is a valid request, the OAuth token service sends a valid access token in the response. In this case, the audience claim has a value of `http://www.example.com`, and the scope has a value of `http://www.example.com`.

Obtaining an Access Token by Using a Self-Signed User Assertion and a Client Assertion

The OAuth client can request an access token by providing the user assertion and the client assertion.

This workflow has an access token request that uses a user assertion and a JSON web token (JWT) client assertion that is generated by a third party. This is a more secure workflow than when the resource owner's credentials (user name and password) are exposed.

The user assertion grant workflow allows you to obtain an access token by using a user assertion and a client assertion. See User Assertion Workflow to identify the claims that need to be part of the user assertion. See Step-by-Step Workflow of the Client Credentials Grant to identify the claims that need to be part of the client assertion.

Parameters used in the access token request:

- **X-USER-IDENTITY-DOMAIN-NAME**: The name of the identity domain.
- **Content-Type:** The type of content that's sent in the request. It is a URL-encoded application.
- **Request:** The type of request that's sent. In the example that follows, a `POST` request is used to obtain an access token. This is followed by the URL of the authorization server, which provides tokens.
- **grant_type:** The grant type used to obtain the token. In the example that follows, the grant type is a user assertion. The value of `jwt-bearer` is given for this grant type.
- **scope:** The limit of a particular scope for an access token.
- **client_assertion_type:** The type of client assertion that's passed. In Oracle Cloud, it's `jwt_bearer`.
- **client_assertion:** The value of the client token obtained.
- **assertion:** The value of the user token obtained.

The client credentials are available in the form of a third-party generated client assertion. This is sent along with a self-signed user assertion to obtain an access token.

To obtain an access token by using a self-signed user assertion and a client assertion, use the following `cURL` command:

```
curl -i -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
-H 'X-USER-IDENTITY-DOMAIN-NAME: OAuthTestTenant125'
--request POST https://<idm-domain>.identity.<data-center>.oraclecloud.com/oauth/
tokens
-d 'grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer
&client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-
bearer
&assertion=eyJraWQiOiJPQXV0aFRlc3RUZW5hbnQxNTAuaml0aGVuLWNsaWVudDEwLmNlcnQiLCJ0eXAiOi
JKV1QiLCJhbGciOi
JSUzUxMiJ9.eyJpc3MiOiJJX0FNX0dPT0QiLCJvcmFjbGUub2ljLnRva2VuLnR5cGUiOiJVU0VSVE9LRU4iLC
JleHAiOjQ1Nzk2MTcw
MzQ2MTksInBybiI6InRlbmFudEFkbWluVXNlciIsImlhdCI6MTQyNjAxNzAzNDYxOSwib3JhY2xlLm9pYy50b
2tlbi51c2VyX2RuIjo
idWlkPXRlbmFudEFkbWluVXNlciwgY249dGVzdGVyIHRlc3Rlciwgb3U9dGVzdCwgbz1vcmFjbGUsIHN0PWNh
bGlmb3JuaWEsIGM9dX
MifQ.elcNdSL6rl7RjmBPnS0UVN8m7bJP7M7LUGLm6I4LXY3-mPSv1IP-
Mn8r4GfMx7qCSgcCV16Lm3kBeXl9j-YUYg1j2O8Z1AmxzQ
x_P3OvmRokUOv1SlCvW8Z560vrX3o1bhdfniFOJYef5pJrgTvri9WhNSTVjcYjJFRAxr7Ysfw
&client_assertion=eyJ4NXQiOiJyb2NFQ2NaVDlheG5FdWpQMVVPQVo3ZGNyTmMiLCJraWQiOiJJX0FNX0d
PT0QiLCJ0eXAiOiJ
KV1QiLCJhbGciOiJSUzUxMiJ9.eyJzdWIiOiJhNWQ1MzllYy05MDZmLTQzZDYtOGQ3Ny1hODYzYjhkMzdjZTQ
iLCJpc3MiOiJJX0FNX
0dPT0QiLCJvcmFjbGUub2ljLnRva2VuLnR5cGUiOiJDTElFTlRUT0tFTiIsImV4cCI6NDU3OTUzOTA4MiwicH
JuIjoiYTVkNTM5ZWMt
OTA2Zi00M2Q2LThkNzctYTg2M2I4ZDM3Y2U0IiwiaWF0IjoxNDI1OTM5MDgyLCJvcmFjbGUub2ljLnRva2VuL
```

```
nVzZXJfZG4iOiJ1aWQ
9YTVkNTM5ZWMtOTA2Zi00M2Q2LThkNzctYTg2M2I4ZDM3Y2U0LCBjbj10ZXN0ZXIgdGVzdGVyLCBvdT10ZXN0
LCBvPW9yYWNsZSwgc3
Q9Y2FsaWZvcm5pYSwgYz11cyJ9.VnFBDNzxyL8pPAjUe2ogCYeqRFIWk3_JVTBREiJnOdY79tSEf78rYDefM2
znABSBW_EVow2fIglS
F_aNrvSslL9Ne4eammR9EELNDk5MvLlJOZQ5mt1ZODh2L8fYbt1nlujxYOE6qrrRNzkrase3wLv2Oe8lTsfgL
89Fzbm5p9A
&scope=http://www.example.com'
```

