


Identity Cloud guide

ForgeRock® Identity Platform serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see <https://www.forgerock.com> .

This guide is for ForgeRock Identity Cloud evaluators, administrators, and architects. It provides examples of how to integrate your business application and APIs with Identity Cloud for Single Sign-On and API Security, with ForgeRock Identity Gateway.

Example installation for this guide

Unless otherwise stated, the examples in this guide assume the following installation:

- Identity Gateway installed on `http://ig.example.com:8080`, as described in [Download and start IG](#).
- Sample application installed on `http://app.example.com:8081`, as described in [Using the sample application](#).
- The ForgeRock Identity Cloud, with the default configuration, as described in the [ForgeRock Identity Cloud Docs](#).

When you are using the ForgeRock Identity Cloud, you need to know the value of the following properties:

- The root URL of your ForgeRock Identity Cloud. For example, `https://myTenant.forgeblocks.com`.

The URL of the Access Management component of the ForgeRock Identity Cloud is the root URL of your Identity Cloud followed by `/am`. For example, `https://myTenant.forgeblocks.com/am`.

- The realm where you work. The examples in this document use `alpha`.

Prefix each realm in the hierarchy with the `realms` keyword. For example, `/realms/root/realms/alpha`.

If you use a different configuration, substitute in the procedures accordingly.

Authenticate an IG agent to Identity Cloud

IG agents are automatically authenticated to Identity Cloud by a non-configurable authentication module. Authentication chains and modules are deprecated in Identity Cloud and replaced by journeys.

You can now authenticate IG agents to Identity Cloud with a journey. The procedure is currently optional, but will be required when authentication chains and modules are removed in a future release of Identity Cloud.

For more information, refer to Identity Cloud's [Journeys](#).

This section describes how to create a journey to authenticate an IG agent to Identity Cloud. The journey has the following requirements:

- It must be called Agent
- Its nodes must pass the agent credentials to the Agent Data Store Decision node.

When you define a journey in Identity Cloud, that same journey is used for all instances of IG, Java agent, and Web agent. Consider this point if you change the journey configuration.

1. Log in to the Identity Cloud admin UI as an administrator.
2. Click **Journeys > New Journey**.
3. Add a journey with the following information and click **Create journey**:
 - **Name:** Agent
 - **Identity Object:** The user or device to authenticate.
 - (Optional) **Description:** Authenticate an IG agent to Identity Cloud

The journey designer is displayed, with the **Start** entry point connected to the **Failure** exit point, and a **Success** node.

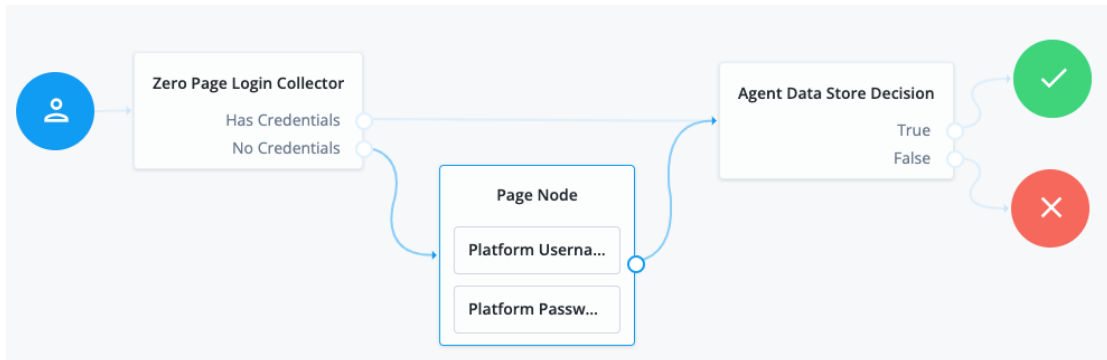
4. Using the **Q Filter nodes** bar, find and then drag the following nodes from the **Components** panel into the designer area:
 - [Zero Page Login Collector](#) node to check whether the agent credentials are provided in the incoming authentication request, and use their values in the following nodes.

This node is required for compatibility with Java agent and Web agent.
 - [Page](#) node to collect the agent credentials if they are not provided in the incoming authentication request, and use their values in the following nodes.
 - [Agent Data Store Decision](#) node to verify the agent credentials match the registered IG agent profile.

IMPORTANT

Many nodes can be configured in the panel on the right side of the page. Unless otherwise stated, do not configure the nodes, and use only the default values.

5. Drag the following nodes from the **Components** panel into the Page node:
 - Platform Username node to prompt the user to enter their username.
 - Platform Password node to prompt the user to enter their password.
6. Connect the nodes as follows and save the journey:



Register an IG agent in Identity Cloud

This procedure registers an agent that acts on behalf of IG.

1. Log in to the Identity Cloud admin UI as an administrator.
2. Click **Gateways & Agents** > **+ New Gateway/Agent** > **Identity Gateway** > **Next**, and add an agent profile:
 - **ID:** agent-name
 - **Password:** agent-password



IMPORTANT

Use secure passwords in a production environment. Consider using a password manager to generate secure passwords.

