How-To Guides

Table of Contents

Adaptive Authentication: How to Improve Security Without Annoying Users . . . 2
- Introduction . . . 2
- How Adaptive Authentication Works . . . 2
- Adaptive Authentication Factors . . . 3

Authentication in a Single‐Page Application via OpenAM using OAuth2 OIDC Protocol . . . 5
- Introduction . . . 5
- OAuth2/OIDC Provider Configuration . . . 6
- React-Based Example SPA Application . . . 9
- Test Solution . . . 11

Authentication modules . . . 14

Captcha module . . . 16
- ReCaptcha Authentication Module . . . 16

Config OpenAM as OAuth2 Service Provider . . . 18

Configuring authentication in OpenAM via Yandex Cloud using the SAML protocol . . . 19
- Introduction . . . 19
- Configuring Yandex Cloud . . . 19
- Configuring OpenAM . . . 22
- Verifying the solution . . . 25
- What’s next . . . 26

How OpenAM Works Simple Explanation . . . 28
- Introduction . . . 28
- OpenAM Ecosystem . . . 28
- How Resource Protection Works . . . 29
- How User Authentication Works . . . 30
- Conclusion . . . 31

How To Run OpenAM in Kubernetes . . . 32
- Preface . . . 32
- Create OpenAM Service . . . 32
- Create OpenAM StatefulSet . . . 33
- OpenAM Setup and Configuration . . . 35
- Add External Access via Ingress . . . 37

How To Setup Active Directory Authentication In OpenAM . . . 40
- Preface . . . 40
- Setup Active Directory User Data Store . . . 40
- Test Data Store and Authentication . . . 41

How to Add Authorization and Protect Your Application With OpenAM and OpenIG Stack . . . 42
- Introduction . . . 42
- Run the Test Application . . . 42

OpenIG Setup . . . 45
Test the Solution . . . 49
How to Create Custom OpenAM DataStore Integration . . . 52
Introduction . . . 52
Custom IdRepo Implementation . . . 52
Service Schema . . . 57
Add Schema to OpenAM Distribution . . . 57
Add Schema to an Exististing OpenAM Deployment . . . 58
IdRepo Plugin in Action . . . 58
Conclusion . . . 60
How to Customise OpenAM . . . 61
Introduction . . . 61
Create Maven Project . . . 61
Run and Debug War File Locally . . . 63
Customisation Example . . . 64
Conclusion . . . 64
Creating a Custom Authentication Module . . . 65
Requirements for Sample Authentication module . . . 65
Prepare your development environment . . . 65
Design your custom authentication module . . . 66
Implement your custom authentication module . . . 67
Compile your custom authentication module . . . 75
Install your custom authentication module . . . 75
Configure your custom Authentication module . . . 76
Test your custom module . . . 77
How to Protect Model Context Protocol (MCP) Servers with OpenAM and OpenIG . . . 78
Introduction . . . 78
Project description . . . 78
Preparing for launch . . . 79
MCP Server . . . 79
Configuring OpenAM . . . 81
Configuring OpenIG . . . 85
Conclusion . . . 94
How to Setup 2FA with Google Authenticator in OpenAM . . . 95
Introduction . . . 95
OpenAM Setup . . . 95
Test Solution . . . 99
How to Setup Active Directory Authentication in a Spring Boot Application . . . 101
Introduction . . . 101
Spring Boot Application Setup . . . 104
Test Solution . . . 105

How to Setup OAuth2 OIDC Federation in OpenAM . . . 108
Introduction . . . 108
OpenAM Instances Installation . . . 108
OAuth2/OIDC Server Setup . . . 110
OAuth2/OIDC Client Setup . . . 112
Test the Solution . . . 115
How to Setup SAMLv2 Federation in OpenAM . . . 117
Introduction . . . 117
OpenAM Instances Installation . . . 117
Identity Provider и Service Provider Configuration . . . 119
Test Account Creation . . . 126
Test the Solution . . . 127
How to Setup Two Factor Authentication in OpenAM . . . 129
Introduction . . . 129
OpenAM Setup . . . 129
Test the Solution . . . 131
How to Setup WebAuthn Authentication in OpenAM . . . 134
Introduction . . . 134
OpenAM Setup . . . 134
Test the Solution . . . 138
How to Start OpenAM and OpenDJ in Separate Docker Containers . . . 142
Requeirements . . . 142
OpenAM Perparation . . . 142
OpenDJ Preparation . . . 142
How to Use Apache Cassandra as User and Session DataStore in OpenAM . . . 148
Introduction . . . 148
Prepare Apache Cassandra . . . 148
OpenAM Setup . . . 149
How to disable XUI by default . . . 153
Legacy UI . . . 153
XUI Interface . . . 153
How to make OpenAM log more verbose . . . 154
How to setup Kerberos Authentication with OpenAM . . . 155
Introduction . . . 155
Prerequisites . . . 155
Windows setup . . . 155
OpenAM Configuration . . . 157
Test Solution . . . 158
Login and Password Authentication . . . 159
Introduction . . . 159
Implementation . . . 159

Pros and Cons . . . 160
Conclusion . . . 160
Login via Gosuslugi OAuth2 Identity Provider . . . 161
Preparation . . . 161
OpenAM Installation . . . 161
OpenAM Configuration . . . 162
Test the solution . . . 165
Login via Facebook OAuth2 Identity Provider . . . 166
Create Facebook Applicaton . . . 166
Setup OpenAM . . . 166
Login via Odnoklassniki OAuth2 Identity Provider . . . 168
Create Odnoklassniki Applicaton . . . 168
Setup OpenAM . . . 168
Login via VK OAuth2 Identity Provider . . . 170
Create Applicaton . . . 170
Setup OpenAM . . . 170
Migrate OpenAM to Apache Cassandra without Single Point of Failure . . . 172
Initial Data Storage Scheme . . . 172
Data Storage Scheme for the Number of Credentials >5 Million . . . 172
OpenAM Monitoring Using Prometheus . . . 176
Preface . . . 176
Setup OpenAM and SNMP Monitoring . . . 176
Setup SNMP Exporter . . . 177
Setup Prometheus . . . 179
OpenAM QR Authentication Module . . . 181
Introduction . . . 181
Process Description . . . 181
Mobile Application Configuration . . . 183
Testing Solution . . . 185
OpenAM and Spring Boot 3 Integration via OIDC OAuth2 Protocol . . . 188
Introduction . . . 188
OpenAM Configuration . . . 188
Spring Boot Application . . . 190
Test the Solution . . . 192
OpenAM and Spring Boot 3 Integration via OpenAM Cookie . . . 195
Introduction . . . 195
Prerequisites . . . 195
OpenAM Configuration . . . 195
Spring Application Configuration . . . 195
Test Solution . . . 199
OpenAM and Spring Boot 3 Integration via SAMLv2 Protocol . . . 202

Introduction . . . 202
OpenAM Configuration . . . 202
Spring Application Configuration . . . 204
Setup Service Provider in OpenAM . . . 207
Test the Solution . . . 207
OpenAM and Zero Trust: Confirming Critical Operations . . . 209
Introduction . . . 209
Solution Design . . . 209
Preparation . . . 209
Demo Application . . . 210
OpenAM Authentication Service Configuration . . . 211
Configure the OpenIG authorization gateway . . . 222
Test the Solution . . . 226
OpenAM vs Keycloak . . . 228
General Information . . . 228
Single Sign On (SSO) and Federation . . . 228
Isolation (realms) . . . 229
Interfaces . . . 230
Authentication Sessions . . . 230
Identity Repositories . . . 230
Audit logging and Monitoring . . . 231
Useful Links . . . 231
Passwordless Authentication Methods . . . 232
Intro . . . 232
Passwordless Authenticaion Methods . . . 232
Conclusion . . . 235
Persistent Cookie module . . . 236
ssoadm service name: **iPlanetAMAuthPersistentCookieService** . . . 238
Quick Start Guide . . . 240
Preparations . . . 240
OpenAM Configuration . . . 240
Apache HTTP Server Configuration . . . 247
SAML Authentication in WordPress via OpenAM . . . 249
Introduction . . . 249
Some terminology . . . 249
Test environment . . . 249
WordPress Configuration . . . 251
OpenAM Setup . . . 254
WordPress OpenAM Identity Provider Configuration . . . 257
Test the Solution . . . 258
SAML Authentication in Yandex Cloud via OpenAM . . . 261

Getting started . . . 261
Setting up Federation in OpenAM . . . 261
Creating and Setting up a Federation in Yandex Cloud Organization . . . 264
Creating a Remote Service Provider в OpenAM . . . 265
Adding Users to Yandex Cloud Organization . . . 266
Testing Authentication . . . 267
SSO Configuration: OpenIG as SAML Service Provider for OpenAM . . . 268
Introduction . . . 268
Preparation . . . 268
Configuring OpenAM . . . 268
OpenIG Setup . . . 270
Test the Solution . . . 274
Conclusion. . . . 275
Setup Tracing with OpenTelemetry and Jaeger . . . 276
Preconditions . . . 276
OpenAM . . . 276
OpenIG . . . 277
Stateful vs Stateless Authentication . . . 278
Preface . . . 278
Stateful Authentication . . . 278
Stateless Authentication . . . 278
Conclusion . . . 279
Bootstrap OpenAM and OpenDJ Docker Containers . . . 280
How to Enable CORS . . . 283
How to Pass Additional System Property to CATALINA_OPTS to OpenAM Docker Container . . . 284
How to Persist OpenAM Docker Container Data Between Restarts . . . 285
Quick OpenAM Docker Configuration From a Command Line . . . 286
Using Google Authenticator with OpenAM . . . 288
Stack Used . . . 288
Installing OpenAM . . . 288
Configuring OpenAM . . . 288
Setting up Google Authenticator. . . . 291
Using Microsoft Authenticator with OpenAM . . . 297
Stack Used . . . 297
Installing OpenAM . . . 297
Configuring OpenAM . . . 297
Setting up Microsoft Authenticator. . . . 300
What is Single Sign On and How does it Work . . . 310
What is Single Sign-On . . . 310
How does Single Sign On Work . . . 310

***Practical tutorials for deploying, integrating, configuring, and extending** OpenAM. Examples may target particular product versions; check the prerequisites in each tutorial.*

Adaptive Authentication: How to Improve Security Without Annoying Users

- Introduction
- How Adaptive Authentication Works
- Adaptive Authentication Factors
- Adaptive Authentication Policy Setup

Introduction

Your applications will be attacked sooner or later. Especially if they have become popular and your users don't particularly care about information security.

In my 10 years of developing authentication systems, I've seen a constant battle between security requirements and a positive user experience. The easier it is for a user to log into a service, the less annoyed they are. On the other hand, the more complex the authentication process, the easier it is for an attacker to break into the system. Is there a compromise possible here?

In my opinion, yes. An adaptive authentication system can recognize a potential attacker and make the authentication process much complex on the fly to prevent hacking.

How Adaptive Authentication Works

If the user used the application before, logged in from the same IP address and used the same device or browser, and used the same device or browser, you can log in the user without asking for credentials.

If the user is trying to login using an unknown device or browser, or user's IP address in another country since the moment of the previous login, the user probably is an attacker.

Attackers come in several types - bots or real people, who use social engineering for attacks. For each type of attacker. For each type of attacker, you have to react differently.

If an attacker is a bot, you must use a method that only humans can pass - captcha. The most common types of captchas are images of text or certain objects. To authenticate, the user must enter the text from the image, or mark certain objects.

An alternative authentication option is one-time-password (OTP) authentication. The random code is sent to the user via SMS, email, or push notification in a mobile application. In that case, an attacker will not be able to access the application, unless he has access to the user's phone or email. This method is more expensive than captcha but can protect against both bots or real people.

If there is a high probability of hacking, authentication should be disabled for IP address or user account.

To figure out if a user is an attacker, during the authentication process you must collect certain data

that can be used to detect attack risk level. This data is called authentication factors.

So what factors can be collected during the authentication process?

Adaptive Authentication Factors

IP address. Using an IP address you can determine several factors:

If the IP address belongs to a cloud provider. Hackers use cloud servers to deploy attackers bots or use a cloud server as a VPN server.

If the IP address is an exit node or anonymous network. Hackers can use anonymous internet, for example, TOR, for attacks to hide their real location.

User geographical location. If the location of current authentication is uncommon for most of your users it could be a hacker attack.

User speed. If the user tries to authenticate from one country and five minutes later from another country, this is likely an attack. A real person can’t cover such distance so fast.

The number of unsuccessful authentication attempts for a certain time. If there are more than 10 unsuccessful authentication attempts in a minute, it is suspected that there is a password brute force attempt.

The number of unsuccessful authentication attempts per time for an IP address. If there are more than 100 unsuccessful authentication attempts per minute, there is a suspicion of an attack attempt.

Unknown or known device or browser. After successful authentication, the service sets a special tag (cookie) for a device or browser. The tag is transmitted at the next login attempts and allows to determine whether the device is known or not.

A time from the start of the authentication process to entering your username and password. A human will not be able to enter credentials and press the "Login" button in a fraction of a second

A time since the last successful authentication. If the user has not logged in for several months, it may be worth requesting additional data from the user.

Adaptive Authentication Policy Setup

Adaptive authentication policies could be determined by security requirements and configured manually. The second option is to use machine learning methods.

Manual Configuration

You determine by yourself which factors are the most potentially dangerous. For example, it is critical to you that when using a new device or browser, the user must additionally confirm his or her identity. When using a known device, you can log the user in without authentication.

Advantages

- Transparency - you determine which security policy is acceptable to you and configure it accordingly
- Easy maintenance
- Low consumption of computing resources

Disadvantages

- To setup policy experience in the subject area is required

Automatic Policy Configuration Using Machine Learning

I intentionally do not use the hype terms "artificial intelligence" or "neural networks" because classification methods or decision trees are sufficient for adaptive authentication. Besides, the result of training neural networks cannot be interpreted. So, you will not be able to determine which authentication factors are most important in attack detection.

Advantages

- Machine learning algorithms can identify factors or combinations of authentication factors that are not obvious to the expert
- The possibility of continuous learning while accumulating authentication data and automatic algorithm adjustment when the logic of the authentication services changes

Disadvantages

- Requires qualified personnel to set up
- Cold start problem - the system cannot work until enough data has been collected for machine learning
- Using multiple models simultaneously increases the consumption of computing resources in the process, especially while training the models.

Authentication in a Single Page Application via OpenAM using OAuth2 OIDC Protocol

Introduction

This article is intended for developers of single-page applications (SPAs) who seek to configure user authentication. The authentication process will utilize the OAuth2/OIDC protocol with PKCE for enhanced security. OpenAM will serve as a designated authentication server.

OpenAM Configuration

Install OpenAM

Let OpenAM hostname is `openam.example.org` Before starting, add the hostname and IP address to the `hosts` file, for example `127.0.0.0.1 openam.example.org`.

On Windows systems, the `hosts` file is located at `C:Windows/System32/drivers/etc/hosts` , on Linux and Mac it is located at `/etc/hosts`.

Next, start the OpenAM Docker container. Run the following command:

```
docker run -h openam.example.org -p 8080:8080 --name openam
openidentityplatform/openam
```

There is no need to configure OpenAM manually via UI. The following command helps to automate the process:

```
 docker exec -w '/usr/openam/ssoconfiguratortools' openam bash -c \
'echo "ACCEPT_LICENSES=true
SERVER_URL=http://openam.example.org:8080
DEPLOYMENT_URI=/$OPENAM_PATH
BASE_DIR=$OPENAM_DATA_DIR
locale=en_US
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=passw0rd
AMLDAPUSERPASSWD=p@passw0rd
COOKIE_DOMAIN=example.org
ACCEPT_LICENSES=true
DATA_STORE=embedded
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=openam.example.org
DIRECTORY_PORT=50389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1689
ROOT_SUFFIX=dc=openam,dc=example,dc=org
```

```
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=passw0rd" > conf.file && java -jar openam-configurator-tool*.jar
--file conf.file'
```

After successful configuration, you can proceed with further customization.

OAuth2/OIDC Provider Configuration

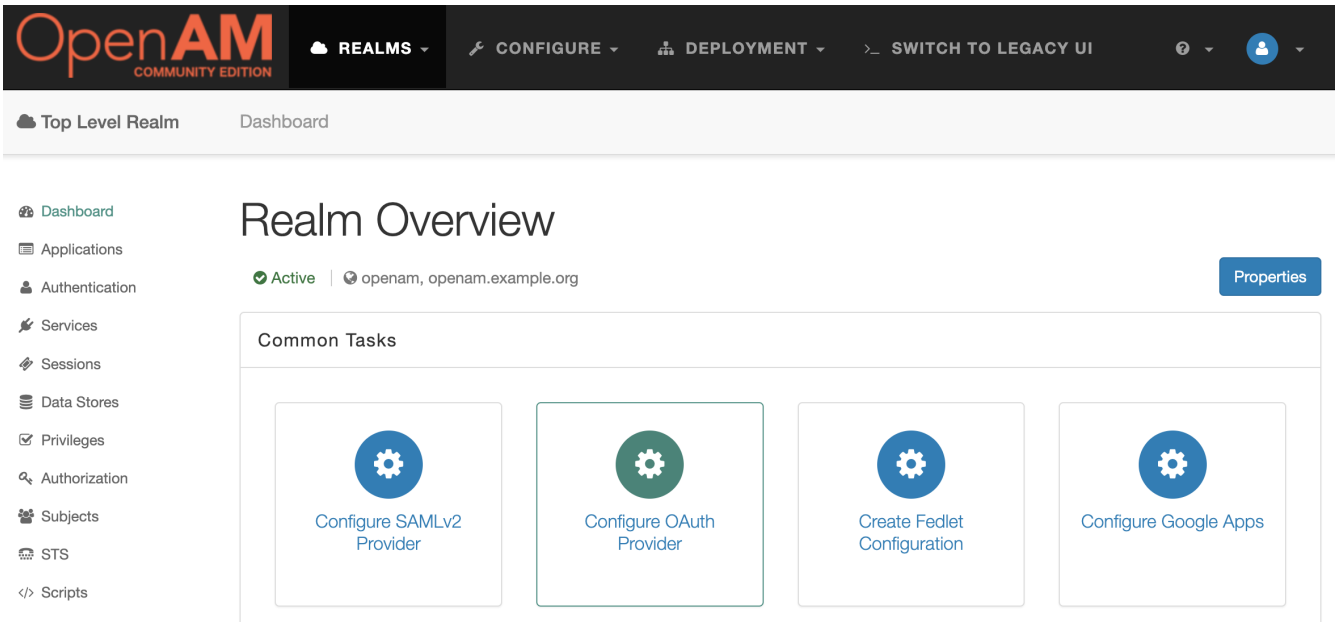
Go to the OpenAM administrator console at

http://openam.example.org:8080/openam/XUI/#login/

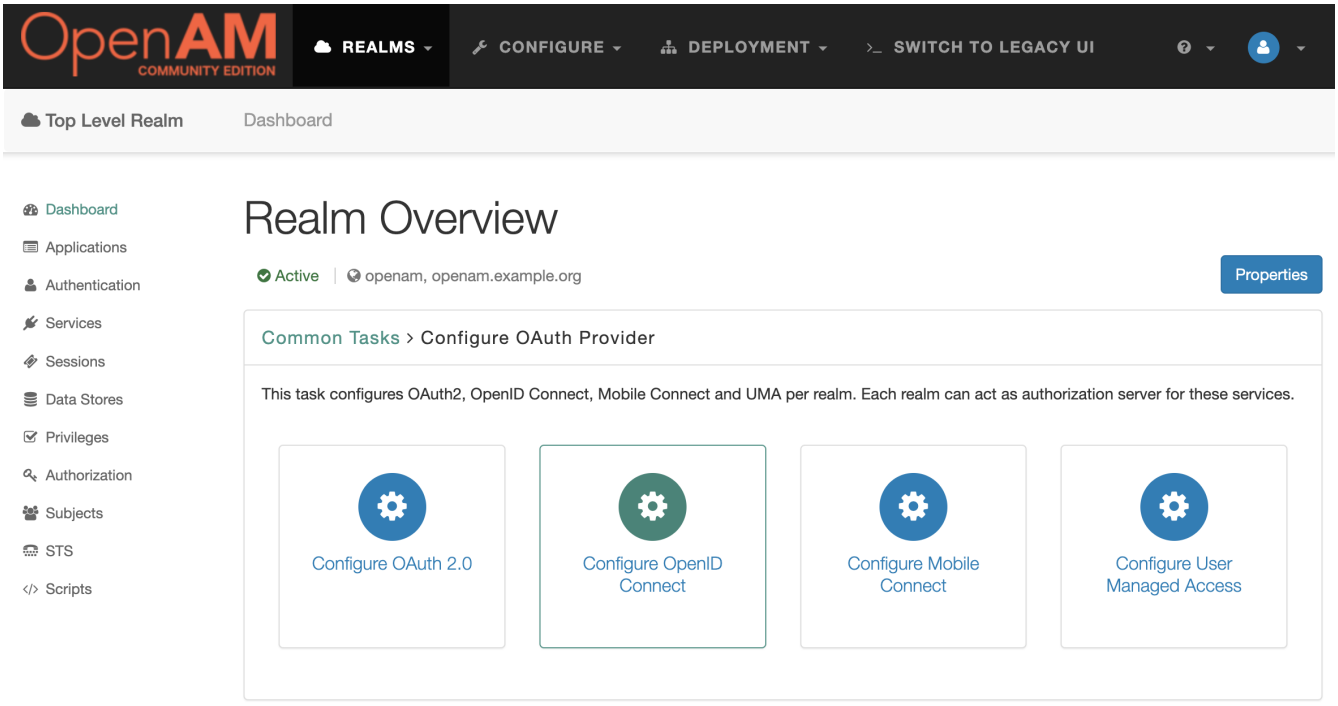
In the login field enter the value `amadmin`, in the password field enter the value from the `ADMIN_PWD` parameter of the setup command, in this case `passw0rd`.

OAuth2/OIDC Configuration

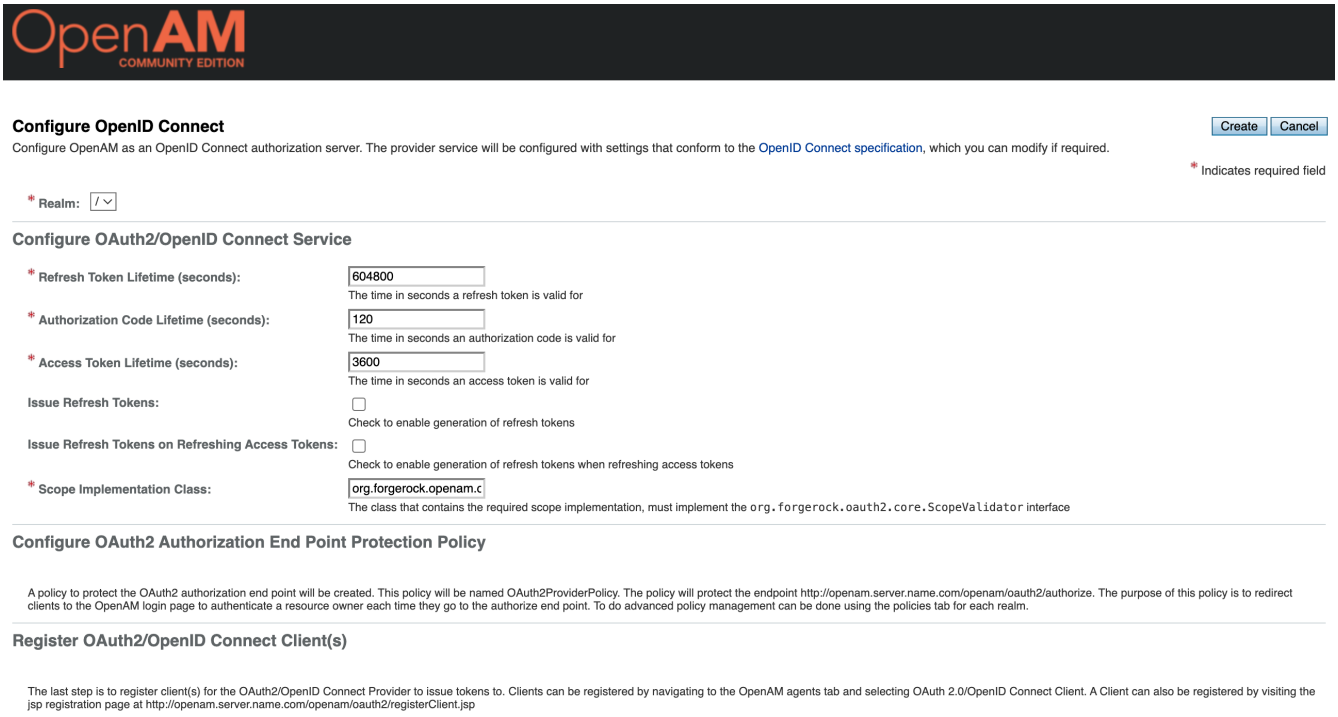
Select the desired realm. In the Dashboard section, click the Configure OAuth Provider item.



Next, Configure OpenID Connect



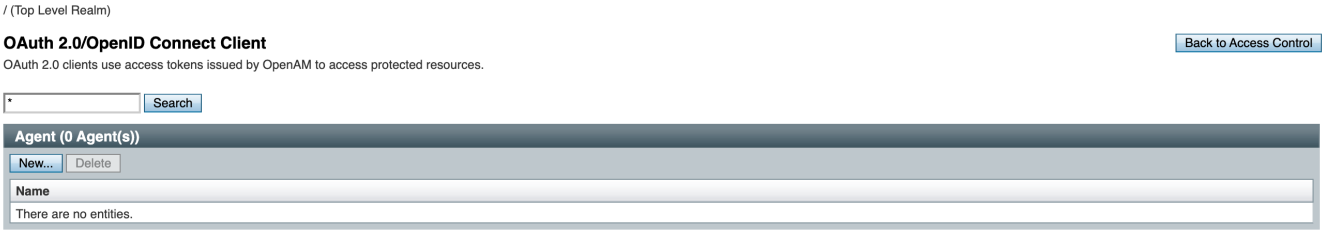
In the opened form, leave all settings unchanged and click the `Create` button



Now let's create an OAuth2/OIDC client that will use SPA application for authentication.

Go to the admin console, select the required realm, select Applications and then OAuth 2.0 from the left menu

In the Agents table click the `New` button

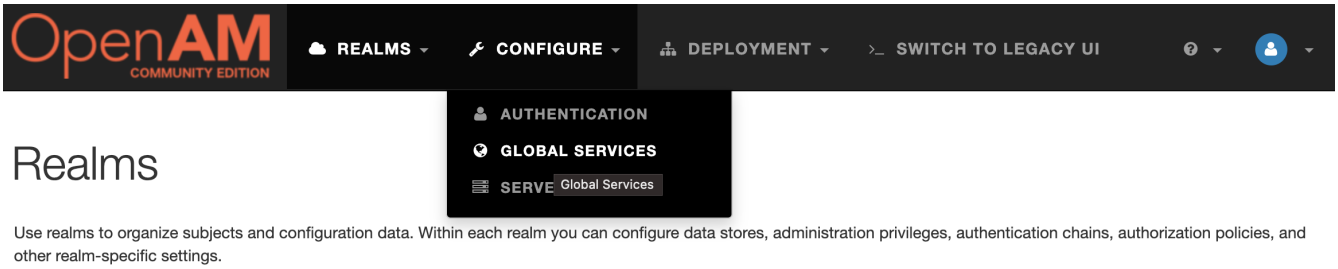


- Enter the Name (client_id) `test_client_id` and Password (client_secret) `changeit` of the new application
- Open the application settings
- Set Client type to Public
- Add the URI of your SPA application to the list of Redirection URIs. In our case, it will be http://localhost:5173/
- Add the `openid` value to the scope list, this is to get the user ID from the returned `id_token` object.
- Token Endpoint Authentication Method, set `client_secret_post`.

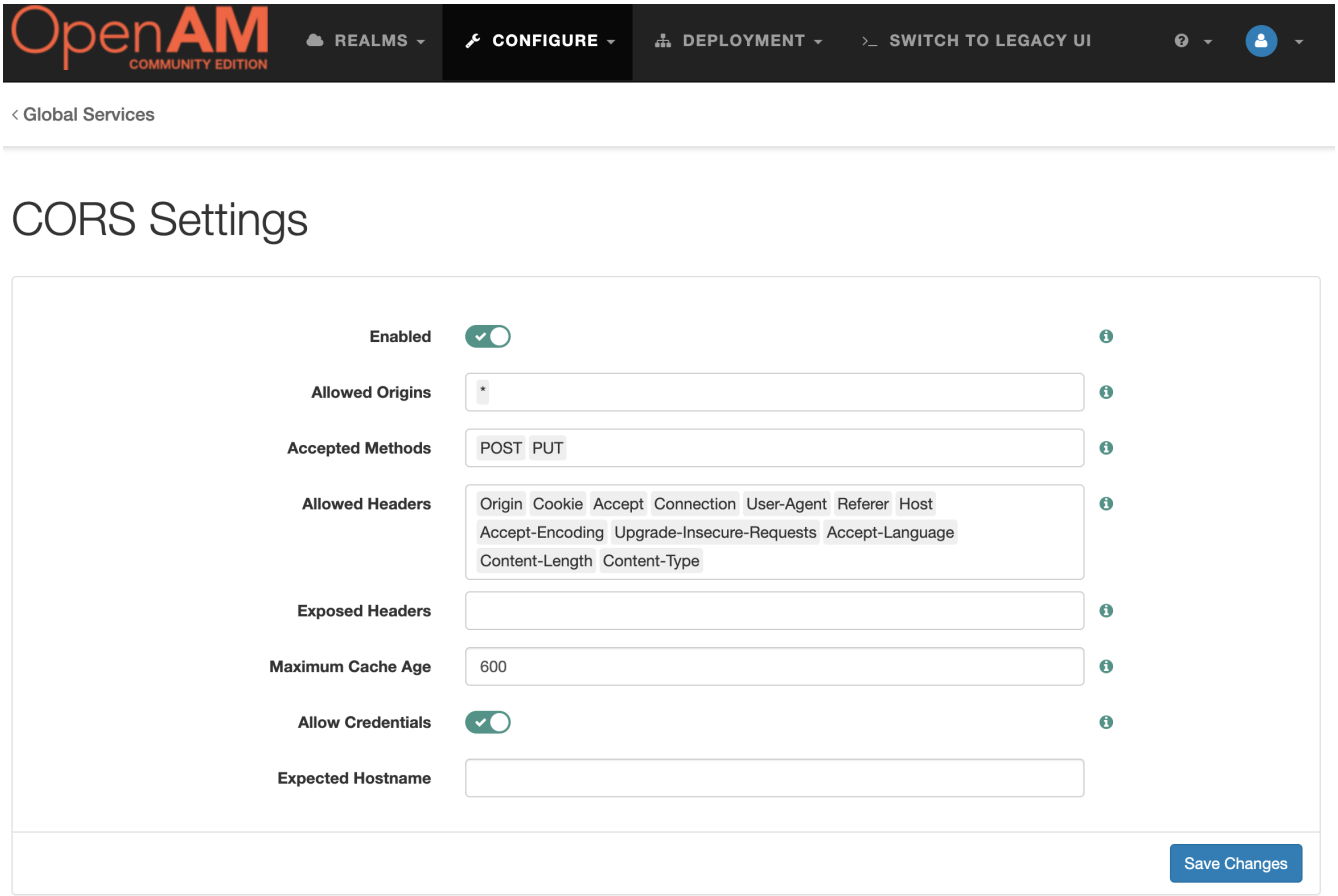
CORS Setup

SPA performs cross-domain queries to get access_token and id_token. To prevent these requests from being blocked by the browser, you need to enable CORS support in OpenAM.

Open the administrator console. In the top menu, select Configure → Global Services.



Next, go to CORS Settings and enable CORS support



Pres the `Save Changes` button.

React-Based Example SPA Application

As an example, we will use a React application. For simplicity, we will not validate the state parameter, verify the correctness of the signature of the returned id_token, and so on. However, in a production environment, we highly recommend doing so.

To create a new application, execute the following command in the console:

```
npm create vite@latest react-openam-example -- --template react
```

Add the CryptoJS library to the dependencies. It will be needed for `code_challenge` generation.

```
cd react-openam-example
npm install crypto-js
```

Replace the contents of the `react-openam-example/src/App.jsx` file with the following code:

```
import { useEffect, useState } from 'react'

import CryptoJS from 'crypto-js';
```

```
import './App.css'

const OPENAM_URL = "http://openam.example.org:8080/openam";
const OAUTH2_ENDPOINT = OPENAM_URL + "/oauth2";
const OAUTH2_AUTHORIZE_ENDPOINT = OAUTH2_ENDPOINT + "/authorize";
const OAUTH2_TOKEN_ENDPOINT = OAUTH2_ENDPOINT + "/access_token";
const CLIENT_ID = "test_client";
const SCOPE = "openid";

function App() {

  const [user, setUser] = useState("");

  //TODO should be randomly generated, saved and then restored in production
evironment
  const codeVerifier = "a116cb8c-5a1e-4918-a164-255ae3d8f1b1";

  useEffect(() => {
    const params = new URLSearchParams(window.location.search)
    const code = params.get('code')
    if(!code) {
      return;
    }
    getToken(code)
  }, [])

  const getToken = async (code) => {
    const resp = await fetch(OAUTH2_TOKEN_ENDPOINT, {
      method: "POST",
      mode: "cors",
      cache: "no-cache",
      credentials: "include",
      headers: {'content-type': 'application/x-www-form-urlencoded'},
      redirect: "follow",
      referrerPolicy: "no-referrer",
      body: new URLSearchParams({
        grant_type: 'authorization_code',
        client_id: CLIENT_ID,
        code_verifier: codeVerifier,
        code: code,
        redirect_uri: window.location.origin
      }),
    });
    if(resp.ok) {
      const accessToken = await resp.json()
      //TODO verify id_token signature
      const idToken = accessToken['id_token'];
      const parts = idToken.split('.')
      const payload = parts[1];
      const jsonPayload = JSON.parse(atob(payload));
      const sub = jsonPayload["sub"]
```

```
      setUser(sub)
      console.log(sub, "authenticated")
    } else {
      console.log(resp.status)
    }
  }


  const authOpenAM = () => {
    const state = "state";
    const codeChallenge = CryptoJS.SHA256(codeVerifier).toString(CryptoJS.enc
.Base64url);
    console.log(codeChallenge);
    const queryString = "?redirect_uri=" + encodeURIComponent(window.location.origin)
+
    "&client_id=" + CLIENT_ID +
    "&response_type=code" +
    "&state=" + state +
    "&scope=" + encodeURIComponent(SCOPE) +
    "&code_challenge=" + codeChallenge +
    "&code_challenge_method=S256";
    window.location = OAUTH2_AUTHORIZE_ENDPOINT + queryString;
  }

  const getComponent = () => {
    if (!user) {
      return <>
      <div>
        <h1>Not authenticated</h1>
      </div>
      <button onClick={authOpenAM}>Login</button>
    </>
    } else {
      return <h1>User {user} authenticated</h1>
    }
  }
  return getComponent()
}

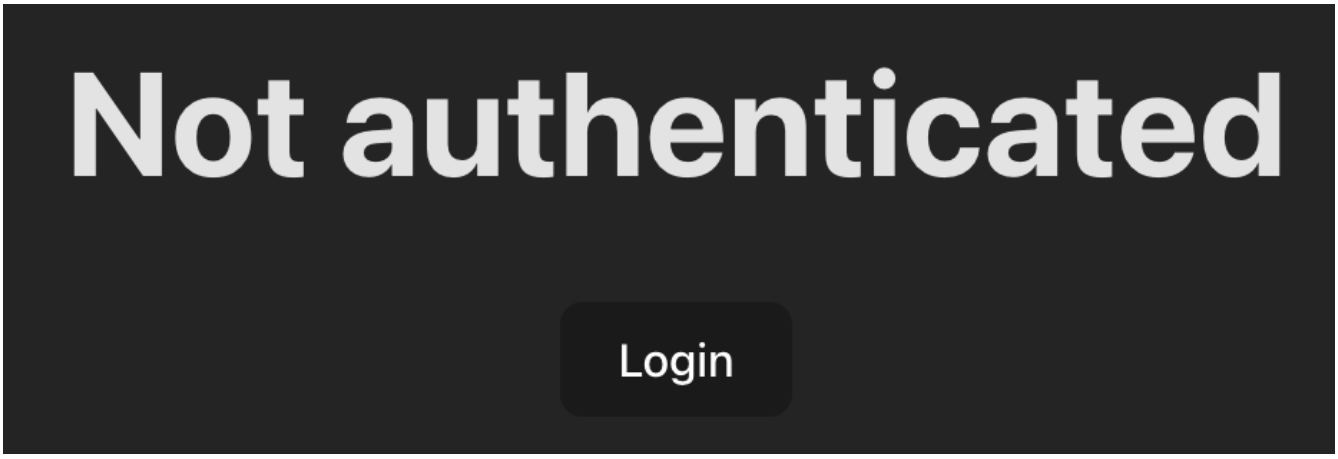
export default App
```

Test Solution

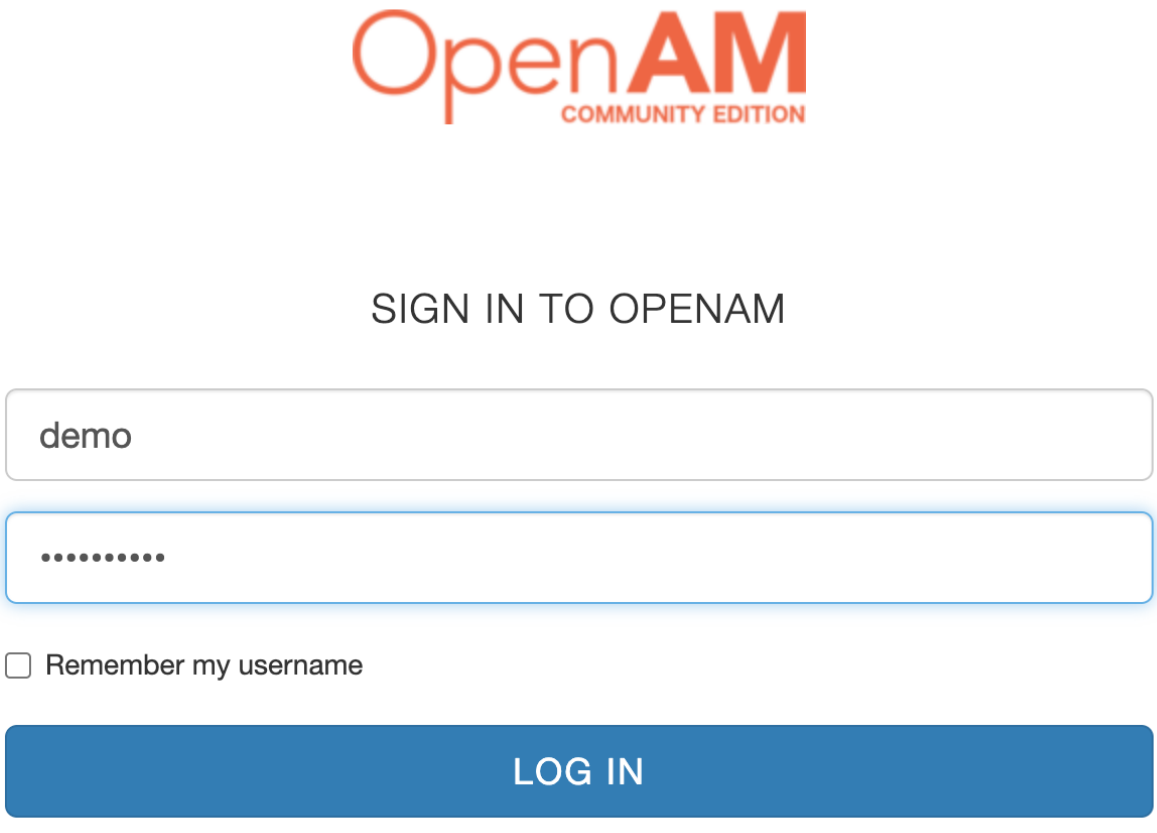
Start the SPA with the following command:

```
npm run dev
```

Open the application in your browser by URL http://localhost:5173/

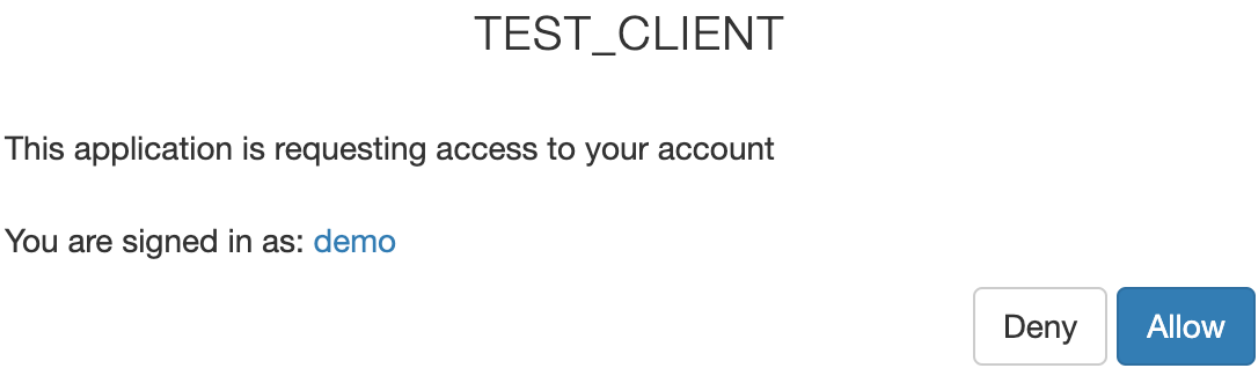


Click the `Login` button. You will be redirected to OpenAM authentication. Enter the user login `demo` and password `changeit`.

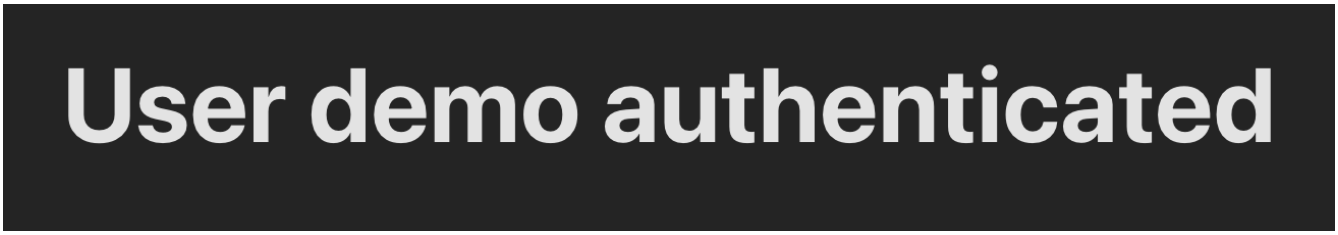


Confirm consent to access the data





After that, the browser will redirect back to the SPA application and successfully authenticate the user. If everything is configured correctly, the SPA application will display a message about successful authentication:



Authentication modules

- Active Directory module
- Adaptive Risk module
- Anonymous module
- Captcha module
- Certificate module
- Check channel over send and verify one time code module
- Confirmation module
- Data Store module
- Device Id (Save) module
- Federation module
- Authenticator (OATH) module
- Authenticator (Push) module
- Authenticator (Push) Registration module
- HOTP module
- HTTP Basic module
- HTTP Header Enrichment module
- JDBC module
- LDAP module
- Membership module
- NTLM module
- OATH module
- OAuth 2.0 / OpenID Connect module
- OAuth-2 (auth) module
- OAuth-2 (sso) module
- OAuth-2 (token) module
- OpenID Connect id_token bearer module
- Persistent Cookie module
- RADIUS module
- reCaptcha module
- SAE module
- SAML2 module
- Windows Desktop SSO module
- Windows NT module

- WSSAuth module

How-to Write a custom authentication module

Captcha module

ReCaptcha Authentication Module

ReCaptcha authentication module allows use Google ReCaptcha to make sure that authenticating user is a real person and not a Robot.

This module included in OpenAM distribution since 14.1.1 version

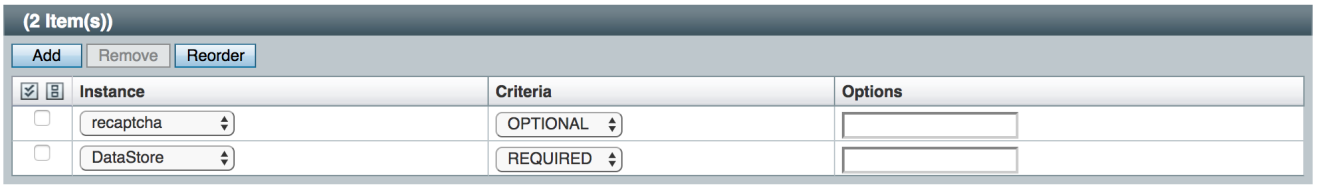
To use this module, at first you should receive recevie ReCaptcha's **key** and **secret key** from **Google**.

Setup Authentication Module Gobally

To configure the reCaptcha module globally in the AM console, navigate to Configuration > Authentication, and then, click **ReCaptcha**. Module settings are shown on the picture below:



Setup Authentication Chain



In this picture, at first, user solves recaptcha, and then recaptcha module allows him to authenticate in DataStore authentication module. As there is no Pricipal at authentication chain start, if user solves ReCaptcha, it returns ignore and skips user to DataStore authentication module. If users does not solve ReCaptcha, authentication starts over.

ssoadm service name: sunAMAuthReCaptchaService

Key

Specifies the key, obtained from Google in ReCaptcha administartion console

ssoadm attribute: **org.openidentityplatform.openam.authentication.modules.recaptcha.ReCaptcha.key**

Secret Key

Specifies the secret key, obtained from Google in ReCaptcha administartion console

ssoadm attribute: **org.openidentityplatform.openam.authentication.modules.recaptcha.ReCaptcha.secret**

ReCaptcha JavaScript URL

Google’s ReCaptcha JavaScript library URL

ssoadm attribute: **org.openidentityplatform.openam.authentication.modules.recaptcha.ReCaptcha.jsUrl**

ReCaptcha Verify URL

Google’s ReCaptcha verification endpoint

ssoadm attribute: **org.openidentityplatform.openam.authentication.modules.recaptcha.ReCaptcha.verifyUrl**

ReCaptcha Verify URL connect timeout

Google’s ReCaptcha verification endpoint connect timeout (default 1500 ms)

ssoadm attribute: **org.openidentityplatform.openam.authentication.modules.recaptcha.ReCaptcha.connect.timeout**

ReCaptcha Verify URL read timeout

Google’s ReCaptcha verification endpoint read timeout (default 2500 ms)

ssoadm attribute: **org.openidentityplatform.openam.authentication.modules.recaptcha.ReCaptcha.read.timeout**

Config OpenAM as OAuth2 Service Provider

- Login via Facebook OAuth2 Identity Provider
- Login via MSLiveID OAuth2 Identity Provider
- Login via VK OAuth2 Identity Provider
- Login via Odnoklassniki OAuth2 Identity Provider
- Login via "Госуслуги РФ" OAuth2 Identity Provider

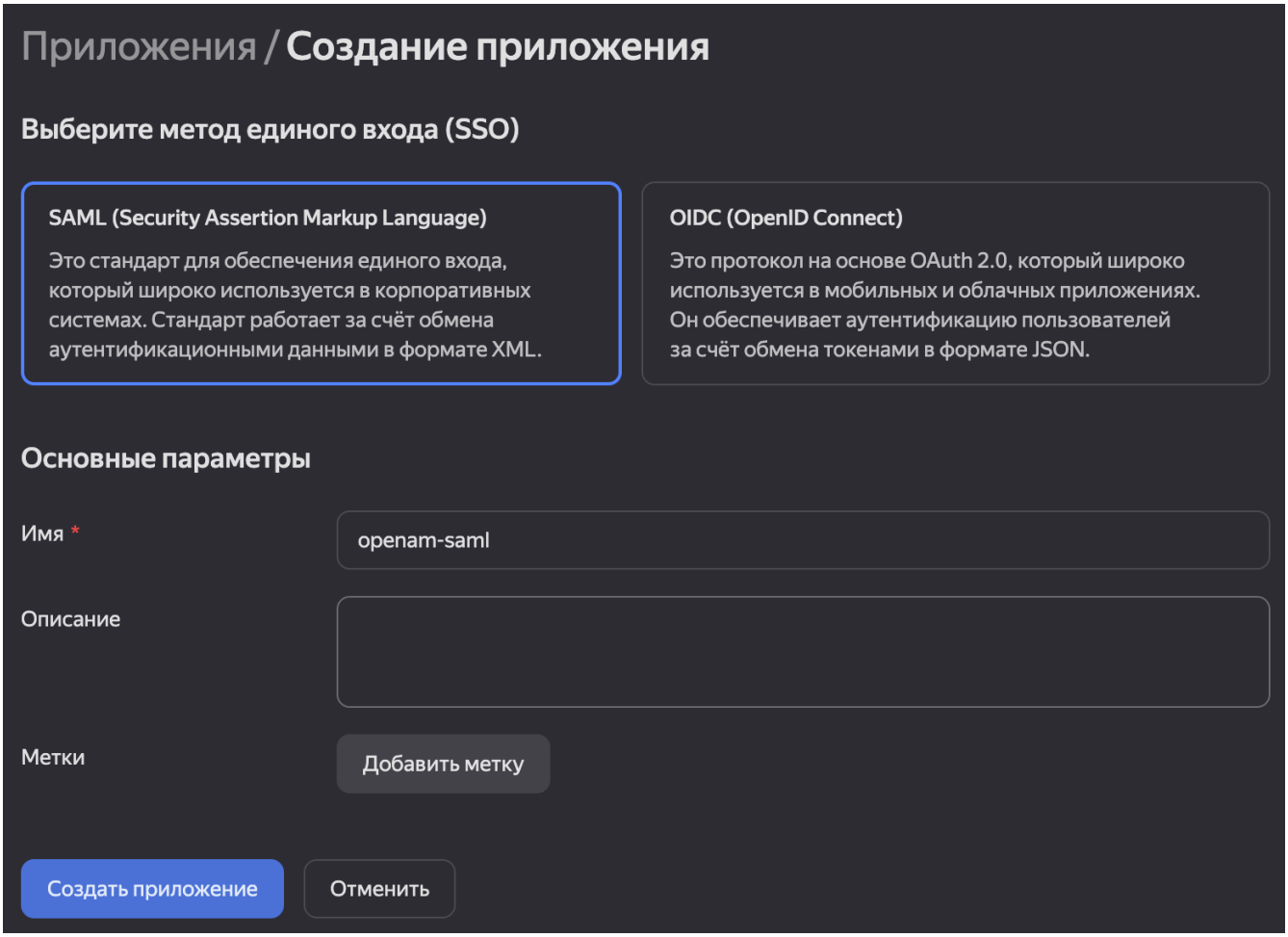
Configuring authentication in OpenAM via Yandex Cloud using the SAML protocol

Introduction

In this article, we will configure authentication in OpenAM using user accounts via Yandex Cloud. This will allow you to configure authentication in your corporate applications via OpenAM using Yandex Cloud as the Identity Provider (IdP) and OpenAM as the Service Provider (SP).

Configuring Yandex Cloud

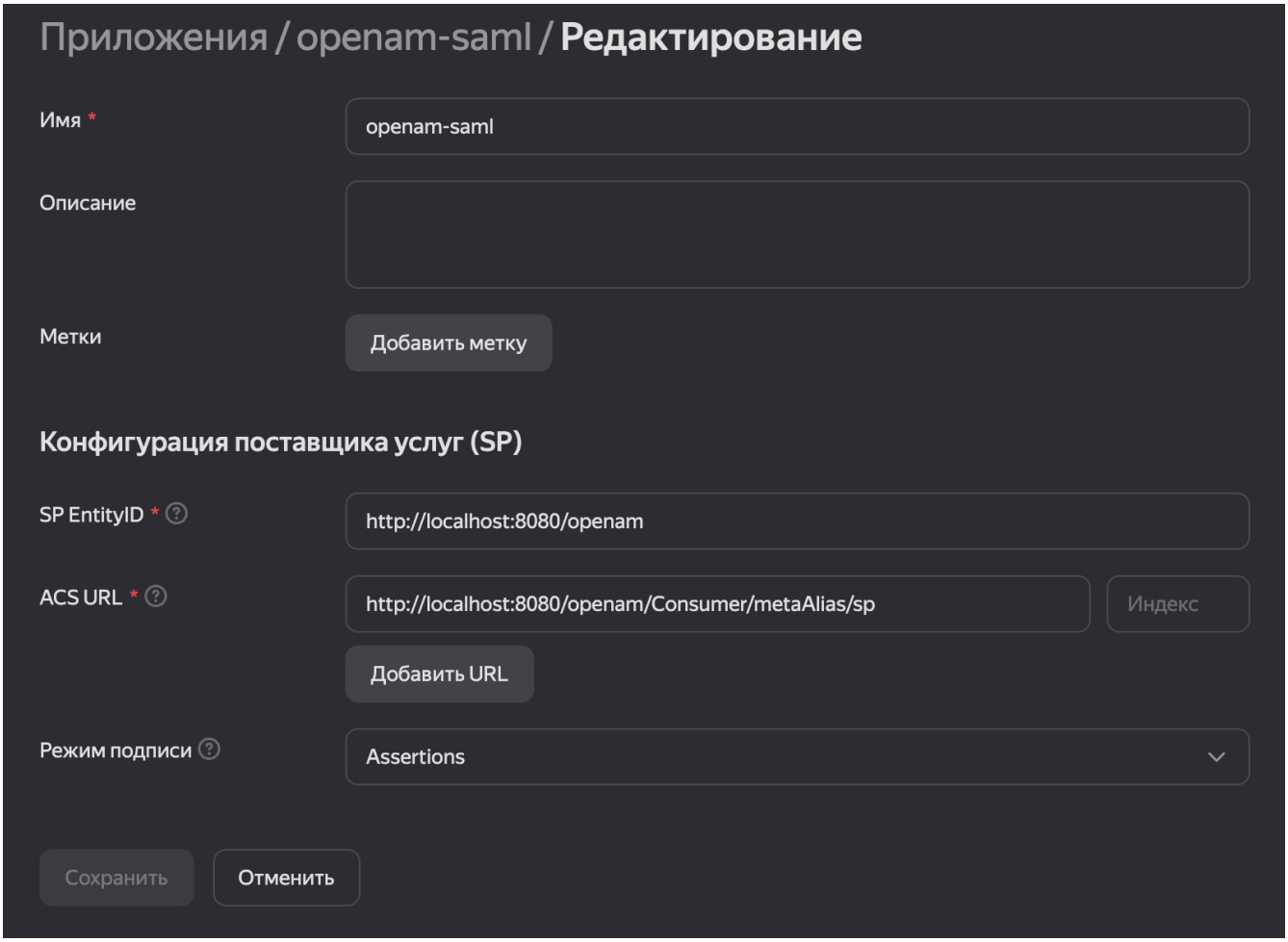
1. Go to the Yandex Cloud Organization service.
2. Open the **Identity Hub** tab.
3. In the left pane, select the Applications section.
4. Click the **Create Application** button
5. Select the SAML (Security Assertion Markup Language) application type
6. Enter the application name, for example, **openam-saml**
7. Click the **Create Application** button



After creating the application, open its settings and click the **Download Certificate** button. We will need it for further configuration of OpenAM.

Service Provider Configuration

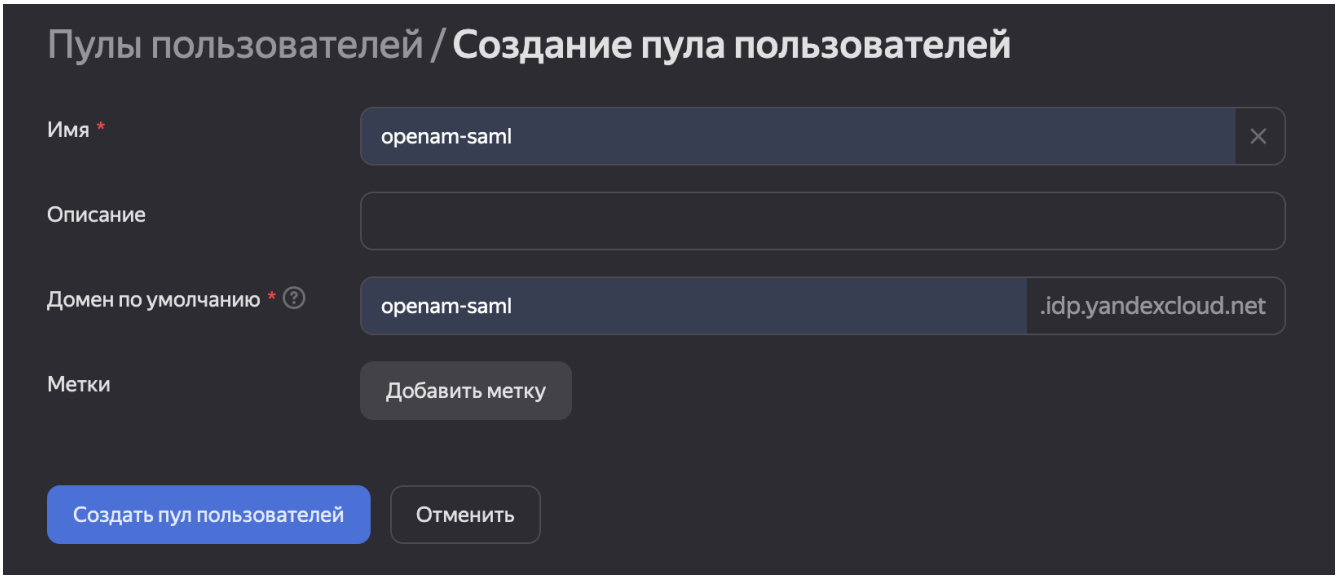
1. Open the Yandex settings for the **openam-saml** application and configure the settings:
2. **SP Entity ID:** `http://localhost:8080/openam`
3. **ACS URL:** `http://localhost:8080/openam/Consumer/metaAlias/sp`
4. Save the changes.



Adding users

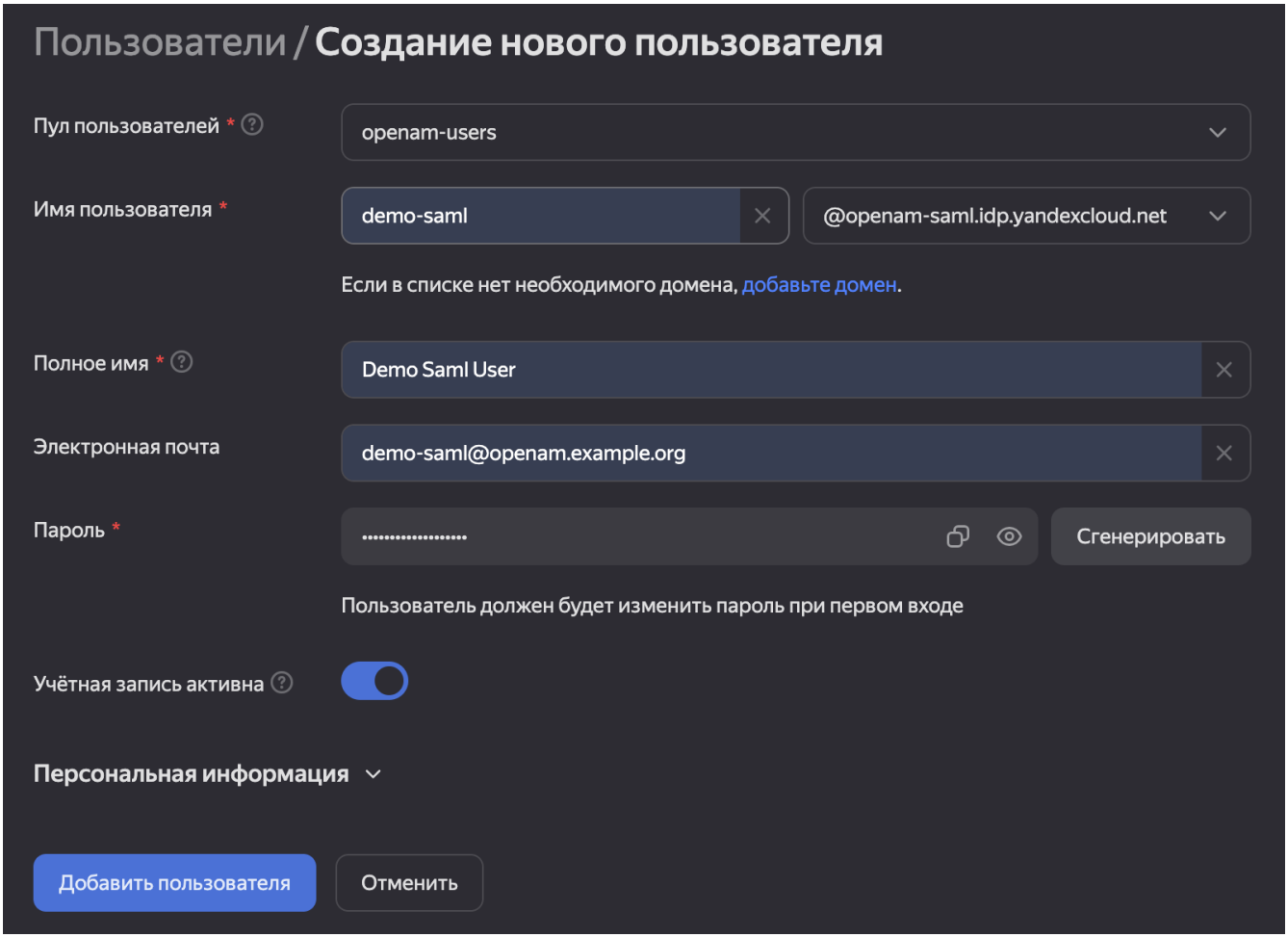
Add a user pool:

1. In the **Identity Hub** section, select **User Pools** from the menu on the left.
2. Click the **Create User Pool** button.
3. Enter the pool details and click **Create User Pool**.



Add a user to the pool

1. In the **Identity Hub** section of the left-hand menu, select **Users**
2. Click the **Add User** button
3. In the pop-up menu, select **Create New User**
4. Remember the password, as you will need it to log in to OpenAM
5. Enter the user details and click **Add User**



Configuring OpenAM

Installing OpenAM

For simplicity, deploy OpenAM in a Docker container with the command

```
docker run -p 8080:8080 --name openam openidentityplatform/openam
```

And perform the initial setup

```
docker exec -w '/usr/openam/ssoconfiguratortools' openam bash -c \
'echo "ACCEPT_LICENSES=true
SERVER_URL=http://localhost:8080
DEPLOYMENT_URI=/$OPENAM_PATH
BASE_DIR=$OPENAM_DATA_DIR
locale=en_US
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=passw0rd
AMLDAPUSERPASSWD=p@passw0rd
COOKIE_DOMAIN=localhost
ACCEPT_LICENSES=true
DATA_STORE=embedded
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=localhost
DIRECTORY_PORT=50389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1689
ROOT_SUFFIX=dc=openam,dc=example,dc=org
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=passw0rd" > conf.file && java -jar openam-configurator-tool*.jar
--file conf.file'
```

Add the certificate you downloaded earlier for the Yandex Cloud application you created to the OpenAM keystore.

To do this, copy the certificate to the container.

```
docker cp openam-saml.cer openam:/usr/openam/config/openam
```

The password for the keystore is located in the file `/usr/openam/config/openam/.storepass`.

You can view it using the command

```
docker exec openam bash -c 'cat /usr/openam/config/openam/.storepass'
```

Import the certificate into the OpenAM keystore

```
docker exec -it -w '/usr/openam/config/openam' openam bash -c 'keytool -importcert \
        -alias "yandex-cloud-cert" \
        -keystore keystore.jceks \
        -storetype JCEKS \
        -file openam-saml.cer'
```

Enter the password and confirm that the certificate is trusted.

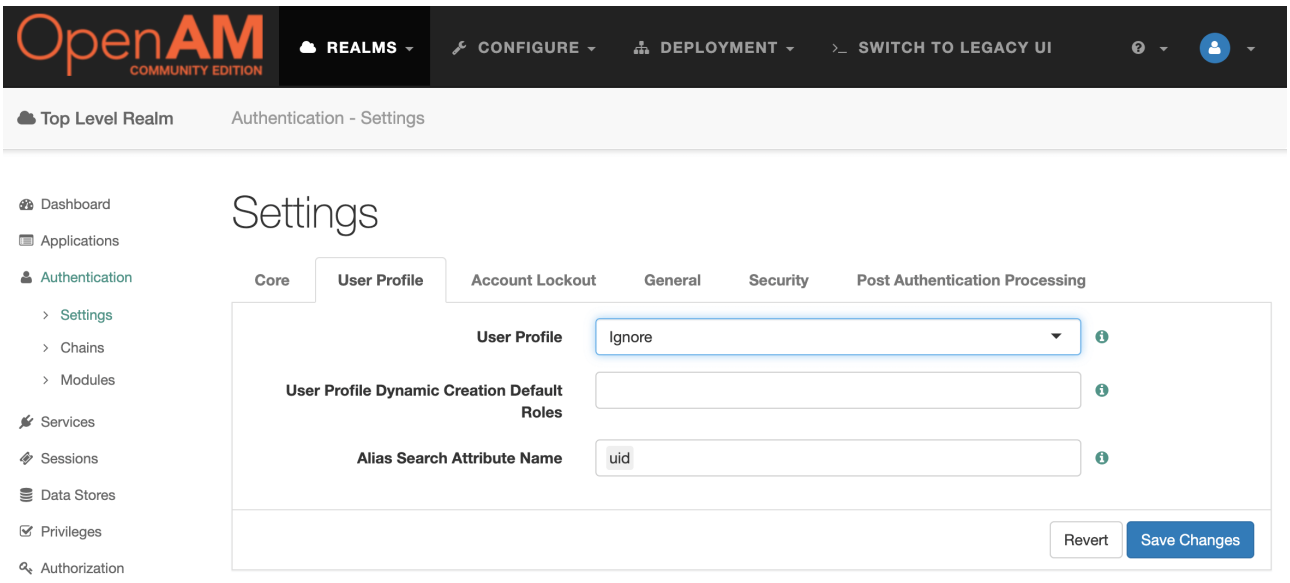
Restart the OpenAM container.

```
docker restart openam
```

Realm Configuration

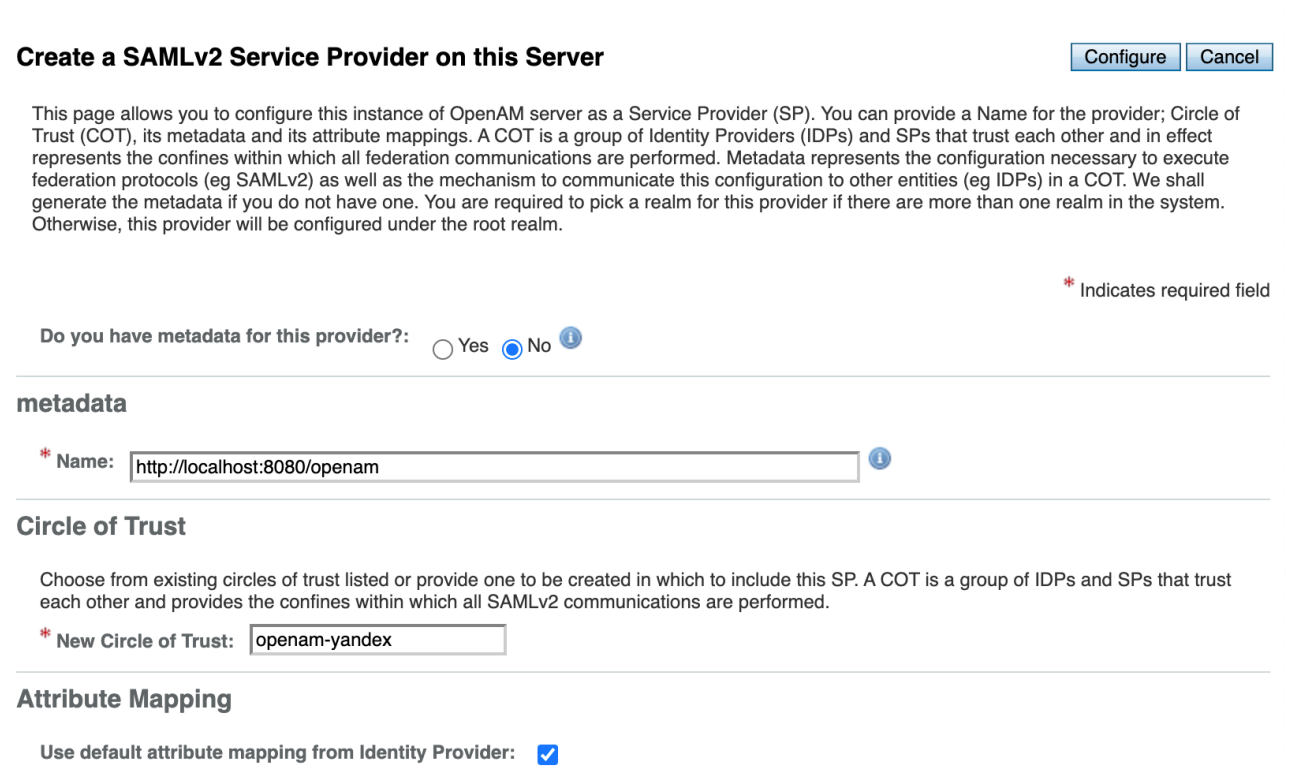
Log in to the administrator console at http://localhost:8080/openam. Use the login `amadmin` and password `passw0rd` respectively.