The output of the cURL command is:

```
{
"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsIng1dCI6ImtoNlhyVE42V2p6dmhIOExrN
nNLaWVpUDVodyIsImtp
ZCI6Ik9BdXRoVGVzdFRlbmFudDE1MC5jZXJ0In0.eyJzdWIiOiJ0ZW5hbnRBZG1pblVzZXIiLCJvcmFjbGUub
2F1dGgudXNlcl9vcmln
aW5faWQiOiJ0ZW5hbnRBZG1pblVzZXIiLCJpc3MiOiJPQXV0aFRlc3RUZW5hbnQxNTAiLCJvcmFjbGUub2F1d
Gguc3ZjX3BfbiI6Ik9B
dXRoVGVzdFRlbmFudDE1MFNlcnZpY2VQcm9maWxlIiwiaWF0IjoxNDI2MDIxNjY1MDAwLCJleHAiOjE0MjYwM
jUyNjUwMDAsIm9yYWNs
ZS5vYXV0aC50a19jb250ZXh0IjoicmVzb3VyY2VfYWNjZXNzX3RrIiwiYXVkIjpbImh0dHA6Ly93d3cuZXhhb
XBsZS5jb20iXSwicHJu
IjoidGVuYW50QWRtaW5Vc2VyIiwianRpIjoiZjMzMDQwNmUtZmYzNy00MTUwLTg0N2EtY2Q4MzdjYzM3MDI1I
iwib3JhY2xlLm9hdXRo
LmNsaWVudF9vcmlnaW5faWQiOiJhNWQ1MzllYy05MDZmLTQzZDYtOGQ3Ny1hODYzYjhkMzdjZTQiLCJvcmFjb
GUub2F1dGguc2NvcGUi
OiJodHRwOi8vd3d3LmV4YW1wbGUuY29tIiwidXNlci50ZW5hbnQubmFtZSI6Ik9BdXRoVGVzdFRlbmFudDE1M
CIsIm9yYWNsZS5vYXV0
aC5pZF9kX2lkIjoiMzAxNjc0NTU5NTM0NDcwODEifQ.ZMsXIfjE3PuE_jA-jJXaSjQtXqQZUQ-
nINQ1SW9T9VK8Yhx9ARptk4oYhZ6cQ
p_Wgq9Lw_hxEiOnlJY9blJBPO3f3r_SHUvNhKwPyHsQ9WyqAgOzJkjeUMrD2Z90N3mdRJqFKP7N1rphJbU6rr
D0Ko_nKenwBReX0-mPj
V_-qC4JxvdVsnLHiLFQW0MlFKUTmG2NafA-t14RO63hCxKa09gjIxgWCHnBdD--
YDvLsr3n6lZnKhIZg5IkKHAt2IR7wnIINOanAsvFI
RN36_pAVnGSfV7xAnrybVkyRPK13ltOfdUcvhKSvwqJaTtML8vVOfIO9qUUFjFfb_FkJFoIdA"
}
```