3. Click **Save Profile** > **Done**. The agent profile page is displayed.
4. To add a redirect URL for CDSSO, go to the agent profile page and add the URL.
5. To change the introspection scope, click **Native Consoles** > **Access Management**, and update the agent in the AM admin UI. By default, the agent can introspect OAuth 2.0 tokens issued to any client, in the realm and subrealm where it is created.

Set up a demo user in Identity Cloud

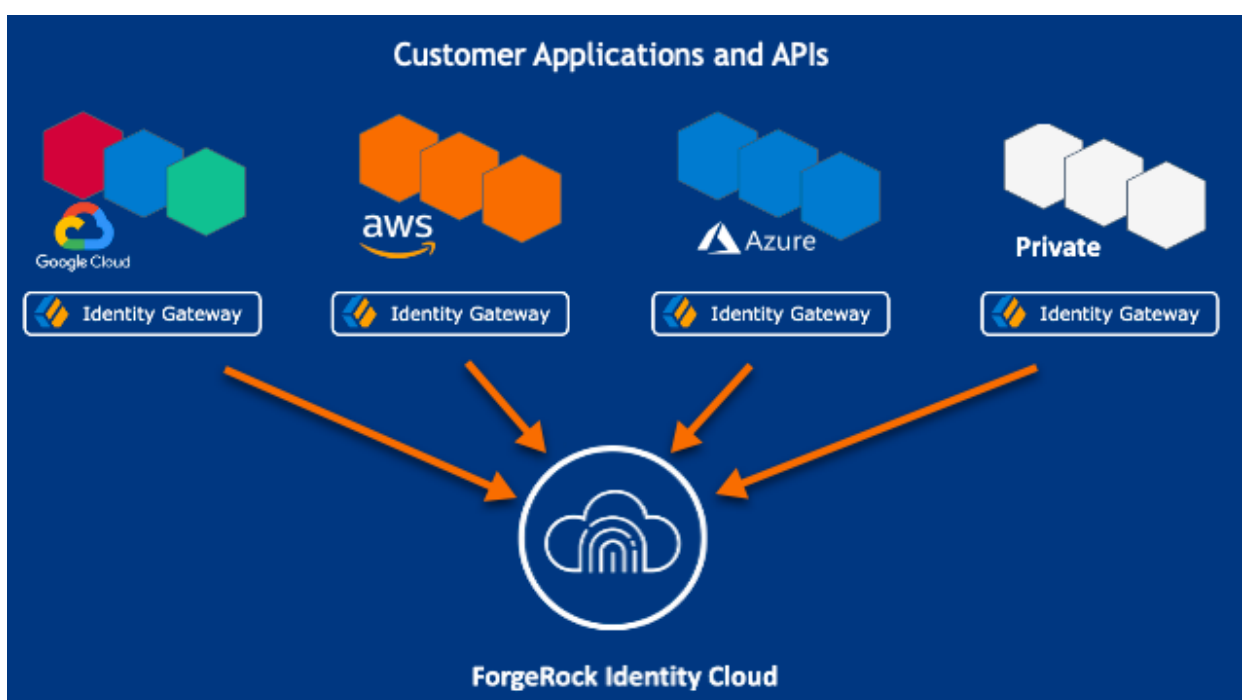
This procedure sets up a demo user in the alpha realm.

- a. Log in to the Identity Cloud admin UI as an administrator.
- b. Go to  **Identities > Manage >  Alpha realm - Users**, and add a user with the following values:
 - **Username:** demo
 - **First name:** demo
 - **Last name:** user
 - **Email Address:** demo@example.com
 - **Password:** Ch4ng3!t

About Identity Gateway and the ForgeRock Identity Cloud

ForgeRock Identity Cloud simplifies the consumption of ForgeRock as an Identity Platform. However, many organizations have business web applications and APIs deployed across multiple clouds, or on-premise.

Identity Gateway facilitates non-intrusive integration of your web applications and APIs with the Identity Cloud, for SSO and API Security. The following image illustrates how Identity Gateway bridges your business to the ForgeRock Identity Cloud:



For information about the ForgeRock Identity Cloud, refer to the [ForgeRock Identity Cloud Docs](#).

API security with OAuth 2.0 and the ForgeRock Identity Cloud




This example sets up OAuth 2.0, using the standard introspection endpoint, where ForgeRock Identity Cloud is the authorization server, and Identity Gateway is the resource server.

For more information about Identity Gateway as an OAuth 2.0 resource server, refer to [Validate access tokens through the introspection endpoint](#).

IMPORTANT

This procedure uses the *Resource Owner Password Credentials* grant type. According to information in the [The OAuth 2.0 Authorization Framework](#), minimize use of this grant type and utilize other grant types whenever possible.

Before you start, prepare Identity Cloud, IG, and the sample application as described in [Example installation for this guide](#).

1. Set up Identity Cloud:
 - a. Log in to the Identity Cloud admin UI as an administrator.
 - b. Go to  **Identities > Manage >  Alpha realm - Users**, and add a user with the following values:
 - **Username:** demo
 - **First name:** demo
 - **Last name:** user
 - **Email Address:** demo@example.com
 - **Password:** Ch4ng3!t
 - c. Make sure you are managing the alpha realm. If not, click the current realm at the top of the screen, and switch realm.
 - d. Add a web application:
 - i. In the Identity Cloud admin UI, click  **Applications > + Add Application > Web**, and add a web application with the following values:
 - **Client ID:** oauth2-client
 - **Client Secret:** password

- ii. On the application page, add the following general settings:
 - **Grant Types:** Resource Owner Password Credentials
 - **Scopes:** mail
- e. Register an IG agent with the following values, as described in [Register an IG agent in Identity Cloud](#):
 - **ID:** ig_agent
 - **Password:** password
- f. (Optional) Authenticate the agent to Identity Cloud as described in [Authenticate an IG agent to Identity Cloud](#).