1. Open **Top Level Realm.**
2. In the menu on the left, go to **Authentication** → **Settings**.
3. Go to the **User Profile** tab and set the **User Profile** setting to **Ignore**.
4. Click **Save Changes.**

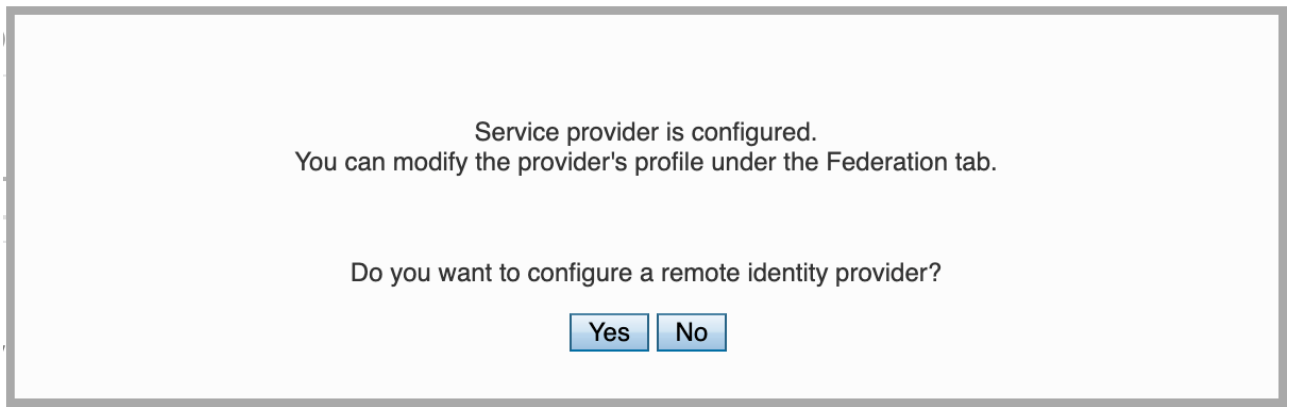


Configuring the Service Provider

1. In the administrator console, select **Top Level Realm**
2. On the **Common Tasks** panel, click **Configure SAMLv2 Provider**
3. Next, click **Create Hosted Service Provider**
4. Enter any name for **Circle Of Trust**, for example, `openam-yandex`, and click **Configure.**

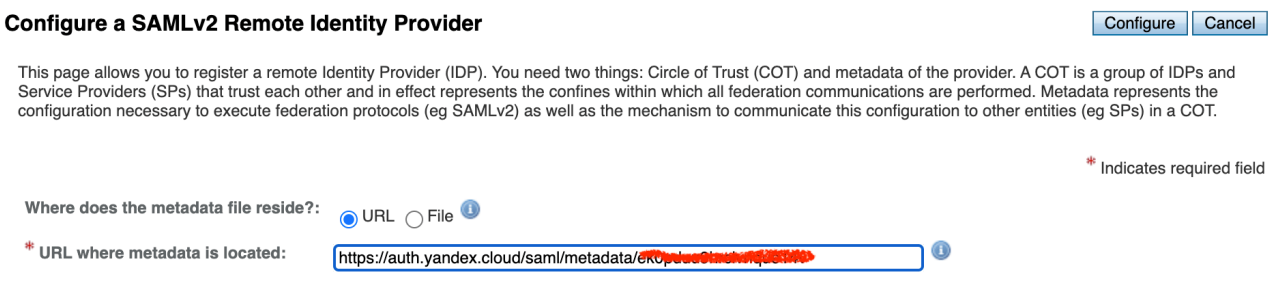


5. OpenAM will prompt you to configure Remote Identity Provider. Click Yes.



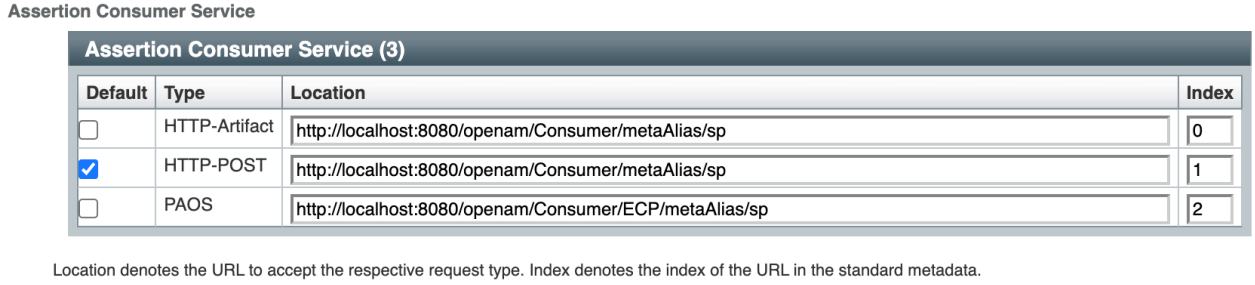
Configuring the Identity Provider

1. Enter the metadata URL from the Yandex application settings and click Configure.



2. Reopen Top Level Realm
3. In the left pane, navigate to **Applications → SAML 2.0**

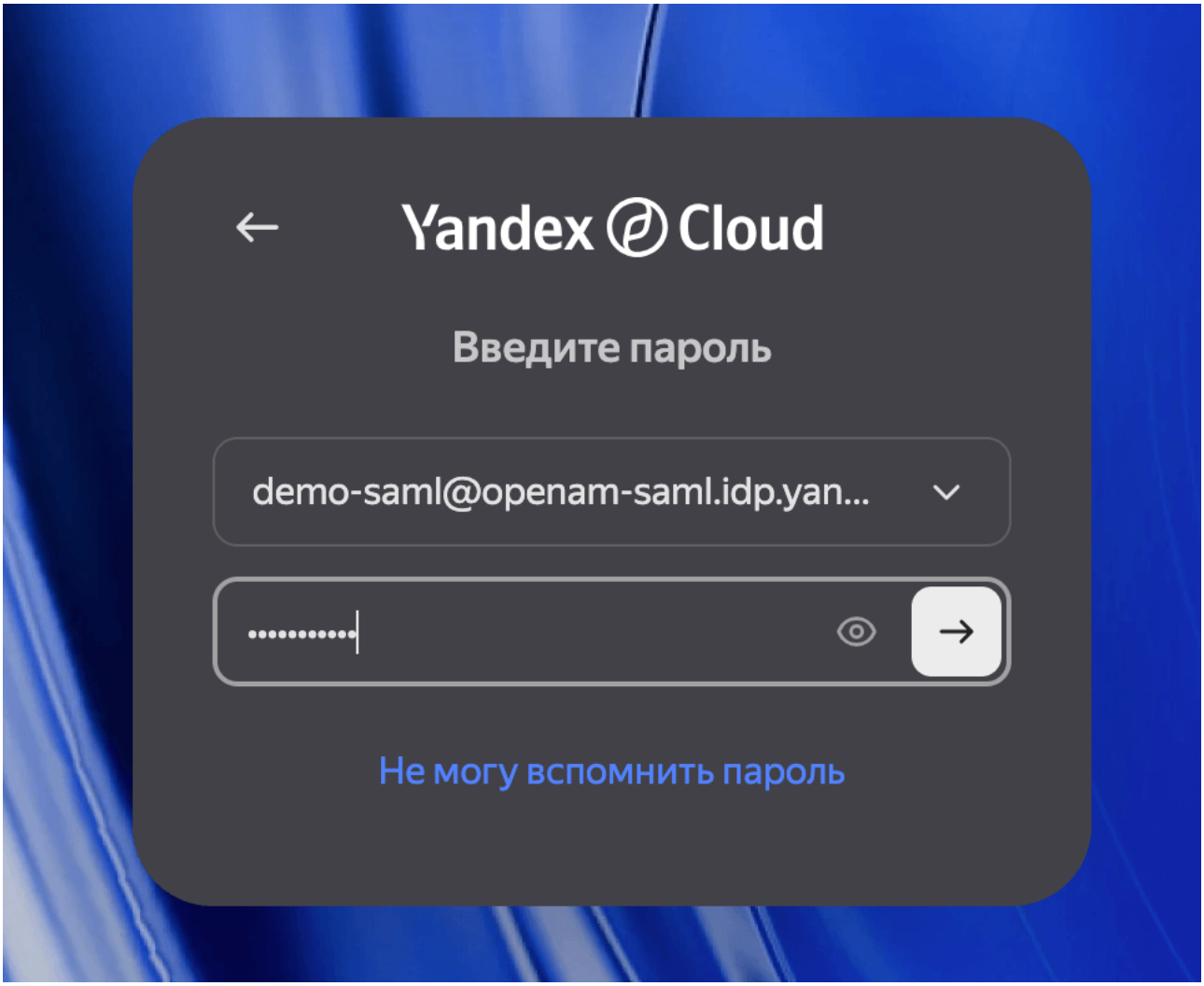
4. In the **Entity Providers** list, open **http://localhost:8080/openam**
5. On the **Assertion Content** tab, find the **Authentication Context** section
6. Set the **Default Authentication Context** setting to `Password`
7. In the **Authentication Context** table, select the values `Password` and `Password Protected Password`
8. Click **Save**
9. Go to the **Assertion Processing** tab
10. In the **Attribute Mapper** section, set the **Attribute Map** setting to `emailaddress=mail`
11. In the **Auto Federation** section, select the Enabled checkbox and set Attribute to `emailaddress`.
12. Click **Save**.
13. Go to the **Services** tab.
14. In the **SP Service Attributes** section, in the **Assertion Consumer Service** table, select `HTTP-POST`.



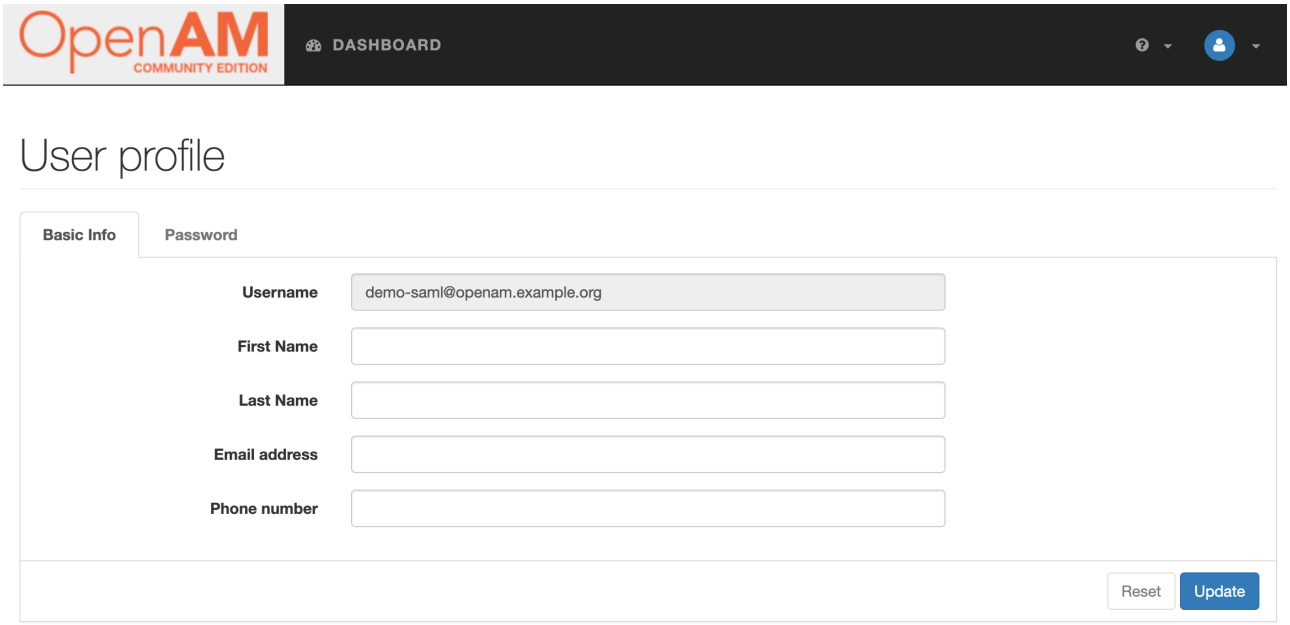
15. Click **Save**

Verifying the solution

1. Exit the admin console, Yandex console, or open your browser in incognito mode.
2. Follow the authentication link: http://localhost:8080/openam/spssoinit?metaAlias=/sp&idpEntityID=https%3A%2F%2Fauth.yandex.cloud%2Fsaml%2Fek0pduu9hrclvnque14v&RelayState=http%3A%2F%2Flocalhost%3A8080%2Fopenam
3. The Yandex Cloud authentication window will open.
4. In the email field, enter your user ID: `demo-saml@openam-saml.idp.yandexcloud.net` and click →
5. In the password field, enter the corresponding password for your account
6. Click →



7. After successful authentication, you will be redirected to the OpenAM console with your Yandex Cloud credentials.



What's next

For production use, OpenAM must be deployed using a secure SSL connection, for example on a

host and using an FQDN, such as https://openam.example.org/openam.

Next, you can use the OpenIG authorization gateway to set up single sign-on (SSO) for your applications.

For more details on configuring OpenAM and OpenIG, please refer to the documentation at OpenAM documentation and https://doc.openidentityplatform.org/openig.

How OpenAM Works Simple Explanation

- Introduction
- OpenAM Ecosystem
- How Resource Protection Works
- How User Authentication Works
- Conclusion

Introduction

It's not obvious to understand how to setup Access Management and (Single Sign On) SSO with OpenAM at glance. There's an explanation of how OpenAM SSO and authentication works in the article below.

OpenAM Ecosystem

There are several components in the Open Identity Platform in addition to OpenAM to setup Access Management system.

Components

OpenAM

The core component of the authentication management system. It is responsible for the authentication process, user management, session, and access policy management.

Configuration DataStore

Stores OpenAM configuration data - general settings, realms, authentication modules and so on. For storage, its configuration data OpenAM uses OpenDJ - open source directory server, the part of Open Identity Platform.

Web Agent

It could be a web server plugin or a standalone web application. Web agent works as PEP (Policy Enforcement Point). Web server plugin should be installed on the server, which process requests to protected resources. Web Agent decides whether an authenticated user could access to a certain resource or not.

Open Identity Platform provides Web Agent as a standalone web application - OpenIG and some web server plugins - for Microsoft Internet Information Services, Apache Tomcat, and Apache HTTP Server.

User Data Store

Storage for the different kinds of identities, which OpenAM uses for authentication. By default,

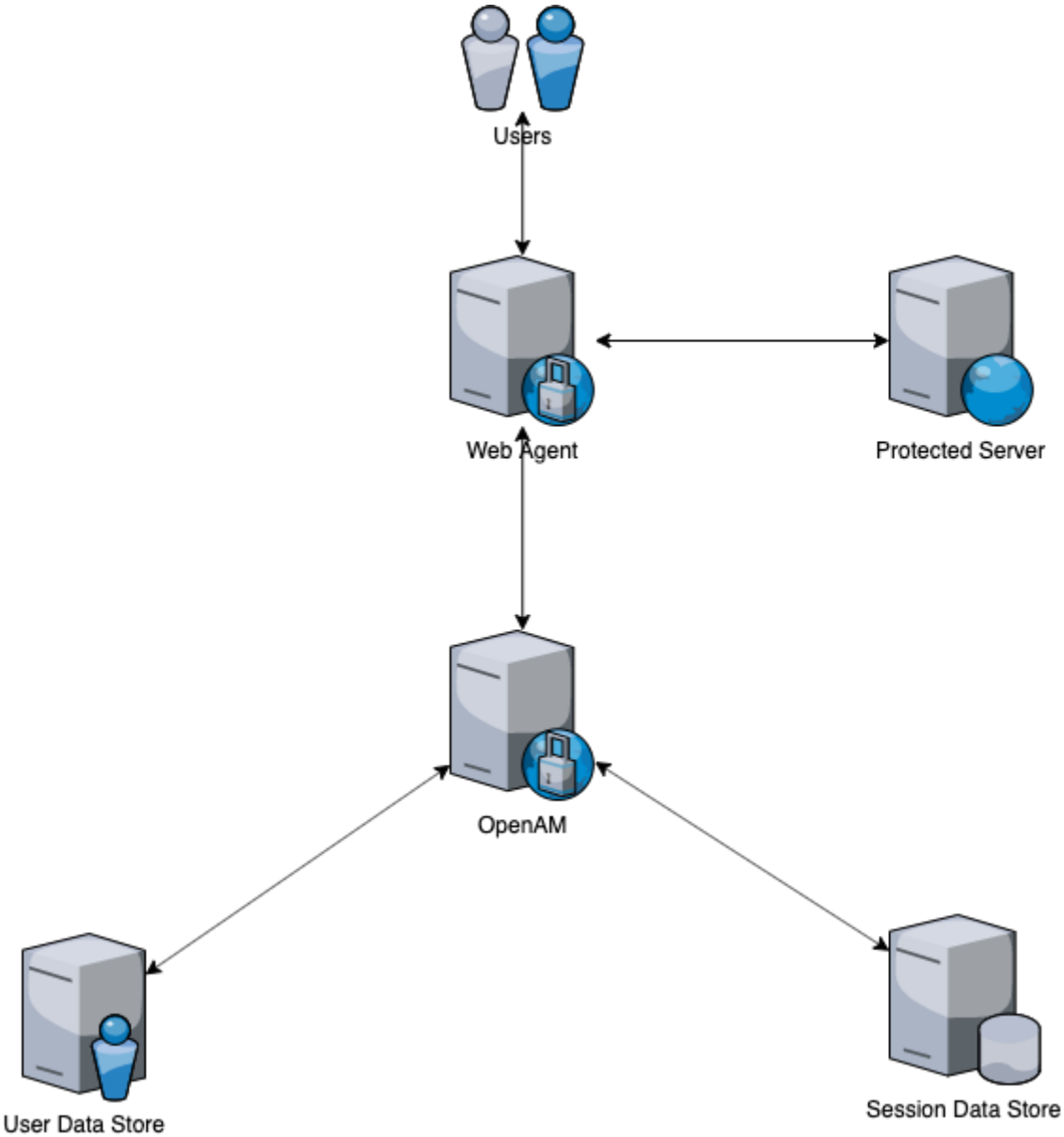
OpenAM uses OpenDJ as a user data store, but also could use Microsoft Active Directory, or any LDAP compatible server. OpenAM also supports any relational database via JDBC and Cassandra as a user data store.

Session Data Store

Persistent storage for session information. If OpenAM restarts or a request switch to another instance of OpenAM, then session information could be restored from the session data store.

Scheme

Simple OpenAM scheme shown on the picture below.



How Resource Protection Works

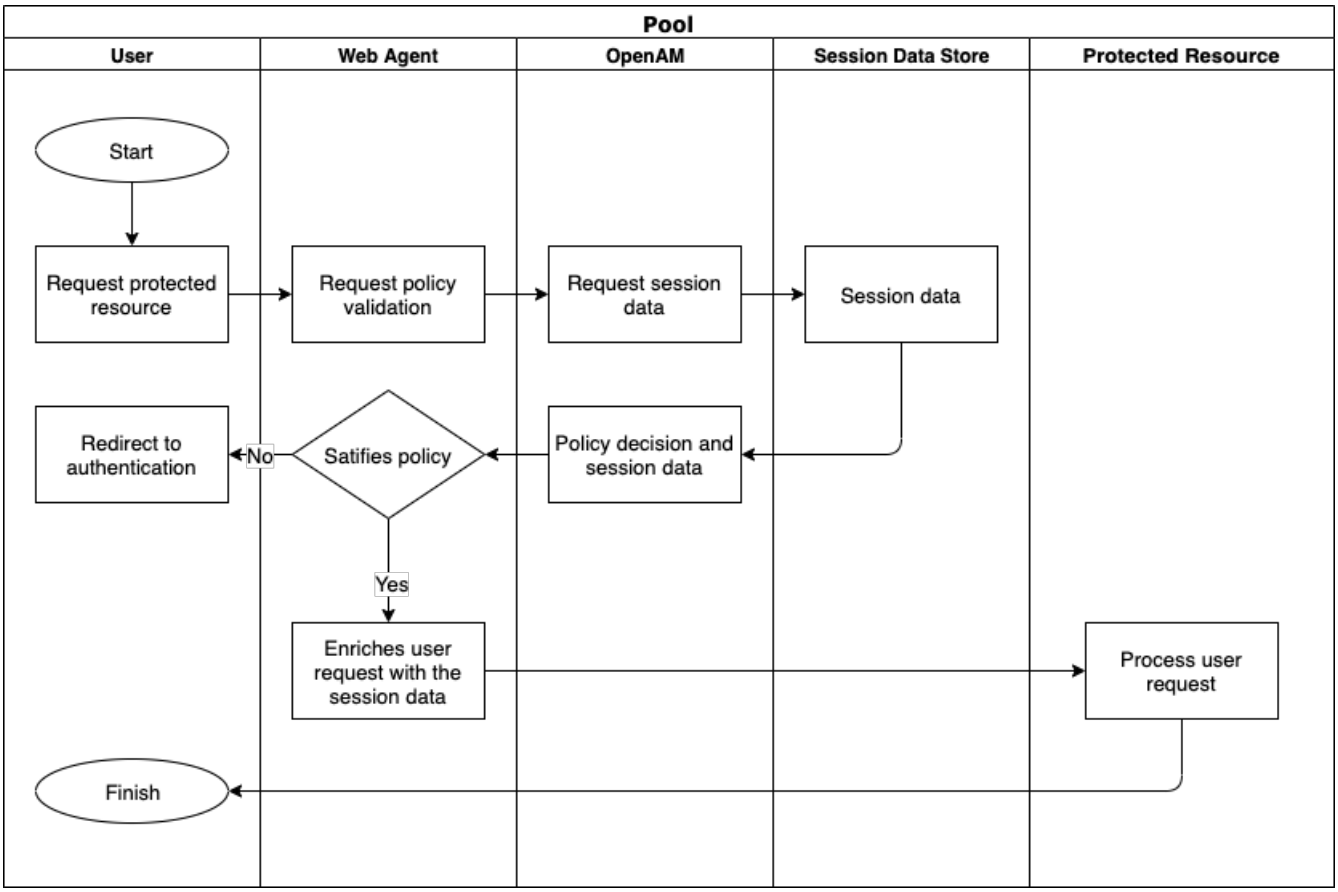
User tries to access a protected resource, Web Agent asks OpenAM whether user request satisfies the security policy or not.

If the request satisfies policy OpenAM returns session to Web Agent. Web Agent enriches the user

request with session data and forwards request to the protected resource and proxies protected server response back to the user.

If the user is not authenticated or the user session violates a policy, Web Agent redirects the user to authentication to the OpenAM server for authentication or additional authorization.

The process is shown on the image below:



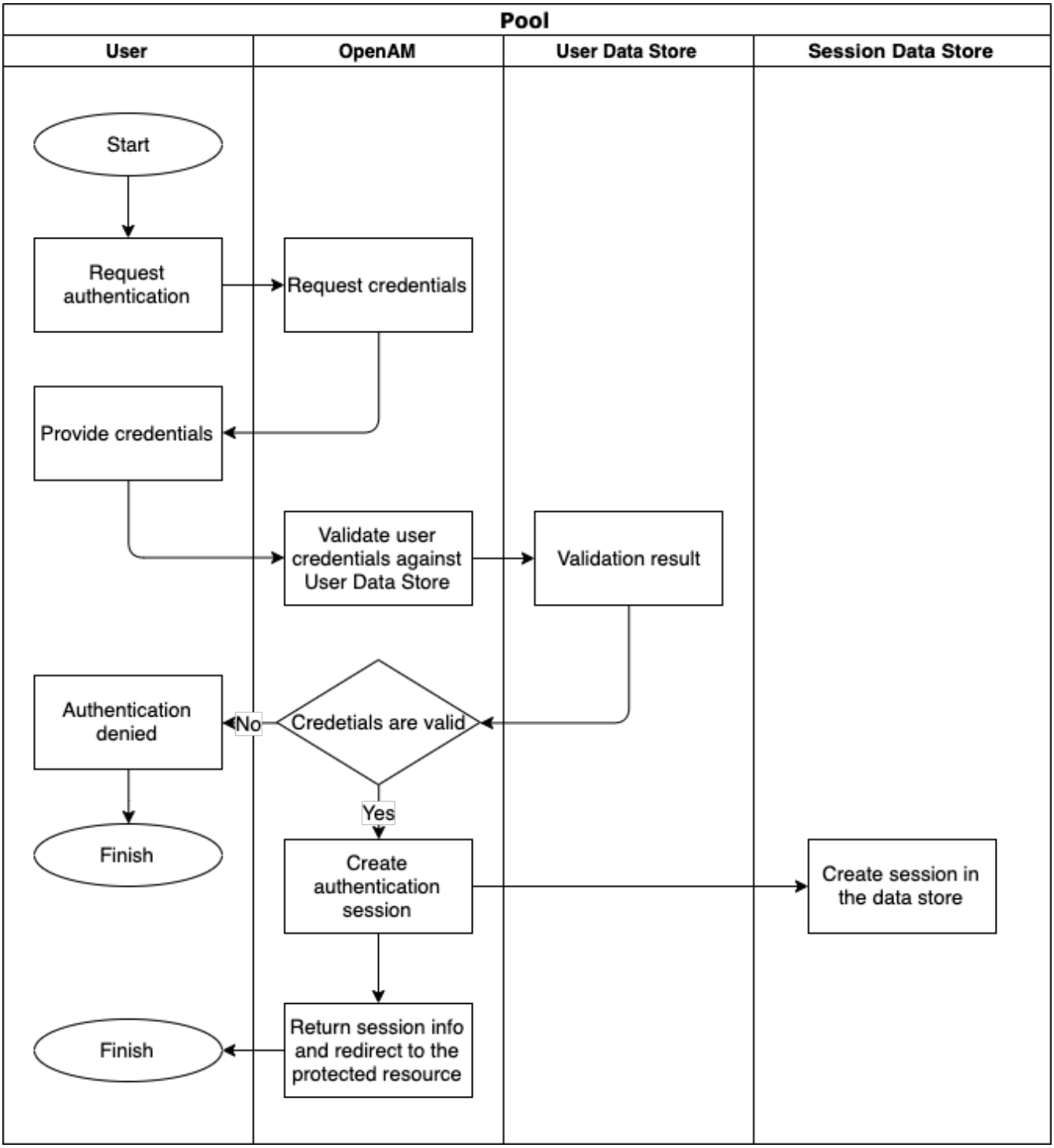
How User Authentication Works

While authentication OpenAM asks the user to provide his credentials. OpenAM checks if the user exists User Data Store and validates credentials.

If provided credentials are valid, OpenAM creates a new authenticated session in the Session Data Store and enriches the session with user data and authentication process details.

If credentials are invalid, OpenAM denies authentication.

After successful authentication, the user is redirected to the protected resource. This process is shown in the diagram below.



Conclusion

Of course, this document does not fully cover all OpenAM functionality as Federation Authentication, adaptive authentication, how to organize high availability and many more other functionalities. Its purpose just is just to make a more familiar and basic understanding of how OpenAM works.

How To Run OpenAM in Kubernetes

Preface

This article describes how to deploy OpenAM with embedded OpenDJ on Kubernetes. We will use OpenAM Docker image from Docker Hub Registry https://hub.docker.com/r/openidentityplatform/openam/

In this tutorial, we will use Minikube as a local Kubernetes cluster. `kubectl` should also be installed

Start Minikube with at least 4 cores, and 8 GB of memory to avoid insufficient resources errors.

```
minikube start --cpus=4 --memory=8g
```

Create OpenAM Service

The following service will be used for DNS lookups between OpenAM Pods and clients.

Create `openam-service.yaml` file with the following contents:

`openam-service.yaml`:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: openam
  name: openam
spec:
  clusterIP: None
  ports:
  - port: 8080
  selector:
    app: openam
```

Then create service in Kubernetes using `kubectl`:

```
kubectl apply -f openam-service.yaml
```

Make sure the service has been created:

```
kubectl get svc openam

NAME            TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
```

```
openam          ClusterIP   None         <none>        8080/TCP   7s
```

Create OpenAM StatefulSet

Create `openam-statefulset.yaml` file with the following contents:

`openam-statefulset.yaml`:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: openam
  labels:
    app: openam
spec:
  serviceName: openam
  replicas: 1
  selector:
    matchLabels:
      app: openam
  template:
    metadata:
      labels:
        app: openam
    spec:
      containers:
      - name: openam
        image: openidentityplatform/openam
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
          protocol: TCP
        - containerPort: 4444
          protocol: TCP
        - containerPort: 50389
          protocol: TCP
        env:
          - name: CATALINA_OPTS
            value: "-Xmx2048m -server
-Dcom.iplanet.services.configpath=/home/openam/openam
-Dcom.sun.identity.configuration.directory=/home/openam/openam"
        livenessProbe:
          failureThreshold: 3
          httpGet:
            path: /openam/isAlive.jsp
            port: 8080
            scheme: HTTP
          initialDelaySeconds: 120
          periodSeconds: 10
```

```
          successThreshold: 1
          timeoutSeconds: 3
        readinessProbe:
          failureThreshold: 3
          httpGet:
            path: /openam/isAlive.jsp
            port: 8080
            scheme: HTTP
          initialDelaySeconds: 120
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 3
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - name: openam-data
          mountPath: /usr/openam/config
  volumeClaimTemplates:
  - metadata:
      name: openam-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 5Gi
```

Create OpenAM StatefulSet in Kubernetes

```
kubectl apply -f openam-statefulset.yaml
```

Check if StatefulSet is wokring

```
kubectl get statefulset openam

NAME     READY   AGE
openam   1/1     5m
```

Wait for pod is created

Get created Pods

```
kubectl get pods -l="app=openam"
NAME       READY   STATUS    RESTARTS   AGE
openam-0   1/1     Running   0          5m
```

OpenAM Setup and Configuration

Lets check OpenAM pod status

```
$ kubectl exec -it openam-0 -- curl -v  http://localhost:8080/openam/isAlive.jsp
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /openam/isAlive.jsp HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.52.1
> Accept: */*
>
< HTTP/1.1 302
< Location: http://localhost:8080/openam/config/options.htm
< Content-Length: 0
< Date: Mon, 16 Dec 2019 07:23:26 GMT
<
* Curl_http_done: called premature == 0
* Connection #0 to host localhost left intact
```

isAlive response redirects to the configuration page, that means OpenAM is not configured. Let's configure the instance.

In `/etc/hosts` file add the following entry:

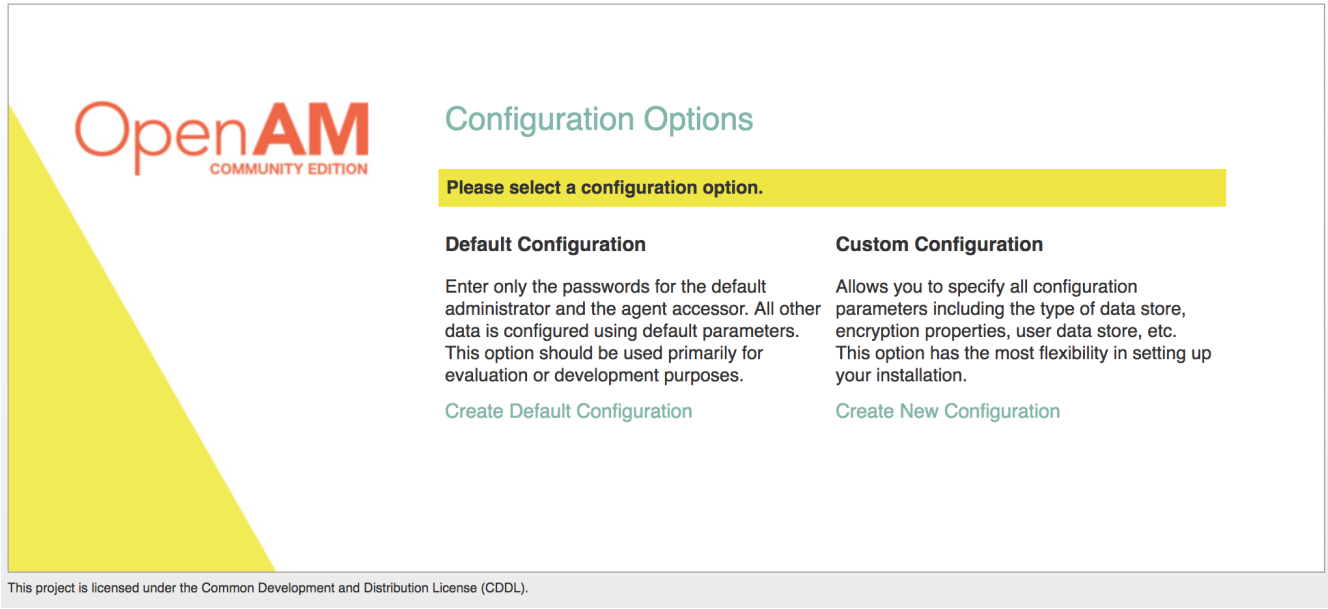
```
127.0.0.1   openam-0.openam.default.svc.cluster.local
```

Then forward OpenAM port from the service

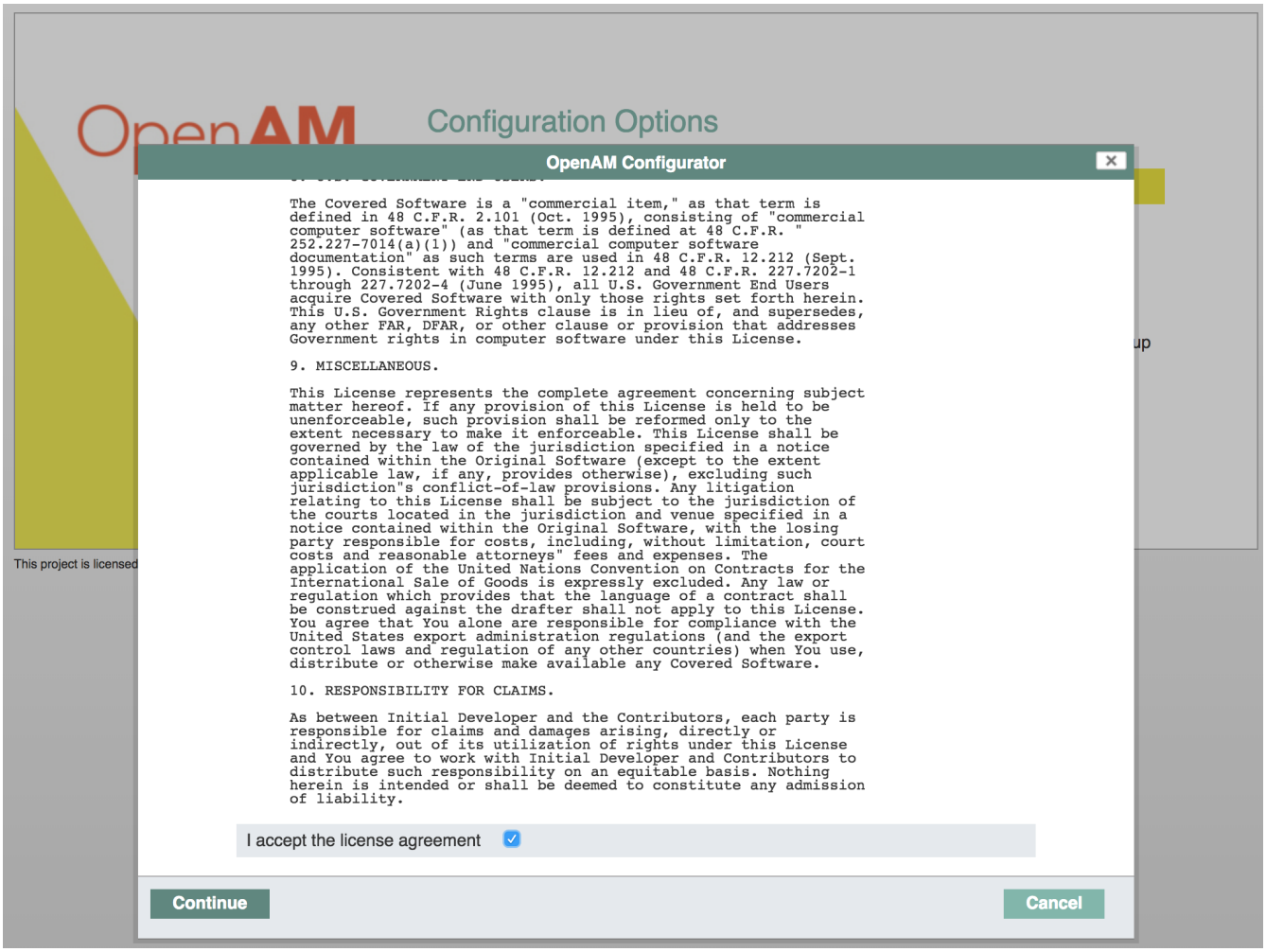
```
kubectl port-forward svc/openam 8080
```

Open http://openam-0.openam.default.svc.cluster.local:8080/openam/ url in your browser.

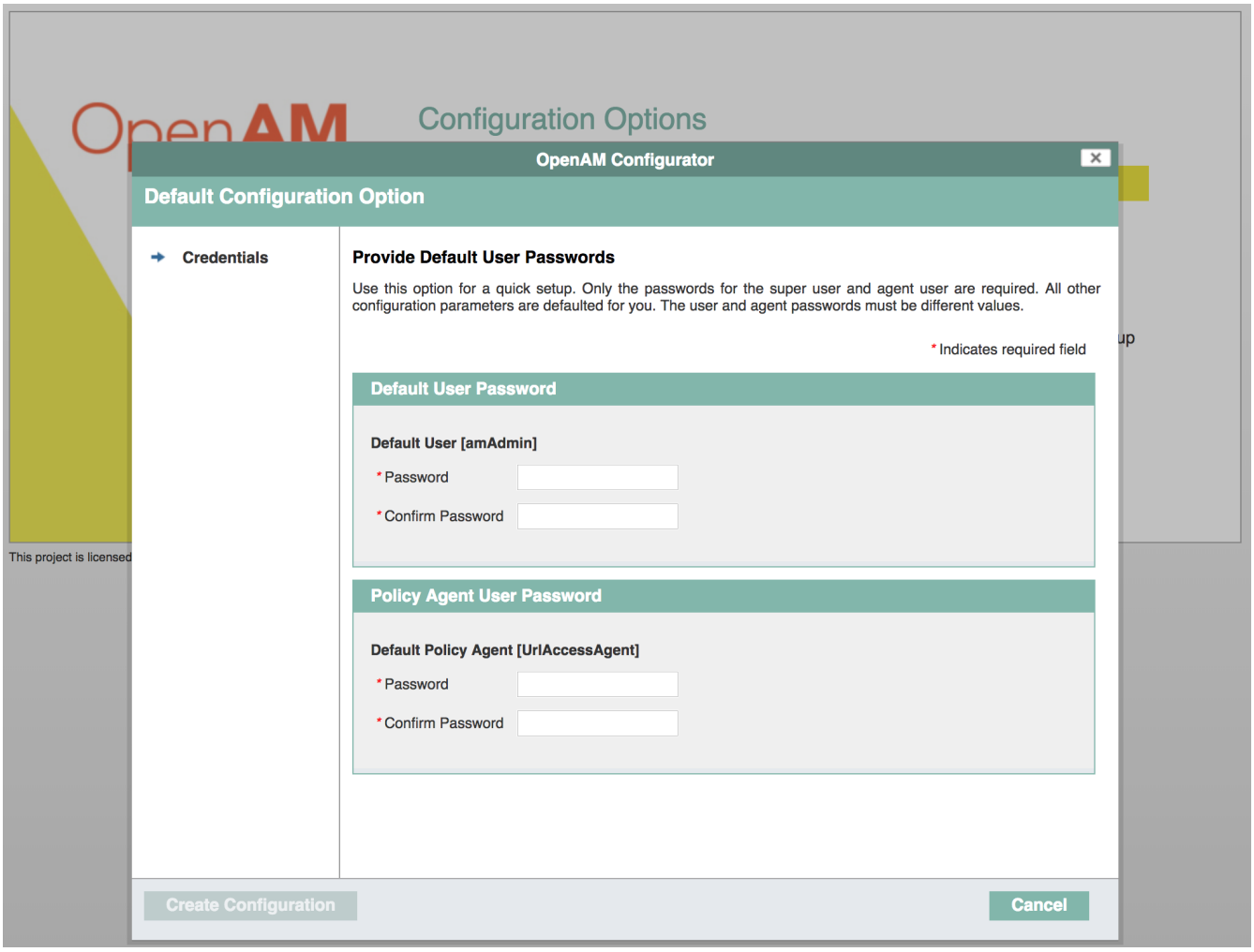
You will be redirected to the configuration screen



Click **Create Default Configuration**.



Accept License Agreement



Set password for default admin user and policy agent

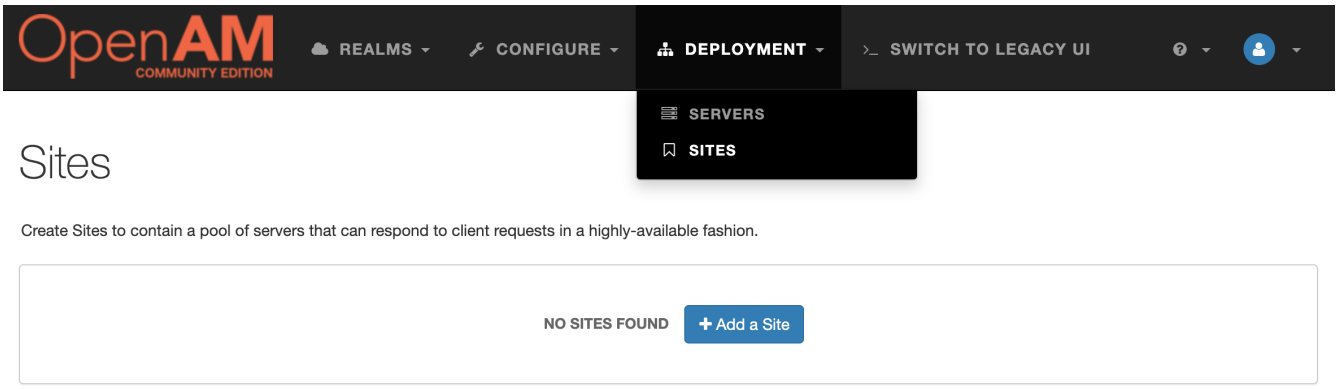
Press **Create Configuration**. After configuration successfully created, press **Proceed to Login** or open http://openam-0.openam.default.svc.cluster.local:8080/openam/console link in browser.

Add External Access via Ingress

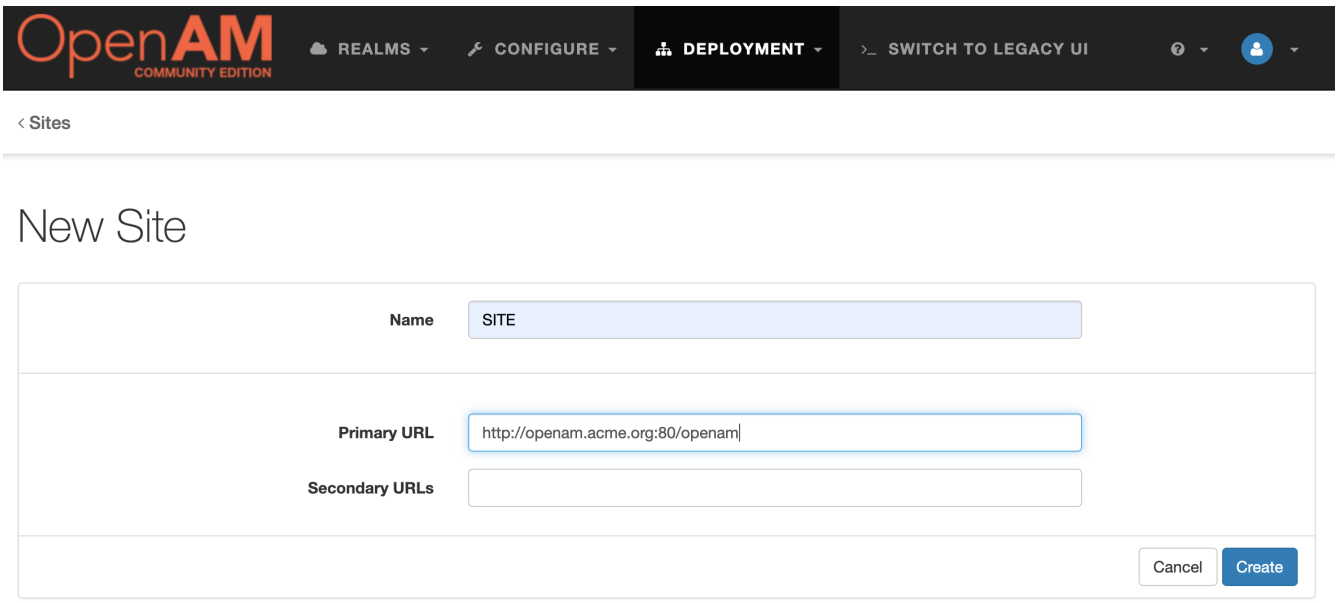
Preparation

For example you need OpenAM address be http://openam.acme.org/openam

Goto OpenAM console, **Deployment** -> **Sites** and create new site, for example, http://openam.acme.org:80/openam



Then goto OpenAM console, **Configure** -> **Global Services** -> **Platform** and add Cookie domain as shown on the picture below



Setup ingress

To make OpenAM accessible form external network we will use Ingress

If you use Minikube, enable ingress using the following command

```
minikube addons enable ingress
```

Create `openam-ingerss.yaml` file

`openam-ingress.yaml`:

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: openam-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /openam
spec:
  rules:
  - http:
      paths:
        - path: /openam
          backend:
            serviceName: openam
            servicePort: 8080
```

And create Ingress for OpenAM service in Kubernetes

```
kubectl apply -f openam-ingress.yaml
```

Test Access

Get your minikube IP

```
echo $(minikube ip)
192.168.64.15
```

And add the following entry to your `/etc/hosts` file

```
192.168.64.15   openam.acme.org
```

Then open http://openam.acme.org/openam url in your browser, and you should see authentication screen

How To Setup Active Directory Authentication In OpenAM

Preface

If your organization uses Microsoft Active Directory for user storage, it is a good practice to use Active Directory user accounts to authenticate in all your enterprise applications. OpenAM supports Microsoft Active Directory

But setting up Microsoft Active Directory as a user data store could be tricky. In this article, we'll help you to set up user data store in OpenAM

Setup Active Directory User Data Store

Enterprise users should be in a separate realm.

NOTE

Of course, you can use an existing realm or even use different data stores in a single realm in OpenAM. But in this manual, we will create a separate realm and a sigle data store for employees

---

So login in OpenAM console as amadmin and create realm `/staff`. Delete default user Data Store in `/staff` realm.

Then create Active Directory data store with type Active Directory.

There are the most important settings in a table below:

Setting	Value
Ldap Server	AD host and port, for example: ad.example.com:389
LDAP Bind DN	Bund DN or user name for AD, for example `EXAMPLE\Administrator`
LDAP Bind Password	Bind DN password
LDAP Organization DN	DN where users are located DC=ad,DC=example,DC=com
LDAP Connection Pool Maximum Size	128
Attribute Name Mapping	uid=sAMAccountName userPassword=unicodePwd
LDAPv3 Plug-in Supported Types and Operations	user=read group=read realm=read

Setting	Value
LDAP Users Search Attribute:	sAMAccountName
LDAP Users Search Filter	(objectclass=person)
DN Cache	Enabled

Test Data Store and Authentication

If you set all settings correctly, you should see user account form your active directory, in Subjects tab in the realm.

Then test authentication: Open OpenAM URL in your browser, for example

For XUI:

http://openam.example.org:8080/openam/XUI/?org=/staff#login/

For legacy UI:

http://openam.example.org:8080/openam/UI/Login?org=/staff

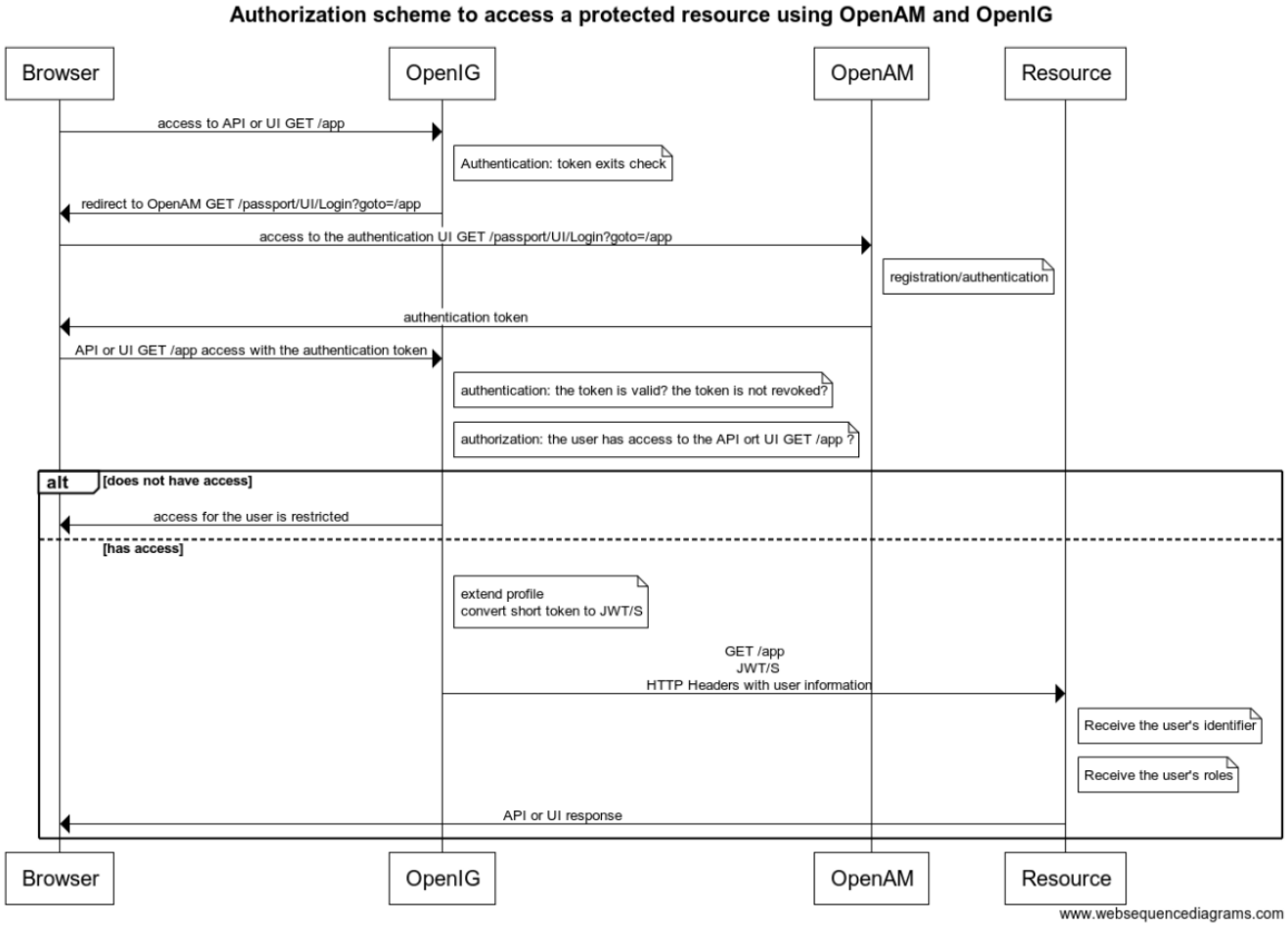
Enter your Active Directory credentials, and you should be successfully authenticated

How to Add Authorization and Protect Your Application With OpenAM and OpenIG Stack

Introduction

In the following article, we will set up centralized authentication with OpenAM and set up gateway access to a test application with OpenIG. OpenIG will use an authentication session issued by OpenAM for authorization. We will use an application developed using Spring Boot and Spring Security as the protected application. The source code of the application is located on GitHub. For the demonstration purposes, we will run services in Docker containers with the `docker compose` tool.

The authorization diagram is in the picture below:



Run the Test Application

Create a `docker-compose.yaml` file and add `spring-service` to the services section. Map the 8081 port to test for a functional check

`docker-compose.yaml`

```
services:
  spring-service:
```

```
    container_name: spring-service
    image: openidentityplatform/spring-security-openam-example
    restart: always
    ports:
      - "8081:8081"
    environment:
      JAVA_OPTS: -Dspring.profiles.active=jwt
    networks:
      openam_network:
        aliases:
          - spring-service

networks:
  openam_network:
    driver: bridge
```

Once the application is running, let's check access to the API for which we need to add authentication.

```
curl http://localhost:8081/api/protected-jwt | json_pp
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   105    0   105    0     0  12684      0 --:--:-- --:--:-- --:--:-- 13125
{
   "error" : "Unauthorized",
   "path" : "/protected-jwt",
   "status" : 401,
   "timestamp" : "2024-04-16T06:01:25.331+00:00"
}
```

For successful authentication, a valid JWT must be passed to the API in the Authorization HTTP header. This is what the OpenAM and OpenIG stack will be responsible for. Shut down the test service with the `docker compose down` command for now.

OpenAM Setup

Let's configure the OpenAM authentication service. It will be responsible for authentication and converting the authentication token to JWT (see below).

Add the OpenAM and OpenIG hostnames to the `hosts` file, for example `127.0.0.1 openam.example.org openig.example.org`. On Windows systems, the hosts file is located at `C:\Windows\System32\drivers\etc\hosts`, on Linux and Mac it is located at `/etc/hosts`. Add the OpenAM service to the `docker-compose.yaml` file:

```
...
  openam:
    image: openidentityplatform/openam:latest
    container_name: openam
```

```
    ports:
      - "8080:8080"
    networks:
      openam_network:
        aliases:
          - openam.example.org
...
```

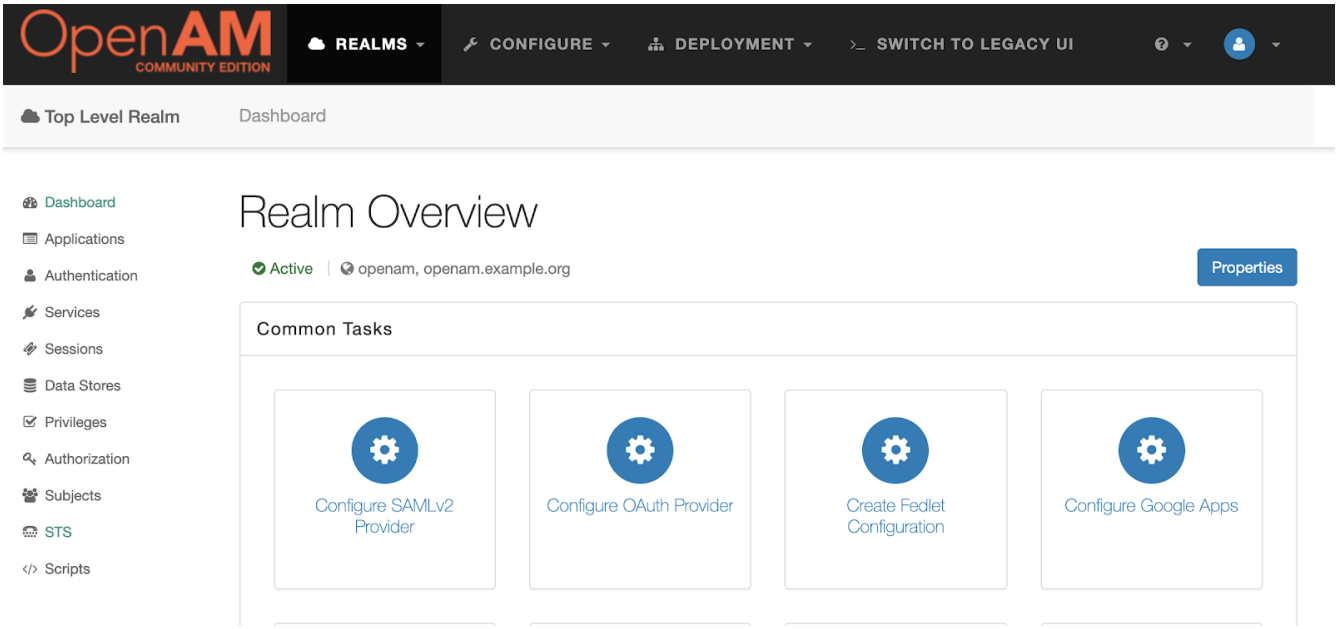
Start OpenAM with the `docker compose up openam` command. Once OpenAM is running, configure using the following command:

```
docker exec -w '/usr/openam/ssoconfiguratortools' openam bash -c \
'echo "ACCEPT_LICENSES=true
SERVER_URL=http://openam.example.org:8080
DEPLOYMENT_URI=/$OPENAM_PATH
BASE_DIR=$OPENAM_DATA_DIR
locale=en_US
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=passw0rd
AMLDAPUSERPASSWD=p@passw0rd
COOKIE_DOMAIN=example.org
ACCEPT_LICENSES=true
DATA_STORE=embedded
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=openam.example.org
DIRECTORY_PORT=50389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1689
ROOT_SUFFIX=dc=openam,dc=example,dc=org
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=passw0rd" > conf.file && java -jar openam-configurator-tool*.jar
--file conf.file'
```

And wait for the command execution to complete.

STS Setup

STS (Security Token Service) converts an OpenAM token to a JWT. After authentication, OpenAM returns an authentication token, which is a randomly generated sequence of characters. The STS service is responsible for converting the authentication token into a JWT, which contains information about the authenticated user. OpenIG will use STS for authorization. To configure STS, go to the administrator console URL http://openam.example.org:8080/openam/XUI/#login/ In the login field enter the `amadmin` value, in the password field enter the value from the `ADMIN_PWD` parameter of the setup command, in this case, `passw0rd`.



In the left menu click **STS** and create a new **STS Instance** with the following settings:

Setting	Value
Supported Token Transforms	OPENAM->OPENIDCONNECT;don't invalidate interim OpenAM session
Deployment Url Element	jwt
The id of the OpenID Connect Token Provider	https://openam.example.org/openam
Client secret	changeme
Confirm client secret	changeme
The audience for issued tokens	https://openam.example.org/openam

Then press the **Create** button

Setting up a test user

In the OpenAM admin console, navigate to the root realm and select `Subjects` from the left menu. Set the password for the `demo` user. To do this, select it in the list of users, and click the `Edit` link under Password. Enter and save the new password. Next, log out of the administrator console.

OpenIG Setup

OpenIG is responsible for authorization policies for back-end services. It converts the authentication token to a JWT in the OpenAM service and after successful authorization, passes the JWT to the back-end services in the HTTP Authorization header.

Create an `openig-config` folder and add two files to it: `admin.json`

```
{
    "prefix" : "openig",
    "mode": "PRODUCTION"
```

```
}
```

and `config.json`:

```
{
  "heap": [
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [

      ],
      "handler": {
        "type": "Router",
        "name": "_router",
        "capture": "all"
      }
    }
  }
}
```

Next, add a route that will proxy user requests to the test application, enriching the request with an Authorization header with a JWT token obtained from OpenAM. Create a folder `openig-config/routes/` and add the `10-protected.json` file to the folder.

`10-protected.json`

```
{
    "name": "${matches(request.uri.path, '^/api/protected-jwt') ||
matches(request.uri.path, '^/protected-jwt')}",
    "condition": "${matches(request.uri.path, '^/api/protected-jwt') ||
matches(request.uri.path, '^/protected-jwt')}",
    "monitor": true,
    "timer": true,
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "type": "ConditionalFilter",
                    "config": {
                        "condition": "${empty contexts.sts.issuedToken and not empty
request.cookies['iPlanetDirectoryPro'][0].value}",
                        "delegate": {
                            "type": "TokenTransformationFilter",
                            "config": {
                                "openamUri": "${system['openam']}",
                                "realm": "/",
```

```
                                "instance": "jwt",
                                "from": "OPENAM",
                                "to": "OPENIDCONNECT",
                                "idToken":
"${request.cookies['iPlanetDirectoryPro'][0].value}"
                            }
                        }
                    }
                },
                {
                    "type": "ConditionalFilter",
                    "config": {
                        "condition": "${not empty contexts.sts.issuedToken}",
                        "delegate": {
                            "type": "HeaderFilter",
                            "config": {
                                "messageType": "REQUEST",
                                "remove": [
                                    "Authorization",
                                    "JWT"
                                ],
                                "add": {
                                    "Authorization": [
                                        "Bearer ${contexts.sts.issuedToken}"
                                    ]
                                }
                            }
                        }
                    }
                },
                {
                    "type": "ConditionEnforcementFilter",
                    "config": {
                        "condition": "${not empty contexts.sts.issuedToken}",
                        "failureHandler": {
                            "type": "StaticResponseHandler",
                            "config": {
                                "status": 302,
                                "reason": "Found",
                                "headers": {
                                    "Content-Type": [
                                        "application/json"
                                    ],
                                    "Location": [

"${system['openam']}/UI/Login?org=/&goto=${urlEncode(contexts.router.originalUri)}"
                                    ]
                                },
                                "entity": "{ \"Redirect\":
\"${system['openam']}/UI/Login?org=/&goto=${urlEncode(contexts.router.originalUri)}\"}
"
```

```
                            }
                        }
                    }
                }
            ],
            "handler": "EndpointHandler"
        }
    },
    "heap": [
        {
            "name": "EndpointHandler",
            "type": "DispatchHandler",
            "config": {
                "bindings": [
                    {
                        "handler": "ClientHandler",
                        "capture": "all",
                        "baseURI": "${matchingGroups(system['spring-service'],
\"((http|https):\/\/(.[^\/]*))\")[1]}"
                    }
                ]
            }
        }
    ]
}
```

This route authorizes two endpoints: API - `/api/protected-jwt` and UI - `/protected-jwt`. Add the OpenIG service to the `docker-compose.yaml` file and remove the port mapping from the `spring-service`. Now this service is available only through OpenIG.

```
services:
  openam:
    image: openidentityplatform/openam:latest
    container_name: openam
    ports:
      - "8080:8080"
    networks:
      openam_network:
        aliases:
          - openam.example.org

  openig:
    image: openidentityplatform/openig:latest
    container_name: openig
    volumes:
      - ./openig-config:/usr/local/openig-config:ro
    environment:
      CATALINA_OPTS: -Dopenig.base=/usr/local/openig-config -Dspring
-service=http://spring-service:8081 -Dopenam=http://openam.example.org:8080/openam
    ports:
```

```
      - "8081:8080"
    networks:
      openam_network:
        aliases:
          - openig.example.org

  spring-service:
    container_name: spring-service
    image: openidentityplatform/spring-security-openam-example
    restart: always
    # ports:
    #   - "8081:8081"
    environment:
      JAVA_OPTS: -Dspring.profiles.active=jwt
    networks:
      openam_network:
        aliases:
          - spring-service

networks:
  openam_network:
    driver: bridge
```

Start OpenIG and the demo application services with the `docker compose up openig spring-service` command.

Test the Solution

Test the API Authorization

Get the OpenAM authentication token for the `demo` user:

```
curl -X POST -H "X-OpenAM-Username: demo" -H "X-OpenAM-Password: passw0rd" \
  -H "Content-Type: application/json" -H "Accept-API-Version: resource=2.1" \
  http://openam.example.org:8080/openam/json/realms/root/authenticate | json_pp
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   159  100   159    0     0   4197      0 --:--:-- --:--:-- --:--:--  4297
{
   "realm" : "/",
   "successUrl" : "/openam/console",
   "tokenId" :
"AQIC5wM2LY4Sfcze3DbBXVSXggTyZNpGfwOoFPLnHwmqLG0.*AAJTSQACMDEAAlNLABM2MTY1Mjg2MzI5Mzc4
ODM0MzQ5AAJTMQAA*"
}
```

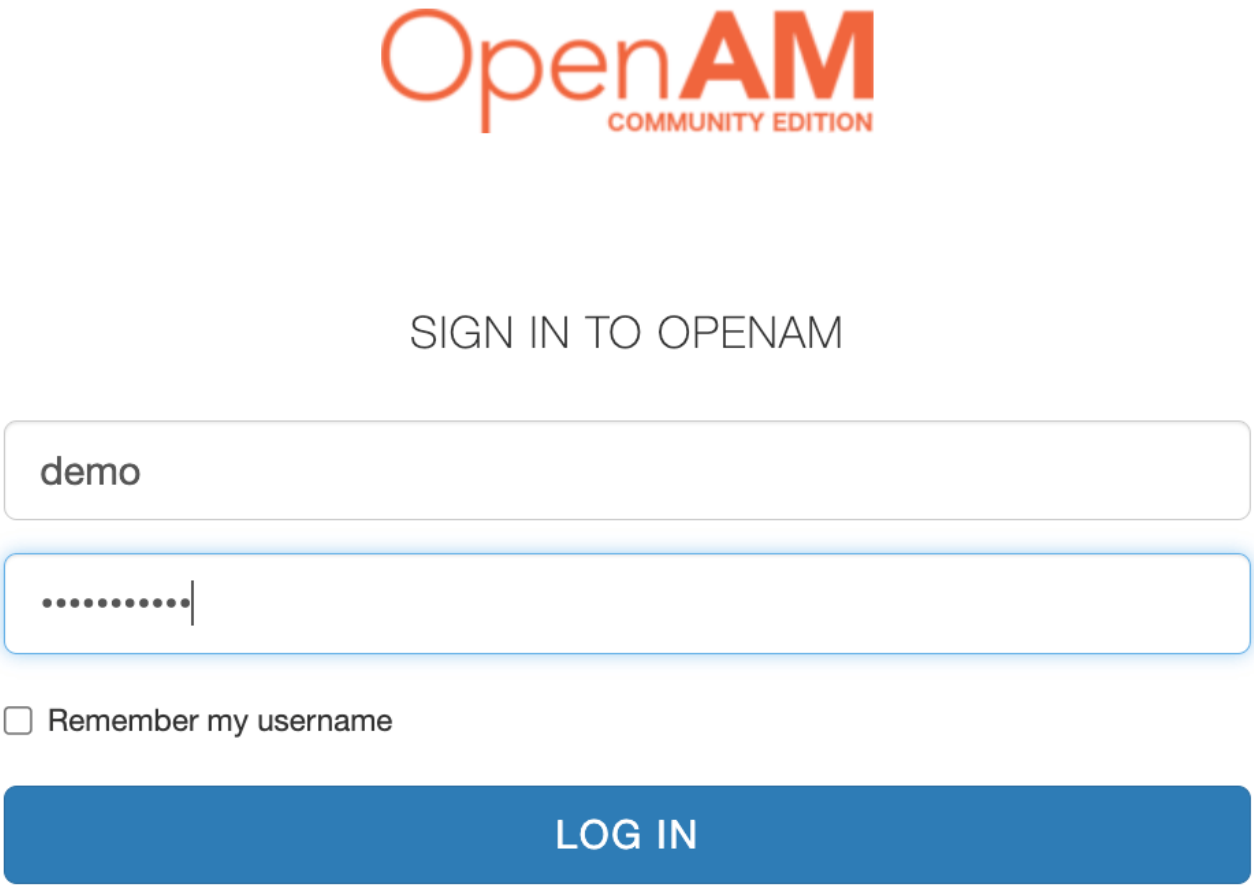
Call with the received token `/api/protected-jwt` endpoint:

```
curl  --cookie
"iPlanetDirectoryPro=AQIC5wM2LY4Sfcze3DbBXVSXggTyZNpGfwOoFPLnHwmqLG0.*AAJTSQACMDEAAlNL
ABM2MTY1Mjg2MzI5Mzc4ODM0MzQ5AAJTMQAA*" \
 "http://openig.example.org:8081/api/protected-jwt" | json_pp
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    30    0    30    0     0   1071      0 --:--:-- --:--:-- --:--:--  1111
{
   "method" : "JWT",
   "user" : "demo"
}
```

OpenIG, via OpenAM, converted the authentication token passed in the `iPlanetDirectoryPro` cookie into a JWT and passed this JWT to the `spring-service` test application. The test application retrieved the user information from the JWT and returned a successful response.

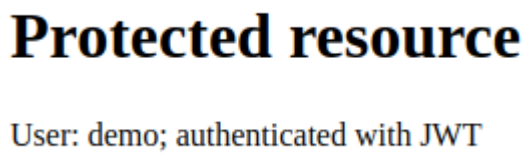
Test the UI Authorization

Log out of the OpenAM administrator console or open the browser in Incognito mode Open the http://openig.example.org:8081/protected-jwt URL in your browser. OpenIG will not find the authentication token cookie and will redirect the browser to OpenAM authentication. Enter the login `demo`, and password and click the `Log In` button.



After successful authentication, the gateway will redirect the request to the Spring Boot application.

It will receive a cookie with an authentication token from the http request, convert the token in OpenAM in STS service to JWT, and pass the received JWT to the demo application. The demo application will verify the JWT and return a successful response.



The source code for the demo application is available at https://github.com/OpenIdentityPlatform/spring-security-openam-example.

The source code for the docker compose configuration and OpenIG routes for the article can be found at https://github.com/OpenIdentityPlatform/openam-openig-springboot-example.

How to Create Custom OpenAM DataStore Integration

- Introduction
- Custom IdRepo Implementation
- Service Schema
- Add Schema to OpenAM Distribution
- Add Schema to an Exististing OpenAM Deployment
- IdRepo Plugin in Action
- Conclusion

Introduction

OpenAM already have implementations for variuos user data sources - LDAP, Active Directiry, relational databases via JDBC driver and Apache Cassandra. But there are databases that are not integratied into OpenAM. The following manual shows how to implement a custom data source to use for authentication. We will write together MongoDB integration to authenticate users in MongoDB datastore. The implementations will support only read users from the datastore and won't support write operation or groups, but it would be enough for get the point.

Custom IdRepo Implementation

To write your own integration with a custom data source you should extend com.sun.identity.idm.IdRepo abstract class.

To simplify the example our repository will be only able to show users, and authenticate against database.

Lets create `org.openidentityplatform.openam.mongodb.Repo` that extends `com.sun.identity.idm.IdRepo` class. Override `getSupportedOperations`, `getSupportedTypes` and `supportsAuthentication` functions.

```
public class Repo extends IdRepo {

    @Override
    public Set<IdOperation> getSupportedOperations(IdType type) {
        return Collections.singleton(IdOperation.READ);
    }

    @Override
    public Set<IdType> getSupportedTypes() {
        return Collections.singleton(IdType.USER);
    }

    @Override
```

```
    public boolean supportsAuthentication() {
        return true;
    }


}
```

Then add MongoDB connection settings and override the `initialize` function.

