Platform Setup Guide

/ ForgeRock Identity Platform 7.1

Latest update: 7.1.0

ForgeRock AS.
201 Mission St., Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Abstract

Guide to setting up the ForgeRock Identity Platform™ locally.

Table of Contents

Overview

The ForgeRock Identity Platform is the only offering for access management, identity management, user-managed access, directory services, and an identity gateway, designed and built as a single, unified platform.

This guide lets you set up the platform components without using the ForgeRock Identity Cloud.

Quick Start

 Start Here Learn about the two sample deployments shown in this guide.	 Deployment One Set up the ForgeRock Identity Platform with separate identity stores between AM and IDM.
 Deployment Two Set up the ForgeRock Identity Platform with a DS identity store that is shared between AM and IDM.	 Platform UIs Learn about the three Platform UIs and deploy them with either Deployment One or Deployment Two.

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.

The ForgeRock Common REST API works across the platform to provide common ways to access web resources and collections of resources.

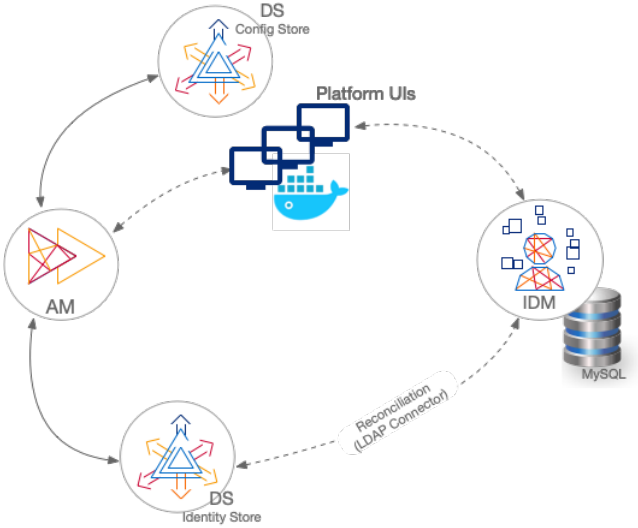
Chapter 1

Deployment Overview

This guide shows two sample deployments:

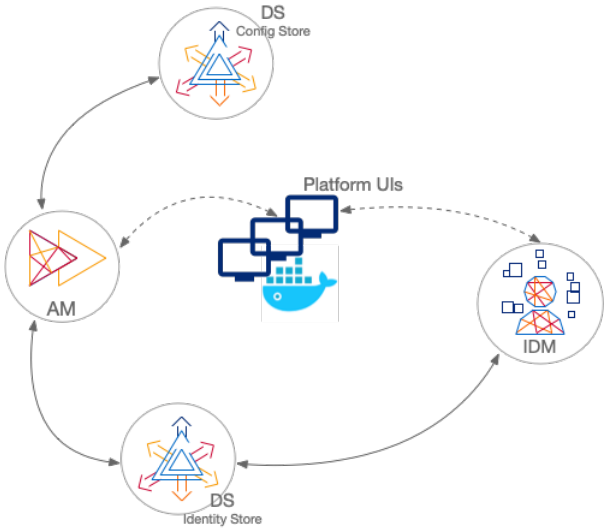
"Deployment One - Separate Identity Stores"

This deployment has an external DS instance configured as the AM configuration store, and a second external DS instance configured as the AM identity store. The IDM repository is an external JDBC database. The sample was tested with MySQL. The deployment uses an LDAP connector to synchronize the identities between IDM and AM:



"Deployment Two - Shared Identity Store"

This deployment has an external DS instance configured as the AM configuration store. The AM and IDM servers share an external DS instance as the identity store, and no synchronization configuration is required:

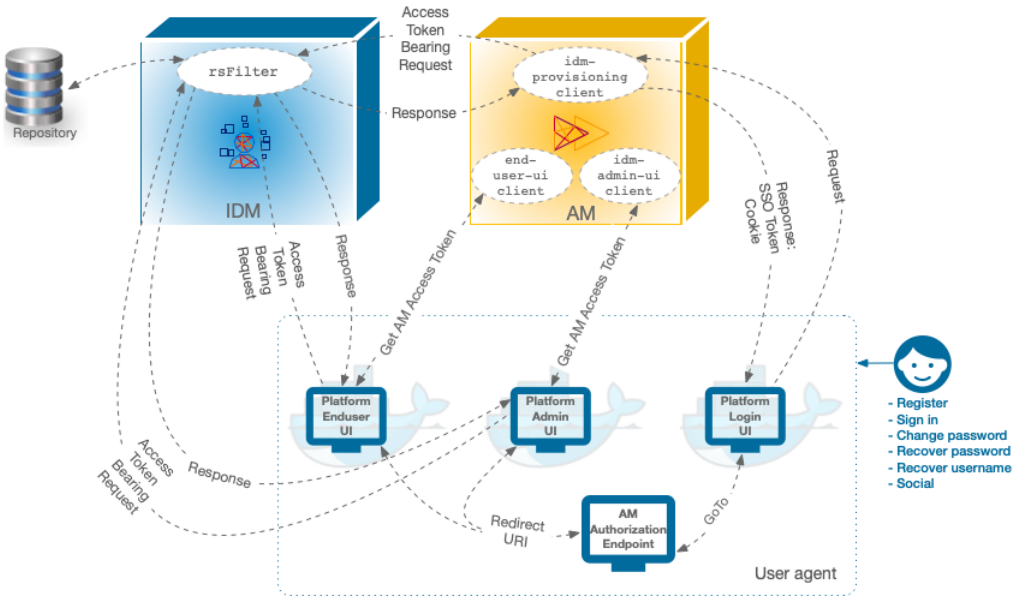


Important

In both deployments, the Platform UIs run in separate Docker containers. If you want to use the Platform UIs, get Docker before you start.

Component Interaction

A platform configuration relies on multiple components working together. The following image shows how the AM OAuth 2 clients interact with the IDM resource server filter (`rsFilter`) to grant access through the Platform UIs:



1. The Platform UIs send a request to the AM Authorization Endpoint.
2. If the end user is authenticated, the user agent is redirected back to the UI, according to the Redirection URI request parameter.
3. If the end user is not authenticated, the AM Authorization Endpoint redirects the user agent to the Platform Login UI.
4. After successful authentication, the Platform Login UI redirects the user agent back to the AM Authorization Endpoint, according to the GoTo request parameter.

Important

- Do *not* use the IDM stand-alone end-user UI if you are deploying AM and IDM as a platform. This UI is delivered with IDM under `ui/enduser`. It is *not* the same as the `platform-enduser` UI and will not work in a platform deployment.
- By default, the IDM Admin UI only supports users from the AM root realm. If you need to support users from other realms, adjust the `/oauth2/` references in the `/path/to/openidm/ui/admin/default/index.html` file:

```
commonSettings.authorizationEndpoint = calculatedAMUriLink.href + '/oauth2/authorize';

AppAuthHelper.init({
    clientId: commonSettings.clientId,
    authorizationEndpoint: commonSettings.authorizationEndpoint,
    tokenEndpoint: calculatedAMUriLink.href + '/oauth2/access_token',
    revocationEndpoint: calculatedAMUriLink.href + '/oauth2/token/revoke',
    endSessionEndpoint: calculatedAMUriLink.href + '/oauth2/connect/endSession',
```

For example, if your realm is named SUBREALM>, replace `/oauth2/` with `/oauth2/realms/root/realms/SUBREALM/`.

Server Settings

This guide assumes that all servers are deployed on their own hosts, with the following server settings. Adjust the settings to match your own deployment.

Important

To deploy the entire platform on a single computer, the recommended alternative is to use the ForgeOps Cloud Developer's Kit (CDK) on Minikube.

If you nevertheless choose to demonstrate these deployments on your computer, add aliases for the fully qualified domain names used for AM, IDM and platform UIs to your `/etc/hosts` file:

```
127.0.0.1    am.example.com
127.0.0.1    openidm.example.com
127.0.0.1    admin.example.com
127.0.0.1    enduser.example.com
127.0.0.1    login.example.com
```

Also, use a single DS server with all applicable DS setup profiles for your deployment. You can use `localhost` as the domain for the DS server for demonstration purposes. Adapt the instructions to configure AM and IDM accordingly.

- AM host: `am.example.com`
- AM port: `8081`
- External DS configuration store host: `config.example.com`
- External DS configuration store ports:

  `adminConnectorPort 4444`
  `ldapPort 1389`
  `ldapsPort 1636`

- External DS identity store host: `identities.example.com`

  These settings apply to both the separate and shared DS identity stores.

- External DS identity store ports:

  `adminConnectorPort 4444`
  `ldapPort 1389`
  `ldapsPort 1636`

- IDM host: `openidm.example.com`
- IDM ports: `HTTP 8080`, `HTTPS 8443`
- Platform Admin UI: `http://admin.example.com:8082`
- Platform Login UI: `http://login.example.com:8083`
- Platform End User UI: `http://enduser.example.com:8888`

Chapter 2

Deployment One - Separate Identity Stores

This deployment assumes that you are using the following data stores:

- An external DS instance as the AM configuration store.
- A separate external DS instance as the AM identity store.
- A MySQL repository as the IDM data store.

Note

The IDM End User UI is not supported in a platform deployment, as it does not support authentication through AM. You can use the Platform UIs with this deployment, or create your own login and end user UIs that support authentication through AM.

Set up Your Data Stores

Configuration Store

1. Set up a DS server as an external configuration data store, using the `am-config` setup profile.

   This example also adds the `am-cts` setup profile to the external configuration data store. In production, consider using separate DS servers for the AM CTS store. For more information about AM CTS, see the *AM Core Token Service Guide (CTS)*.

   For more information about DS setup profiles, see setup profiles in the *DS Installation Guide*.

   This command sets up the config store with the parameters listed in "Server Settings".

```
/path/to/opendj/setup \
--deploymentKeyPassword password \
--rootUserDN uid=admin \
--rootUserPassword str0ngAdm1nPa55word \
--monitorUserPassword str0ngMon1torPa55word \
--hostname config.example.com \
--adminConnectorPort 4444 \
--ldapPort 1389 \
--enableStartTls \
--ldapsPort 1636 \
--profile am-config \
--set am-config/amConfigAdminPassword:5up35tr0ng \
--profile am-cts \
--set am-cts/amCtsAdminPassword:5up35tr0ng \
--set am-cts/tokenExpirationPolicy:am-sessions-only \
--acceptLicense
```

Make a note of the generated deployment key. You will need it to export the server certificate later in this procedure. You can set the deployment key as a variable in this terminal window. For example:

```
export DEPLOYMENT_KEY=deployment-key
```

Note

For simplicity, this example uses a standalone directory server that does not replicate directory data. (No `--replicationPort` or `--bootstrapReplicationServer` options.)

In production deployments, replicate directory data for availability and resilience. For details, see the DS installation documentation.

2. Start the DS server:

```
/path/to/opendj/bin/start-ds
```

Identity Store

- Set up a DS server as an AM identity store, using the `am-identity-store` setup profile.

  For more information about this step, see setup profiles in the *DS Installation Guide*.

  This command sets up the identity store with the parameters listed in "Server Settings":

```
/path/to/opendj/setup \
--deploymentKeyPassword password \
--rootUserDN uid=admin \
--rootUserPassword str0ngAdm1nPa55word \
--monitorUserPassword str0ngMon1torPa55word \
--hostname identities.example.com \
--adminConnectorPort 4444 \
--ldapPort 1389 \
--enableStartTls \
--ldapsPort 1636 \
--profile am-identity-store \
--set am-identity-store/amIdentityStoreAdminPassword:5up35tr0ng \
--acceptLicense
```

Note

For simplicity, this example uses standalone directory servers that do not replicate directory data. (No `--replicationPort` or `--bootstrapReplicationServer` options.)

In production deployments, replicate directory data for availability and resilience. For details, see the DS installation documentation.

Make a note of the generated deployment key. You will need it to export the server certificate later in this procedure. You can set the deployment key as a variable in this terminal window. For example:

```
export DEPLOYMENT_KEY=deployment-key
```

Start the DS server:

```
/path/to/opendj/bin/start-ds
```

Set Up a Container

- Install a Java container to deploy AM.

  This guide assumes that you are using Apache Tomcat.

  For instructions on setting up Tomcat, see Preparing Apache Tomcat in the *AM Installation Guide*.

Secure Connections

Important

From DS 7 onwards, you *must* secure connections to DS servers.

1. Create a new directory that will house a dedicated truststore for AM:

```
mkdir -p /path/to/openam-security/
```

2. *On each DS server,* export the DS server certificate.

   You must run these commands in the same terminal window where you set the `DEPLOYMENT_KEY` variable.

   On `config.example.com`:

   ```
   /path/to/opendj/bin/dskeymgr export-ca-cert \
   --deploymentKey $DEPLOYMENT_KEY \
   --deploymentKeyPassword password \
   --outputFile config-ca-cert.pem
   ```

   On `identities.example.com`:

   ```
   /path/to/opendj/bin/dskeymgr export-ca-cert \
   --deploymentKey $DEPLOYMENT_KEY \
   --deploymentKeyPassword password \
   --outputFile identities-ca-cert.pem
   ```

3. Import each DS server certificate into the dedicated AM truststore. If you are not testing this example on a single host, you might need to copy each certificate file onto the AM host machine first:

   ```
   keytool \
   -importcert \
   -trustcacerts \
   -alias config-ca-cert \
   -file /path/to/config-ca-cert.pem \
   -keystore /path/to/openam-security/truststore \
   -storepass changeit \
   -storetype JKS
   Owner: CN=Deployment key, O=ForgeRock.com
   Issuer: CN=Deployment key, O=ForgeRock.com
   ...
   Trust this certificate? [no]:  yes
   Certificate was added to keystore

   keytool \
   -importcert \
   -trustcacerts \
   -alias identities-ca-cert \
   -file /path/to/identities-ca-cert.pem \
   -keystore /path/to/openam-security/truststore \
   -storepass changeit \
   -storetype JKS
   Owner: CN=Deployment key, O=ForgeRock.com
   Issuer: CN=Deployment key, O=ForgeRock.com
   ...
   Trust this certificate? [no]:  yes
   Certificate was added to keystore
   ```

4. (Optional) List the certificates in the new truststore and verify that the two certificates you added are there:

```
keytool \
-list \
-keystore /path/to/openam-security/truststore \
-storepass changeit
```

5. Point Apache Tomcat to the path of the new truststore so that AM can access it.

   Append the truststore settings to the `CATALINA_OPTS` variable in the `$CATALINA_BASE/bin/setenv.sh` file.

   For example:

   ```
   CATALINA_OPTS="-Djavax.net.ssl.trustStore=/path/to/openam-security/truststore\
   -Djavax.net.ssl.trustStorePassword=changeit\
   -Djavax.net.ssl.trustStoreType=jks"
   ```

   Refer to your specific container's documentation for information on configuring truststores.

6. (Optional) At this stage, it is worth checking that you can use the dedicated AM connect to each DS server:

   On `config.example.com`:

   ```
   /path/to/opendj/bin/ldapsearch \
   --hostname config.example.com \
   --port 1636 \
   --useSsl \
   --useJavaTrustStore /path/to/openam-security/truststore \
   --trustStorePassword changeit \
   --bindDn uid=am-config,ou=admins,ou=am-config \
   --bindPassword 5up35tr0ng \
   --baseDn ou=am-config \
   "(&)" \
   1.1
   dn: ou=am-config

   dn: ou=admins,ou=am-config

   dn: uid=am-config,ou=admins,ou=am-config
   ```

   On `identities.example.com`:

```
/path/to/opendj/bin/ldapsearch \
--hostname identities.example.com \
--port 1636 \
--useSsl \
--useJavaTrustStore /path/to/openam-security/truststore \
--trustStorePassword changeit \
--bindDn uid=am-identity-bind-account,ou=admins,ou=identities \
--bindPassword 5up35tr0ng \
--baseDn ou=identities \
"(&)" \
1.1
dn: ou=identities

dn: ou=people,ou=identities

dn: ou=groups,ou=identities

dn: ou=admins,ou=identities

dn: uid=am-identity-bind-account,ou=admins,ou=identities
```

Configure AM

When your external data stores are configured, follow these procedures to configure AM with the ForgeRock Identity Platform:

- "Install AM"
- "Configure OAuth Clients"
- "Configure an OAuth 2.0 Provider Service"
- "Configure an IDM Provisioning Service"
- "Enable CORS Support"
- "Configure Authentication Trees"
- "Map Authentication Trees"
- "Enable the Password Update Tree (optional)"

Install AM

1. Follow the instructions in the *AM Installation Guide* to download AM. Make sure you download the `.zip` file, not just the `.war` file.
2. Follow the instructions in the *AM Installation Guide* to prepare your environment, and prepare a web application container.

Use Apache Tomcat as the application container, listening on port `8081`. This non-default port requires that you update Tomcat's `conf/server.xml` file. Instead of the default line, `<Connector port="8080" protocol="HTTP/1.1"`, use:

```
    <!-- Use 8081 to avoid a clash with IDM on 8080. -->
    <Connector port="8081" protocol="HTTP/1.1"
```

3. Copy the AM `.war` file to deploy in Apache Tomcat as `am.war`:

```
cp AM-7.1.0.war /path/to/tomcat/webapps/am.war
```

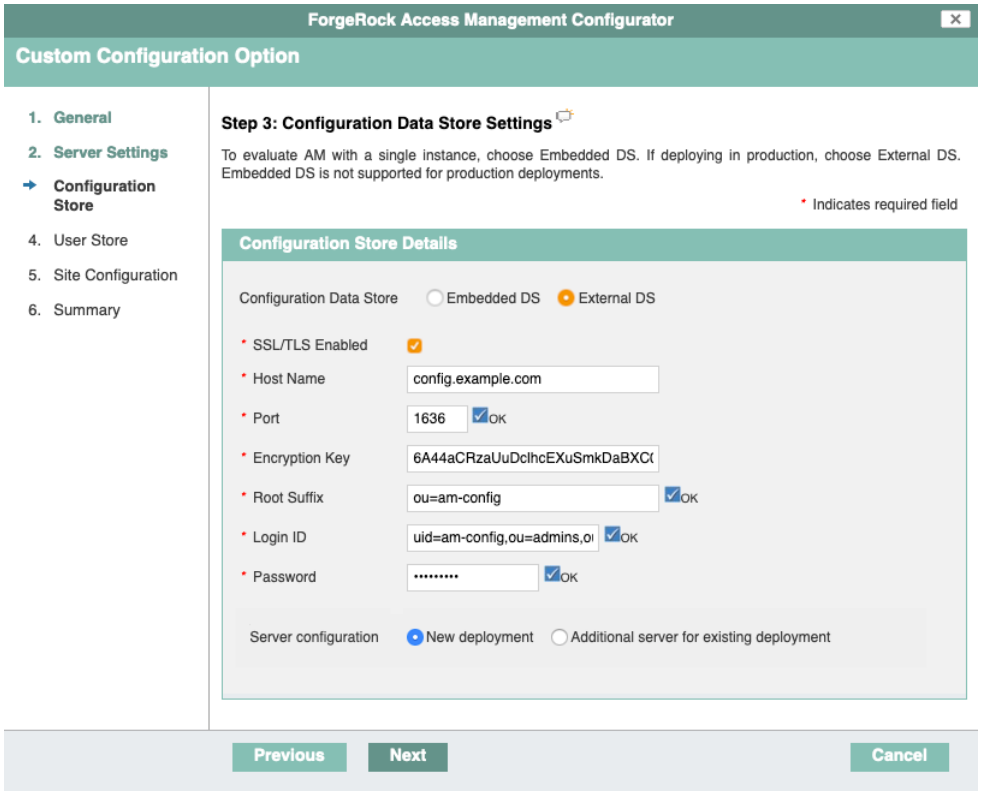
4. Start Tomcat if it is not already running.
5. Navigate to the deployed AM application; for example, http://am.example.com:8081/am/.
6. Select Create New Configuration to create a custom configuration.
7. Accept the license agreement and click Continue.
8. Set a password for the default user, `amAdmin`.

   This guide assumes that the `amAdmin` password is `Passw0rd`.
9. On the Server Settings screen, enter your AM server settings; for example:
   - Server URL: `http://am.example.com:8081`
   - Cookie Domain: `example.com`
   - Platform Locale: `en_US`
   - Configuration Directory: `/path/to/am`
10. On the Configuration Data Store Settings screen, select External DS, and enter the details for the DS instance that you set up as a configuration store.

    This list reflects the DS configuration store installed with the listed "Server Settings".
    - SSL/TLS: `Enabled`

      TLS is required in a production deployment.
    - Host Name: `config.example.com`
    - Port: `1636`
    - Encryption Key: (generated encryption key)
    - Root Suffix: `ou=am-config`
    - Login ID: `uid=am-config,ou=admins,ou=am-config`

- Password: `5up35tr0ng`
- Server configuration: `New deployment`



11. On the User Data Store Settings screen, select External User Data Store, and enter the details for the DS instance that you set up as an identity store.

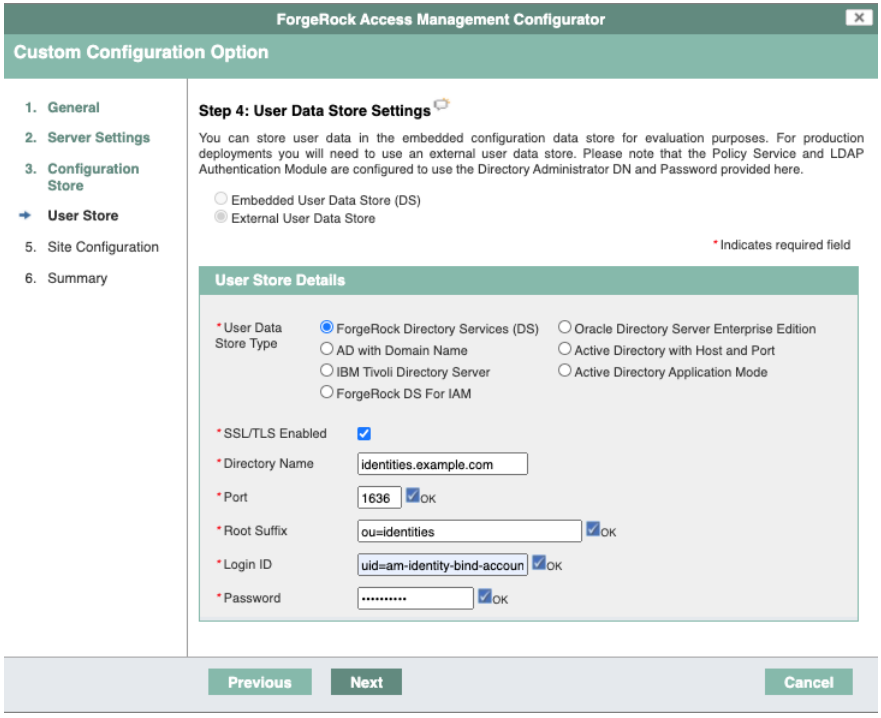
    This list reflects the DS identity store installed with the listed "Server Settings".

    - User Data Store Type: `ForgeRock Directory Services (DS)`
    - SSL/TLS: `Enabled`

      TLS is required in a production deployment.

    - Host Name: `identities.example.com`
    - Port: `1636`

- Root Suffix: `ou=identities`
- Login ID: `uid=am-identity-bind-account,ou=admins,ou=identities`
- Password: `5up35tr0ng`



12. On the Site Configuration screen, select No.
13. Click Create Configuration.

Important

This sample deployment uses only the AM top level realm for simplicity. In production, use AM subrealms.

This sample deployment may show limitations inherent in not using additional realms as suggested. For example, to create an external application store, you must first set up a separate subrealm, and then create the application store there.

Configure OAuth Clients

This procedure configures *four* OAuth 2.0 clients. All of these clients are configured in the Top Level Realm. If you are using sub-realms, you must configure the clients for *each* realm.

1. If you're not currently logged in to the AM console as the `amAdmin` user, log in.
2. Configure an `idm-resource-server` client to introspect access tokens.

   Note

   This is not a *real* OAuth 2.0 client, in the traditional sense. Rather, it is an OAuth 2.0 Resource Server account, used exclusively for token introspection. As such, you do not need to specify redirect URIs or grant types.

   a. Select Applications > OAuth 2.0 > Clients, and click Add Client.

   b. Enter the following details:

   - Client ID: `idm-resource-server`
   - Client secret: `password`

     The value of this field must match the `clientSecret` that you will set in the `rsFilter` module in the IDM authentication configuration (`/path/to/openidm/conf/authentication.json`) during your IDM setup.

   - Scopes: `am-introspect-all-tokens`

     This scope gives the client the right to introspect tokens issued to other clients. If you expect to accept access tokens issued from sub-realms, you must also include the scope `am-introspect-all-tokens-any-realm`.

   c. Click Create.