IMPORTANT

IG agents are automatically authenticated to Identity Cloud by a deprecated authentication module in Identity Cloud. This step is currently optional, but will be required when authentication chains and modules are removed in a future release of Identity Cloud.

2. Set up Identity Gateway:

- a. Set an environment variable for the IG agent password, and then restart IG:

```
$ export AGENT_SECRET_ID='cGFzc3dvcmQ='
```

The password is retrieved by a SystemAndEnvSecretStore, and must be base64-encoded.

- b. Add the following route to Identity Gateway, replacing the value for the property `amInstanceUrl`:
 1. Linux
 2. Windows

```
$HOME/.openig/config/routes/oauth2rs-idx.json
```

```
%appdata%\OpenIG\config\routes\oauth2rs-idx.json
```

```
{
  "name": "oauth2rs-idx",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/oauth2rs-idx')}",
  "properties": {
    "amInstanceUrl":
      "https://myTenant.forgeblocks.com/am"
  }
}
```

```

    },
    "heap": [
      {
        "name": "SystemAndEnvSecretStore-1",
        "type": "SystemAndEnvSecretStore"
      },
      {
        "name": "AmService-1",
        "type": "AmService",
        "config": {
          "url": "&{amInstanceUrl}",
          "realm": "/alpha",
          "agent": {
            "username": "ig_agent",
            "passwordSecretId": "agent.secret.id"
          },
          "secretsProvider": "SystemAndEnvSecretStore-1"
        }
      }
    ],
    "handler": {
      "type": "Chain",
      "config": {
        "filters": [
          {
            "name": "OAuth2ResourceServerFilter-1",
            "type": "OAuth2ResourceServerFilter",
            "config": {
              "scopes": [
                "mail"
              ],
              "requireHttps": false,
              "realm": "OpenIG",
              "accessTokenResolver": {
                "name":
"TokenIntrospectionAccessTokenResolver-1",
                "type":
"TokenIntrospectionAccessTokenResolver",
                "config": {
                  "amService": "AmService-1",
                  "providerHandler": {
                    "type": "Chain",
                    "config": {
                      "filters": [

```

```

        "type":
"HttpBasicAuthenticationClientFilter",
        "config": {
            "username": "ig_agent",
            "passwordSecretId":
"agent.secret.id",
            "secretsProvider":
"SystemAndEnvSecretStore-1"
        }
    },
    ],
    "handler": "ForgeRockClientHandler"
}

}

}

}

}

}
],
"handler": {
    "type": "StaticResponseHandler",
    "config": {
        "status": 200,
        "headers": {
            "Content-Type": [ "text/html; charset=UTF-
8" ]
        },
        "entity": "<html><body><h2>Decoded
access_token: ${contexts.oauth2.accessToken.info}</h2>
</body></html>"
    }
}
}
}
}
}
}

```

Notice the following features of the route compared to `rs-introspect.json` in [Validate access tokens through the introspection endpoint](#), where a local Access Management instance is the authorization server:

- The `AmService URL` points to Access Management in the Identity Cloud.
- The `AmService realm` points to the realm where you have configured your web application and the IG agent.

3. Test the setup:

- a. In a terminal, export an environment variable for URL of Access Management in the Identity Cloud:

```
$ export amInstanceUrl='myAmInstanceUrl'
```

- b. Use a `curl` command similar to the following to retrieve an access token:

```
$ mytoken=$(curl -s \  
--user "oauth2-client:password" \  
--data \  
'grant_type=password&username=demo&password=Ch4ng3!t&sc  
ope=mail' \  
$amInstanceUrl/oauth2/realms/alpha/access_token | jq -r \  
".access_token")
```

- c. Validate the access token returned in the previous step:

```
$ curl -v http://ig.example.com:8080/oauth2rs-idc --  
header "Authorization: Bearer ${mytoken}"  
  
{  
  active = true,  
  scope = mail,  
  realm = /alpha,  
  client_id = oauth2-client,  
  ...  
}
```




Single sign-on with OpenID Connect and the ForgeRock Identity Cloud

This example sets up ForgeRock Identity Cloud as an OpenID Connect identity provider, and Identity Gateway as a relying party.

For more information about Identity Gateway and OpenID Connect, refer to [OpenID Connect](#).

Before you start, prepare Identity Cloud, IG, and the sample application as described in [Example installation for this guide](#).