```
    private static final String MONGODB_SERVER =
            "sun-idrepo-ldapv3-config-ldap-server";

    private static final String MONGODB_DATABASE =
            "sun-idrepo-ldapv3-config-organization_name";

    private static final String USER_PASSWORD_ATTRIBUTE = "userpassword";

    String mongoDbUrl;

    MongoDatabase mongoDatabase;

    String databaseName;

    String usersMongoCollection = "users";

    @Override
    public void initialize(Map<String, Set<String>> configParams) throws
IdRepoException  {
        super.initialize(configParams);
        mongoDbUrl = "mongodb://".concat(String.join(",", configParams.get
(MONGODB_SERVER)));
        databaseName = CollectionHelper.getMapAttr(configParams, MONGODB_DATABASE);
        MongoClient mongoClient = MongoClients.create(mongoDbUrl);
        mongoDatabase = mongoClient.getDatabase(databaseName);
    }
```

Override the functions responsible to search and retrieve attributes from a database:

```
    @Override
    public RepoSearchResults search(SSOToken token, IdType type, CrestQuery
crestQuery, int maxTime, int maxResults,
                                    Set<String> returnAttrs, boolean returnAllAttrs,
int filterOp,
                                    Map<String, Set<String>> avPairs, boolean
recursive) throws IdRepoException, SSOException {

        final Map<String, Map<String,Set<String>>> result = new HashMap<>();

        FindIterable<Document> findIterable = mongoDatabase.getCollection
```

```
(usersMongoCollection).find().limit(maxResults);
        if(CollectionUtils.isNotEmpty(returnAttrs)) {
            Bson projection = Projections.fields(Projections.include(new ArrayList
<>(returnAttrs)));
            findIterable = findIterable.projection(projection);
        }

        if(MapUtils.isNotEmpty(avPairs)) {
            Bson filters = Filters.empty();
            for(Map.Entry<String, Set<String>> entry: avPairs.entrySet()) {
                List<Bson> entryFilters = entry.getValue().stream().map(v -> Filters
.eq(entry.getKey(), v)).collect(Collectors.toList());
                Bson entryFilter = Filters.or(entryFilters);
                filters = Filters.and(entryFilter);
            }
            findIterable = findIterable.filter(filters);
        }

        try(MongoCursor<Document> iterator = findIterable.iterator()) {
            while (iterator.hasNext()) {
                Document doc = iterator.next();
                String id = doc.getObjectId("_id").toHexString();
                Map<String, Set<String>> attributesValues = documentToUserAttributes
(doc);
                result.put(id, attributesValues);
            }
        }

        int errCode = (maxResults > 0 && result.size() > maxResults) ?
RepoSearchResults.SIZE_LIMIT_EXCEEDED : RepoSearchResults.SUCCESS;
        return new RepoSearchResults(result.keySet(), errCode, result,type);
    }

    @Override
    public Map<String, Set<String>> getAttributes(SSOToken token, IdType type, String
name) throws IdRepoException, SSOException {
        return getAttributes(token, type, name, null);
    }

    @Override
    public Map<String, Set<String>> getAttributes(SSOToken token, IdType type, String
name, Set<String> attrNames) throws IdRepoException, SSOException {
        try {
            final Bson filter = Filters.eq("_id", new ObjectId(name));
            Bson projection = Projections.exclude();
            if(attrNames != null) {
                projection = Projections.fields(Projections.include(new ArrayList
<>(attrNames)));
            }
            Document doc = mongoDatabase.getCollection(usersMongoCollection).find
(filter).projection(projection).first();
```

```
            if (doc == null) {
                return Collections.emptyMap();
            }
            return documentToUserAttributes(doc);

        } catch (Exception e) {
            throw new IdRepoException(e.getMessage());
        }
    }

     @Override
    public boolean isExists(SSOToken token, IdType type, String name) throws
IdRepoException, SSOException {
        try {
            final Bson filter = Filters.eq("_id", new ObjectId(name));
            Document doc = mongoDatabase.getCollection(usersMongoCollection).find
(filter).first();
            return doc != null;
        } catch (Exception e) {
            throw new IdRepoException(e.getMessage());
        }
    }

    @Override
    public boolean isActive(SSOToken token, IdType type, String name) throws
IdRepoException, SSOException {
        return true; //consider all identities are active
    }
```

Then implement authenticate function, which is responsible to authenticate users against a database. Password hashing is not supported, but should be implemented for production.

```
    @Override
    public boolean authenticate(Callback[] credentials) throws IdRepoException,
AuthLoginException {
        String userName = null;
        String password = null;
        for (Callback callback : credentials) {
            if (callback instanceof NameCallback) {
                userName = ((NameCallback) callback).getName();
            } else if (callback instanceof PasswordCallback) {
                password = new String(((PasswordCallback) callback).getPassword());
            }
        }
        if (userName == null || password == null) {
            throw new IdRepoException(IdRepoErrorCode.UNABLE_TO_AUTHENTICATE,Repo
.class.getName());
        }
        try {
            Map<String, Set<String>> res = getAttributes(null, IdType.USER, userName,
```

```
Collections.singleton(USER_PASSWORD_ATTRIBUTE));
            if (res!=null && res.containsKey(USER_PASSWORD_ATTRIBUTE) && res.get
(USER_PASSWORD_ATTRIBUTE).size() > 0) {
                final String storedPassword=res.get(USER_PASSWORD_ATTRIBUTE).
iterator().next();
                return storedPassword.equals(password);
            }
        } catch (SSOException e) {
            throw new AuthLoginException(e);
        }
        return false;
    }
```

Remaining functions should return default result or throw `IdRepoException`, for example

```
    private final static IdRepoException OPERATION_NOT_SUPPORTED_EXCEPTION = new
IdRepoException("operation not supported");

    @Override
    public Map<String, byte[][]> getBinaryServiceAttributes(SSOToken token, IdType
type, String name, String serviceName, Set<String> attrNames) throws IdRepoException,
SSOException {
        return Collections.emptyMap();
    }

    @Override
    public void modifyService(SSOToken token, IdType type, String name, String
serviceName, SchemaType sType, Map<String, Set<String>> attrMap) throws
IdRepoException, SSOException {
        throw OPERATION_NOT_SUPPORTED_EXCEPTION;
    }
```

Build the class and include it into classpath of OpenAM war file as desctibed in How to Customise OpenAM article. Dependencies for the maven project

```
 <dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-sync</artifactId>
    <version>4.9.1</version>
</dependency>
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.11</version>
</dependency>
```

Service Schema

OpenAM MongoDM service sub-schema will look like this.

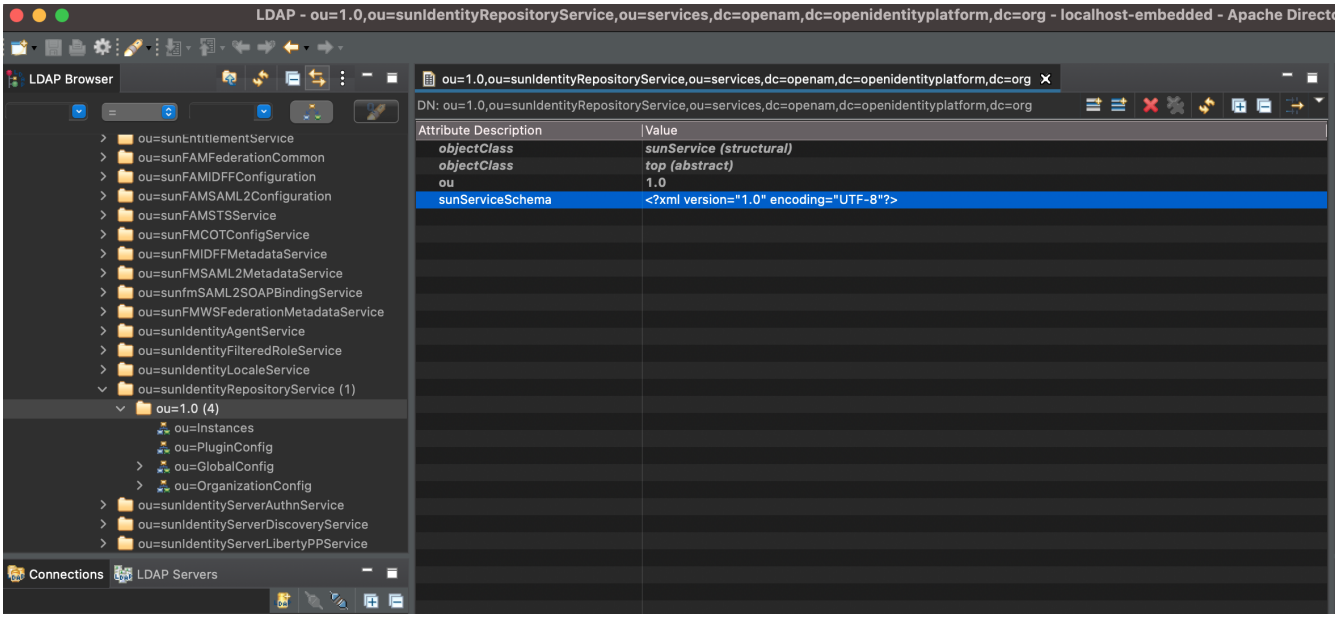
```
<SubSchema i18nKey="MongoDB" inheritance="multiple"
    maintainPriority="no" name="MongoDBRepo"
    supportsApplicableOrganization="no" validate="yes">
    <AttributeSchema cosQualifier="default"
        isSearchable="no" name="RequiredValueValidator" syntax="string"
        type="validator">
        <DefaultValues>
            <Value>com.sun.identity.sm.RequiredValueValidator</Value>
        </DefaultValues>
    </AttributeSchema>
    <AttributeSchema any="required" cosQualifier="default"
        i18nKey="a4010" isSearchable="no" name="sunIdRepoClass" syntax="string"
        type="single" validator="RequiredValueValidator">
        <DefaultValues>
            <Value>org.openidentityplatform.openam.mongodb.Repo</Value>
        </DefaultValues>
    </AttributeSchema>
    <AttributeSchema any="required" cosQualifier="default"
                        i18nKey="Servers" isSearchable="no"
                        name="sun-idrepo-ldapv3-config-ldap-server" syntax="string"
type="list"
                        validator="RequiredValueValidator">
        <DefaultValues>
            <Value>localhost:27017</Value>
        </DefaultValues>
    </AttributeSchema>
    <AttributeSchema cosQualifier="default" i18nKey="Database"
        isSearchable="no" name="sun-idrepo-ldapv3-config-organization_name"
syntax="string"
        type="single">
        <DefaultValues>
            <Value>authentication</Value>
        </DefaultValues>
    </AttributeSchema>
</SubSchema>
```

Add Schema to OpenAM Distribution

To let OpenAM know about the new Data Source, the subschema should be added to OpenAM `sunIdentityRepositoryService` service configuration. If you plan to add the repository plugin to OpenAM distribution and/or make a contribution to OpenAM. Add MongoDB subschema to subschema list to openam-server-only/src/main/resources/services/idRepoService.xml file.

Add Schema to an Exististing OpenAM Deployment

If you have an existing OpenAM deployment, connect to OpenDJ instance for OpenAM configuration in any LDAP client Application, for example, Apache Directory Studio, and add sub schema to `sunIdentityRepositoryService` xml configuration as shown in the picture below. Then restart OpenAM instance.



IdRepo Plugin in Action

After we have added a new datastore to the schema, create a new MongoDB Data Store in OpenAM and see it in action.

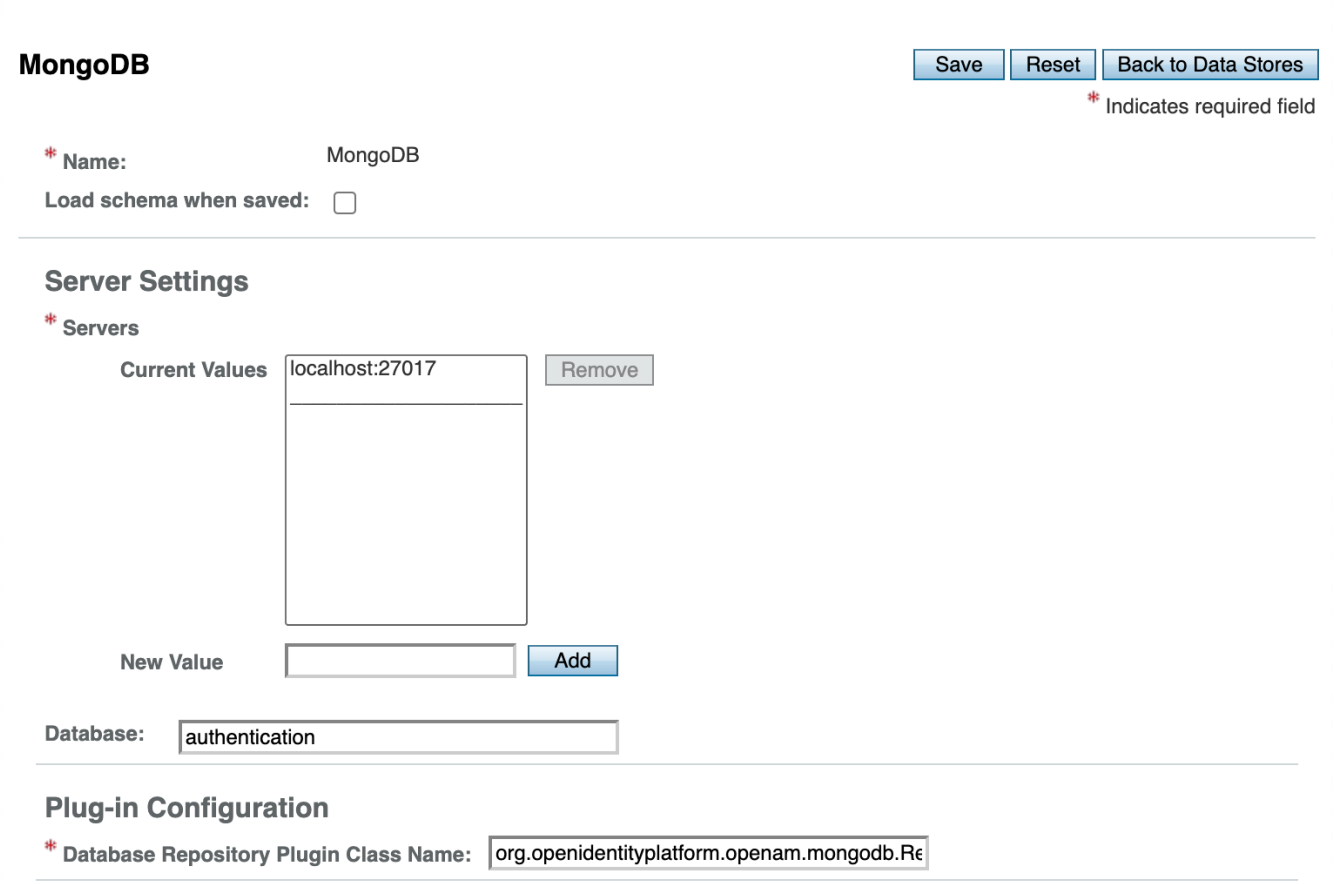
Run MongoDB in docker container in a local environment:

```
docker run -p 27017:27017 --name test-mongo mongo:latest
```

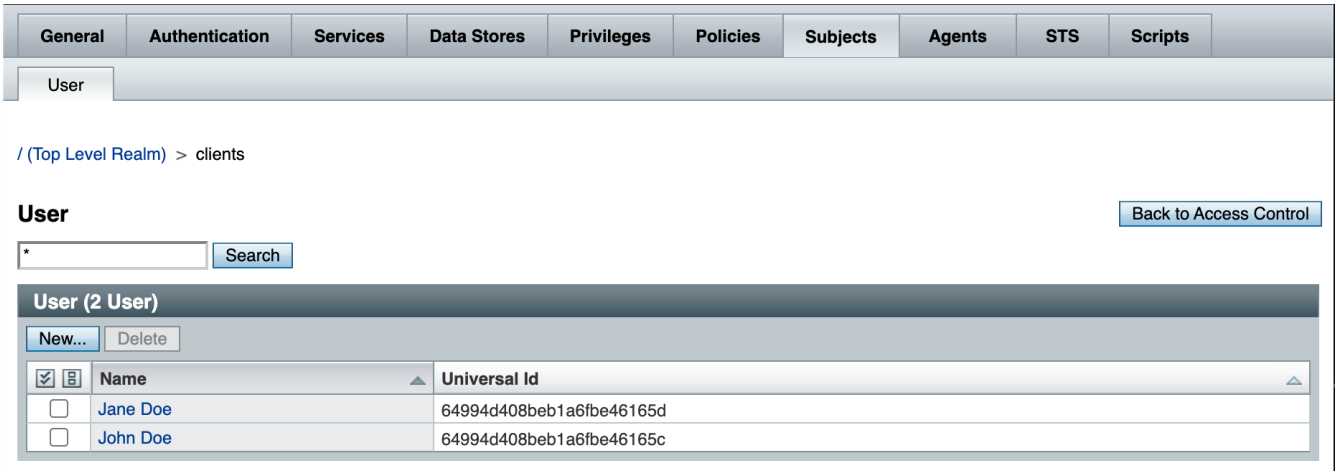
Add a couple of users to the MongoDB with the following script:

```
db = connect( 'mongodb://localhost:27017/authentication');
db.users.insertMany([
    {
        cn: 'John Doe',
        sunidentitymsisdnnumber: "123-456-78-90",
        userpassword: 'passw0rd',
    },
    {
        cn: 'Jane Doe',
        sunidentitymsisdnnumber: "123-456-78-91",
        userpassword: 'passw0rd',
    },
])
```

Open the OpenAM console, create a new realm `/clients`, remove the existing DataStore, and add a new MongoDB DataStore to the realm.



Open the Subjects tab and see if there are user entries:



Test authentication via API, using Universal Id value as login and password from the script above.

```
% curl --location --request POST
'http://localhost:8080/openam/json/authenticate?realm=/clients' \
--header 'X-OpenAM-Username: 64994f6b8beb1a6fbe461660' \
--header 'X-OpenAM-Password: passw0rd' \
--header 'Content-Type: application/json'

{"tokenId":"AQIC5wM2LY4Sfcx71mFbrHLrhHsv5hFyLEM_7S-
```

nbKBNrVg.*AAJTSQACMDEAAlNLABI0OTk2NDUxMTgwNTY4NTQ1NjUAAlMxAAA.*","successUrl":"/openam/console","realm":"/clients"}

Conclusion

In the following article, we have developed a very simple User Data Store without any modification, just to understand how it works. For a production environment password hashing connection pooling etc, should be implemented.

How to Customise OpenAM

NOTE	This example was written for OpenAM 14.7.1. Use dependencies and plugins compatible with your OpenAM version.

- Introduction
- Create Maven Project
- Run and Debug War File Locally
- Customisation Example
- Conclusion

Introduction

OpenAM is a reliable and mature product itself out of the box, have rich functionality and a good extension potential. But there is always a question - how to develop and debug the extensions? In the following article we will create a maven project and do simple OpenAM customisation.

Create Maven Project

Create a maven project with the following `pom.xml` file:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.openidentityplatform.openam</groupId>
        <artifactId>openam</artifactId>
        <version>14.7.1</version>
    </parent>
    <repositories>
        <repository>
            <id>ossrh-releases</id>
            <name>Sonatype OSS</name>
            <url>https://oss.sonatype.org/content/repositories/releases/</url>
        </repository>
    </repositories>
    <artifactId>openam-extended</artifactId>
    <packaging>war</packaging>
        <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <configuration>
```

```
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

The parent project will be `org.openidentityplatform.openam:openam` so in the following project we can use proper OpenAM dependencies in our project.

Project packaging should be war, as we build a web application that extends OpenAM server application.

To override OpenAM server add `org.openidentityplatform.openam:openam-server` dependency to the build section and overlay to `maven-war-plugin`

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.openidentityplatform.openam</groupId>
        <artifactId>openam</artifactId>
        <version>14.7.1</version>
    </parent>
    <repositories>
        <repository>
            <id>ossrh-releases</id>
            <name>Sonatype OSS</name>
            <url>https://oss.sonatype.org/content/repositories/releases/</url>
        </repository>
    </repositories>
    <artifactId>openam-extended</artifactId>
    <packaging>war</packaging>
    <dependencies>
        <dependency>
            <groupId>org.openidentityplatform.openam</groupId>
            <artifactId>openam-server</artifactId>
            <type>war</type>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <configuration>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
```

```
                    <overlays>
                        <overlay>
                            <groupId>org.openidentityplatform.openam</groupId>
                            <artifactId>openam-server</artifactId>
                        </overlay>
                    </overlays>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Run `mvn package` command and after a successful build, you can see `openam-extended-14.7.1.war` file. This file you can use this instead of the default OpenAM server war file.

Run and Debug War File Locally

Add to the maven pom to the plugins section `cargo-maven2-plugin`:

```
...
<plugin>
    <groupId>org.codehaus.cargo</groupId>
    <artifactId>cargo-maven2-plugin</artifactId>
    <configuration>
        <container>
            <containerId>tomcat9x</containerId>
            <type>embedded</type>
            <systemProperties>
                                <!--configuration directory path-->
                <com.iplanet.services.configpath>
${basedir}/config</com.iplanet.services.configpath>
                <com.sun.identity.configuration.directory>
${basedir}/config</com.sun.identity.configuration.directory>
            </systemProperties>
        </container>
        <configuration>
            <properties>
                                <!--openam port-->
                <cargo.servlet.port>8080</cargo.servlet.port>
                <cargo.rmi.port>8206</cargo.rmi.port>
                <cargo.jvmargs> ${java.surefire.options} -Xms4g -Xmx4g</cargo.jvmargs>
            </properties>
        </configuration>
        <deployables>
            <deployable>
                <type>war</type>
                <properties>
                    <context>openam</context>
                </properties>
```

```
                <pingURL>http://localhost:8080/openam</pingURL>
            </deployable>
        </deployables>
    </configuration>
</plugin>
...
```

Add desired OpenAM FQDN to your hosts file like this:

```
127.0.0.1    openam.example.com
```

To repackage and run war file execute the following command: `mvn package cargo:run`

You should see in the console something like this

```
[INFO] [beddedLocalContainer] Tomcat 9.x Embedded started on port [8080]
[INFO] Press Ctrl-C to stop the container...
```

Open OpenAM console in the browser with the following link http://openam.example.com:8080/openam and OpenAM will prompt you to configure a new instance. How to configure a new OpenAM instance is described in Quick Start Guide. After OpenAM configuration you will be able to use OpenAM locally within your development environment.

OpenAM configuration and log files will be stored in your project in `/config` directory

Customisation Example

After configuring OpenAM, let's do something simple to demonstrate customisation. We will modify the default OpenAM login page. We will add Acme corp logo instead of standard OpenAM logo.

The header template is in `XUI/templates/common/LoginHeaderTemplate.html` file. Create `src/main/webapp/XUI/templates/common/LoginHeaderTemplate.html` file in the project root folder. Then add the following contents:

```
<h1>ACME corp.</h1>
```

Start OpenAM and navigate to the login page. We can see that OpenAM uses the custom header. You can deploy the custom war file on your production web server

Conclusion

This article has only shown a simple setup in order to demonstrate the overall approach. In future articles we will show a more complex setup, such as developing a custom authentication module.

Creating a Custom Authentication Module

Related source code: OpenAM Sample Custom Authentication Module.

The listings below preserve the wiki tutorial's variant of the sample: package `com.forgerock.openam.examples`, user `test`, and configuration attribute `sampleauth-service-specific-attribute`. The repository sample instead uses class `org.forgerock.openam.examples.SampleAuth`, user `demo`, and attribute `specificAttribute`. Follow the listings and registration command below together; if you build the repository sample, register `org.forgerock.openam.examples.SampleAuth` and use its configuration files and credentials.

OpenAM comes complete with a wide variety of built-in custom authentication modules, but sometimes customer requirements step outside the functionality provided by the built-in modules. In these instances, it is time to consider writing your own custom authentication module. The OpenAM developer can use the source of the built-in modules as their starting point or start from scratch. If the customer requirement is a very close match to existing functionality then using a built-in module as a starting point will be a good idea. The OpenAM developer should start from scratch if the customer requirements does not match any of the built-in modules. In this HowTo we look at how to create and configure custom authentication modules.

Requirements for Sample Authentication module

Our sample authentication module takes username and password as input from the user. This data along with a service specific attribute and the program logic is used to authenticate the user. During this process some errors can occur which will be shown to user in detail. All texts displayed during the authentication must be easily configurable and translatable.

In short - we have following requirements

- User will be asked for name and password
- Module has one specific attribute which needs to be configured via OpenAM web-interface
- The authentication UI is configurable and translatable
- Some errors have to be displayed to user in detail

Prepare your development environment

We will stick with using the basic complier in this HowTo to keep things simple. In order to compile your custom authentication module either on the command line or in your favourite IDE you will need the following OpenAM jar files.

- openam-core*.jar
- openam-shared*.jar

Copy these JAR files from the deployed OpenAM application's `WEB-INF/lib` directory into the `lib` directory used by the compilation command below.

Design your custom authentication module

First important decision is to pick a name for our authentication module. We decide to call our module SampleAuth. An OpenAM custom authentication module is comprised of the following components:

- Authentication logic (SampleAuth.java) which extends the com.sun.identity.authentication.spi.AMLoginModule abstract class
- Principal (SampleAuthPrincipal.java) which implements the java.security.Principal interface
- Callbacks XML file (SampleAuth.xml) which describes the states of authentication logic and the user input needed during those states
- Service configuration XML file (amAuthSampleAuth.xml) which defines how service is configured on OpenAM side
- Authentication module properties file (amAuthSampleAuth.properties)

The Callbacks XML file

The custom authentication module is designed to ask the user a series of questions in order assert their identity. The order and detail of these questions is set out in the Callbacks XML file. The structure of the XML file is defined in the OPENAM_WAR/WEB-INF/Auth_Module_Properties.dtd file.

All of the built-in modules XML files are located in the OPENAM_WAR/config/auth/default directory. The OpenAM developer can use these as a starting point for their own callback XML file.

The custom authentication module

All custom authentication modules must extend from the com.sun.identity.authentication.spi.AMLoginModule abstract class. The abstract methods that must be implemented in the custom authentication module are listed below and the OpenAM Javadoc provides useful reference information.

```
public void init(Subject subject, Map sharedState, Map options)

public int process(Callback[] callbacks, int state)
throws LoginException

public Principal getPrincipal()
```

The init method is called once by OpenAM when the module is created. The process method is called into when the user submits callbacks back via the Authentication interface. The process method determines what happens next in the authentication process; success, failure or the next state (order) in the XML file. The getPrincipal method is called by OpenAM and must return an implementation of the java.security.Principal interface.

OpenAM does not reuse authentication module instances. This means that we can store

information specific to the authentication process in the instance.

OpenAM Service and Properties file

The OpenAM administration will require a way to configure any custom authentication modules. OpenAM mandates that all authentication modules; built-in or custom are configured in the same way by means of an OpenAM service. At the bare minimum the service should contain an authentication level attribute.

The module can find the value of these attributes in the options Map passed as a parameter to the init method. The properties file is a standard Java properties file that contains the description of the attributes that will be displayed in the console.

Implement your custom authentication module

Sample Callback XML file

Simple callbacks file might look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ModuleProperties PUBLIC "=//iPlanet//Authentication Module Properties XML
Interface 1.0 DTD//EN"
        "jar://com/sun/identity/authentication/Auth_Module_Properties.dtd">

<ModuleProperties moduleName="SampleAuth" version="1.0" >
    <Callbacks length="2" order="1" timeout="600" header="Please Login" >
        <NameCallback isRequired="true">
            <Prompt> User Name: </Prompt>
        </NameCallback>
        <PasswordCallback echoPassword="false" >
            <Prompt> Password: </Prompt>
        </PasswordCallback>
    </Callbacks>
</ModuleProperties>
```

This file defines authentication module SampleAuth with one possible state (order 1) in which 2 pieces of information are asked from the user. All strings that are used to draw the UI are typed into this file.

According to our requirements we need a little-bit more sophisticated callbacks file. We need dynamic text and we need error state. Following callbacks file achieves this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ModuleProperties PUBLIC "=//iPlanet//Authentication Module Properties XML
Interface 1.0 DTD//EN"
        "jar://com/sun/identity/authentication/Auth_Module_Properties.dtd">

<ModuleProperties moduleName="SampleAuth" version="1.0" >
```

```
    <Callbacks length="0" order="1" timeout="600" header="#WILL NOT BE SHOWN#" />
    <Callbacks length="2" order="2" timeout="600" header="#WILL BE SUBSTITUTED#" >
        <NameCallback isRequired="true">
            <Prompt>#USERNAME#</Prompt>
        </NameCallback>
        <PasswordCallback echoPassword="false" >
            <Prompt>#PASSWORD#</Prompt>
        </PasswordCallback>
    </Callbacks>
    <Callbacks length="1" order="3" timeout="600" header="#WILL BE SUBSTITUTED#"
error="true" >
        <NameCallback>
            <Prompt>#THE DUMMY WILL NEVER BE SHOWN#</Prompt>
        </NameCallback>
    </Callbacks>
</ModuleProperties>
```

Few comments:

- We need order 1 with 0 callbacks to change all UI strings for following orders. The trial-and-error shows that this dummy state is necessary to achieve dynamic UI-strings.
- The placeholders '#FOO BAR#' have no real meaning, this is just to show that those texts get replaced.
- The order 3 has attribute error set to "true". If the authentication module state machine reaches this order then the authentication has failed. The NameCallback is actually not used and not displayed to user, but the callbacks with length at least 1 is necessary, otherwise the OpenAM won't let us to substitute the header. So it's yet another hack forced upon us.

Sample Module Code

```
package com.forgerock.openam.examples;

import com.sun.identity.shared.datastruct.CollectionHelper;
import com.sun.identity.authentication.spi.AMLoginModule;
import com.sun.identity.authentication.spi.AuthLoginException;
import com.sun.identity.authentication.spi.InvalidPasswordException;
import com.sun.identity.authentication.util.ISAuthConstants;
import com.sun.identity.shared.debug.Debug;
import java.security.Principal;
import java.util.Map;
import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.login.LoginException;

import java.util.ResourceBundle;
import com.sun.identity.shared.locale.AMResourceBundleCache;
```

```
public class SampleAuth extends AMLoginModule {

    // Name for the debug-log
    private final static String DEBUG_NAME = "SampleAuth";

    // Name of the resource-bundle
    private final static String amAuthSampleAuth = "amAuthSampleAuth";

    // usernames for authentication logic
    private final static String USERNAME = "test";
    private final static String ERROR_1_NAME = "test1";
    private final static String ERROR_2_NAME = "test2";

    // orders defined in the callbacks file
    private final static int STATE_BEGIN = 1;
    private final static int STATE_AUTH = 2;
    private final static int STATE_ERROR = 3;


    private final static Debug debug = Debug.getInstance(DEBUG_NAME);

    private Map options;
    private ResourceBundle bundle;

    public SampleAuth() {
        super();
    }

    @Override
    // In this method we store service attributes and localized properties
    // for later use
    public void init(Subject subject, Map sharedState, Map options) {
        if (debug.messageEnabled()) {
            debug.message("SampleAuth::init");
        }
        this.options = options;
        bundle = amCache.getResBundle(amAuthSampleAuth, getLoginLocale());
    }

    @Override
    public int process(Callback[] callbacks, int state) throws LoginException {

        if (debug.messageEnabled()) {
            debug.message("SampleAuth::process state: " + state);
        }

        switch (state) {

            case STATE_BEGIN:
                // No time wasted here - simply modify the UI and
```

```
                // proceed to next state
                substituteUIStrings();
                return STATE_AUTH;

            case STATE_AUTH:

                // Get data from callbacks. Refer to callbacks xml file!
                NameCallback nc = (NameCallback) callbacks[0];
                PasswordCallback pc = (PasswordCallback) callbacks[1];
                String username = nc.getName();
                String password = new String(pc.getPassword());

                // First errorstring is stored in "sampleauth-error-1" property
                if (username.equals(ERROR_1_NAME)) {
                    setErrorText("sampleauth-error-1");
                    return STATE_ERROR;
                }

                // Second errorstring is stored in "sampleauth-error-2" property
                if (username.equals(ERROR_2_NAME)) {
                    setErrorText("sampleauth-error-2");
                    return STATE_ERROR;
                }

                if (username.equals(USERNAME) && password.equals("password")) {
                    return ISAuthConstants.LOGIN_SUCCEED;
                }

                throw new InvalidPasswordException("password is wrong", USERNAME);

            case STATE_ERROR:
                return STATE_ERROR;
            default:
                throw new AuthLoginException("invalid state");

        }
    }

    @Override
    public Principal getPrincipal() {
        return new SampleAuthPrincipal(USERNAME);
    }


    private void setErrorText(String err) throws AuthLoginException
    {
        // Receive correct string from properties and substitute the
        // header in callbacks order 3.
        substituteHeader(STATE_ERROR, bundle.getString(err));
    }
```

```
    private void substituteUIStrings() throws AuthLoginException
    {
        // Get service specific attribute configured in OpenAM
        String ssa = CollectionHelper.getMapAttr(options, "sampleauth-service-
specific-attribute");

        // Get property from bundle
        String new_hdr = ssa + " " + bundle.getString("sampleauth-ui-login-header");
        substituteHeader(STATE_AUTH, new_hdr);

        Callback[] cbs_phone = getCallback(STATE_AUTH);

        replaceCallback(STATE_AUTH, 0,
                    new NameCallback(bundle.getString("sampleauth-ui-username-
prompt")));

        replaceCallback(STATE_AUTH, 1,
                    new PasswordCallback(bundle.getString("sampleauth-ui-password-
prompt"), false));
    }

}
```

You will notice the code above allows the user "test" to log in with a password of "password". In order for this to work you will need to create a user called test in the OpenAM administration pages. This is because the code goes on to retrieve the user profile for the user, and if one doesn't exist, it will be this subsequent step that will fail and prevent you from logging in.

In the OpenAM console, logged in as amadmin, select "Access Control", Top Level Realm, "Subjects" and click "New..."

Here is the code for the principal implementation

```
package com.forgerock.openam.examples;

import java.io.Serializable;
import java.security.Principal;

/**
 *
 * @author Steve Ferris steve.ferris@forgerock.com
 */
public class SampleAuthPrincipal implements Principal, Serializable {
    private String name;
    private final static String CLASSNAME = "SampleAuthPrincipal";
    private final static String COLON = " : ";

    public SampleAuthPrincipal(String name) {
        if (name == null) {
            throw new NullPointerException("illegal null input");
```

```
        }

        this.name = name;
    }

    /**
     * Return the LDAP username for this <code> SampleAuthPrincipal </code>.
     *
     * <p>
     *
     * @return the LDAP username for this <code> SampleAuthPrincipal </code>
     */
    public String getName() {
        return name;
    }

    /**
     * Return a string representation of this <code> SampleAuthPrincipal </code>.
     *
     * <p>
     *
     * @return a string representation of this <code>TestAuthModulePrincipal</code>.
     */
    @Override
    public String toString() {
        return new StringBuilder().append(CLASSNAME).append(COLON).append(name
).toString();
    }

    /**
     * Compares the specified Object with this <code> SampleAuthPrincipal </code>
     * for equality.  Returns true if the given object is also a
     * <code> SampleAuthPrincipal </code> and the two SampleAuthPrincipal
     * have the same username.
     *
     * <p>
     *
     * @param o Object to be compared for equality with this
     *      <code> SampleAuthPrincipal </code>.
     *
     * @return true if the specified Object is equal equal to this
     *      <code> SampleAuthPrincipal </code>.
     */
    @Override
    public boolean equals(Object o) {
        if (o == null) {
            return false;
        }

        if (this == o) {
            return true;
```

```
        }

        if (!(o instanceof SampleAuthPrincipal)) {
            return false;
        }
        SampleAuthPrincipal that = (SampleAuthPrincipal) o;

        if (this.getName().equals(that.getName())) {
            return true;
        }
        return false;
    }

    /**
     * Return a hash code for this <code> SampleAuthPrincipal </code>.
     *
     * <p>
     *
     * @return a hash code for this <code> SampleAuthPrincipal </code>.
     */
    @Override
    public int hashCode() {
        return name.hashCode();
    }
}
```

Service configuration XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ServicesConfiguration
        PUBLIC "=//iPlanet//Service Management Services (SMS) 1.0 DTD//EN"
        "jar://com/sun/identity/sm/sms.dtd">

<ServicesConfiguration>
    <Service name="iPlanetAMAuthSampleAuthService" version="1.0">
        <Schema

serviceHierarchy="/DSAMEConfig/authentication/iPlanetAMAuthSampleAuthService"
                i18nFileName="amAuthSampleAuth"
                revisionNumber="10"
                i18nKey="sampleauth-service-description"
                resourceName="iPlanetAutSampleAuthService">
            <Organization>
                <AttributeSchema name="iplanet-am-auth-sampleauth-auth-level"
                                 type="single"
                                 syntax="number_range" rangeStart="0"
rangeEnd="2147483647"
                                 i18nKey="a500"
                                 resourceName="iplanetamauthsampleautlevel">
```

```
                    <DefaultValues>
                        <Value>1</Value>
                    </DefaultValues>
                </AttributeSchema>

                <AttributeSchema name="sampleauth-service-specific-attribute"
                                 type="single"
                                 syntax="string"
                                 validator="no"
                                 i18nKey="a501"
                                 resourceName="sampleauthservicespecattr">
                    <DefaultValues>
                        <Value></Value>
                    </DefaultValues>
                </AttributeSchema>

                <SubSchema name="serverconfig" inheritance="multiple"
resourceName="USE-PARENT">
                    <AttributeSchema name="iplanet-am-auth-sampleauth-auth-level"
                                     type="single"
                                     syntax="number_range" rangeStart="0"
rangeEnd="2147483647"
                                     i18nKey="a500"
                                     resourceName="iplanetamauthsampleautlevel">
                        <DefaultValues>
                            <Value>1</Value>
                        </DefaultValues>
                    </AttributeSchema>

                    <AttributeSchema name="sampleauth-service-specific-attribute"
                                     type="single"
                                     syntax="string"
                                     validator="no"
                                     i18nKey="a501"
                                     resourceName="sampleauthservicespecattr">
                        <DefaultValues>
                            <Value></Value>
                        </DefaultValues>
                    </AttributeSchema>

                </SubSchema>
            </Organization>
        </Schema>
    </Service>
</ServicesConfiguration>
```

- The service must have a name: **'iPlanetAMAuth'** + **modulename** + **'Service'**
- The service must have a description
- i18nFileName refers to the properties file and i18nKey to the key in the properties file

- sampleauth-service-specific-attribute configurable via OpenAM is defined here

The 'iplanet-am-auth-sampleauth-auth-level' is the obligatory authentication level attribute. Here naming is very important. The name of the attribute must be: 'iplanet-am-auth-' + lowercase(modulename) + '-auth-level'. If the attribute name does not follow the pattern your authentication module will not function the way you want it to. (There is also a second naming convention; please refer to the source code.)

Properties file

```
sampleauth-service-description=Sample Authentication Module
a500=Authentication Level
a501=Service Specific Attribute

sampleauth-ui-login-header=Login
sampleauth-ui-username-prompt=User Name:
sampleauth-ui-password-prompt=Password:

sampleauth-error-1=Error 1 occured during the authentication
sampleauth-error-2=Error 2 occured during the authentication
```

It's a simple properties file. Be careful to match all the identifier.

Compile your custom authentication module

Next we need to compile the custom authentication module. Run these commands from the directory containing `SampleAuth.java`, `SampleAuthPrincipal.java`, and `lib/`. Keep `amAuthSampleAuth.properties`, `SampleAuth.xml`, and `amAuthSampleAuth.xml` in this directory for the installation steps below.

```
$ javac -d classes -classpath 'lib/*' SampleAuth.java SampleAuthPrincipal.java
```

This will create our class file that we need to install into OpenAM. Next we should package the classes files up into a jar file.

```
$ jar cf sampleauth.jar -C classes .
```

Install your custom authentication module

Copy the jar file

We copy the jar file into the WEB-INF/lib directory within the openam web application deployed into Tomcat.

```
$ cp sampleauth.jar $TOMCAT_HOME/webapps/openam/WEB-INF/lib/
```

Copy the service properties file

We copy the properties file into the WEB-INF/classes directory within the openam web application deployed into Tomcat.

```
$ cp amAuthSampleAuth.properties $TOMCAT_HOME/webapps/openam/WEB-INF/classes/
```

Copy the authentication module XML file

We copy the authentication module XML file into the config/auth/default directory within the openam web application deployed into Tomcat.

```
$ cp SampleAuth.xml $TOMCAT_HOME/webapps/openam/config/auth/default
```

Register the module and the service with OpenAM

Use the ssoadm command to register the service and the module with OpenAM. The service can only be registered with OpenAM using ssoadm.

```
$ ssoadm create-svc -u amadmin -f pwdfile -X /path/to/amAuthSampleAuth.xml
$ ssoadm register-auth-module -u amadmin -f pwdfile -a
com.forgerock.openam.examples.SampleAuth
```

Note the service definition file is only needed for creating the service. When calling ssoadm create-svc, be sure to pass the full path to your service definition file.

Also following subcommands might be useful to you during the testing: delete-auth-cfgs, delete-auth-instances, delete-svc, list-auth-cfgs, list-auth-instances, remove-svc-realm, show-auth-modules, show-realm-svcs, unregister-auth-module, update-svc

It is a good idea at this point to restart Tomcat (or whatever container you are using) to ensure the Authentication Module and service definition can be loaded.

Configure your custom Authentication module

Log into the Console and navigate to Access Control >> Realm >> Authentication and click New under Authentication modules. Name your module and select the custom authentication module type from the list. Click OK. You should now see your custom module listed, you can select it and configure it as appropriate.

Test your custom module

The final step is to test out your custom authentication module. To do this you just need to login successfully to the realm where the custom authentication module is configured. This URL should do the trick:

http://hostname.domainname/openam/UI/Login?module=auth_module_name

You should be presented with the login screen and asked for the credentials required by your custom authentication module.

How to Protect Model Context Protocol (MCP) Servers with OpenAM and OpenIG

Introduction

Large language model (LLM) agents can perform various tasks, from writing code or texts to booking airline tickets. Agents consist of a client that interacts with the user and a server that performs the required tasks. The interaction between the client, server, and LLM occurs via the Model Context Protocol MCP.

MCP servers often have access to sensitive information, such as an internal source code repository or a customer database. Of course, not all users should have access to this data, even through an agent. To protect against unauthorized access, the Model Context Protocol specification describes the possibility of authorization based on OAuth 2.1: https://modelcontextprotocol.io/specification/2025-06-18/basic/authorization.

In this article, we will deploy a simple MCP server developed based on Spring AI and close it with the OpenIG authorization gateway. The OpenAM authentication service will be responsible for authentication.

We will use VS Code with the Copilot extension as the MCP client.

Project description

The source code for the OpenAM, OpenIG, and MCP server configuration is available at: https://github.com/OpenIdentityPlatform/openam-openig-mcp-example

The project consists of three services described in the `docker-compose.yml` file.

```
services:
  openig:
    build:
      context: ./openig-docker
      dockerfile: Dockerfile
    container_name: openig
    volumes:
      - ./openig-config:/usr/local/openig-config:ro
    ports:
      - "8081:8080"
    environment:
      CATALINA_OPTS: -Dopenig.base=/usr/local/openig-config
-Dopenam=http://openam.example.org:8080/openam
    networks:
      openam_network:
        aliases:
          - openig.example.org
```

```
  openam:
    build:
      context: ./openam-docker
      dockerfile: Dockerfile
    container_name: openam
    restart: always
    hostname: openam.example.org
    ports:
      - "8080:8080"
    volumes:
      - openam-data:/usr/openam/config
    networks:
      openam_network:
        aliases:
          - openam.example.org

  time-mcp-server:
    build:
      context: ./timeserver
      dockerfile: Dockerfile
    container_name: time-mcp-server
    ports:
      - "8082:8080"
    networks:
      openam_network:
        aliases:
          - timeserver.example.org
networks:
  openam_network:
    driver: bridge

volumes:
  openam-data:
```

Preparing for launch

For example, the host name for OpenAM will be `openam.example.org`, and for OpenIG it will be `openig.example.org`. Open the `hosts` file and add the host names and IP addresses to it, for example

```
127.0.0.1 openam.example.org openig.example.org
```

On Windows systems, the hosts file is located in the `C:\Windows/System32/drivers/etc/hosts` directory, and on Linux or Mac OS in `/etc/hosts`.

MCP Server

The MCP server has a method for returning the current time in ISO 8601 format.

```
@Service
public class TimeService {

    @Tool(name = "current_time_service", description = "Returns current time in ISO
8601 format")
    public String getTime() {
        return  Instant.now().toString();
    }
}
```

For more details on creating an MCP server, please refer to the documentation or in the Spring AI blog.

Start the OpenAM, OpenIG, and MCP server Docker containers with the command:

```
docker compose up --build
```

Check the availability of the running MCP server with the command:

```
curl -X POST  --location  "http://localhost:8082/mcp" \
    -H "Content-Type: application/json" \
    -H "Accept: application/json, text/event-stream" \
    -d '{
  "jsonrpc": "2.0",
  "id": 0,
  "method": "initialize",
  "params": {
    "protocolVersion": "2025-06-18",
    "capabilities": {}
  }
}'

{
  "id": 0,
  "jsonrpc": "2.0",
  "result": {
    "capabilities": {
      "completions": {},
      "prompts": {
        "listChanged": false
      },
      "resources": {
        "listChanged": false,
        "subscribe": false
      },
      "tools": {
        "listChanged": false
      }
```

```
    },
    "protocolVersion": "2025-03-26",
    "serverInfo": {
      "name": "time-server-mcp",
      "version": "0.0.1"
    }
  }
}
```

Let's check the availability of tools in the MCP server:

```
curl -X POST  --location  "http://localhost:8082/mcp" \
    -H "Content-Type: application/json" \
    -H "Accept: application/json, text/event-stream" \
    -d '{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "tools/list",
  "params": {}
}'

{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "tools": [
      {
        "name": "current_time_service",
        "description": "Returns current time in ISO 8601 format",
        "inputSchema": {
          "type": "object",
          "properties": {},
          "required": [],
          "additionalProperties": false
        }
      }
    ]
  }
}
```

Configuring OpenAM

OpenAM will be responsible for user authentication, issuing OAuth 2 `access_token` tokens, and validating them.

If you have not yet configured OpenAM, perform a quick setup by running the command:

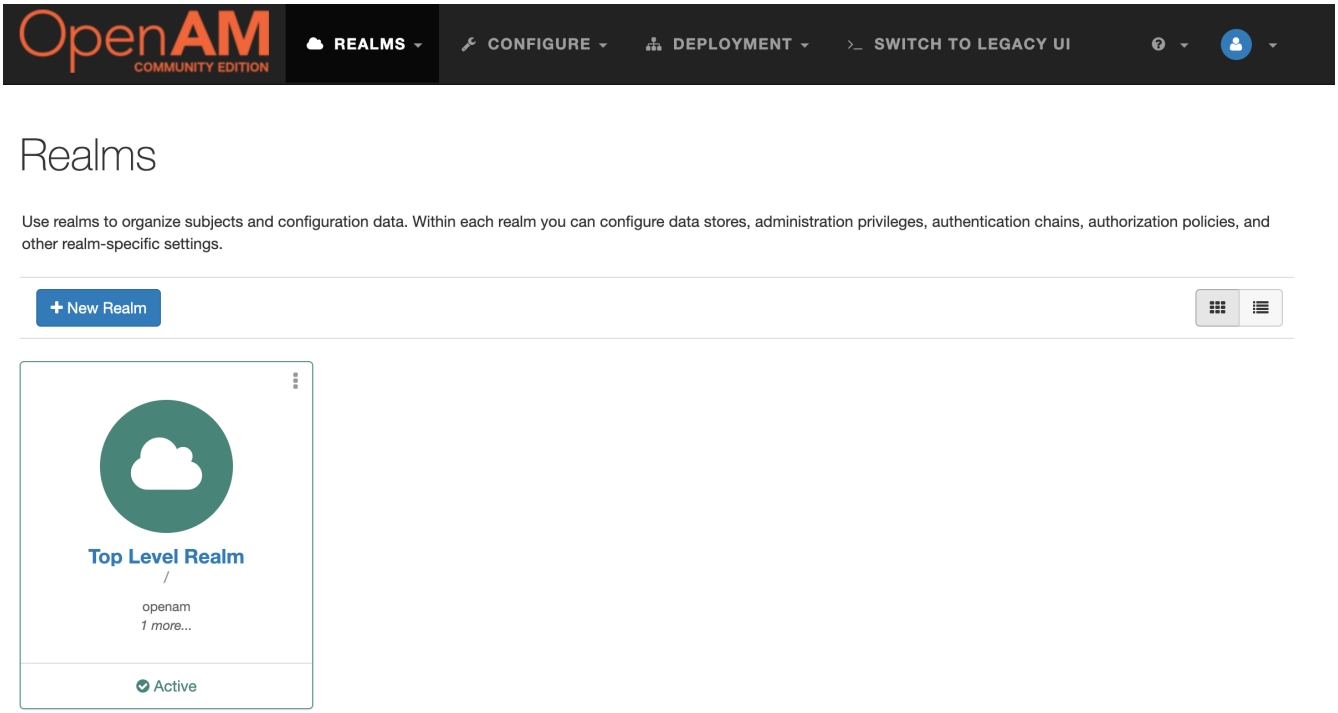
```
docker exec -w '/usr/openam/ssoconfiguratortools' openam bash -c \
```

```
'echo "ACCEPT_LICENSES=true
SERVER_URL=http://openam.example.org:8080
DEPLOYMENT_URI=/$OPENAM_PATH
BASE_DIR=$OPENAM_DATA_DIR
locale=en_US
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=passw0rd
AMLDAPUSERPASSWD=p@passw0rd
COOKIE_DOMAIN=example.org
ACCEPT_LICENSES=true
DATA_STORE=embedded
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=openam.example.org
DIRECTORY_PORT=50389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1689
ROOT_SUFFIX=dc=openam,dc=example,dc=org
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=passw0rd" > conf.file && java -jar openam-configurator-tool*.jar
--file conf.file'
```

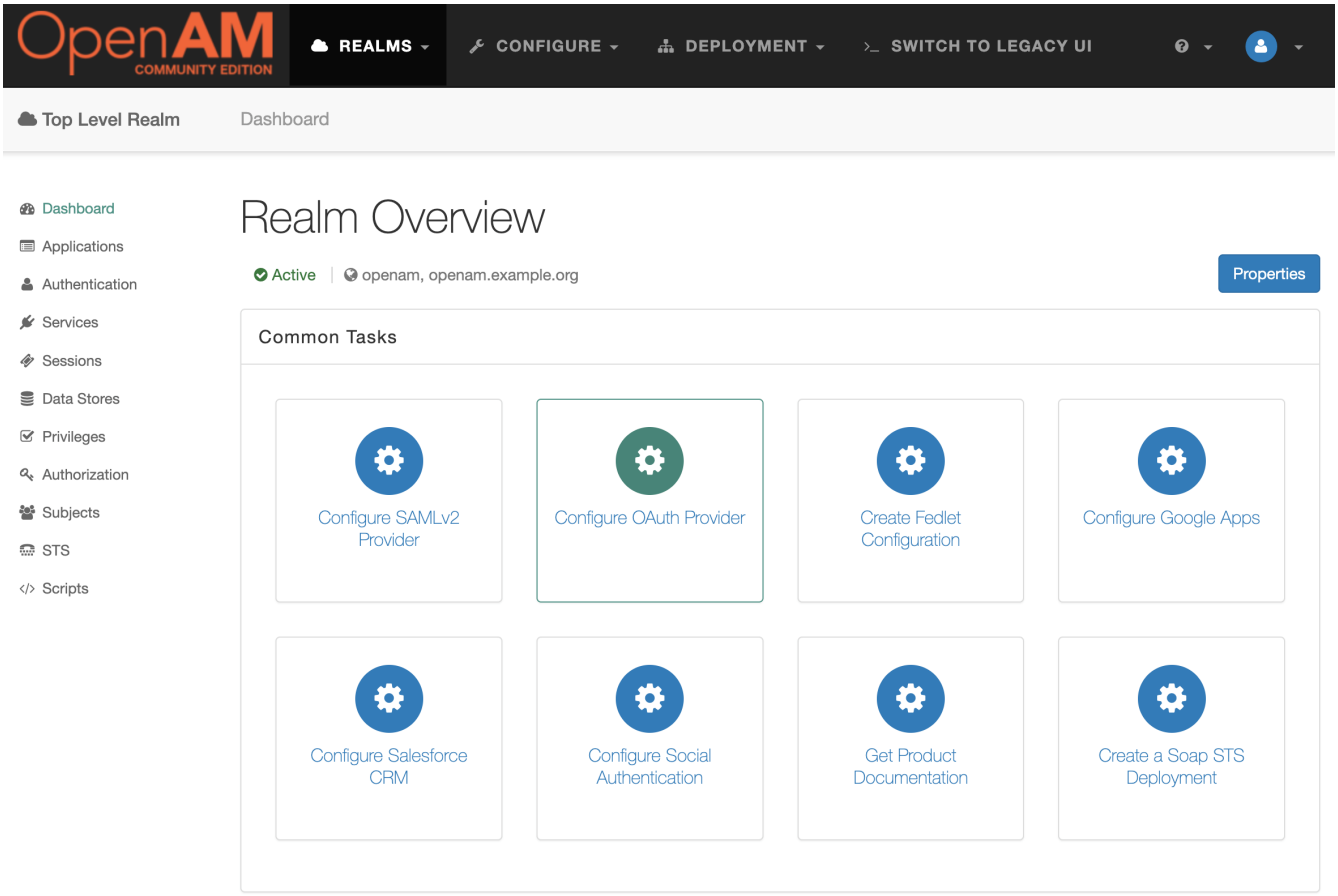
Configuring OAuth 2 in OpenAM

Open the OpenAM console at http://openam.example.org:8080/openam/console. Enter the administrator login and password in the `User Name` and `Password` fields. In this case, they will be `amadmin` and `passw0rd`, respectively.

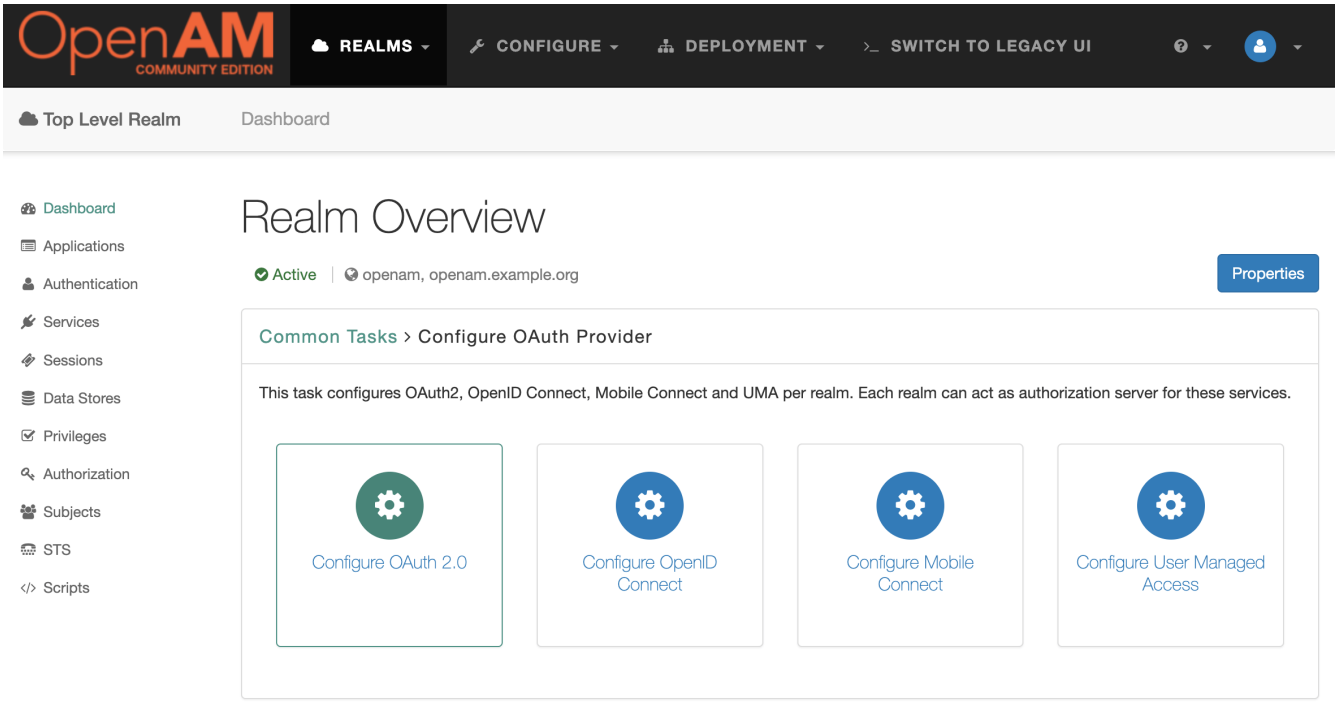
Select `Top Level Realm` from the Realm list.



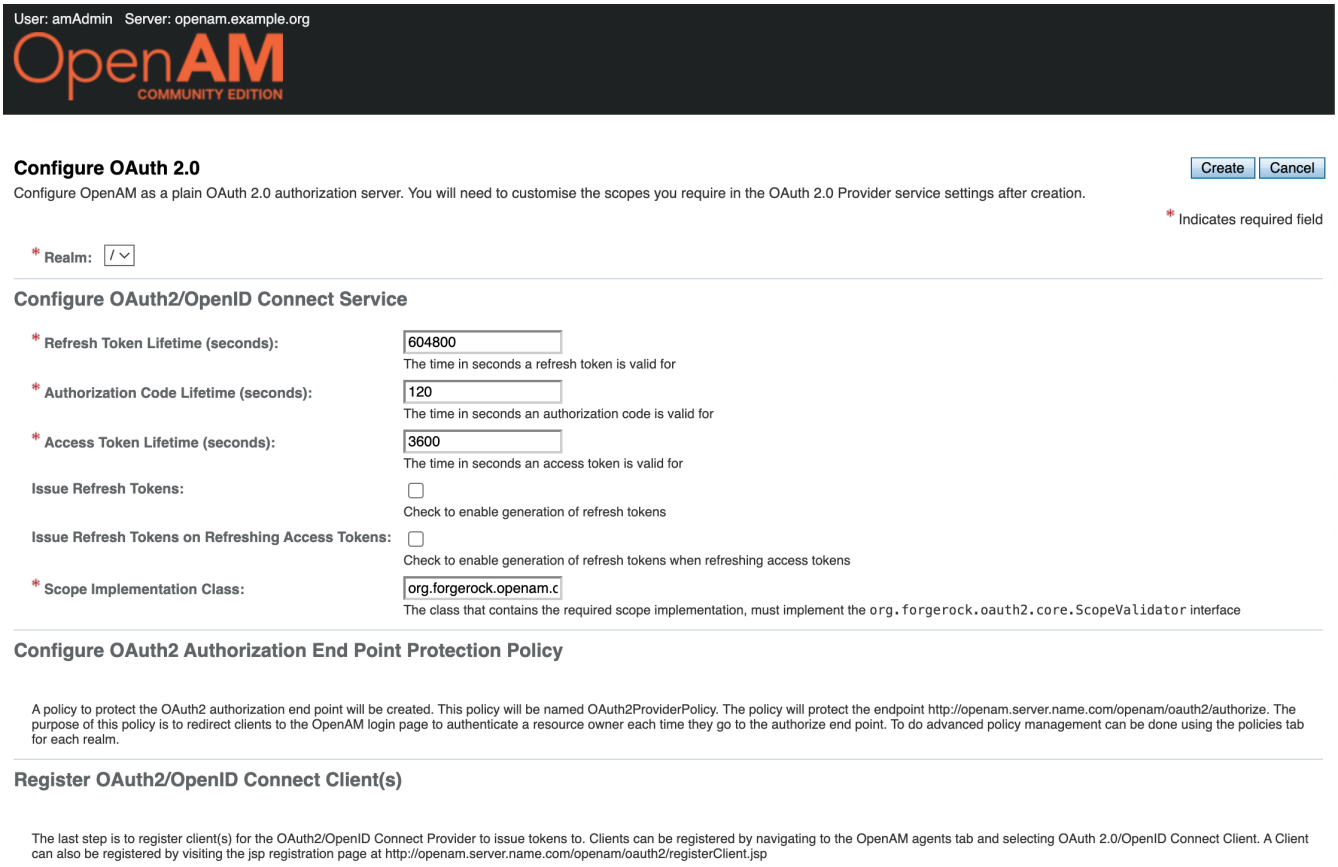
Next, `Configure OAuth Provider`.



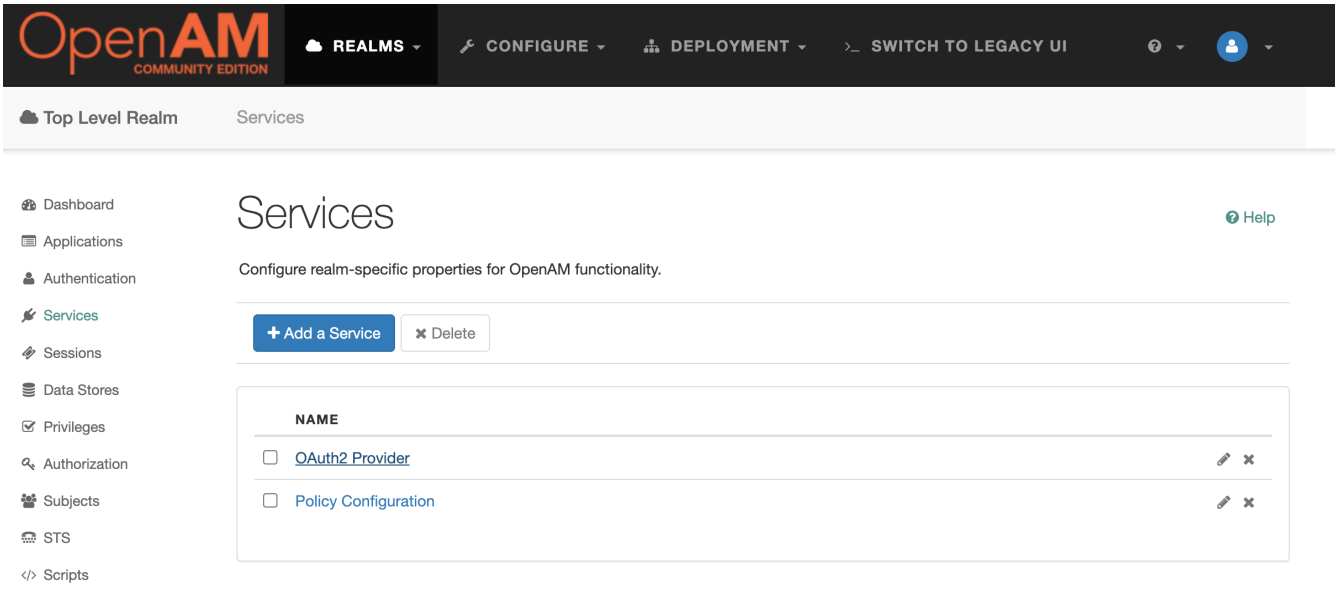
Then select `Configure OAuth 2.0`.



In the form that opens, you can leave the default settings unchanged. Click `Create`.



In the Realm settings, select Services from the menu on the left and open the OAuth2 Provider settings.



Add the value `profile` to the `Scopes` and `Default Clients Scopes` settings. This scope will allow you to obtain basic information about the user. Enable the `Issue Refresh Tokens` and `Issue Refresh Tokens on Refreshing Access Tokens` options. Also, allow dynamic client registration by enabling the `Allow Open Dynamic Client Registration` option. This will allow the MCP client (VS Code) to automatically register with OpenAM without requiring any additional action from the user.

For more information on configuring OpenAM, please refer to the documentation.

Configuring OpenIG

OpenIG will be responsible for authorizing requests. It will check the validity of `access_token` issued by OpenAM and proxy requests to OpenAM and the MCP server.

Now let's check the OpenIG route configuration for proxying requests.

Proxying requests to the MCP server.

The route will receive the `access_token` issued by OpenAM, passed in the `Authorization` header. If the `access_token` is valid, it will pass the request to the MCP server and return a response. If the `access_token` is invalid, OpenIG will return HTTP status 401.

`openig-config/config/routes/10-mcp.json`

```
{
   "name": "${matches(request.uri.path, '^/mcp')}",
   "condition": "${matches(request.uri.path, '^/mcp')}",
   "monitor": true,
   "timer": true,
   "handler": {
      "type": "Chain",
      "config": {
         "filters": [
            {
               "type": "OAuth2ResourceServerFilter",
               "config": {
                  "requireHttps": false,
                  "providerHandler": "ClientHandler",
                  "scopes": [
                     "profile"
                  ],
                  "tokenInfoEndpoint":
"${system['openam'].concat('/oauth2/tokeninfo')}"
               }
            },
            {
               "type": "ConditionEnforcementFilter",
               "config": {
                  "condition": "${not empty contexts['oauth2']}",
                  "failureHandler": "RequireAuth"
               }
            }
         ],
         "handler": "EndpointHandler"
      }
   },
   "heap": [
      {
         "name": "RequireAuth",
```

```
         "type": "StaticResponseHandler",
         "config": {
            "status": 401,
            "headers": {
               "WWW-Authenticate": [
                  "Bearer realm=\"OpenIG\""
               ]
            },
            "entity": "Authentication required"
         }
      },
      {
         "name": "EndpointHandler",
         "type": "DispatchHandler",
         "config": {
            "bindings": [
               {
                  "handler": "ClientHandler",
                  "baseURI": "http://time-mcp-server:8080/mcp"
               }
            ]
         }
      }
   ]
}
```

The route consists of two filters. The first filter, `OAuth2ResourceServerFilter`, validates the `access_token` and, if successful, writes the data received from the `access_token` to the request context. The second filter, `ConditionEnforcementFilter`, checks the context and, if successful, forwards the request to the MCP server. Otherwise, it returns HTTP status 401.

Let's make an unauthorized request to the MCP server and verify that OpenIG requires authorization.

```
curl -v http://openig.example.org:8081/mcp
*   Trying 127.0.0.1:8081...
* Connected to openig.example.org (127.0.0.1) port 8081 (#0)
> GET /mcp HTTP/1.1
> Host: openig.example.org:8081
> User-Agent: curl/8.1.2
> Accept: */*
>
< HTTP/1.1 401
< WWW-Authenticate: Bearer realm="OpenIG"
< Content-Length: 0
< Date: Mon, 22 Sep 2025 08:00:48 GMT
```

Proxying to `.well-known` endpoints

According to the MCP specification, the client obtains data about the authorization server from endpoints located at the URL `<MCP server host>/.well-known/*`. The endpoints are located on OpenAM at the URL `<OpenAM host>/openam/.well-known`. The route for forwarding HTTP requests to MCP on OpenAM is as follows:

openig-config/config/routes/20-well-known.json

```
{
  "name": "${matches(request.uri.path, '^/.well-known/.*}",
  "condition": "${matches(request.uri.path, '^/.well-known/.*')}",
  "monitor": true,
  "timer": true,
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "HeaderFilter",
          "config": {
            "messageType": "REQUEST",
            "add": {
              "Host": [
                "${matchingGroups(system['openam'],\"(http|https):\/\/(.[^\/]*)
\")[2]}"
              ]
            },
            "remove": [
              "Host",
              "Origin"
            ]
          }
        }
      ],
      "handler": "EndpointHandler"
    }
  },
  "heap": [
    {
      "name": "EndpointHandler",
      "type": "DispatchHandler",
      "config": {
        "bindings": [
          {
            "expression": "${matches(request.uri.path, '^/.well-known/openid-
configuration$')}",
            "handler": "ClientHandler",
            "baseURI": "${system['openam'].concat('/oauth2/.well-known/openid-
configuration')}"
          }
        ]
      }
```

```
    }
  ]
}
```

The `HeaderFilter` filter adds the OpenAM Host HTTP header specified in the `openam` system parameter in the `docker-compose.yaml` file, and the `EndpointHandler` handler forwards the request to the `/openam/.well-known/openid-configuration` endpoint deployed in the `openam` Docker container.

Let's check the endpoint operation:

```
curl -v http://openig.example.org:8081/.well-known/openid-configuration

{
  "acr_values_supported" : [],
  "authorization_endpoint" :
"http://openam.example.org:8080/openam/oauth2/authorize",
  "check_session_iframe" :
"http://openam.example.org:8080/openam/oauth2/connect/checkSession",
  "claims_parameter_supported" : false,
  "claims_supported" : [],
  "device_authorization_endpoint" :
"http://openam.example.org:8080/openam/oauth2/device/code",
  "end_session_endpoint" :
"http://openam.example.org:8080/openam/oauth2/connect/endSession",
  "id_token_encryption_alg_values_supported" : [
     "RSA-OAEP",
     "RSA-OAEP-256",
     "A128KW",
     "RSA1_5",
     "A256KW",
     "dir",
     "A192KW"
  ],
  "id_token_encryption_enc_values_supported" : [
     "A256GCM",
     "A192GCM",
     "A128GCM",
     "A128CBC-HS256",
     "A192CBC-HS384",
     "A256CBC-HS512"
  ],
  "id_token_signing_alg_values_supported" : [
     "ES384",
     "HS256",
     "HS512",
     "ES256",
     "RS256",
     "HS384",
     "ES512"
  ],
```

```
   "issuer" : "http://openam.example.org:8080/openam/oauth2",
   "jwks_uri" : "http://openam.example.org:8080/openam/oauth2/connect/jwk_uri",
   "registration_endpoint" :
"http://openam.example.org:8080/openam/oauth2/connect/register",
   "response_types_supported" : [
      "code",
      "code token",
      "token"
   ],
   "scopes_supported" : [],
   "subject_types_supported" : [
      "public"
   ],
   "token_endpoint" : "http://openam.example.org:8080/openam/oauth2/access_token",
   "token_endpoint_auth_methods_supported" : [
      "client_secret_post",
      "private_key_jwt",
      "none",
      "client_secret_basic"
   ],
   "userinfo_endpoint" : "http://openam.example.org:8080/openam/oauth2/userinfo",
   "version" : "3.0"
}
```

For more details on configuring OpenIG, please refer to the documentation.

Configuring VS Code to work with the MCP server

You must have the extensions for working with Copilot, GitHub Copilot, and GitHub Copilot Chat installed and configured. Instructions on how to do this are available at: https://code.visualstudio.com/docs/copilot/setup.

Add the MCP server to VS Code.