The JWT obtained can be decoded, and the claims in the access token can be viewed as follows:

Access token:

```
{
 alg: "RS256",
 typ: "JWT",
 x5t: "Wwrepu2dasaIpGR-AlVpHkUB6Jg",
 kid: "OAuthTestTenant125.cert"
}.
{
 sub: "tenantAdminUser",
 oracle.oauth.user_origin_id_type: "LDAP_UID",
 oracle.oauth.user_origin_id: "tenantAdminUser",
 iss: "OAuthTestTenant125",
 oracle.oauth.svc_p_n: "OAuthTestTenant125ServiceProfile",
 iat: 1425424318000,
 oracle.oauth.prn.id_type: "LDAP_UID",
 oracle.oauth.tk_context: "resource_access_tk",
 exp: 1425427918000,
 aud: [
```

```
  "http://www.example.com"
 ],
 prn: "tenantAdminUser",
 jti: "d385cc71-8f18-46a5-9ae4-6ab6f085badb",
 oracle.oauth.client_origin_id: "303a2492-d64f-4e04-b78f-b4330047312b",
 oracle.oauth.scope: "http://www.example.com",
 user.tenant.name: "OAuthTestTenant125",
 oracle.oauth.id_d_id: "13463675138302566"
}.
[signature]
```

Note:

In the regular flow the access token's expiry claim is obtained from the configuration and the expiry time of the access token is by default 1 hour. The OAuth Server looks for the `exp` claim in the user assertion to determine the expiry claim of the resulting access token. However, only if you are using a the self-signed user assertion and client credentials flow, the expiry time of the access token can be modified to a value up to 90 days.

Audience and scope claims in the output:

The audience claim in an access token contains the API path of the resource. The `oracle.oauth.scope` claim contains the valid API path with the scope in the response. In the prior example, the incoming request has a scope of `http://www.example.com`. The client audience configuration has a value of `http://www.example.com::*`. The OAuth token service validates the incoming request scope with the value found in the client audience configuration. Because this is a valid request, the OAuth token service sends a valid access token in the response. In this case, the audience claim has a value of `http://www.example.com` and the scope has a value of `http://www.example.com`.

Successful Authorization

During an authorization request, if the validations pass successfully, then the authorization server sends a response with an access token.

When the authorization server handles the request from the client, the following occur:

- Validation of the client assertion, also ensuring that the client is authorized to make the request. This is done by using the client certificate that was imported when the client was registered with the authorization server.
- Validation of the user, also ensuring that the user is authorized to make the request. Either the user's credentials or the user assertion is validated.
- Validation of the audience claim in the client profile in the OAuth service (information stored when the client is registered) against the scope in the incoming access token request.

If the validations pass successfully, the authorization server sends a response with an access token.