1. Set up Identity Cloud:

- a. Log in to the Identity Cloud admin UI as an administrator.
- b. Go to  **Identities > Manage >  Alpha realm - Users**, and add a user with the following values:
 - **Username:** demo
 - **First name:** demo
 - **Last name:** user
 - **Email Address:** demo@example.com
 - **Password:** Ch4ng3!t
- c. Make sure you are managing the alpha realm. If not, click the current realm at the top of the screen, and switch realm.
- d. Add a web application:
 - i. In the Identity Cloud admin UI, click  **Applications > + Add Application > Web**, and add a web application with the following values:
 - **Client ID:** oidc-client
 - **Client Secret:** password
 - ii. In **General Settings** on the application page, add the following values:
 - **Sign-in URLs:**
http://ig.example.com:8080/home/id_token/callback
 - **Grant Types:** Authorization Code
 - **Scopes:** openid, profile, mail
 - iii. Click **Show advanced settings > Authentication**, and click **Implied Consent:**

The resource owner is not asked for consent during authorization flows.

2. Set up Identity Gateway:

- a. Set an environment variable for the oidc-client password, and then restart IG:

```
$ export CLIENT_SECRET_ID='cGFZc3dvcmQ='
```

- a. Add the following route to IG, to serve .css and other static resources for the sample application:
 1. Linux
 2. Windows

```
$HOME/.openig/config/routes/static-resources.json
```

```
%appdata%\OpenIG\config\routes\static-resources.json
```

```
{  
  "name" : "sampleapp-resources",  
  "baseURI" : "http://app.example.com:8081",  
  "condition": "${find(request.uri.path, '^/css')}",  
  "handler": "ReverseProxyHandler"  
}
```

b. Add the following route to Identity Gateway, replacing the value for the property `amInstanceUrl`:

1. Linux
2. Windows

```
$HOME/.openig/config/routes/oidc-oidc.json
```

```
%appdata%\OpenIG\config\routes\oidc-oidc.json
```

```
{  
  "name": "oidc-oidc",  
  "baseURI": "http://app.example.com:8081",  
  "condition": "${find(request.uri.path,  
  '^/home/id_token')}",  
  "properties": {  
    "amInstanceUrl":  
    "https://myTenant.forgeblocks.com/am"  
  },  
  "heap": [  
    {  
      "name": "SystemAndEnvSecretStore-1",  
      "type": "SystemAndEnvSecretStore"  
    }  
  ],  
  "handler": {  
    "type": "Chain",  
    "config": {  
      "filters": [  
        {  
          "name": "AuthorizationCodeOAuth2ClientFilter-
```

```

1",
    "type":
"AuthorizationCodeOAuth2ClientFilter",
    "config": {
        "clientEndpoint": "/home/id_token",
        "failureHandler": {
            "type": "StaticResponseHandler",
            "config": {
                "status": 500,
                "headers": {
                    "Content-Type": [
                        "text/plain"
                    ]
                },
                "entity": "Error in OAuth 2.0 setup."
            }
        },
        "registrations": [
            {
                "name": "oauth2-client",
                "type": "ClientRegistration",
                "config": {
                    "clientId": "oidc-client",
                    "clientSecretId": "client.secret.id",
                    "issuer": {
                        "name": "Issuer",
                        "type": "Issuer",
                        "config": {
                            "wellKnownEndpoint": "&
{amInstanceUrl}/oauth2/realms/alpha/.well-known/openid-
configuration"
                        }
                    },
                    "scopes": [
                        "openid",
                        "profile",
                        "mail"
                    ],
                    "secretsProvider":
"SystemAndEnvSecretStore-1",
                    "tokenEndpointAuthMethod":
"client_secret_basic"
                }
            }
        ],

```

```
        "requireHttps": false,  
        "cacheExpiration": "disabled"  
    }  
  },  
  ],  
  "handler": "ReverseProxyHandler"  
}  
}
```

Notice the following features of the route compared to `07-openid.json` in [Use AM As a Single OpenID Connect Provider](#), where Access Management is running locally:

- The `ClientRegistration wellKnownEndpoint` points to the Identity Cloud.

3. Test the setup:

- Go to http://ig.example.com:8080/home/id_token. The Identity Cloud login page is displayed.
- Log in to Identity Cloud as user `demo`, password `Ch4ng3!t`. The home page of the sample application is displayed.



Cross-domain single sign-on

For organizations relying on AM's session and policy services with SSO, consider cross-Domain Single Sign-On (CDSSO) as an alternative to SSO through OpenID Connect.

This example sets up ForgeRock Identity Cloud as an SSO authentication server for requests processed by Identity Gateway. For more information about Identity Gateway and CDSSO, refer to [Authenticate with CDSSO](#).

Before you start, prepare Identity Cloud, IG, and the sample application as described in [Example installation for this guide](#).


1. Set up Identity Cloud:

- Log in to the Identity Cloud admin UI as an administrator.
- Make sure you are managing the `alpha` realm. If not, click the current realm at the top of the screen, and switch realm.
- Go to  **Identities > Manage >  Alpha realm - Users**, and add a user with the following values:
 - **Username:** `demo`

- **First name:** demo
 - **Last name:** user
 - **Email Address:** demo@example.com
 - **Password:** Ch4ng3!t
- d. Register an IG agent with the following values, as described in [Register an IG agent in Identity Cloud](#):
- **ID:** ig_agent
 - **Password:** password
 - **Redirect URLs:** https://ig.ext.com:8443/home/cdsso/redirect
- e. (Optional) Authenticate the agent to Identity Cloud as described in [Authenticate an IG agent to Identity Cloud](#).

IMPORTANT

IG agents are automatically authenticated to Identity Cloud by a deprecated authentication module in Identity Cloud. This step is currently optional, but will be required when authentication chains and modules are removed in a future release of Identity Cloud.