For example, to add MCP to your workspace, create a file named `mcp.json` in the `.vscode` directory of your workspace:

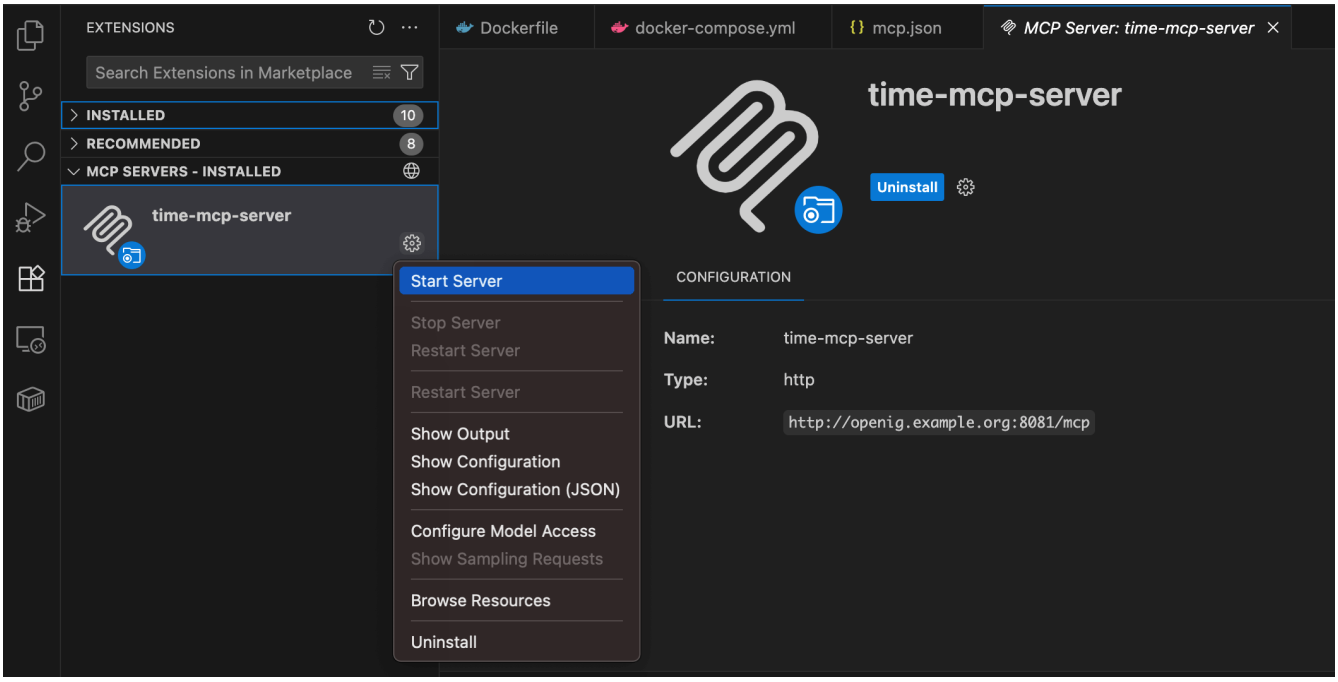
`mcp.json`:

```
{
  "servers": {
    "time-mcp-server": {
      "type": "http",
      "url": "http://openig.example.org:8081/mcp"
    }
  }
}
```

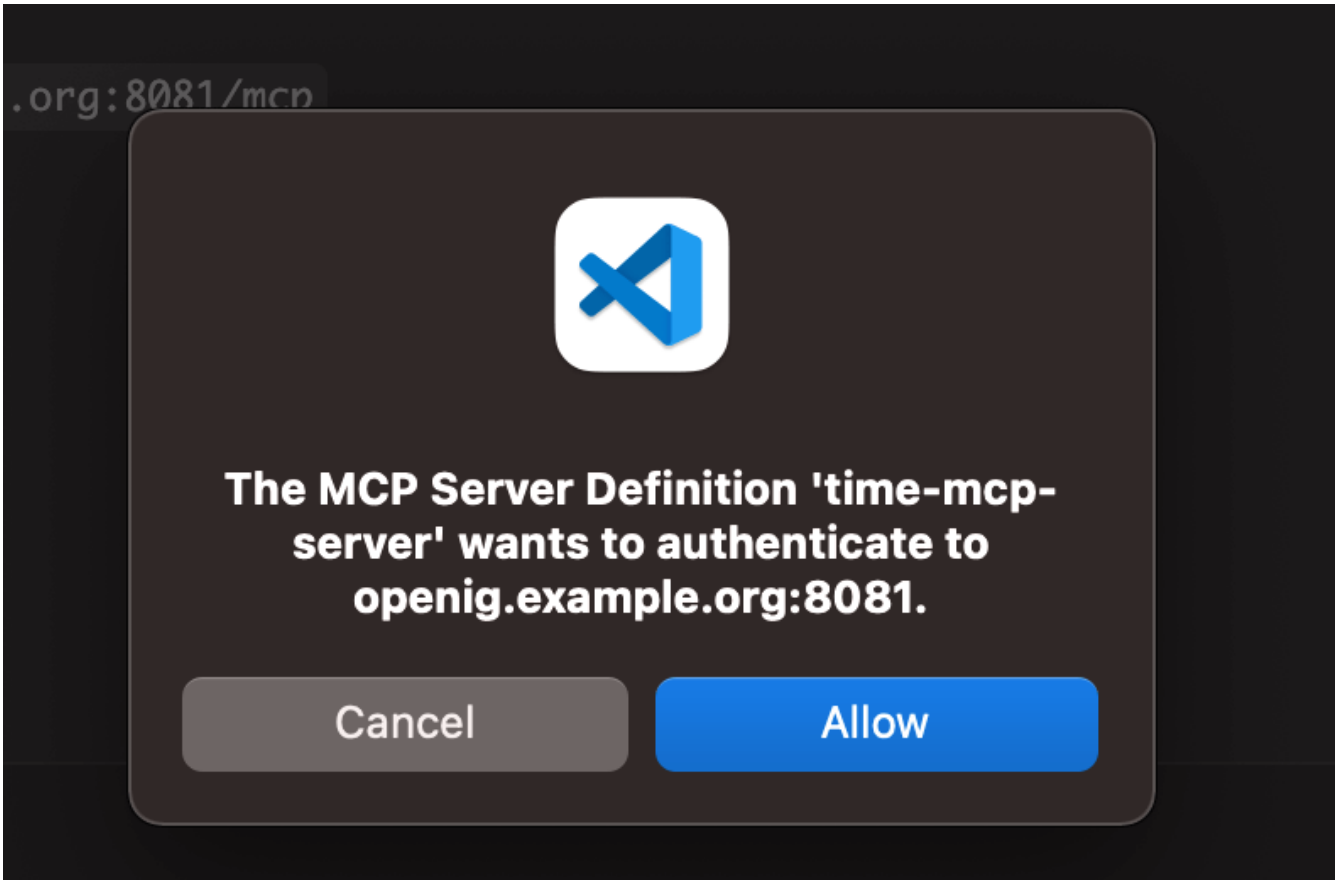
Other ways to add an MCP server are described at:

https://code.visualstudio.com/docs/copilot/customization/mcp-servers#_add-an-mcp-server

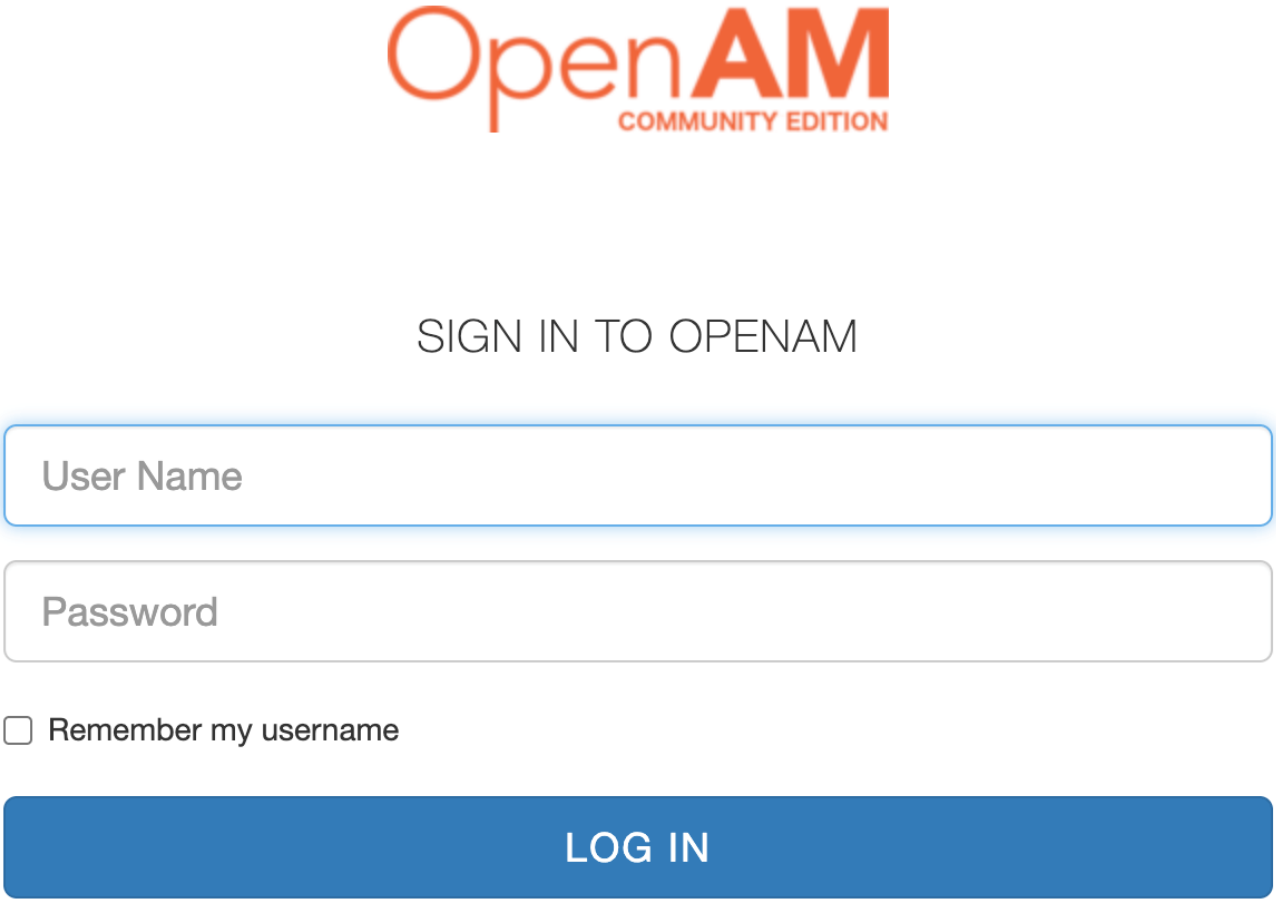
In the VS Code extensions list, click on the settings for the added MCP server and select `Start Server` from the menu that appears.



Allow the MCP server to authenticate on the OpenIG host

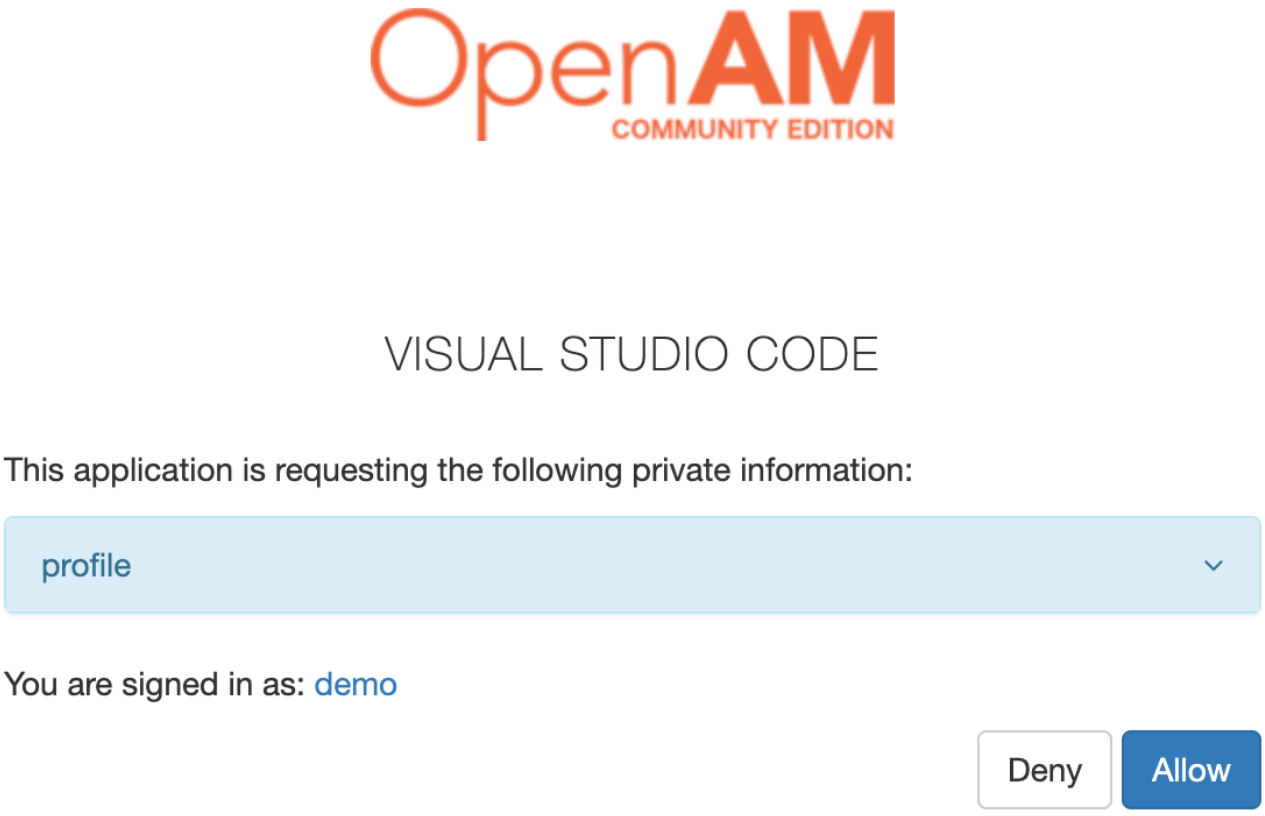


A browser window with authentication will open.



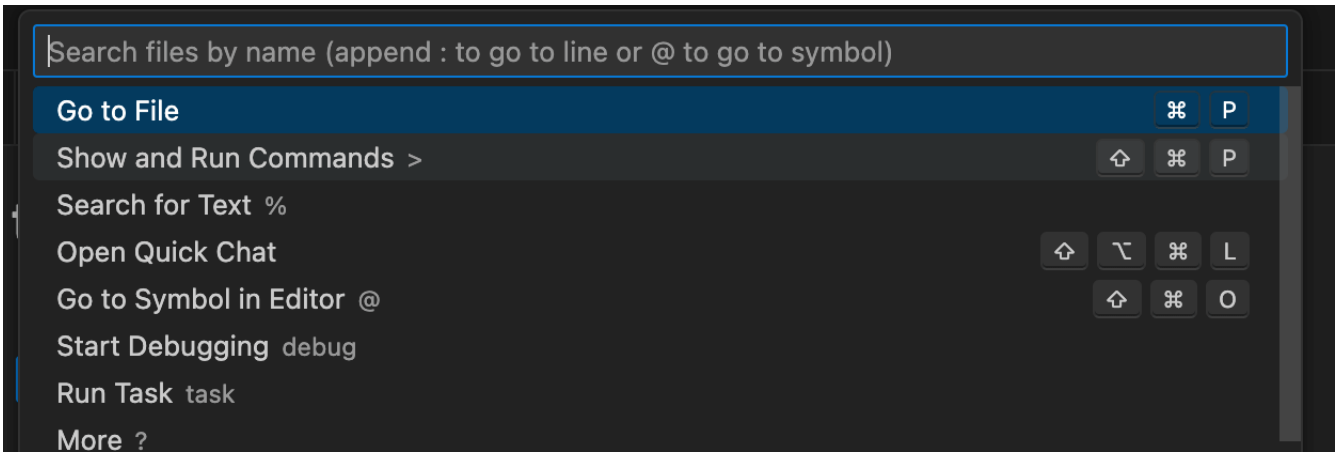
Enter the login and password for the test user: `demo` and `changeit`, respectively

Confirm access to data for the Visual Studio Code application. If you want to disable the data access confirmation dialog, enable `Allow clients to skip consent` in the OAuth2 Provider settings in OpenAM.

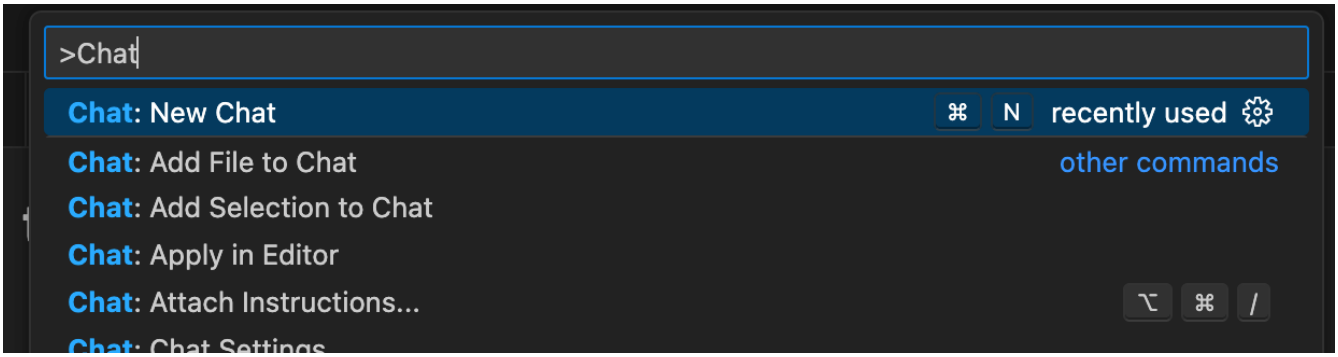


After confirming, you will be redirected back to VS Code.

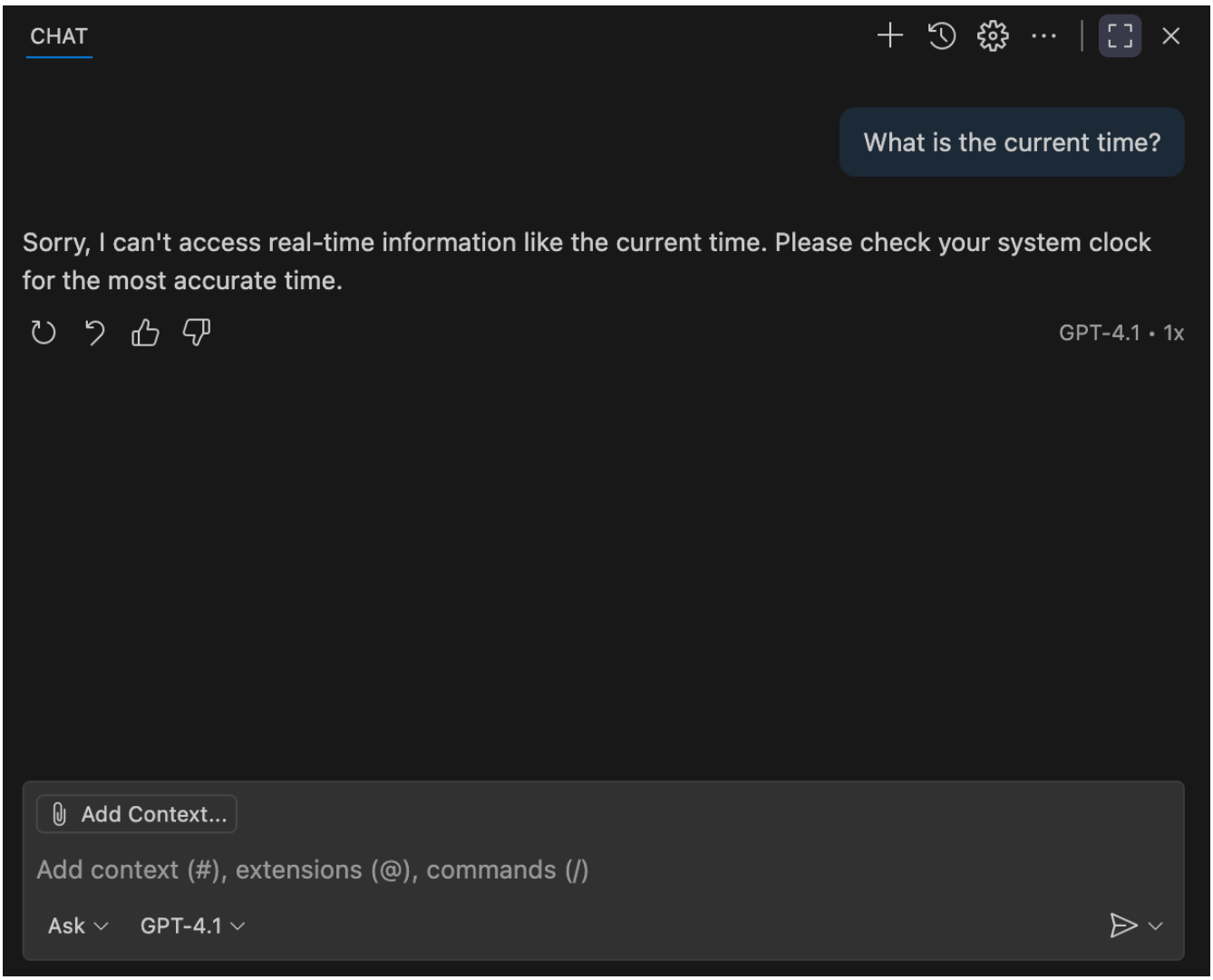
Open the chat with GitHub Copilot. To do this, select **Show and Run Commands** from the command menu:



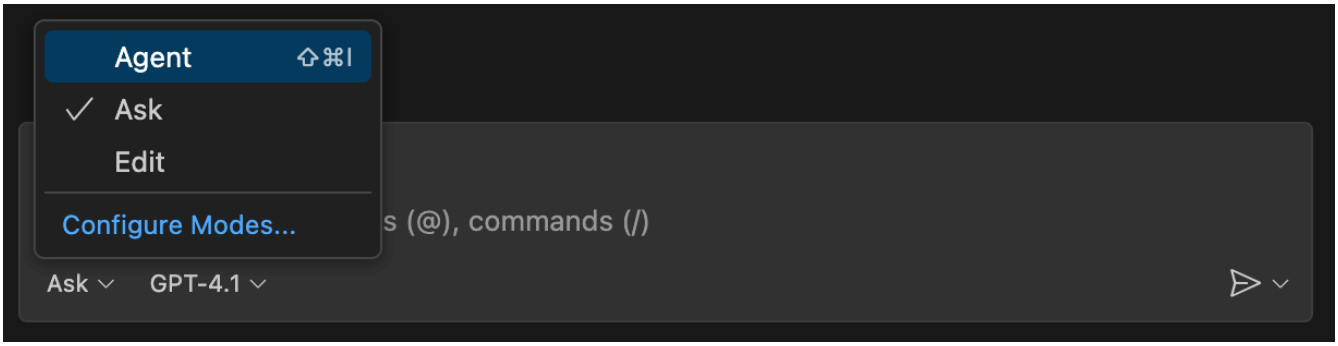
Then select **Chat: New Chat**:



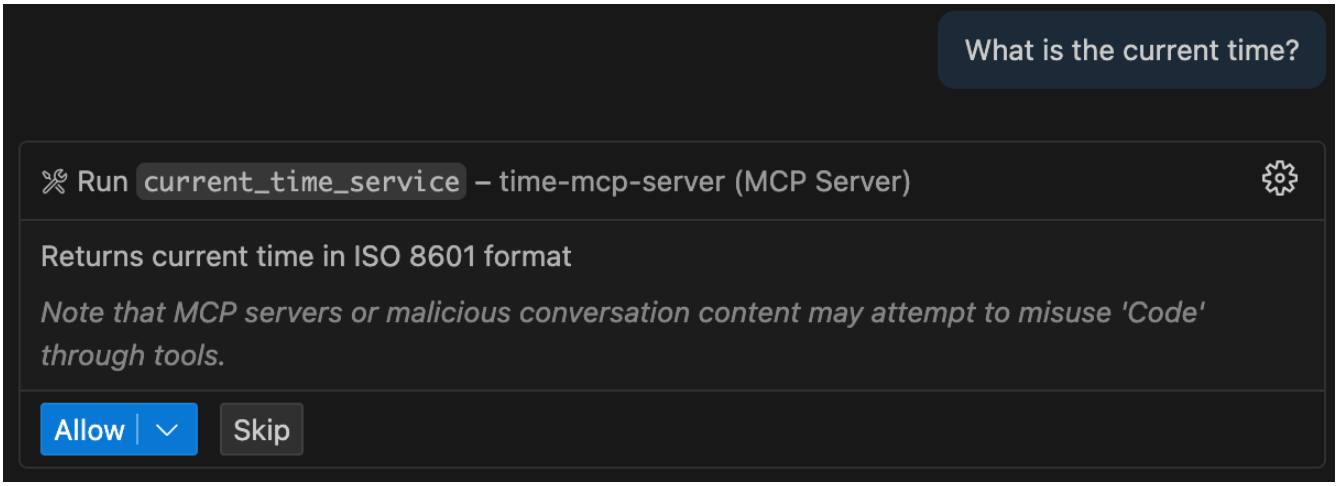
In the chat window that opens, enter the question: `What is the current time?`. Copilot will respond that it does not have access to the current time:



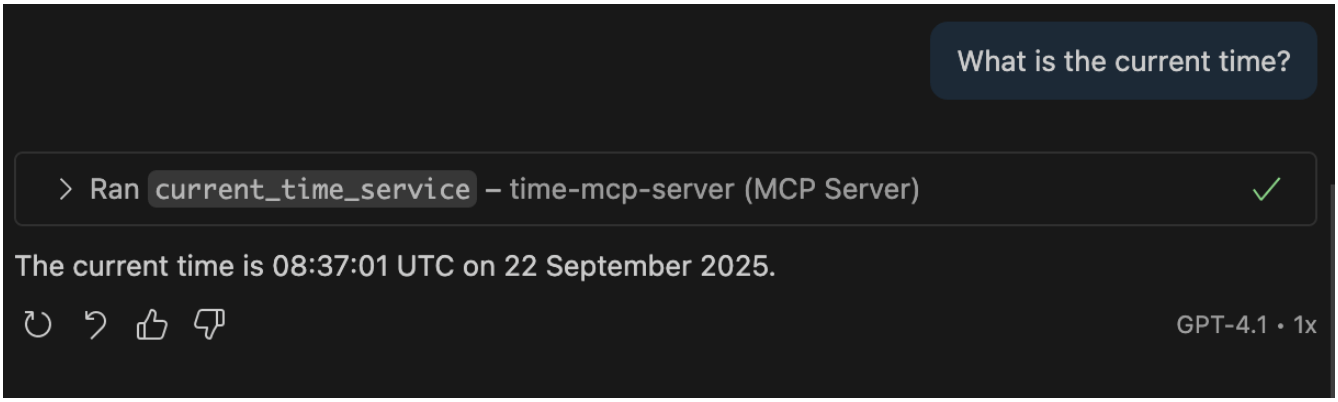
Now switch the chat to Agent mode at the bottom and ask the question again:



Allow access to the function for obtaining the current time in the MCP server:



Copilot will receive information about the current time from the MCP server and return the correct response.



Conclusion

In this article, we demonstrated the practical integration of OpenAM and OpenIG to provide secure access to the MCP server based on OAuth 2.1. OpenAM acts as a reliable authentication and authorization center, issuing and validating tokens, while OpenIG filters requests, blocking unauthorized access and proxying traffic to protected resources. This approach minimizes the risk of sensitive data leaks - from internal repositories to customer databases.

Download the source code from GitHub, test the configuration, and integrate it into your projects. For in-depth study, refer to the official documentation: OpenAM and OpenIG.

How to Setup 2FA with Google Authenticator in OpenAM

- Introduction
- OpenAM Setup
  - Setup Authentication Module
  - Setup Authentication Chain
  - Setup User Accounts
- Test Solution

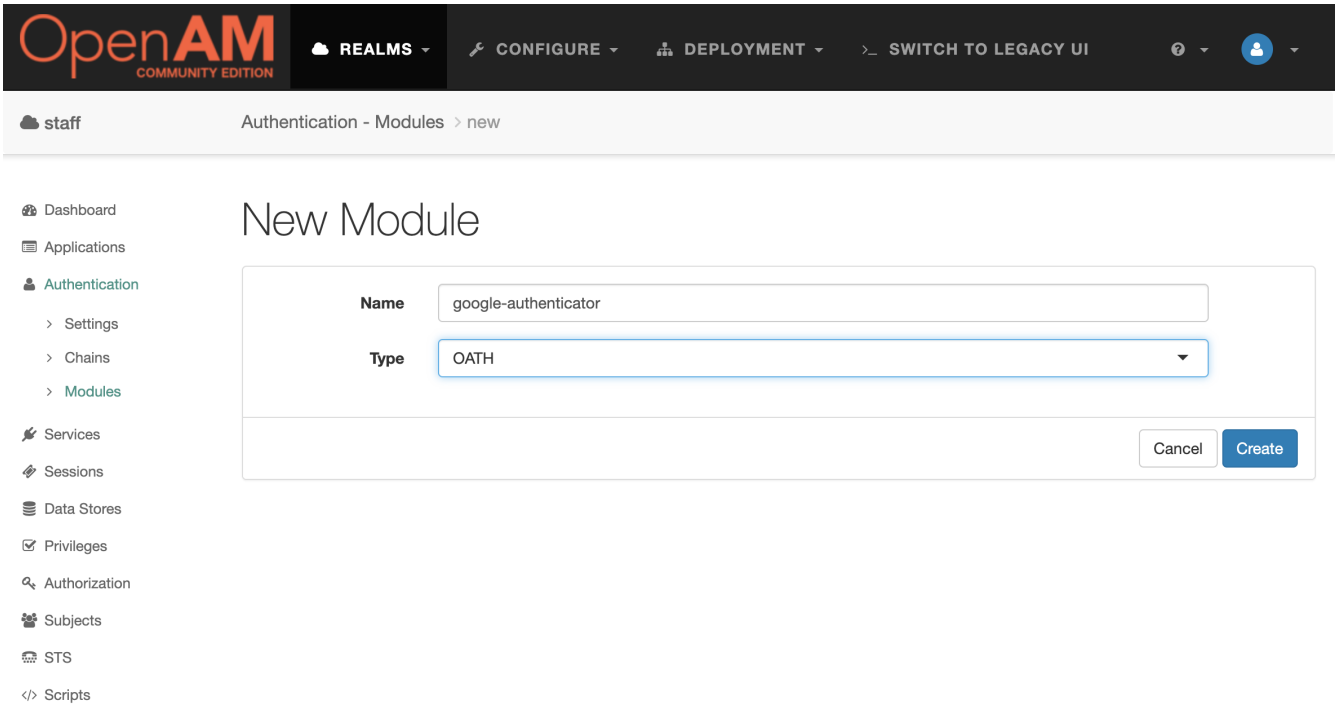
Introduction

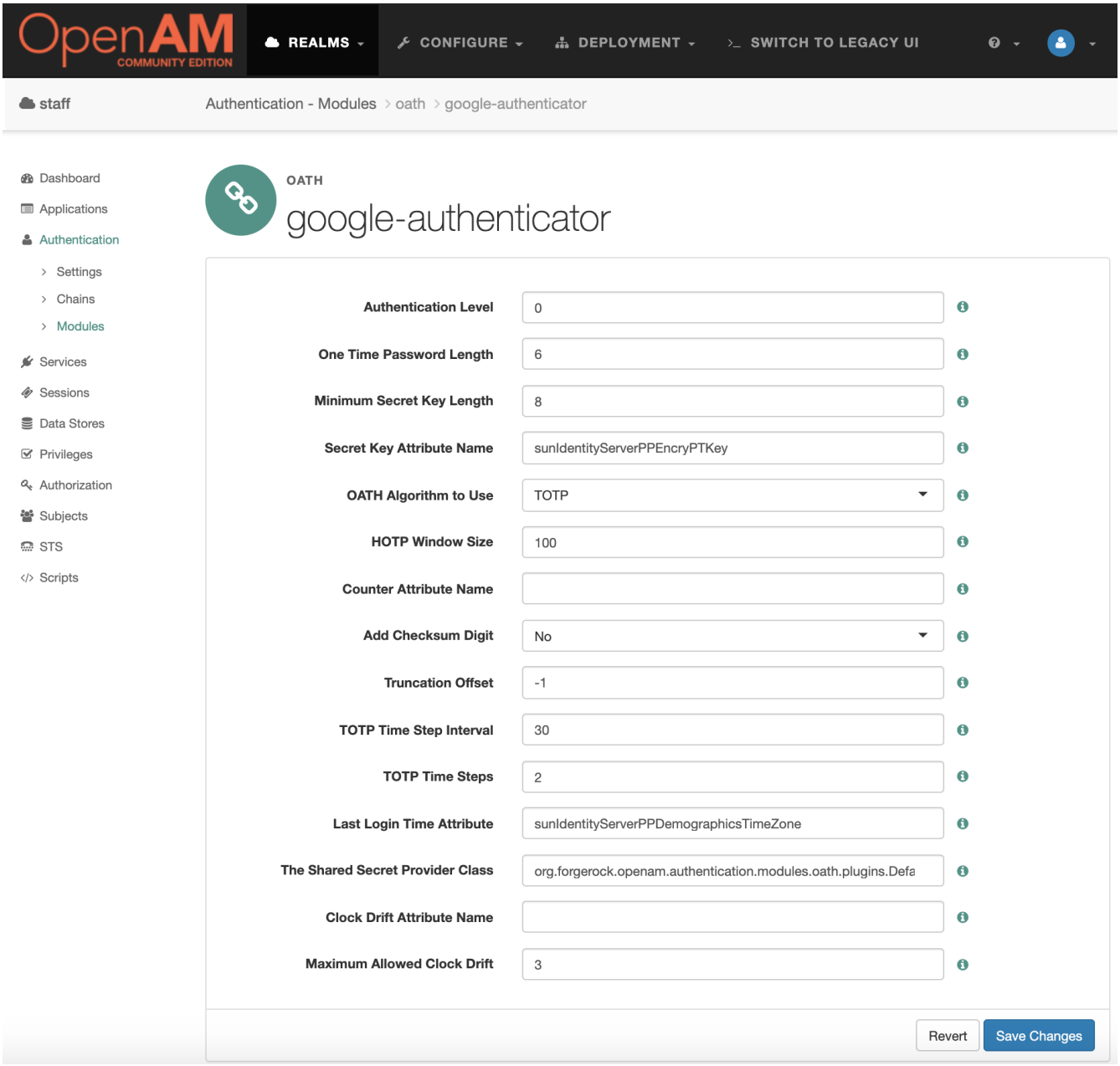
Google Authenticator is one of the most commonly used applications for two-factor authentication (2FA). It runs both on Android and iOS devices. In the following article, we will describe how to set up 2FA in OpenAM with Google Authenticator.

OpenAM Setup

Setup Authentication Module

For example, we have realm `staff` where we want to set up 2FA with Google Authenticator. Login into admin console as `amadmin` user, go to realm `staff`, and create new `OATH` module `google-authenticator` with the following settings:



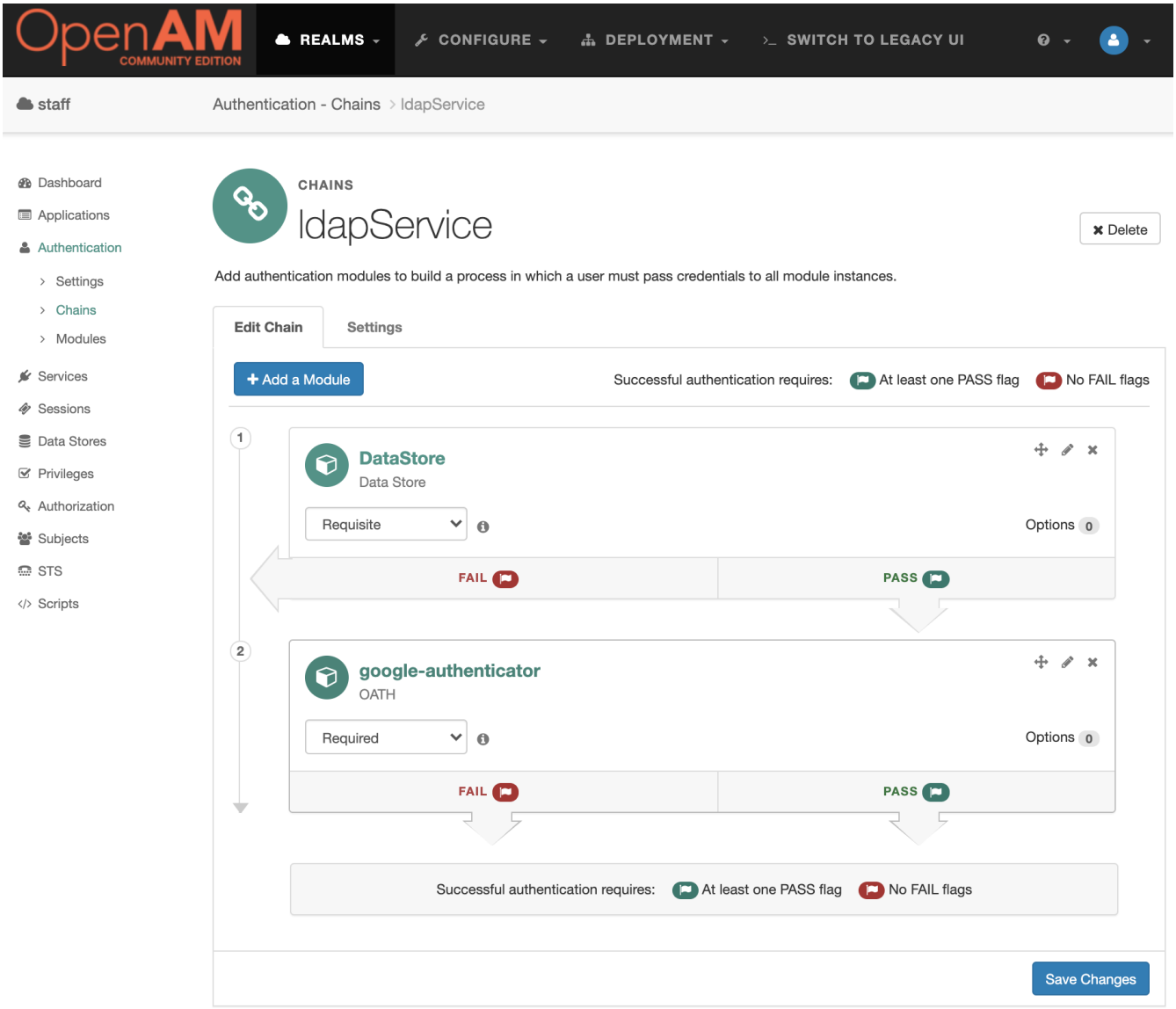


There are some important settings:

- **One Time Password Length**: Should be 6 as in Google Authenticator by default
- **Secret Key Attribute Name**: User attribute in where a secret key is stored.
- **OATH Algorithm to Use**: For Google Authenticator should be TOTP
- **Last Login Time Attribute**: User attribute, where users last login time in UNIX format is stored.
- **Maximum Allowed Clock Drift**: Should be greater than TOTP time steps, so in the following example it is 3.

Setup Authentication Chain

By default, there is `ldapService` authentication chain, which uses only `DataStore` authentication module. Add `google-authenticator` module in the chain.



Setup User Accounts

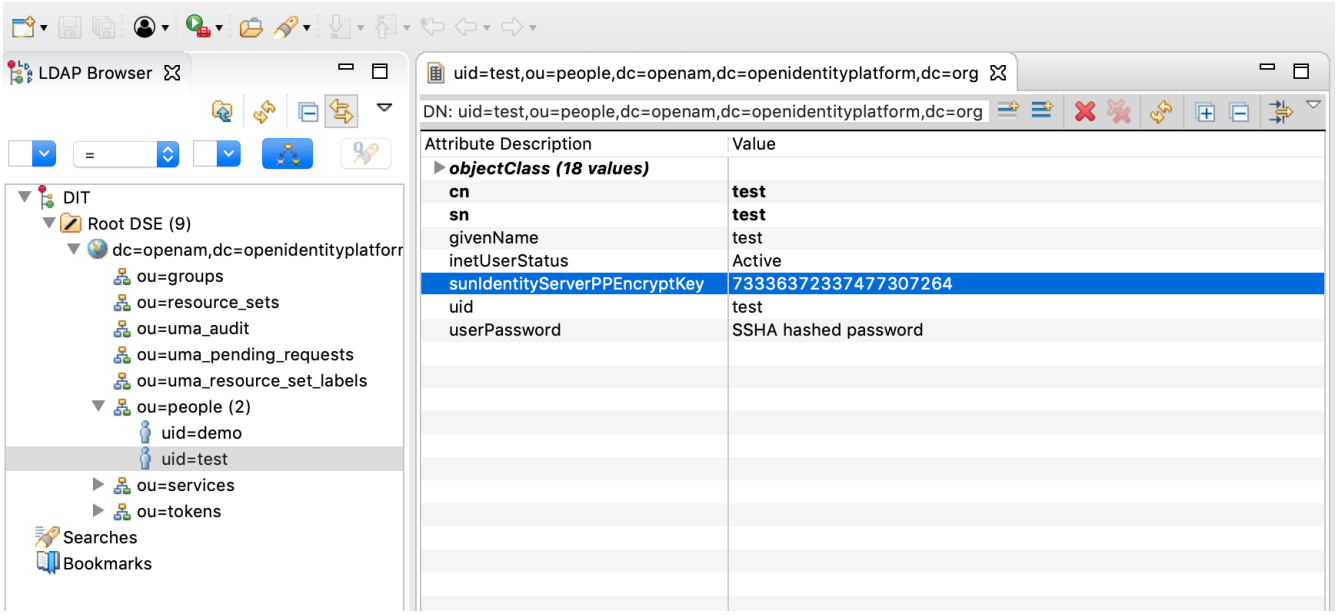
The next part would be a little bit tricky. To authenticate in Google Authenticator we need to set up user accounts first. Then generate for each user account shared secret. As an example, we will take `test` account and for this account shared secret will be `s3cr3tw0rd`.

Encode into Base16. You can or use online service https://simplycalc.com/base16-encode.php either use the following Java function:

```
private String toBase16(String str) {
  return String.format("%x", new BigInteger(1, str.getBytes(Charset.
defaultCharset())));
}
```

Base16 encoded value is `73336372337477307264`.

Open User account properties in Apache Directory Studio (or any other LDAP client application) and add `sunIdentityServerPPEncryptKey` value `73336372337477307264` as shown on the picture below.



Then generate a QR code to register the user in the Google Authenticator application. QR for Google Authenticator is URI in the following format `otpauth://totp/<account id>@<issuer>?secret=<base32 encoded secret>&issuer=<Issuer Name>`

Encode secret as Base32. You can use or online service https://simplycalc.com/base32-encode.php, ether following Java function:

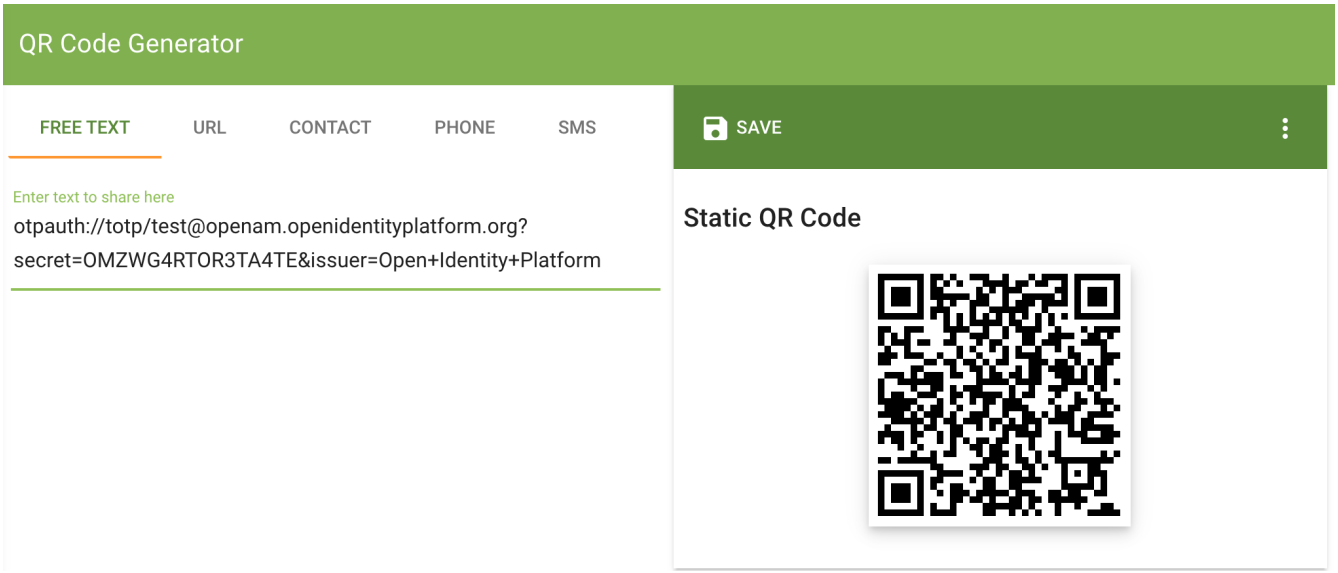
```
import org.apache.commons.codec.binary.Base32;

private String toBase32(String str) {
    return new Base32().encodeAsString(str.getBytes());
}
```

Base32 encoded value is `OMZWG4RTOR3TA4TE`

The final URI will be: `otpauth://totp/test@openam.openidentityplatform.org?secret=OMZWG4RTOR3TA4TE&issuer=Open+Identity+Platform`

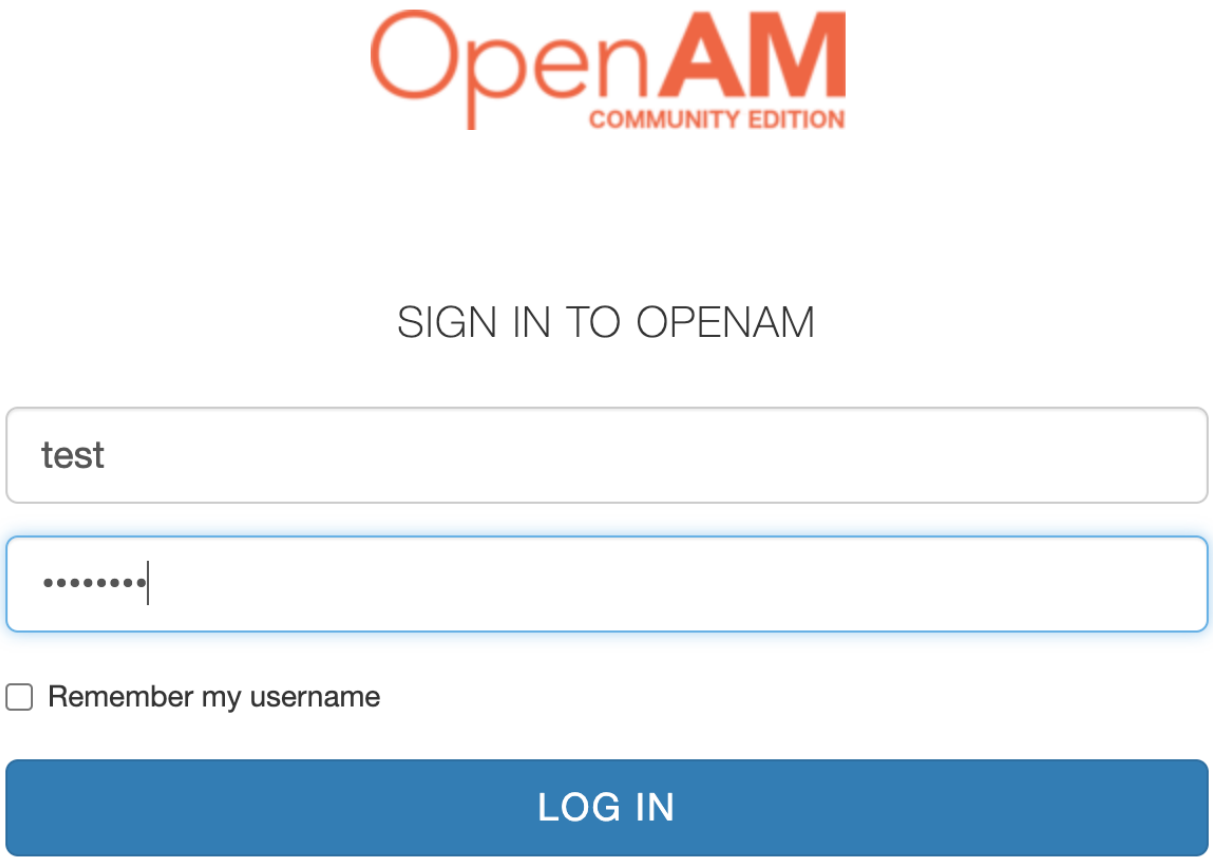
Let's use the online service https://www.the-qrcode-generator.com to generate a QR code image:



Scan QR in Google Authenticator application. A new account will be added.

Test Solution

Let's sign in into realm `staff` c with `test` account:



You will be prompted for OTP from Google Authenticator.



Enter the code from Google Authenticator, and you will be successfully authenticated.

How to Setup Active Directory Authentication in a Spring Boot Application

Introduction

Almost every organization uses Active Directory to manage employee accounts. And using existing accounts to access corporate applications is a recommendend practice. In this article, we will configure authentication in the demo Spring Boot application via an Active Directory server in OpenAM.

OpenAM Configuration

Install OpenAM

Let OpenAM hostname is `openam.example.org`, and the Spring Boot application hostname is `app.example.org`.

If you already have OpenAM installed, you can skip this step. The easiest way to deploy OpenAM is in a Docker container. Before starting, add the hostname and IP address to the `hosts` file, for example `127.0.0.1 app.example.org openam.example.org` .

On Windows systems, the `hosts` file is located at `C:Windows/System32/drivers/etc/hosts` , on Linux and Mac it is located at `/etc/hosts` .

Create a network in Docker for OpenAM

```
docker network create openam
```

Next, start the Docker container OpenAM Execute the following command:

```
docker run -h openam.example.org -p 8080:8080 --network openam --name openam
openidentityplatform/openam
```

Once the server is up and running, run the initial OpenAM configuration. Run the following command:

```
docker exec -w '/usr/openam/ssoconfiguratortools' openam bash -c \
'echo "ACCEPT_LICENSES=true
SERVER_URL=http://openam.example.org:8080
DEPLOYMENT_URI=/$OPENAM_PATH
BASE_DIR=$OPENAM_DATA_DIR
locale=en_US
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=passw0rd
```

```
AMLDAPUSERPASSWD=p@passw0rd
COOKIE_DOMAIN=example.org
ACCEPT_LICENSES=true
DATA_STORE=embedded
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=openam.example.org
DIRECTORY_PORT=50389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1689
ROOT_SUFFIX=dc=openam,dc=example,dc=org
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=passw0rd" > conf.file && java -jar openam-configurator-tool*.jar
--file conf.file'
```

After successful configuration you can proceed to further steps.

Note the `COOKIE_DOMAIN` parameter - the authentication session cookie must be set to a common top-level domain for OpenAM and the application.

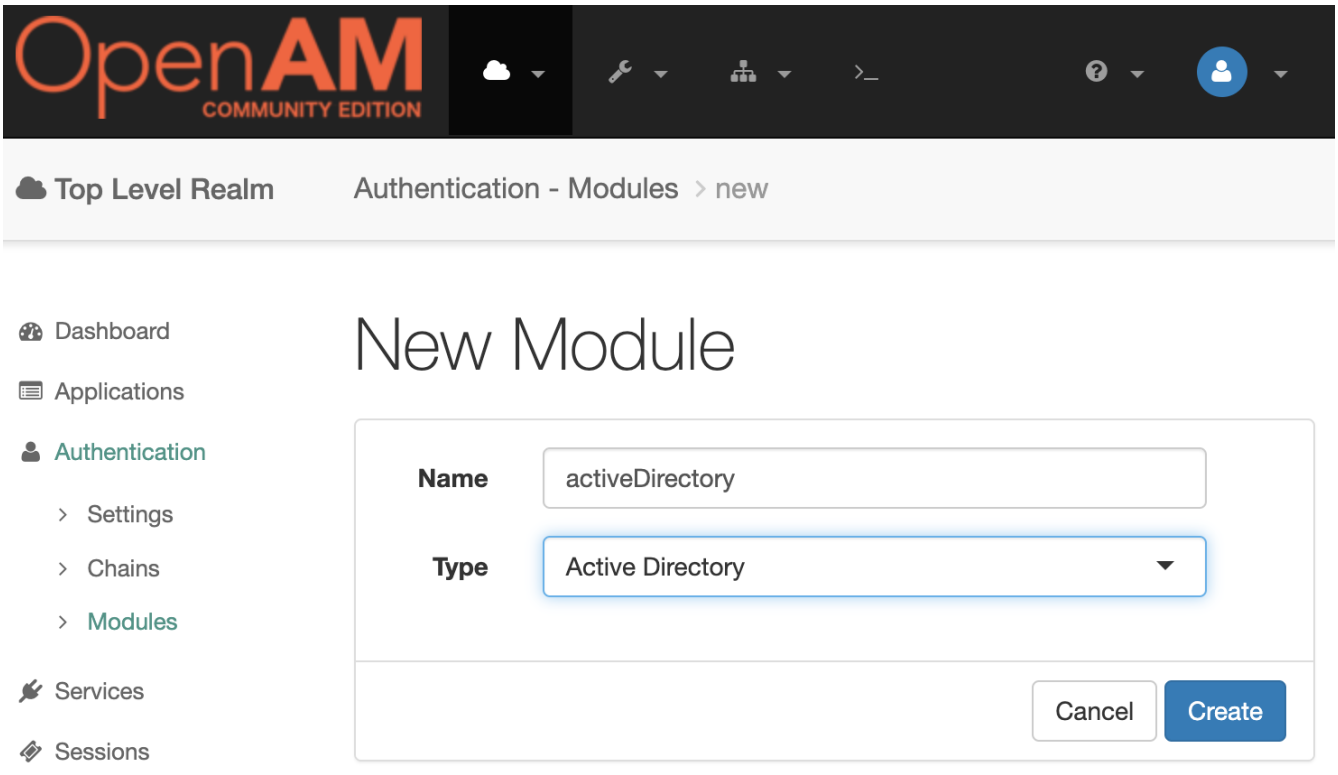
Authentication Module Configuration

Go to the OpenAM administrator console at

http://openam.example.org:8080/openam/XUI/#login/

In the login field enter the value `amadmin`, in the password field enter the value from the `ADMIN_PWD` parameter of the setup command, in this case `passw0rd`.

Select the root realm and select Authentication → Modules from the menu. Create a new Active Directory authentication module.



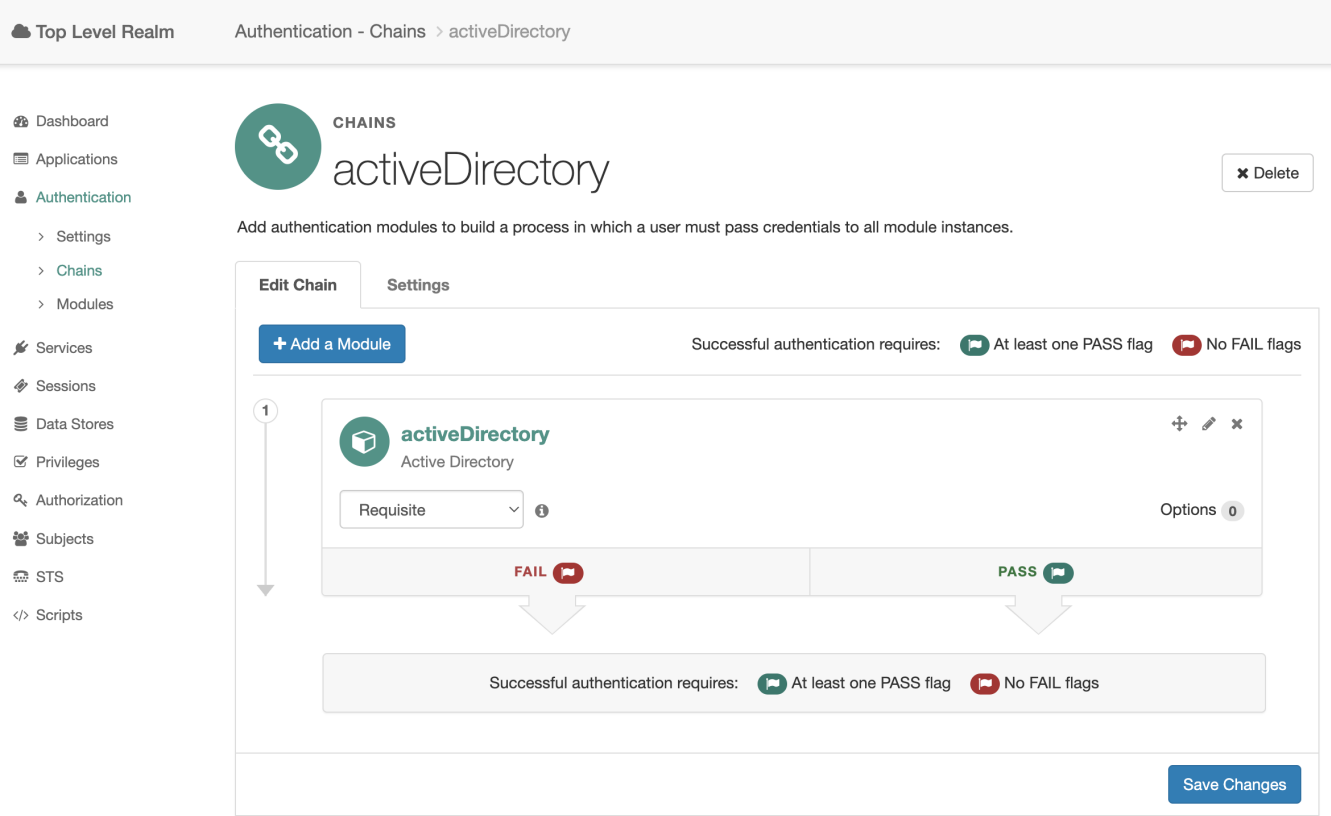
Set the module settings according to the table

Setting	Description
Primary Active Directory Server	AD host and port number, for example: ad.example.org:389
Users Domain	
Bind User DN	Leave blank. If empty, binding authentication is used and you do not need to specify Bind user password, User Search Filter, and DN to Start User Search.

You can leave other settings unchanged.

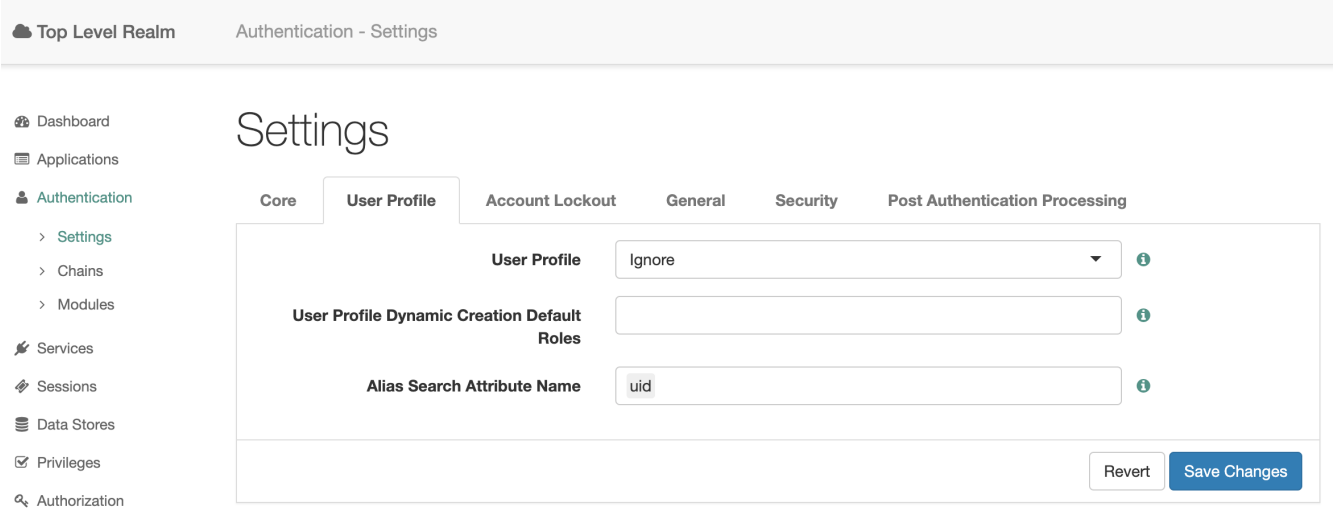
Authentication Chain Configuration

Go to the admin console, select the root realm and select Authentication → Chains from the menu. Create an `activeDirectory` authentication chain with the recently created `activeDirectory` module.



Configure realm

Go to Authentication → Chains for realm and on the User Profile tab, set the `User Profile` setting to `Ignore`.



So, you can authenticate to Active Directory without setting up Active Directory as a User Data Store in OpenAM.

Spring Boot Application Setup

The recommended way is to integrate through Spring Security.

The application uses a filter that can be attached to a Spring REST or a Web Controller.

The filter receives a Cookie with the authentication session ID set by OpenAM on successful authentication from the HttpServletRequest. Then the filter calls the OpenAM API to retrieve the user from the session. If the cookie does not exist or the session retrieved from the cookie is not valid, the user is redirected to authentication.

An example filter is in the listing below:

```
@Component
public class OpenAmAuthFilter implements Filter {

    final static String OPENAM_URI = "http://openam.example.org:8080/openam";

    final static String OPENAM_COOKIE_NAME = "iPlanetDirectoryPro";

    final static String REDIRECT_URI_TEMPLATE = OPENAM_URI.concat
("/XUI#login/&service=activeDirectory&goto=");

    private final String OPENAM_USER_INFO_URI = OPENAM_URI.concat
("/json/users?_action=idFromSession");

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
        HttpServletRequest request = (HttpServletRequest) servletRequest;
        HttpServletResponse response = (HttpServletResponse) servletResponse;

        //reading OpenAM session cookie
```

```
        Optional<Cookie> openamCookie = Optional.empty();
        if(request.getCookies() != null) {
            openamCookie = Arrays.stream(request.getCookies())
                    .filter(c -> c.getName().equals(OPENAM_COOKIE_NAME)).findFirst();
        }
        String redirectUri = request.getRequestURL().toString();
        if(openamCookie.isEmpty()) {
            response.sendRedirect(REDIRECT_URI_TEMPLATE  + URLEncoder.encode
(redirectUri, StandardCharsets.UTF_8));
        } else {
            //retrieve userID from by session cookie value from OpenAM
            String userId = getUserIdFromSession(openamCookie.get().getValue());
            if (userId == null) {
                response.sendRedirect(REDIRECT_URI_TEMPLATE + URLEncoder.encode
(redirectUri, StandardCharsets.UTF_8));
                return;
            }
            request.setAttribute("openam.userId", userId);
            filterChain.doFilter(servletRequest, servletResponse);
        }
    }

    private String getUserIdFromSession(String sessionId) {
        RestTemplate restTemplate = new RestTemplate();
        ParameterizedTypeReference<Map<String, String>> responseType = new
ParameterizedTypeReference<>() {};
        HttpHeaders headers = new HttpHeaders();
        headers.add(OPENAM_COOKIE_NAME, sessionId);
        headers.setContentType(MediaType.APPLICATION_JSON);
        HttpEntity<?> entity = new HttpEntity<>(headers);
        ResponseEntity<Map<String, String>> response
                = restTemplate.exchange(OPENAM_USER_INFO_URI, HttpMethod.POST, entity,
responseType);
        Map<String, String> body = response.getBody();
        if (body == null) {
            return null;
        }
        return body.get("id");
    }
}
```

Test Solution

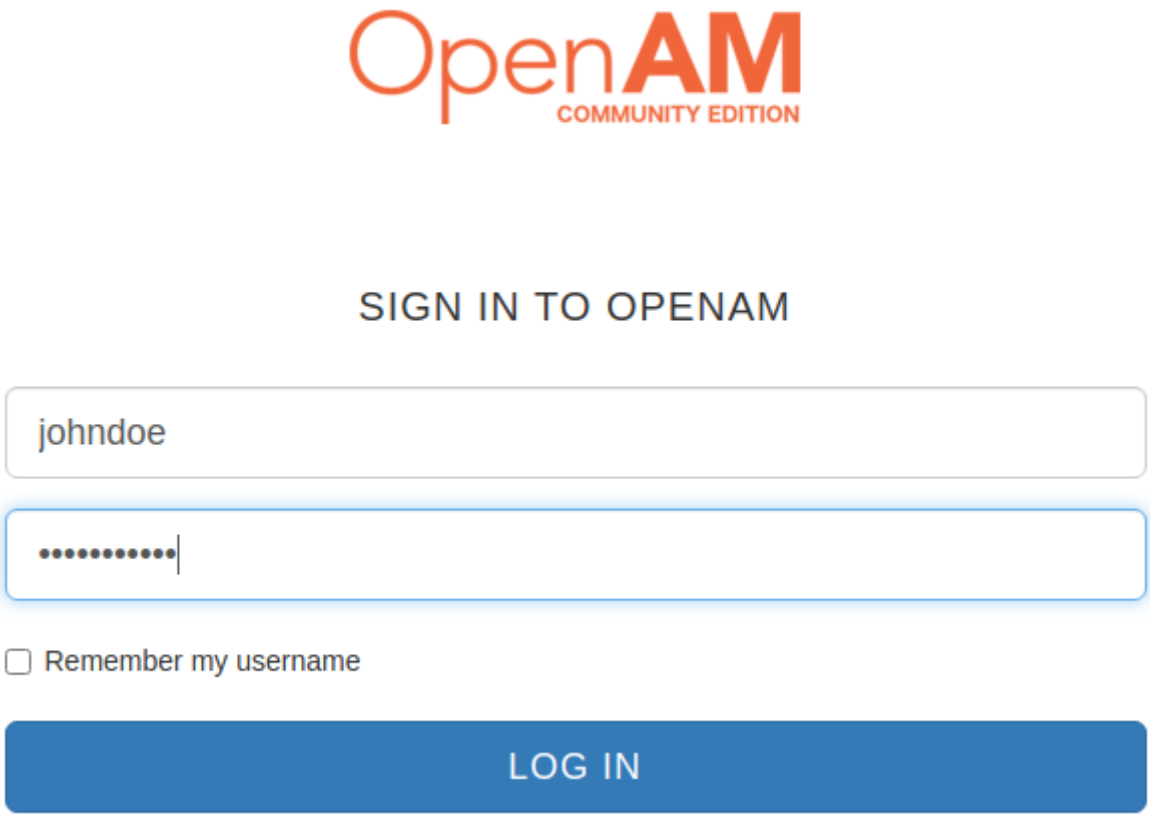
Let's run a demo Spring Boot application in a Docker container

```
docker run -h app.example.org -p 8081:8081 --network openam --name spring-security-
openam-example openidentityplatform/spring-security-openam-example
```

Exit the console and open the demo application link in your browser. http://app.example.org:8081/ Click the `OpenAM Cookie` link.



The filter will not find a valid session and will redirect to OpenAM authentication. Enter the user's credentials from Active Directory.



After successful authentication, the user will be redirected back to the application. The `openam.userId` attribute of the HttpServletRequest object will be set to the user ID from Active Directory.



Further reading

- OpenAM and Spring Boot 3 Integration via OIDC OAuth2 Protocol
- OpenAM and Spring Boot 3 Integration via SAMLv2 Protocol
- OpenAM and Spring Boot 3 Integration via OpenAM Cookie
- How to Add Authorization and Protect Your Application With OpenAM and OpenIG Stack

How to Setup OAuth2 OIDC Federation in OpenAM

Introduction

In the following manual, we will set up a federation between two OpenAM instances via OAuth2/OIDC protocol. One OpenAM instance we will set up as OAuth2/OIDC Server - Identity Provider (IdP), and another instance OAuth2/OIDC client - Service Provider (SP).Thus, you can authenticate to an OpenAM client instance (SP) using OpenAM instance credentials (IdP) using the OAuth2/OIDC protocol.

OpenAM Instances Installation

If you already have OpenAM instances installed, you can skip this section. For demonstration purposes, we will install OpenAM IdP and SP in Docker containers.

Network Setup

Add hostnames and IP adress to the `hosts` file.

```
127.0.0.1 idp.acme.org sp.mycompany.org
```

In Windows `hosts` file located at `C:\Windows\System32\drivers\etc\hosts` directory. Un Linux and Mac the file location is `/etc/hosts`

Create a Docker network for OpenAM instances

```
docker network create openam-oauth
```

OpenAM IdP Installation

Run OpenAM Docker Container

```
docker run -h idp.acme.org -p 8080:8080 --network openam-oauth --name openam-idp
openidentityplatform/openam
```

Once the OpenAM server is running, perform the initial configuration by running the following command and wait for the configuration to complete.

```
docker exec -w '/usr/openam/ssoconfiguratortools' openam-idp bash -c \
'echo "ACCEPT_LICENSES=true
SERVER_URL=http://idp.acme.org:8080
DEPLOYMENT_URI=/$OPENAM_PATH
```

```
BASE_DIR=$OPENAM_DATA_DIR
locale=en_US
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=passw0rd
AMLDAPUSERPASSWD=p@passw0rd
COOKIE_DOMAIN=idp.acme.org
ACCEPT_LICENSES=true
DATA_STORE=embedded
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=idp.acme.org
DIRECTORY_PORT=50389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1689
ROOT_SUFFIX=dc=openam,dc=example,dc=org
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=passw0rd" > conf.file && java -jar openam-configurator-tool*.jar
--file conf.file'
```

OpenAM SP Installation

Run OpenAM Docker Container

```
docker run -h sp.mycompany.org -p 8081:8080  --network openam-oauth --name openam-sp
openidentityplatform/openam
```

Once the OpenAM server is running, perform the initial configuration by running the following command and wait for the configuration to complete.