3. Configure an `idm-provisioning` client to make calls to IDM:

   a. Select Applications > OAuth 2.0 > Clients, and click Add Client.

   b. Enter the following details:

   - Client ID: `idm-provisioning`
   - Client secret: `openidm`

- Scopes: `fr:idm:*`

c. Click Create.

d. On the Advanced tab:

- Response Types: Check that `token` is present (it should be there by default).
- Grant Types: Remove `Authorization Code` and add `Client Credentials`.

e. Click Save Changes.

4. Configure an `idm-admin-ui` client that will be used by the Platform Admin UI:

a. In the Top Level Realm, select Applications > OAuth 2.0 > Clients, and click Add Client.

b. Enter the following details:

- Client ID: `idm-admin-ui`
- Client Secret: (no client secret is required)
- Redirection URIs: `http://openidm.example.com:8080/platform/appAuthHelperRedirect.html` `http://openidm.example.com:8080/platform/sessionCheck.html` `http://openidm.example.com:8080/admin/appAuthHelperRedirect.html` `http://openidm.example.com:8080/admin/sessionCheck.html` `http://admin.example.com:8082/appAuthHelperRedirect.html` `http://admin.example.com:8082/sessionCheck.html` `http://am.example.com:8081/platform/appAuthHelperRedirect.html` `http://am.example.com:8081/platform/sessionCheck.html`
- Scopes: `openid` `fr:idm:*`

> **Note**
>
> At a minimum, the scopes that you set here must include the scopes that you will set in the `rsFilter` authentication configuration (`/path/to/openidm/conf/authentication.json`) during your IDM setup.

c. Click Create.

d. On the Core tab:

- Client type: Select `Public`.

Click Save Changes.

e. On the Advanced tab:

- Grant Types: Add `Implicit`.

- Token Endpoint Authentication Method: Select `none`.
- Implied consent: Enable.

Click Save Changes.

5. Configure an `end-user-ui` client that will be used by the Platform End User UI:
   a. In the Top Level Realm, select Applications > OAuth 2.0 > Clients, and click Add Client.
   b. Enter the following details:
      - Client ID: `end-user-ui`
      - Client Secret: (no client secret is required)
      - Redirection URIs: `http://enduser.example.com:8888/appAuthHelperRedirect.html` `http://enduser.example.com:8888/sessionCheck.html` `http://am.example.com:8081/enduser/appAuthHelperRedirect.html` `http://am.example.com:8081/enduser/sessionCheck.html`
      - Scopes: `openid` `fr:idm:*`

      Note

      At a minimum, the scopes that you set here must include the scopes that you will set in the `rsFilter` authentication configuration (`/path/to/openidm/conf/authentication.json`) during your IDM setup.

   c. Click Create.
   d. On the Core tab:
      - Client type: Select `Public`.

      Click Save Changes.
   e. On the Advanced tab:
      - Grant Types: Add `Implicit`.
      - Token Endpoint Authentication Method: Select `none`.
      - Implied Consent: Enable.

      Click Save Changes.

Configure an OAuth 2.0 Provider Service

1. If you're not currently logged in to the AM console as the `amAdmin` user, log in.

2. In the Top Level Realm, select Services, and click Add a Service.
3. Under Choose a service type, select OAuth2 Provider.
4. For Client Registration Scope Whitelist, add the following scopes:
   - `fr:idm:*`
   - `am-introspect-all-tokens`
   - `openid`
5. Click Create.
6. On the Advanced tab, make sure that Response Type Plugins includes the following values:

```
id_token|org.forgerock.openidconnect.IdTokenResponseTypeHandler
```

```
code|org.forgerock.oauth2.core.AuthorizationCodeResponseTypeHandler
```

7. Click Save Changes.
8. On the Consent tab, enable Allow Clients to Skip Consent.
9. Click Save Changes.

Configure an IDM Provisioning Service

1. If you're not currently logged in to the AM console as the `amAdmin` user, log in.
2. From the top menu, select Configure > Global Services > IDM Provisioning.
3. Set the following fields:
   - Enabled
   - Deployment URL: `http://openidm.example.com:8080`
   - Deployment Path: `openidm`
   - IDM Provisioning Client: `idm-provisioning`
4. Click Save Changes.

Configure a Validation Service

The Platform UIs need this validation service allow listing for `goto` redirection.

1. If you're not currently logged in to the AM console as the `amAdmin` user, log in.

2. In the Top Level Realm, select Services, and click Add a Service.
3. Under Choose a service type, select Validation Service.
4. For Valid goto URL Resources, add the URLs for the Platform UI:

```
http://admin.example.com:8082/*
http://admin.example.com:8082/*?*
http://login.example.com:8083/*
http://login.example.com:8083/*?*
http://enduser.example.com:8888/*
http://enduser.example.com:8888/*?*
```

5. Click Create.

Enable CORS Support

Cross-origin resource sharing (CORS) lets user agents make requests across domains.

1. If you're not currently logged in to the AM console as the `amAdmin` user, log in.
2. From the top menu, select Configure > Global Services > CORS Service.
3. On the Secondary Configurations tab, click Add a Secondary Configuration.
4. On the New Configuration screen, enter the following values:
   - Name: `Cors Configuration`
   - Accepted Origins:

     ```
     http://login.example.com:8083
     http://admin.example.com:8082
     http://enduser.example.com:8888
     http://openidm.example.com:8080
     https://openidm.example.com:8443
     ```

     List only the origins that will be hosting OAuth 2.0 clients (such as the `platform-enduser` and IDM Admin UIs).
   - Accepted Methods:

     ```
     HEAD
     DELETE
     POST
     GET
     PUT
     PATCH
     ```

   - Accepted Headers:

`authorization`
`x-openidm-username`
`if-none-match`
`x-openidm-nosession`
`x-openidm-password`
`accept-api-version`
`x-requested-with`
`content-type`
`if-match`
`cache-control`
`user-agent`

- Exposed Headers: `WWW-Authenticate`

5. Click Create.
6. On the Cors Configuration screen, set the following values:
   - Enable the CORS filter: Enable
   - Max Age: `600`
   - Allow Credentials: Enable
7. Click Save Changes.

Configure Authentication Trees

The platform deployment relies on three authentication trees to enable authentication through AM. When you extract the AM `.zip` file, you will get a `sample-trees-7.1.0.zip` file that contains a number of sample authentication trees, in JSON files. Use the Amster command-line utility to import the platform authentication trees into your AM configuration:

1. Extract the `sample-trees-7.1.0.zip` file and list the sample trees in the `/path/to/openam-samples/root/AuthTree` directory:

```
ls /path/to/openam-samples/root/AuthTree
Agent.json                          PlatformForgottenUsername.json
Example.json                        PlatformLogin.json
Facebook-ProvisionIDMAccount.json   PlatformProgressiveProfile.json
Google-AnonymousUser.json           PlatformRegistration.json
Google-DynamicAccountCreation.json  PlatformResetPassword.json
HmacOneTimePassword.json            PlatformUpdatePassword.json
PersistentCookie.json               RetryLimit.json
```

2. Download and install Amster.
3. Start Amster, then connect to your AM instance:

```
./amster
Amster OpenAM Shell (7.1.0 build @build.number@, JVM: version)
Type ':help' or ':h' for help.
---------------------------------------------------------------------------
am> connect --interactive http://am.example.com:8081/am
Sign in
User Name: amAdmin
Password: ********
amster am.example.com:8081>
```

4. Import the sample authentication trees and nodes:

```
amster am.example.com:8081> import-config --path /path/to/openam-samples/root
Importing directory /path/to/openam-samples/root/AcceptTermsAndConditions
Imported /path/to/openam-samples/root/AcceptTermsAndConditions/b4a0e915-
c15d-4b83-9c9d-18347d645976.json
...
Import completed successfully
```

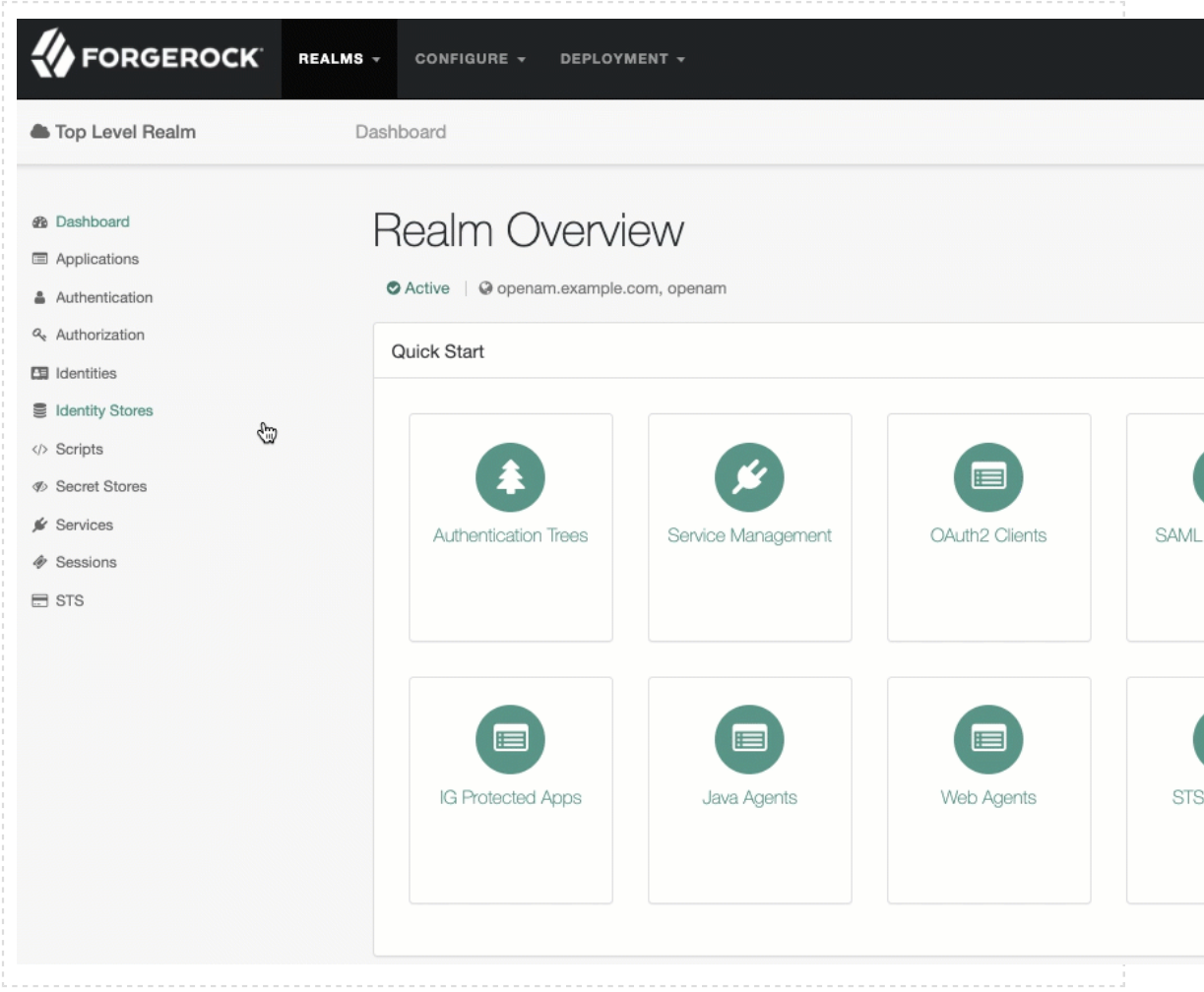
5. If you're not currently logged in to the AM console as the `amAdmin` user, log in.

6. Configure the `PlatformRegistration` tree:

   a. In the Top Level Realm, select Authentication > Trees, and click on PlatformRegistration.

   b. On the `PlatformRegistration` tree, add a `Success URL` node between `Increment Login Count` and `Success`, and set its value to `http://enduser.example.com:8888`.

      + *Show Me*



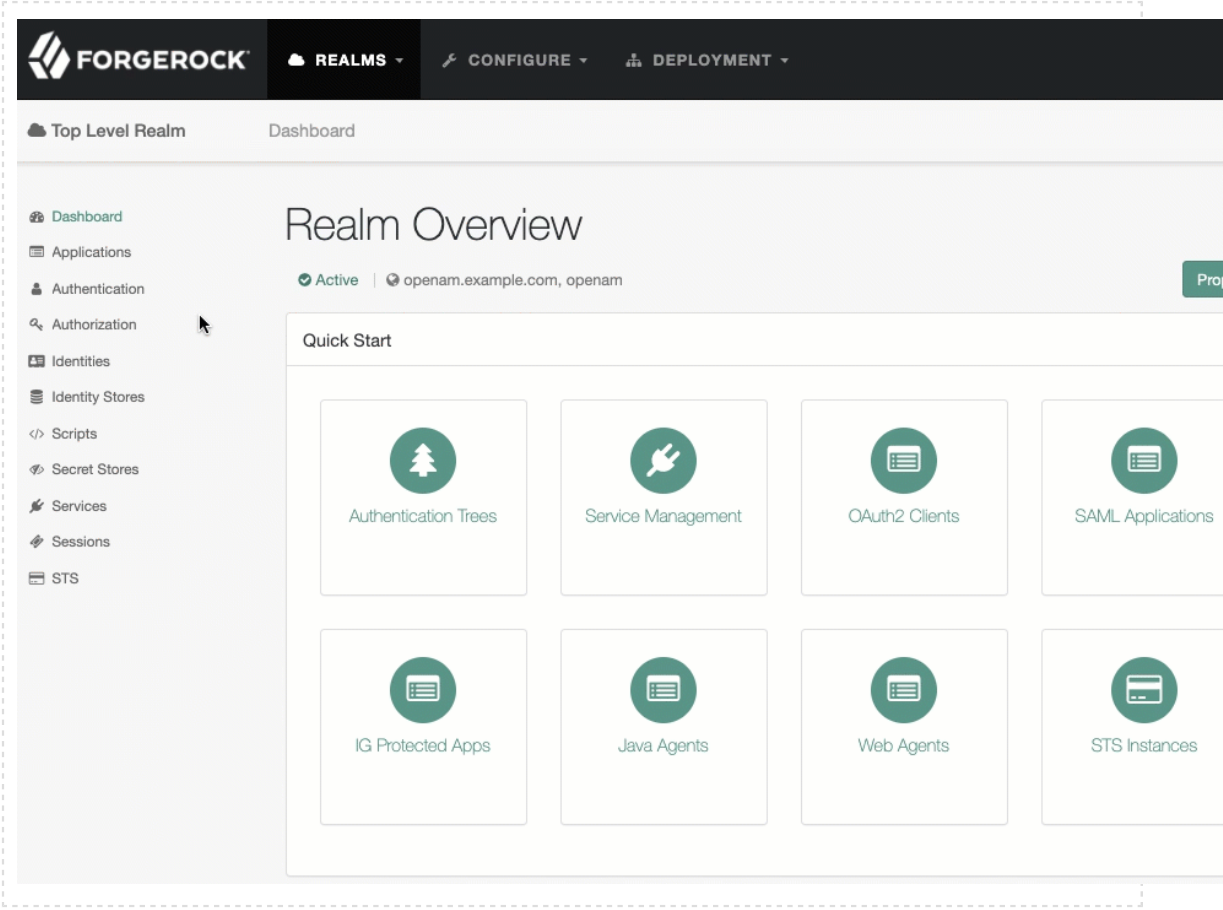
c. Click Save.

7. Configure the `PlatformLogin` tree:

   a. In the Top Level Realm, select Authentication > Trees, and click on PlatformLogin.

   b. On the `PlatformLogin` tree, add a `Success URL` node between `Inner Tree Evaluator` and `Success`, and set its value to `http://enduser.example.com:8888`.

      + *Show Me*



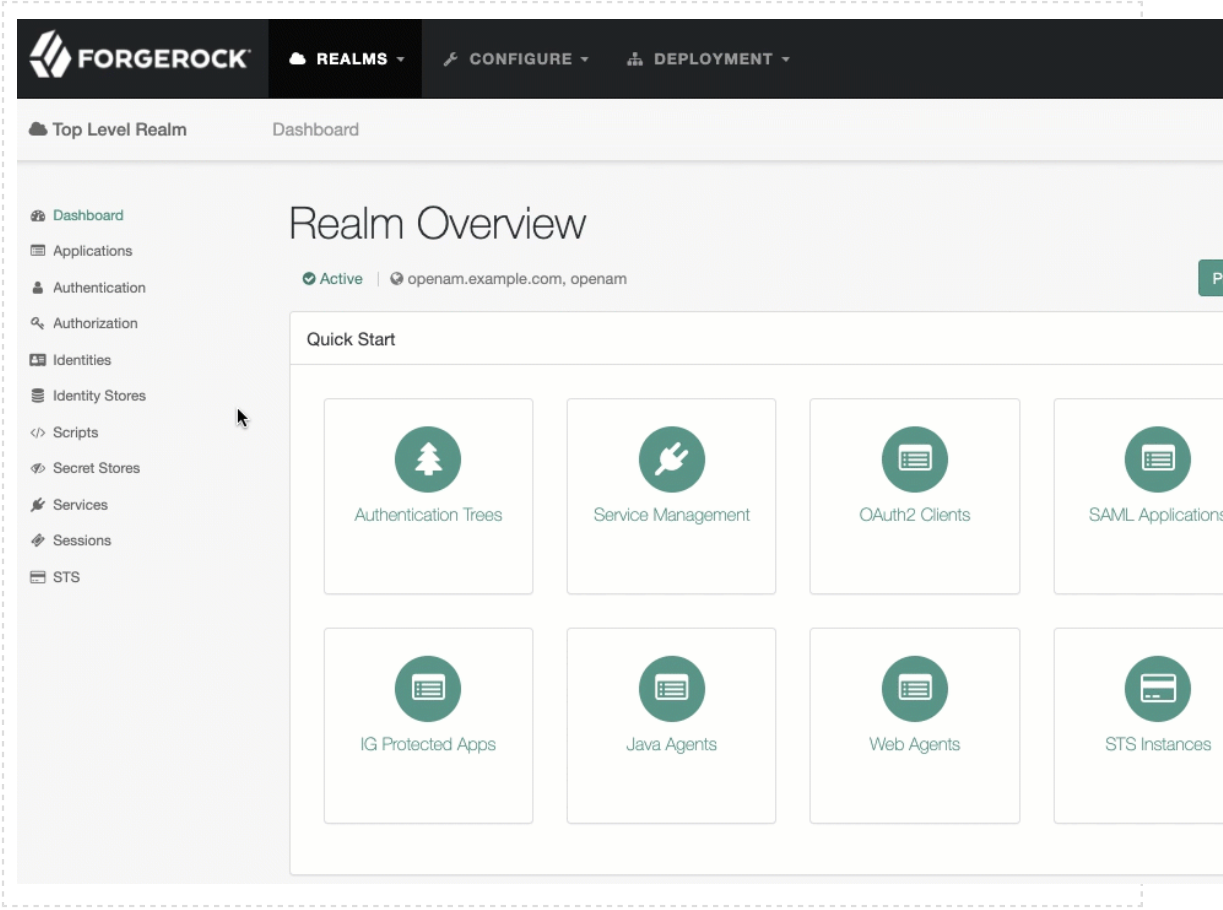
c. Click Save.

8. Configure the `PlatformResetPassword` tree:

   a. In the Top Level Realm, select Authentication > Trees, and click on PlatformResetPassword.

   b. On the `PlatformResetPassword` tree, add a `Success URL` node between `Patch Object` and `Success`, and set its value to `http://enduser.example.com:8888`.

      + *Show Me*



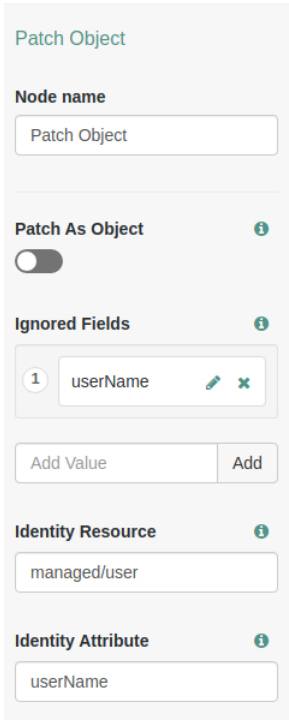
c. Click Save.

9. Configure the `PlatformUpdatePassword` tree:

   a. In the Top Level Realm, select Authentication > Trees, and click on PlatformUpdatePassword.

   b. On the `PlatformUpdatePassword` tree, select the `Patch Object` node, and set its `Patch As Object` value to `false`.

      + *Show Me*



c. Click Save.

10. For the authentication trees that require email, set the External Login Page URL.

    In the Top Level Realm, select Authentication > Settings, and click the General tab. Set External Login Page URL to `http://login.example.com:8083`, then click Save Changes.

Map Authentication Trees

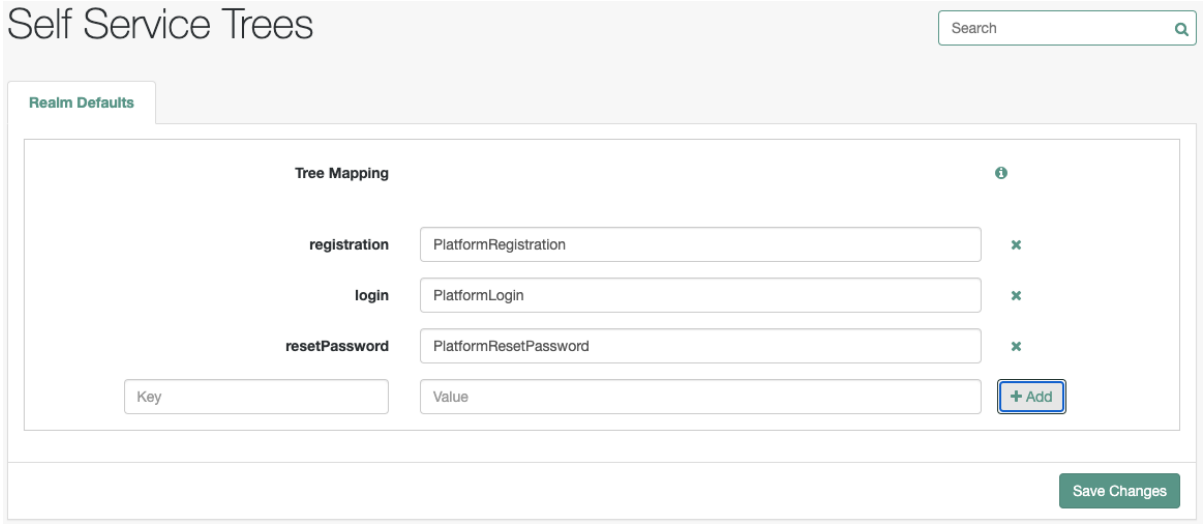
Map the platform trees to the corresponding Self-Service endpoints. For more information about this step, see "Configure Self-Service Trees Endpoints" in the *Platform Self-Service Guide*.

1. From the top menu, select Configure > Global Services > Self Service Trees.

2. Add the following Tree Mappings:

Key	Value
registration	PlatformRegistration
login	PlatformLogin

Key	Value
resetPassword	PlatformResetPassword



3. Click Save Changes.

Enable the Password Update Tree (optional)

To let end users update their own passwords (using the `PlatformUpdatePassword` tree), add the `idm-provisioning` service as an authorized client of the `OAuth2 Provider` service.