- f. Add a Validation Service:
- i. In Identity Cloud, select  **Native Consoles > Access Management**. The AM admin UI is displayed.
 - ii. Select **Services**, and add a validation service with the following **Valid goto URL Resources**:
 - https://ig.ext.com:8443/*
 - https://ig.ext.com:8443/*?*
2. Set up Identity Gateway:
- a. Set up IG for HTTPS, as described in [Configure IG for HTTPS \(server-side\)](#).
 - b. Add the following session configuration to `admin.json`, to ensure that the browser passes the session cookie in the form-POST to the redirect endpoint (step 6 of [Information flow during CDSSO](#)):

```
{
  "connectors": [...],
  "session": {
    "cookie": {
      "sameSite": "none",
      "secure": true
    }
  },
}
```

```
"heap" : [...]  
}
```

This step is required for the following reasons:

- When `sameSite` is `strict` or `lax`, the browser does not send the session cookie, which contains the nonce used in validation. If IG doesn't find the nonce, it assumes that the authentication failed.
- When `secure` is `false`, the browser is likely to reject the session cookie.

For more information, refer to [admin.json](#).

c. Set an environment variable for the IG agent password, and then restart IG:

```
$ export AGENT_SECRET_ID='cGFzc3dvcmQ='
```

The password is retrieved by a `SystemAndEnvSecretStore`, and must be base64-encoded.

d. Add the following route to IG, to serve `.css` and other static resources for the sample application:

1. Linux
2. Windows

```
$HOME/.openig/config/routes/static-resources.json
```

```
%appdata%\OpenIG\config\routes\static-resources.json
```

```
{  
  "name" : "sampleapp-resources",  
  "baseURI" : "http://app.example.com:8081",  
  "condition": "${find(request.uri.path, '^/css')}",  
  "handler": "ReverseProxyHandler"  
}
```

e. Add the following route to Identity Gateway, and correct the value for the property `amInstanceUrl`:

1. Linux
2. Windows

```
$HOME/.openig/config/routes/cdsso-idc.json
```

```
%appdata%\OpenIG\config\routes\cdsso-idx.json
```

```
{
  "name": "cdsso-idx",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path,
'^/home/cdsso')}",
  "properties": {
    "amInstanceUrl":
"https://myTenant.forgeblocks.com/am"
  },
  "heap": [
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "url": "&{amInstanceUrl}",
        "realm": "/alpha",
        "agent": {
          "username": "ig_agent",
          "passwordSecretId": "agent.secret.id"
        },
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "sessionCache": {
          "enabled": false
        }
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "CrossDomainSingleSignOnFilter-1",
          "type": "CrossDomainSingleSignOnFilter",
          "config": {
            "redirectEndpoint": "/home/cdsso/redirect",
            "authCookie": {
              "path": "/home",

```



```

        "name": "ig-token-cookie"
    },
    "amService": "AmService-1",
    "verificationSecretId": "verify",
    "secretsProvider": {
        "type": "JwkSetSecretStore",
        "config": {
            "jwkUrl": "&
{amInstanceUrl}/oauth2/realms/alpha/connect/jwk_uri"
        }
    }
},
],
"handler": "ReverseProxyHandler"
}
}
}

```

Notice the following features of the route compared to `cdsso.json` in CDSO for IG in standalone mode, where Access Management is running locally:

- The `AmService` URL points to Access Management in the Identity Cloud.
- The `AmService realm` points to the realm where you configure your IG agent.

WARNING

For the security of your deployment, always configure `verificationSecretId` in `CrossDomainSingleSignOnFilter`. When `verificationSecretId` is not configured, IG does not verify the signature of AM session tokens, increasing the risk of CDSO token tampering.

- f. Restart IG.
3. Test the setup:
 - a. Go to <https://ig.ext.com:8443/home/cdsso> .

If you see warnings that the site is not secure, respond to the warnings to access the site.

The Identity Cloud login page is displayed.

- b. Log in to Identity Cloud as user `demo`, password `Ch4ng3!t`.

Access Management calls `/home/cdsso/redirect`, and includes the CDSSO token. The `CrossDomainSingleSignOnFilter` passes the request to sample app.



Policy enforcement

The following procedure gives an example of how to request and enforce policy decisions from Identity Cloud.

Enforce a simple policy

Before you start, set up and test the example in [Cross-domain single sign-on](#).

1. Set up Identity Cloud:

- a. In the Identity Cloud admin UI, select  **Native Consoles > Access Management**. The AM admin UI is displayed.
- b. Select  **Authorization > Policy Sets > New Policy Set**, and add a policy set with the following values:
 - **Id** : PEP-CDSSO
 - **Resource Types** : URL
- c. In the new policy set, add a policy with the following values:
 - **Name** : CDSSO
 - **Resource Type** : URL
 - **Resource pattern** : `*://*:*/*`
 - **Resource value** : `http://app.example.com:8081/home/cdsso`

This policy protects the home page of the sample application.
- d. On the **Actions** tab, add an action to allow HTTP GET .
- e. On the **Subjects** tab, remove any default subject conditions, add a subject condition for all **Authenticated Users** .