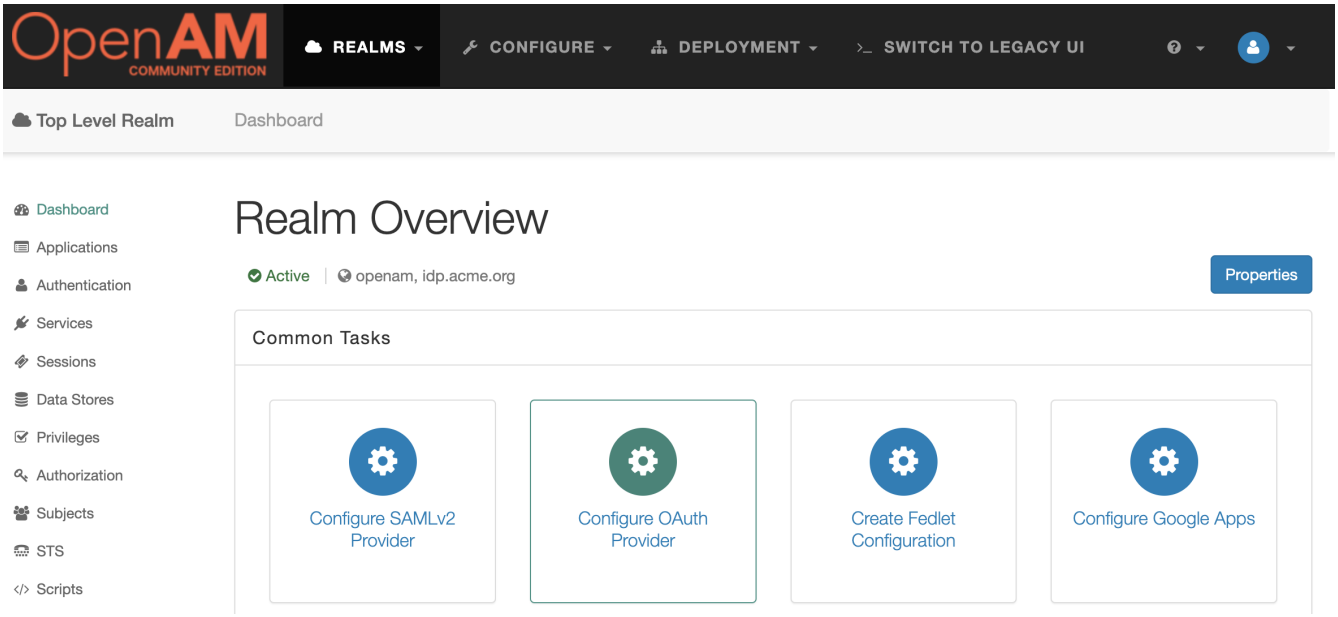
```
docker exec -w '/usr/openam/ssoconfiguratortools' openam-sp bash -c \
'echo "ACCEPT_LICENSES=true
SERVER_URL=http://sp.mycompany.org:8080
DEPLOYMENT_URI=/$OPENAM_PATH
BASE_DIR=$OPENAM_DATA_DIR
locale=en_US
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=passw0rd
AMLDAPUSERPASSWD=p@passw0rd
COOKIE_DOMAIN=sp.mycompany.org
ACCEPT_LICENSES=true
DATA_STORE=embedded
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=sp.mycompany.org
DIRECTORY_PORT=50389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1689
ROOT_SUFFIX=dc=openam,dc=example,dc=org
```

```
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=passw0rd" > conf.file && java -jar openam-configurator-tool*.jar
--file conf.file'
```

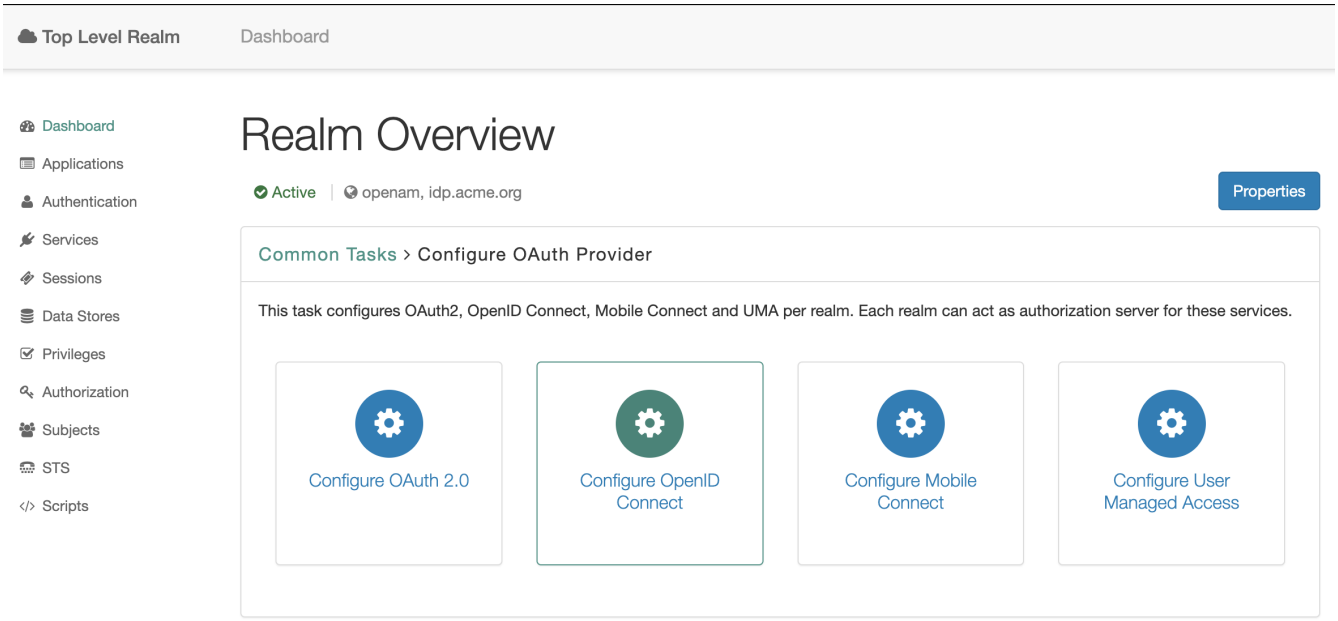
OAuth2/OIDC Server Setup

Open the OpenAM console, which will be in the role of the OAuth2/OIDC server at http://idp.acme.org:8080/openam. In the login field enter the `amadmin` value. In the password field enter the value from `ADMIN_PWD` option, in this case `passw0rd`.

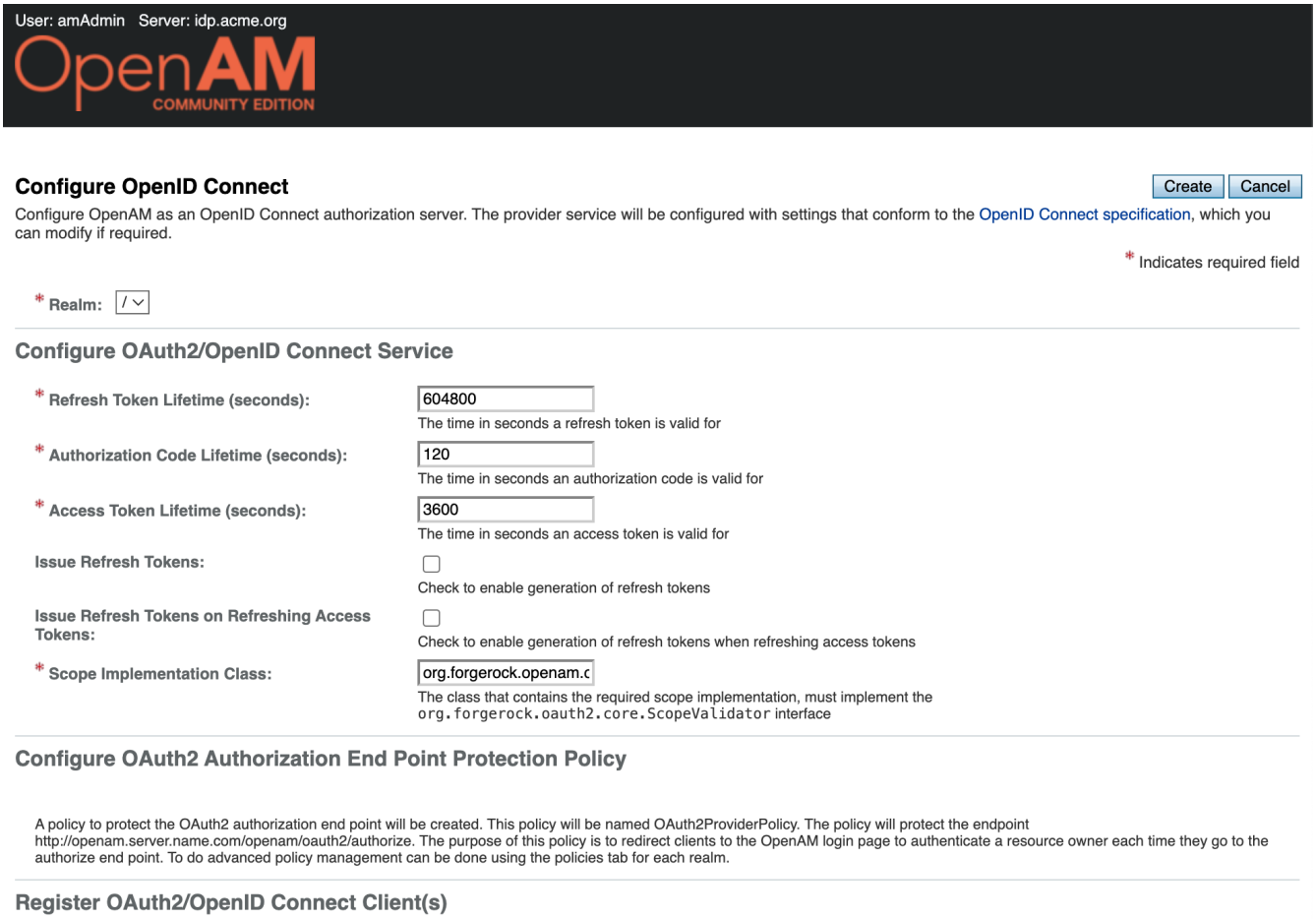
Go to the root realm and in the Dashboard select `Configure OAuth Provider`.



Next, `Configure OpenID Connect`.

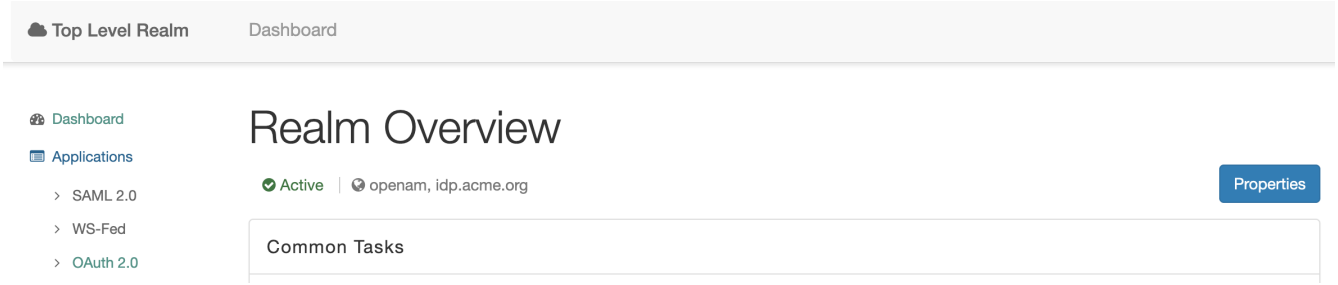


Leave the settings unchanged and click the `Create` button.

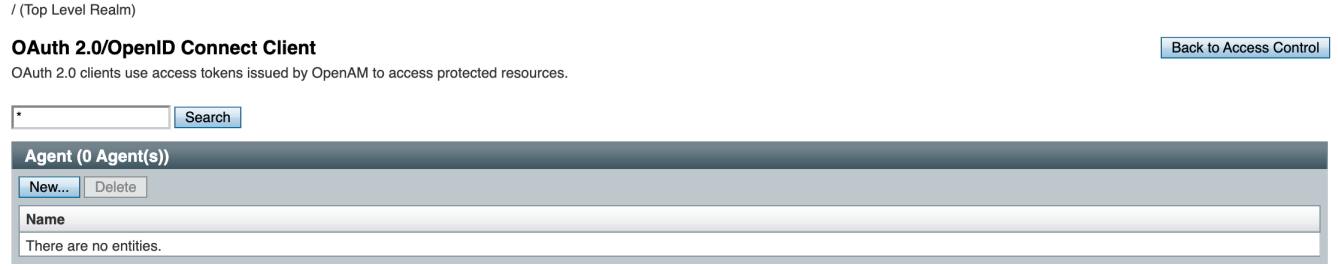


Create a Client Application

Open the OAuth2/OIDC server admin console, navigate to the desired realm and select `Applications` → `OAuth 2.0` from the left menu

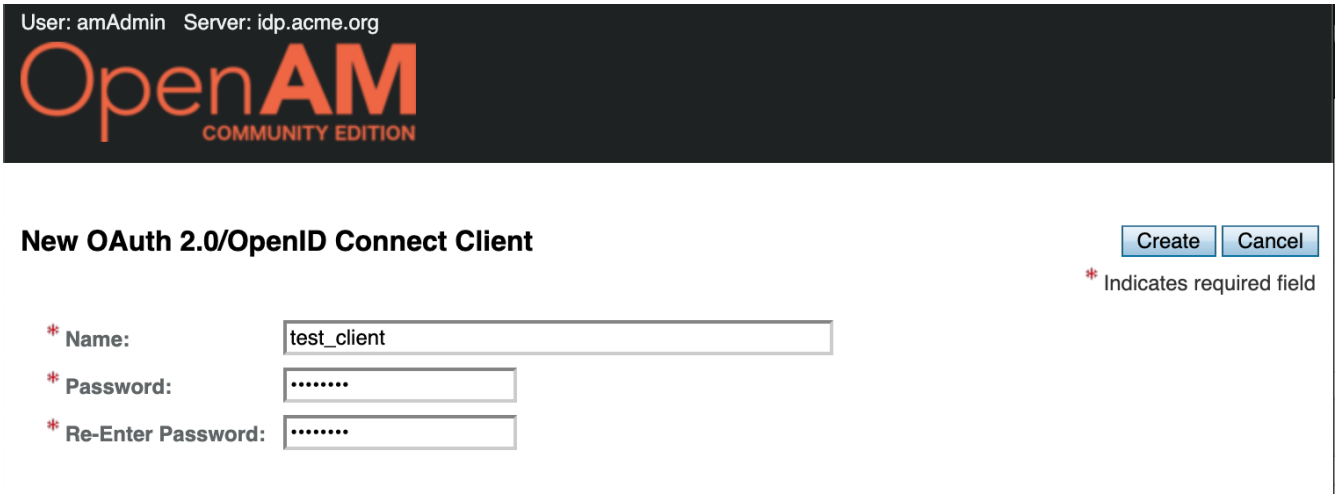


In the list, click the `New` button.



Fill the `Name` (client_id) and `Password` (client_secret) fields. Repeat the password and press the `Create`

button.



Open the created application and fill the settings.

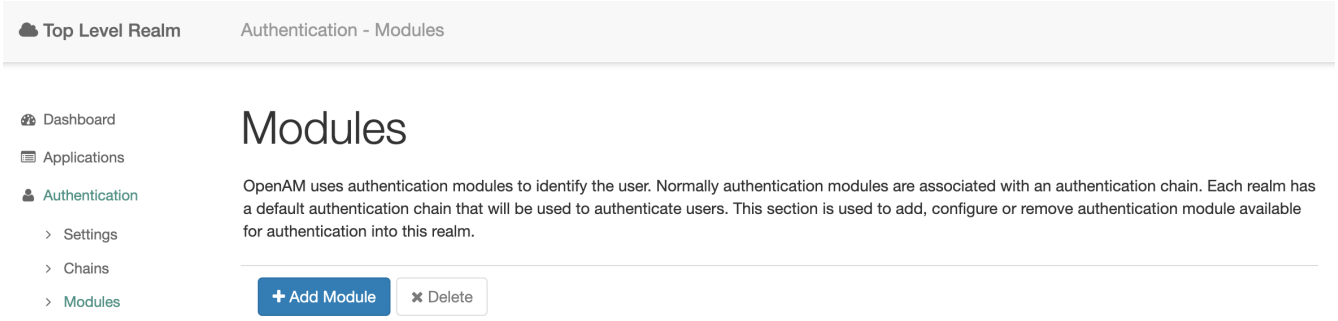
Setting	Value
Redirection URIs	http://sp.mycompany.org:8081/openam/oauth2c/OAuthProxy.jsp
Scope	openid
Token Endpoint Authentication Method	client_secret_post
ID Token Signing Algorithm	RS256

OAuth2/OIDC Client Setup

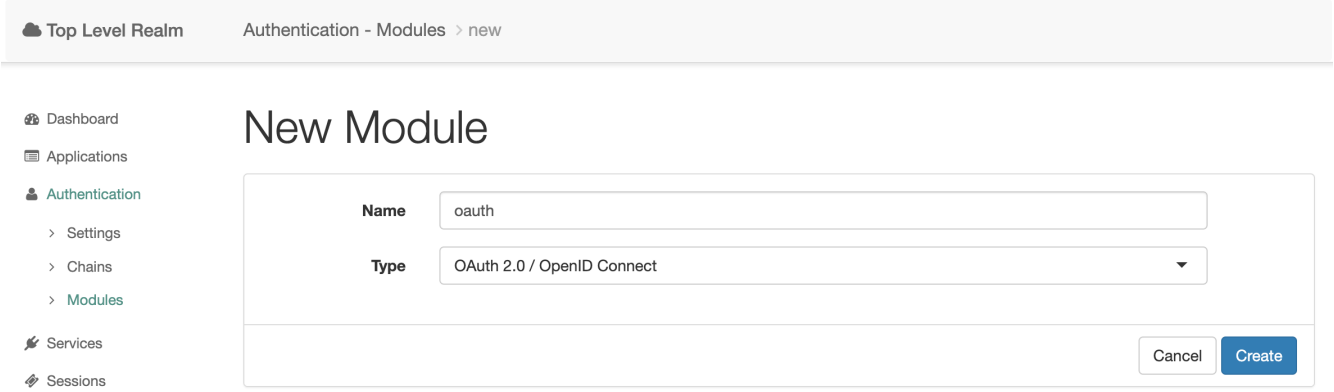
Create OAuth2/OIDC Authentication Module

Open the console of OpenAM, which will be in the role of OAuth2/OIDC client at http://openam-sp.example.org:8081/openam. In the login field enter the value `amadmin`, in the password field enter the value specified in the `ADMIN_PWD` setting, in this case `passw0rd`.

Open the realm and select `Authentication` → `Modules` from the left menu. Click the `Add Module` button.



Select the module type `OAuth2/OpenID Connect`, the module name can be anything, the path will be `oauth`.



Press the `Create` button.

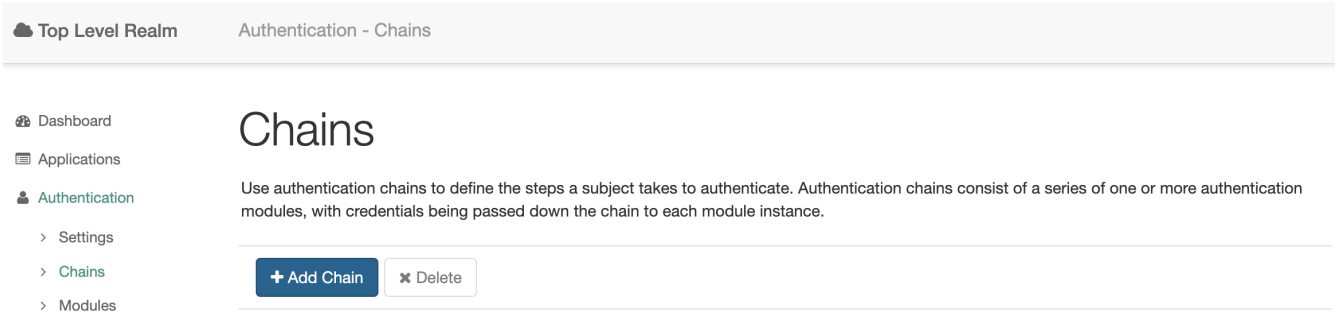
In the list, open the settings of the created module and fill in the settings:

Setting	Value
Client Id	test_client
Client Secret	Password specified when registering the application
Authentication Endpoint URL	http://idp.acme.org:8080/openam/oauth2/authorize
Access Token Endpoint URL	http://idp.acme.org:8080/openam/oauth2/access_token
User Profile Service URL	http://idp.acme.org:8080/openam/oauth2/tokeninfo
Scope	openid
OAuth2 Access Token Profile Service Parameter name	access_token
Proxy URL	http://sp.mycompany.org:8081/openam/oauth2c/OAuthProxy.jsp
Account Mapper	org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper
Account Mapper Configuration	sub=uid
Attribute Mapper	org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper
Attribute Mapper Configuration	sub=uid
Create account if it does not exist	disabled
Prompt for password setting and activation code	disabled
Map to anonymous user	disabled
OpenID Connect validation configuration type	.well-known/openid-configuration_url
OpenID Connect validation configuration value	http://idp.acme.org:8080/openam/oauth2/.well-known/openid-configuration

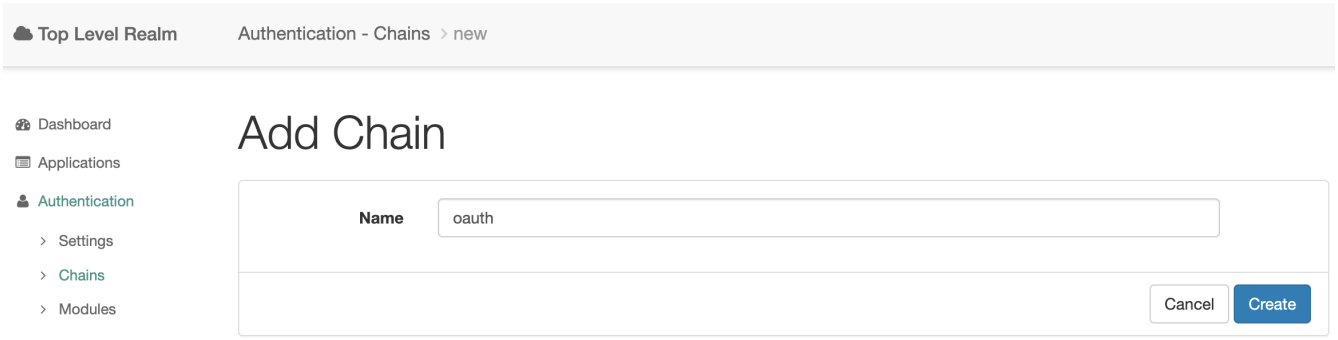
Setting	Value
Name of OpenID Connect ID Token Issuer	http://idp.acme.org:8080/openam/oauth2

Setup OAuth2/OIDC Authentication Chain

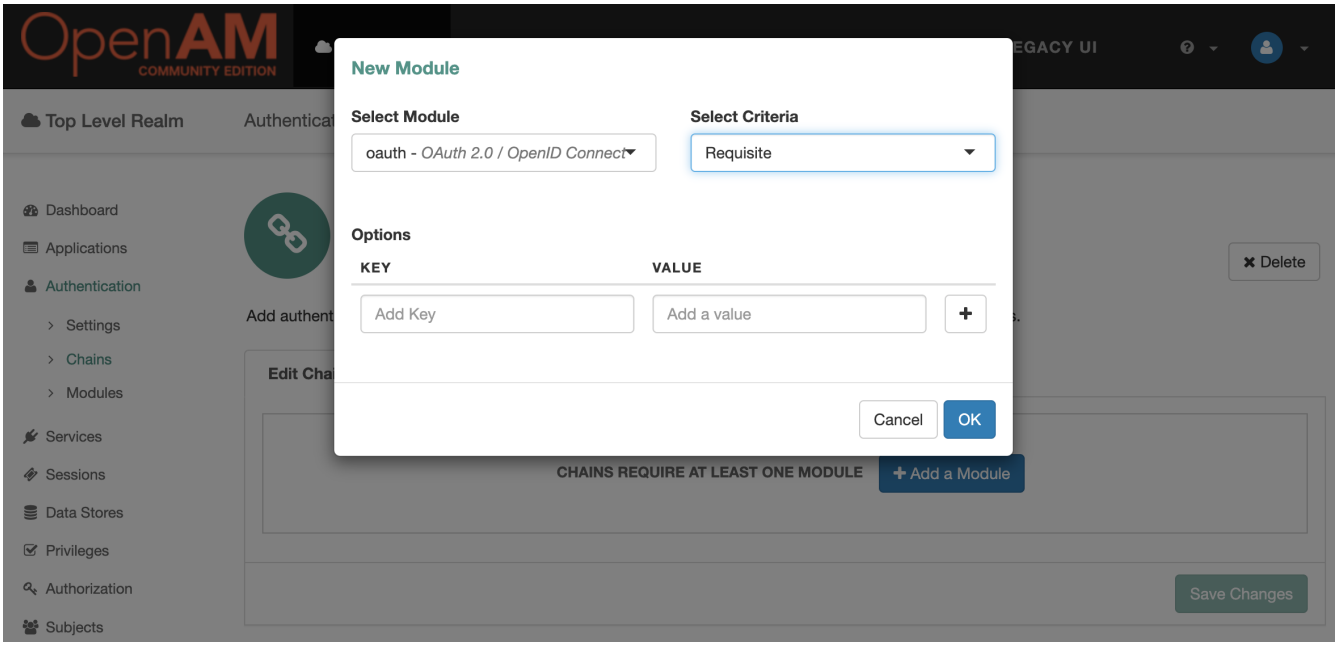
Open the OpenAM Service Provider administrator console. Select the realm and in the left menu go to `Authentication` → `Chains`.



Create a new authentication chain

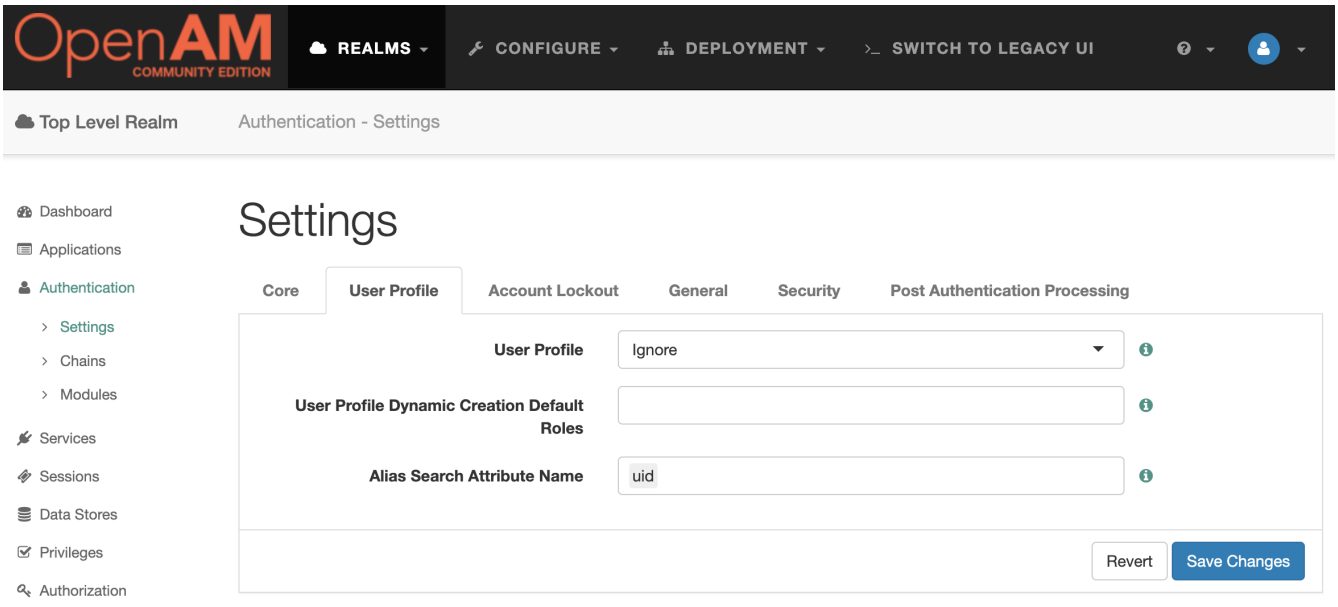


Click the `Add a Module` button and add the `oauth` module. Set the Criteria to `Requisite`. Click `OK` and then `Save Changes`.



Realm Setup

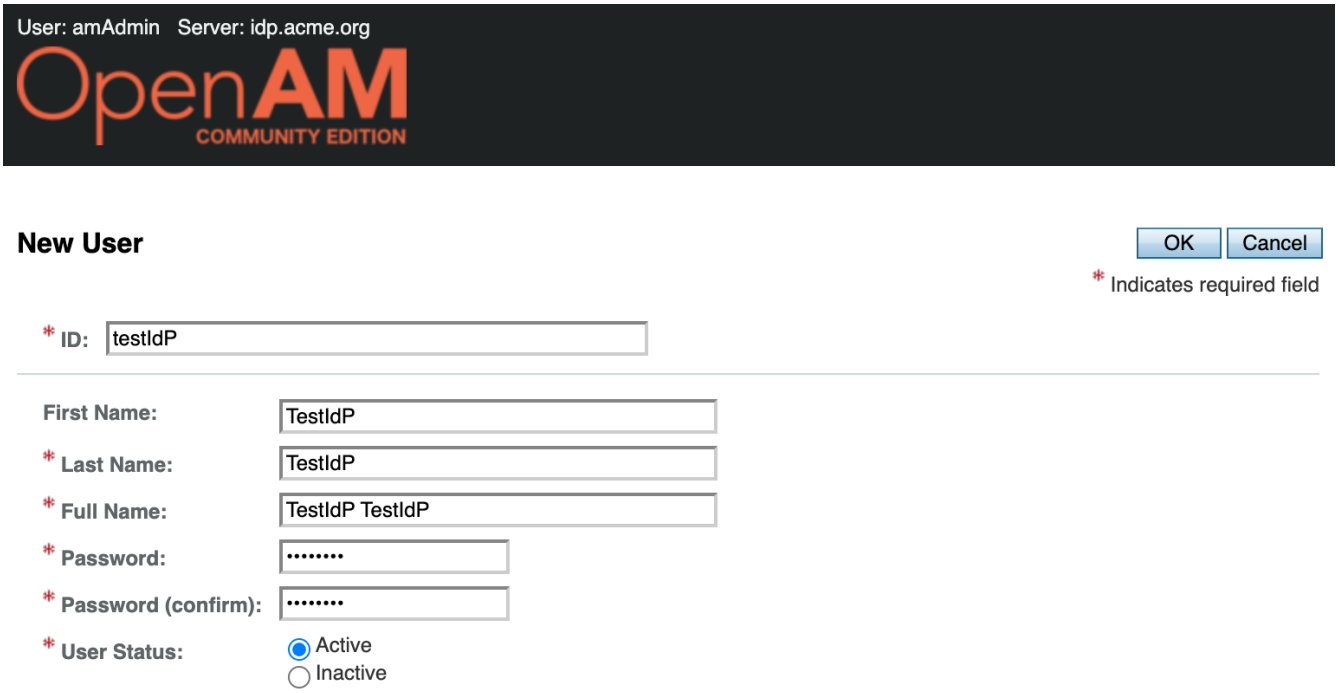
Go to the OpenAM SP administrator console. In the left menu, go to `Authentication` → `Settings`. On the `User Profile` tab, select `Ignore`. Save the changes.



Test the Solution

Go to the OpenAM OAuth2/OIDC Server admin console, select realm, under Dashboard in the left menu select `Subjects`.

This will open a list of users. Create a new `testIdp` account



Log out of the OpenAM OAuth2/OIDC Server administration console and the OpenAM OAuth2/OIDC Client administration console or open a browser in incognito mode.

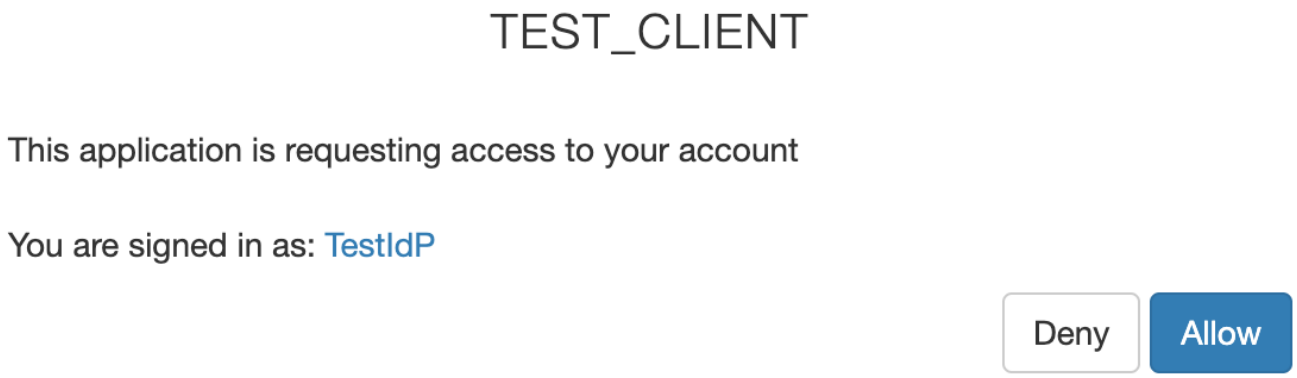
Open the OAuth2/OIDC Client authentication URL of the `oauth` chain

http://sp.mycompany.org:8081/openam/XUI/?service=oauth

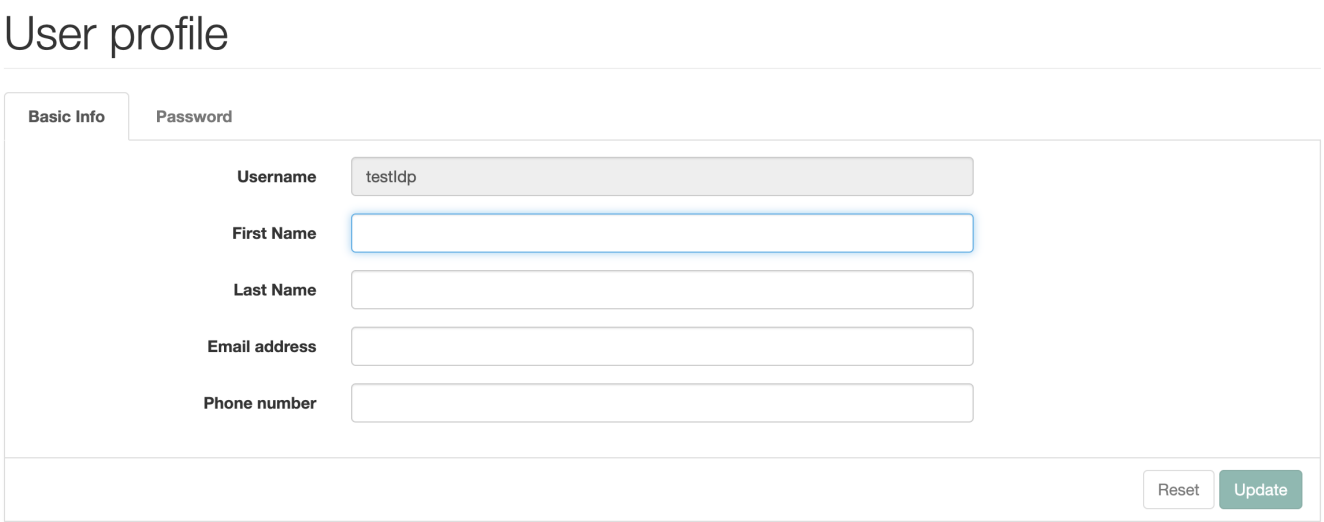
You will be redirected to authenticate to the OAuth2/OIDC Server. Enter the users testIdP credentials.

Confirm consent to access the `test_client` application user data





After accepting the consent, you well be redirected to OpenAM OAuth2/OIDC Client console with the OpenAM OAuth2/OIDC Server user credentials.



How to Setup SAMLv2 Federation in OpenAM

Introduction

In the following manual, we will set up a federation between two OpenAM instances. One instance will act as an Identity Provider (IdP), and another as a Service Provider (SP). Therefore, you will be able to authenticate in OpenAM SP with OpenAM IdP credentials.

OpenAM Instances Installation

If you already have OpenAM instances installed, you can skip this section. For demonstration purposes, we will install OpenAM IdP and SP in Docker containers.

Network Setup

Add hostnames and IP adress to the `hosts` file.

```
127.0.0.1 idp.acme.org sp.mycompany.org
```

In Windows `hosts` file located at `C:\Windows\System32\drivers\etc\hosts` directory. Un Linux and Mac the file location is `/etc/hosts`

Create a Docker network for OpenAM instances

```
docker network create openam-saml
```

OpenAM IdP Installation

Run OpenAM Docker Container

```
docker run -h idp.acme.org -p 8080:8080 --network openam-saml --name openam-idp
openidentityplatform/openam
```

Once the OpenAM server is running, perform the initial configuration by running the following command and wait for the configuration to complete.

```
docker exec -w '/usr/openam/ssoconfiguratortools' openam-idp bash -c \
'echo "ACCEPT_LICENSES=true
SERVER_URL=http://idp.acme.org:8080
DEPLOYMENT_URI=/$OPENAM_PATH
BASE_DIR=$OPENAM_DATA_DIR
locale=en_US
```

```
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=passw0rd
AMLDAPUSERPASSWD=p@passw0rd
COOKIE_DOMAIN=idp.acme.org
ACCEPT_LICENSES=true
DATA_STORE=embedded
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=idp.acme.org
DIRECTORY_PORT=50389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1689
ROOT_SUFFIX=dc=openam,dc=example,dc=org
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=passw0rd" > conf.file && java -jar openam-configurator-tool*.jar
--file conf.file'
```

OpenAM SP Installation

Run OpenAM Docker Container

```
docker run -h sp.mycompany.org -p 8081:8080  --network openam-saml --name openam-sp
openidentityplatform/openam
```

Once the OpenAM server is running, perform the initial configuration by running the following command and wait for the configuration to complete.

```
docker exec -w '/usr/openam/ssoconfiguratortools' openam-sp bash -c \
'echo "ACCEPT_LICENSES=true
SERVER_URL=http://sp.mycompany.org:8080
DEPLOYMENT_URI=/$OPENAM_PATH
BASE_DIR=$OPENAM_DATA_DIR
locale=en_US
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=passw0rd
AMLDAPUSERPASSWD=p@passw0rd
COOKIE_DOMAIN=sp.mycompany.org
ACCEPT_LICENSES=true
DATA_STORE=embedded
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=sp.mycompany.org
DIRECTORY_PORT=50389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1689
ROOT_SUFFIX=dc=openam,dc=example,dc=org
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=passw0rd" > conf.file && java -jar openam-configurator-tool*.jar
```

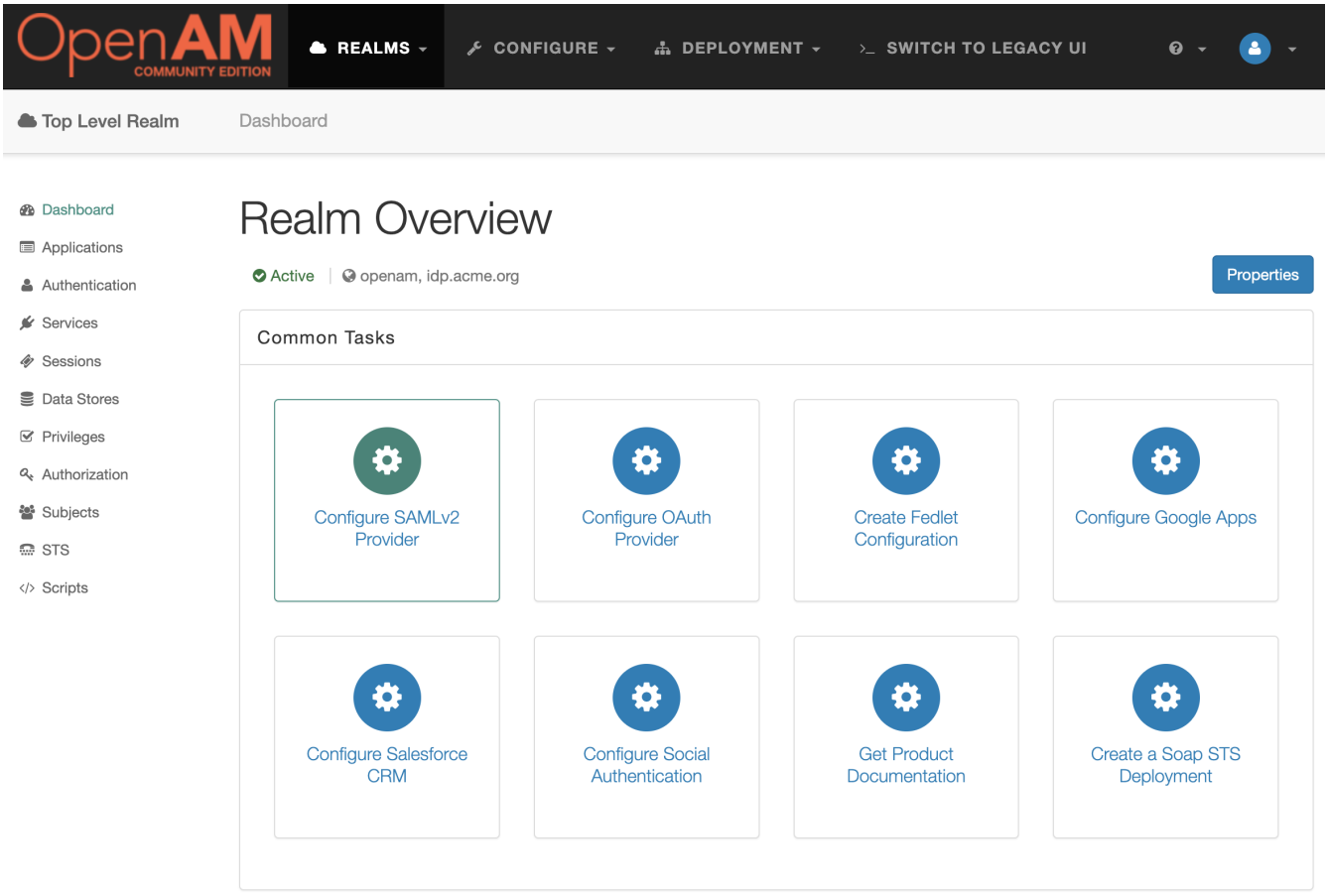
```
--file conf.file'
```

Identity Provider и Service Provider Configuration

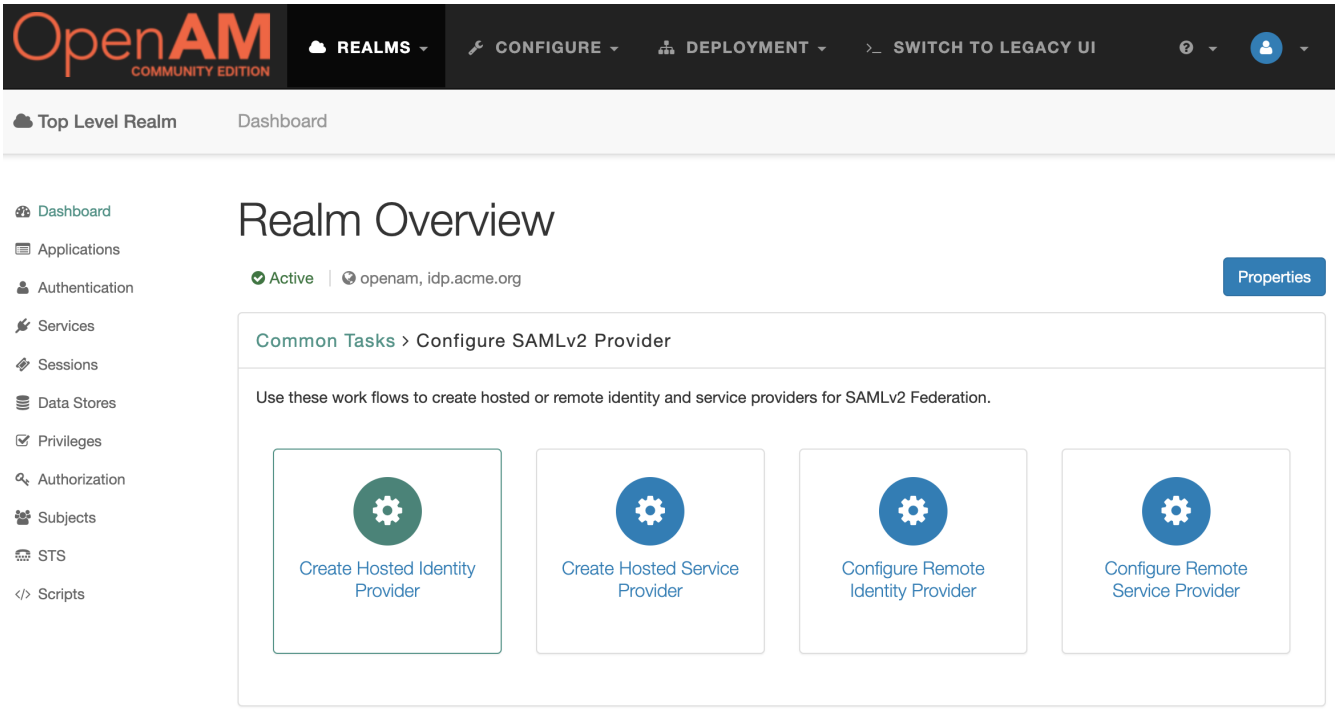
Hosted Identity Provider Configuration

Open the OpenAM Identity provider administartion console at http://idp.acme.org:8080/openam. In the login field enter the `amadmin` value. In the password field enter the value from `ADMIN_PWD` option, in this case `passw0rd`.

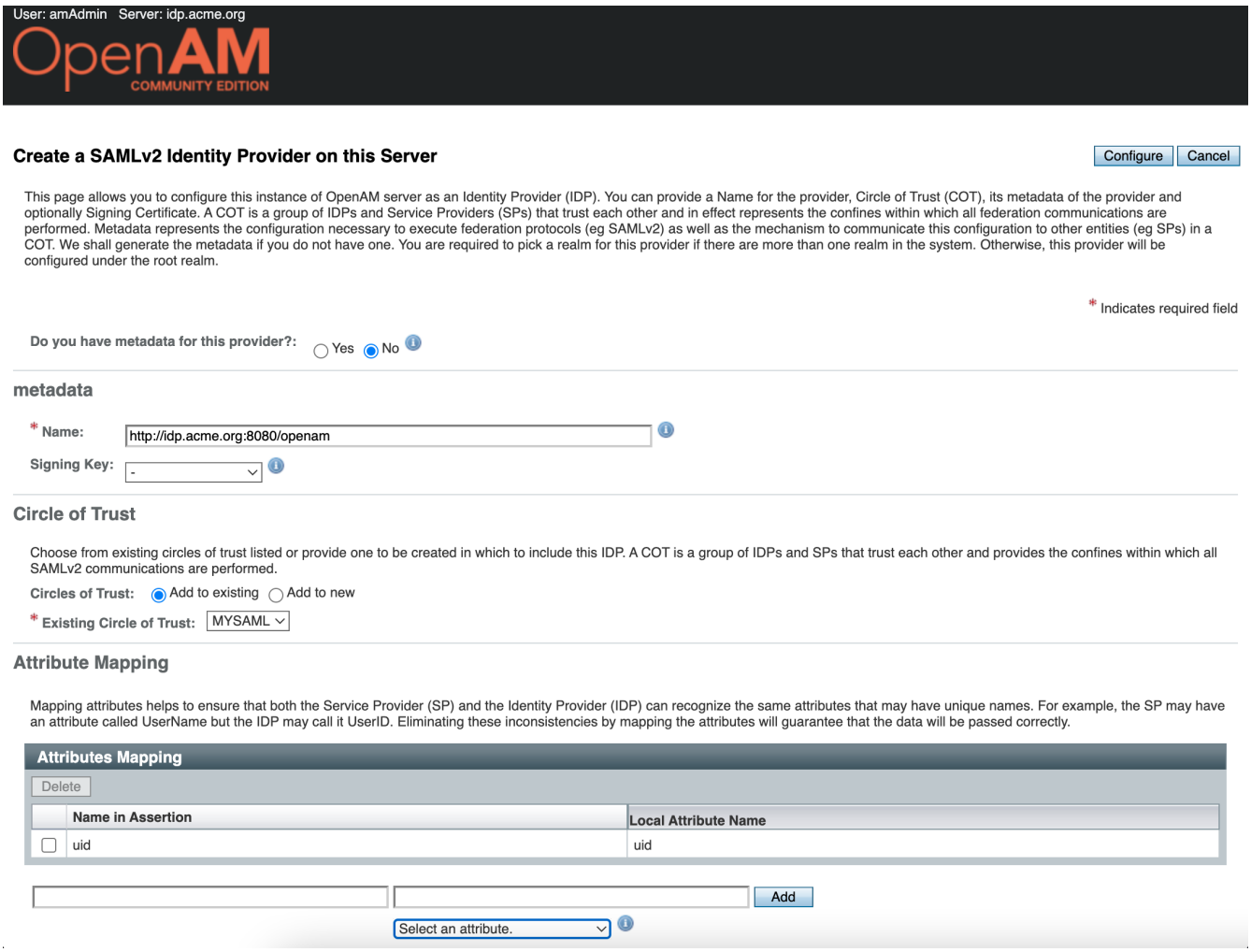
Go to the root realm and in the Dashboard select `Configure SAMLv2 Provider`.



Then `Create Hosted Identity Provider`



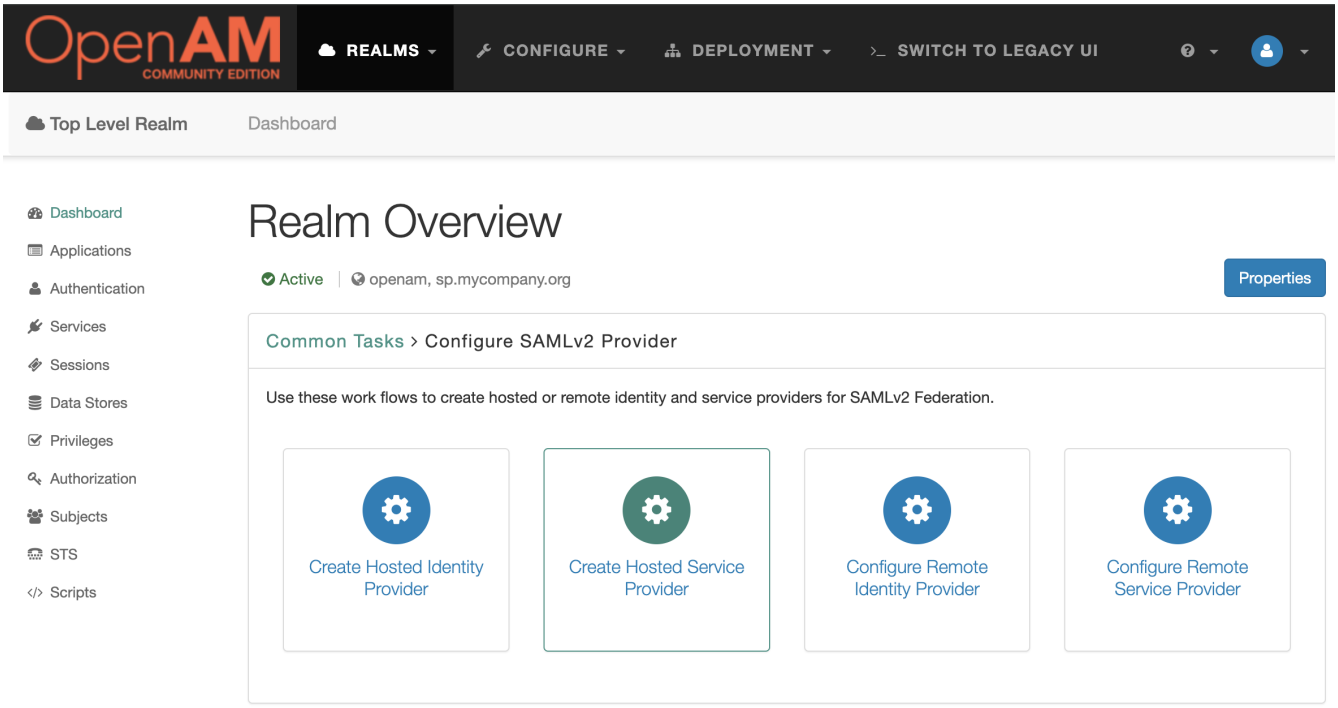
In the `Signing Key` setting for the demonstration purposes select the `test` value, then enter any `Circle of Trust` value. Then add `uid` in the `Attribute Mapping` setting.



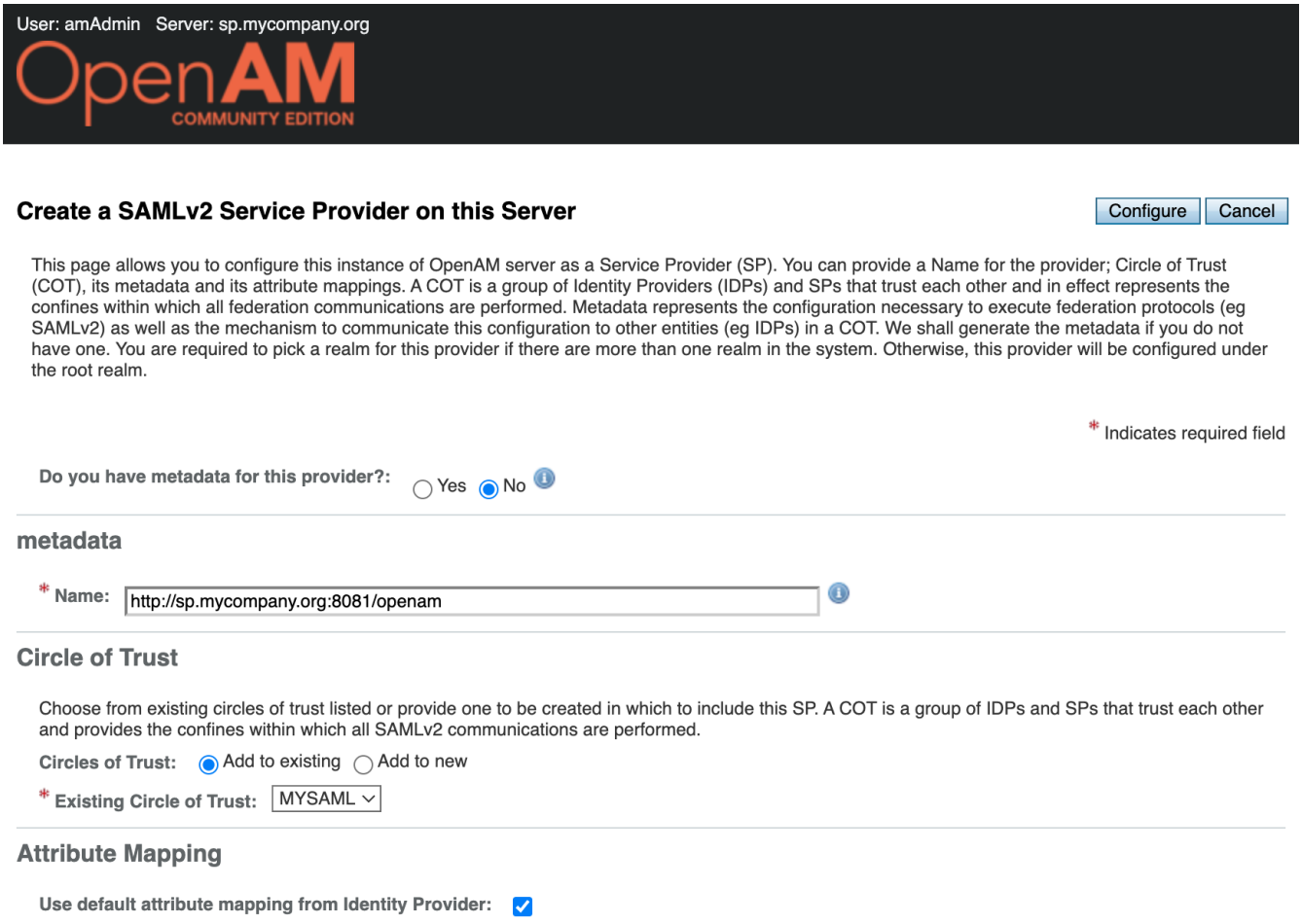
Press the `Configure` button. Next, OpenAM will prompt you to configure the Remote Service Provider. Since we do not have it configured yet, click the `Finish` button

Hosted Service Provider Configuration

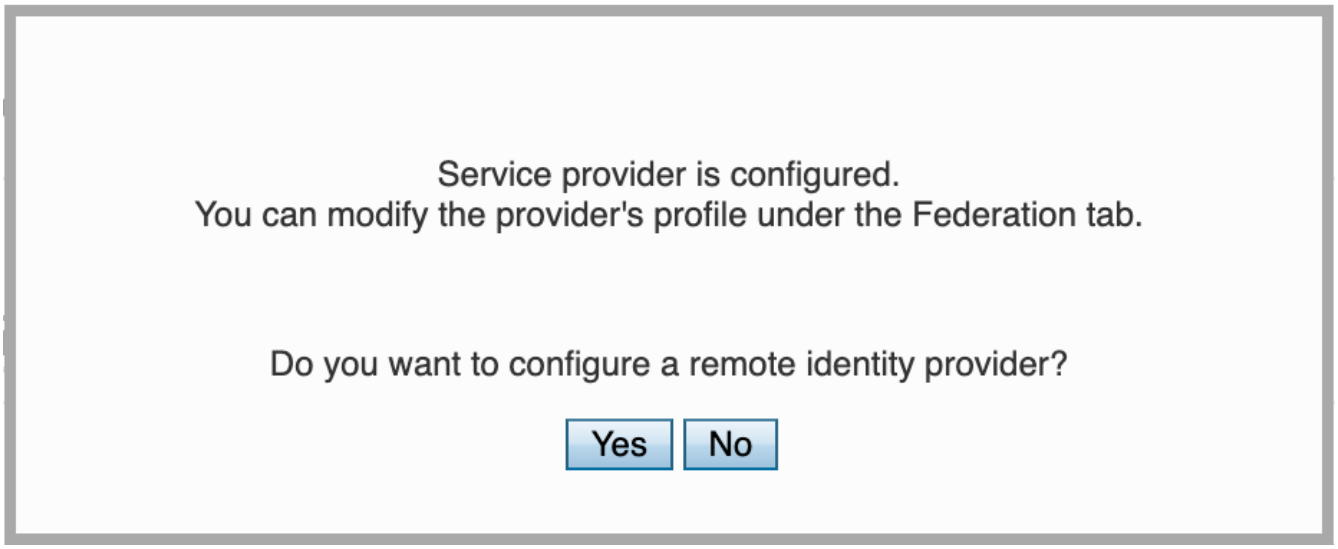
Open the OpenAM Service Provider administarion console at http://sp.mycompany.org:8081/openam. In the login field enter the `amadmin` value. In the password field enter the value from `ADMIN_PWD` option, in this case `passw0rd`. Go to the root realm and in the Dashboard select `Configure SAMLv2 Provider`. Next `Create Hosted Service Provider`.



Enter the circle of trust name, you can leave the other settings unchanged.



Press the `Configure` button.

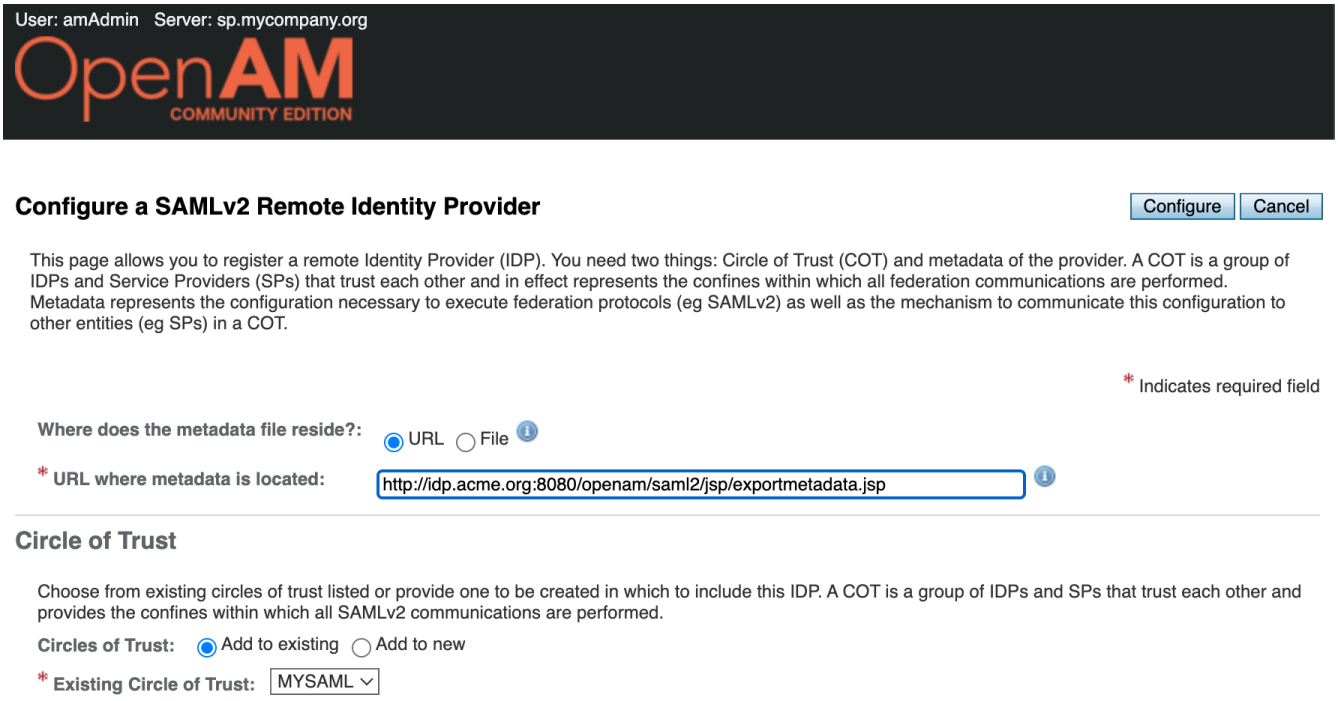


You will be prompted to configure the remote identity provider. Since we have already configured the Identity Provider in the previous step, you can click `Yes`. This will open the remote identity provider configuration window.

Remote Identity Provider Setup

Select the identity provider metadata location - URL. Enter the identity provider metadata URL in the field.

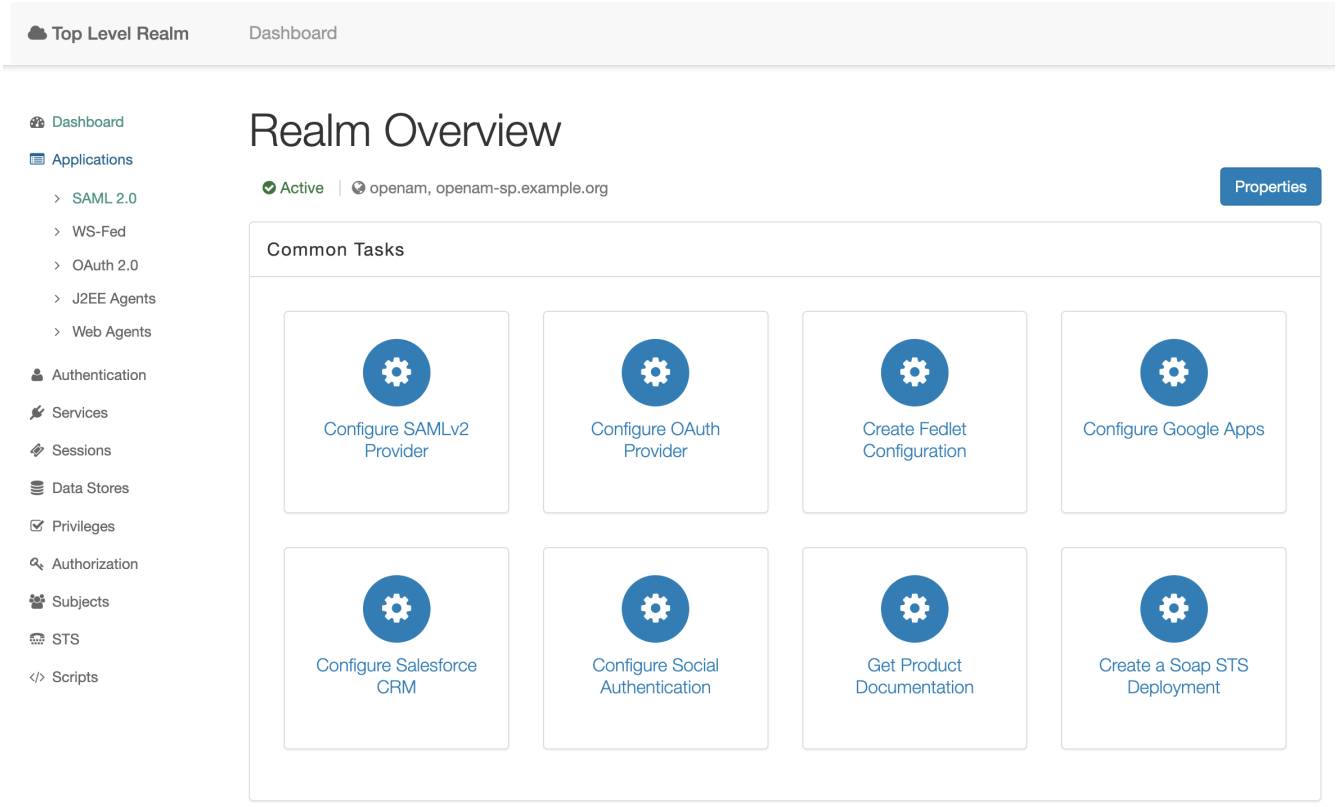
http://idp.acme.org:8080/openam/saml2/jsp/exportmetadata.jsp



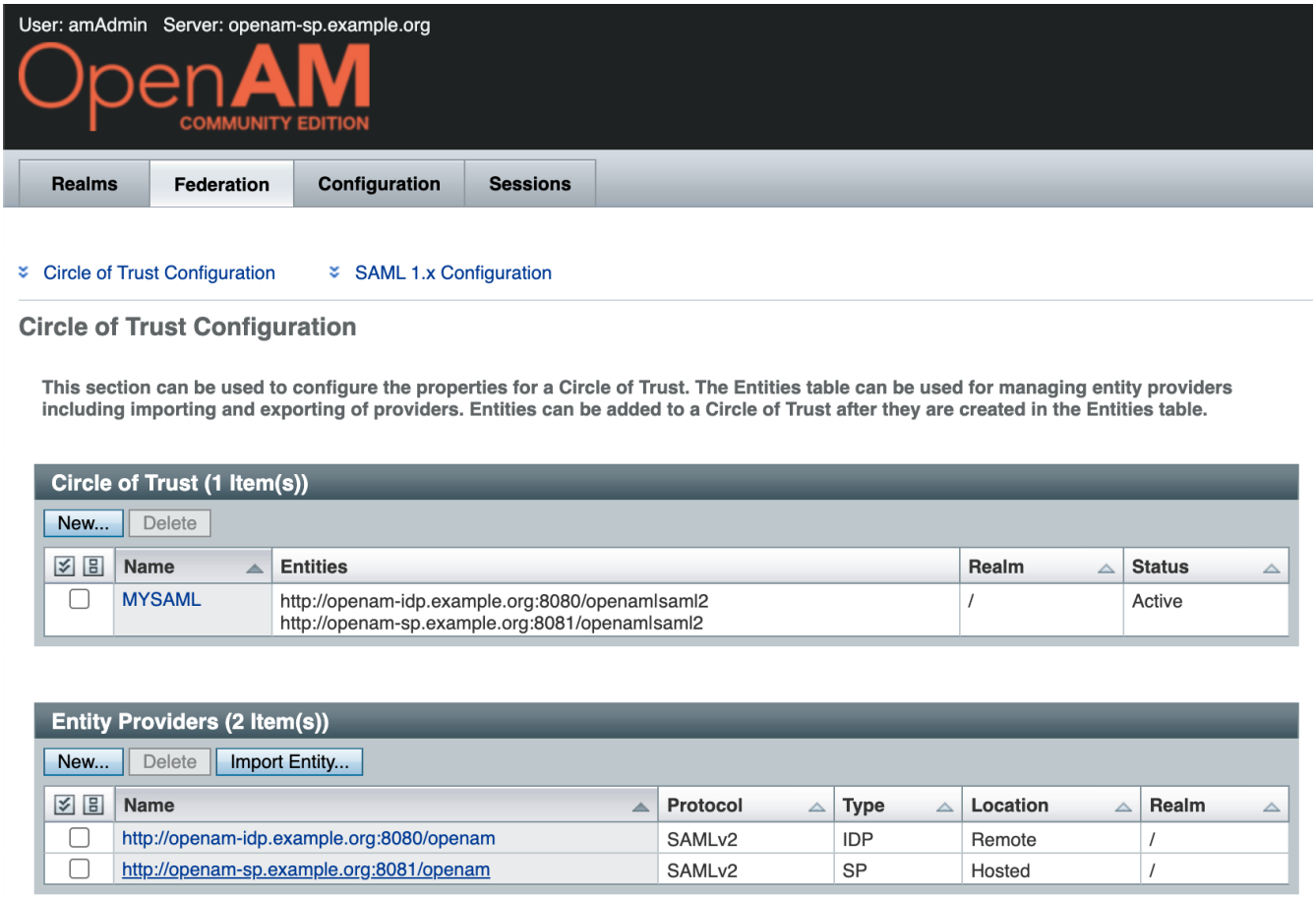
Click the `Configure` button.