You cannot use the UI for this step—you must update the OAuth 2.0 configuration over REST:

1. Use the following request to get the token ID:

```
curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: amAdmin" \
--header "X-OpenAM-Password: Passw0rd" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'http://am.example.com:8081/am/json/realms/root/authenticate'
{
  "tokenId": "tokenId",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

2. Make a GET request to the `oauth-oidc` endpoint to return the current OAuth configuration:

   + *Example Curl Request*

```
curl \
-X GET \
-H 'Accept-API-Version: protocol=1.0,resource=1.0' \
-H 'X-Requested-With: XMLHttpRequest' \
-H 'Origin: http://am.example.com:8081' \
-H 'Referer: http://am.example.com:8081/am/ui-admin/' \
-H 'Cookie: amlbcookie=01; iPlanetDirectoryPro=tokenId' \
"http://am.example.com:8081/am/json/realms/root/realm-config/services/oauth-oidc"
{
  "coreOAuth2Config": {
    "refreshTokenLifetime": 604800,
    "accessTokenLifetime": 3600,
    "usePolicyEngineForScope": false,
    "codeLifetime": 120,
    "issueRefreshTokenOnRefreshedToken": true,
    "macaroonTokensEnabled": false,
    "issueRefreshToken": true,
    "accessTokenModificationScript": "d22f9a0c-426a-4466-b95e-d0f125b0d5fa",
    "statelessTokensEnabled": false
  },
  "coreOIDCConfig": {
    "supportedIDTokenEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "jwtTokenLifetime": 3600,
    "supportedClaims": [],
    "supportedIDTokenEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedIDTokenSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
```

```
    "oidcClaimsScript": "36863ffb-40ec-48b9-94b1-9a99f71cc3b5"
  },
  "advancedOAuth2Config": {
    "supportedScopes": [
      "openid",
      "fr:idm:*",
      "am-introspect-all-tokens"
    ],
    "tlsCertificateRevocationCheckingEnabled": false,
    "codeVerifierEnforced": "false",
    "tokenSigningAlgorithm": "HS256",
    "authenticationAttributes": [
      "uid"
    ],
    "passwordGrantAuthService": "[Empty]",
    "defaultScopes": [],
    "tlsClientCertificateHeaderFormat": "URLENCODED_PEM",
    "scopeImplementationClass": "org.forgerock.openam.oauth2.OpenAMScopeValidator",
    "responseTypeClasses": [
      "code|org.forgerock.oauth2.core.AuthorizationCodeResponseTypeHandler",
      "id_token|org.forgerock.openidconnect.IdTokenResponseTypeHandler",
      "device_code|org.forgerock.oauth2.core.TokenResponseTypeHandler",
      "token|org.forgerock.oauth2.core.TokenResponseTypeHandler"
    ],
    "tlsCertificateBoundAccessTokensEnabled": true,
    "hashSalt": "changeme",
    "moduleMessageEnabledInPasswordGrant": false,
    "tokenEncryptionEnabled": false,
    "tokenCompressionEnabled": false,
    "grantTypes": [
      "implicit",
      "urn:ietf:params:oauth:grant-type:saml2-bearer",
      "refresh_token",
      "password",
      "client_credentials",
      "urn:ietf:params:oauth:grant-type:device_code",
      "authorization_code",
      "urn:openid:params:grant-type:ciba",
      "urn:ietf:params:oauth:grant-type:uma-ticket",
      "urn:ietf:params:oauth:grant-type:jwt-bearer"
    ],
    "displayNameAttribute": "cn",
    "macaroonTokenFormat": "V2",
    "supportedSubjectTypes": [
      "public"
    ]
  },
  "advancedOIDCConfig": {
    "storeOpsTokens": true,
    "defaultACR": [],
    "supportedRequestParameterEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "claimsParameterSupported": false,
```

```
    "amrMappings": {},
    "supportedUserInfoEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "authorisedIdmDelegationClients": [],
    "alwaysAddClaimsToToken": false,
    "supportedUserInfoSigningAlgorithms": [
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512"
    ],
    "supportedRequestParameterEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "ECDH-ES+A128KW",
      "RSA-OAEP",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenIntrospectionResponseEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "supportedTokenIntrospectionResponseSigningAlgorithms": [
      "PS384",
      "RS384",
      "EdDSA",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "authorisedOpenIdConnectSSOClients": [],
    "idTokenInfoClientAuthenticationEnabled": true,
    "supportedRequestParameterSigningAlgorithms": [
      "PS384",
```

```
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "supportedUserInfoEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenIntrospectionResponseEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenEndpointAuthenticationSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "loaMapping": {}
  },
  "clientDynamicRegistrationConfig": {
    "dynamicClientRegistrationSoftwareStatementRequired": false,
    "dynamicClientRegistrationScope": "dynamic_client_registration",
    "requiredSoftwareStatementAttestedAttributes": [
      "redirect_uris"
    ],
    "generateRegistrationAccessTokens": true,
```

```
    "allowDynamicRegistration": false
  },
  "cibaConfig": {
    "supportedCibaSigningAlgorithms": [
      "ES256",
      "PS256"
    ],
    "cibaAuthReqIdLifetime": 600,
    "cibaMinimumPollingInterval": 2
  },
  "consent": {
    "enableRemoteConsent": false,
    "supportedRcsRequestSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "supportedRcsResponseSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "clientsCanSkipConsent": true,
    "supportedRcsRequestEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedRcsResponseEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
```

```
      "A256CBC-HS512"
    ],
    "supportedRcsRequestEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "remoteConsentServiceId": "[Empty]",
    "supportedRcsResponseEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "ECDH-ES+A128KW",
      "RSA-OAEP",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ]
  },
  "deviceCodeConfig": {
    "devicePollInterval": 5,
    "deviceCodeLifetime": 300
  },
  "_id": "",
  "_type": {
    "_id": "oauth-oidc",
    "name": "OAuth2 Provider",
    "collection": false
  }
}
```

3. Make a PUT request to the `oauth-oidc` endpoint to submit the JSON payload returned in the previous step, replacing the value of the `authorisedIdmDelegationClients` property under `advancedOIDCConfig` with `idm-provisioning`:

```
"authorisedIdmDelegationClients": [ "idm-provisioning"],
```

+ *Example Curl Request*

```
curl \
--request PUT \
-H 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:70.0) Gecko/20100101
 Firefox/70.0' \
-H 'Accept: application/json, text/javascript, */*; q=0.01' \
-H 'Accept-Language: en-US' \
--compressed \
-H 'Content-Type: application/json' \
-H 'Accept-API-Version: protocol=1.0,resource=1.0' \
-H 'X-Requested-With: XMLHttpRequest' \
-H 'Cache-Control: no-cache' \
```

```
-H 'Origin: http://am.example.com:8081' \
-H 'DNT: 1' \
-H 'Connection: keep-alive' \
-H 'Referer: http://am.example.com:8081/am/ui-admin/' \
-H 'Cookie: amlbcookie=01; iPlanetDirectoryPro=tokenId' \
--data '{
  "coreOAuth2Config": {
    "refreshTokenLifetime": 604800,
    "accessTokenLifetime": 3600,
    "usePolicyEngineForScope": false,
    "codeLifetime": 120,
    "issueRefreshTokenOnRefreshedToken": true,
    "macaroonTokensEnabled": false,
    "issueRefreshToken": true,
    "accessTokenModificationScript": "d22f9a0c-426a-4466-b95e-d0f125b0d5fa",
    "statelessTokensEnabled": false
  },
  "coreOIDCConfig": {
    "supportedIDTokenEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "jwtTokenLifetime": 3600,
    "supportedClaims": [],
    "supportedIDTokenEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedIDTokenSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "oidcClaimsScript": "36863ffb-40ec-48b9-94b1-9a99f71cc3b5"
  },
  "advancedOAuth2Config": {
    "supportedScopes": [
      "openid",
```

```
      "fr:idm:*",
      "am-introspect-all-tokens"
    ],
    "tlsCertificateRevocationCheckingEnabled": false,
    "codeVerifierEnforced": "false",
    "tokenSigningAlgorithm": "HS256",
    "authenticationAttributes": [
      "uid"
    ],
    "passwordGrantAuthService": "[Empty]",
    "defaultScopes": [],
    "tlsClientCertificateHeaderFormat": "URLENCODED_PEM",
    "scopeImplementationClass": "org.forgerock.openam.oauth2.OpenAMScopeValidator",
    "responseTypeClasses": [
      "code|org.forgerock.oauth2.core.AuthorizationCodeResponseTypeHandler",
      "id_token|org.forgerock.openidconnect.IdTokenResponseTypeHandler",
      "device_code|org.forgerock.oauth2.core.TokenResponseTypeHandler",
      "token|org.forgerock.oauth2.core.TokenResponseTypeHandler"
    ],
    "tlsCertificateBoundAccessTokensEnabled": true,
    "hashSalt": "changeme",
    "moduleMessageEnabledInPasswordGrant": false,
    "tokenEncryptionEnabled": false,
    "tokenCompressionEnabled": false,
    "grantTypes": [
      "implicit",
      "urn:ietf:params:oauth:grant-type:saml2-bearer",
      "refresh_token",
      "password",
      "client_credentials",
      "urn:ietf:params:oauth:grant-type:device_code",
      "authorization_code",
      "urn:openid:params:grant-type:ciba",
      "urn:ietf:params:oauth:grant-type:uma-ticket",
      "urn:ietf:params:oauth:grant-type:jwt-bearer"
    ],
    "displayNameAttribute": "cn",
    "macaroonTokenFormat": "V2",
    "supportedSubjectTypes": [
      "public"
    ]
  },
  "advancedOIDCConfig": {
    "storeOpsTokens": true,
    "authorisedIdmDelegationClients": ["idm-provisioning"],
    "defaultACR": [],
    "supportedRequestParameterEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "claimsParameterSupported": false,
    "amrMappings": {},
    "supportedUserInfoEncryptionEnc": [
      "A256GCM",
      "A192GCM",
```

```
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "alwaysAddClaimsToToken": false,
    "supportedUserInfoSigningAlgorithms": [
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512"
    ],
    "supportedRequestParameterEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "ECDH-ES+A128KW",
      "RSA-OAEP",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenIntrospectionResponseEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "supportedTokenIntrospectionResponseSigningAlgorithms": [
      "PS384",
      "RS384",
      "EdDSA",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "authorisedOpenIdConnectSSOClients": [],
    "idTokenInfoClientAuthenticationEnabled": true,
    "supportedRequestParameterSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
```

```
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "supportedUserInfoEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenIntrospectionResponseEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenEndpointAuthenticationSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "loaMapping": {}
  },
  "clientDynamicRegistrationConfig": {
    "dynamicClientRegistrationSoftwareStatementRequired": false,
    "dynamicClientRegistrationScope": "dynamic_client_registration",
    "requiredSoftwareStatementAttestedAttributes": [
      "redirect_uris"
    ],
    "generateRegistrationAccessTokens": true,
    "allowDynamicRegistration": false
  },
  "cibaConfig": {
    "supportedCibaSigningAlgorithms": [
      "ES256",
```

```
      "PS256"
    ],
    "cibaAuthReqIdLifetime": 600,
    "cibaMinimumPollingInterval": 2
  },
  "consent": {
    "enableRemoteConsent": false,
    "supportedRcsRequestSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "supportedRcsResponseSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "clientsCanSkipConsent": true,
    "supportedRcsRequestEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedRcsResponseEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "supportedRcsRequestEncryptionMethods": [
      "A256GCM",
      "A192GCM",
```

```
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "remoteConsentServiceId": "[Empty]",
    "supportedRcsResponseEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "ECDH-ES+A128KW",
      "RSA-OAEP",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ]
  },
  "deviceCodeConfig": {
    "devicePollInterval": 5,
    "deviceCodeLifetime": 300
  },
  "_id": "",
  "_type": {
    "_id": "oauth-oidc",
    "name": "OAuth2 Provider",
    "collection": false
  }
}' \
"http://am.example.com:8081/am/json/realms/root/realm-config/services/oauth-oidc"
{
  "coreOAuth2Config": {
    "refreshTokenLifetime": 604800,
    "accessTokenLifetime": 3600,
    "usePolicyEngineForScope": false,
    "codeLifetime": 120,
    "issueRefreshTokenOnRefreshedToken": true,
    "macaroonTokensEnabled": false,
    "issueRefreshToken": true,
    "accessTokenModificationScript": "d22f9a0c-426a-4466-b95e-d0f125b0d5fa",
    "statelessTokensEnabled": false
  },
  "coreOIDCConfig": {
    "supportedIDTokenEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "jwtTokenLifetime": 3600,
    "supportedClaims": [],
    "supportedIDTokenEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
```

```
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedIDTokenSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "oidcClaimsScript": "36863ffb-40ec-48b9-94b1-9a99f71cc3b5"
  },
  "advancedOAuth2Config": {
    "supportedScopes": [
      "openid",
      "fr:idm:*",
      "am-introspect-all-tokens"
    ],
    "tlsCertificateRevocationCheckingEnabled": false,
    "codeVerifierEnforced": "false",
    "tokenSigningAlgorithm": "HS256",
    "authenticationAttributes": [
      "uid"
    ],
    "passwordGrantAuthService": "[Empty]",
    "defaultScopes": [],
    "tlsClientCertificateHeaderFormat": "URLENCODED_PEM",
    "scopeImplementationClass": "org.forgerock.openam.oauth2.OpenAMScopeValidator",
    "responseTypeClasses": [
      "code|org.forgerock.oauth2.core.AuthorizationCodeResponseTypeHandler",
      "id_token|org.forgerock.openidconnect.IdTokenResponseTypeHandler",
      "device_code|org.forgerock.oauth2.core.TokenResponseTypeHandler",
      "token|org.forgerock.oauth2.core.TokenResponseTypeHandler"
    ],
    "tlsCertificateBoundAccessTokensEnabled": true,
    "hashSalt": "changeme",
    "moduleMessageEnabledInPasswordGrant": false,
    "tokenEncryptionEnabled": false,
    "tokenCompressionEnabled": false,
    "grantTypes": [
      "implicit",
      "urn:ietf:params:oauth:grant-type:saml2-bearer",
      "refresh_token",
      "password",
      "client_credentials",
      "urn:ietf:params:oauth:grant-type:device_code",
      "authorization_code",
      "urn:openid:params:grant-type:ciba",
```

```
      "urn:ietf:params:oauth:grant-type:uma-ticket",
      "urn:ietf:params:oauth:grant-type:jwt-bearer"
    ],
    "displayNameAttribute": "cn",
    "macaroonTokenFormat": "V2",
    "supportedSubjectTypes": [
      "public"
    ]
  },
  "advancedOIDCConfig": {
    "storeOpsTokens": true,
    "defaultACR": [],
    "supportedRequestParameterEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "claimsParameterSupported": false,
    "amrMappings": {},
    "supportedUserInfoEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
   "authorisedIdmDelegationClients": ["idm-provisioning"],
   "alwaysAddClaimsToToken": false,
    "supportedUserInfoSigningAlgorithms": [
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512"
    ],
    "supportedRequestParameterEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "ECDH-ES+A128KW",
      "RSA-OAEP",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenIntrospectionResponseEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
```

```
      "A256CBC-HS512"
    ],
    "supportedTokenIntrospectionResponseSigningAlgorithms": [
      "PS384",
      "RS384",
      "EdDSA",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "authorisedOpenIdConnectSSOClients": [],
    "idTokenInfoClientAuthenticationEnabled": true,
    "supportedRequestParameterSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "supportedUserInfoEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenIntrospectionResponseEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenEndpointAuthenticationSigningAlgorithms": [
      "PS384",
```

```
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "loaMapping": {}
  },
  "clientDynamicRegistrationConfig": {
    "dynamicClientRegistrationSoftwareStatementRequired": false,
    "dynamicClientRegistrationScope": "dynamic_client_registration",
    "requiredSoftwareStatementAttestedAttributes": [
      "redirect_uris"
    ],
    "generateRegistrationAccessTokens": true,
    "allowDynamicRegistration": false
  },
  "cibaConfig": {
    "supportedCibaSigningAlgorithms": [
      "ES256",
      "PS256"
    ],
    "cibaAuthReqIdLifetime": 600,
    "cibaMinimumPollingInterval": 2
  },
  "consent": {
    "enableRemoteConsent": false,
    "supportedRcsRequestSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "supportedRcsResponseSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
```

```
      "RS512"
    ],
    "clientsCanSkipConsent": true,
    "supportedRcsRequestEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedRcsResponseEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "supportedRcsRequestEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "remoteConsentServiceId": "[Empty]",
    "supportedRcsResponseEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "ECDH-ES+A128KW",
      "RSA-OAEP",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ]
  },
  "deviceCodeConfig": {
    "devicePollInterval": 5,
    "deviceCodeLifetime": 300
  },
  "_id": "",
  "_type": {
    "_id": "oauth-oidc",
    "name": "OAuth2 Provider",
    "collection": false
  }
```

```
}
```

Set up IDM

This procedure sets up IDM with an external MySQL repository. The procedure assumes that IDM is installed according to the listed "Server Settings".

1. Follow the instructions in the *IDM Installation Guide* to install and run IDM.
2. Edit the `/path/to/openidm/resolver/boot.properties` file to set the hostname:

   ```
   openidm.host=openidm.example.com
   ```
3. Configure your IDM repository. This procedure was tested with a MySQL repository. Follow the instructions in the *IDM Installation Guide* to set up a MySQL repository.
4. (Optional) Configure social authentication.

   In your project's `conf/managed.json` file:

   a. Add an `aliasList` property to the `user` object:

   ```
   {
     "objects": [
       {
         "name": "user",
         ...
         "schema": {
           "properties": {
             ...
             "aliasList": {
               "title": "User Alias Names List",
               "description": "List of identity aliases used primarily to record social IdP subjects
    for this user",
               "type": "array",
               "items": {
                 "type": "string",
                 "title": "User Alias Names Items"
               },
               "viewable": false,
               "searchable": false,
               "userEditable": true,
               "returnByDefault": false,
               "isVirtual": false
             }
       ...
     ]
   }
   ```

   b. Update the `password` property to ensure that users update their passwords through the self-service APIs, not directly:

   ```
   "userEditable" : false
   ```

5. Change the authentication mechanism to `rsFilter` only:

   - Replace the default `conf/authentication.json` file with this file.
   - Check that the `clientSecret` matches the `Client secret` that you set for the `idm-resource-server` client in AM (see "Configure OAuth Clients").
   - Check that the `scopes` match the `Scope(s)` that you set for the `idm-admin-ui` and `end-user-ui` clients in AM (see "Configure OAuth Clients").

   Note

   The `subject` property in the `staticUserMapping` in this module is case-insensitive. So logging into the UI as user `amAdmin` or as user `amadmin` will work.

   For more information about authenticating using the `rsFilter`, see Authenticating through AM in the *IDM Authentication and Authorization Guide*.

6. Edit the IDM Admin UI configuration so that you can still authenticate through the IDM Admin UI:

   a. In your `conf/ui-configuration.json` file, insert a `platformSettings` object into the `configuration` object:

   ```
   {
       "configuration" : {
           ...,
           "platformSettings" : {
               "adminOauthClient" : "idm-admin-ui",
               "adminOauthClientScopes" : "fr:idm:*",
               "amUrl" : "http://am.example.com:8081/am",
               "loginUrl" : ""
           }
       }
   }
   ```

   This object tells the IDM Admin UI that it is operating in "platform mode" (that is, as an OAuth 2.0 client of AM).

   b. In your `conf/ui.context-admin.json` file, check that `X-Frame-Options` is set to `SAMEORIGIN`:

   + *Sample ui.context-admin.json*

```
{
  "enabled" : true,
  "urlContextRoot" : "/admin",
  "defaultDir" : "&{idm.install.dir}/ui/admin/default",
  "extensionDir" : "&{idm.install.dir}/ui/admin/extension",
  "responseHeaders" : {
      "X-Frame-Options" : "SAMEORIGIN"
  }
}
```

You should now be able to access the IDM Admin UI at http://openidm.example.com:8080/admin. When you log in to the Admin UI, use the default AM administrative user (`amAdmin`), and not `openidm-admin`.

7. Configure the CORS servlet filter.

   Replace the default `conf/servletfilter-cors.json` file with this file.

8. Configure synchronization between the IDM repository and the AM identity store.

   a. Add a configuration for the LDAP connector.

      Create a configuration file named `provisioner.openicf-ldap.json` in the `/path/to/openidm/conf` directory. Use this file as a template.

      Pay particular attention to the connection properties, `host`, `port`, `principal`, and `credentials`. These must match the configuration of the DS server that you set up as the identity store.

   b. Add a mapping between IDM managed user objects, and AM identities stored in DS.

      Create a mapping file named `sync.json` in the `/path/to/openidm/conf` directory. Use this file as a template.

9. Secure the connection to the DS server. This step assumes that you have set up the identity store, and exported the DS CA certificate from `identities.example.com` CA certificate from `identities.example.com` (as shown in Step 4 of "Secure Connections").

   Import the DS CA certificate into the IDM truststore:

```
keytool \
-importcert \
-alias identities-ca-cert \
-file /path/to/identities-ca-cert.pem \
-keystore /path/to/openidm/security/truststore \
-storepass:file /path/to/openidm/security/storepass
Owner: CN=Deployment key, O=ForgeRock.com
Issuer: CN=Deployment key, O=ForgeRock.com
...
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

10. (Optional) If you want to use the `PlatformForgottenUsername` or `PlatformResetPassword` trees, configure outbound email.

> **Note**
>
> After you have installed the Platform UI, you can configure email through the UI at `http://openidm.example.com:8080/admin`.

IDM is now configured. Move on to setting up the Platform UI.

Chapter 3

Deployment Two - Shared Identity Store

This deployment assumes that you are using the following data stores:

- An external DS instance as the AM configuration store.
- A separate external DS instance that is shared between AM and IDM as the identity store.

Note

The IDM End User UI is not supported in a platform deployment, as it does not support authentication through AM. You can use the Platform UIs with this deployment, or create your own UI that supports authentication through AM.

Set up Your Data Stores

Configuration Store

1. Set up a DS server as an external configuration data store, using the `am-config` setup profile.

   This example also adds the `am-cts` setup profile to the external configuration data store. In production, consider using separate DS servers for the AM CTS store. For more information about AM CTS, see the *AM Core Token Service Guide (CTS)*.

   For more information about DS setup profiles, see setup profiles in the *DS Installation Guide*.

   This command sets up the config store with the parameters listed in "Server Settings".

```
/path/to/opendj/setup \
--deploymentKeyPassword password \
--rootUserDN uid=admin \
--rootUserPassword str0ngAdm1nPa55word \
--monitorUserPassword str0ngMon1torPa55word \
--hostname config.example.com \
--adminConnectorPort 4444 \
--ldapPort 1389 \
--enableStartTls \
--ldapsPort 1636 \
--profile am-config \
--set am-config/amConfigAdminPassword:5up35tr0ng \
--profile am-cts \
--set am-cts/amCtsAdminPassword:5up35tr0ng \
--set am-cts/tokenExpirationPolicy:am-sessions-only \
--acceptLicense
```

Make a note of the generated deployment key. You will need it to export the server certificate later in this procedure. You can set the deployment key as a variable in this terminal window. For example:

```
export DEPLOYMENT_KEY=deployment-key
```

Note

For simplicity, this example uses a standalone directory server that does not replicate directory data. (No `--replicationPort` or `--bootstrapReplicationServer` options.)

In production deployments, replicate directory data for availability and resilience. For details, see the DS installation documentation.

2. Start the DS server:

```
/path/to/opendj/bin/start-ds
```

Identity Store

- Set up a DS server as a shared identity store, using the `am-identity-store` and `idm-repo` setup profiles.

  For more information about this step, see setup profiles in the *DS Installation Guide*.

  This command sets up the identity store with the parameters listed in "Server Settings":

```
/path/to/opendj/setup \
--deploymentKeyPassword password \
--rootUserDN uid=admin \
--rootUserPassword str0ngAdm1nPa55word \
--monitorUserPassword str0ngMon1torPa55word \
--hostname identities.example.com \
--adminConnectorPort 4444 \
--ldapPort 1389 \
--enableStartTls \
--ldapsPort 1636 \
--profile am-identity-store \
--set am-identity-store/amIdentityStoreAdminPassword:5up35tr0ng \
--profile idm-repo \
--set idm-repo/domain:forgerock.io \
--acceptLicense
```

Note

For simplicity, this example uses standalone directory servers that do not replicate directory data. (No `--replicationPort` or `--bootstrapReplicationServer` options.)

In production deployments, replicate directory data for availability and resilience. For details, see the DS installation documentation.

Make a note of the generated deployment key. You will need it to export the server certificate later in this procedure. You can set the deployment key as a variable in this terminal window. For example:

```
export DEPLOYMENT_KEY=deployment-key
```

Start the DS server:

```
/path/to/opendj/bin/start-ds
```

Set Up a Container

- Install a Java container to deploy AM.

  This guide assumes that you are using Apache Tomcat.

  For instructions on setting up Tomcat, see Preparing Apache Tomcat in the *AM Installation Guide*.

Secure Connections

Important

From DS 7 onwards, you *must* secure connections to DS servers.

1. Create a new directory that will house a dedicated truststore for AM:

   ```
   mkdir -p /path/to/openam-security/
   ```

2. *On each DS server,* export the DS server certificate.

   You must run these commands in the same terminal window where you set the `DEPLOYMENT_KEY` variable.