2. Set up IG:

- a. Replace `cdsso-idc.json` with the following route, and correct the value for the property `amInstanceUrl`:
 1. Linux
 2. Windows

```
$HOME/.openig/config/routes/pep-cdsso-idc.json
```

```
%appdata%\OpenIG\config\routes\pep-cdsso-idc.json
```

```
{
  "name": "pep-cdsso-idc",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path,
'^/home/cdsso')}",
  "properties": {
    "amInstanceUrl":
"https://myTenant.forgeblocks.com/am"
  },
  "heap": [
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "url": "&{amInstanceUrl}",
        "realm": "/alpha",
        "agent": {
          "username": "ig_agent",
          "passwordSecretId": "agent.secret.id"
        },
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "sessionCache": {
          "enabled": false
        }
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "CrossDomainSingleSignOnFilter-1",
          "type": "CrossDomainSingleSignOnFilter",
          "config": {
```

```

        "redirectEndpoint": "/home/cdsso/redirect",
        "authCookie": {
            "path": "/home",
            "name": "ig-token-cookie"
        },
        "amService": "AmService-1",
        "verificationSecretId": "verify",
        "secretsProvider": {
            "type": "JwkSetSecretStore",
            "config": {
                "jwkUrl": "&
{amInstanceUrl}/oauth2/realms/alpha/connect/jwk_uri"
            }
        }
    },
    {
        "name": "PolicyEnforcementFilter-1",
        "type": "PolicyEnforcementFilter",
        "config": {
            "application": "PEP-CDSSO",
            "ssoTokenSubject":
"${contexts.cdsso.token}",
            "amService": "AmService-1"
        }
    }
],
    "handler": "ReverseProxyHandler"
}
}
}

```

Note the following feature of the route compared to `cdsso-idx.json` :

- The `CrossDomainSingleSignOnFilter` is followed by a `PolicyEnforcementFilter` to enforce the policy `PEP-CDSSO` .

WARNING

For the security of your deployment, always configure `verificationSecretId` in `CrossDomainSingleSignOnFilter`. When `verificationSecretId` is not configured, IG does not verify the signature of AM session tokens, increasing the risk of CDSSO token tampering.

3. Test the setup:

a. Go to <https://ig.ext.com:8443/home/cdsso>.

If you have warnings that the site is not secure respond to the warnings to access the site.

IG redirects you to Identity Cloud for authentication.

b. Log in to Identity Cloud as user `demo`, password `Ch4ng3!t`.

Identity Cloud redirects you back to the request URL, and IG requests a policy decision. Identity Cloud returns a policy decision that grants access to the sample application.

Step up authorization for a transaction

Before you start, set up and test the example in [pep.adoc#pep-cdsso](#).

1. In the Identity Cloud admin UI, select **<> Scripts > Auth Scripts > New Script > Journey Decision Node > Next**, and add a default Journey Decision Node Script script called `TxTestPassword`:

```
/*
 * - Data made available by nodes that have already
 *   executed are available in the sharedState variable.
 * - The script should set outcome to either "true" or
 *   "false".
 */

var givenPassword = nodeState.get("password").asString()

if (givenPassword.equals("7890")) {
    outcome = "true"
} else {
    outcome = "false"
}
```

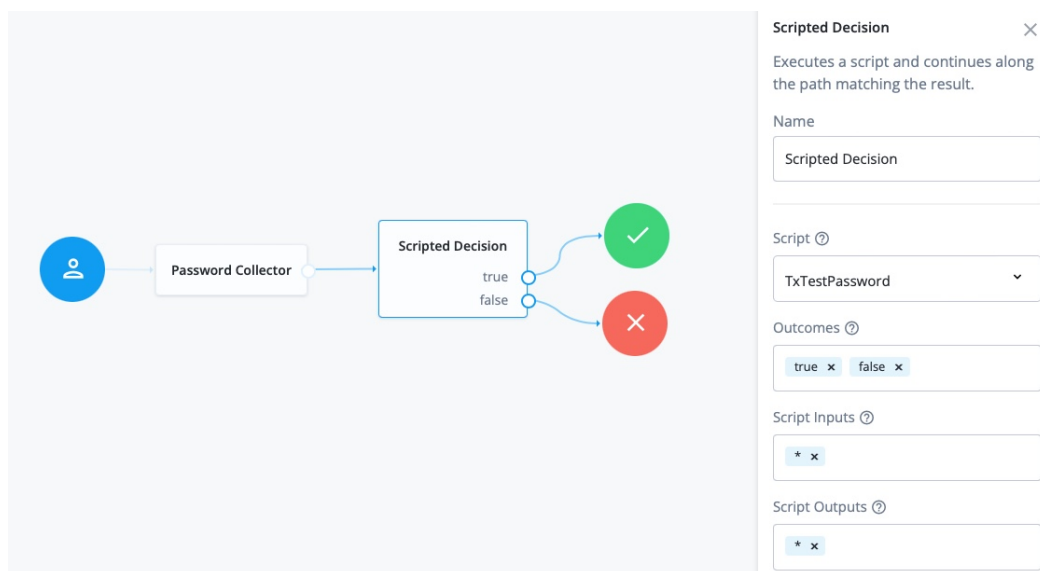
2. Configure a journey:

a. Click **Journeys** and add a journey with the following configuration:

- **Name:** `Tx01_Tree`
- **Identity Object:** `Alpha realm users`

The journey canvas is displayed.