User Mapping Configuration

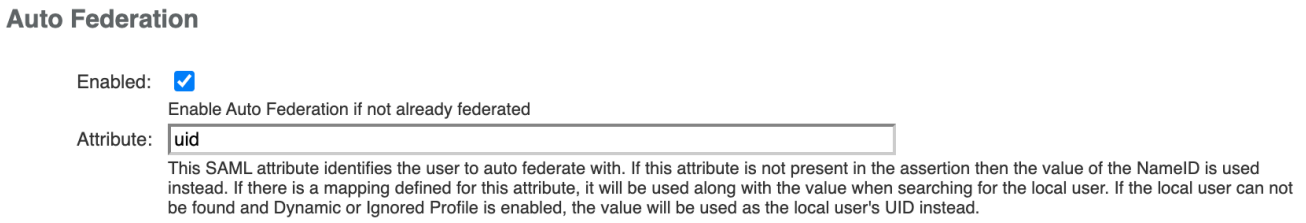
Open the OpenAM SP admin console. In the Dashboard in the left menu go to `Applications` → `SAML 2.0`



This will open the SAML federation configuration window. Navigate to `Entity Providers` → `http://sp.mycompany.org:8081/openam`



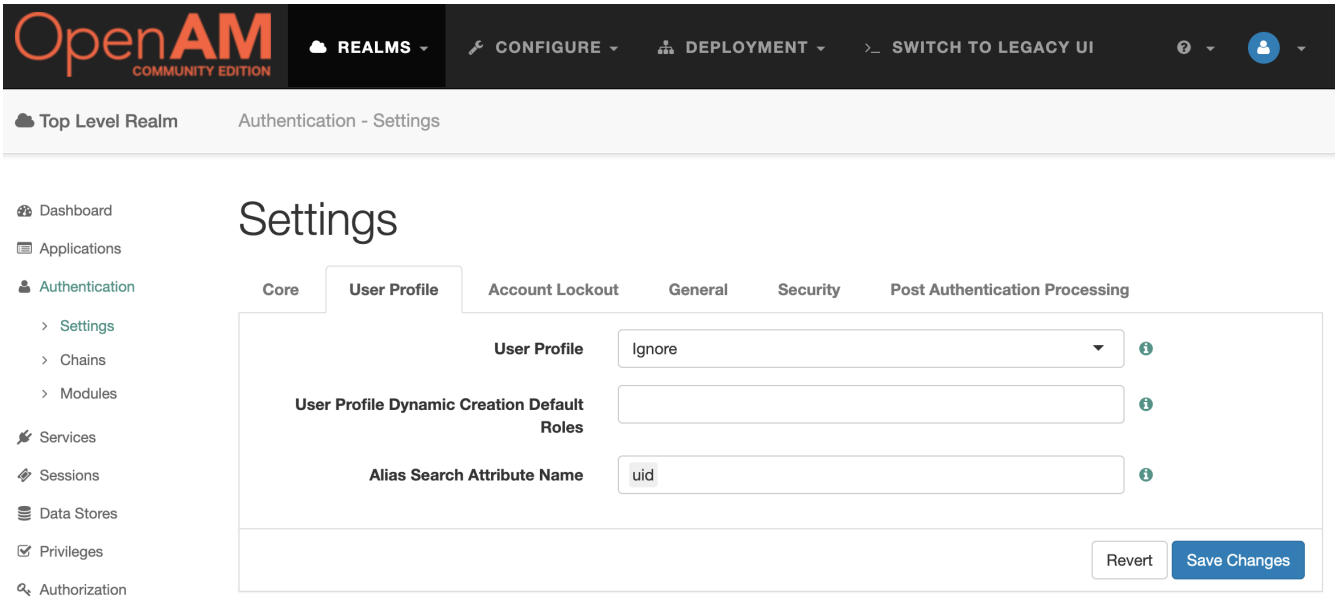
Next, click the `Assertions Processing` tab. Enable automatic federation by the `uid` attribute.



Press the `Save` button.

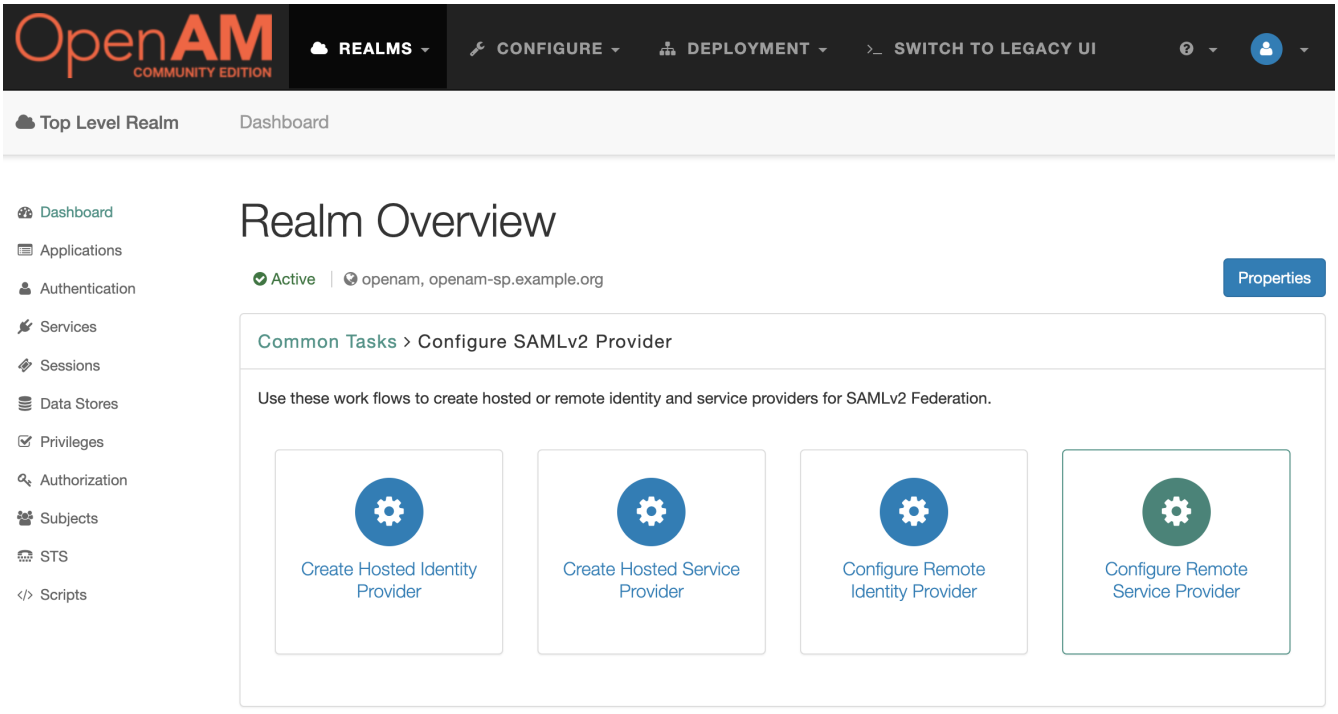
Configure Realm for Service Provider

Go to the OpenAM SP administrator console. In the left menu, go to `Authentication` → `Settings` . On the `User Profile` tab, select `Ignore` . Save the changes.

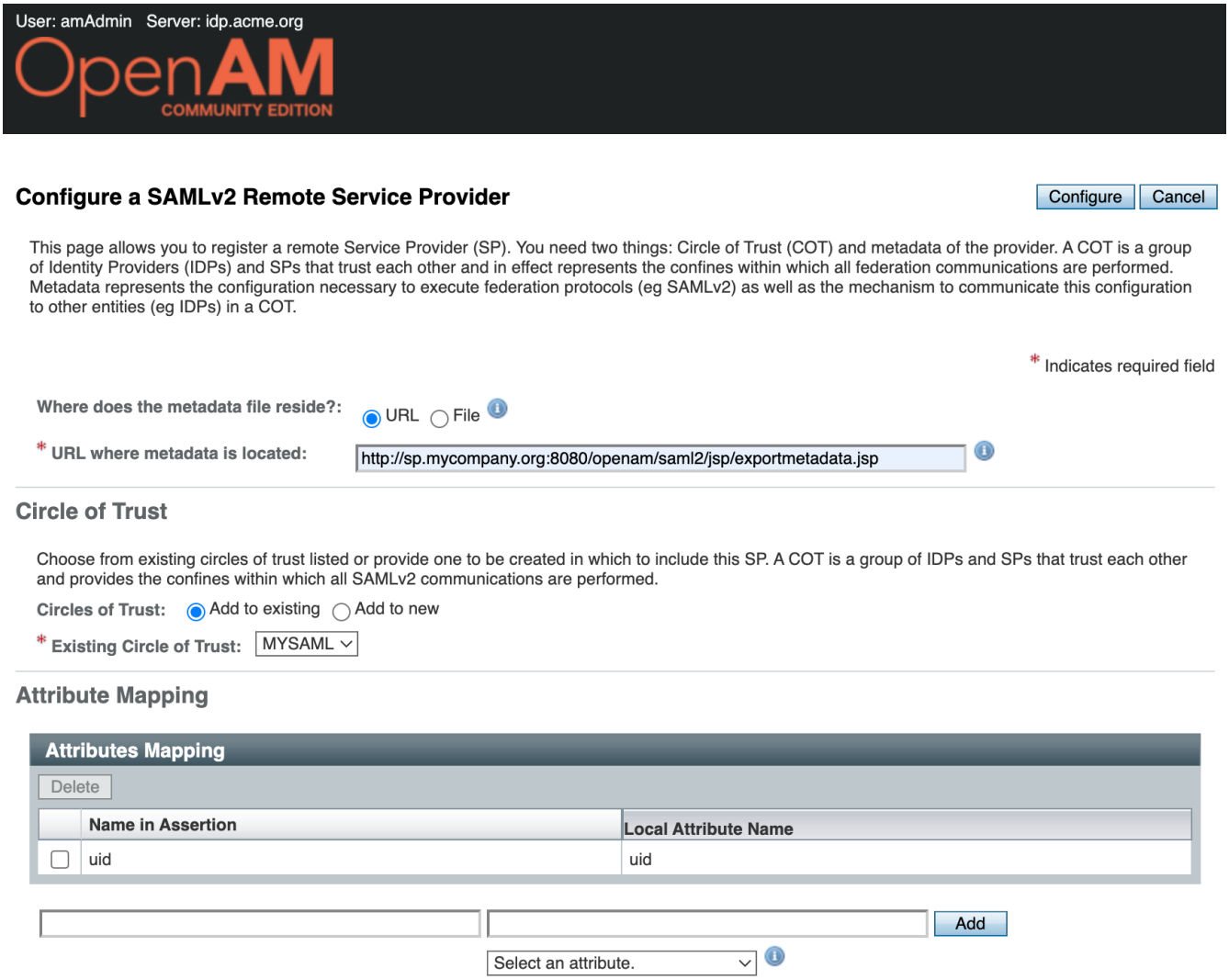


Remote Service Provider Setup

Go to OpenAM IdP administration console at `http://openam-idp.example.org:8080/openam`. Open the root realm and in the Dashboard select `Configure SAMLv2 Provider`, then `Configure Remote Service Provider`.



Add the service provider metadata URL http://sp.mycompany.org:8080/openam/saml2/jsp/exportmetadata.jsp. Note that the OpenAM Service Provider port is `8080` since the OpenAM instances are on the same network OpenAM IdP connects to the SP container on port `8080`.

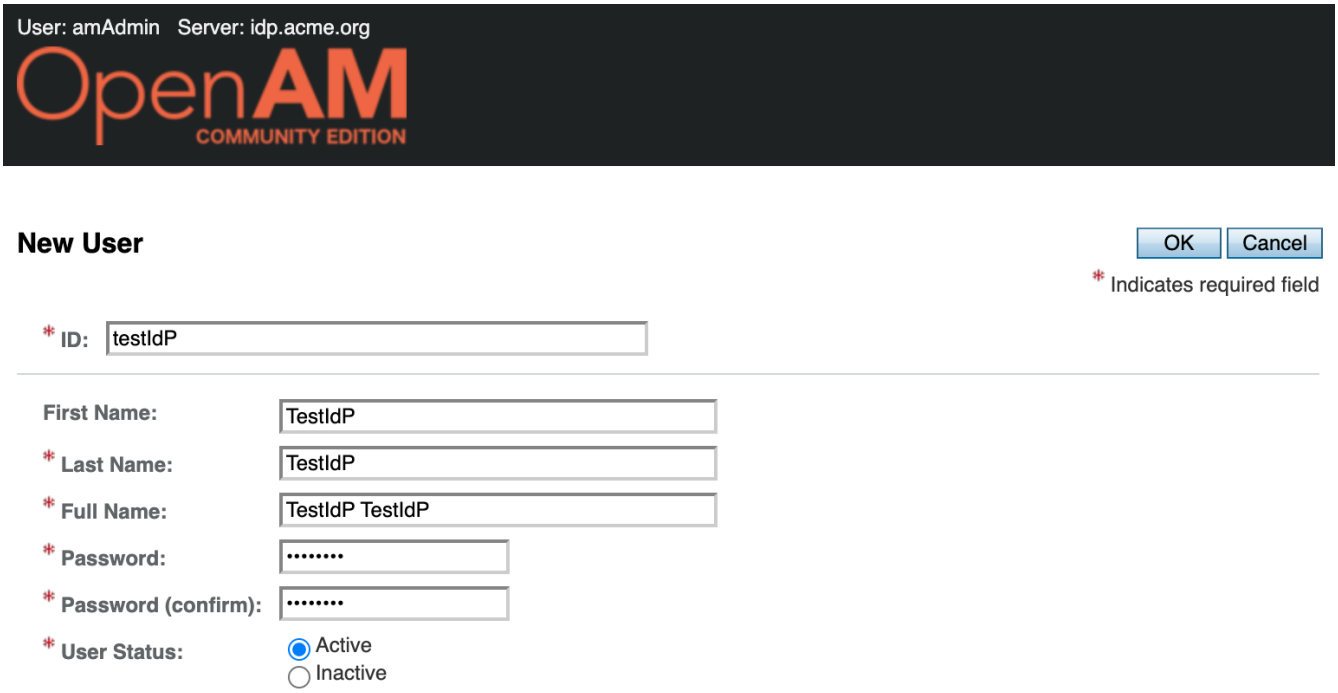


Click the `Configure` button. A message indicating that the remote service provider has been successfully created will be displayed.

Test Account Creation

Go to the OpenAM IdP admin console, select realm, in the Dashboard in the left hand menu select `Subjects`.

A list of users will open. Create a new `testIdp` account

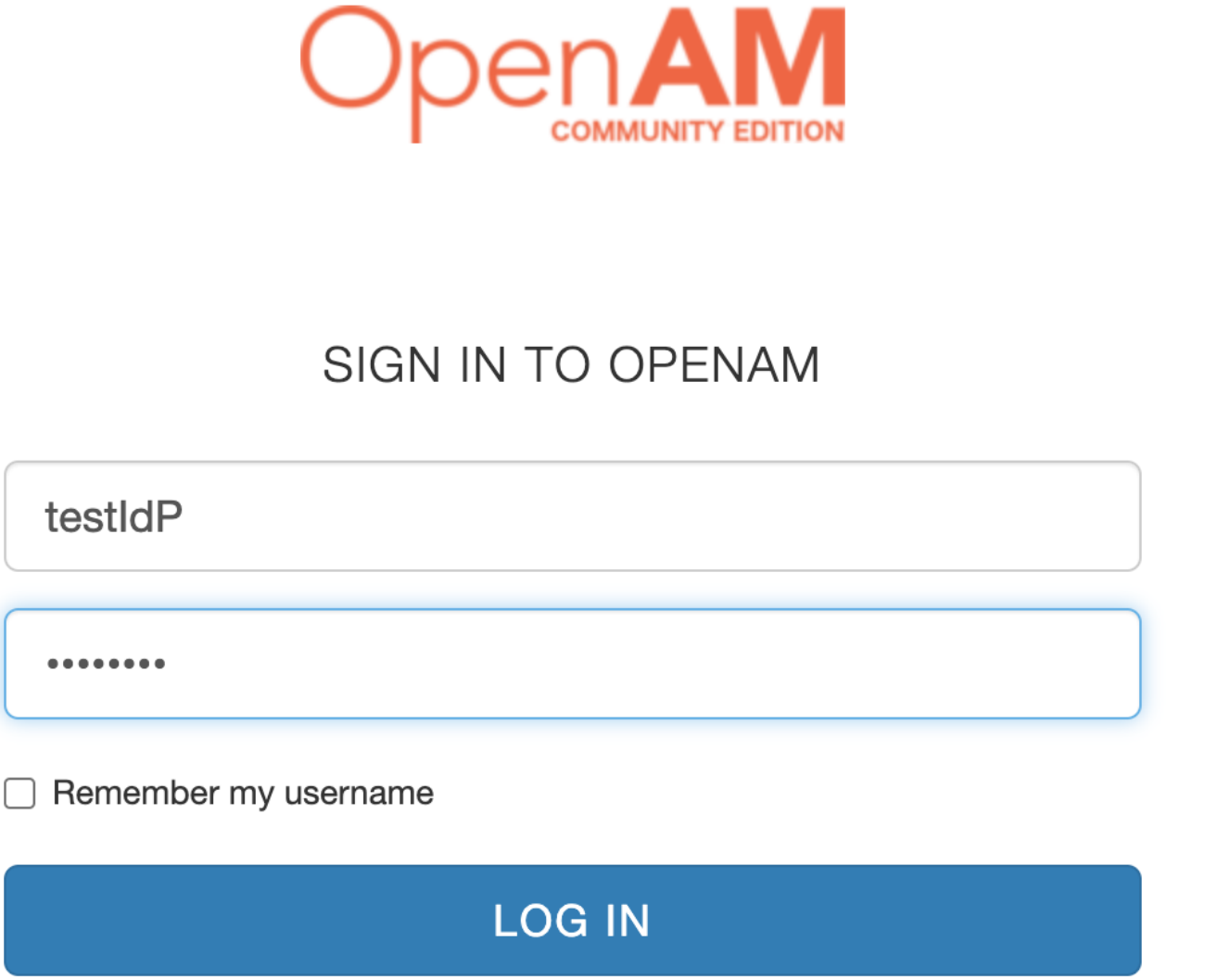


Test the Solution

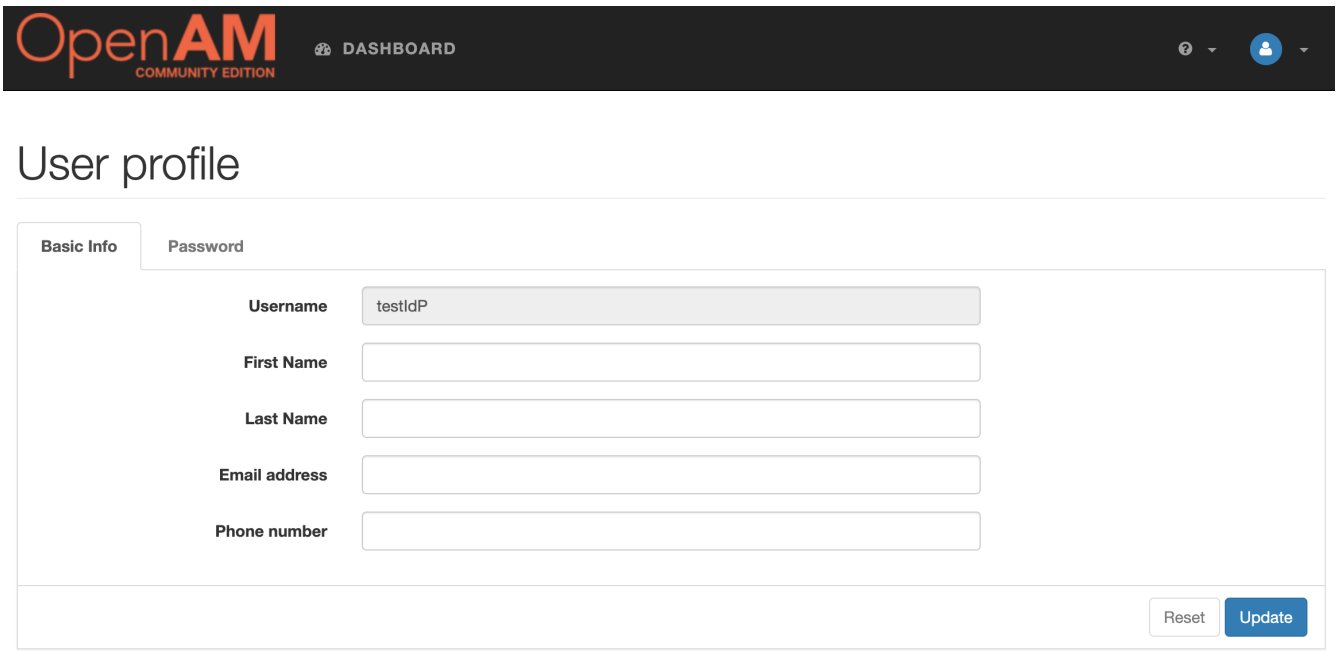
Log out of both OpenAM consoles and open the Service Provider authentication initialization link in your browser. http://sp.mycompany.org:8081/openam/spssoinit?metaAlias=/sp&idpEntityID=http%3A//idp.acme.org%3A8080/openam&RelayState=http%3A//sp.mycompany.org%3A8081/openam

As an alternative, you can use the Identity Provider authentication initalization link http://idp.acme.org:8080/openam/idpssoinit?metaAlias=/idp&spEntityID=http%3A//sp.mycompany.org%3A8081/openam&RelayState=http%3A//sp.mycompany.org%3A8081/openam

You will be redirected to the Identity Provider authentication. Enter the credentials for the user `testIdP`.



After successful authentication, the SP console with the authenticated user `testIdP` will be opened



How to Setup Two Factor Authentication in OpenAM

Introduction

In the following article, we will set up OpenAM authentication with a Time-Based One-Time Password (TOTP, RFC 6238).

OpenAM Setup

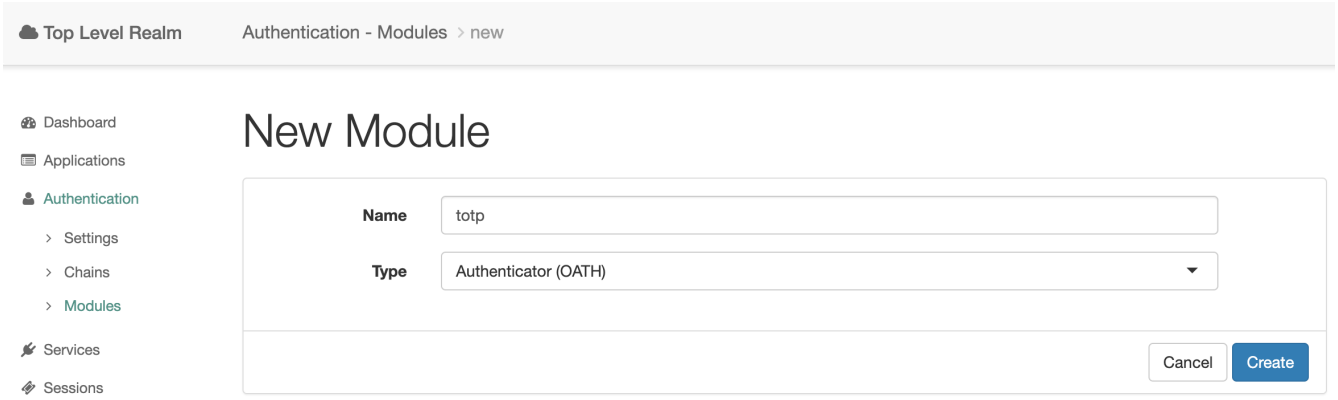
OpenAM Installation

If you have not already had OpenAM installed, you can do it as described here.

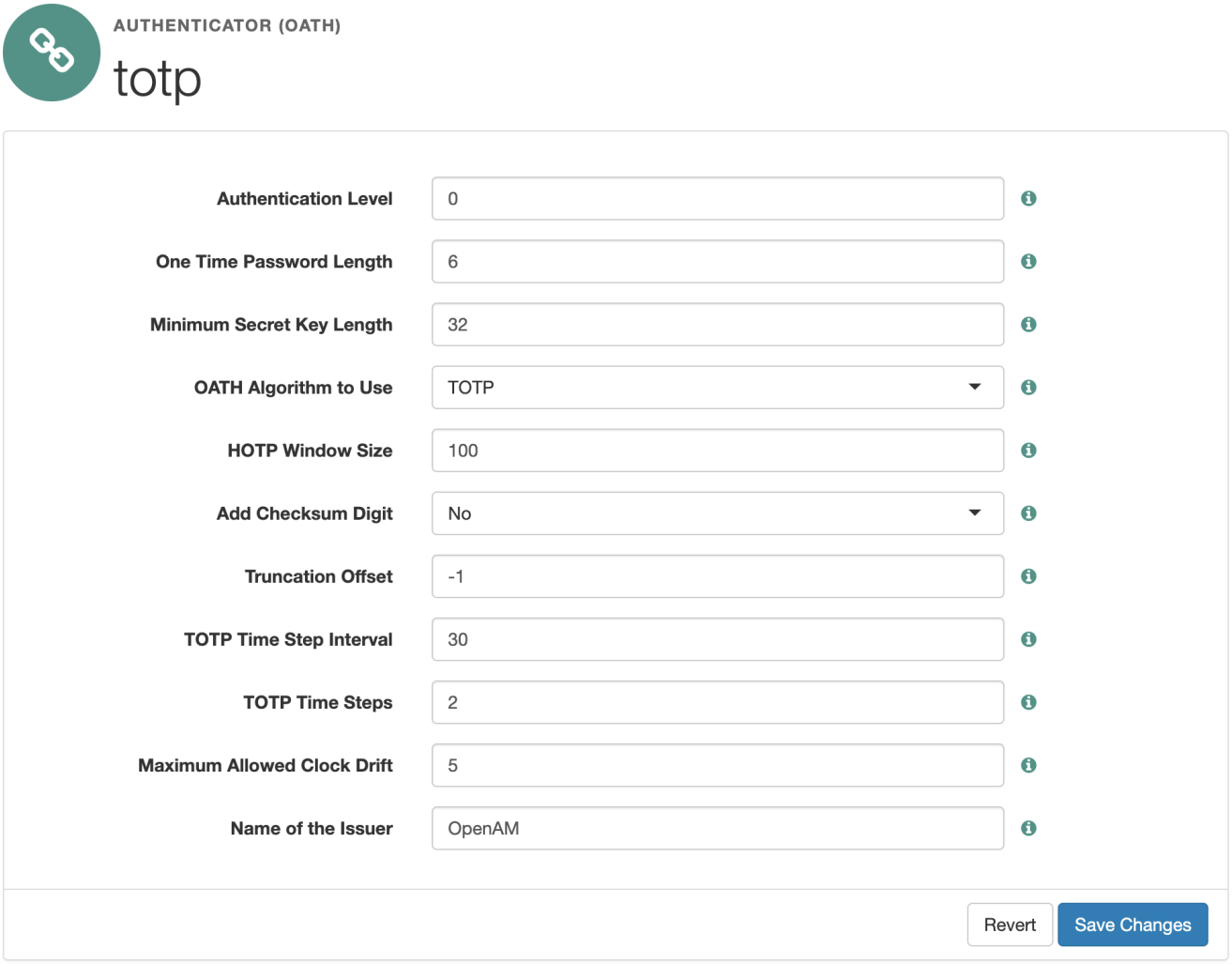
Authentication Module Setup

Open the OpenAM administration console using the link http://openam.example.org:8080/openam/console. In the login field enter the `amadmin` value, in the password field enter the value from the `ADMIN_PWD` parameter of the setup command, in this case, `passw0rd`.

Open the root realm, in the left menu, select Authentication → Modules, and press the `Add Module` button. In the form enter the name of the module, for example, `totp` and choose the module type `Authenticator (OATH)`. Press the `Create` button.

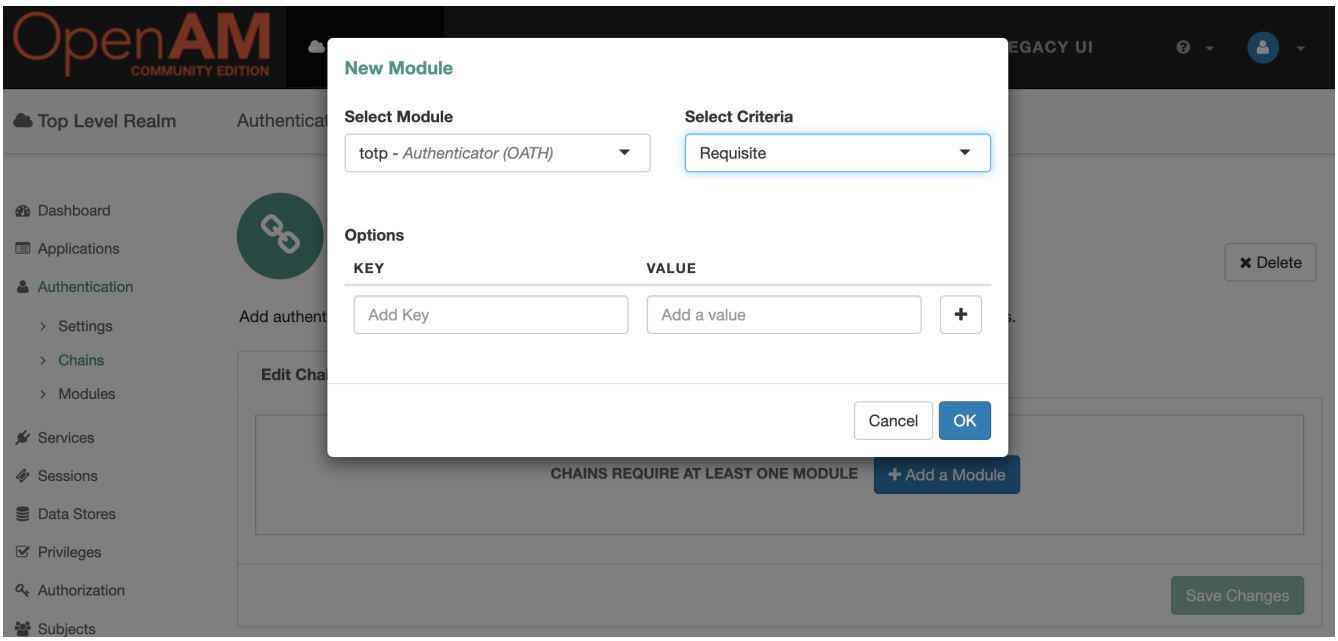


Set the `OATH Algorithm to Use` setting to `TOTP`, set the `Name of the Issuer` field to any non-empty value, for example, `OpenAM` and click the `Save Changes` button.



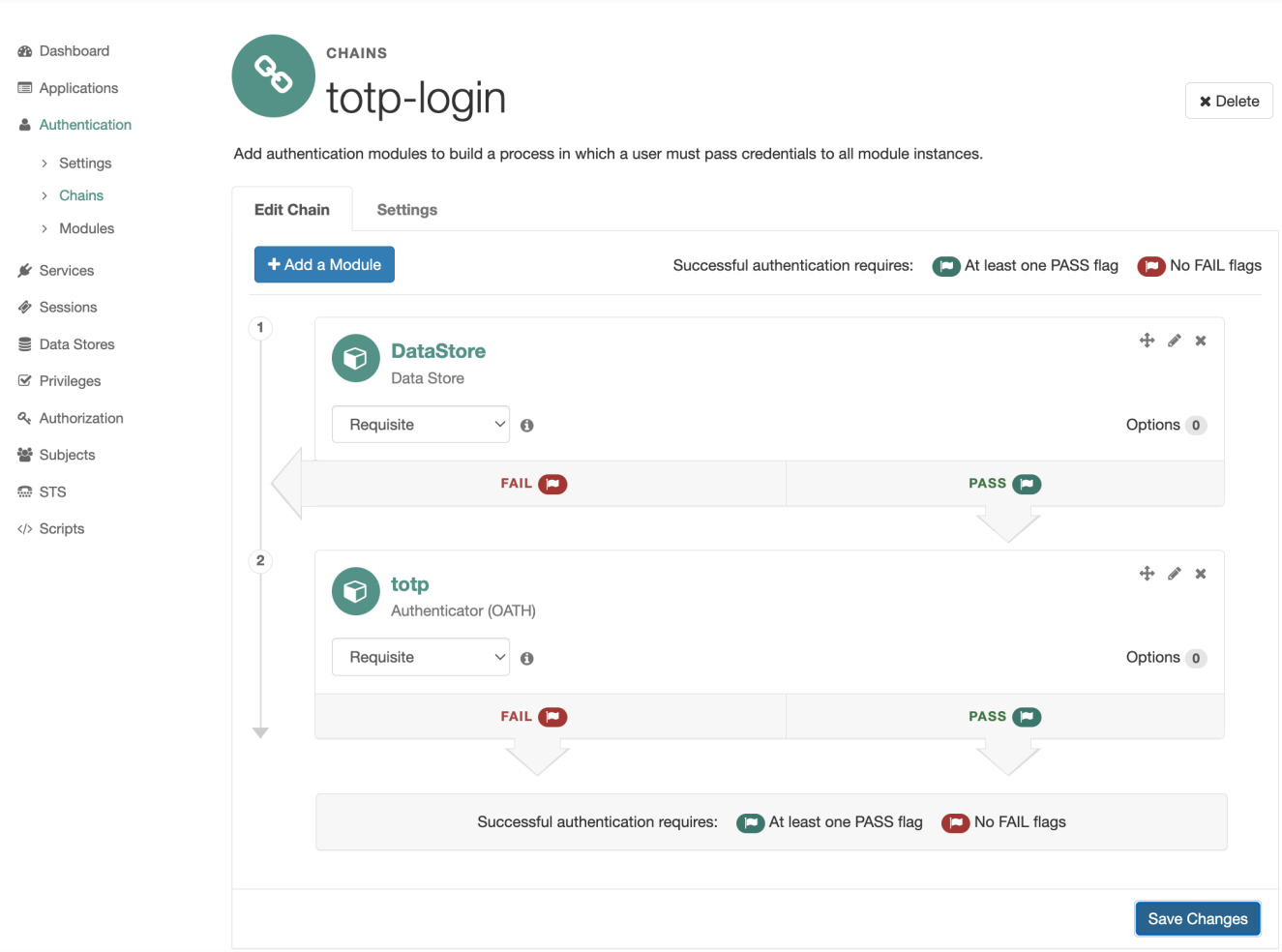
Device Registration Authentication Chain Setup

In the administration console in the realm settings in the left menu select Authentication → Chains and in the chain list press the `Add Chain` button. Enter the authentication chain name `totp-register` and press the `Create` button. In the chain settings press the `Add a Module` button and add the `totp` authentication module as shown in the picture below. Press the `OK` button and next `Save Changes`.



Login Authentication Chain Setup

In the administration console in the realm settings in the left menu select Authentication → Chains and in the chain list press the `Add Chain` button. Enter the authentication chain name `totp-login` and press the `Create` button. Add the login and password `DataStore` authentication module. Next, add the one-time password `totp` authentication module. Press the `Save Changes` button.



Test the Solution

Download on your mobile device Microsoft Authenticator or Google Authenticator application.

Setting Up a Test User

In the OpenAM admin console, navigate to the root realm and select `Subjects` from the left menu. Set the password for the `demo` user. To do this, select it in the list of users, and click the `Edit` link under Password. Enter and save the new password. Next, log out of the administrator console.

Device Registration

Log in to the console with the `demo` user account. To do this, log out of the administrator console or open a browser in Incognito mode. Go to the URL http://openam.example.org:8080/openam/XUI/#login/ and log in to OpenAM with the `demo` user credentials. After successful authentication, open the registration chain link in the browser

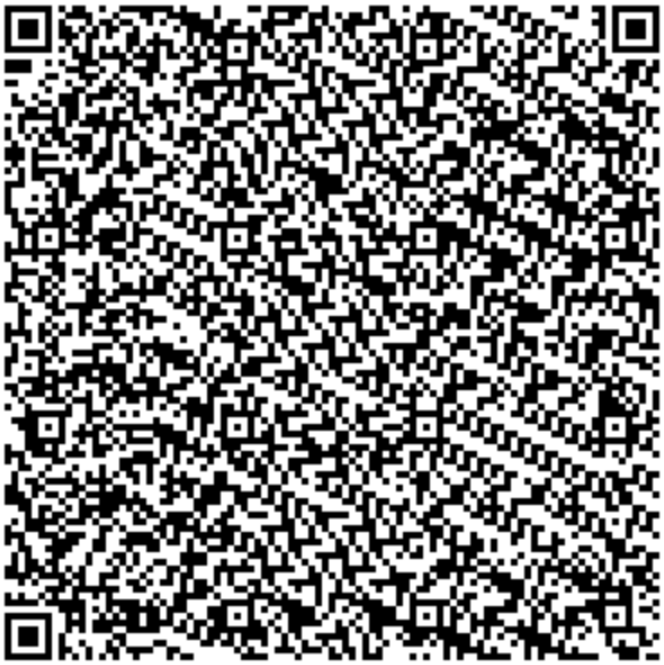
http://openam.example.org:8080/openam/XUI/#login&service=totp-register.

Press the `Register Device` button. There will be a QR code in the browser. Open the authenticator application on your mobile device and tap the add account button. Scan the QR code from the browser. A `demo` user account for OpenAM will be added to the authenticator mobile app.



REGISTER YOUR DEVICE WITH OPENAM

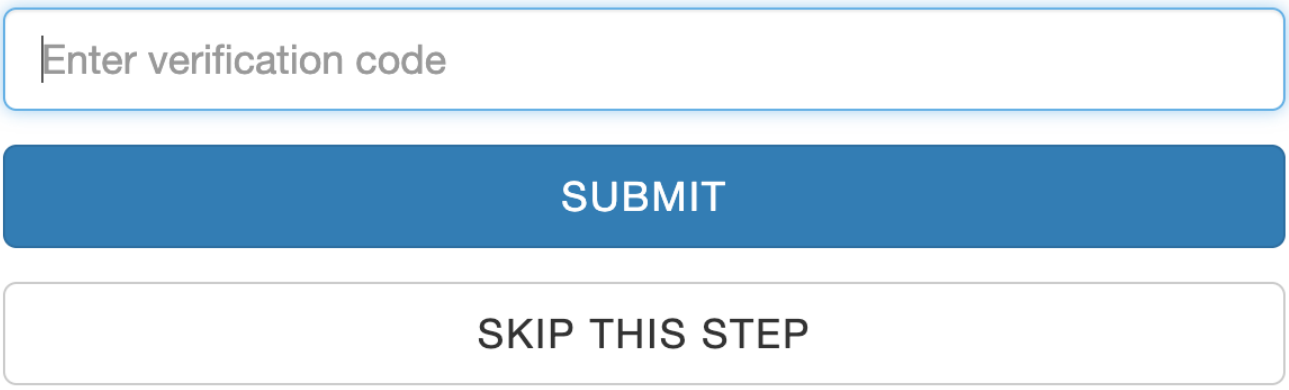
Scan the barcode image below with the ForgeRock Authenticator App. Once registered click the button to enter your verification code and login.



ON A MOBILE DEVICE?

LOGIN USING VERIFICATION CODE

Next, click the `Login Using Verification Code` button in the browser. Enter the code from the mobile app and click `Submit`.



The application has been registered.

One-Time Password Authentication

Log out of the OpenAM console or open your browser in Incognito mode. Open the http://openam.example.org:8080/openam/XUI/#login&service=totp-login link in the browser.

Enter the username and password for the user `demo`. After entering the username and password, OpenAM will prompt for a one-time password from the mobile app. Open the mobile app, select the `demo` user account, enter the one-time password from the mobile app in the browser, and press the `Submit` button. After entering the correct one-time password, authentication will be completed successfully.

How to Setup WebAuthn Authentication in OpenAM

Introduction

WebAuthn is a W3C and FIDO standard that describes Web public key authentication. For authentication, a client can use Hardware USB, Bluetooth, or NFC tokens, or mobile or laptop biometric authentication, such as fingerprint or FaceID.

In short, WebAuthn uses mutual authentication, utilizing an asymmetric encryption algorithm and exchange of credential messages encrypted with public keys (passkeys). Therefore, WebAuthn is resistant to phishing.

More details about the standard can be found at https://www.w3.org/TR/webauthn-3/

Browsers Support

WebAuthn is supported by most modern browsers, including Google Chrome, Mozilla Firefox (with partial support), Apple Safari, and Microsoft Edge, including their mobile versions. For up-to-date information about browsers and devices supporting WebAuthn, visit https://caniuse.com/?search=webauthn.

Devices Support

Passkeys created on iPhone, iPad, or Mac can be used on the same device or another iPhone, iPad, or Mac with the same Apple ID. The passkeys are synchronized automatically.

Passkeys created on Android devices can be used on an Android device with the same Google account. The passkeys are synchronized automatically

More details at the link https://passkeys.dev/device-support/

OpenAM Setup

Since WebAuthn in a browser only works over HTTPS or on the localhost domain, for demonstration purposes we will deploy OpenAM in a Docker container on localhost.

Create a network in Docker for OpenAM

```
docker network create openam
```

Then run the OpenAM Docker container with the following command

```
docker run -p 8080:8080 --network openam --name openam openidentityplatform/openam
```

Once the OpenAM server is running, perform the initial configuration by running the following command and wait for the configuration to complete.

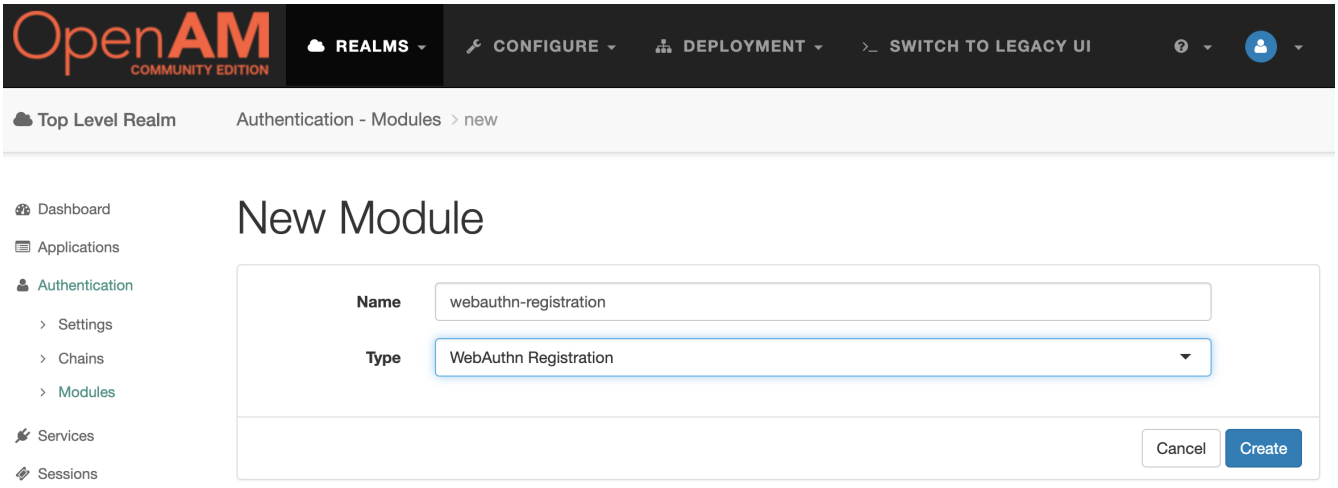
```
docker exec -w '/usr/openam/ssoconfiguratortools' openam bash -c \
'echo "ACCEPT_LICENSES=true
SERVER_URL=http://localhost:8080
DEPLOYMENT_URI=/$OPENAM_PATH
BASE_DIR=$OPENAM_DATA_DIR
locale=en_US
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=passw0rd
AMLDAPUSERPASSWD=p@passw0rd
COOKIE_DOMAIN=localhost
ACCEPT_LICENSES=true
DATA_STORE=embedded
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=localhost
DIRECTORY_PORT=50389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1689
ROOT_SUFFIX=dc=openam,dc=example,dc=org
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=passw0rd" > conf.file && java -jar openam-configurator-tool*.jar
--file conf.file'
```

After successful configuration, you can proceed to further configuration. Let's configure the WebAuthn registration and authentication chains.

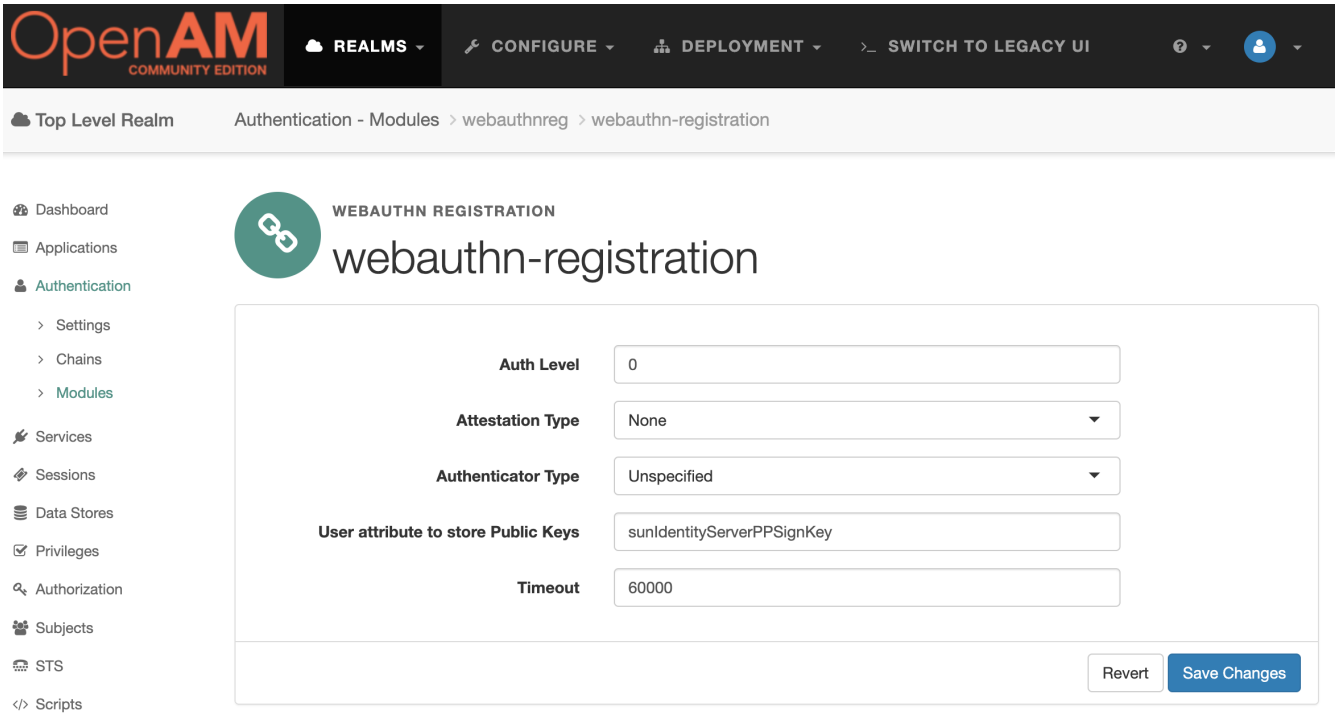
WebAuthn Registration Setup

Authentication Module Setup

Open the OpenAM administration console at http://localhost:8080/openam/XUI/#login/ In the login field enter the `amadmin` value, in the password field enter the value from the `ADMIN_PWD` parameter of the setup command, in this case, `passw0rd`. Select the root realm and then goto Authentication → Modules in the left menu. Create a new authentication module `WebAuthn Registration`

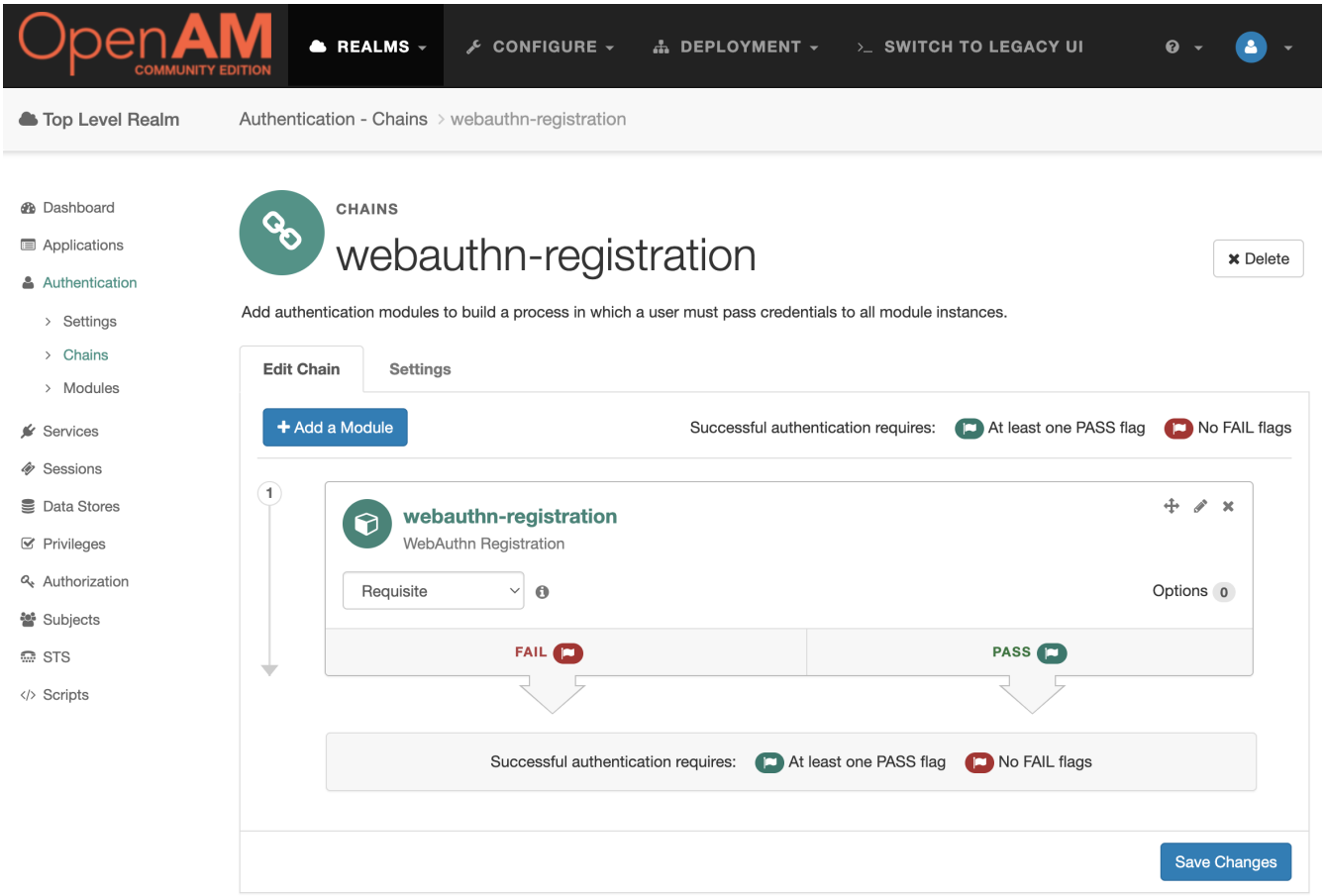


The default module settings can be left unchanged.



Authentication Chain Setup

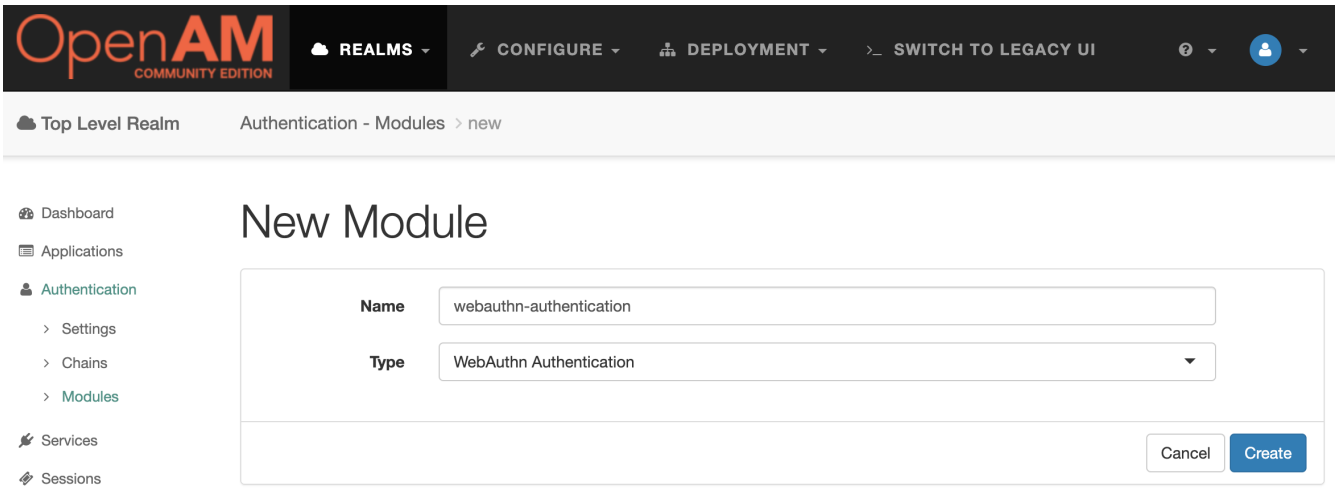
Go to the admin console, select the root realm and select Authentication → Chains from the menu. Create a `webauthn-registration` authentication chain with the created `webauthn-registration` module.



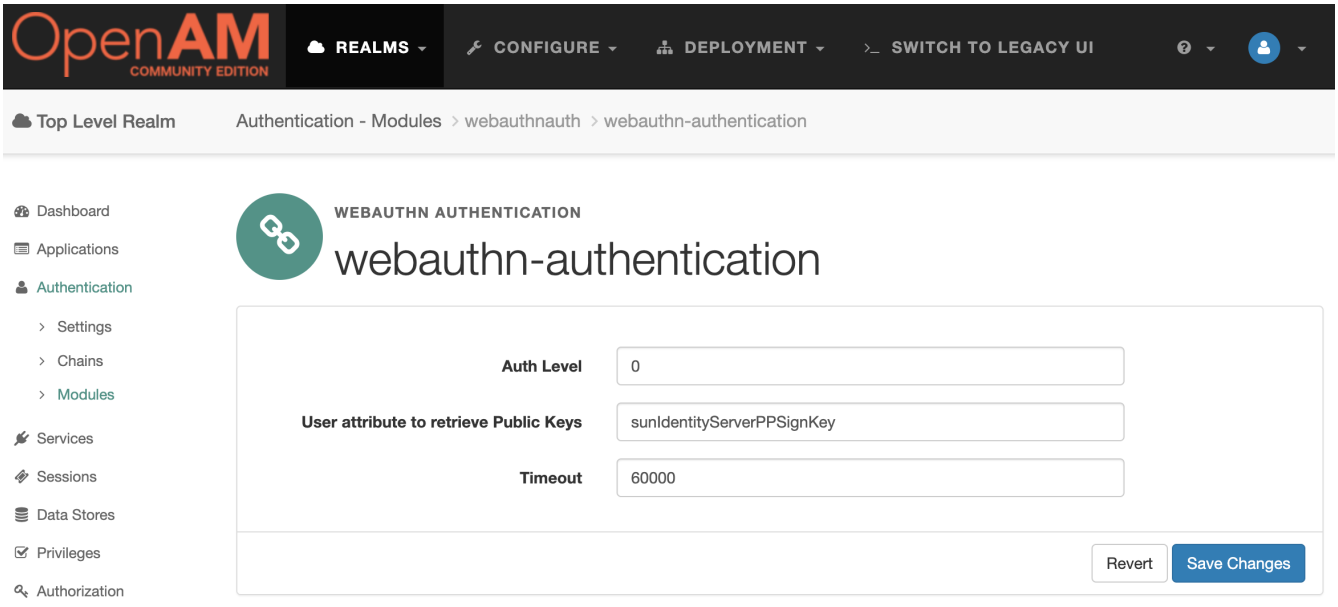
WebAuthn Authentication Setup

Authentication Module Setup

Open the OpenAM administration console. Select the root realm and then goto Authentication → Modules in the left menu. Create a new authentication module `WebAuthn Authentication`

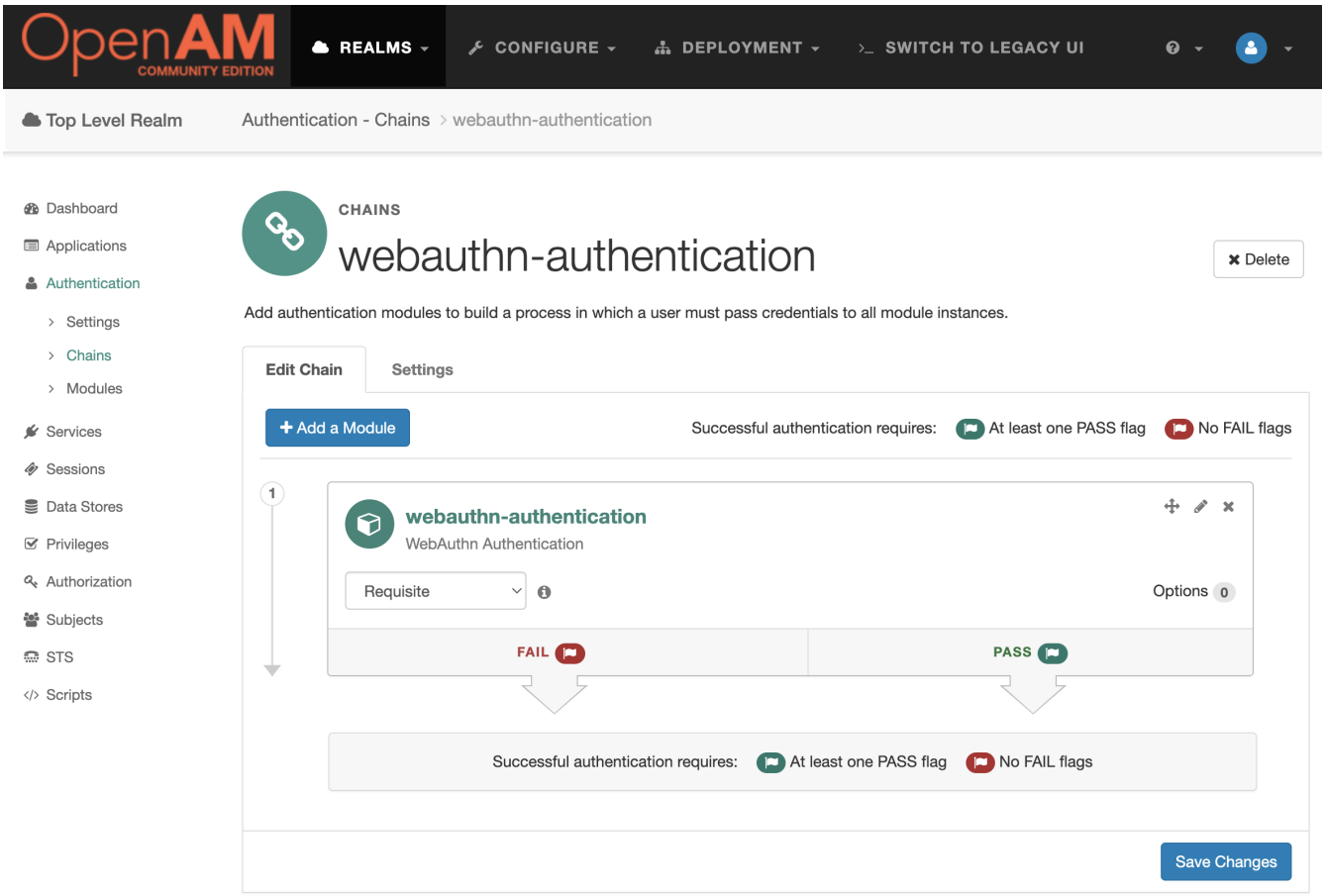


The default module settings can be left unchanged.



Authentication Chain Setup

Go to the admin console, select the root realm and select Authentication → Chains from the menu. Create a `webauthn-authentication` authentication chain with the created `webauthn-authentication` module.



Test the Solution

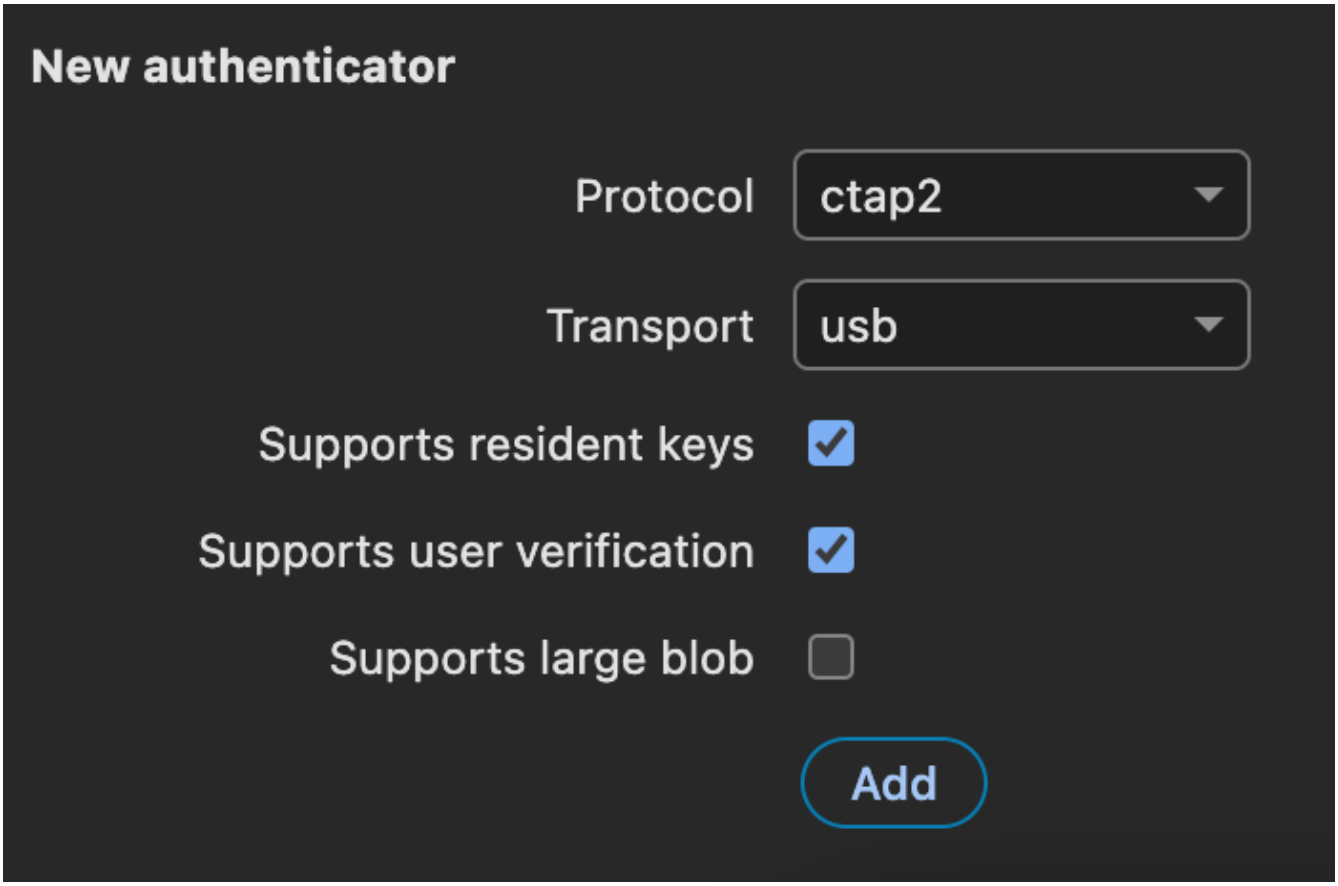
Log out of the OpenAM admin console or open a browser in Incognito mode and log in at http://localhost:8080/openam/XUI/#login with the user credentials `demo`. In the login field enter `demo`

in the password field enter `changeit`.

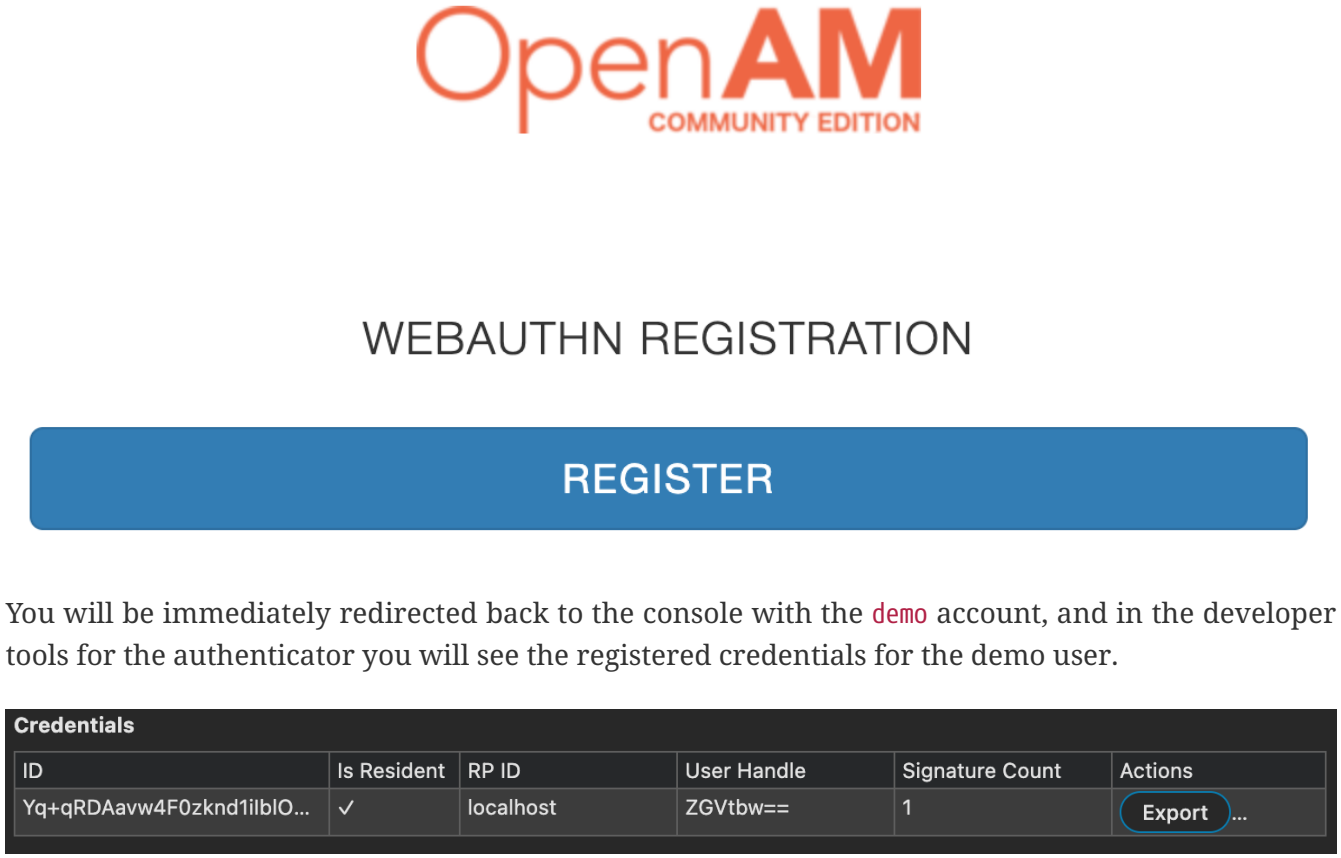
Registration

For demonstration purposes, we will use the WebAuthn emulator built into the browser. How to enable it is described at https://developer.chrome.com/docs/devtools/webauthn.

Add a virtual authenticator with Supports resident keys and Supports user verification settings enabled.



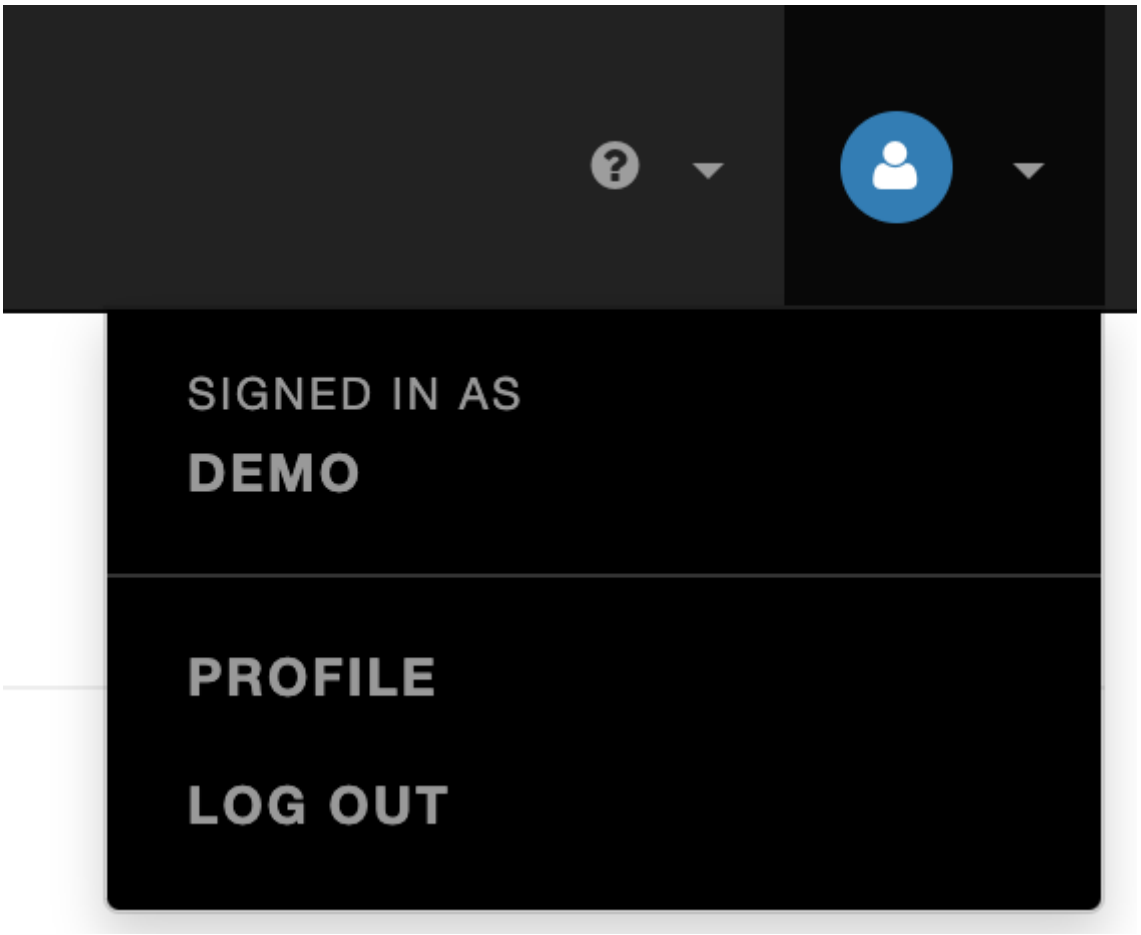
Open the registration authentication chain at http://localhost:8080/openam/XUI/#login&service=webauthn-registration and click the `Register` button.