   On `config.example.com`:

   ```
   /path/to/opendj/bin/dskeymgr export-ca-cert \
   --deploymentKey $DEPLOYMENT_KEY \
   --deploymentKeyPassword password \
   --outputFile config-ca-cert.pem
   ```

   On `identities.example.com`:

   ```
   /path/to/opendj/bin/dskeymgr export-ca-cert \
   --deploymentKey $DEPLOYMENT_KEY \
   --deploymentKeyPassword password \
   --outputFile identities-ca-cert.pem
   ```

3. Import each DS server certificate into the dedicated AM truststore. If you are not testing this example on a single host, you might need to copy each certificate file onto the AM host machine first:

```
keytool \
-importcert \
-trustcacerts \
-alias config-ca-cert \
-file /path/to/config-ca-cert.pem \
-keystore /path/to/openam-security/truststore \
-storepass changeit \
-storetype JKS
Owner: CN=Deployment key, O=ForgeRock.com
Issuer: CN=Deployment key, O=ForgeRock.com
...
Trust this certificate? [no]:  yes
Certificate was added to keystore

keytool \
-importcert \
-trustcacerts \
-alias identities-ca-cert \
-file /path/to/identities-ca-cert.pem \
-keystore /path/to/openam-security/truststore \
-storepass changeit \
-storetype JKS
Owner: CN=Deployment key, O=ForgeRock.com
Issuer: CN=Deployment key, O=ForgeRock.com
...
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

4. (Optional) List the certificates in the new truststore and verify that the two certificates you added are there:

```
keytool \
-list \
-keystore /path/to/openam-security/truststore \
-storepass changeit
```

5. Point Apache Tomcat to the path of the new truststore so that AM can access it.

   Append the truststore settings to the `CATALINA_OPTS` variable in the `$CATALINA_BASE/bin/setenv.sh` file.

   For example:

```
CATALINA_OPTS="-Djavax.net.ssl.trustStore=/path/to/openam-security/truststore\
-Djavax.net.ssl.trustStorePassword=changeit\
-Djavax.net.ssl.trustStoreType=jks"
```

   Refer to your specific container's documentation for information on configuring truststores.

6. (Optional) At this stage, it is worth checking that you can use the dedicated AM connect to each DS server:

   On `config.example.com`:

```
/path/to/opendj/bin/ldapsearch \
--hostname config.example.com \
--port 1636 \
--useSsl \
--useJavaTrustStore /path/to/openam-security/truststore \
--trustStorePassword changeit \
--bindDn uid=am-config,ou=admins,ou=am-config \
--bindPassword 5up35tr0ng \
--baseDn ou=am-config \
"(&)" \
1.1
dn: ou=am-config

dn: ou=admins,ou=am-config

dn: uid=am-config,ou=admins,ou=am-config
```

On `identities.example.com`:

```
/path/to/opendj/bin/ldapsearch \
--hostname identities.example.com \
--port 1636 \
--useSsl \
--useJavaTrustStore /path/to/openam-security/truststore \
--trustStorePassword changeit \
--bindDn uid=am-identity-bind-account,ou=admins,ou=identities \
--bindPassword 5up35tr0ng \
--baseDn ou=identities \
"(&)" \
1.1
dn: ou=identities

dn: ou=people,ou=identities

dn: ou=groups,ou=identities

dn: ou=admins,ou=identities

dn: uid=am-identity-bind-account,ou=admins,ou=identities
```

Configure AM

When your external data stores are configured, follow these procedures to configure AM with the ForgeRock Identity Platform:

- "Install AM"
- "Adjust the Identity Store Configuration for IDM"
- "Configure OAuth Clients"
- "Configure an OAuth 2.0 Provider Service"
- "Configure an IDM Provisioning Service"

- "Enable CORS Support"
- "Configure Authentication Trees"
- "Map Authentication Trees"
- "Enable the Password Update Tree (optional)"

Install AM

1. Follow the instructions in the *AM Installation Guide* to download AM. Make sure you download the `.zip` file, not just the `.war` file.
2. Follow the instructions in the *AM Installation Guide* to prepare your environment, and prepare a web application container.

   Use Apache Tomcat as the application container, listening on port `8081`. This non-default port requires that you update Tomcat's `conf/server.xml` file. Instead of the default line, `<Connector port="8080" protocol="HTTP/1.1"`, use:

   ```
   <!-- Use 8081 to avoid a clash with IDM on 8080. -->
   <Connector port="8081" protocol="HTTP/1.1"
   ```
3. Copy the AM `.war` file to deploy in Apache Tomcat as `am.war`:

   ```
   cp AM-7.1.0.war /path/to/tomcat/webapps/am.war
   ```
4. Start Tomcat if it is not already running.
5. Navigate to the deployed AM application; for example, http://am.example.com:8081/am/.
6. Select Create New Configuration to create a custom configuration.
7. Accept the license agreement, and click Continue.
8. Set a password for the default user, `amAdmin`.

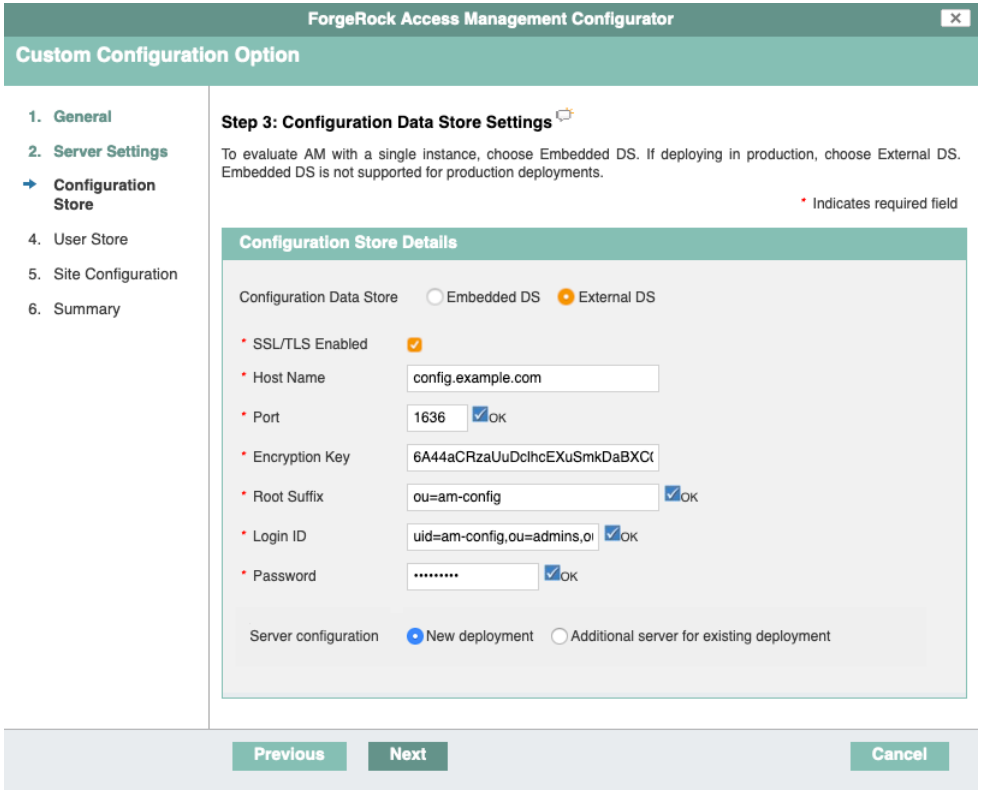
   This guide assumes that the `amAdmin` password is `Passw0rd`.
9. On the Server Settings screen, enter your AM server settings; for example:
   - Server URL: `http://am.example.com:8081`
   - Cookie Domain: `example.com`
   - Platform Locale: `en_US`
   - Configuration Directory: `/path/to/am`
10. On the Configuration Data Store Settings screen, select External DS, and enter the details for the DS instance that you set up as a configuration store.

This list reflects the DS configuration store installed with the listed "Server Settings".

- SSL/TLS: `Enabled`

  TLS is required in a production deployment.

- Host Name: `config.example.com`
- Port: `1636`
- Encryption Key: (generated encryption key)
- Root Suffix: `ou=am-config`
- Login ID: `uid=am-config,ou=admins,ou=am-config`
- Password: `5up35tr0ng`
- Server configuration: `New deployment`



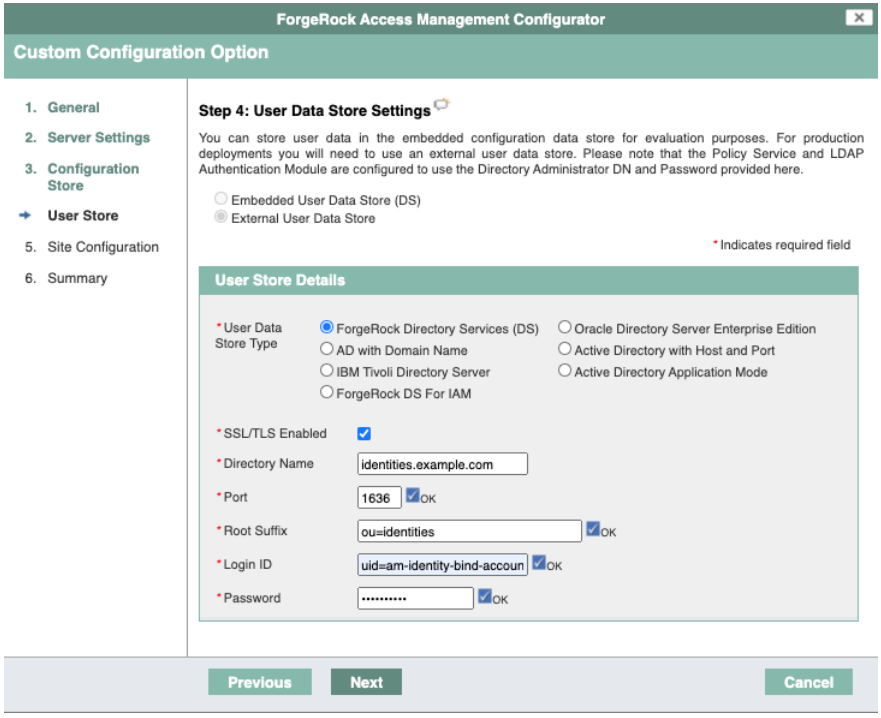
11. On the User Data Store Settings screen, select External User Data Store, and enter the details for the DS instance that you set up as an identity store.

    This list reflects the DS identity store installed with the listed "Server Settings".

    - User Data Store Type: `ForgeRock Directory Services (DS)`
    - SSL/TLS: `Enabled`

      TLS is required in a production deployment.

    - Host Name: `identities.example.com`
    - Port: `1636`
    - Root Suffix: `ou=identities`
    - Login ID: `uid=am-identity-bind-account,ou=admins,ou=identities`
    - Password: `5up35tr0ng`



12. On the Site Configuration screen, select No, then click Next.

13. Review the Configurator Summary Details, then click Create Configuration.

Important

This sample deployment uses only the AM top level realm for simplicity. In production, use AM subrealms.

This sample deployment may show limitations inherent in not using additional realms as suggested. For example, to create an external application store, you must first set up a separate subrealm, and then create the application store there.

Adjust the Identity Store Configuration for IDM

1. If you're not currently logged in to the AM console as the `amAdmin` user, log in.
2. In the Top Level Realm, select Identity Stores then click OpenDJ.
3. On the Server Settings tab, set LDAPv3 Plug-in Search Scope to `SCOPE_ONE`, then click Save Changes.
4. On the User Configuration tab, set LDAP Users Search Attribute to `fr-idm-uuid`, then click Save Changes.
5. On the Authentication Configuration tab, check that the Authentication Naming Attribute is set to `uid`, then click Save Changes.

Configure OAuth Clients

This procedure configures *four* OAuth 2.0 clients. All of these clients are configured in the Top Level Realm. If you are using sub-realms, you must configure the clients for *each* realm.

1. If you're not currently logged in to the AM console as the `amAdmin` user, log in.
2. Configure an `idm-resource-server` client to introspect access tokens.

   Note

   This is not a *real* OAuth 2.0 client, in the traditional sense. Rather, it is an OAuth 2.0 Resource Server account, used exclusively for token introspection. As such, you do not need to specify redirect URIs or grant types.

   a. Select Applications > OAuth 2.0 > Clients, and click Add Client.

   b. Enter the following details:

   - Client ID: `idm-resource-server`
   - Client secret: `password`

The value of this field must match the `clientSecret` that you will set in the `rsFilter` module in the IDM authentication configuration (`/path/to/openidm/conf/authentication.json`) during your IDM setup.

- Scopes: `am-introspect-all-tokens`

  This scope gives the client the right to introspect tokens issued to other clients. If you expect to accept access tokens issued from sub-realms, you must also include the scope `am-introspect-all-tokens-any-realm`.

c. Click Create.

3. Configure an `idm-provisioning` client to make calls to IDM:

   a. Select Applications > OAuth 2.0 > Clients, and click Add Client.

   b. Enter the following details:

   - Client ID: `idm-provisioning`
   - Client secret: `openidm`
   - Scopes: `fr:idm:*`

   c. Click Create.

   d. On the Advanced tab:

   - Response Types: Check that `token` is present (it should be there by default).
   - Grant Types: Remove `Authorization Code` and add `Client Credentials`.

   e. Click Save Changes.

4. Configure an `idm-admin-ui` client that will be used by the Platform Admin UI:

   a. In the Top Level Realm, select Applications > OAuth 2.0 > Clients, and click Add Client.

   b. Enter the following details:

   - Client ID: `idm-admin-ui`
   - Client Secret: (no client secret is required)
   - Redirection URIs: `http://openidm.example.com:8080/platform/appAuthHelperRedirect.html` `http://openidm.example.com:8080/platform/sessionCheck.html` `http://openidm.example.com:8080/admin/appAuthHelperRedirect.html` `http://openidm.example.com:8080/admin/sessionCheck.html` `http://admin.example.com:8082/appAuthHelperRedirect.html` `http://admin.example.com:8082/sessionCheck.html` `http://am.example.com:8081/platform/appAuthHelperRedirect.html` `http://am.example.com:8081/platform/sessionCheck.html`

- Scopes: `openid` `fr:idm:*`

Note

At a minimum, the scopes that you set here must include the scopes that you will set in the `rsFilter` authentication configuration (`/path/to/openidm/conf/authentication.json`) during your IDM setup.

c. Click Create.

d. On the Core tab:

- Client type: Select `Public`.

Click Save Changes.

e. On the Advanced tab:

- Grant Types: Add `Implicit`.
- Token Endpoint Authentication Method: Select `none`.
- Implied consent: Enable.

Click Save Changes.

5. Configure an `end-user-ui` client that will be used by the Platform End User UI:

a. In the Top Level Realm, select Applications > OAuth 2.0 > Clients, and click Add Client.

b. Enter the following details:

- Client ID: `end-user-ui`
- Client Secret: (no client secret is required)
- Redirection URIs: `http://enduser.example.com:8888/appAuthHelperRedirect.html` `http://enduser.example.com:8888/sessionCheck.html` `http://am.example.com:8081/enduser/appAuthHelperRedirect.html` `http://am.example.com:8081/enduser/sessionCheck.html`
- Scopes: `openid` `fr:idm:*`

Note

At a minimum, the scopes that you set here must include the scopes that you will set in the `rsFilter` authentication configuration (`/path/to/openidm/conf/authentication.json`) during your IDM setup.

c. Click Create.

d. On the Core tab:

- Client type: Select `Public`.

Click Save Changes.

e. On the Advanced tab:

- Grant Types: Add `Implicit`.
- Token Endpoint Authentication Method: Select `none`.
- Implied Consent: Enable.

Click Save Changes.

Configure an OAuth 2.0 Provider Service

1. If you're not currently logged in to the AM console as the `amAdmin` user, log in.
2. In the Top Level Realm, select Services, and click Add a Service.
3. Under Choose a service type, select OAuth2 Provider.
4. For Client Registration Scope Whitelist, add the following scopes:
   - `fr:idm:*`
   - `am-introspect-all-tokens`
   - `openid`
5. Click Create.
6. On the Advanced tab, make sure that Response Type Plugins includes the following values:

```
id_token|org.forgerock.openidconnect.IdTokenResponseTypeHandler
```

```
code|org.forgerock.oauth2.core.AuthorizationCodeResponseTypeHandler
```

7. Click Save Changes.
8. On the Consent tab, enable Allow Clients to Skip Consent.
9. Click Save Changes.

Configure an IDM Provisioning Service

1. If you're not currently logged in to the AM console as the `amAdmin` user, log in.

2. From the top menu, select Configure > Global Services > IDM Provisioning.
3. Set the following fields:
   - Enabled
   - Deployment URL: `http://openidm.example.com:8080`
   - Deployment Path: `openidm`
   - IDM Provisioning Client: `idm-provisioning`
4. Click Save Changes.

Configure a Validation Service

The Platform UIs need this validation service allow listing for `goto` redirection.

1. If you're not currently logged in to the AM console as the `amAdmin` user, log in.
2. In the Top Level Realm, select Services, and click Add a Service.
3. Under Choose a service type, select Validation Service.
4. For Valid goto URL Resources, add the URLs for the Platform UI:

   ```
   http://admin.example.com:8082/*
   http://admin.example.com:8082/*?*
   http://login.example.com:8083/*
   http://login.example.com:8083/*?*
   http://enduser.example.com:8888/*
   http://enduser.example.com:8888/*?*
   ```

5. Click Create.

Enable CORS Support

Cross-origin resource sharing (CORS) lets user agents make requests across domains.

1. If you're not currently logged in to the AM console as the `amAdmin` user, log in.
2. From the top menu, select Configure > Global Services > CORS Service.
3. On the Secondary Configurations tab, click Add a Secondary Configuration.
4. On the New Configuration screen, enter the following values:
   - Name: `Cors Configuration`

- Accepted Origins:

  ```
  http://login.example.com:8083
  http://admin.example.com:8082
  http://enduser.example.com:8888
  http://openidm.example.com:8080
  https://openidm.example.com:8443
  ```

  List only the origins that will be hosting OAuth 2.0 clients (such as the `platform-enduser` and IDM Admin UIs).

- Accepted Methods:

  ```
  HEAD
  DELETE
  POST
  GET
  PUT
  PATCH
  ```

- Accepted Headers:

  ```
  authorization
  x-openidm-username
  if-none-match
  x-openidm-nosession
  x-openidm-password
  accept-api-version
  x-requested-with
  content-type
  if-match
  cache-control
  user-agent
  ```

- Exposed Headers: `WWW-Authenticate`

5. Click Create.
6. On the Cors Configuration screen, set the following values:
   - Enable the CORS filter: Enable
   - Max Age: `600`
   - Allow Credentials: Enable
7. Click Save Changes.

Configure Authentication Trees

The platform deployment relies on three authentication trees to enable authentication through AM. When you extract the AM `.zip` file, you will get a `sample-trees-7.1.0.zip` file that contains a number of sample authentication trees, in JSON files. Use the Amster command-line utility to import the platform authentication trees into your AM configuration:

1. Extract the `sample-trees-7.1.0.zip` file and list the sample trees in the `/path/to/openam-samples/root/AuthTree` directory:

```
ls /path/to/openam-samples/root/AuthTree
Agent.json                          PlatformForgottenUsername.json
Example.json                        PlatformLogin.json
Facebook-ProvisionIDMAccount.json   PlatformProgressiveProfile.json
Google-AnonymousUser.json           PlatformRegistration.json
Google-DynamicAccountCreation.json  PlatformResetPassword.json
HmacOneTimePassword.json            PlatformUpdatePassword.json
PersistentCookie.json               RetryLimit.json
```

2. Download and install Amster.

3. Start Amster, then connect to your AM instance:

```
./amster
Amster OpenAM Shell (7.1.0 build @build.number@, JVM: version)
Type ':help' or ':h' for help.
---------------------------------------------------------------------------
am> connect --interactive http://am.example.com:8081/am
Sign in
User Name: amAdmin
Password: ********
amster am.example.com:8081>
```

4. Import the sample authentication trees and nodes:

```
amster am.example.com:8081> import-config --path /path/to/openam-samples/root
Importing directory /path/to/openam-samples/root/AcceptTermsAndConditions
Imported /path/to/openam-samples/root/AcceptTermsAndConditions/b4a0e915-
c15d-4b83-9c9d-18347d645976.json
...
Import completed successfully
```

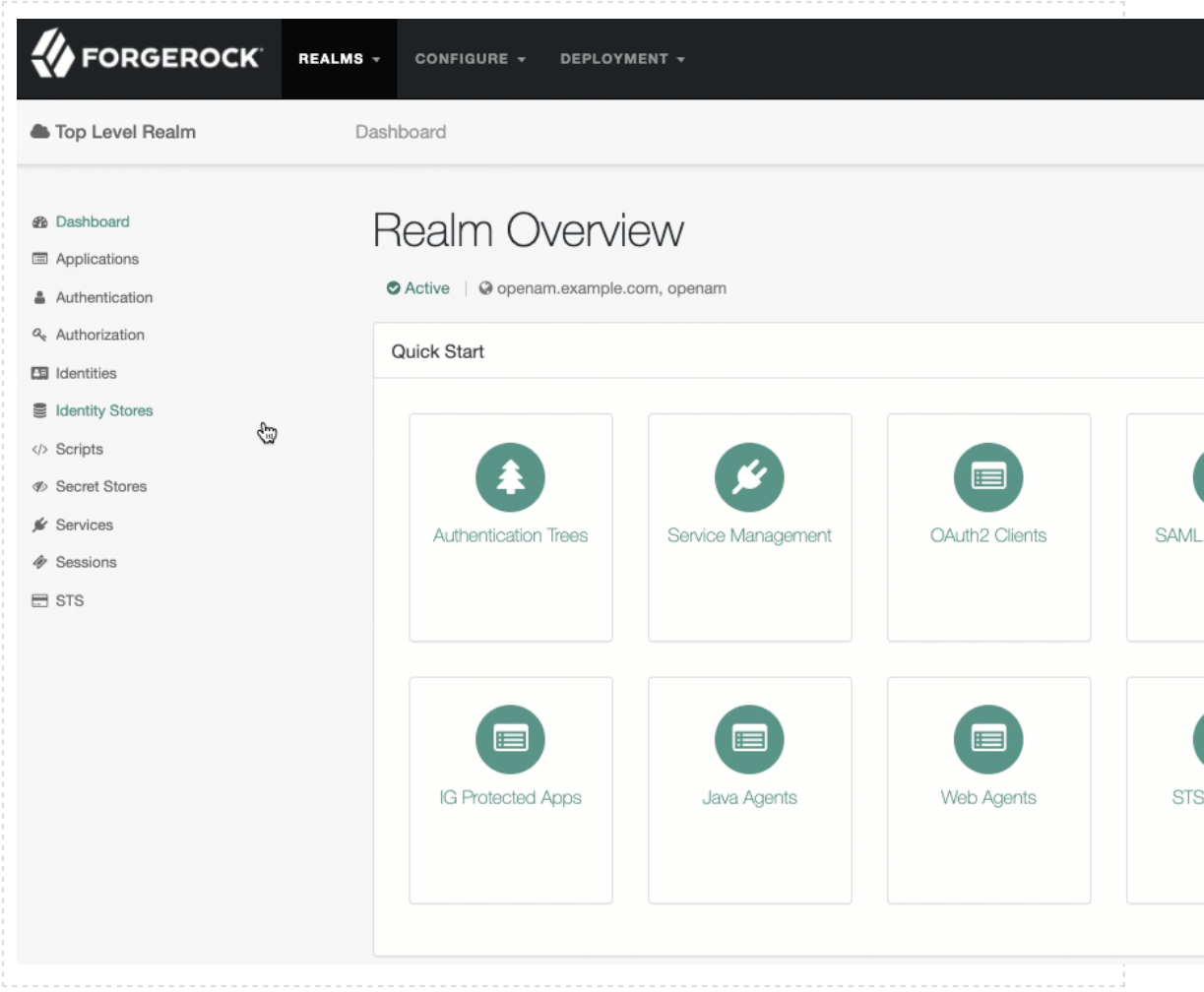
5. If you're not currently logged in to the AM console as the `amAdmin` user, log in.

6. Configure the `PlatformRegistration` tree:

   a. In the Top Level Realm, select Authentication > Trees, and click on PlatformRegistration.

   b. On the `PlatformRegistration` tree, add a `Success URL` node between `Increment Login Count` and `Success`, and set its value to `http://enduser.example.com:8888`.

      + *Show Me*



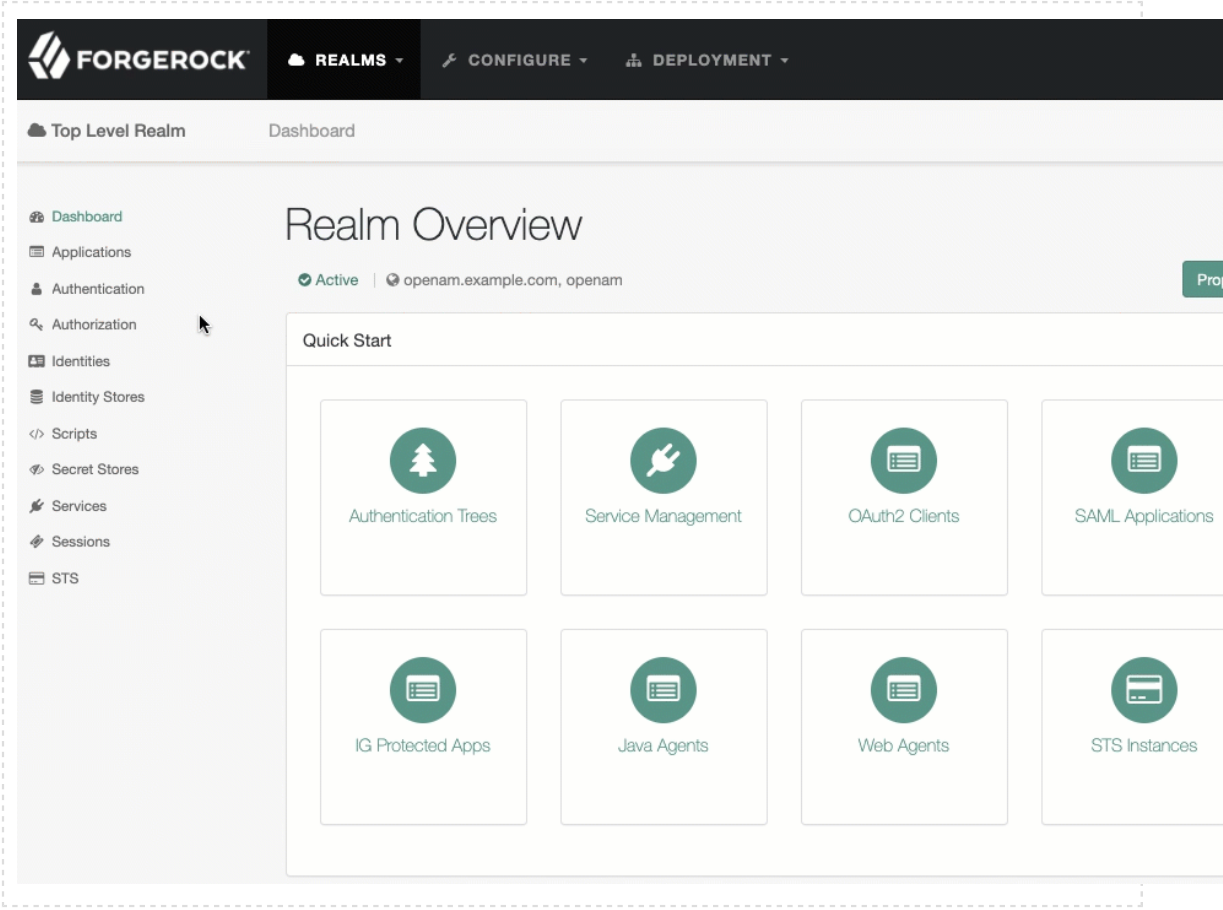
c. Click Save.

7. Configure the `PlatformLogin` tree:

   a. In the Top Level Realm, select Authentication > Trees, and click on PlatformLogin.

   b. On the `PlatformLogin` tree, add a `Success URL` node between `Inner Tree Evaluator` and `Success`, and set its value to `http://enduser.example.com:8888`.

      + *Show Me*



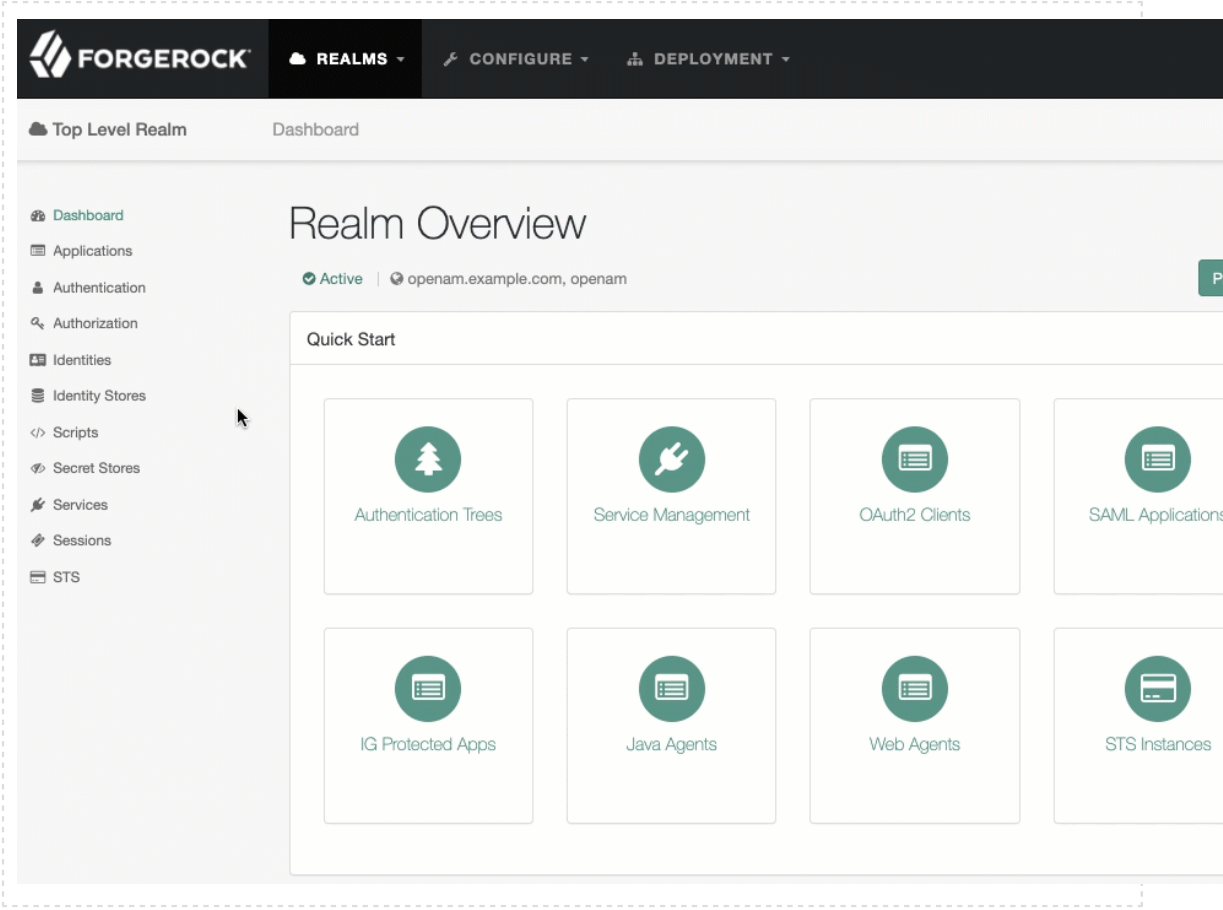
c. Click Save.

8. Configure the `PlatformResetPassword` tree:

   a. In the Top Level Realm, select Authentication > Trees, and click on PlatformResetPassword.

   b. On the `PlatformResetPassword` tree, add a `Success URL` node between `Patch Object` and `Success`, and set its value to `http://enduser.example.com:8888`.

      + *Show Me*



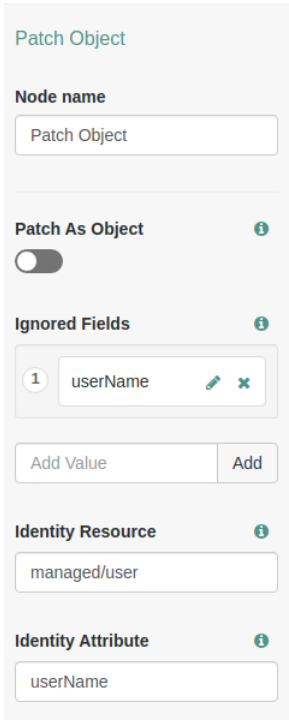
c. Click Save.

9. Configure the `PlatformUpdatePassword` tree:

   a. In the Top Level Realm, select Authentication > Trees, and click on PlatformUpdatePassword.

   b. On the `PlatformUpdatePassword` tree, select the `Patch Object` node, and set its `Patch As Object` value to `false`.

      + *Show Me*



c. Click Save.

10. For the authentication trees that require email, set the External Login Page URL.

    In the Top Level Realm, select Authentication > Settings, and click the General tab. Set External Login Page URL to `http://login.example.com:8083`, then click Save Changes.

Map Authentication Trees

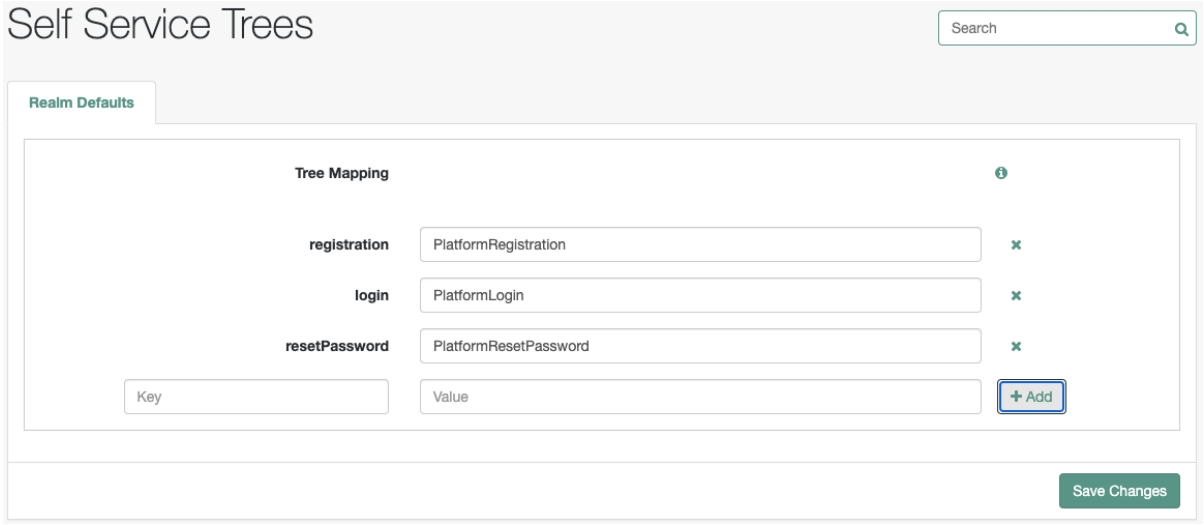
Map the platform trees to the corresponding Self-Service endpoints. For more information about this step, see "Configure Self-Service Trees Endpoints" in the *Platform Self-Service Guide*.

1. From the top menu, select Configure > Global Services > Self Service Trees.

2. Add the following Tree Mappings:

Key	Value
registration	PlatformRegistration
login	PlatformLogin

Key	Value
resetPassword	PlatformResetPassword



3. Click Save Changes.

Enable the Password Update Tree (optional)

To let end users update their own passwords (using the `PlatformUpdatePassword` tree), add the `idm-provisioning` service as an authorized client of the `OAuth2 Provider` service.

You cannot use the UI for this step—you must update the OAuth 2.0 configuration over REST:

1. Use the following request to get the token ID:

```
curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: amAdmin" \
--header "X-OpenAM-Password: Passw0rd" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'http://am.example.com:8081/am/json/realms/root/authenticate'
{
  "tokenId": "tokenId",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

2. Make a GET request to the `oauth-oidc` endpoint to return the current OAuth configuration:

   + *Example Curl Request*

```
curl \
-X GET \
-H 'Accept-API-Version: protocol=1.0,resource=1.0' \
-H 'X-Requested-With: XMLHttpRequest' \
-H 'Origin: http://am.example.com:8081' \
-H 'Referer: http://am.example.com:8081/am/ui-admin/' \
-H 'Cookie: amlbcookie=01; iPlanetDirectoryPro=tokenId' \
"http://am.example.com:8081/am/json/realms/root/realm-config/services/oauth-oidc"
{
  "coreOAuth2Config": {
    "refreshTokenLifetime": 604800,
    "accessTokenLifetime": 3600,
    "usePolicyEngineForScope": false,
    "codeLifetime": 120,
    "issueRefreshTokenOnRefreshedToken": true,
    "macaroonTokensEnabled": false,
    "issueRefreshToken": true,
    "accessTokenModificationScript": "d22f9a0c-426a-4466-b95e-d0f125b0d5fa",
    "statelessTokensEnabled": false
  },
  "coreOIDCConfig": {
    "supportedIDTokenEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "jwtTokenLifetime": 3600,
    "supportedClaims": [],
    "supportedIDTokenEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedIDTokenSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
```

```
    "oidcClaimsScript": "36863ffb-40ec-48b9-94b1-9a99f71cc3b5"
  },
  "advancedOAuth2Config": {
    "supportedScopes": [
      "openid",
      "fr:idm:*",
      "am-introspect-all-tokens"
    ],
    "tlsCertificateRevocationCheckingEnabled": false,
    "codeVerifierEnforced": "false",
    "tokenSigningAlgorithm": "HS256",
    "authenticationAttributes": [
      "uid"
    ],
    "passwordGrantAuthService": "[Empty]",
    "defaultScopes": [],
    "tlsClientCertificateHeaderFormat": "URLENCODED_PEM",
    "scopeImplementationClass": "org.forgerock.openam.oauth2.OpenAMScopeValidator",
    "responseTypeClasses": [
      "code|org.forgerock.oauth2.core.AuthorizationCodeResponseTypeHandler",
      "id_token|org.forgerock.openidconnect.IdTokenResponseTypeHandler",
      "device_code|org.forgerock.oauth2.core.TokenResponseTypeHandler",
      "token|org.forgerock.oauth2.core.TokenResponseTypeHandler"
    ],
    "tlsCertificateBoundAccessTokensEnabled": true,
    "hashSalt": "changeme",
    "moduleMessageEnabledInPasswordGrant": false,
    "tokenEncryptionEnabled": false,
    "tokenCompressionEnabled": false,
    "grantTypes": [
      "implicit",
      "urn:ietf:params:oauth:grant-type:saml2-bearer",
      "refresh_token",
      "password",
      "client_credentials",
      "urn:ietf:params:oauth:grant-type:device_code",
      "authorization_code",
      "urn:openid:params:grant-type:ciba",
      "urn:ietf:params:oauth:grant-type:uma-ticket",
      "urn:ietf:params:oauth:grant-type:jwt-bearer"
    ],
    "displayNameAttribute": "cn",
    "macaroonTokenFormat": "V2",
    "supportedSubjectTypes": [
      "public"
    ]
  },
  "advancedOIDCConfig": {
    "storeOpsTokens": true,
    "defaultACR": [],
    "supportedRequestParameterEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "claimsParameterSupported": false,
```

```
    "amrMappings": {},
    "supportedUserInfoEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "authorisedIdmDelegationClients": [],
    "alwaysAddClaimsToToken": false,
    "supportedUserInfoSigningAlgorithms": [
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512"
    ],
    "supportedRequestParameterEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "ECDH-ES+A128KW",
      "RSA-OAEP",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenIntrospectionResponseEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "supportedTokenIntrospectionResponseSigningAlgorithms": [
      "PS384",
      "RS384",
      "EdDSA",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "authorisedOpenIdConnectSSOClients": [],
    "idTokenInfoClientAuthenticationEnabled": true,
    "supportedRequestParameterSigningAlgorithms": [
      "PS384",
```

```
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "supportedUserInfoEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenIntrospectionResponseEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenEndpointAuthenticationSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "loaMapping": {}
  },
  "clientDynamicRegistrationConfig": {
    "dynamicClientRegistrationSoftwareStatementRequired": false,
    "dynamicClientRegistrationScope": "dynamic_client_registration",
    "requiredSoftwareStatementAttestedAttributes": [
      "redirect_uris"
    ],
    "generateRegistrationAccessTokens": true,
```

```
    "allowDynamicRegistration": false
  },
  "cibaConfig": {
    "supportedCibaSigningAlgorithms": [
      "ES256",
      "PS256"
    ],
    "cibaAuthReqIdLifetime": 600,
    "cibaMinimumPollingInterval": 2
  },
  "consent": {
    "enableRemoteConsent": false,
    "supportedRcsRequestSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "supportedRcsResponseSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "clientsCanSkipConsent": true,
    "supportedRcsRequestEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedRcsResponseEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
```

```
      "A256CBC-HS512"
    ],
    "supportedRcsRequestEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "remoteConsentServiceId": "[Empty]",
    "supportedRcsResponseEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "ECDH-ES+A128KW",
      "RSA-OAEP",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ]
  },
  "deviceCodeConfig": {
    "devicePollInterval": 5,
    "deviceCodeLifetime": 300
  },
  "_id": "",
  "_type": {
    "_id": "oauth-oidc",
    "name": "OAuth2 Provider",
    "collection": false
  }
}
```

3. Make a PUT request to the `oauth-oidc` endpoint to submit the JSON payload returned in the previous step, replacing the value of the `authorisedIdmDelegationClients` property under `advancedOIDCConfig` with `idm-provisioning`:

```
"authorisedIdmDelegationClients": [ "idm-provisioning"],
```

+ *Example Curl Request*

```
curl \
--request PUT \
-H 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:70.0) Gecko/20100101
 Firefox/70.0' \
-H 'Accept: application/json, text/javascript, */*; q=0.01' \
-H 'Accept-Language: en-US' \
--compressed \
-H 'Content-Type: application/json' \
-H 'Accept-API-Version: protocol=1.0,resource=1.0' \
-H 'X-Requested-With: XMLHttpRequest' \
-H 'Cache-Control: no-cache' \
```

```
-H 'Origin: http://am.example.com:8081' \
-H 'DNT: 1' \
-H 'Connection: keep-alive' \
-H 'Referer: http://am.example.com:8081/am/ui-admin/' \
-H 'Cookie: amlbcookie=01; iPlanetDirectoryPro=tokenId' \
--data '{
  "coreOAuth2Config": {
    "refreshTokenLifetime": 604800,
    "accessTokenLifetime": 3600,
    "usePolicyEngineForScope": false,
    "codeLifetime": 120,
    "issueRefreshTokenOnRefreshedToken": true,
    "macaroonTokensEnabled": false,
    "issueRefreshToken": true,
    "accessTokenModificationScript": "d22f9a0c-426a-4466-b95e-d0f125b0d5fa",
    "statelessTokensEnabled": false
  },
  "coreOIDCConfig": {
    "supportedIDTokenEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "jwtTokenLifetime": 3600,
    "supportedClaims": [],
    "supportedIDTokenEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedIDTokenSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "oidcClaimsScript": "36863ffb-40ec-48b9-94b1-9a99f71cc3b5"
  },
  "advancedOAuth2Config": {
    "supportedScopes": [
      "openid",
```

```
      "fr:idm:*",
      "am-introspect-all-tokens"
    ],
    "tlsCertificateRevocationCheckingEnabled": false,
    "codeVerifierEnforced": "false",
    "tokenSigningAlgorithm": "HS256",
    "authenticationAttributes": [
      "uid"
    ],
    "passwordGrantAuthService": "[Empty]",
    "defaultScopes": [],
    "tlsClientCertificateHeaderFormat": "URLENCODED_PEM",
    "scopeImplementationClass": "org.forgerock.openam.oauth2.OpenAMScopeValidator",
    "responseTypeClasses": [
      "code|org.forgerock.oauth2.core.AuthorizationCodeResponseTypeHandler",
      "id_token|org.forgerock.openidconnect.IdTokenResponseTypeHandler",
      "device_code|org.forgerock.oauth2.core.TokenResponseTypeHandler",
      "token|org.forgerock.oauth2.core.TokenResponseTypeHandler"
    ],
    "tlsCertificateBoundAccessTokensEnabled": true,
    "hashSalt": "changeme",
    "moduleMessageEnabledInPasswordGrant": false,
    "tokenEncryptionEnabled": false,
    "tokenCompressionEnabled": false,
    "grantTypes": [
      "implicit",
      "urn:ietf:params:oauth:grant-type:saml2-bearer",
      "refresh_token",
      "password",
      "client_credentials",
      "urn:ietf:params:oauth:grant-type:device_code",
      "authorization_code",
      "urn:openid:params:grant-type:ciba",
      "urn:ietf:params:oauth:grant-type:uma-ticket",
      "urn:ietf:params:oauth:grant-type:jwt-bearer"
    ],
    "displayNameAttribute": "cn",
    "macaroonTokenFormat": "V2",
    "supportedSubjectTypes": [
      "public"
    ]
  },
  "advancedOIDCConfig": {
    "storeOpsTokens": true,
    "authorisedIdmDelegationClients": ["idm-provisioning"],
    "defaultACR": [],
    "supportedRequestParameterEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "claimsParameterSupported": false,
    "amrMappings": {},
    "supportedUserInfoEncryptionEnc": [
      "A256GCM",
      "A192GCM",
```

```
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "alwaysAddClaimsToToken": false,
    "supportedUserInfoSigningAlgorithms": [
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512"
    ],
    "supportedRequestParameterEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "ECDH-ES+A128KW",
      "RSA-OAEP",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenIntrospectionResponseEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "supportedTokenIntrospectionResponseSigningAlgorithms": [
      "PS384",
      "RS384",
      "EdDSA",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "authorisedOpenIdConnectSSOClients": [],
    "idTokenInfoClientAuthenticationEnabled": true,
    "supportedRequestParameterSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
```

```
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "supportedUserInfoEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenIntrospectionResponseEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenEndpointAuthenticationSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "loaMapping": {}
  },
  "clientDynamicRegistrationConfig": {
    "dynamicClientRegistrationSoftwareStatementRequired": false,
    "dynamicClientRegistrationScope": "dynamic_client_registration",
    "requiredSoftwareStatementAttestedAttributes": [
      "redirect_uris"
    ],
    "generateRegistrationAccessTokens": true,
    "allowDynamicRegistration": false
  },
  "cibaConfig": {
    "supportedCibaSigningAlgorithms": [
      "ES256",
```

```
      "PS256"
    ],
    "cibaAuthReqIdLifetime": 600,
    "cibaMinimumPollingInterval": 2
  },
  "consent": {
    "enableRemoteConsent": false,
    "supportedRcsRequestSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "supportedRcsResponseSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "clientsCanSkipConsent": true,
    "supportedRcsRequestEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedRcsResponseEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "supportedRcsRequestEncryptionMethods": [
      "A256GCM",
      "A192GCM",
```

```
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "remoteConsentServiceId": "[Empty]",
    "supportedRcsResponseEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "ECDH-ES+A128KW",
      "RSA-OAEP",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ]
  },
  "deviceCodeConfig": {
    "devicePollInterval": 5,
    "deviceCodeLifetime": 300
  },
  "_id": "",
  "_type": {
    "_id": "oauth-oidc",
    "name": "OAuth2 Provider",
    "collection": false
  }
}' \
"http://am.example.com:8081/am/json/realms/root/realm-config/services/oauth-oidc"
{
  "coreOAuth2Config": {
    "refreshTokenLifetime": 604800,
    "accessTokenLifetime": 3600,
    "usePolicyEngineForScope": false,
    "codeLifetime": 120,
    "issueRefreshTokenOnRefreshedToken": true,
    "macaroonTokensEnabled": false,
    "issueRefreshToken": true,
    "accessTokenModificationScript": "d22f9a0c-426a-4466-b95e-d0f125b0d5fa",
    "statelessTokensEnabled": false
  },
  "coreOIDCConfig": {
    "supportedIDTokenEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "jwtTokenLifetime": 3600,
    "supportedClaims": [],
    "supportedIDTokenEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
```

```
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedIDTokenSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "oidcClaimsScript": "36863ffb-40ec-48b9-94b1-9a99f71cc3b5"
  },
  "advancedOAuth2Config": {
    "supportedScopes": [
      "openid",
      "fr:idm:*",
      "am-introspect-all-tokens"
    ],
    "tlsCertificateRevocationCheckingEnabled": false,
    "codeVerifierEnforced": "false",
    "tokenSigningAlgorithm": "HS256",
    "authenticationAttributes": [
      "uid"
    ],
    "passwordGrantAuthService": "[Empty]",
    "defaultScopes": [],
    "tlsClientCertificateHeaderFormat": "URLENCODED_PEM",
    "scopeImplementationClass": "org.forgerock.openam.oauth2.OpenAMScopeValidator",
    "responseTypeClasses": [
      "code|org.forgerock.oauth2.core.AuthorizationCodeResponseTypeHandler",
      "id_token|org.forgerock.openidconnect.IdTokenResponseTypeHandler",
      "device_code|org.forgerock.oauth2.core.TokenResponseTypeHandler",
      "token|org.forgerock.oauth2.core.TokenResponseTypeHandler"
    ],
    "tlsCertificateBoundAccessTokensEnabled": true,
    "hashSalt": "changeme",
    "moduleMessageEnabledInPasswordGrant": false,
    "tokenEncryptionEnabled": false,
    "tokenCompressionEnabled": false,
    "grantTypes": [
      "implicit",
      "urn:ietf:params:oauth:grant-type:saml2-bearer",
      "refresh_token",
      "password",
      "client_credentials",
      "urn:ietf:params:oauth:grant-type:device_code",
      "authorization_code",
      "urn:openid:params:grant-type:ciba",
```

```
      "urn:ietf:params:oauth:grant-type:uma-ticket",
      "urn:ietf:params:oauth:grant-type:jwt-bearer"
    ],
    "displayNameAttribute": "cn",
    "macaroonTokenFormat": "V2",
    "supportedSubjectTypes": [
      "public"
    ]
  },
  "advancedOIDCConfig": {
    "storeOpsTokens": true,
    "defaultACR": [],
    "supportedRequestParameterEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "claimsParameterSupported": false,
    "amrMappings": {},
    "supportedUserInfoEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
   "authorisedIdmDelegationClients": ["idm-provisioning"],
   "alwaysAddClaimsToToken": false,
    "supportedUserInfoSigningAlgorithms": [
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512"
    ],
    "supportedRequestParameterEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "ECDH-ES+A128KW",
      "RSA-OAEP",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenIntrospectionResponseEncryptionEnc": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
```

```
      "A256CBC-HS512"
    ],
    "supportedTokenIntrospectionResponseSigningAlgorithms": [
      "PS384",
      "RS384",
      "EdDSA",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "authorisedOpenIdConnectSSOClients": [],
    "idTokenInfoClientAuthenticationEnabled": true,
    "supportedRequestParameterSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "supportedUserInfoEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenIntrospectionResponseEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedTokenEndpointAuthenticationSigningAlgorithms": [
      "PS384",
```

```
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "loaMapping": {}
  },
  "clientDynamicRegistrationConfig": {
    "dynamicClientRegistrationSoftwareStatementRequired": false,
    "dynamicClientRegistrationScope": "dynamic_client_registration",
    "requiredSoftwareStatementAttestedAttributes": [
      "redirect_uris"
    ],
    "generateRegistrationAccessTokens": true,
    "allowDynamicRegistration": false
  },
  "cibaConfig": {
    "supportedCibaSigningAlgorithms": [
      "ES256",
      "PS256"
    ],
    "cibaAuthReqIdLifetime": 600,
    "cibaMinimumPollingInterval": 2
  },
  "consent": {
    "enableRemoteConsent": false,
    "supportedRcsRequestSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
    ],
    "supportedRcsResponseSigningAlgorithms": [
      "PS384",
      "RS384",
      "ES384",
      "HS256",
      "HS512",
      "ES256",
      "RS256",
      "HS384",
      "ES512",
      "PS256",
      "PS512",
```

```
      "RS512"
    ],
    "clientsCanSkipConsent": true,
    "supportedRcsRequestEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "RSA-OAEP",
      "ECDH-ES+A128KW",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ],
    "supportedRcsResponseEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "supportedRcsRequestEncryptionMethods": [
      "A256GCM",
      "A192GCM",
      "A128GCM",
      "A128CBC-HS256",
      "A192CBC-HS384",
      "A256CBC-HS512"
    ],
    "remoteConsentServiceId": "[Empty]",
    "supportedRcsResponseEncryptionAlgorithms": [
      "ECDH-ES+A256KW",
      "ECDH-ES+A192KW",
      "ECDH-ES+A128KW",
      "RSA-OAEP",
      "RSA-OAEP-256",
      "A128KW",
      "A256KW",
      "ECDH-ES",
      "dir",
      "A192KW"
    ]
  },
  "deviceCodeConfig": {
    "devicePollInterval": 5,
    "deviceCodeLifetime": 300
  },
  "_id": "",
  "_type": {
    "_id": "oauth-oidc",
    "name": "OAuth2 Provider",
    "collection": false
  }
```

```
}
```

Set up IDM

This procedure sets up IDM with an external DS instance as the repository. The DS instance is shared with AM as the identity store. The procedure assumes that both IDM and the DS instance have been installed with the listed "Server Settings".