- b. In **Nodes > Basic Authentication**, drag a **Password Collector** node onto the canvas.
- c. In **Nodes > Utilities**, drag a **Scripted decision** node onto the canvas.
- d. Configure the scripted decision node as follows:
 - **Script:** select TxTestPassword
 - **Outcomes:** enter true and false
- e. Connect the nodes as shown:



For information about configuring trees, refer to [ForgeRock Identity Cloud Docs](#)

3. Edit the authorization policy:

- a. In the Identity Cloud admin UI, select [Native Consoles > Access Management](#). The AM admin UI is displayed.
- b. Select [Authorization > Policy Sets > PEP-CDSSO](#), and add the following environment condition to the CDSSO policy:
 - All of
 - **Type:** Transaction
 - **Script name:** Authenticate to tree
 - **Strategy Specifier:** Tx01_Tree

4. Test the setup:

- a. In a browser, go to <https://ig.ext.com:8443/home/cdsso>.

If you have not previously authenticated to Identity Cloud, the CrossDomainSingleSignOnFilter redirects the request to Identity Cloud for authentication.

- b. Log in to Identity Cloud as user `demo`, password `Ch4ng3!t`.

c. Enter the password 7890 required by the script TxTestPassword .

Identity Cloud redirects you back to the request URL, and IG requests a policy decision. Identity Cloud returns a policy decision based on the authentication journey.

Pass runtime data downstream in a JWT

This example sets up Identity Cloud as an identity provider, to pass identity or other runtime information downstream, in a JWT signed with a PEM.

For more information about using runtime data, refer to [Passing data along the chain](#). To help with development, the sample application includes a `/jwt` endpoint to display the JWT, verify its signature, and decrypt it.

Before you start, prepare Identity Cloud, IG, and the sample application as described in [Example installation for this guide](#).

1. Set up secrets:

a. Locate a directory for secrets, and go to it:

```
$ cd /path/to/secrets
```

b. Create the following secret key and certificate pair as PEM files:

```
$ openssl req \  
-newkey rsa:2048 \  
-new \  
-nodes \  
-x509 \  
-days 3650 \  
-subj \  
"/CN=ig.example.com/OU=example/O=com/L=fr/ST=fr/C=fr" \  
-keyout ig.example.com-key.pem \  
-out ig.example.com-certificate.pem
```

Two PEM files are created, one for the secret key, and another for the associated certificate.

c. Map the key and certificate to the same secret ID in IG:

```
$ cat ig.example.com-key.pem ig.example.com-
```

```
certificate.pem > key.manager.secret.id.pem
```

d. Generate PEM files to sign and verify the JWT:

```
$ openssl req \  
-newkey rsa:2048 \  
-new \  
-nodes \  
-x509 \  
-days 3650 \  
-subj \  
"/CN=ig.example.com/OU=example/O=com/L=fr/ST=fr/C=fr" \  
-keyout id.key.for.signing.jwt.pem \  
-out id.key.for.verifying.jwt.pem
```

e. Make sure the following files have been added to your secrets directory:

- id.key.for.signing.jwt.pem
- id.key.for.verifying.jwt.pem
- key.manager.secret.id.pem
- ig.example.com-certificate.pem
- ig.example.com-key.pem

2. Set up Identity Cloud:

a. Log in to the Identity Cloud admin UI as an administrator.

b. Go to  **Identities > Manage >  Alpha realm - Users**, and add a user with the following values:

- **Username:** demo
- **First name:** demo
- **Last name:** user
- **Email Address:** demo@example.com
- **Password:** Ch4ng3!t

c. Register an IG agent with the following values, as described in [Register an IG agent in Identity Cloud](#):


- **ID:** ig_agent_jwt
- **Password:** password
- **Redirect URLs:** https://ig.example.com:8443/jwt/redirect

d. (Optional) Authenticate the agent to Identity Cloud as described in [Authenticate an IG agent to Identity Cloud](#).

IMPORTANT

IG agents are automatically authenticated to Identity Cloud by a deprecated authentication module in Identity Cloud. This step is currently optional, but will be required when authentication chains and modules are removed in a future release of Identity Cloud.

e. Add a Validation Service:

- i. In Identity Cloud, select  **Native Consoles > Access Management**. The AM admin UI is displayed.
- ii. Select **Services**, and add a validation service with the following **Valid goto URL Resources**:
 - `https://ig.example.com:8443/*`
 - `https://ig.example.com:8443/*?*`

3. Set up IG:

- a. Set up TLS by adding the following file to IG, replacing the value for the property `secretsDir`:

1. Linux
2. Windows

```
$HOME/.openig/config/admin.json
```

```
%appdata%\OpenIG\config\admin.json
```

```
{
  "mode": "DEVELOPMENT",
  "properties": {
    "secretsDir": "/path/to/secrets"
  },
  "connectors": [
    {
      "port": 8080
    },
    {
      "port": 8443,
      "tls": "ServerTlsOptions-1"
    }
  ],
  "session": {
    "cookie": {
      "sameSite": "none",

```

```

        "secure": true
      }
    },
    "heap": [
      {
        "name": "ServerTlsOptions-1",
        "type": "ServerTlsOptions",
        "config": {
          "keyManager": {
            "type": "SecretsKeyManager",
            "config": {
              "signingSecretId": "key.manager.secret.id",
              "secretsProvider": "ServerIdentityStore"
            }
          }
        }
      },
      {
        "name": "ServerIdentityStore",
        "type": "FileSystemSecretStore",
        "config": {
          "format": "PLAIN",
          "directory": "&{secretsDir}",
          "suffix": ".pem",
          "mappings": [{
            "secretId": "key.manager.secret.id",
            "format": {
              "type": "PemPropertyFormat"
            }
          }
        ]
      }
    ]
  }
}

```

- b. Set an environment variable for the IG agent password, and then restart IG:

```
$ export AGENT_SECRET_ID='cGFzc3dvcmQ='
```

The password is retrieved by a SystemAndEnvSecretStore, and must be base64-encoded.