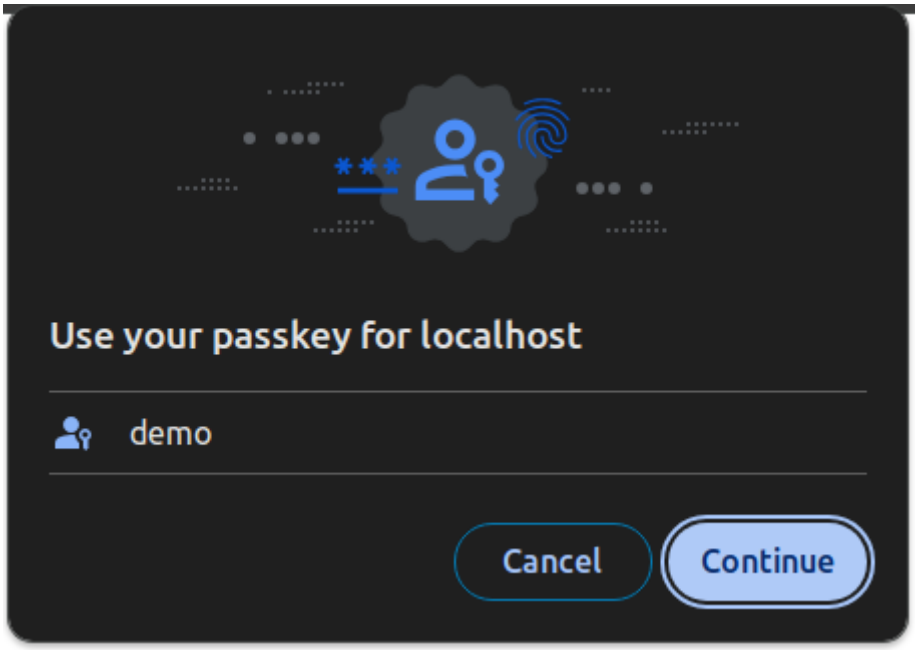
You will be immediately redirected back to the console with the `demo` account, and in the developer tools for the authenticator you will see the registered credentials for the demo user.

Authentication

In the Console under the user `demo` click on the user icon and select Logout



Go to http://localhost:8080/openam/XUI/#login&service=webauthn-authentication and click the `Log In` button. An account selection window will pop up.



Select the `demo` account and click Continue. You will be authenticated with the `demo` account .

How to Start OpenAM and OpenDJ in Separate Docker Containers

Requeirements

To reproduce steps in the following article, Docker and Docker Compose should be installed.

The sample configuration used in the following article could be found here

OpenAM Perparation

Create `docker-compose.yml` file and add OpenAM service:

```
version: '3.9'

services:
  openam:
    image: openidentityplatform/openam:latest
    ports:
      - "8080:8080"
    hostname: openam.example.com
    volumes:
      - ./persistence/openam:/usr/openam/config
```

Add OpenAM FQDN to your `hosts` file on your machine, for example

```
127.0.0.1   auth.example.com
```

In the `persistence/openam` folder OpenAM will store its configuration data

OpenDJ Preparation

Add OpenDJ service to `docker-compose.yml` file:

```
  opendj:
    image: openidentityplatform/opendj:latest
    hostname: opendj.example.com
    ports:
      - "1389:1389"
      - "1636:1636"
      - "4444:4444"
    volumes:
      - ./opendj/bootstrap/data/:/opt/opendj/bootstrap/data #initial data
      - ./opendj/bootstrap/schema/:/opt/opendj/bootstrap/schema #schema
```

```
      - ./persistence/opendj:/opt/opendj/data #opendj data
    environment:
      - BASE_DN=dc=openam,dc=openidentityplatform,dc=org #should be yours base DN
```

`opendj/boststrap` directory should exist on the host machine and contain two subfolders: `data` - for initial data и `schema` - schema files.

The files could be found here

If base DN id different from `dc=openam,dc=openidentityplatform,dc=org` it should be changed in `opendj/boststrap/data/samlple.ldif` file as well

Start and Configuration

Start Services

Start OpenAM and OpenDJ with the following command:

```
docker-compose up
```

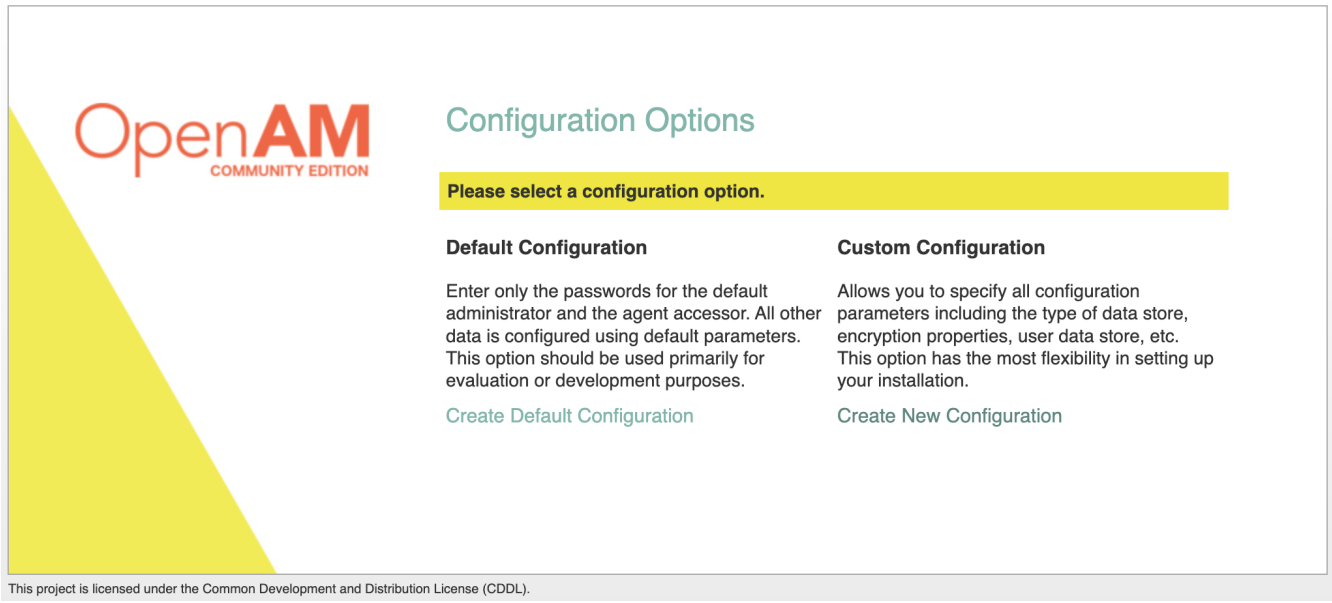
You shoudl see the following lines in the logs:

```
openidentityplatform-docker-openam-1  | 15-Jun-2022 10:09:46.063 INFO [main]
org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-8080"]
openidentityplatform-docker-openam-1  | 15-Jun-2022 10:09:46.133 INFO [main]
org.apache.catalina.startup.Catalina.start Server startup in [36437] milliseconds
openidentityplatform-docker-opendj-1  | OpenDJ is started
```

OpenAM Configuration

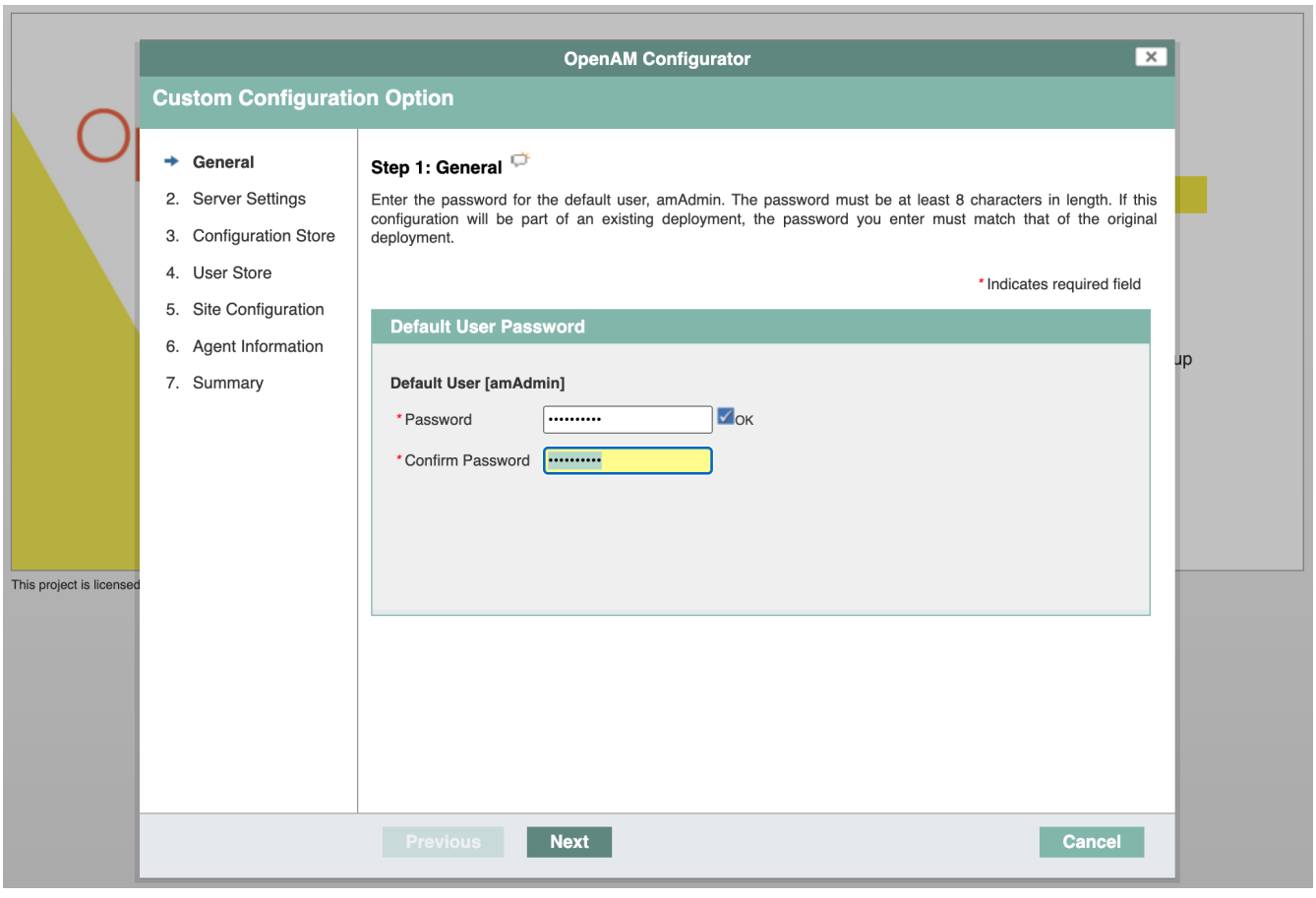
After OpenAM and OpenDJ are started, open OpenAM URL in the browser, for example http://auth.example.com:8080/openam

OpenAM configuration window should appear:

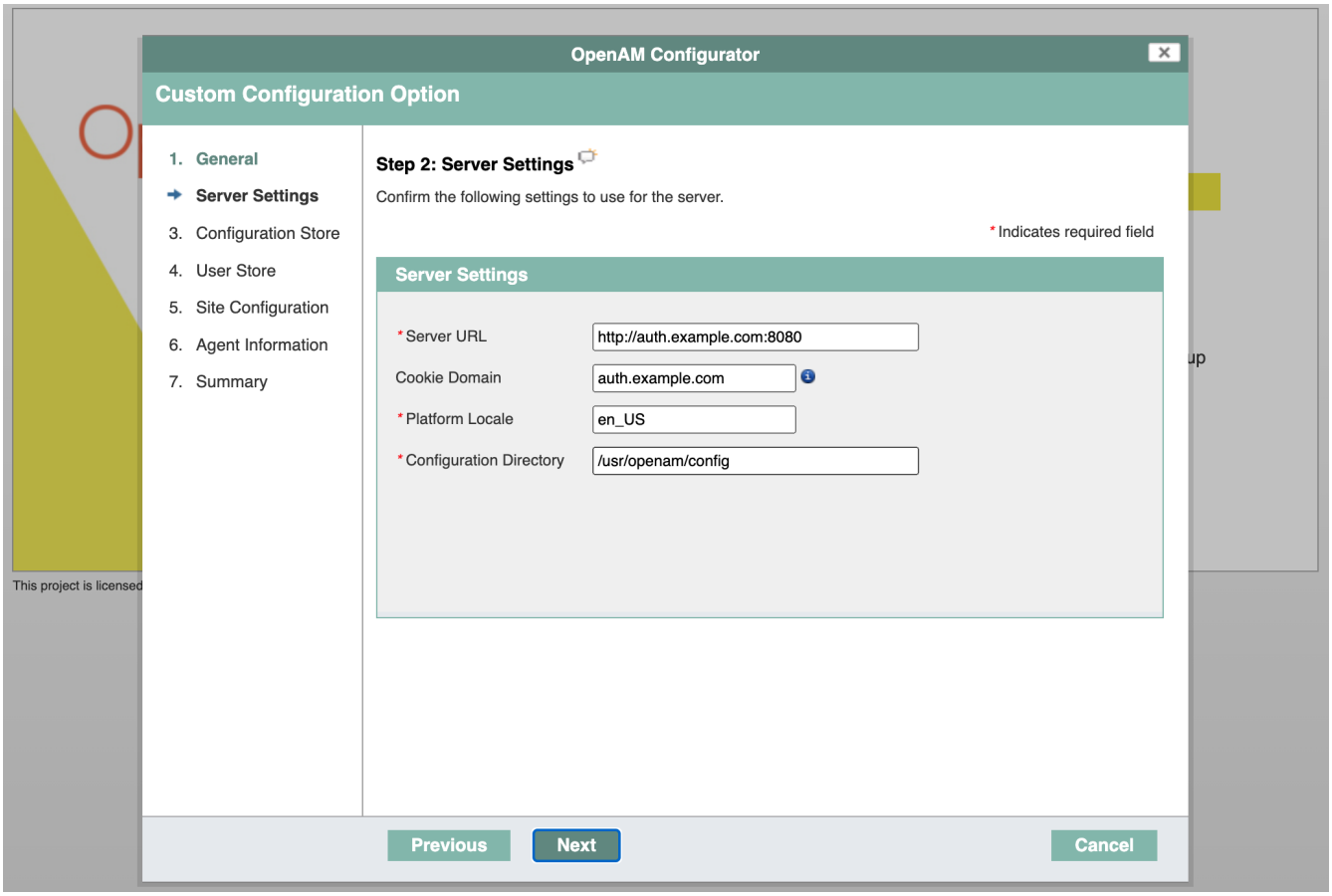


Select Custom Configuration and press Create New Configuration. Read and accept the license agreement.

Enter and confirm amAdmin password, then press the Next button.



Leave server settings unchanged and press the Next button.

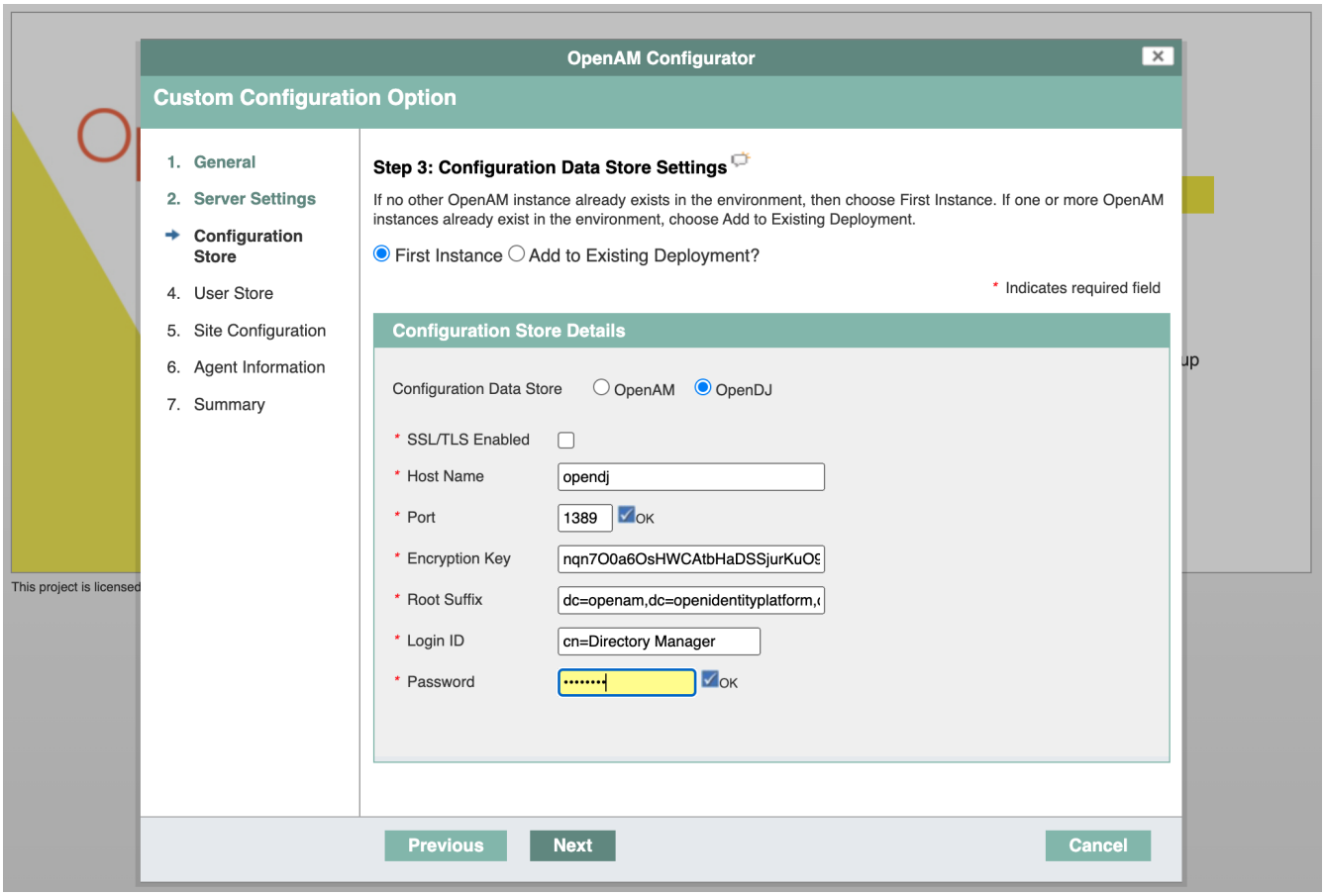


On the Configuration Data Store Settings step set the following settings:

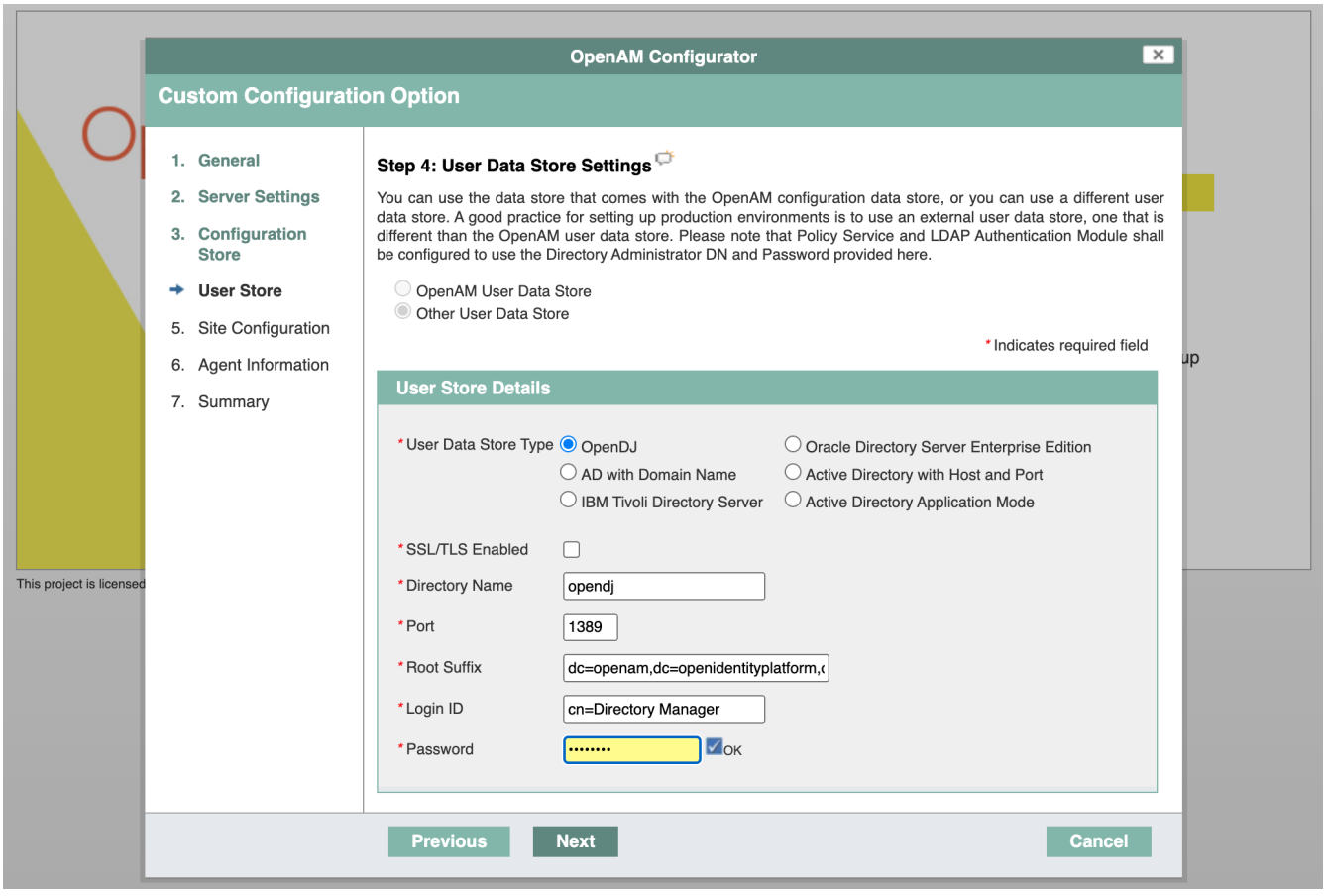
- **Configuration Data Store** radiobutton to **OpenDJ**,
- Set **Host Name** to **opendj**, as OpenDJ Docker internal hostname
- Set **Port** to 1389 as specified for OpenDJ Docker container
- Set OpenDJ password as specified for OpenDJ Docker container as well (`password` by default).

If you changed base DN for OpenDJ, set the required Root Suffix as well.

Then press the Next button.



On the User datastore settings set OpenDJ password as it has been set in the previous step and press the Next button.

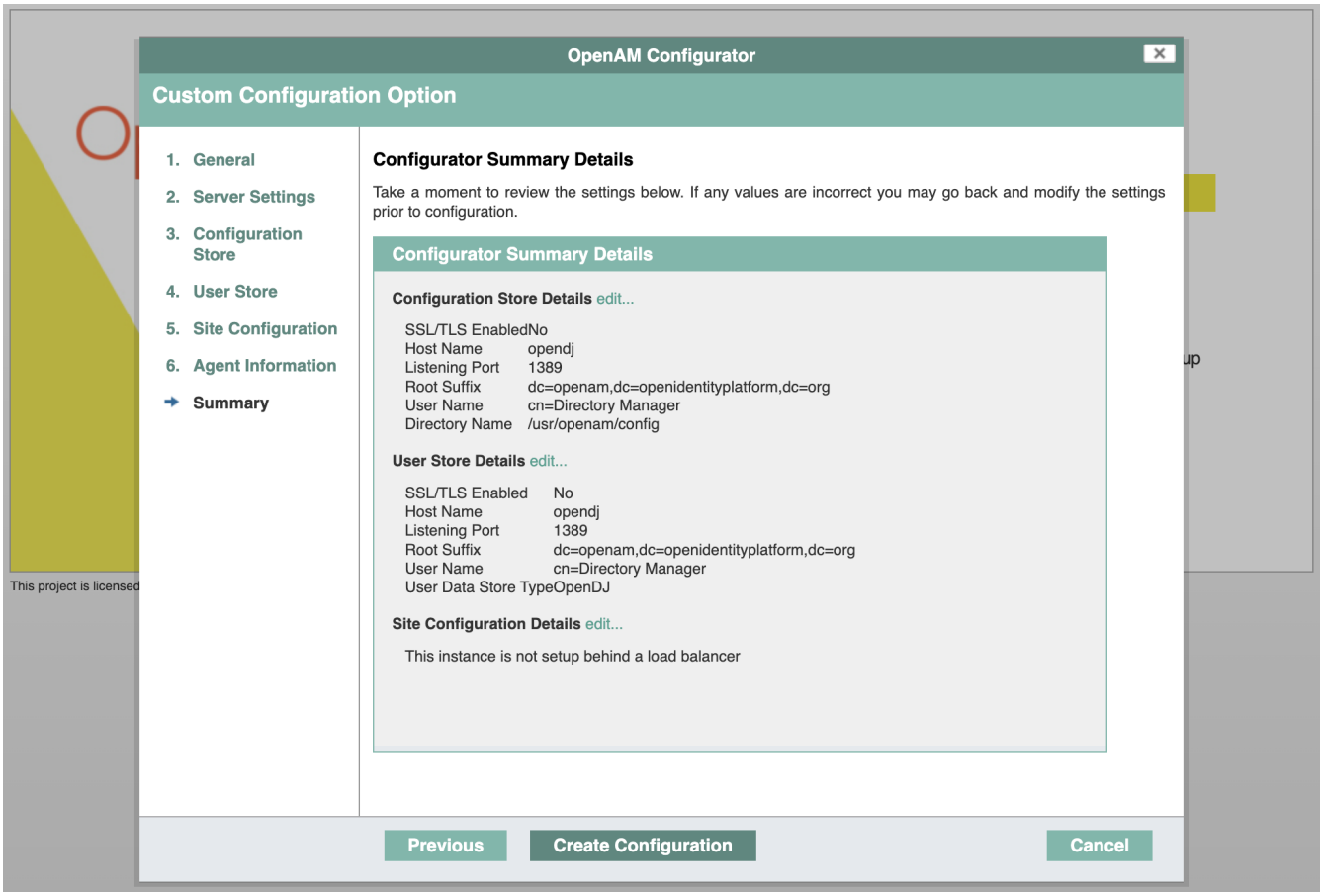


If this instance is behind a load balancer, set the site configuration settings. If it's not, just press the

Next button.

On the Step 6: Default Policy Agent User, enter and confirm default policy agent password and press the Next button.

Review configuration summary and press the Create Configuration button.



If everything is ok, proceed to login page.

If you have any additional questions, feel free to ask us!

How to Use Apache Cassandra as User and Session DataStore in OpenAM

Introduction

One of the performance bottlenecks of authentication is User DataStore. This becomes noticeable when the total number of user accounts exceeds `4 000 000`.

To maintain acceptable performance, the most obvious solution is to scale User DataStore horizontally or vertically. But with an increasing amount of data, DataStore scaling does not affect performance.

With the increasing load on the authentication system, the most obvious solution is to scale horizontally or vertically the number of user repository nodes.

Another alternative solution is to use storage specifically designed for working with large amounts of data. One of such storages is Apache Cassandra. It has proved itself as a high-availability and high-performance datastore.

Prepare Apache Cassandra

Allow materialized views for Cassandra 4+ (in version 3 enabled by default) in conf/cassandra.yaml:

```
# Enables materialized view creation on this node.
# Materialized views are considered experimental and are not recommended for
production use.
enable_materialized_views: true
```

Prepare Apache Cassandra to use as User DataStore

Create the corresponding Keyspaces and tables, for each OpenAM realm, from template import.cql

```
bin/cqlsh -f import.cql
```

Prepare Apache Cassandra to use as CTS

Create the corresponding Keyspaces and tables, for OpenAM CTS, from template schema.cqlsh

```
bin/cqlsh -f schema.cqlsh
```

OpenAM Setup

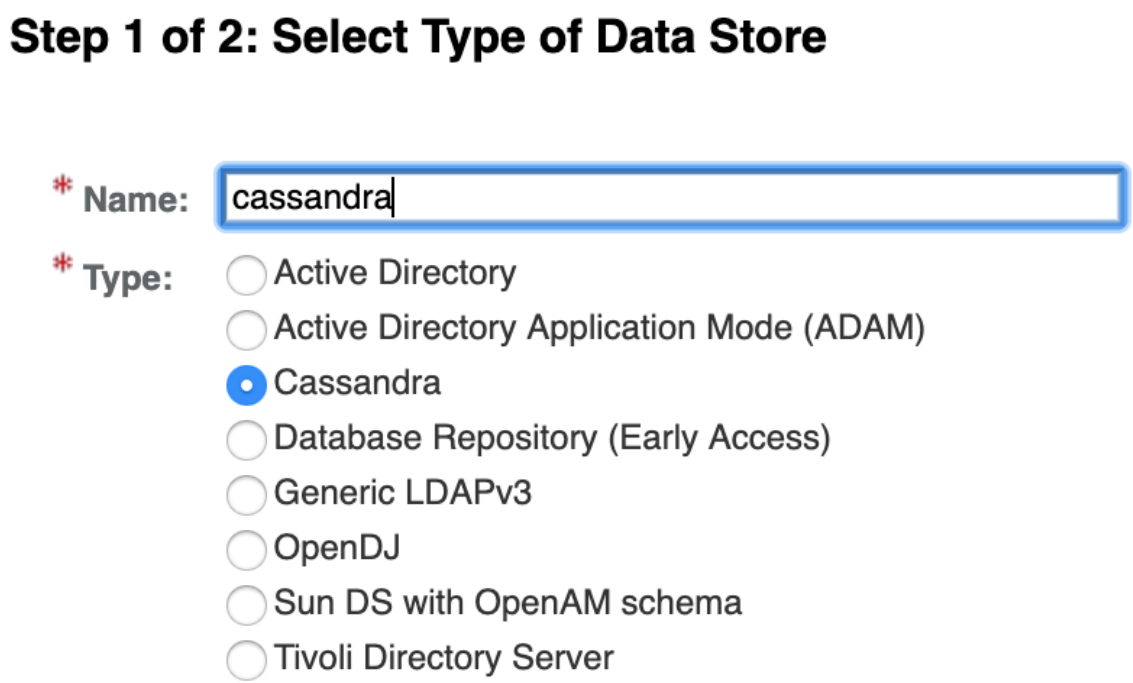
Add cassandra contact-points info in bin/setenv.sh and restart OpenAM:

```
export CATALINA_OPTS=" ... \
-Ddatastax-java-driver.basic.contact-points.0=ip1:9042 \
-Ddatastax-java-driver.basic.contact-points.1=ip2:9042 \
-Ddatastax-java-driver.basic.load-balancing-policy.local-datacenter=datacenter1 \
-Ddatastax-java-driver.advanced.auth-provider.class=PlainTextAuthProvider \
-Ddatastax-java-driver.advanced.auth-provider.username=user \
-Ddatastax-java-driver.advanced.auth-provider.password=password \
"
```

OpenAM create DataStore

Open OpenAM administration console, goto `Realms` select target realm, then goto `DataStores` and create new DataStore.





Then setup following settings:

Server settings

Setting	Value
Servers	don't use (leave blank)
User name	don't use (leave blank)
Password	don't use (leave blank)
Password (confirm)	don't use (leave blank)
Keyspace	Cassandra keyspace for this realm `realm_name`

Plugin configuration

Setting	Value
Database Repository Plugin Class Name	`org.openidentityplatform.openam.cassandra.Repo`
Tables	`group=groupuser=userrealm=realm`
Operations	`realm=read,create,edit,delete,service` group=read,create,edit,delete ` `user=read,create,edit,delete,service`

User Configuration

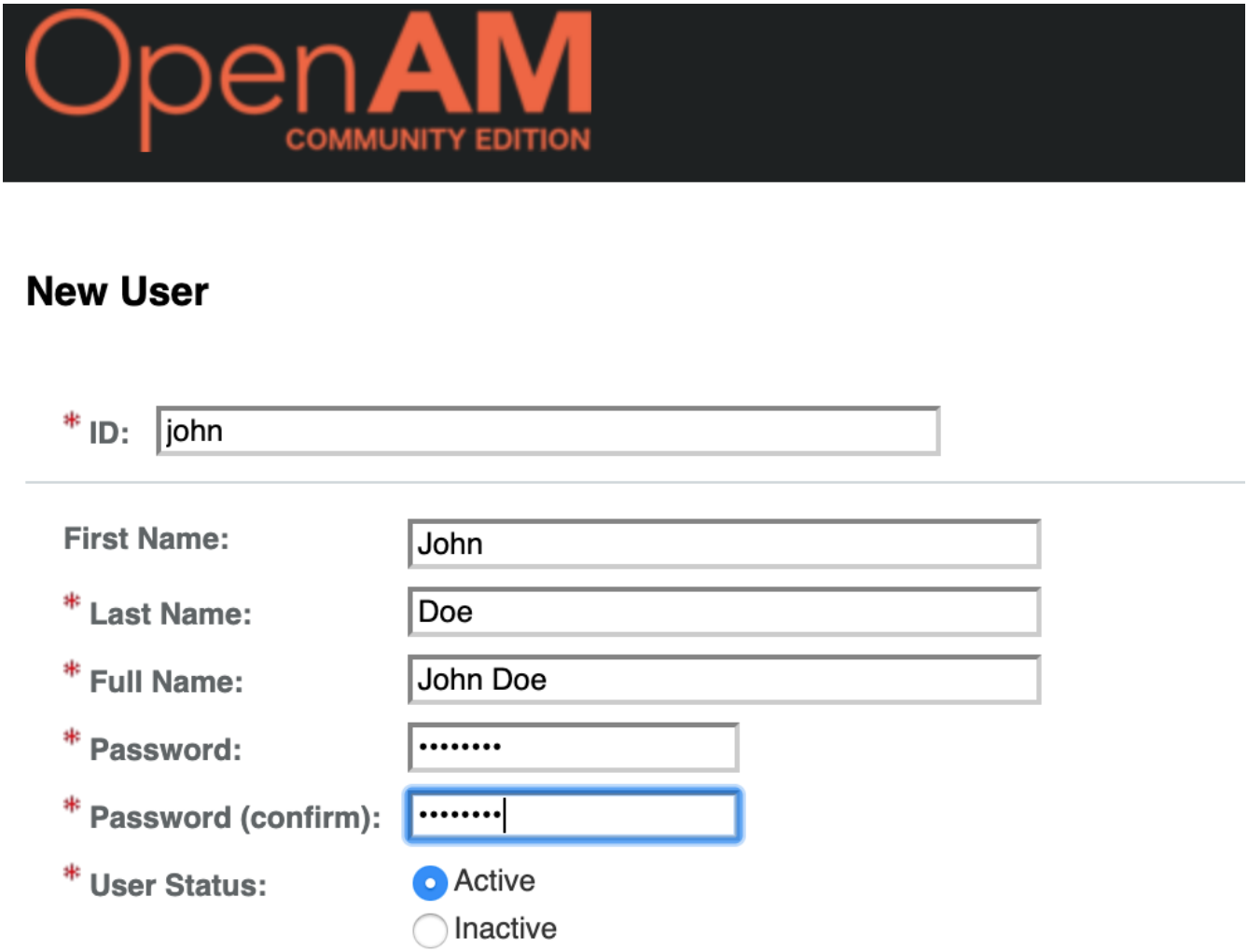
Setting	Value
TTL	`realm:attr1=86400` user:attr2=86400 ` `group:attr3=86400`
Attribute Name of User Status	inetuserstatus
User Status Active Value	Active

Group Configuration

Setting	Value
Attribute Name for Group Membership	memberOf

Click the `Save` button

You can check datastore. Goto `Subjects -> Users` and create a new user account. For example:



Click Save and make sure, the record exists in Cassandra:

```
cassandra@cqlsh:realm_name> SELECT * FROM realm_name.values ;
```

OpenAM CTS Setup

Add cassandra external CTS settings in bin/setenv.sh:

```
export CATALINA_OPTS=" ... \
-Dorg.forgerock.openam.sm.datalayer.module.CTS_ASYNC=org.openidentityplatform.openam.cassandra.CTSAsyncConnectionModule  \
-Dorg.forgerock.openam.sm.datalayer.module.CTS_MAX_SESSION_TIMEOUT_WORKER=org.openidentityplatform.openam.cassandra.CTSConnectionModule  \
-Dorg.forgerock.openam.sm.datalayer.module.CTS_SESSION_IDLE_TIMEOUT_WORKER=org.openidentityplatform.openam.cassandra.CTSConnectionModule  \
-Dorg.forgerock.openam.sm.datalayer.module.CTS_EXPIRY_DATE_WORKER=org.openidentityplatform.openam.cassandra.CTSConnectionModule  \
-Dorg.forgerock.services.cts.store.root.suffix=cts.tokens  \
"
```

restart OpenAM, login and check session info in Cassandra

```
cassandra@cqlsh:cts> SELECT * FROM cts.tokens ;
```

How to disable XUI by default

Legacy UI

For one instance in cluster

Goto Configuration > Servers and Sites > Default Server Settings > Advanced and add:

```
XUI.enable=false
```

or change java system property:

```
-DXUI.enable=false
```

For all instances in cluster

Goto Configuration > Servers and Sites > Select server > Advanced and add:

```
XUI.enable=false
```

XUI Interface

For one instance in cluster

Goto Deployment > Servers > Choose server > Advanced and add:

```
XUI.enable=false
```

For all instances in cluster

Goto Configure > Server Defaults > Advanced and add:

```
XUI.enable=false
```

How to make OpenAM log more verbose

To make OpenAM log more verbose please add **-Dcom.iplanet.services.debug.level=message** to Tomcat's **setenv.sh** file. Hope this will help to figure out the cause of the problem

How to setup Kerberos Authentication with OpenAM

Introduction

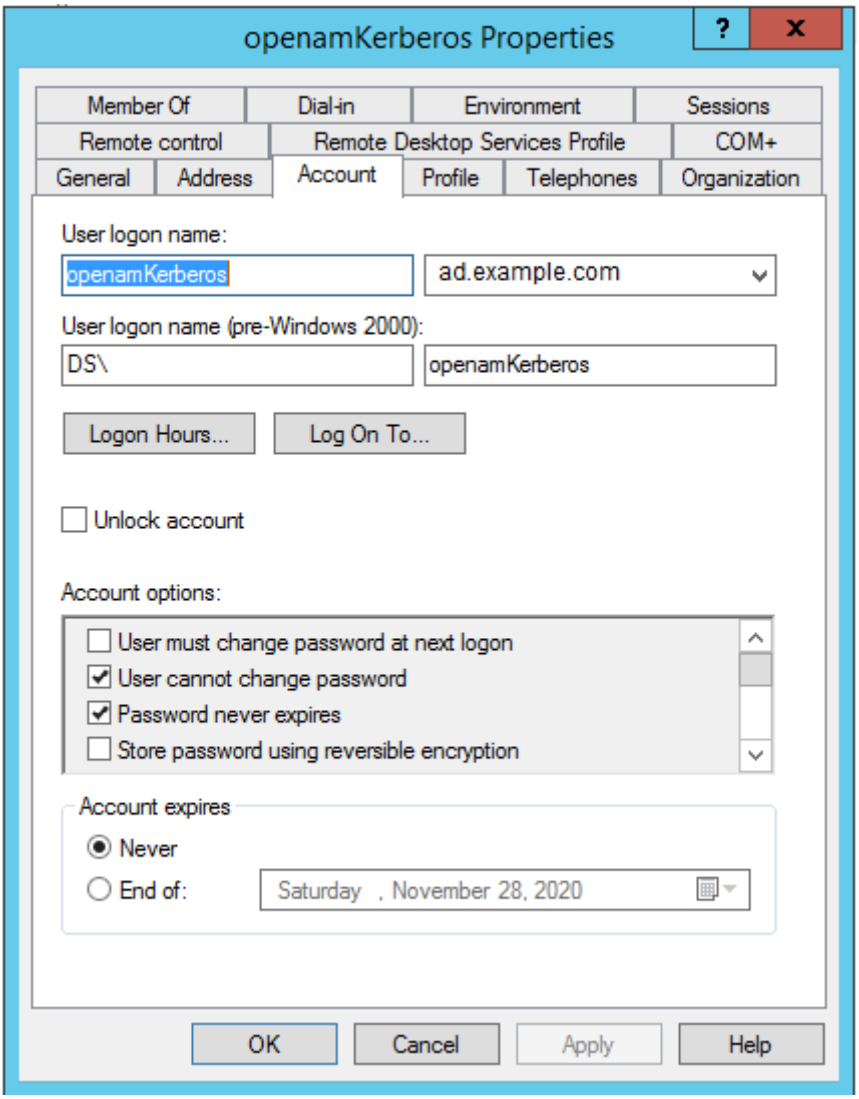
There are several ways how enterprise users can authenticate in enterprise applications. If there are many enterprise applications users should authenticate into each application and enter login and password. Even if applications use the same user account, it is painful to enter credentials every time. A solution is to use a single sign (SSO) technology. For Windows users it is Kerberos. With Kerberos, users could authenticate to web applications seamlessly using their Active Directory accounts.

Prerequisites

You have Windows Server and user accounts stored in the Active directory. You must also have OpenAM installed. Here is how to quickly install OpenAM.

Windows setup

Create an account for Kerberos authentication in the Active Directory. When creating an account, set checkboxes `User cannot change password` и `Password never expires enabled` as shown in the picture below.



In user account properties in the Account tab in Account Options enable the checkbox `This account supports Kerberos AES-256 bit encryption`.

On the domain controller create a keytab file openamKerberos.keytab in a current directory. To do this, execute the following command in the Windows terminal:

```
ktpass -out openamKerberos.keytab -princ HTTP/openam.example.com@AD.EXAMPLE.COM -pass
+rndPass -maxPass 256 -mapuser openamKerberos -crypto AES256-SHA1 -ptype
KRB5_NT_PRINCIPAL
```

In this command in `-princ` parameter openam.example.com - is your OpenAM hostname and EXAMPLE.COM - your Active Directory domain name should be uppercase.

Copy openamKerberos.keytab file to a directory, where OpenAM can read the file.

On your firewall open network access from OpenAM to TCP/UDP ad.example.com:88

Check keytab file on OpenAM machine:

```
$ klist -k -t openamKerberos.keytab
Keytab name: FILE:openamKerberos.keytab
```

```
KVNO Timestamp Principal
---- ------------------ ------------------------------------------------------
0 01.01.1970 03:00:00 HTTP/openam.example.com@AD.EXAMPLE.COM
```

OpenAM Configuration

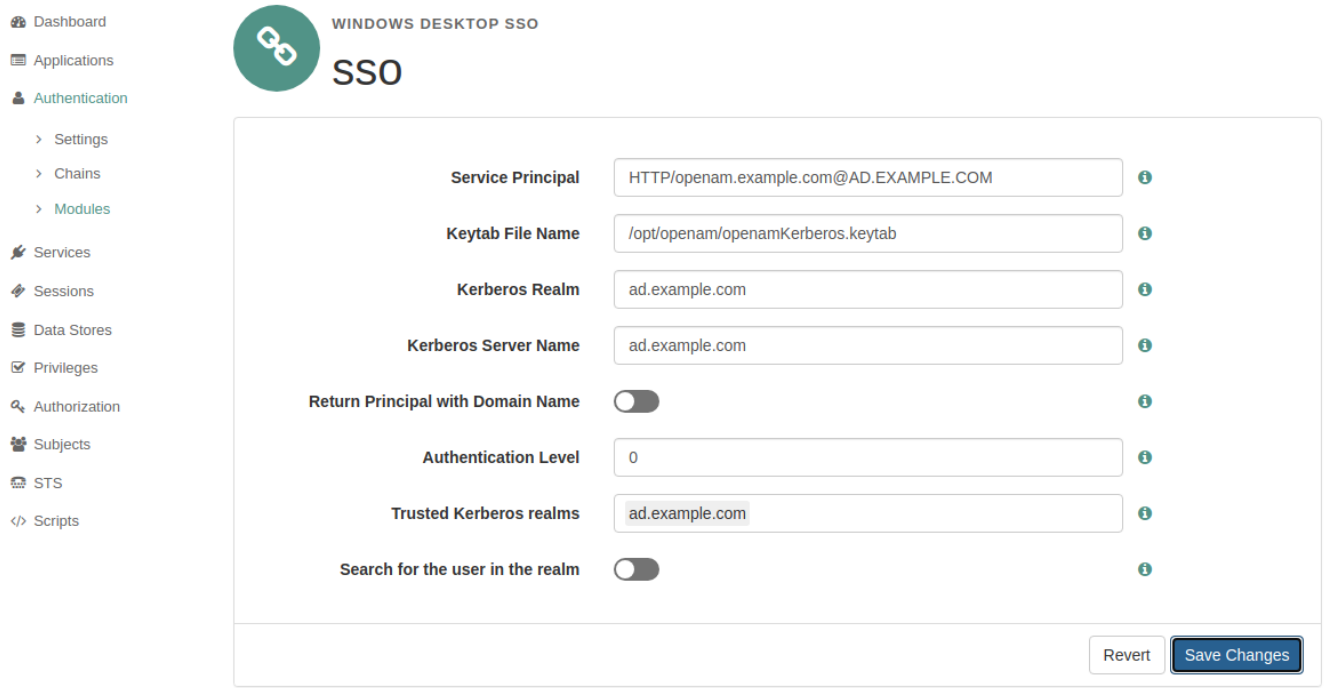
Setup Authentication Module

Go to the OpenAM administrator console at http://openam.example.org:8080/openam/XUI/#login/ In the login field enter the value `amadmin`, in the password field enter the value from the `ADMIN_PWD` parameter of the setup command, in this case, `passw0rd`.

Select the root realm and select Authentication → Modules from the menu. Create a new Active Directory authentication module.

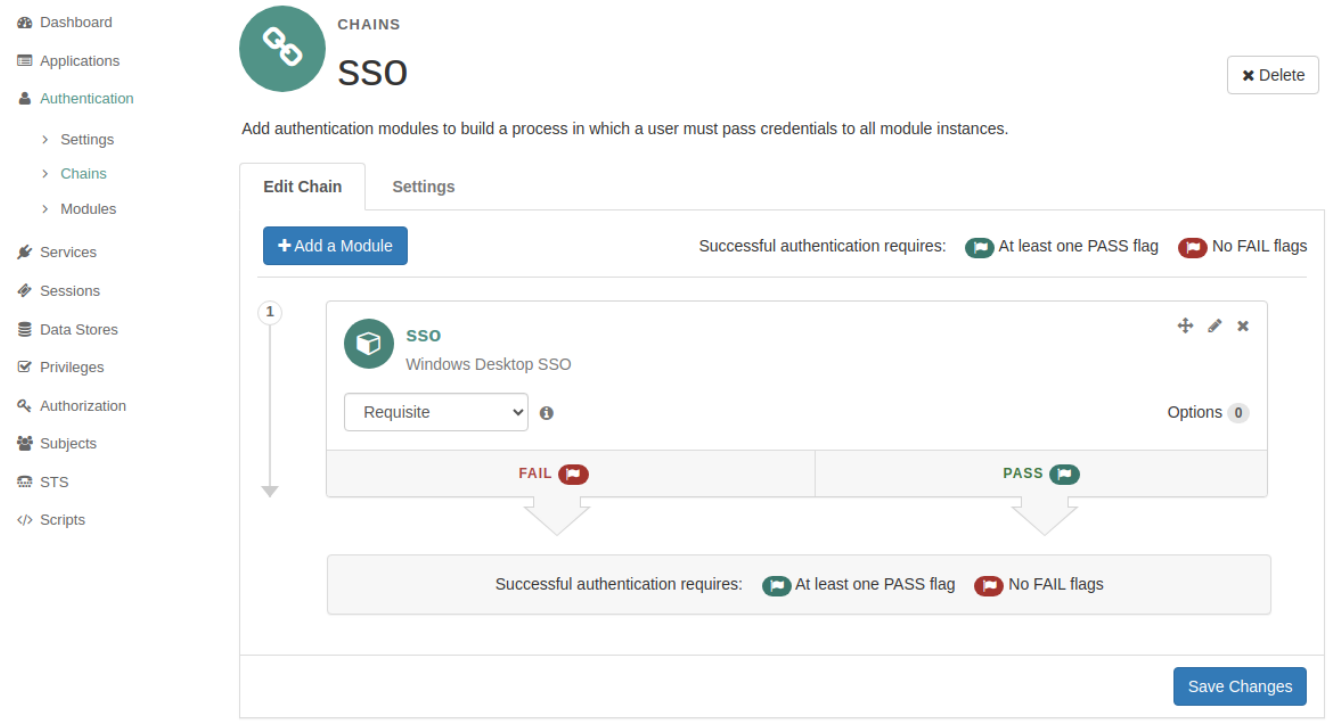
Edit the created `sso` authentication module settings.

Set `Service Principal`, as in `ktpass` command. `Keytab File Name` should be `openamKerberos.keytab` file location on the OpenAM server. Set `Kerberos Realm`, `Kerberos Server Name`, and `Trusted Kerberos realms` according to your environment.



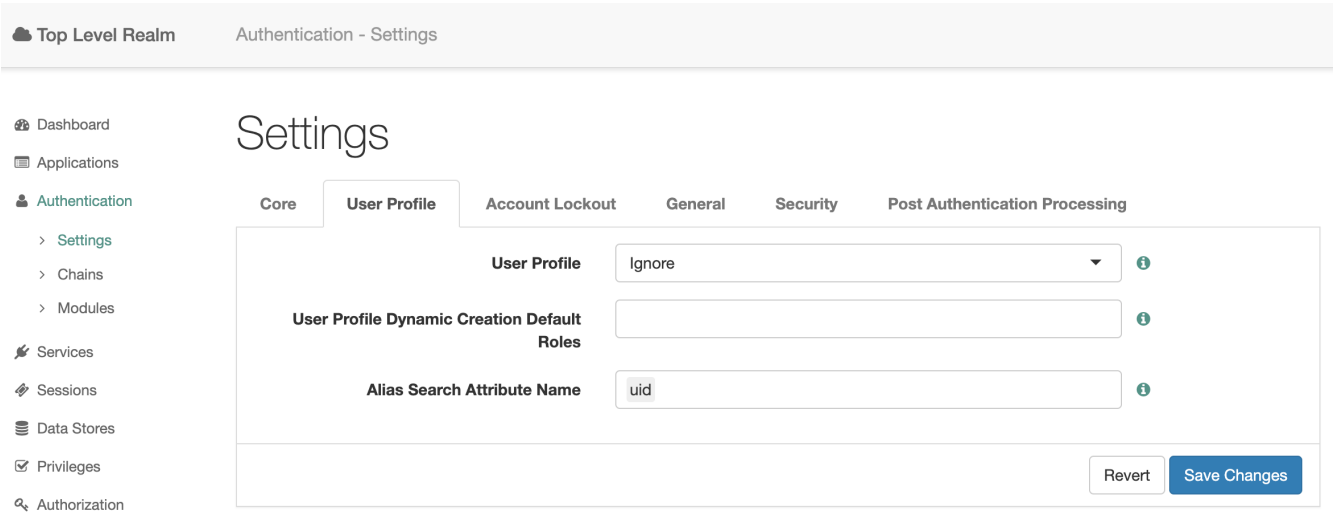
Setup Authentication Chain

Go to the admin console, select the root realm, and select Authentication → Chains from the menu. Create an `sso` authentication chain with the recently created `sso` module.



Realm Configuration

Go to Authentication → Chains for realm and on the User Profile tab, set the `User Profile` setting to `Ignore`.



So, you can authenticate with Kerberos without setting up Active Directory as a User Data Store in OpenAM.

Test Solution

On a Windows machine, authenticate with your Active Directory account and go to http://openam.example.com:8080/openam/XUI/#login/&realm=/&service=sso

You should be seamlessly authenticated with an Active Directory account without prompting credentials.

Login and Password Authentication

- Introduction
- Implementation
- Pros and Cons
- Conclusion

Introduction

The following article is intended for newbies, who want to develop authentication for their services. In the article, we will look at the most common authentication method - login and password authentication, its implementation, advantages, and disadvantages.

Authentication is an identity verification process. For software, authentication is used for verifying the identities of users or client applications. The most common way to authenticate users is login and password authentication.

User login could be public but the password should be only in the user's memory (and not on a piece of paper under the keyboard or taped to the monitor!) and used for verification than login belongs to the only user who knows the password.

Implementation

User identity data usually stored in a user database. And the only authentication system should have access to the database to minimize the risk of credentials leak.

Login in user database stored in plaintext to quick search for the user account. Password should be stored as its hash and never in plaintext. During authentication, the hash of the password entered by the user is calculated, compared with the value stored in the database and, if the values match, authentication is successful.

Password Hashing

To store a password in a database relatively secure, not the password itself is stored in the database, but password hash. The hash is calculated by the formula: `password_hash = hash(password)`. To slow-down dictionary password attack, it is necessary to add so-called **salt** to password. **Salt** - is a random value, stored within the password hash. And password hash is calculated with function from password itself and hash `password_hash = hash(password, salt)`. More information about salt in Wikipedia: Salt_(cryptography).

Database Authentication

Some databases support internal authentication. In this case, the database itself is responsible for login and password authentication. Authentication service sends authentication request with login and password to the database, and the database returns authentication result: whether it is successful or not. For example, such an approach uses Microsoft Active Directory.

Implementation Tips

- If authentication was failed, do not tell users whether login exists or not
- Implement user account lock policy to prevent possible password brute force
- Use strong hashing function for example [bcrypt](https://en.wikipedia.org/wiki/Bcrypt), [scrypt](https://en.wikipedia.org/wiki/Scrypt) or `[PBKDF2](https://en.wikipedia.org/wiki/PBKDF2)` to prevent hash passwords reverse engineering in case user credentials database was stolen
- Use salt when hashing a password

Pros and Cons

Pros:

- relatively simple implementation

Cons:

- the need for users to remember a password for each service.
- the need to implement additional service to recover/change password
- the security policy of some organizations requires periodic password changes, which impairs user experience
- the password could be stolen using fishing or social engineering

Conclusion

Login and password authentication is the most common way to authenticate users, but it is also one of the most insecure, so it is recommended to use authentication without a password (login through social networks, biometrics, use of hardware security keys), or strengthen security by using the second authentication factor.

Login via Gosuslugi OAuth2 Identity Provider

Preparation

Register an Organization on the Gosuslugi Portal

In order to use authentication through the ESIA, you need to register your organization on the Gosuslugi portal. To do this, you need to have a confirmed account of the head of the organization on the portal.

Next, you need to register your information system on the ESIA technology portal. To do this, go to the link https://esia.gosuslugi.ru/console/tech and authenticate with the account of the head of the organization. On the "Информационные системы" tab, click "Добавить систему" and fill in the properties on the link. The main thing to pay attention to is the system mnemonic, i.e. client_id, system URL - a list of allowed URLs to which redirection will be performed after successful or unsuccessful authentication in ESIA, as well as the algorithm of electronic signature formation - must be GOST3410_2012_256.

The process is described in more detail in document https://digital.gov.ru/ru/documents/6190/ section 3.1.1

Send System Registration Application to the Ministry of Digitization

After you have registered an information system, you must obtain a certificate from an accredited certification center (List of accredited certification centers at https://digital.gov.ru/ru/activity/govservices/certification_authority/).

The algorithm of electronic signature formation is GOST R 34.10-2012. The certificate must contain the OGRN of the legal entity that is the operator of the information system and can be issued for the employee of the legal entity or for the organization.

Next, you need to fill an application to gain access to the test environment as described in https://digital.gov.ru/ru/documents/4244/ paragraph 9 and send it to the e-mail address of the Ministry of Digitization.

After your system is registered in the test environment, you can proceed to OpenAM configuration.

OpenAM Installation

If you have already installed OpenAM, you can skip this section. For demonstration purposes, we will configure OpenAM in a Docker container.

Network Setup

Add the OpenAM hostname and IP address to the `hosts` file

```
127.0.0.1 openam.example.org
```

On Windows systems, the `hosts` file is located at `C:Windows/System32/drivers/etc/hosts`; on Linux and Mac it is located at `/etc/hosts`.

Install OpenAM in Docker

Start the Docker image of OpenAM by mounting the key and certificate issued earlier to the container.

```
docker run -h openam.example.org -p 8080:8080 --name openam \
 -v ./openam-esia.key:/usr/openam/esia/openam-esia.key:ro \.
 -v ./openam-esia.pem:/usr/openam/esia/openam-esia.pem:ro
openidentityplatform/openam:latest
```

Once the OpenAM server is running, perform the initial configuration by running the following command and wait for the configuration to complete.

```
docker exec -w '/usr/openam/ssoconfiguratortools' openam bash -c \
'echo "ACCEPT_LICENSES=true
SERVER_URL=http://openam.example.org:8080
DEPLOYMENT_URI=/$OPENAM_PATH
BASE_DIR=$OPENAM_DATA_DIR
locale=en_US
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=passw0rd
AMLDAPUSERPASSWD=p@passw0rd
COOKIE_DOMAIN=openam.example.org
ACCEPT_LICENSES=true
DATA_STORE=embedded
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=openam.example.org
DIRECTORY_PORT=50389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1689
ROOT_SUFFIX=dc=openam,dc=example,dc=org
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=passw0rd" > conf.file && java -jar openam-configurator-tool*.jar
--file conf.file'
```

OpenAM Configuration

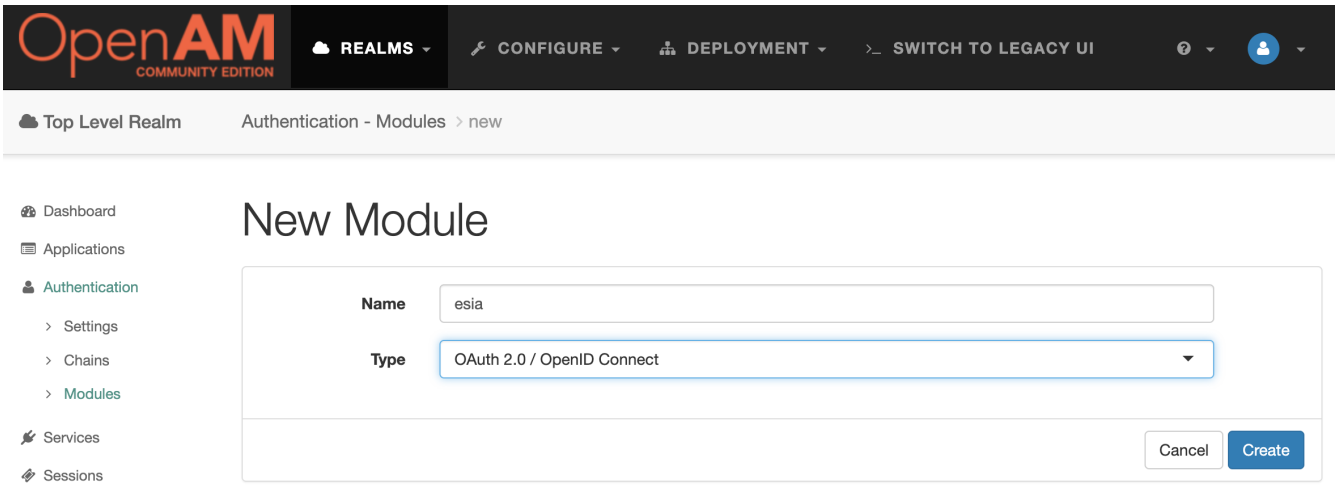
Setup ESIA Authentication Module

Open the OpenAM administrator console at http://openam.example.org:8080/openam. In the login

field enter the `amadmin` value, in the password field enter the value specified in the `ADMIN_PWD` setting, in this case `passw0rd`.

In the console, select the root realm and then select Authentication → Modules from the left menu. In the opened list, click the `Add Module` button.

Enter the module name, for example, `esia` and the module type - `OAuth 2.0 / OpenID Connect`.



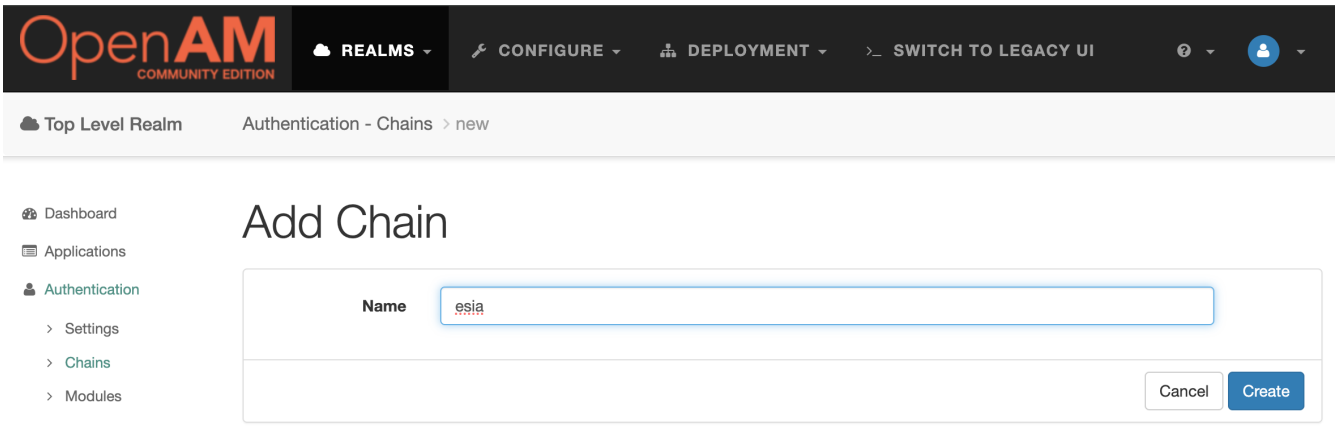
Press the `Create` button . Fill in the settings according to the table:

Setting	Value
Client Id	Mnemonic of the information system specified in the ESIA technology portal
Client Secret	Not to be filled in, client secret is calculated dynamically for ESIA
Authentication Endpoint URL	https://esia-portal1.test.gosuslugi.ru/aas/oauth2/ac
Access Token Endpoint URL	https://esia-portal1.test.gosuslugi.ru/aas/oauth2/te
User Profile Service URL	https://esia-portal1.test.gosuslugi.ru/rs/prns
Scope	Scope of the profile, for demonstration purposes we will specify fullname birthdate gender. You can read more about the scope description in the document at https://digital.gov.ru/ru/documents/6186/
OAuth2 Access Token Profile Service Parameter name	access_token
Proxy URL	http://openam.example.org:8080/openam/oauth2c/OAuthProxy.jsp
Account Provider	org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider

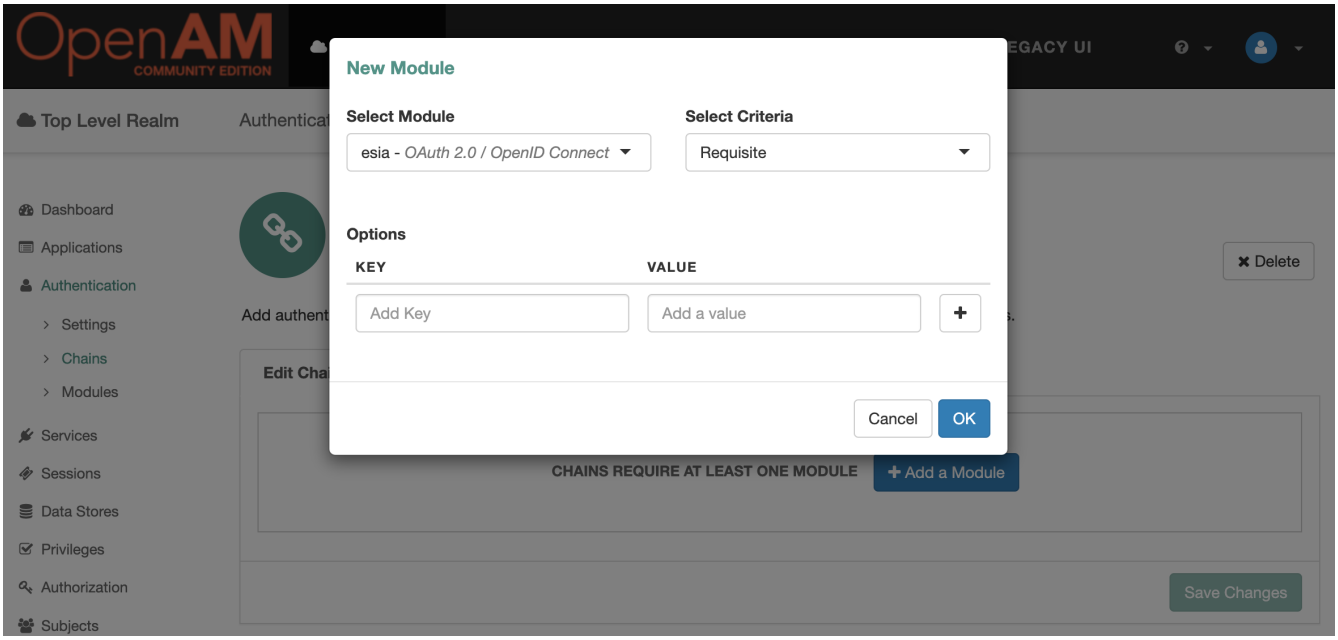
Setting	Value
Account Mapper	org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper
Account Mapper Configuration	oid=uid
Attribute Mapper	org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper
Attribute Mapper Configuration	oid=uid, remote-json=sn
Custom Properties	
esia-key-path	path to the private key file, e.g. /usr/openam/esia/openam-esia.key
esia-cert-path	the path to the certificate file, e.g. /usr/openam/esia/openam-esia.pem
esia-org-scope	organization scope if you want to get information about the organization of the ESIA user
esia-org-info-url	URL to retrieve organization information, e.g. https://esia-portal1.test.gosuslugi.ru/rs/orgs/ - for test ESIA

Configuring the ESIA Authentication Chain

Open the OpenAM admin console, select realm and from the left menu select `Authentication → Chains` . Click the `Add Chain` button and enter a name for the chain, for example `esia` . Click the `Create` button.



Press the `Add a Module` button.



Add module `esia` , `Criteria` set to `Requisite`. Press the `OK button .

Test the solution

Logout the OpenAM admin console, and open the ESUA authentication chain link: http://openam.example.org:8080/openam/XUI/?service=esia. If everything is configured correctly, you will be redirected to the ESIA authentication portal. Enter the user's credentials. After successful authentication to the ESIA, the ESIA will request consent to access the data passed in the scope parameter. After accepting the consent, you will be redirected to the OpenAM console with the user's ESIA credentials.

Login via Facebook OAuth2 Identity Provider

Create Facebook Applicaton

Goto https://developers.facebook.com/apps/ and create new Facebook Application Goto Application Dashboard. Then goto **Settings -> Basic**. There will be your app App Id and App Secret.

Setup OpenAM

Legacy UI

Login into console. Goto **Access Control** then select target realm. Goto **Authentication**

Create Authentication Module

Under section **Module Instances** create new **Authentication Module**. Enter new module instance name, for example **facebook**. Authentication module type is **OAuth 2.0 / OpenID Connect**

Then select module, you've just created from module list and enter following settings:

Setting	Value
Client Id	Your Facebook Application App Id
Client Secret	Your Facebook Application App Secret
Authentication Endpoint URL	https://www.facebook.com/dialog/oauth
Access Token Endpoint URL	https://graph.facebook.com/oauth/access_token
User Profile Service URL	https://graph.facebook.com/me
Scope	*Here you should enter scope, according to Facebook documentation* for example **email**
OAuth2 Access Token Profile Service Parameter name	access_token
Proxy URL	*[Your OpenAM URL]/oauth2c/OAuthProxy.jsp* for example: https://openam.example.com/openam/oauth2c/OAuthProxy.jsp
Account Provider	org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider
Account Mapper	org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper

Setting	Value
Account Mapper Configuration	Attribute configuration that will be used to map the account of the user authenticated in the OAuth 2.0 Provider to the local data store in the OpenAM. Example: id=facebook-id email=mail
Attribute Mapper	org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper
Attribute Mapper Configuration	Attribute configuration that will be used to map the user info obtained from the OAuth 2.0 Provider to the local user data store in the OpenAM. Example: last_name=snlast_name=facebook-lnameemail=mailfirst_name=givennamefirst_name=facebook-fnamename=cnemail=facebook-emailid=facebook-id

You can setup remaining attributes on your own, depending your authentication process requirement and press **Save** and then **Back to Authentication**

Create Authentication Chain

Under section **Authentication Chaining** create new Authentication Chain, enter its name, for example, **facebook** and add recently created module **facebook**

Your authentication chain should look like this:

Instance	Criteria	Options
facebook	Required	

Test your Authentication Chain

Goto [Your OpenAM URL]/UI/Login?org=[your org]&service=[facebook auth chain], for example, http://example.openam.com/openam/UI/Login?org=/&service=facebook and you should see facebook authentication dialog

Login via Odnoklassniki OAuth2 Identity Provider

Create Odnoklassniki Applicaton

Create Odnoklassniki application as described here https://apiok.ru/dev/app/create. After you create application you'll receive: Application ID, Application Key and Aplication Secret.

Setup OpenAM

Legacy UI

Login into console. Goto **Access Control** then select target realm. Goto **Authentication**

Create Authentication Module

Under section **Module Instances** create new **Authentication Module**. Enter new module instance name, for example **odnoklassniki**. Authentication module type is **OAuth 2.0 / OpenID Connect**

Then select module, you've just created from module list and enter following settings:

Setting	**Value**
Client Id	Your Odnoklassniki Application Id
Client Secret	Your Odnoklassniki Application Secret
Authentication Endpoint URL	https://connect.ok.ru/oauth/authorize
Access Token Endpoint URL	https://api.ok.ru/oauth/token.do
User Profile Service URL	https://api.ok.ru/api/users/getCurrentUser
Scope	*Here you should enter scope, according to Odnoklassniki documentation* for example **VALUABLE_ACCESS;LONG_ACCESS_TOKEN**
OAuth2 Access Token Profile Service Parameter name	access_token
Proxy URL	*[Your OpenAM URL]/oauth2c/OAuthProxy.jsp* for example: https://openam.example.com/openam/oauth2c/OAuthProxy.jsp
Account Provider	org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider
Account Mapper	org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper

Setting	Value
Account Mapper Configuration	Attribute configuration that will be used to map the account of the user authenticated in the OAuth 2.0 Provider to the local data store in the OpenAM. Example: uid=uid
Attribute Mapper	org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper
Attribute Mapper Configuration	Attribute configuration that will be used to map the user info obtained from the OAuth 2.0 Provider to the local user data store in the OpenAM. Example: first_name=cnlast_name=givenName
Custom Properties	Add Oddnoklassniki Application Key as **ok-public-key** property, for examplpe **ok-public-key=AAAAAAAAAAAAA**

You can setup remaining attributes on your own, depending your authentication process requirement and press **Save** and then **Back to Authentication**

Create Authentication Chain

Under section **Authentication Chaining** create new Authentication Chain, enter its name, for example, **odnoklassniki** and add recently created module **odnoklassniki**

Your authentication chain should look like this:

Instance	Criteria	Options
odnoklassniki	Required	

Test your Authentication Chain

Goto [Your OpenAM URL]/UI/Login?org=[your org]&service=[ok auth chain], for example, http://example.openam.com/openam/UI/Login?org=/&service=odnoklassniki and you should see Odnoklassniki authentication dialog

Login via VK OAuth2 Identity Provider

Create Applicaton

Goto https://vk.com/apps?act=manage and create new VKontakte Application Goto Application Setting. There will be your app App Id and App Secret.