1. Follow the instructions in the *IDM Installation Guide* to install IDM.
2. Edit the `/path/to/openidm/resolver/boot.properties` file to set the hostname:

   ```
   openidm.host=openidm.example.com
   ```
3. Configure the shared IDM repository.

   This step assumes that you have set up the identity store, and exported the DS CA certificate from `identities.example.com` (as shown in Step 4 of "Secure Connections").

   a. Import the DS CA certificate into the IDM truststore:

   ```
   keytool \
   -importcert \
   -alias identities-ca-cert \
   -file /path/to/identities-ca-cert.pem \
   -keystore /path/to/openidm/security/truststore \
   -storepass:file /path/to/openidm/security/storepass
   Owner: CN=Deployment key, O=ForgeRock.com
   Issuer: CN=Deployment key, O=ForgeRock.com
   ...
   Trust this certificate? [no]:  yes
   Certificate was added to keystore
   ```

   b. Replace the default `conf/repo.ds.json` file with this file.

   Important

   Replace the values of `fileBasedTrustManagerFile` and `fileBasedTrustManagerPasswordFile` with the path to your IDM truststore and truststore password file.

   Check that the properties under `ldapConnectionFactories` match the DS server that you set up as the identity store.

   It is worth starting IDM at this point, to make sure that it can connect to the external DS repository. You must see `OpenIDM ready` in the output:

   ```
   /path/to/openidm/startup.sh
   -> OpenIDM version "7.1.0" build-info
   OpenIDM ready
   ```

4. Update the `user` object in your managed object configuration by making the following changes to `conf/managed.json`.

   All changes are made to the object `user` > `schema` > `properties`:

   ```
   {
       "objects": [
           {
               "name": "user",
               ...
               "schema": {
               "properties": {
                   ...
               }
               ...
           }
       ]
   }
   ```

   a. Remove encryption from the `password` property. Remove:

      ```
      "encryption" : {
          "purpose": "idm.password.encryption"
      }
      ```

      Password *encryption* is not supported in a shared identity store setup. Instead, the shared identity store supports only *hashed* passwords, a format that both AM and IDM can use. This differs from "*Deployment One - Separate Identity Stores*", where AM and IDM do not share the identity store, and so each service can store passwords in different formats.

   b. Update the `password` property to ensure that users update their passwords through the self-service APIs, not directly:

      ```
      "userEditable" : false
      ```

   c. Add a `cn` property to the `user` object:

```
{
    "objects": [
        {
            "name": "user",
            ...
            "schema": {
                "properties" : {
                    "_id" : {
                        ...
                    },
                    "cn": {
                        "title": "Common Name",
                        "description": "Common Name",
                        "type": "string",
                        "viewable": false,
                        "searchable": false,
                        "userEditable": false,
                        "scope": "private",
                        "isPersonal": true,
                        "isVirtual": true,
                        "onStore": {
                            "type": "text/javascript",
                            "source": "object.cn || (object.givenName + ' ' + object.sn)"
                        }
                    },
                    ...
            }
            ...
        }
    ]
}
```

d. (Optional) Configure social authentication.

Add an `aliasList` property to the `user` object:

```
"aliasList": {
    "title": "User Alias Names List",
    "description": "List of identity aliases used primarily to record social IdP subjects for this
 user",
    "type": "array",
    "items": {
        "type": "string",
        "title": "User Alias Names Items"
    },
    "viewable": false,
    "searchable": false,
    "userEditable": true,
    "returnByDefault": false,
    "isVirtual": false
}
```

5. Change the authentication mechanism to `rsFilter` only:

- Replace the default `conf/authentication.json` file with this file.

- Check that the `clientSecret` matches the `Client secret` that you set for the `idm-resource-server` client in AM (see "Configure OAuth Clients").
- Check that the `scopes` match the `Scope(s)` that you set for the `idm-resource-server` client in AM (see "Configure OAuth Clients").

For more information about authenticating using the `rsFilter`, see Authenticating through AM in the *IDM Authentication and Authorization Guide*.

6. Edit the IDM Admin UI configuration so that you can still authenticate through the IDM Admin UI:

   a. In your `/path/to/openidm/conf/ui-configuration.json` file, insert a `platformSettings` object into the `configuration` object:

   ```
   {
       "configuration" : {
           ...
           "platformSettings" : {
               "adminOauthClient" : "idm-admin-ui",
               "adminOauthClientScopes" : "fr:idm:*",
               "amUrl" : "http://am.example.com:8081/am",
               "loginUrl" : ""
           }
       }
   }
   ```

   This object tells the IDM Admin UI that it is operating in "platform mode" (that is, as an OAuth 2.0 client of AM).

   b. In your `conf/ui.context-admin.json` file, check that `X-Frame-Options` is set to `SAMEORIGIN`:

   + *Sample ui.context-admin.json*

   ```
   {
       "enabled" : true,
       "urlContextRoot" : "/admin",
       "defaultDir" : "&{idm.install.dir}/ui/admin/default",
       "extensionDir" : "&{idm.install.dir}/ui/admin/extension",
       "responseHeaders" : {
           "X-Frame-Options" : "SAMEORIGIN"
       }
   }
   ```

7. Configure the CORS servlet filter.

   Replace the default `conf/servletfilter-cors.json` file with this file.

8. If you have not already started IDM, start it now, and test that you can access the IDM Admin UI at http://openidm.example.com:8080/admin. When you log in to the Admin UI, use the default AM administrative user (`amAdmin`), and not `openidm-admin`.

9. (Optional) If you want to use the `PlatformForgottenUsername` or `PlatformResetPassword` trees, configure outbound email.

   Note

   After you have installed the Platform UI, you can configure email through the UI at `http://openidm.example.com:8080/admin`.

IDM is now configured. Move on to setting up the Platform UI.

Chapter 4

Deploy the Platform UIs

The ForgeRock Identity Platform® includes three UIs:

Admin UI

Lets platform administrators manage applications, identities, end user journeys, and so on.

End User UI

An example UI that demonstrates profile maintenance, consent management, workflow and delegated administration capabilities. This UI makes REST calls to IDM (using an OAuth 2 bearer token) and to AM (using an SSO cookie).

> **Important**
>
> This UI is *not* the same as the standalone IDM End User UI. The platform End User UI is designed specifically for an integrated platform deployment. The IDM End User UI should be used only in a standalone IDM deployment.

Login UI

A unified login UI that lets both administrators and end users log in to the platform.

This guide describes two ways to deploy the UIs:

- By running Docker images
- By installing a `.zip` file in the AM web container

Run Docker Images

In this deployment, the three UIs are deployed from three Docker images.

> **Important**
>
> Either deploy the Platform UI by running Docker images or by installing a `.zip` file, but do not attempt to deploy the UI both ways. If you have already deployed the Platform UI by installing a `.zip` file, do not perform these steps.

1. Download the Docker images:

a. Download the Admin UI:

```
docker pull gcr.io/forgerock-io/platform-admin-ui:7.1.0
```

b. Download the Enduser UI:

```
docker pull gcr.io/forgerock-io/platform-enduser-ui:7.1.0
```

c. Download the Login UI:

```
docker pull gcr.io/forgerock-io/platform-login-ui:7.1.0
```

2. Create an environment file named `platform_env` that will be passed to the `docker` command.

   The file should have the following contents:

```
AM_URL=http://am.example.com:8081/am
AM_ADMIN_URL=http://am.example.com:8081/am/ui-admin
IDM_REST_URL=http://openidm.example.com:8080/openidm
IDM_ADMIN_URL=http://openidm.example.com:8080/admin
IDM_UPLOAD_URL=http://openidm.example.com:8080/upload
IDM_EXPORT_URL=http://openidm.example.com:8080/export
ENDUSER_UI_URL=http://enduser.example.com:8888
PLATFORM_ADMIN_URL=http://admin.example.com:8082
ENDUSER_CLIENT_ID=end-user-ui
ADMIN_CLIENT_ID=idm-admin-ui
THEME=default
PLATFORM_UI_LOCALE=en
```

3. Start each Docker image respectively, specifying the environment file:

   a. Change to the the directory in which you saved the `platform_env` file, then start the Login UI:

```
docker run -t -i --rm -p 8083:8080 --env-file=platform_env \
gcr.io/forgerock-io/platform-login-ui:7.1.0
Replacing env vars in JS

Setting AM URL as http://am.example.com:8081/am
Setting AM ADMIN URL as http://am.example.com:8081/am/ui-admin
Setting IDM REST URL as http://openidm.example.com:8080/openidm
Setting IDM ADMIN URL as http://openidm.example.com:8080/admin
...
```

   Note

   The `--rm` option causes the **docker run** command to remove the container on exit. Remove this option if you want to keep a container when it exits.

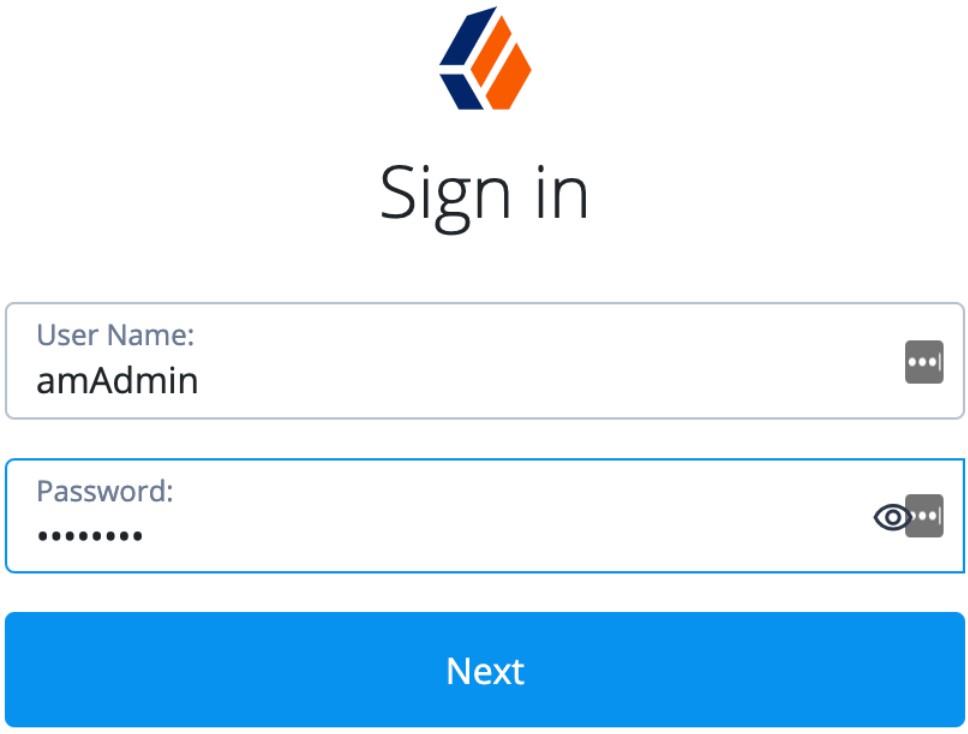
   b. In a new terminal, change to the the directory in which you saved the `platform_env` file, then start the Platform Admin UI:

```
docker run -t -i --rm -p 8082:8080 --env-file=platform_env \
gcr.io/forgerock-io/platform-admin-ui:7.1.0
Replacing env vars in JS

Setting AM URL as http://am.example.com:8081/am
Setting AM ADMIN URL as http://am.example.com:8081/am/ui-admin
Setting IDM REST URL as http://openidm.example.com:8080/openidm
Setting IDM ADMIN URL as http://openidm.example.com:8080/admin
...
```

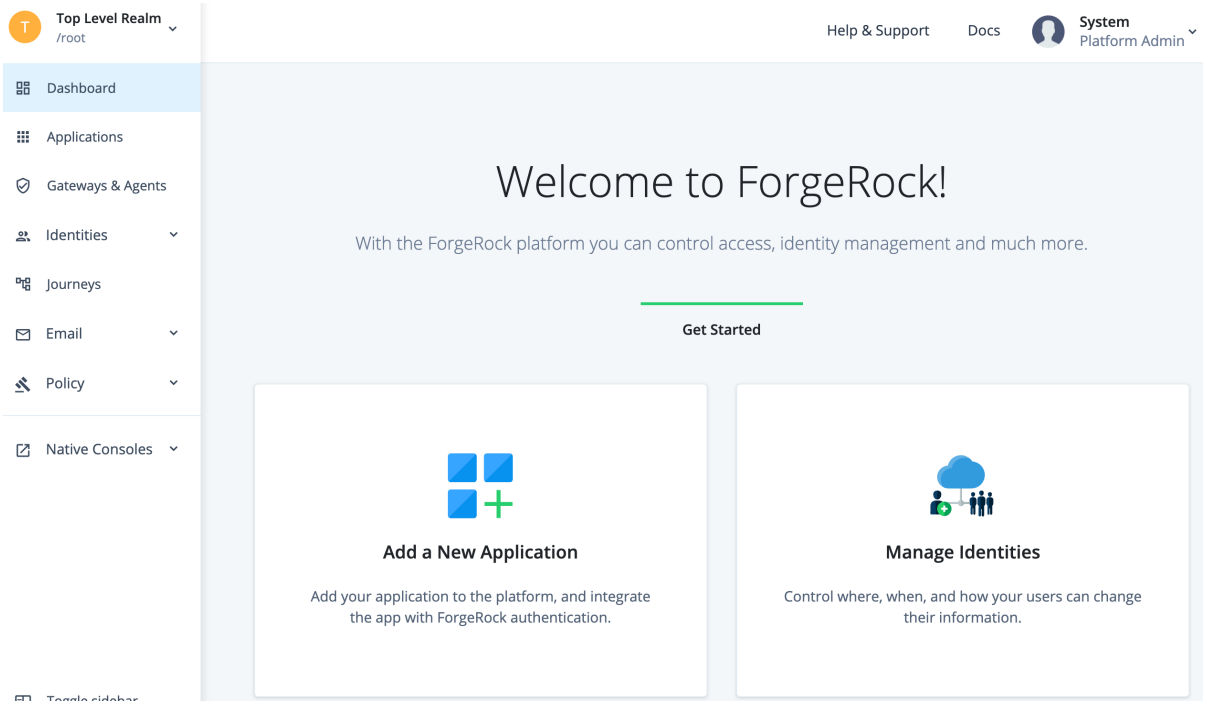
c. In a new terminal, change to the the directory in which you saved the `platform_env` file, then start the End User UI:

```
docker run -t -i --rm -p 8888:8080 --env-file=platform_env \
gcr.io/forgerock-io/platform-enduser-ui:7.1.0
Replacing env vars in JS

Setting AM URL as http://am.example.com:8081/am
Setting AM ADMIN URL as http://am.example.com:8081/am/ui-admin
Setting IDM REST URL as http://openidm.example.com:8080/openidm
Setting IDM ADMIN URL as http://openidm.example.com:8080/admin
...
```

4. You can test access to the Admin UI by logging in through the Login UI at http://login.example.com:8083/?realm=/#/?goto=http%3A%2F%2Flocahost%3A8082.



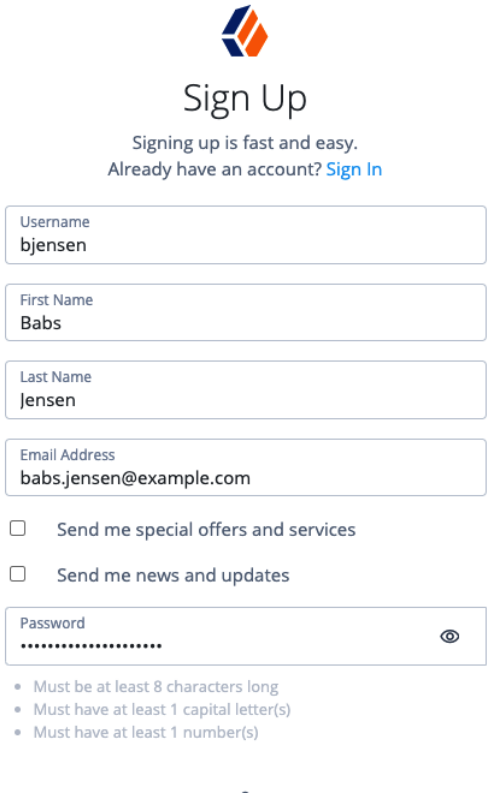
5. Log in as `amAdmin`. You should be redirected to the Platform Admin UI:



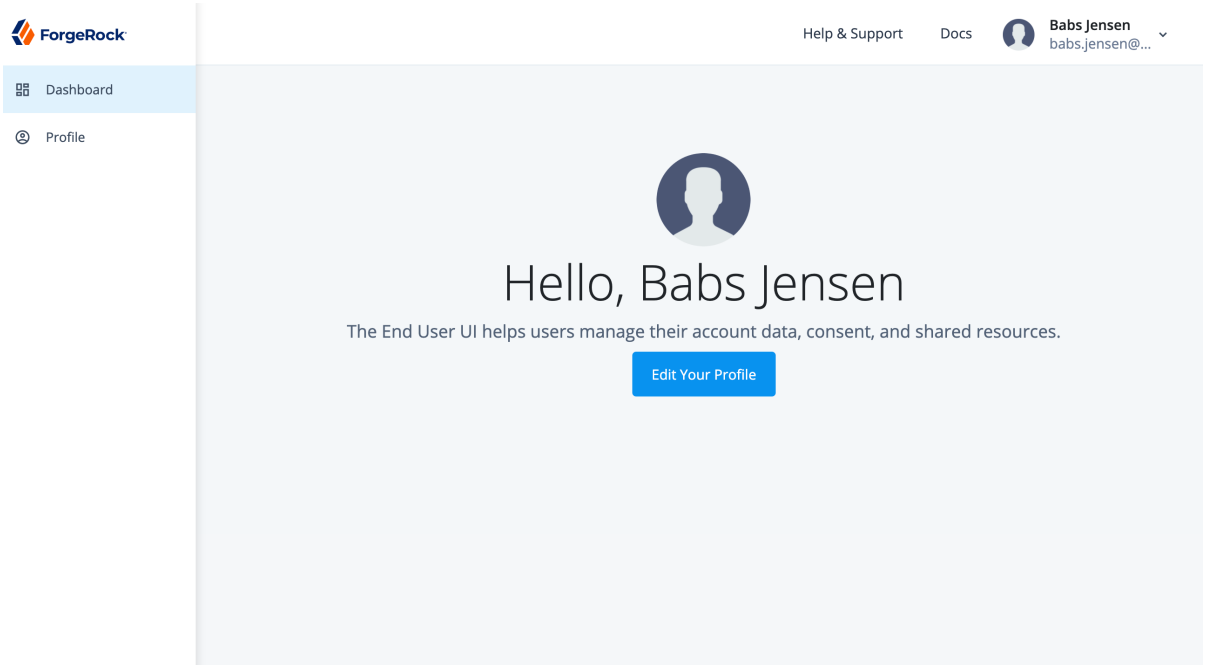
6. Log out of the Admin UI.

7. To test the End User UI, self-register as a new user.

   Browse to http://login.example.com:8083/?realm=/#/service/PlatformLogin?goto=http%3A%2F%2Flocahost%3A8888 and click Create an account:



8. When the required fields have been completed, you are logged into the End User UI, as that user:



Install a `.zip` File

In this deployment, you obtain the three UIs from a `.zip` file that you get from ForgeRock. Then, you install them in the AM web container. You also tweak the AM and IDM configurations so that they work with this UI installation.

Important

Either deploy the Platform UI by running Docker images or by installing a `.zip` file, but do not attempt to deploy the UI both ways. If you have already deployed the Platform UI by running Docker images, do not perform these steps.

1. Get the UI zip file, and unzip it into the `tmp` directory:
   a. Create the `/tmp/ui-zip` directory.
   b. Download the most recent version of the 7.1.x UI `.zip` file from the Platform section of the ForgeRock downloads site.
   c. Copy the UI `.zip` file into the `/tmp/ui-zip` directory.
   d. Unzip the UI `.zip` file.

2. Replace URLs in the UI application with URLs for your ForgeRock Identity Platform deployment:

   a. Change to the `/tmp/ui-zip/PlatformUI` directory.

   b. Run the following commands:

   ```
   export AM_URL=http://am.example.com:8081/am
   export AM_ADMIN_URL=http://am.example.com:8081/am/ui-admin
   export IDM_REST_URL=http://openidm.example.com:8080/openidm
   export IDM_ADMIN_URL=http://openidm.example.com:8080/admin
   export IDM_UPLOAD_URL=http://openidm.example.com:8080/upload
   export IDM_EXPORT_URL=http://openidm.example.com:8080/export
   export ENDUSER_UI_URL=http://am.example.com:8081/enduser
   export PLATFORM_ADMIN_URL=http://am.example.com:8081/platform
   export ENDUSER_CLIENT_ID=end-user-ui
   export ADMIN_CLIENT_ID=idm-admin-ui
   export THEME=default
   export PLATFORM_UI_LOCALE=en
   ```

   c. Run the `variable_replacement.sh` script.

   This script replaces variables in the `.zip` file's UI web application with the values of the environment variables that you set in the previous step:

   ```
   ./variable_replacement.sh www/platform/js/*.js www/enduser/js/*.js www/login/js/*.js
   Replacing env vars in JS

   Setting AM URL as http://am.example.com:8081/am
   Setting AM ADMIN URL as http://am.example.com:8081/am/ui-admin
   Setting IDM REST URL as http://openidm.example.com:8080/openidm
   . . .
   ```

3. Install the UI:

   a. Install the `platform` and `enduser` web applications into AM's web container:

   ```
   cd /path/to/tomcat/webapps
   cp -r /tmp/ui-zip/PlatformUI/www/platform .
   cp -r /tmp/ui-zip/PlatformUI/www/enduser .
   ```

   b. Modify AM to use the new Login UI:

   ```
   cd /path/to/tomcat/webapps/am/XUI
   cp -r /tmp/ui-zip/PlatformUI/www/login/* .
   ```

4. Revise the AM and IDM configurations to work with the new UI installation:

   a. If you're not currently logged in to the AM console as the `amAdmin` user, log in.

   b. Revise the success URL for three authentication trees, so that they refer to the End User UI:

      i. Change the success URL for the `Platform Regstration` tree.