Access Token in the Response

In the body of the response, a JSON (or XML or other) object is included, representing the response, as shown in the following example:

```
{"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsIng1dCI6Ild3cmVwdTJkYXNhSXBHUi1Bb
FZwSGtVQjZKZyIsImtpZCI6I
k9BdXRoVGVzdFRlbmFudDEyNS5jZXJ0In0.eyJzdWIiOiJ0ZW5hbnRBZG1pblVzZXIiLCJvcmFjbGUub2F1dG
gudXNlcl9vcmlnaW5faWRfdH
lwZSI6IkxEQVBfVUlEIiwib3JhY2xlLm9hdXRoLnVzZXJfb3JpZ2luX2lkIjoidGVuYW50QWRtaW5Vc2VyIiw
iaXNzIjoiT0F1dGhUZXN0VGV
uYW50MTI1Iiwib3JhY2xlLm9hdXRoLnN2Y19wX24iOiJPQXV0aFRlc3RUZW5tbnQxMjVTZXJ2aWNlUHJvZmls
ZSIsImlhdCI6MTQyNTQyNzc1
NjAwMCwib3JhY2xlLm9hdXRoLnBybi5pZF90eXBlIjoiTERBUF9VSUQiLCJvcmFjbGUub2F1dGgudGtfY29ud
GV4dCI6InJlc291cmNlX2FjY
2Vzc190ayIsImV4cCI6MTQyNTQzMTM1NjAwMCwiYXVkIjpbImh0dHA6Ly93d3cuZXhhbXBsZS5jb20iXSwicH
JuIjoidGVuYW50QWRtaW5Vc2
VyIiwianRpIjoiZWM3ZWRhOWUtMTMyMS00MWI0LTg1MTYtMDQzNDRhMDY4MjE4Iiwib3JhY2xlLm9hdXRoLmN
saWVudF9vcmlnaW5faWQiOiI
zMDNhMjQ5Mi1kNjRmLTRlMDQtYjc4Zi1iNDMzMDA0NzMxMmIiLCJvcmFjbGUub2F1dGguc2NvcGUiOiJodHRw
Oi8vd3d3LmV4YW1wbGUuY29t
IiwidXNlci50ZW5hbnQubmFtZSI6Ik9BdXRoVGVzdFRlbmFudDEyNSIsIm9yYWNsZS5vYXV0aC5pZF9kX2lkI
joiMTM0NjM2NzUxMzgzMDI1N
jYifQ.maEhtSbOWqKZPj_Wb6vezEcusm8SHLLBrdqKRjh3aKyga4_vY8cYiug59gF7t9xeVY_wps2mOI6uaAm
8J0Fw0jZo2_NZpS5nXq-PiUN
CTHWiUQkb8GFv8Gd6qGc2M7Y3cBLG5w-
VVRCtRVBUtpTkaTg_VWzS_iygsSIBJjp6KsIqKRtE_DjGtp_E3AaBnfmB19r754NJl3_R8BQ1Wpbj
EL-8zFZzBa-4bTC0DQ58w4ssOOLT3aZWv9UhJ0Ok_pT-ZDcFxFLUGoMnTD-
pms9h4m58mGDtrRi5WBPC8n3RuOrmZBJjzOWAjCC0Tb-ZdR65q
7ewaxX_V6PccCHeIoFPmg"
}
```


The fields that are a part of the response are:

- **expires_in**: An optional and recommended parameter that specifies the lifetime of the access token in seconds. In the prior example, the access token is valid for 1 hour (or 3600 seconds).
- **token_type**: A mandatory parameter that specifies the type of token that's returned in the response. In the prior example, the `token_type` is `Bearer`

  .
- **access_token**: A mandatory parameter that has the actual access token as its value. This is the access token, in JSON web token (JWT) format, that the client application may store and use later. The token can be decoded to see the various claims in the access token response.

An example of a decoded JWT:

```
{
 alg: "RS256",
 typ: "JWT",
 x5t: "Wwrepu2dasaIpGR-AlVpHkUB6Jg",
 kid: "OAuthTestTenant125.cert"
}.
{
 sub: "tenantAdminUser",
 oracle.oauth.user_origin_id_type: "LDAP_UID",
 oracle.oauth.user_origin_id: "tenantAdminUser",
 iss: "OAuthTestTenant125",
 oracle.oauth.svc_p_n: "OAuthTestTenant125ServiceProfile",
 iat: 1425424318000,
 oracle.oauth.prn.id_type: "LDAP_UID",
```

```
 oracle.oauth.tk_context: "resource_access_tk",
 exp: 1425427918000,
 aud: [
  "http://www.example.com"
 ],
 prn: "tenantAdminUser",
 jti: "d385cc71-8f18-46a5-9ae4-6ab6f085badb",
 oracle.oauth.client_origin_id: "303a2492-d64f-4e04-b78f-b4330047312b",
 oracle.oauth.scope: "http://www.example.com",
 user.tenant.name: "OAuthTestTenant125",
 oracle.oauth.id_d_id: "13463675138302566"
}.
[signature]
```

Claims in the Access Token

An access token has a header, and standard and custom claims.