- c. Add the following route to IG, to serve .css and other static resources for the sample application:

1. Linux

2. Windows

```
$HOME/.openig/config/routes/static-resources.json
```

```
%appdata%\OpenIG\config\routes\static-resources.json
```

```
{  
  "name" : "sampleapp-resources",  
  "baseURI" : "http://app.example.com:8081",  
  "condition": "${find(request.uri.path, '^/css')}",  
  "handler": "ReverseProxyHandler"  
}
```

d. Add the following route to IG, replacing the value for the properties `secretsDir` and `amInstanceUrl`:

1. Linux
2. Windows

```
$HOME/.openig/config/routes/jwt-idc.json
```

```
%appdata%\OpenIG\config\routes\jwt-idc.json
```

```
{  
  "name": "jwt-idc",  
  "condition": "${find(request.uri.path, '/jwt')}",  
  "baseURI": "http://app.example.com:8081",  
  "properties": {  
    "secretsDir": "/path/to/secrets",  
    "amInstanceUrl":  
    "https://myTenant.forgeblocks.com/am"  
  },  
  "heap": [  
    {  
      "name": "SystemAndEnvSecretStore-1",  
      "type": "SystemAndEnvSecretStore"  
    },  
    {  
      "name": "AmService-1",  
      "type": "AmService",  
      "config": {  
        "url": "&{amInstanceUrl}",  

```

```

    "realm": "/alpha",
    "agent": {
      "username": "ig_agent_jwt",
      "passwordSecretId": "agent.secret.id"
    },
    "secretsProvider": "SystemAndEnvSecretStore-1",
    "sessionCache": {
      "enabled": false
    }
  }
},
{
  "name": "pemPropertyFormat",
  "type": "PemPropertyFormat"
},
{
  "name": "FileSystemSecretStore-1",
  "type": "FileSystemSecretStore",
  "config": {
    "format": "PLAIN",
    "directory": "&{secretsDir}",
    "suffix": ".pem",
    "mappings": [{
      "secretId": "id.key.for.signing.jwt",
      "format": "pemPropertyFormat"
    }]
  }
}
],
"handler": {
  "type": "Chain",
  "config": {
    "filters": [
      {
        "name": "CrossDomainSingleSignOnFilter-1",
        "type": "CrossDomainSingleSignOnFilter",
        "config": {
          "redirectEndpoint": "/jwt/redirect",
          "authCookie": {
            "path": "/jwt",
            "name": "ig-token-cookie"
          },
        },
        "amService": "AmService-1",
        "verificationSecretId": "verify",
        "secretsProvider": {

```

```

        "type": "JwkSetSecretStore",
        "config": {
            "jwkUrl": "&
{amInstanceUrl}/oauth2/realms/alpha/connect/jwk_uri"
        }
    },
    {
        "name": "UserProfileFilter",
        "type": "UserProfileFilter",
        "config": {
            "username":
"${contexts.ssoToken.info.uid}",
            "userService": {
                "type": "UserProfileService",
                "config": {
                    "amService": "AmService-1"
                }
            }
        }
    },
    {
        "name": "JwtBuilderFilter-1",
        "type": "JwtBuilderFilter",
        "config": {
            "template": {
                "name":
"${contexts.userProfile.commonName}",
                "email":
"${contexts.userProfile.rawInfo.mail[0]}"
            },
            "secretsProvider": "FileSystemSecretStore-
1",
            "signature": {
                "secretId": "id.key.for.signing.jwt",
                "algorithm": "RS512"
            }
        }
    },
    {
        "name": "HeaderFilter-1",
        "type": "HeaderFilter",
        "config": {
            "messageType": "REQUEST",

```

```
        "add": {
            "x-openig-user":
["${contexts.jwtBuilder.value}"]
        }
    },
    ],
    "handler": "ReverseProxyHandler"
}
}
```

WARNING

For the security of your deployment, always configure `verificationSecretId` in `CrossDomainSingleSignOnFilter`. When `verificationSecretId` is not configured, IG does not verify the signature of AM session tokens, increasing the risk of CDSSO token tampering.

4. Test the setup:

- a. Go to <https://ig.example.com:8443/jwt>.

If you receive warnings that the site is not secure, respond to the warnings to access the site. The Identity Cloud login page is displayed.

- b. Log in to Identity Cloud as user `demo`, password `Ch4ng3!t`. The sample app displays the signed JWT along with its header and payload.
- c. In `USE PEM FILE`, enter the absolute path to `id.key.for.verifying.jwt.pem` to verify the JWT signature.