In the Top Level Realm, select Authentication > Trees, and click on PlatformRegistration. Select the Success URL node. Change the Success URL value to http://am.example.com:8081/enduser, and then click Save.

ii. Use the same technique to change the Success URL value for the `PlatformLogin` tree to http://am.example.com:8081/enduser.

iii. Use the same technique to change the Success URL value for the `PlatformResetPassword` tree to http://am.example.com:8081/enduser.

c. Clear the External Login Page URL value.

In the Top Level Realm, select Authentication > Settings, and click the General tab. Set External Login Page URL to an empty string, and then click Save Changes.

d. Log out of the AM console.

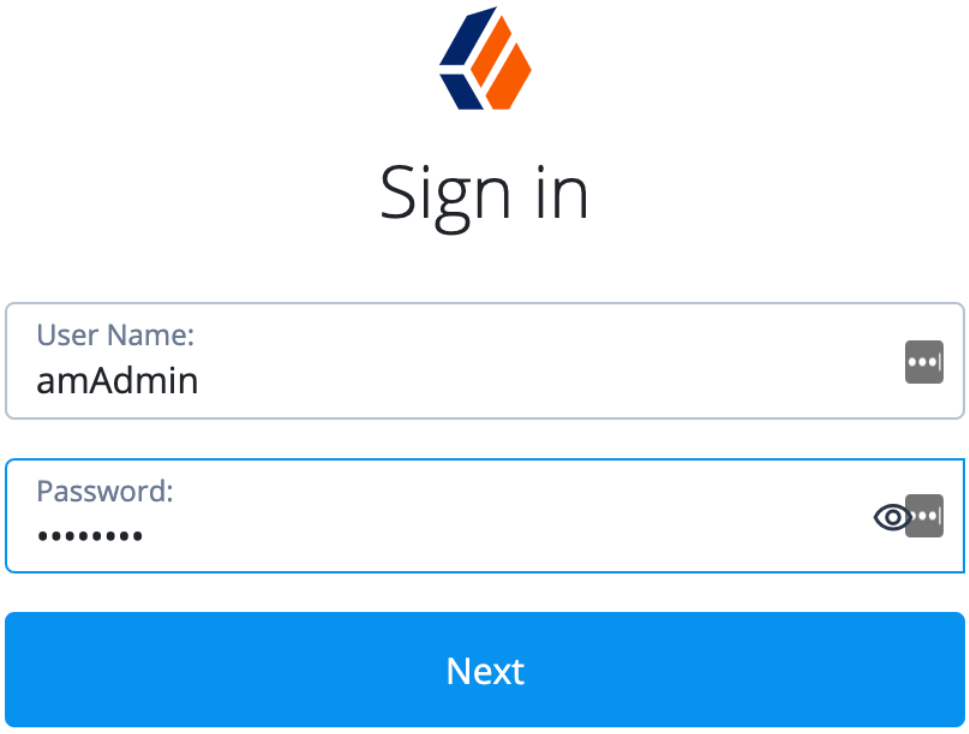
e. Create the `/path/to/openidm/conf/ui-themerealm.json` file with the following contents:

```
{
    "realm": {}
}
```

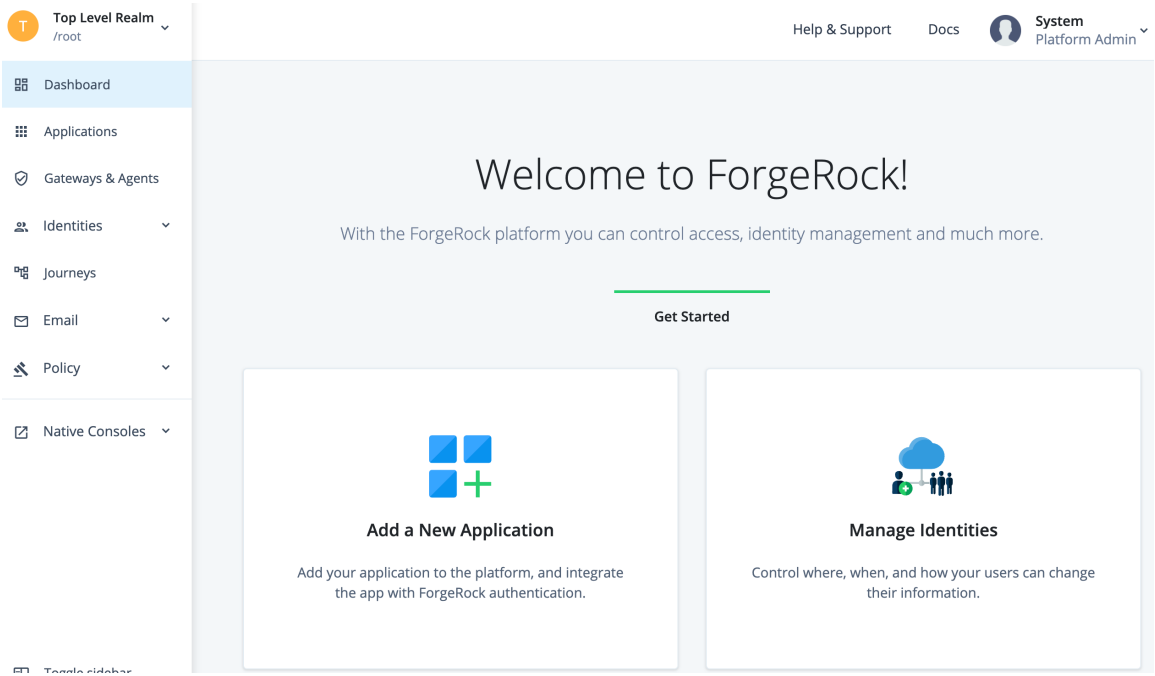
5. Test access to the Admin UI:

a. Go to http://am.example.com:8081/platform.

The Sign in page appears:



b. Log in as the `amAdmin` user. You should be redirected to the Admin UI:



6. Test access to the End User UI:
   a. In the Admin UI, create a test user:
      i. Click Identities > Manage.
      ii. Click + New User.
      iii. Fill in the user details.
      iv. Click Save.
   b. Log out of the Admin UI.
   c. Go to http://am.example.com:8081/enduser.

      The Sign in page appears.
   d. Log in as the test user you just created. You should be redirected to the End User UI.

Chapter 5

Protect the Deployment

After you set up a sample deployment according to "*Deployment One - Separate Identity Stores*" or "*Deployment Two - Shared Identity Store*", you have a functionally complete ForgeRock Identity Platform installation.

Notice, however, that your deployment is lacking security. Except when connecting to the directory service, connections use HTTP, not HTTPS. Users send their login credentials in unprotected cleartext.

The instructions that follow show how to add ForgeRock Identity Gateway (IG) to your deployment, providing a single point of entry to ForgeRock Identity Platform, and securing communications from outside with HTTPS.

Important

- The following examples deploy IG on `platform.example.com` using port 9443 for HTTPS. Adapt the examples as necessary for your deployment.

  For example, if you are demonstrating the deployment on your computer, add an alias for the fully qualified domain name to your `/etc/hosts` file:

  ```
  127.0.0.1    platform.example.com
  ```

- Before using IG to protect the deployment, make sure that AM uses `example.com` as the cookie domain.

  This setting ensures that your browser can share cookies across subdomains in `example.com`. If AM is using the default cookie domain, such as `am.example.com`, then users will not be able to sign in through IG.

  You can check by logging in to the AM native console as `amAdmin`, browsing to Global Services > Platform, and verifying that Cookie Domains is set to `example.com`.

The instructions that follow are sufficient to add IG to your sample deployment. For in-depth documentation on using IG, see the IG product documentation.

Prepare Keys

HTTPS requires a server key pair for IG.

In a public deployment where IG is the entry point, get the IG server certificate signed by a well-known CA. A server certificate signed by a well-known CA will be trusted by other systems and browsers, because the CA certificate is distributed with the system or the browser.

The examples that follow do not use a well-known CA. Instead, they use an IG server certificate generated with the DS deployment key and password. This is a DS deployment key and password that you used to set up the deployment. You will generate an IG server key pair using the DS **dskeymgr** command. You will export and trust the deployment key CA certificate to demonstrate and test the deployment. As long as no one else knows the deployment key and password combination, that is safe, because only you have the secrets to generate signed keys.

Generate the keys on a system where you installed DS:

1. Save a password file for the keystore:

```
mkdir -p /path/to/security/
touch /path/to/security/keystore.pin
chmod 600 /path/to/security/keystore.pin
echo -n password > /path/to/security/keystore.pin
```

2. Set the deployment key in an environment variable that you will use in **dskeymgr** commands:

```
export DEPLOYMENT_KEY=deployment-key
```

3. Generate a server key pair for IG to use for HTTPS:

```
/path/to/opendj/bin/dskeymgr \
 create-tls-key-pair \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --keyStoreFile /path/to/security/keystore \
 --keyStorePassword:file /path/to/security/keystore.pin \
 --hostname localhost \
 --hostname platform.example.com \
 --subjectDn CN=platform.example.com,O=ForgeRock
```

4. Inspect the contents of the keystore you just created to find the key alias is `ssl-key-pair`:

```
keytool -list -keystore /path/to/security/keystore -storepass:file /path/to/security/keystore.pin
Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 1 entry

ssl-key-pair, date, PrivateKeyEntry,
Certificate fingerprint (SHA-256): fingerprint
```

5. Copy the keystore and password to the system where IG runs.

6. Export the deployment key CA certificate to trust:

```
/path/to/opendj/bin/dskeymgr export-ca-cert \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --outputFile ca-cert.pem
```

   Use this `ca-cert.pem` file when you need to trust the IG server certificate. You can import it into your browser to trust the platform URLs in this example.

Install IG

This example uses IG in standalone mode. In standalone mode, IG runs in the web container provided in the .zip file.

1. Download IG-7.1.0.zip.
2. Unpack the files.

   This example shows the files unpacked in the `/path/to/identity-gateway/` directory.
3. Create a configuration directory for IG:

   ```
   mkdir -p $HOME/.openig/config/
   ```

   The default IG configuration directory is `$HOME/.openig/config` on Linux and UNIX, `%appdata%\OpenIG\config` on Windows.
4. Add an configuration file for HTTPS support at the root of the IG configuration directory.

   + *Sample admin.json*

   This example expects the keystore and `keystore.pin` file in the `/path/to/security/` directory:

   ```
   {
       "heap": [
           {
               "name": "TlsConf",
               "type": "ServerTlsOptions",
               "config": {
                   "keyManager": {
                       "type": "SecretsKeyManager",
                       "config": {
                           "signingSecretId": "key.manager.secret.id",
                           "secretsProvider": "ServerIdentityStore"
                       }
                   }
               }
           },
           {
               "name": "SecretsPasswords",
               "type": "FileSystemSecretStore",
               "config": {
                   "directory": "/path/to/security/",
                   "format": "PLAIN"
               }
           },
           {
               "name": "ServerIdentityStore",
               "type": "KeyStoreSecretStore",
               "config": {
                   "file": "/path/to/security/keystore",
                   "storePassword": "keystore.pin",
                   "secretsProvider": "SecretsPasswords",
   ```

```
                "mappings": [
                    {
                        "secretId": "key.manager.secret.id",
                        "aliases": [
                            "ssl-key-pair"
                        ]
                    }
                ]
            }
        }
    ],
    "connectors": [
        {
            "port": 9080
        },
        {
            "port": 9443,
            "tls": "TlsConf"
        }
    ]
}
```

5. Start IG:

```
/path/to/identity-gateway/bin/start.sh
...
[main] INFO ... started in XXXXms on ports : [9080, 9443]
```

6. Test that you can use the IG "ping" endpoint over HTTP:

```
curl -v http://platform.example.com:9080/openig/ping
```

   The output should include an HTTP `200` status code indicating success, and an empty reply from IG.

7. When you can successfully ping IG over HTTP, try HTTPS using your CA certificate file:

```
curl -v --cacert ca-cert.pem https://platform.example.com:9443/openig/ping
```

   The `ca-cert.pem` file is the one you exported with the **dskeymgr** command. If it is not in the current directory, include the path to the file.

   As before, you should receive an empty success response. This demonstrates that you successfully connected to IG over HTTPS.

   Now that you are sure the HTTPS connection works, you can edit `admin.json` to stop using port 9080 for HTTPS.

Trust the Deployment Key Certificate

Important

You generated a deployment key CA certificate used in this example with the **dskeymgr** command. You browser does not recognize this private CA until you trust it by importing the CA certificate.

Make sure this certificate is trusted when negotiating SSL and HTTPS connections.

As long as no one else knows your deployment key and password combination, this is safe, because only you have the secrets to generate this CA certificate, or to generate signed keys:

- Trust the deployment key CA certificate by importing the `ca-cert.pem` file *as a certificate authority* trusted for SSL and HTTPS connections.

  Some browsers call this "Authorities" in the browser settings. See the browser help for details.

Configure IG

IG terminates HTTPS for your deployment. All access to your deployment goes through IG.

You must therefore configure IG to route traffic through to the components of the deployment, and then back to clients of the platform:

1. Add a configuration file at the root of the IG configuration directory.

   + *Sample config.json*

     This example configures the capture decorator, a router, and a default route:

```
{
    "heap": [
        {
            "name": "capture",
            "type": "CaptureDecorator"
        },
        {
            "type": "Router",
            "name": "_router"
        },
        {
            "name": "DefaultRoute",
            "type": "StaticResponseHandler",
            "config": {
                "status": 200,
                "headers": {
                    "Content-Type": [
                        "text/html"
                    ]
                },
                "entity": "<html><p><a href='/enduser-login/?realm=/#/service/PlatformLogin'>End
 user access</a></p><p><a href='/platform-login/?realm=/#/'>Admin access</a></p></html>"
            }
        }
    ],
    "handler": "_router",
    "audit": "global"
}
```

The default route in the sample file defines a simple static HTML page to present when a user accesses the platform before logging in, or after logging out.

In this example, the page includes a link for each of the two user flows:

1. An end user logs in to access their applications and profile page through the platform end user UI.

2. An admin logs in to configure the platform through the platform admin UI and the native consoles.

You can configure IG to redirect users to your site if desired. IG can also serve static pages, so you can add an entry point that matches your site.

2. Create a `routes` directory under the IG configuration file directory:

```
mkdir $HOME/.openig/config/routes
```

3. Add a route file in the `routes` directory.

   The file name ensures this will be the last route:

   + *Sample zz-default.json*

```
{
    "handler": "DefaultRoute"
}
```

4. Add a route file in the `routes` directory.

   This route consumes the paths specified in the links on the default, static page to route users to the correct URL on the platform login UI:

   + *Sample login.json*

```
{
    "name": "login",
    "condition": "${matches(request.uri.path, '^/enduser-login|^/platform-login')}",
    "baseURI": "http://login.example.com:8083",
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "type": "UriPathRewriteFilter",
                    "config": {
                        "mappings": {
                            "/enduser-login": "/",
                            "/platform-login": "/"
                        },
                        "failureHandler": "DefaultRoute"
                    }
                }
            ],
            "handler": "ReverseProxyHandler"
        }
    }
}
```

5. Add an route file in the `routes` directory.

   This route sends end users to the platform end user UI:

   + *Sample enduser-ui.json for users running the platform UI from Docker images*

```
{
    "name": "enduser-ui",
    "condition": "${matches(request.uri.path, '^/enduser-ui')}",
    "baseURI": "http://enduser.example.com:8888",
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "type": "UriPathRewriteFilter",
                    "config": {
                        "mappings": {
                            "/enduser-ui": "/"
                        },
                        "failureHandler": "DefaultRoute"
                    }
                }
            ],
            "handler": "ReverseProxyHandler"
        }
    }
}
```

+ *Sample enduser-ui.json for users who installed the platform UI from a .zip file*

```
{
    "name": "enduser-ui",
    "condition": "${matches(request.uri.path, '^/enduser-ui')}",
    "baseURI": "http://am.example.com:8081",
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "type": "UriPathRewriteFilter",
                    "config": {
                        "mappings": {
                            "/enduser-ui": "/enduser"
                        },
                        "failureHandler": "DefaultRoute"
                    }
                }
            ],
            "handler": "ReverseProxyHandler"
        }
    }
}
```

6. Add a route file in the `routes` directory.

   This route sends administrators to the platform admin UI:

   + *Sample platform-ui.json for users running the platform UI from Docker images*

```
{
    "name": "platform-ui",
    "condition": "${matches(request.uri.path, '^/platform-ui')}",
    "baseURI": "http://admin.example.com:8082",
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "type": "UriPathRewriteFilter",
                    "config": {
                        "mappings": {
                            "/platform-ui": "/"
                        },
                        "failureHandler": "DefaultRoute"
                    }
                }
            ],
            "handler": "ReverseProxyHandler"
        }
    }
}
```

+ *Sample platform-ui.json for users who installed the platform UI from a .zip file*

```
{
    "name": "platform-ui",
    "condition": "${matches(request.uri.path, '^/platform-ui')}",
    "baseURI": "http://am.example.com:8081",
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "type": "UriPathRewriteFilter",
                    "config": {
                        "mappings": {
                            "/platform-ui": "/platform"
                        },
                        "failureHandler": "DefaultRoute"
                    }
                }
            ],
            "handler": "ReverseProxyHandler"
        }
    }
}
```

7. Add an route file in the `routes` directory.

   This route handles requests to AM, ensuring that the browser gets redirected through IG:

   + *Sample am.json*

```
{
    "name": "am",
    "baseURI": "http://am.example.com:8081",
    "condition": "${matches(request.uri.path, '(?:^/am(?!/XUI))')}",
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "comment": "Always redirect to IG rather than AM",
                    "type": "LocationHeaderFilter",
                    "config": {
                        "baseURI": "https://platform.example.com:9443/"
                    }
                }
            ],
            "handler": "ReverseProxyHandler"
        }
    }
}
```

8. Add an route file in the `routes` directory.

   This route handles requests to IDM, ensuring that the browser gets redirected through IG:

   + *Sample idm.json*

```
{
    "name": "idm",
    "baseURI": "http://openidm.example.com:8080",
    "condition": "${matches(request.uri.path, '(?:^/openidm)|(?:^/admin)|(?:^/upload)|(?:^/
export)')}",
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "comment": "Always redirect to IG rather than IDM",
                    "type": "LocationHeaderFilter",
                    "config": {
                        "baseURI": "https://platform.example.com:9443/"
                    }
                }
            ],
            "handler": "ReverseProxyHandler"
        }
    }
}
```

9. Restart IG to take all your changes into account.

   Verify in the server output that all your routes load successfully.

When troubleshooting and developing routes, know that you can change an IG route configuration while IG is running, and IG will reload changed routes every 10 seconds by default.

The IG capture decorator is particularly useful when troubleshooting routes. To use it quickly, add `"capture": "all"` to an object in a route, and let IG reload the route before you try it. It logs messages in the server output about incoming requests at any point until IG sends the request, and outgoing responses at any point until IG delivers the response.

For more information, see the IG product documentation.

Adapt the AM Configuration

A sample deployment configured according to "*Deployment One - Separate Identity Stores*" or "*Deployment Two - Shared Identity Store*" does not route the traffic through IG. Adapt the AM and OAuth 2.0 client configurations to use IG.

The following minimal changes are sufficient to use the sample deployment. Add any additional changes necessary for your deployment:

1. Log in to the platform admin UI through the existing URL as `amAdmin`.

   The password used in the documentation to set up the platform is `Passw0rd`.

2. Add sign-in URLs through IG for the `end-user-ui` application:

   ```
   https://platform.example.com:9443/enduser-ui/appAuthHelperRedirect.html
   https://platform.example.com:9443/enduser-ui/sessionCheck.html
   ```

   If you use the AM native console, these are redirection URIs of the OAuth 2.0 client.

3. Add sign-in URLs through IG for the `idm-admin-ui` application:

   ```
   https://platform.example.com:9443/platform/appAuthHelperRedirect.html
   https://platform.example.com:9443/platform/sessionCheck.html
   https://platform.example.com:9443/admin/appAuthHelperRedirect.html
   https://platform.example.com:9443/admin/sessionCheck.html
   https://platform.example.com:9443/platform-ui/appAuthHelperRedirect.html
   https://platform.example.com:9443/platform-ui/sessionCheck.html
   ```

   If you use the AM native console, these are redirection URIs of the OAuth 2.0 client.

4. Use the Journeys page to edit Success URLs for the following journeys, which correspond to AM trees:

   PlatformLogin

   Success URL: `https://platform.example.com:9443/enduser-ui/`

PlatformRegistration

Success URL: `https://platform.example.com:9443/enduser-ui/`

PlatformResetPassword

Success URL: `https://platform.example.com:9443/enduser-ui/`

5. Under Native Consoles, select Access Management to open the AM admin console, and update these settings:

   Top Level Realm > Authentication > Settings > General

   External Login Page URL: `https://platform.example.com:9443/enduser-login`

   Top Level Realm > Services > Validation Service

   To Valid goto URL Resources, add:

   ```
   https://platform.example.com:9443/*
   https://platform.example.com:9443/*?*
   ```

   Configure > Global Services > CORS Service > Secondary Configurations > Cors Configuration

   To Accepted Origins, add: `https://platform.example.com:9443`

Adapt the Platform UI Configuration

Adapt the platform UI configuration to use IG:

1. Keep a copy of the current configuration for testing:

   ```
   cp /path/to/platform_env /path/to/platform_env.bak
   ```

2. Replace the `/path/to/platform_env` content with settings that direct traffic through IG:

   ```
   AM_URL=https://platform.example.com:9443/am
   AM_ADMIN_URL=https://platform.example.com:9443/am/ui-admin
   IDM_REST_URL=https://platform.example.com:9443/openidm
   IDM_ADMIN_URL=https://platform.example.com:9443/admin
   IDM_UPLOAD_URL=https://platform.example.com:9443/upload
   IDM_EXPORT_URL=https://platform.example.com:9443/export
   ENDUSER_UI_URL=https://platform.example.com:9443/enduser-ui
   PLATFORM_ADMIN_URL=https://platform.example.com:9443/platform-ui/
   ENDUSER_CLIENT_ID=end-user-ui
   ADMIN_CLIENT_ID=idm-admin-ui
   THEME=default
   PLATFORM_UI_LOCALE=en
   ```

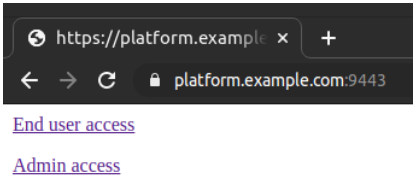
3. Restart the platform UI Docker containers to take the changes into account.

Test the Deployment

Once you have finished changing the configuration to use IG, test your work:

1. Log out of any open platform applications.
2. Browse https://platform.example.com:9443/.

   You should see the default route page:

   If you see a page suggesting that your connection is not private, a warning about a security risk, or any other indication that your browser does not trust the IG server certificate, read "Trust the Deployment Key Certificate" again.
3. Test the flow for an end user:
   a. Click the End user access link.
   b. Sign in as an end user.

      If you have not yet created an end user account, follow the Create an account link and register an end user with the platform.

      In the end, your browser should be directed to the end user page at `https://platform.example.com:9443/enduser-ui/?realm=root#/dashboard`.
   c. Sign out.

      Your browser should be directed to the platform default page.
4. Test the flow for an administrator:
   a. Click the Admin access link.
   b. Sign in as `amAdmin`.

      The password used in the documentation to set up the platform is `Passw0rd`.

      Your browser should be directed to the admin UI page at `https://platform.example.com:9443/platform-ui/?realm=root#/dashboard`.
   c. Browse around the platform admin UI, and open the native consoles using the links provided.

The browser address should show a secure connection through `https://platform.example.com:9443/` for all pages.

d. Sign out.

Your browser should be directed to the platform default page.

You can now route traffic through IG to your platform deployment. If you get the IG server certificate signed by a well-known CA, instead of the private CA, other browsers and systems can connect through IG without additional configuration.

Chapter 6

Migration and Customization

Previous versions of the ForgeRock Identity Platform provided a sample deployment called the Full Stack Sample. This kind of deployment is not supported for new integrations between IDM and AM, and you should follow the steps outlined in this guide to set up a new deployment. There is no automated migration facility for an existing deployment based on the *full stack sample*, but you can follow the tips in this chapter help you move to a new integrated deployment.

Note

This is not an exhaustive list and you will have additional configuration changes to make, based on how you deployed the full stack sample.

Self-Service Configuration

All self-service configuration is now done in the AM UI, with the exception of the following elements:

- KBA
- Terms & Conditions
- Policies
- Privacy & Consent
- Email templates and services

You can therefore delete all the IDM `selfservice*` files from your configuration, apart from `selfservice.kba.json` and `selfservice.terms.json`. These two files are still used by the new AM authentication trees.