Claim Name	Type	Description	Sample
alg	Header	The algorithm used to sign the token.	RS256
typ	Header	The classification type of the token. The default value is JWT. This indicates that this is a JSON web token (JWT).	JWT
x5t	Header	The X.509 certificate thumbprint (x5t) header parameter provides a base64url-encoded SHA-256 thumbprint of the DER encoding of an X.509 certificate that can be used to match a certificate.	_hVX9pXq7pUxkk5ry-8vK8qb8L8
kid	Header	The key ID (kid) header parameter is a hint indicating which specific key owned by the signer should be used to validate the signature. This allows signers to signal a change of the key to recipients explicitly. Omitting this parameter is equivalent to setting it to an empty string. The interpretation of the contents of the kid parameter is unspecified.	oauth_psrtenantx3.cert
sub	Standard Claim	The subject (sub) claim identifies the principal that's the subject of the JWT.	MyAdmin@oracle1.com
prn	Standard Claim	The principal (prn) claim identifies the principal that is the subject of the JWT.	MyAdmin@oracle1.com
iss	Standard Claim	The issuer (iss) claim identifies the principal that supplied the JWT.	oauth_psrtenantx3
iat	Standard Claim	The issued at (iat) claim identifies the time at which the JWT was supplied.	1429128747000
exp	Standard Claim	The expiration time (exp) claim identifies the expiration time on or after which the JWT *must not* be accepted for processing.	1429128747000
aud	Standard Claim	The audience (aud) claim identifies the recipients for which the JWT is intended.	(a list of audiences)

jti	Standard Claim	The JWT ID (jti) claim provides a unique identifier for the JWT.	0565e04e-3823-404f-b950-e970ea17f41f
oracle.oauth.svc_p_n	Custom Claim	IDM OAuth service profile name.	oauth_psrtenantx3ServiceProfile
oracle.oauth.prn.id_type	Custom Claim	Principal ID type. For user assertion, the value is always LDAP_UID.	LDAP_UID
oracle.oauth.sub.id_type	Custom Claim	Subject ID type. For user assertion, the value is always LDAP_UID.	LDAP_UID
oracle.oauth.id_d_id	Custom Claim	IDM OAuth server domain ID.	20625897169639935
oracle.oauth.client_origin_id	Custom Claim	Subject ID for client used when user assertion is generated.	4457b326-fe88-4851-baad-b9488895e808
user.tenant.name	Custom Claim	User tenancy for the OAuth token generated by IDM OAuth server.	oauth_psrtenantx3

Authorization Error

If the access code request fails for any reason, or if one of the request parameters is invalid, then an error occurs.

The authorization server may return a response containing information about the error. This might be in JSON format (or XML or other) and may have the following format:

```
{
  "error":"invalid_request",
  "error_description":"Username parameter missing"
}
```

Possible Error Values

The `error` parameter can contain a number of values that describe the nature of the problem that occurred. These values and descriptions are as follows:

- **invalid_request**: The request is missing a parameter or value, a parameter is included multiple times, or a parameter has a malformed name.
- **invalid_client**: The authentication of the client fails. This can happen if authentication parameters are missing (for example, the client identifier and secret) or if the client tries to authenticate using an unsupported method.
- **invalid_grant**: The grant specified is invalid, expired, or revoked, or supplied to another client. For example, some services don't allow a new access token to be requested until the currently issued token expires.
- **unauthorized_client**: The client was authenticated by the authorization server, but has no authorization to use the requested grant.
- **unsupported_grant_type**: The grant that was requested isn't supported by the authorization server.

- **server_error**: The Authorization Server encountered an unexpected condition that prevented it from fulfilling the request.
- **temporarily_unavailable**: The authorization server is currently unable to handle the request because of a temporary overloading or maintenance of the server.
- **invalid_scope**: The scope specified in the request isn't valid, is unknown, or is malformed. If this occurs, then read the developer documentation associated with the service provider to see which scopes are available and which can be used.

Error Description

Only the `error` parameter is mandatory. But the optional `error_description` parameter may contain a short message explaining the error, for example, indicating a missing user name.