You will need to rebuild all existing self-service processes as AM authentication trees. Note that tree nodes are more modular than the legacy IDM self-service stages. It might take more than one node to replace a self-service stage.

In a platform configuration, you can optionally customize the IDM Admin UI to hide or remove all of the self-service functions that are no longer configured through IDM (such as registration and password reset).

Authentication

Integrated deployments now use a single `rsFilter` authentication module that allows authentication using AM bearer tokens. You must replace your project's `conf/authentication.json` file with the platform authentication.json file.

Configure OAuth clients and the IDM Provisioning Service in the AM UI.

UI Customization

The UI retrieves access tokens from AM. If you have customized your Self-Service UI for a full-stack deployment, it is recommended that you start a new customization, based on the Platform UI. Your existing UI customizations will not work in a new platform deployment.

Migration for Shared Identities

As shown in "*Deployment Two - Shared Identity Store*", AM and IDM can share a DS identity store. Since version 7, this is a well-supported and recommended deployment pattern.

Previously, as shown in "*Deployment One - Separate Identity Stores*" and *Reconciliation and AM* in the IDM 6.5 documentation, AM used a DS-based identity store, IDM used a relational database for its repository, and IDM synchronized identities between them.

When you migrate shared identities to a shared DS identity store, follow these high-level steps:

- Add any custom identity attributes to the shared identity store and platform components.

  Many deployments define custom attributes that serve in profiles or security tokens, which are critical to authentication and authorization, such as OAuth 2.0 access tokens, OpenID Connect ID tokens, and SAML assertions.

  For detailed instructions, see "Add a Custom Attribute" below.

- Use IDM to migrate identity data to the shared DS identity store.

  Make sure that any operations to modify managed objects, such as managed users, performed as part of the mapping (in `sync.json`), are reflected in the managed object schema (in `managed.json`).

  For more on the migration operation, see *Migrate Data* in the IDM documentation.

UI Customization

In version 7 of ForgeRock Identity Platform, the AM and IDM component products continue to provide their own, separate UIs so you can deploy one independently of the other. Version 7 changes the model for deploying AM and IDM together in a ForgeRock Identity Platform configuration. Key changes to the platform deployment model concern OAuth 2.0 and platform UIs.

When you use AM and IDM together in a ForgeRock Identity Platform configuration:

- AM centralizes authentication and authorization services for the platform.

  IDM acts as an OAuth 2.0 client of AM. Even when you log in to administer IDM, you authenticate through AM.

  More generally, all new client applications are expected to obtain an access token from AM, and present it to IDM for authorization; for example, when calling IDM REST endpoints.

  This is why the authentication configuration for IDM includes an `rsFilter` (OAuth 2.0 *r*esource *s*erver) configuration.

- IDM centralizes identity management services.

  IDM relies on the platform UI, which calls AM trees that integrate with IDM for user self-service operations.

New platform UI components replace functions of the AM and IDM native UIs:

- A platform admin UI provides quick access to the configuration capabilities that platform administrators need every day.

  The platform admin UI links to the native AM and IDM admin consoles for additional, less common configuration operations.

  The platform admin UI runs in its own Docker container as an OAuth 2.0 client of AM. It is an administrator-facing service that you run in front of a platform deployment.

  The platform admin UI code is *not available for customization* as of this writing. For configuration of authentication trees, the admin UI uses the self-service tree endpoints described in "Configure Self-Service Trees Endpoints" in the *Platform Self-Service Guide*, and documented in the AM API explorer, which you can access through the AM admin console. If you write your own admin UI, use these endpoints to manage tree configurations.

- A platform end user UI replaces the IDM end user UI in a platform deployment, providing users of your service with personalized pages to access their applications and update their profiles.

  You can customize the platform end user UI for your deployment, as described below.

  The platform end user UI is a single-page application that you can run in its own Docker container, as demonstrated in the sample deployments.

  It operates as an OAuth 2.0 client of AM.

- A platform login UI replaces the AM login UI (XUI) in a platform deployment.

  You can customize the platform login UI for your deployment, as described below.

  The platform login UI is a single-page application that you can run in its own Docker container, as demonstrated in the sample deployments.

It operates as a native AM client, not an OAuth 2.0 client.

High-level instructions for customizing the platform end user and login UIs:

- Clone the platform UI repository.
- Carefully review the README.
- Develop your customizations.
- For each UI that you customize, adjust the variables found in the `.env.production` file to match your production deployment.
- Build the customized UIs.

  You will find the resulting single-page application files in the `dist/` folder of each UI.
- Deploy the files in your environment.

  For details, keep reading.

End User UI Customization

The platform end user UI replaces the IDM end user UI in a platform deployment. IDM still includes the end user UI files, as they are useful in standalone deployments.

Choose how to deploy your customized version of the platform end user UI:

- Deploy your customized end user UI in a separate web server.

  In the AM OAuth 2.0 client profile for the end user UI, set the redirection URIs to reflect the URL to your end user UI.

  This approach is reflected in the sample deployments in this guide.
- Copy the contents of the `dist/*` folder of your customized end user UI over the files in the expanded IDM `ui/enduser` folder, overwriting existing files.

  The end user UI is then in the same domain as IDM.

  If you use a path that is different from `ui/enduser` for the files, also update the `conf/ui.context-enduser.json` configuration to match.

Login UI Customization

The platform login UI replaces the AM login UI (XUI) in a platform deployment. AM still includes the login UI files, as they are useful in standalone deployments.

The login UI operates as a native AM client, capable of working with authentication trees. It is not an OAuth 2.0 client, but instead a component used in OAuth 2.0 flows. AM, acting as an OAuth 2.0 authorization server, relies on the platform login UI for resource owner authentication operations.

The login UI leverages the AM authentication trees that are compatible with ForgeRock Identity Platform, as described in the Platform Self-Service Guide. The login UI translates the AM `/json/authenticate` challenges into web pages for users, and translates user responses back into REST calls.

Choose how to deploy your customized version of the platform login UI:

If you...	How to deploy
Use only OAuth 2.0 clients of AM, not SAML or same-domain policy agents, for example.	Deploy your customized login UI in a separate web server. In AM console, for each realm, browse to Authentication > Settings, and click the General tab. Set the External Login Page URL value to the URL of your customized login UI. If necessary, update the AM CORS and Validation Service configurations. This approach is reflected in the sample deployments in this guide.
Customize the AM .war file for your deployment.	Copy the contents of the `dist/*` folder of your customized login UI over the files in the expanded AM `webapps/am/XUI` folder, overwriting XUI files.

Run AM behind a reverse proxy.	Deploy your customized login UI `dist/*` files in a separate web server, and use the reverse proxy to direct requests for AM XUI to your customized login UI instead.
	
	A few of the JavaScript files from the AM XUI serve to properly render OAuth 2.0 UI screens. Therefore, in addition, customize the AM .war file to move AM XUI files, and specify that location when starting AM.
	
	1. Expand the AM .war.
	
	2. Move the XUI files:
	`mv am/XUI am/OAuth2_XUI`
	
	3. Update runtime options so that they specify the location of the XUI files when AM starts.
	
	For example, if you run in Tomcat as described above, add this option alongside the others in the `setenv.sh` script:
	`-Dorg.forgerock.am.oauth2.consent.xui_path=/OAuth2_XUI`

Add a Custom Attribute

The sample deployments in this guide demonstrate using DS as a shared identity store for AM and IDM. The DS setup profile that configures DS as a shared identity store defines all the platform attributes required by AM and IDM.

Many deployments use additional custom attributes in identity profiles. The following examples show how to add a custom attribute, and how to configure AM and IDM to use it.

Note

This example adds a custom attribute that the platform can retrieve with a user profile. This custom attribute is not searchable, and therefore not indexed.

Before you start, create a `demo` account for test purposes in your sample deployment:

1. Browse to the platform end user UI page of the sample deployment, and click Create an account.
2. Create a user with user identifier `demo`, and whatever other attributes you like.
3. Find this user's entry in the DS shared identity repository:

```
/path/to/opendj/bin/ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=admin \
 --bindPassword str0ngAdm1nPa55word \
 --baseDn ou=identities \
 "(uid=demo)"
```

   Notice that the user's entry is named for its `fr-idm-uuid` attribute.

Define the Attribute in DS

You define the attribute in DS as an attribute type in the LDAP schema. In LDAP, an entry's object classes define which attributes it can have. You therefore also define an object class that lets the entry have the custom attribute.

The example custom attribute is a multi-valued directory string attribute named `customAttribute`. The auxiliary object class that lets the entry have the attribute is named `customAttributeOC`:

1. In DS, add LDAP schema for the new attribute and object class alongside other LDAP schema definitions:

```
/path/to/opendj/bin/ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=admin \
 --bindPassword str0ngAdm1nPa55word << EOF
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( customAttribute-oid
  NAME 'customAttribute'
  EQUALITY caseIgnoreMatch
  ORDERING caseIgnoreOrderingMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  USAGE userApplications )
-
add: objectClasses
objectClasses: ( customAttributeOC-oid
  NAME 'customAttributeOC'
  SUP top
  AUXILIARY
  MAY customAttribute )
EOF
```

By default, DS writes these definitions to the file `/path/to/opendj/db/schema/99-user.ldif`.

2. Test that you can add a custom attribute to the `demo` user entry.

   Use the `fr-idm-uuid` that you got when searching for `uid=demo` in `ou=identities`:

```
/path/to/opendj/bin/ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=admin \
 --bindPassword str0ngAdm1nPa55word << EOF
dn: fr-idm-uuid=<fr-idm-uuid-for-demo-user>,ou=people,ou=identities
changetype: modify
add: objectClass
objectClass: customAttributeOC
-
add: customAttribute
customAttribute: Testing 1, 2...
EOF
```

3. Read the `demo` user entry to check your work:

```
/path/to/opendj/bin/ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=admin \
 --bindPassword str0ngAdm1nPa55word \
 --baseDn ou=identities \
 "(uid=demo)" \
 customAttribute
dn: fr-idm-uuid=<fr-idm-uuid-for-demo-user>,ou=people,ou=identities
customAttribute: Testing 1, 2...
```

Notice that the value of `customAttribute` is set to `Testing 1, 2...`.

LDAP schema features are much richer than this simple example can demonstrate. For details about LDAP schema in DS, see *LDAP Schema*.

Update AM to Use the Attribute

Update the AM configuration to make AM aware of the new object class and attribute:

1. Sign in to the platform admin UI as `amAdmin`.

   The password used in the documentation to set up the platform is `Passw0rd`.

2. Under Native Consoles, select Access Management to open the AM admin console.

3. Under Top Level Realm > Identity Stores > OpenDJ > User Configuration, update these settings:

   LDAP User Object Class

   Add `customAttributeOC`.

   LDAP User Attributes

   Add `customAttribute`.

4. Save your work.

For additional details, see *Adding User Profile Attributes*.

Update IDM to Use the Attribute

Update the IDM configuration to make IDM aware of the attribute:

1. In the `conf/managed.json` file, under user > schema > order, add the custom attribute to the list:

   ```
   "customAttribute",
   ```

2. In the `conf/managed.json` file, under user > schema > properties, define a property corresponding to the custom attribute:

```
"customAttribute" : {
 "title" : "Custom Attribute",
 "type" : "string",
 "viewable" : true,
 "searchable" : false,
 "userEditable" : true
},
```

   Notice that this property is not searchable; meaning, it does not need to be indexed.

3. In the `conf/repo.ds.json` file, under resourceMapping > explicitMapping > managed/user > objectClasses, add the object class:

```
"customAttributeOC",
```

4. In the `conf/repo.ds.json` file, under resourceMapping > explicitMapping > managed/user > properties, add a mapping for the attribute:

```
"customAttribute" : {
 "type" : "simple",
 "ldapAttribute" : "customAttribute"
},
```

5. Restart IDM to take the changes to the `conf/repo.ds.json` file into account.

For additional details, see *Create and Modify Object Types*, and *Explicit Mappings With a DS Repository*.

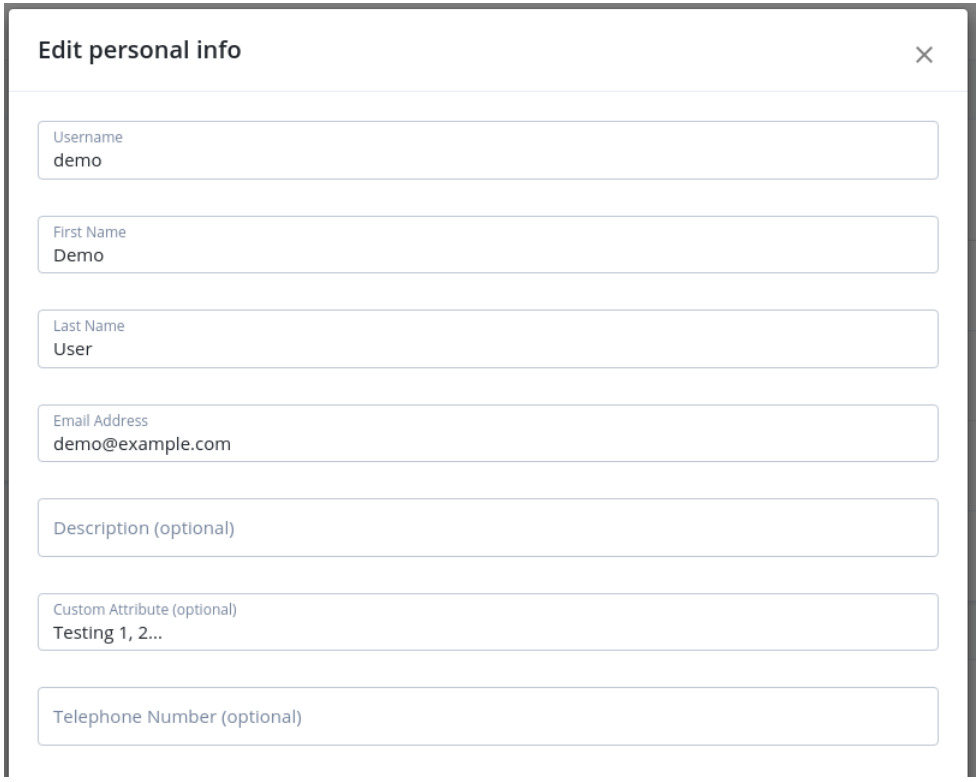
View the Results

After configuring DS, AM, and IDM to use the custom attribute:

1. Browse to the platform end user UI, and sign in as the `demo` user.

2. Click Edit Your Profile, and then click Edit Personal Info.

   Notice that your custom attribute is visible with the value you set:



Chapter 7

HSMs and ForgeRock Software

This part covers how you can use a Hardware Security Module (HSM) to protect ForgeRock software private and secret keys.

On Protecting Secrets

You must protect private keys and secret keys that servers use for cryptographic operations:

Operating System Protections

In many deployments, operating system protections are sufficient. Operating systems are always sufficient for public keys and keystores that do not require protection.

Isolate the server account on the operating system where it runs, and allow only that account to access the keys. If you store the keys with version control software, also control access to that.

This deployment option generally offers a better performance/cost ratio. You must take more care to prevent private and secret keys from being exposed, however.

HSM

For stronger protection, you can use an HSM.

An HSM is a secure device that holds the keys, and protects them from unauthorized access. With an HSM, no one gets direct access to the keys. If an attacker does manage to break into an HSM and theoretically get access to the keys, HSM are designed to make the tampering evident, so you can take action.

To put a private or secret key on an HSM, you have the HSM generate the key. The key never leaves the HSM. Instead, if you need the result of a cryptographic operation involving the key, you authenticate to the HSM and request the operation. The HSM performs the operation without ever exposing any private or secret keys. It only returns the result.

An HSM can be certified to comply with international standards, including FIPS-140 and Common Criteria. An HSM that is certified to comply with these standards can be part of your supported ForgeRock software solution.

ForgeRock software uses the HSM through standard PKCS#11 interfaces, and supports the use of compliant cryptographic algorithms.

An HSM generally offers higher security, but with a significant cost and impact on performance. Good HSMs and standards compliance come with their own monetary costs.

In terms of performance, each cryptographic operation incurs at minimum a round trip on a secure connection, compared with an operation in local memory for keys on the system. Even if the deployment may guarantee the same throughput, take the latency into account when deciding to deploy with HSMs.

Key Management Services

Key management services, such as Google Cloud KMS and others, offer similar advantages and tradeoffs as HSMs.

Latency for online key management services can be very high.

Performance

Throughput, the rate of operations completed, might or might not be impacted by your choice of protecting secrets.

Latency, how long individual operations take, will be impacted by your choice of protecting secrets.

Performance Example

For example, suppose you want a system that signs one million JWTs per second. As long as you can distribute the signing key across enough hardware, your system can sign one million JWTs a second. This is true even if each signing operation takes ten seconds. You simply need ten million systems, and the network hardware to use them in parallel. Throughput therefore depends mainly on how much you can spend.

Suppose, however, that each signing operation must take no longer than ten milliseconds. Furthermore, suppose you have an HSM that can perform a signing operation in one millisecond, but that the network round trip from the system to the HSM takes an average of eleven milliseconds. In this case, you cannot fix performance by buying a faster HSM, or by somehow speeding up the software. You must first reduce the network time if you want to meet your latency requirements.

Perhaps the same signing operation with a key stored on the operating system takes two milliseconds. Consider the options based on your cost and security requirements.

Performance also depends on the following:

- The impact of latency is greater when performing symmetric cryptographic operations (AES, HMAC) compared to public key cryptography (RSA, EC).
- For public key cryptography, only operations that use the private key must contact the HSM (signing, decryption).

  Operations using the public key (verifying a signature, encryption) retrieve the public key from the HSM once, and then use it locally.

- Most public key cryptography uses hybrid encryption, where only a small operation must be done on the HSM, and the rest can be done locally.

  For example, when decrypting a large message, typically, the HSM is only used to decrypt a per-message AES key that is then used to decrypt the rest of the message locally.

  Similarly, for signing, the message to sign is first hashed in-memory using a secure hash algorithm, and only the short hash value is sent to the HSM to be signed.

  In contrast, when using symmetric key algorithms directly with an HSM, the entire message must be streamed to and from the HSM.

How ForgeRock Services Interact with HSMs

ForgeRock services built with Java interact with HSMs through Java PKCS#11 providers.

The PKCS#11 standard defines a cryptographic token interface, a platform-independent API for accessing an HSM, for example. ForgeRock services support the use of the Sun PKCS11 provider.

The Sun PKCS11 provider does not implement every operation and algorithm supported by every HSM. For details on the Sun PKCS11 provider's capabilities, see the JDK Providers Documentation, and the JDK PKCS#11 Reference Guide.

How you configure the JVM to use your HSM depends on the Java environment and on your HSM. ForgeRock services do not implement or manage this configuration, but they do depend on it. Before configuring ForgeRock services to use your HSM, you must therefore configure the JVM's Sun PKCS11 provider to access your HSM. For details on how to configure the Java Sun PKCS11 provider with your HSM, see the documentation for your HSM, and the JDK PKCS#11 Reference Guide.

Sun PKCS11 Provider Hints

The provider configuration depends on your provider.

Compare the following hints for ForgeRock software with the documentation for your HSM:

`name = FooHsm`

The name used to produce the Sun PKCS11 provider instance name for your HSM.

`FooHsm` is just an example.

`CKA_TOKEN = false`

Keys generated using the Java **keytool** command effectively have this set to true. They are permanent keys, to be shared across the deployment.

Set this to false to prevent temporary keys for RSA hybrid encryption used by some platform components from exhausting HSM storage.

`CKA_PRIVATE = true`

The key can only be accessed after authenticating to the HSM.

`CKA_EXTRACTABLE = false`
`CKA_SENSITIVE = true`

`CKA_EXTRACTABLE = false` means the key cannot be extracted *unless it is encrypted*.

`CKA_SENSITIVE = true` means do not let the secret key be extracted out of the HSM. Set this to true for long-term keys.

Only change these settings to back up keys to a different HSM, when you cannot use a proprietary solution. For example, you might change these settings if the HSMs are from different vendors.

`CKA_ENCRYPT = true`

The key can be used to encrypt data.

`CKA_DECRYPT = true`

The key can be used to decrypt data.

`CKA_SIGN = true`

The key can be used to sign data.

`CKA_VERIFY = true`

The key can be used to verify signatures.

`CKA_WRAP = true`

The key can be used to sign data.

`CKA_UNWRAP = true`

The key can be used to unwrap another key.

When Using Keytool

When using the Java **keytool** command to generate or access keys on the HSM, set the following options appropriately:

`-alias` *alias*

If key pair generation fails, use a new *alias* for the next try. This prevents the conflicts where the previous attempt to create a key was only a partial failure, leaving part of key pair using the previous alias.

`-keystore none`

Required setting.

`-providername` *`providerName`*

Required setting.

Prefix `SunPKCS11-` to the name in the configuration. If, in the Sun PKCS11 configuration, `name = FooHSM`, then the *providerName* is `SunPKCS11-FooHSM`.

`-storetype pkcs11`

Required setting.

If necessary, add the following settings:

`-providerClass sun.security.pkcs11.SunPKCS11`

Use this to install the provider dynamically.

`-providerArg` */path/to/config/file.conf*

Use this to install the provider dynamically.

The exact configuration depends on your HSM.

When Java Cannot Find Keys

When keys generated with one Java PKCS#11 provider are later accessed using the Sun PKCS11 provider, the providers may have different naming conventions.

Java's KeyStore abstraction requires that all private key objects have a corresponding Certificate object. SecretKey objects, such as AES or HMAC keys, are excluded from this requirement.

The Sun PKCS11 KeyStore provider loops through all defined private key entries in the HSM (class = private key), and tries to match them up with a corresponding certificate entry (class = certificate) by comparing the `CKA_ID` of the certificate entry to the `CKA_ID` of the private key entry. There may be multiple certificates using the same private key pair The matching process can take several seconds at startup time if you have many keys.

If keys are not found, it is likely that the private key entry `CKA_ID` does not match the certificate entry `CKA_ID`.

HSM Features and ForgeRock Services

This part outlines how some ForgeRock platform features support HSMs.

JWT Encryption Algorithms

JWT Encryption Algorithm	Java Algorithm Equivalent	Supported by Sun PKCS11 Provider?
RSA1_5	RSA/ECB/PKCS1Padding[a]	Yes
RSA-OAEP	RSA/ECB/ OAEPWithSHA-1AndMGF1Padding	No
RSA-OAEP-256	RSA/ECB/ OAEPWithSHA-256AndMGF1Padding	No
ECDH-ES (+A128KW, and so on)[b]	ECDH (KeyAgreement vs. Cipher)	Yes
dir with A128GCM, A192GCM, A256GCM	AES/GCM/NoPadding	Yes
dir with A128CBC-HS256, and so on	AES/CBC/PKCS5Padding + HmacSHA256, and so on	No
A128KW, A192KW, A256KW	AESWrap	No

[a]PKCS#1 version 1.5 padding has known vulnerabilities and should be avoided.

[b] The Sun PKCS11 implementation of ECDH KeyAgreement requires that the derived key be extractable. This is not a security issue, as the derived key is unique to each message, and therefore no more sensitive than the message it is protecting, due to the use of fresh ephemeral keys for each message (ECIES). To ensure that the derived keys are extractable, add the following to the PKCS11 configuration file:

```
attributes(*,CKO_PRIVATE_KEY,CKK_EC) = {
CKA_SIGN = true
CKA_DERIVE = true
CKA_TOKEN = true
}

attributes(generate,CKO_SECRET_KEY,CKK_GENERIC_SECRET) = {
CKA_SENSITIVE = false
CKA_EXTRACTABLE = true
CKA_TOKEN = false
}
```

This also ensures that EC keys generated with the **keytool** command are marked as allowing ECDH key derivation. The derived keys are also marked as `CKA_TOKEN = false`, which ensures that the derived keys are only created as session keys, and automatically deleted when the session ends to prevent filling up the HSM with temporary keys.

JWT Signing Algorithms

JWT Signing Algorithm	Java Algorithm Equivalent	Supported by Sun PKCS11 Provider?
HS256, HS384, HS512	HmacSHA256, HmacSHA384, HmacSHA512	Yes
RS256, RS384, RS512	SHA256WithRSA, SHA384WithRSA, SHA512WithRSA	Yes

JWT Signing Algorithm	Java Algorithm Equivalent	Supported by Sun PKCS11 Provider?
PS256, PS384, PS512	RSASSA-PSS or SHA256WithRSAAndMGF1, and so on	No
ES256, ES384, ES512	SHA256WithECDSA, and so on	Yes
EdDSA	Under development	No

Configure ForgeRock Services to Use an HSM

Once you have configured the Java environment to use your HSM through the Sun PKCS11 Provider, you can configure ForgeRock software to use the HSM:

- AM: See Configuring Secrets, Certificates, and Keys.
- DS: See PKCS#11 Hardware Security Module.
- IDM: See Configuring IDM For a Hardware Security Module (HSM) Device.
- IG: See Secrets, and HsmSecretStore.