Setup OpenAM

Legacy UI

Login into console. Goto **Access Control** then select target realm. Goto **Authentication**

Create Authentication Module

Under section **Module Instances** create new **Authentication Module**. Enter new module instance name, for example **vkontakte**. Authentication module type is **OAuth 2.0 / OpenID Connect**

Then select module, you've just created from module list and enter following settings:

Setting	Value
Client Id	Your VKontakte Application App Id
Client Secret	Your VKontakte Application App Secret
Authentication Endpoint URL	https://oauth.vk.com/authorize
Access Token Endpoint URL	https://oauth.vk.com/access_token
User Profile Service URL	https://api.vk.com/method/users.get
Scope	*Here you should enter scope, according to VKontakte documentation* for example **email**
OAuth2 Access Token Profile Service Parameter name	access_token
Proxy URL	*[Your OpenAM URL]/oauth2c/OAuthProxy.jsp* for example: https://openam.example.com/openam/oauth2c/OAuthProxy.jsp
Account Provider	org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider
Account Mapper	org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper
Account Mapper Configuration	Attribute configuration that will be used to map the account of the user authenticated in the OAuth 2.0 Provider to the local data store in the OpenAM. Example: id=uid

Setting	Value
Attribute Mapper	org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper
Attribute Mapper Configuration	Attribute configuration that will be used to map the user info obtained from the OAuth 2.0 Provider to the local user data store in the OpenAM. Example: last_name=givenName id=uid

You can setup remaining attributes on your own, depending your authentication process requirement and press **Save** and then **Back to Authentication**

Create Authentication Chain

Under section **Authentication Chaining** create new Authentication Chain, enter its name, for example, **vkontakte** and add recently created module **vkontakte**

Your authentication chain should look like this:

Instance	Criteria	Options
vkontakte	Required	

Test your Authentication Chain

Goto [Your OpenAM URL]/UI/Login?org=[your org]&service=[facebook auth chain], for example, http://example.openam.com/openam/UI/Login?org=/&service=vkontakte and you should see facebook authentication dialog

Migrate OpenAM to Apache Cassandra without Single Point of Failure

Initial Data Storage Scheme

Data Type	Storage Method	Fautl Tolerance Method	Disadvantages
OpenAM configuration	OpenDJ (localhost:1389)	Multi-master replication	Configuration update on a single node affects all nodes by replication.
CTS-Core Token Service (Session persistence)			Syncronisation payload processed on all nodes Read performance from single node restricted by single node performance. Replication failure could cause other nodes to read-only mode
Accounts repository (except AD)			

Data Storage Scheme for the Number of Credentials >5 Million

Data Type	Storage Method	Fault Tolerance Method	Disadvantages
OpenAM configuration	OpenDJ (localhost:1389)	Local independent storage as a part of the distribution (war file)	Updating the configuration on one node does not affect other nodes (the nodes are completely independent)

CTS-Core Token Service (Session persistence)	Cassandra Cluster (tcp:9042)	Cluster without a single point of failure with geo-distribution and distribution by rack	Synchronization write payload is processed only on the nodes with the required replication level The reading load from one node is not limited to the performance of one node, but distributed according to the replication level. Node failure does cause replication stop according to the replication level
Accounts repository (except AD)			

Migration Plan

1. Cluster hardware resources planning
2. Deploy the cluster according to the required level of fault tolerance
3. Provide network access OpenAM-> tcp:9042
4. Migration stages (can be done independently):
    a. Switch "CTS - Core Token Service (sessions)"
    b. Switch "Accounts repository (except AD)" with legacy data migration
    c. Switch "OpenAM configuration"

Fault Tolerance Level Planning

Datacenter

Defines geo-distributed storage fault tolerance.

- Minimum number of data centers: 1
- The recommended number of data centers:
    - at least two
    - at least the same number of data centers used for application servers (OpenAM)
- Allowed data center fault tolerance mode:
    - Hot Spare: used for data processing for application servers (OpenAM)
    - Cold Spare: not used for data processing for application servers (OpenAM)

Rack

Minimum fault tolerance unit within a data center for data distribution within a data center, haves:

- Independent disk subsystem array
- Independent virtualization hypervisor (host system)

Amount calculation:

- Minimum quantity inside hot spare data center: 1, but not less than replication level inside the data center
- Minimum quantity inside cold spare datacenter: 1, but not less than replication level inside the datacenter

Node

Defines the unit of information storage and load inside the data center rack

An Example of a Recommended Minimum Configuration

Test

DataCenter	Type	Amount of Copies	Rack	% data	Node	% data
dc01	host	1	rack01	100%	dc01-rack01-node01	100%

Production

DataCenter	Type	Amount of Copies	Rack	data %	Node	data %
dc01	hot	1	rack01	100%	dc01-rack01-node01	50%
					dc01-rack01-node02	50%
		1	rack02	100%	dc01-rack02-node01	50%
					dc01-rack02-node02	50%
dc02	hot	1	rack01	100%	dc02-rack01-node01	50%
					dc02-rack01-node02	50%
		1	rack02	100%	dc02-rack02-node01	50%

					dc02-rack02-node02	50%

Allowed:

- Increase the data center amount without service interruption
- Increase the rack amount without service interruption
- Increase the node amount without service interruption
- Change the number of data copies inside the data center without service interruption

Hardware Requirements For a Single Node

Environment	CPU	RAM	Disk
Test	>=2	>=8	16G HDD
Production	>=8	>=32 ballon=off	64G SSD RAID

OpenAM Monitoring Using Prometheus

Preface

OpenAM can expose SNMP metrics. To expose these metrics to Prometheus we will use snmp_exporter tool, provided by the Prometheus community.

To follow the current manual Docker and Docker Compose should be installed

Configuration files located in this project https://github.com/maximthomas/openam-prometheus

Just clone it and run with docker-compose:

```
git clone https://github.com/maximthomas/openam-prometheus.git
```

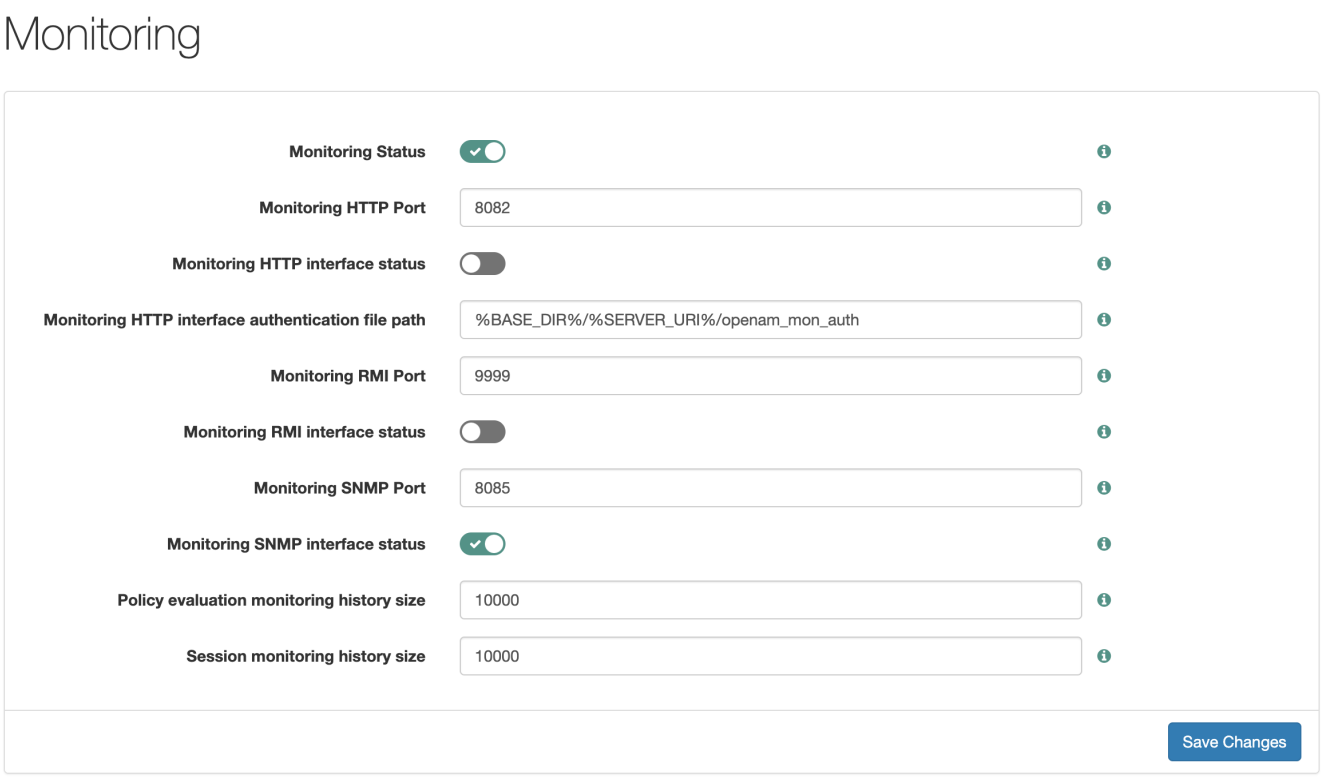
```
cd ./openam-prometheus
```

```
docker-compose up
```

Setup OpenAM and SNMP Monitoring

Open OpenAM URL and create a basic configuration (see OpenAM Quick Start Gude)

Enable SNMP in OpenAM. Login into OpenAM console, goto **Configure -> Global Services -> Monitoring**



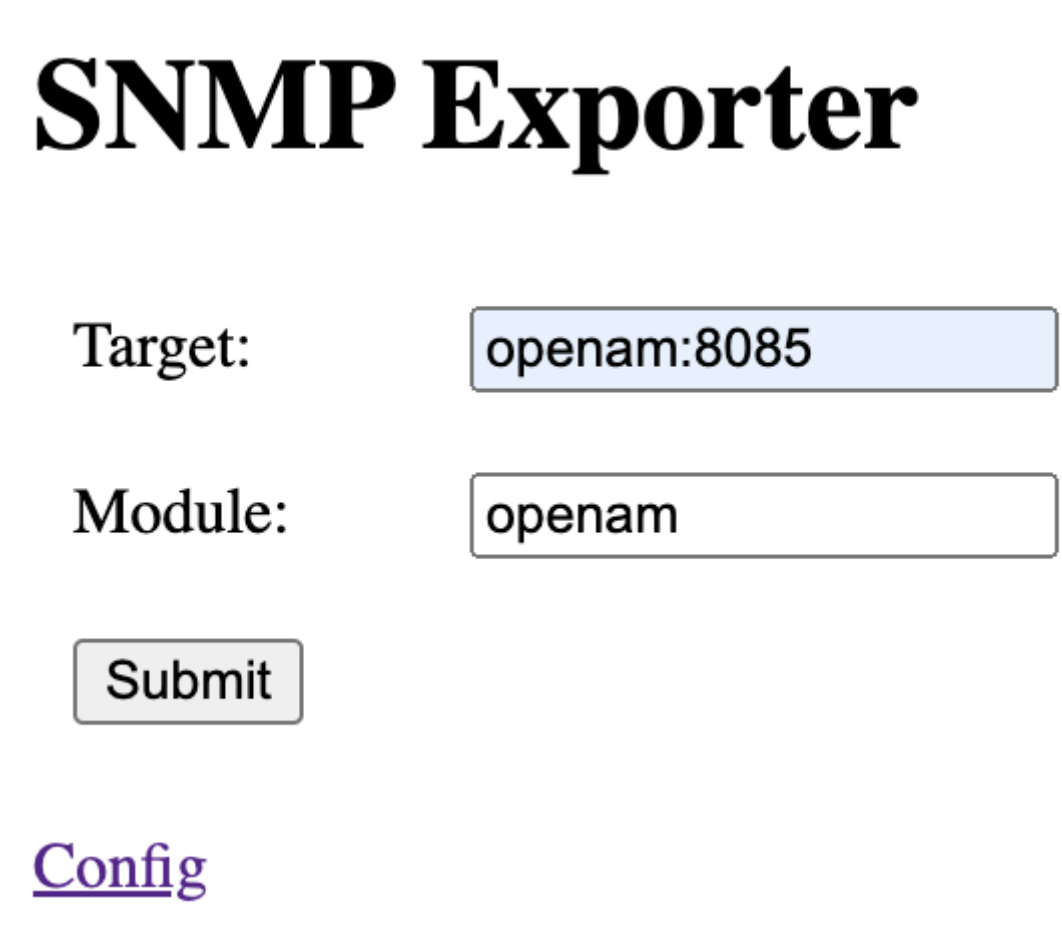
Enable checkboxes **Monitoring Status** and **Monitoring SNMP interface status**

Restart OpenAM instance

Setup SNMP Exporter

Open SNMP exporter URL

http://localhost:9116/



Fill the fields: Target: OpenAM hostname and port, for example `openam:8085` Module: SNMP exporter module, for example `openam`

Press the Submit, then you should see something like this:

```
# HELP authenticationFailureCount Count of unsuccessful authentications -
1.3.6.1.4.1.42.2.230.3.1.1.2.1.10.2
# TYPE authenticationFailureCount counter
authenticationFailureCount 0
# HELP authenticationFailureRate Current Auth failure rate / second based on a fixed
preconfigured window. - 1.3.6.1.4.1.42.2.230.3.1.1.2.1.10.4
# TYPE authenticationFailureRate gauge
authenticationFailureRate 0
# HELP authenticationSuccessCount Count of successful authentications -
1.3.6.1.4.1.42.2.230.3.1.1.2.1.10.1
# TYPE authenticationSuccessCount counter
authenticationSuccessCount 0
# HELP authenticationSuccessRate Current Auth success rate / second based on a fixed
preconfigured window. - 1.3.6.1.4.1.42.2.230.3.1.1.2.1.10.3
# TYPE authenticationSuccessRate gauge
authenticationSuccessRate 0
# HELP averageDuration Average duration of tokens of the specified type in OpenAM in
seconds - 1.3.6.1.4.1.36733.1.2.3.5.1.2
# TYPE averageDuration counter
averageDuration{tokenTableIndex="1"} 1532
averageDuration{tokenTableIndex="10"} 0
```

```
averageDuration{tokenTableIndex="11"} 0
averageDuration{tokenTableIndex="12"} 0
averageDuration{tokenTableIndex="13"} 0
averageDuration{tokenTableIndex="14"} 0
averageDuration{tokenTableIndex="15"} 0
averageDuration{tokenTableIndex="16"} 340
......
```

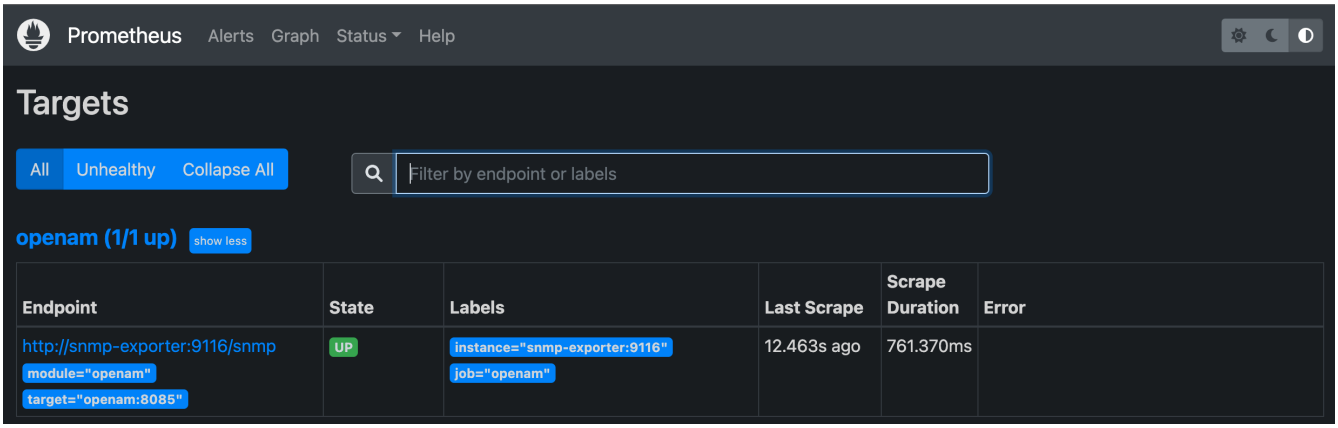
Setup Prometheus

If you have an existing prometheus instance add to `prometheus.yml` job `openam` with a target to SNMP exporter host and port:

```
  - job_name: "openam"
    params:
      target: ["openam:8085"]
      module: ["openam"]
    metrics_path: /snmp
    static_configs:
      - targets: ["snmp-exporter:9116"]
```

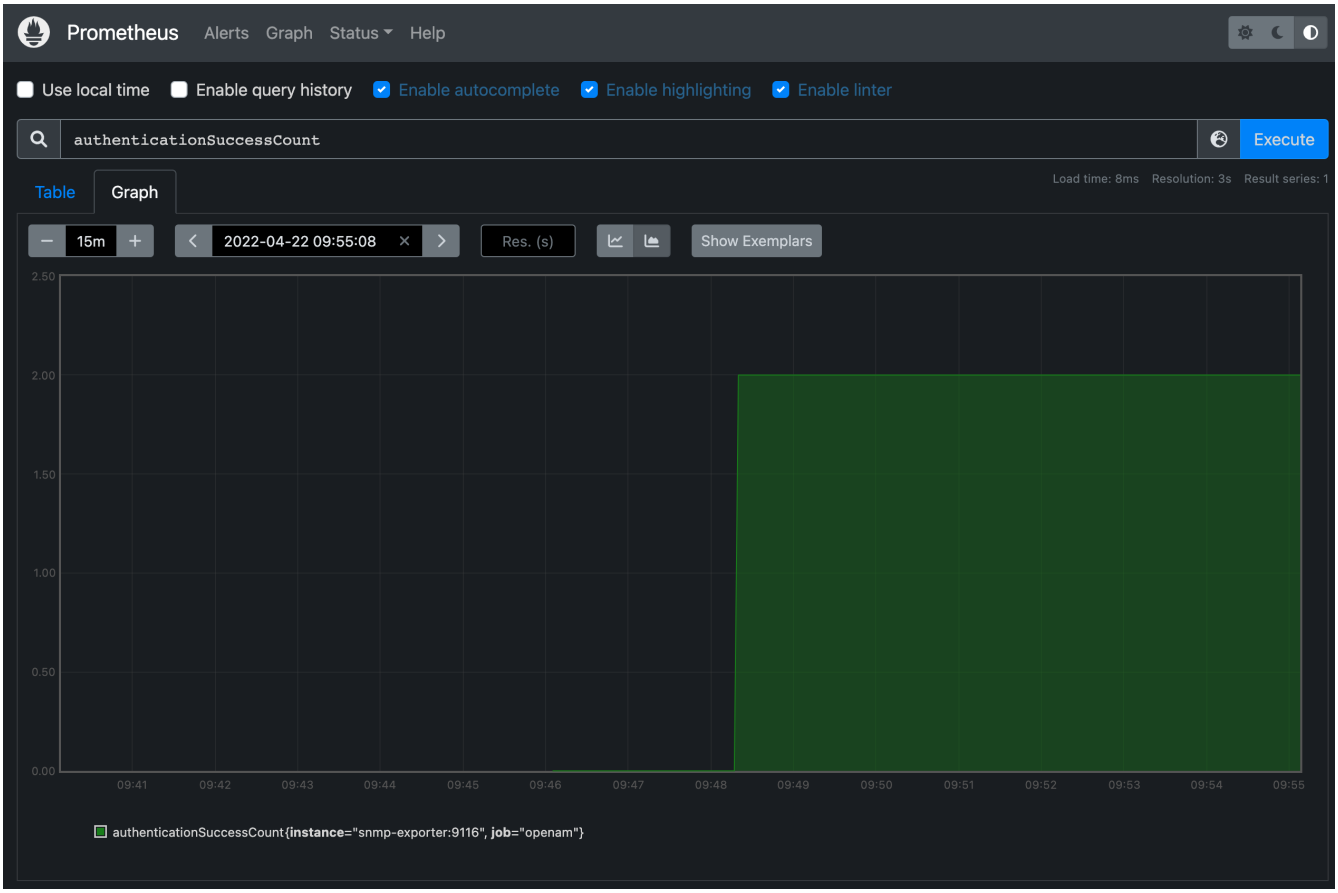
Run Prometheus and check targets:

http://localhost:9090/targets You should see something like this:



Then go to Graph and you will be able to add OpenAM metrics to monitor.

http://localhost:9090/graph



If you have any additional questions, feel free to ask us!

OpenAM QR Authentication Module

- Introduction
- Process Description
- OpenAM Configuration
- Mobile Application Configuration
- Testing Solution

Introduction

The most popular authentication method is also the most inconvenient. It is login and password authentication. A user must remember his login and password for each service, could forget the password, a password could be compromised, etc. The following article describes, how to login into the application in the browser on another device, if the user is already authenticated in a mobile application. The article contains step by step guide on how to set up the process in OpenAM.

Process Description

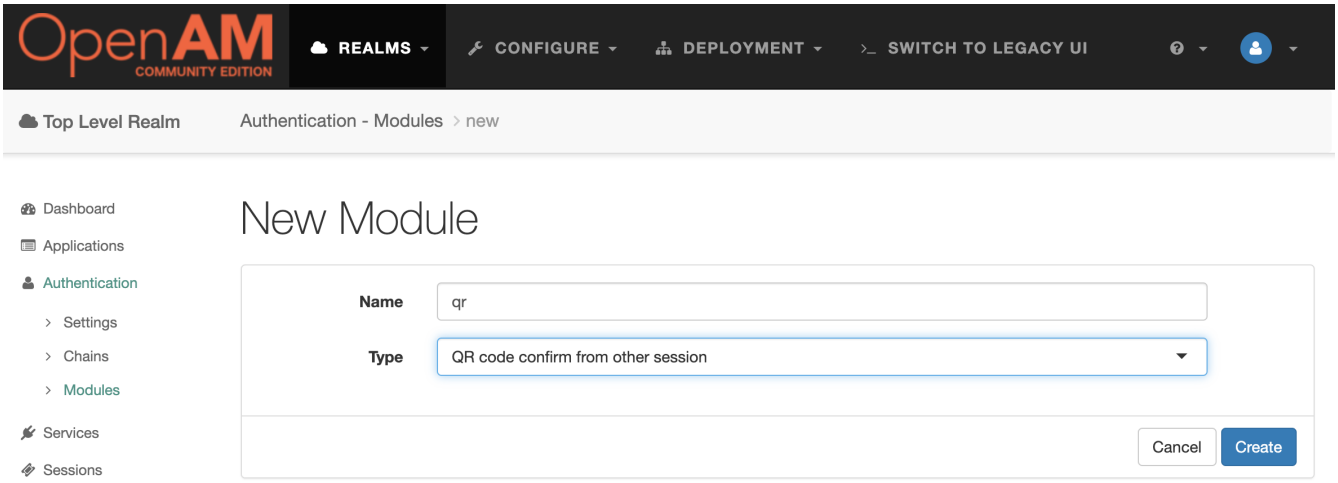
- The user has a valid authentication session in the mobile application.
- The user opens the OpenAM authentication page.
- OpenAM generates a random code with a limited lifetime and displays it to the user as a QR code.
- User scan the QR with the mobile application
- Mobile application sends the code to OpenAM within authenticated session identifier
- Meanwhile, the browser periodically polls OpenAM to see if the code is received along with the authentication session
- If the code was received, authentication is successful and an authentication session is created.

OpenAM Configuration

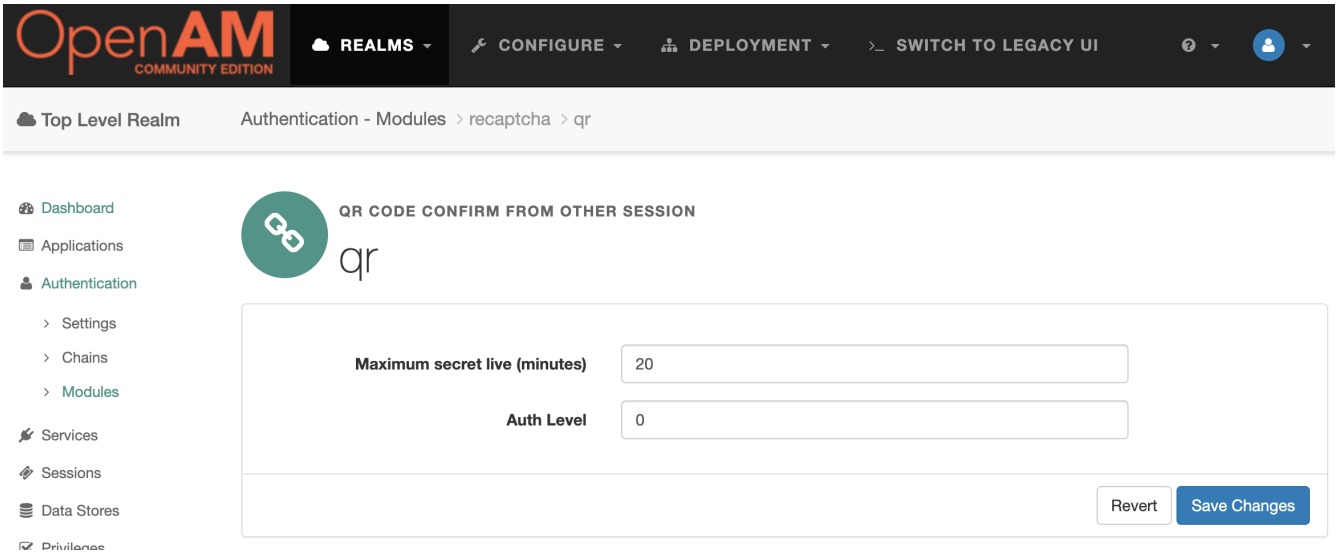
At first, we will set up OpenAM, create an authentication module instance and an authentication chain with the instance.

Create QR Authentication Module Instance

Go to the desired realm, then on the left side of the Authentication → Modules menu. The list of modules opens. Press the Add Module button. Enter the name of the module qr and choose QR code confirm from other session module type.



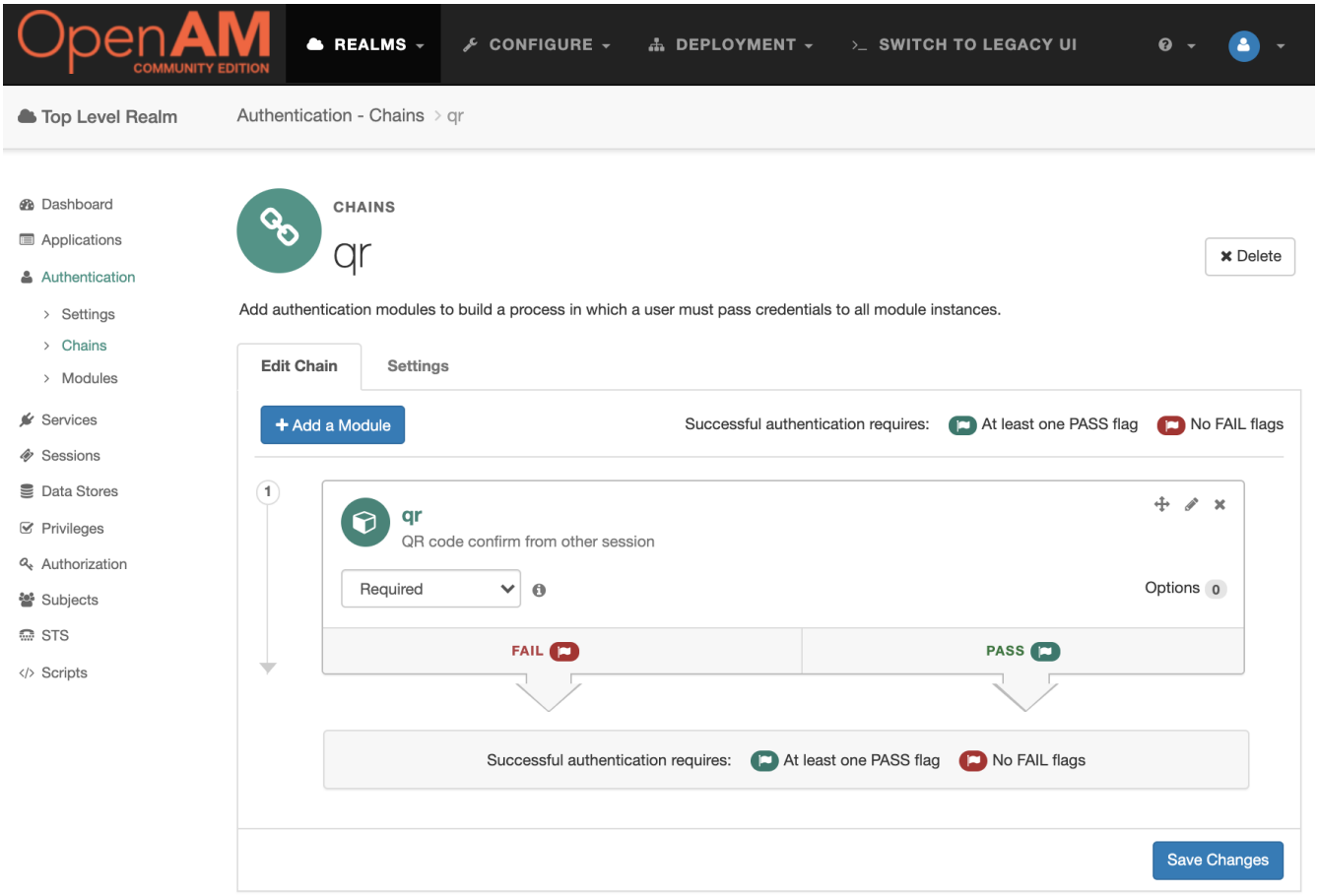
Press the Create button.



Set desired QR code lifetime and authentication level and press the Save Changes button.

Create Authentication Chain

Go to **Authentication** → **Chains**. And create a new **qr** authentication chain. Add the created **qr** module to the chain.



Mobile Application Configuration

Let's install and configure the mobile application. The following mobile application is only for demonstration and is not intended for production usage. But you can use application source code to build your application or modify application source code for your tasks. The application was developed with Flutter

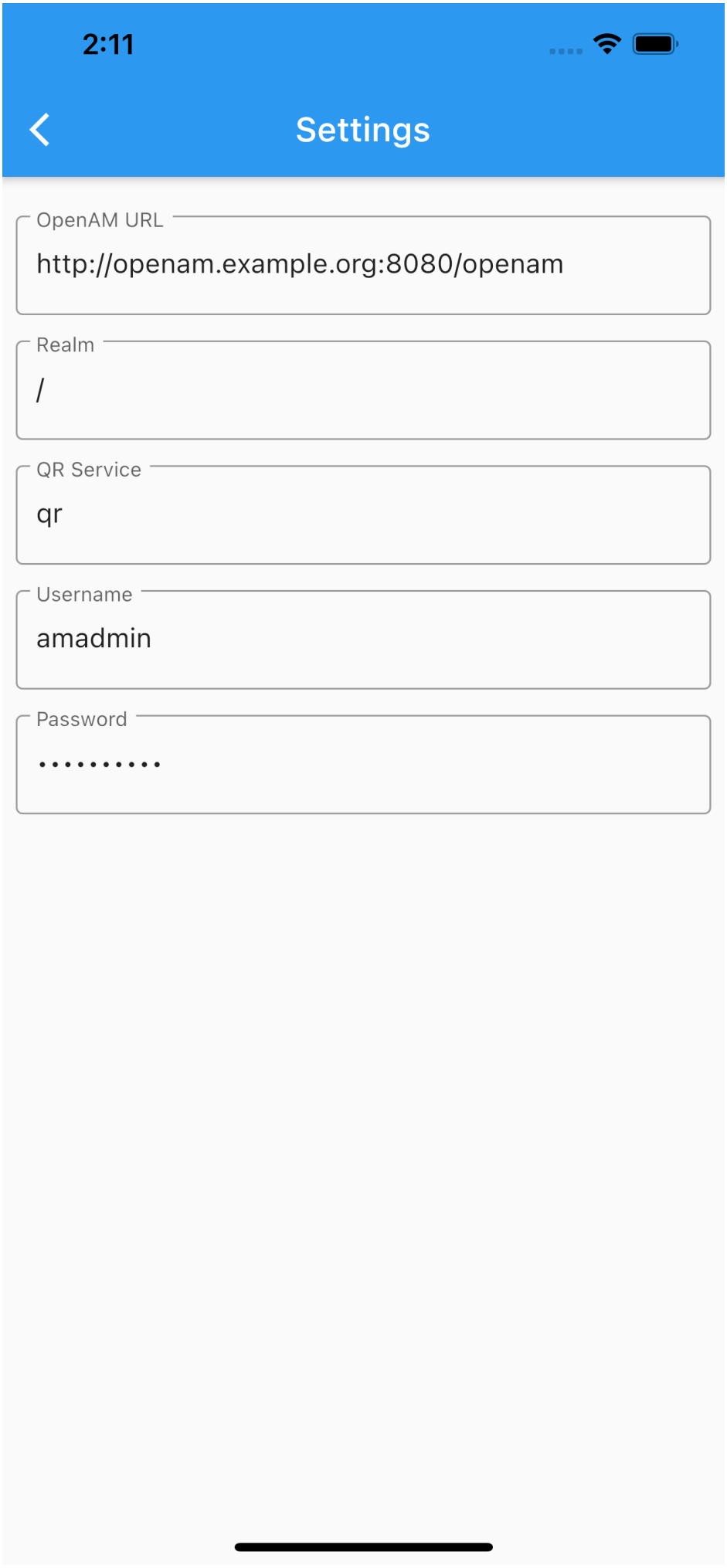
Mobile application source code: https://github.com/OpenIdentityPlatform/openam_demo_authenticator

Run the mobile application:

```
flutter run
```

Mobile Application Setup

Press the Settings icon in the upper-right corner of the application and go to the settings menu. Enter OpenAM URL, realm, OpenAM username and password. Press the Back button in the upper-left corner. Settings data will be stored in the device's secure storage.


2:11
Settings
OpenAM URL
http://openam.example.org:8080/openam
Realm
/
QR Service
qr
Username
amadmin
Password
••••••••••

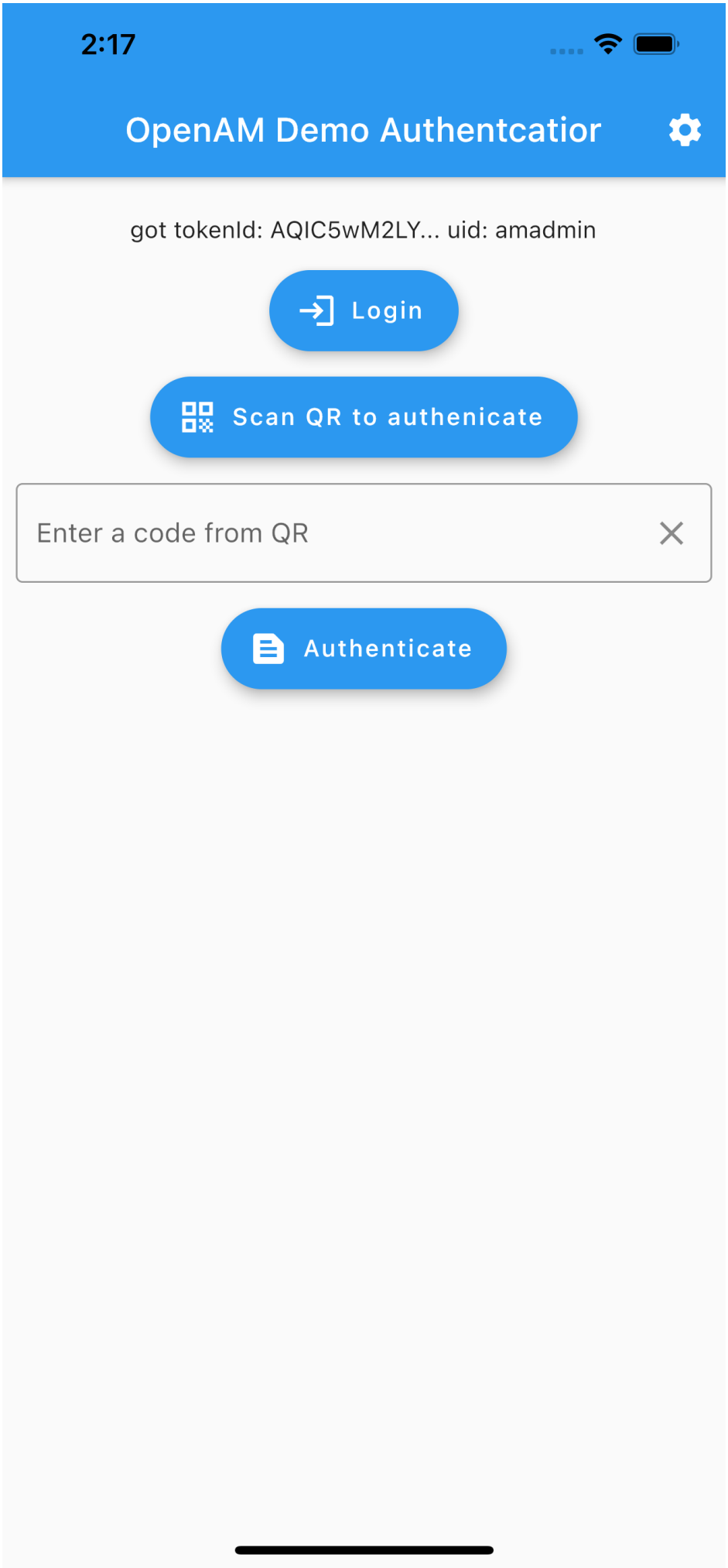
Testing Solution

If you were logged in, log out from OpenAM. Open OpenAM URL with QR authentication chain. For example, http://openam.example.org:8080/openam/XUI/#login/&realm=/&service=qr Replace http://openam.example.org:8080/openam with your OpenAM URL. You will see a QR code that you need to scan. As well as the same code in the form of text as shown in the image below





Open the mobile application, press the Login button. The application will authenticate in OpenAM in the specified realm with specified credentials. If authentication succeeds, you'll see the user name and first symbols of the OpenAM authentication token


2:17
OpenAM Demo Authentcatior
got tokenId: AQIC5wM2LY... uid: amadmin
Login
Scan QR to authenicate
Enter a code from QR
Authenticate

Then press the "Scan QR to authenticate" button, scan the QR code and you will a message about successful authentication. Meanwhile, the browser will authenticate the scanned QR code. If it is not possible to scan a QR code, you can enter the code manually in the "Enter a code from QR" field and press Authenticate. If you entered the code correctly, you will also be authenticated.

OpenAM and Spring Boot 3 Integration via OIDC OAuth2 Protocol

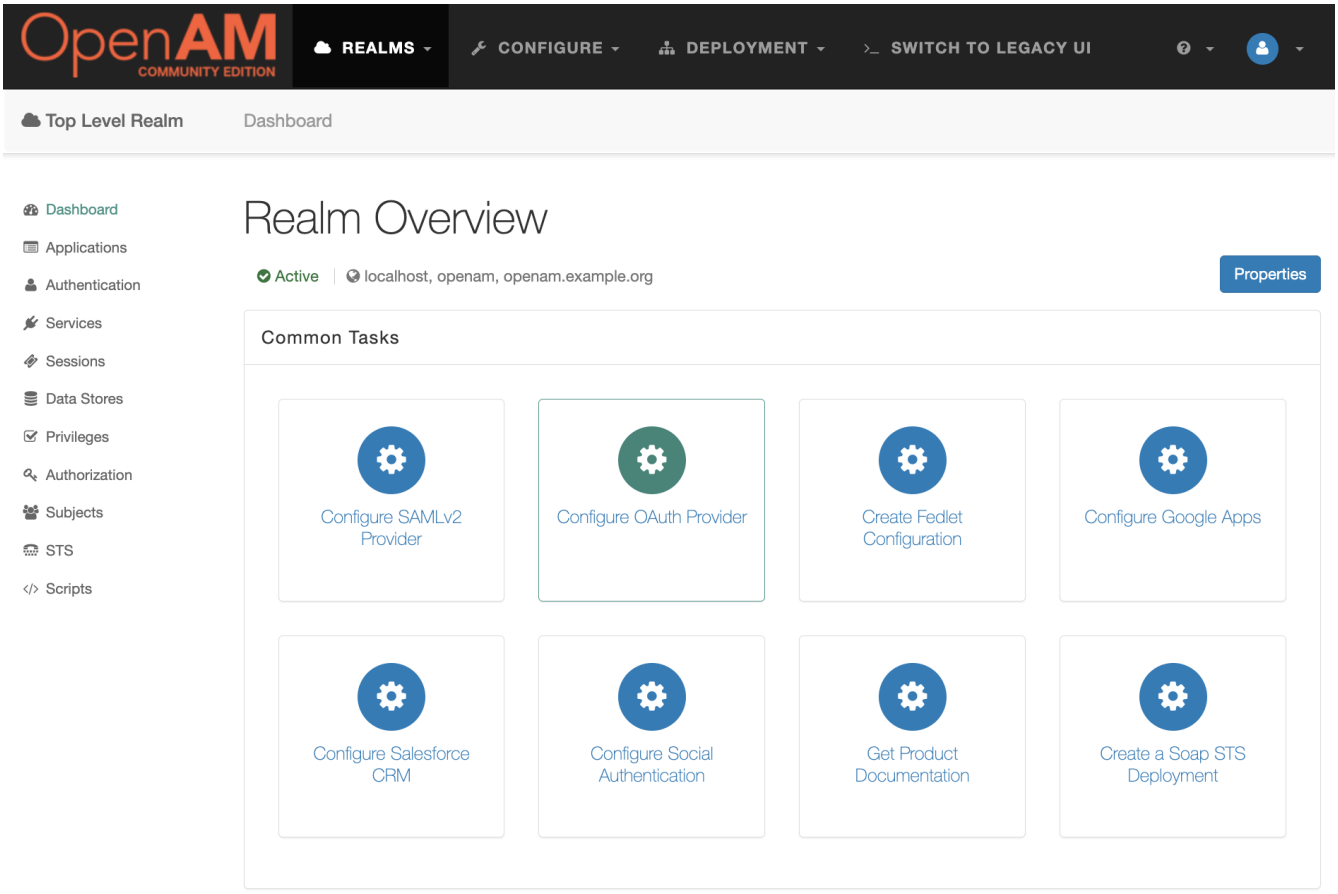
Introduction

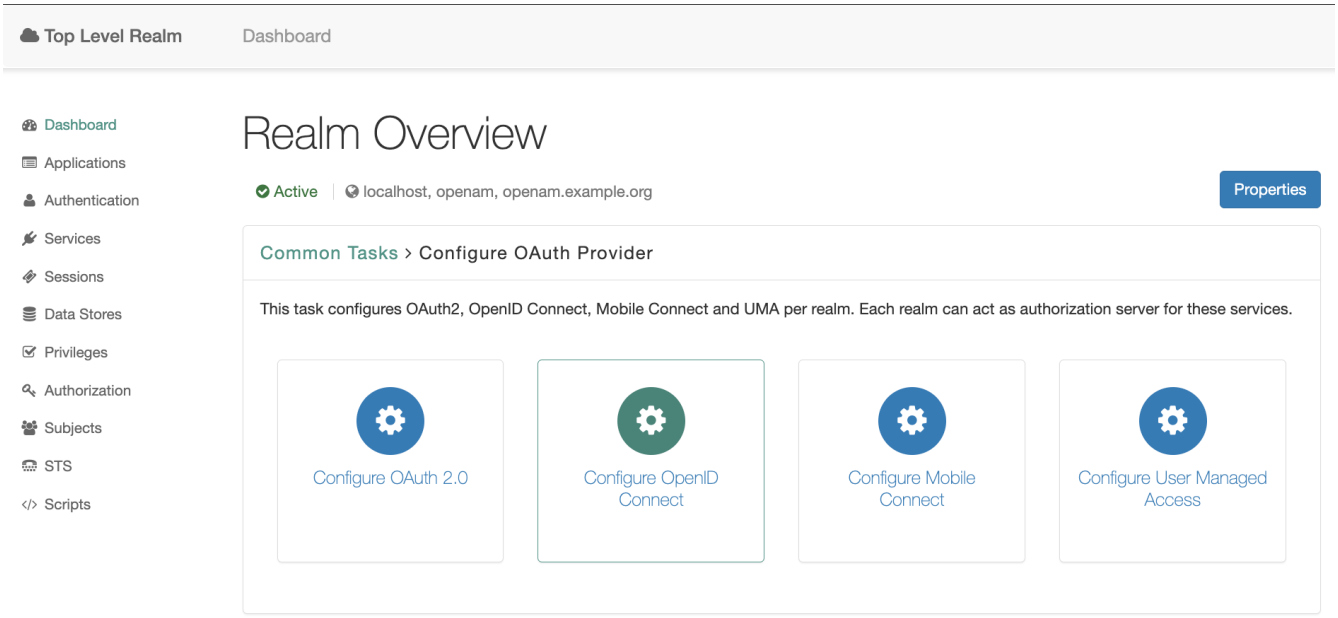
There are several ways to integrate Spring application with OpenAM. In the following tutorial, we will integrate Spring Application with OpenAM using the OIDC/OAuth2 protocol.

OpenAM Configuration

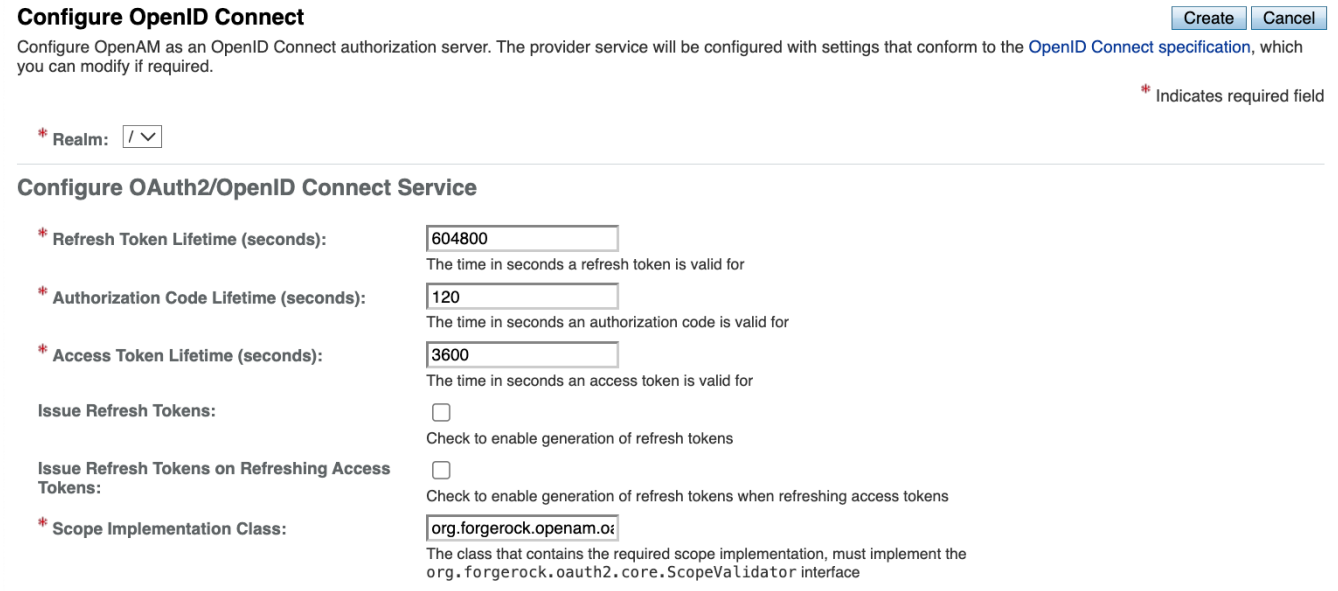
If you have not installed OpenAM yet, you could run OpenAM as a Docker image. Let's assume the OpenAM instance URL is `http://openam.example.org:8080/openam`

Go to the target realm and in the Common Tasks section. Open OpenAM console in a browser and create OAuth2 Provider. Press the `Configure OAuth Provider` button, and then the `Configure OpenID Connect` button.

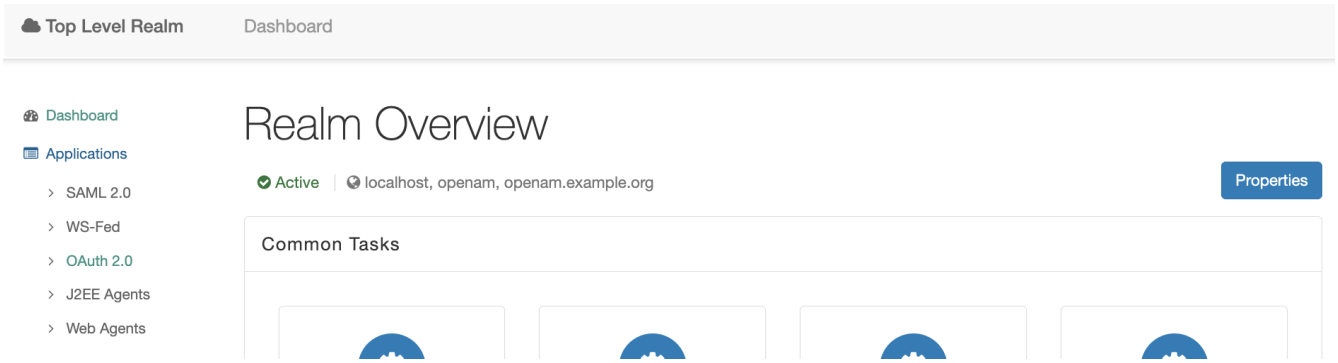




Ajust the required setting and press the Create button.



Go back to the realm and select in the realm's dashboard Applications → OAuth 2.0 in the right menu.



Create a new Agent, set name (client id) and password (client secret) and press the Next button. Go to the new created agent and set the following settings

- Add `http://localhost:8081/login/oauth2/code/openam` to **Redirection URIs** values (is will be our Spring Security application code endpoint)
- Add `profile` and `openid` values to **Scope(s)**
- Set **ID Token Signing Algorithm** setting to RS256.

Spring Boot Application

Create a new Spring Boot application and add the following Maven dependencies.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!--security dependencies-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

Create a controller and two endpoints: `index` and `protected-oauth`. The `index` endpoint will be accessible for anyone and `protected-oauth` endpoint will be accessible for authenticated with OpenAM users via OIDC protocol.

```
@Controller
public class SampleController {

    @GetMapping
    public String index() {
        return "index";
    }

    @GetMapping("/oauth")
    public String oauthProtected(HttpServletRequest request, Model model) {
        Principal token = request.getUserPrincipal();
        model.addAttribute("userName", token.getName());
        model.addAttribute("method", "OAuth2/OIDC");
        return "protected";
    }
}
```

Create the following templates for controllers:

`index.html`

```
<!DOCTYPE html>
<html>
<body>
<h1>OpenAM Spring Security Integration</h1>
<h2>Test Authentication</h2>
<ul>
    <li><a href="/protected-oauth">OAuth2/OIDC</a></li>
</ul>
</body>
</html>
```

`protected.html`

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<body>
<h1>Protected resource</h1>
<a href="/">Back</a></li>
<p><span th:text="${userName}"/> user authenticated with <span th:text=
"${method}"/></p>
</body>
</html>
```

Create Spring Security configuration

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration {
    @Bean
    public SecurityFilterChain securityWebFilterChain(HttpSecurity http) throws
Exception {
        http.authorizeHttpRequests((authorize) -> authorize.requestMatchers("/",
"/oauth2/**")
                        .permitAll()
                        .requestMatchers("/protected-oauth").authenticated())
                .oauth2Login(Customizer.withDefaults()).oauth2Client(Customizer
.withDefaults()
                );
        return http.build();
    }
}
```

Add the following settings to the `application.yml` file:

```
server:
  port: 8081
spring:
  security:
    oauth2:
      client:
        registration:
          openam:
            authorization-grant-type: authorization_code
            client-id: test_client
            client-secret: changeme
            scope:
              - openid
              - profile
        provider:
          openam:
            authorization-uri: http://openam.example.org:8080/openam/oauth2/authorize
            token-uri: http://openam.example.org:8080/openam/oauth2/access_token
            user-name-attribute: sub
            issuer-uri: http://openam.example.org:8080/openam/oauth2
```

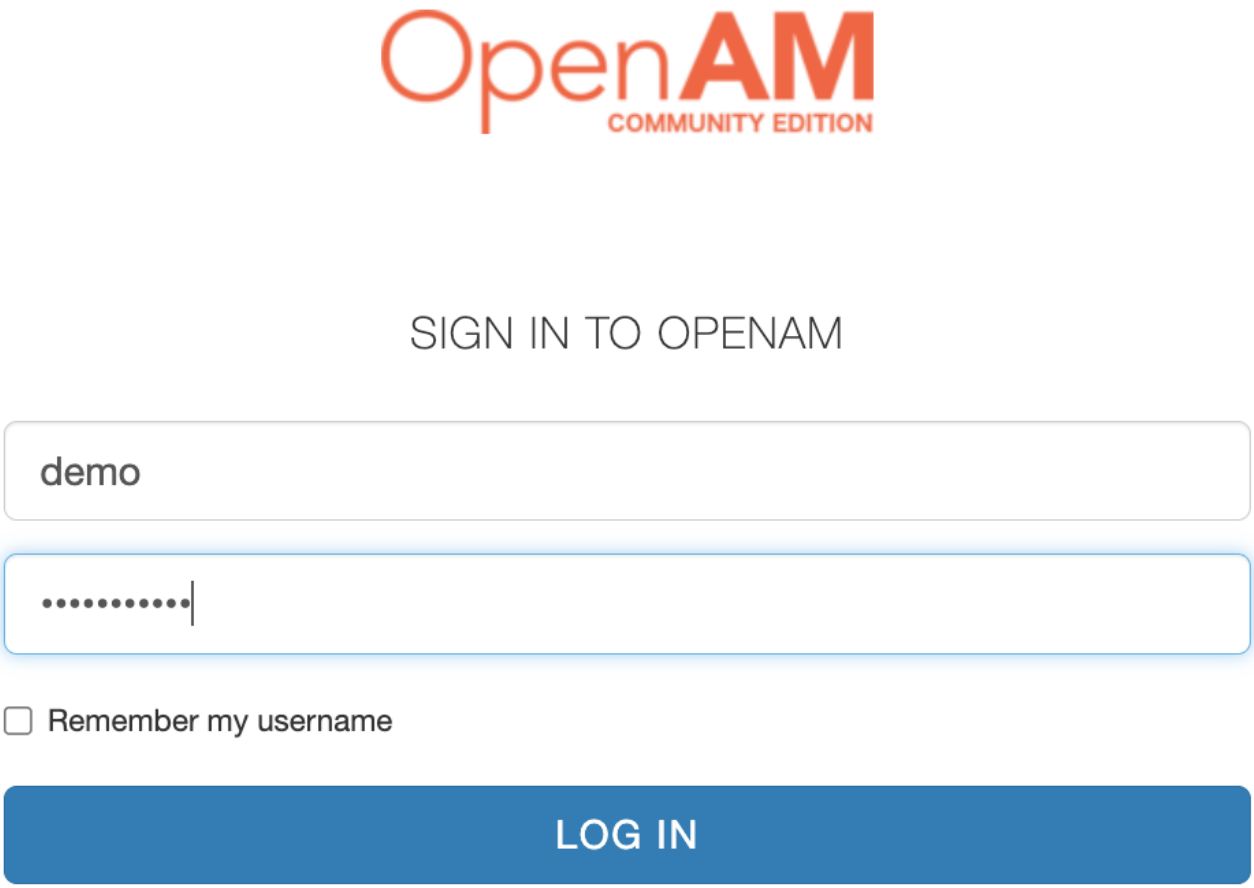
Test the Solution

Logout form OpenAM if you are logged in.

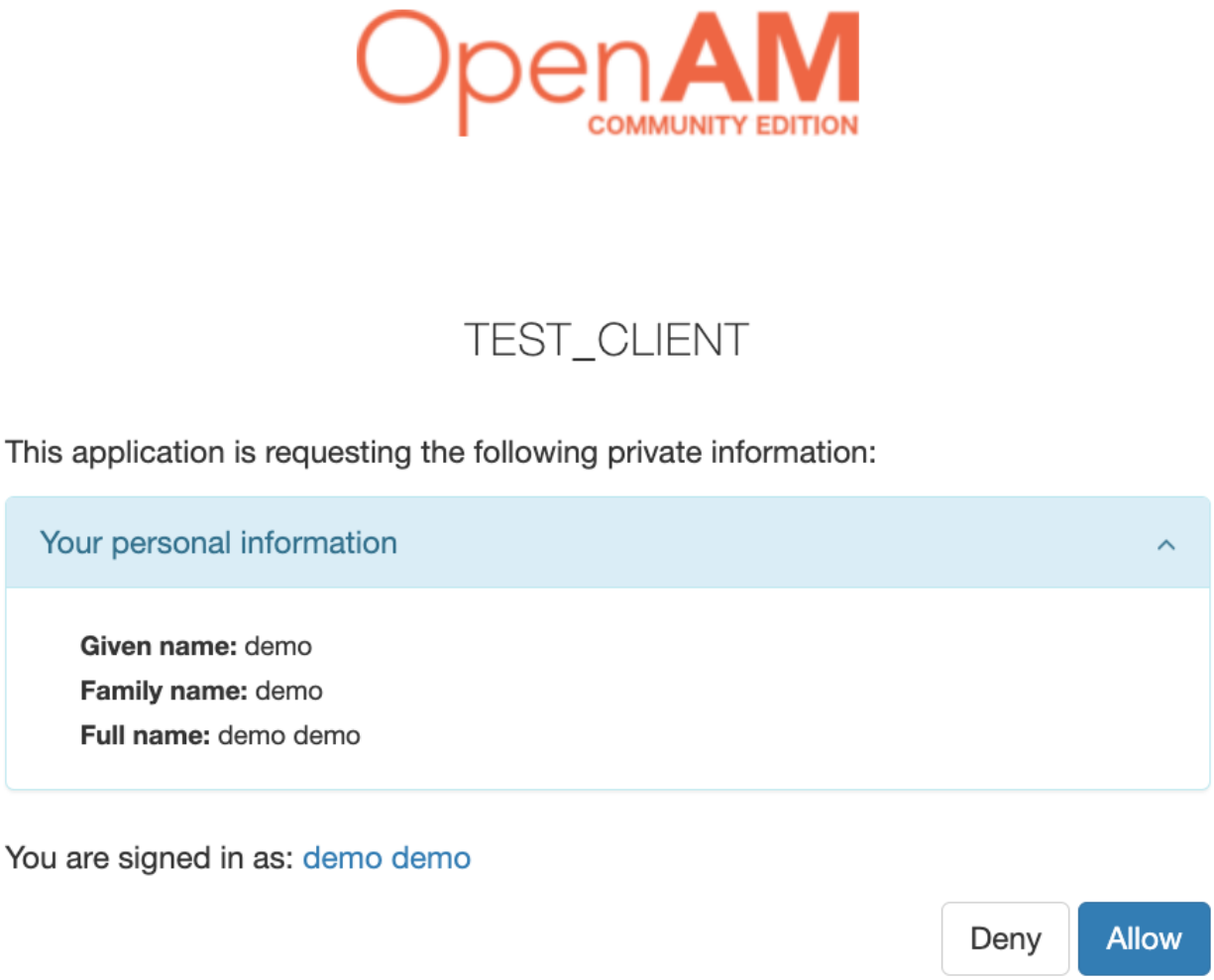
Run the Spring application an open its URL in a browser: http://localhost:8081



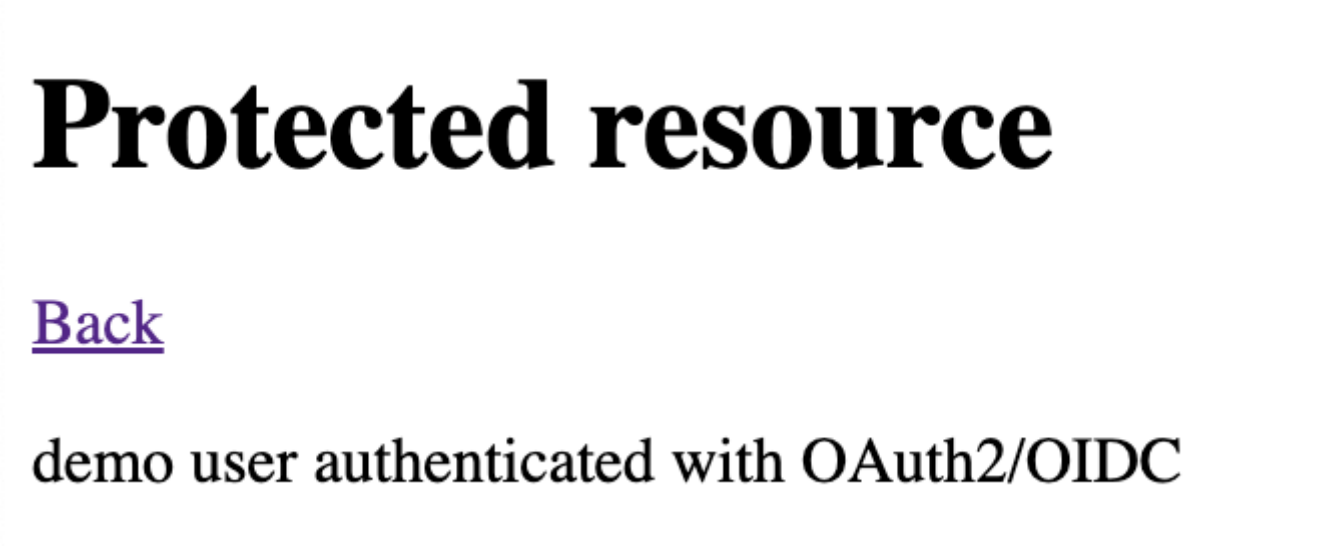
Click on the OAuth2/OIDC navigation link, and you will be redirected to the OpenAM authentication page.



Enter the user's login and password and then press the LOG IN button.



After pressing the Allow button, you will be redirected to the Spring Security Application as an authenticated user.



OpenAM and Spring Boot 3 Integration via OpenAM Cookie

Introduction

In the following article, we will set up OpenAM as an Identity Provider for a Spring Boot Application. In the previous article, we set up a Spring Boot application authentication via OpenAM with SAMLv2 protocol. In the following article, we will set up authentication via OpenAM cookie.

Prerequisites

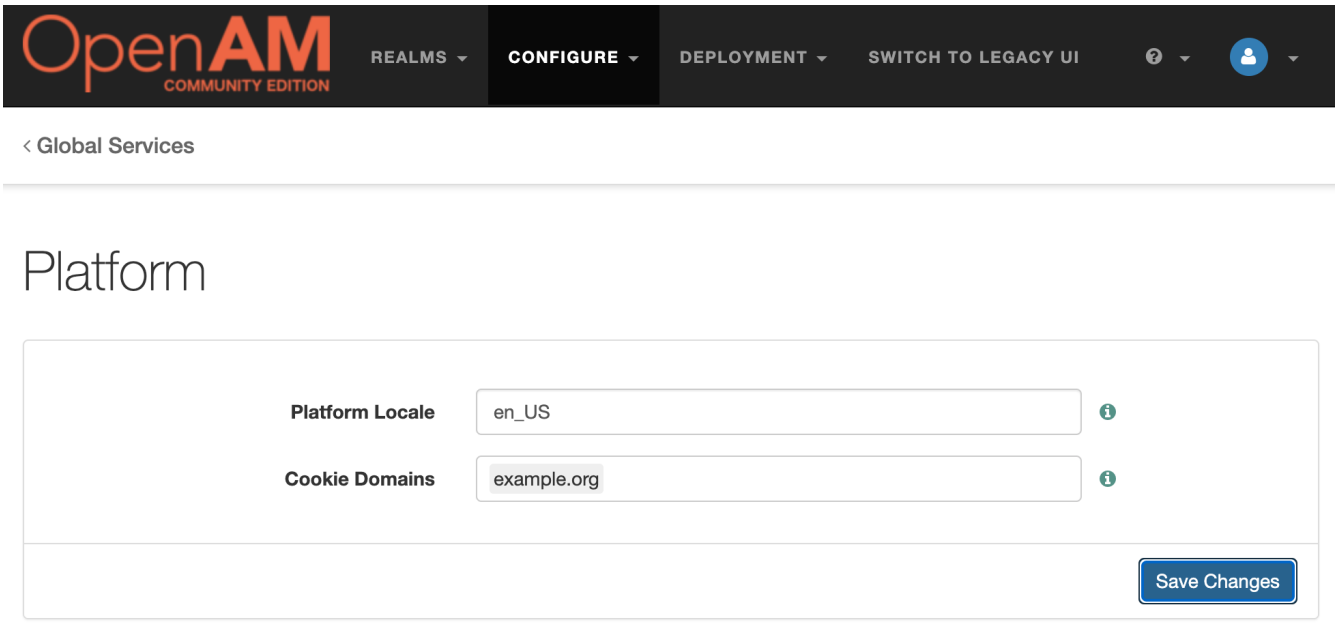
OpenAM and Spring Boot applications should share same domain. In the following example, OpenAM host is `openam.example.org` and Spring Boot application host is `app.example.org`.

To test the example on the local machine, add the following lines to your hosts file:

```
127.0.0.1 openam.example.org app.example.org
```

OpenAM Configuration

Open OpenAM administration console. In the top menu go to Configure → Global Services → Platform. Set Cookie Domains as the common domain for the both applications: `example.org`



Spring Application Configuration

Create a new Spring Boot application and add the following Maven repositories and add the following dependencies:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!--security dependencies-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Add the following settings to the `application.yml` file

```
server:
  port: 8081
```

Create a controller and two endpoints: `index` and `protected-openam`. The `index` endpoint will be accessible for anyone and `protected-openam` endpoint will be accessible for OpenAM authenticated users.

```
@Controller
public class SampleController {

    @GetMapping
    public String index() {
        return "index";
    }

    @GetMapping("/protected-openam")
    public String cookieProtected(Model model,  @AuthenticationPrincipal String
principal) {
        model.addAttribute("userName", principal);
        model.addAttribute("method", "OpenAM Cookie");
        return "protected";
    }
}
```

Create the following templates for controllers:

`index.html`

```
<!DOCTYPE html><html>
<body>
<h1>OpenAM Spring Security Integration</h1>
```

```
<h2>Test Authentication</h2>
<ul>
    <li><a href="/protected-openam">OpenAM Cookie</a></li>
</ul>
</body>
</html>
```

protected.html

```
<!DOCTYPE html><html xmlns:th="http://www.thymeleaf.org">
<body>
<h1>Protected resource</h1>
<a href="/">Back</a></li>
<p><span th:text="${userName}"/> user authenticated with <span th:text=
"${method}"/></p>
</body>
</html>
```

Create Spring Security configuration class:

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration {
    @Bean
    public SecurityFilterChain securityOpenAmFilterChain(HttpSecurity http) throws
Exception {
        http.securityMatcher("/protected-openam", OpenAmAuthenticationFilter
.OPENAM_AUTH_URI)
                .addFilterAt(new OpenAmAuthenticationFilter(),
RememberMeAuthenticationFilter.class)
                .authorizeHttpRequests((authorize) ->
                        authorize.anyRequest().fullyAuthenticated())
                .exceptionHandling(e ->
                        e.authenticationEntryPoint((request, response, authException)
->
                                response.sendRedirect(OpenAmAuthenticationFilter
.OPENAM_AUTH_URI)));
        return http.build();
    }
}
```

Create OpenAM authentication filter:

```
public class OpenAmAuthenticationFilter extends AbstractAuthenticationProcessingFilter
{

    private final String openAmUrl = "http://openam.example.org:8080/openam";
    private final String openAuthUrl = openAmUrl.concat("/XUI/#login");
```

```
    private final String openAmUserInfoUrl = openAmUrl.concat
("/json/users?_action=idFromSession");
    private final String openAmCookieName = "iPlanetDirectoryPro";

    private final String redirectUrl = "http://app.example.org:8081/protected-openam";

    public static final String OPENAM_AUTH_URI = "/openam-auth";

    public OpenAmAuthenticationFilter() {
        super(OPENAM_AUTH_URI, new OpenAmAuthenticationManager());
        setSecurityContextRepository(new HttpSessionSecurityContextRepository());
    }

    private final AuthenticationDetailsSource<HttpServletRequest, ?>
authenticationDetailsSource = new WebAuthenticationDetailsSource();
    @Override
    public Authentication attemptAuthentication(HttpServletRequest request,
HttpServletResponse response)
            throws AuthenticationException, IOException {
        Optional<Cookie> openamCookie = Arrays.stream(request.getCookies())
                .filter(c -> c.getName().equals(openAmCookieName)).findFirst();
        if(openamCookie.isEmpty()) {
           response.sendRedirect(openAuthUrl + "&goto=" + URLEncoder.encode
(redirectUrl, StandardCharsets.UTF_8));
           return null;
        } else {
            String userId = getUserIdFromSession(openamCookie.get().getValue());
            if (userId == null) {
                throw new BadCredentialsException("invalid session!");
            }
            OpenAmAuthenticationToken token = new OpenAmAuthenticationToken(userId);
            token.setDetails(authenticationDetailsSource.buildDetails(request));
            return this.getAuthenticationManager().authenticate(token);
        }
    }

    protected String getUserIdFromSession(String sessionId) {
        RestTemplate restTemplate = new RestTemplate();
        ParameterizedTypeReference<Map<String, String>> responseType =
                new ParameterizedTypeReference<>() {};
        HttpHeaders headers = new HttpHeaders();
        headers.add(openAmCookieName, sessionId);
        headers.setContentType(MediaType.APPLICATION_JSON);
        HttpEntity<?> entity = new HttpEntity<>(headers);
        ResponseEntity<Map<String, String>> response = restTemplate.exchange
(openAmUserInfoUrl, HttpMethod.POST, entity, responseType);
        Map<String, String> body = response.getBody();
        if (body == null) {
            return null;
        }
```

```
        return body.get("id");
    }
}
```

Create OpenAM authentication manager

```
public class OpenAmAuthenticationManager implements AuthenticationManager {
    @Override
    public Authentication authenticate(Authentication authentication) throws
AuthenticationException {
        if(authentication instanceof OpenAmAuthenticationToken) {
            authentication.setAuthenticated(true);
            return authentication;
        }
        authentication.setAuthenticated(false);
        return authentication;
    }
}
```

and OpenAM authentication token class:

```
public class OpenAmAuthenticationToken extends AbstractAuthenticationToken {
    private final String username;

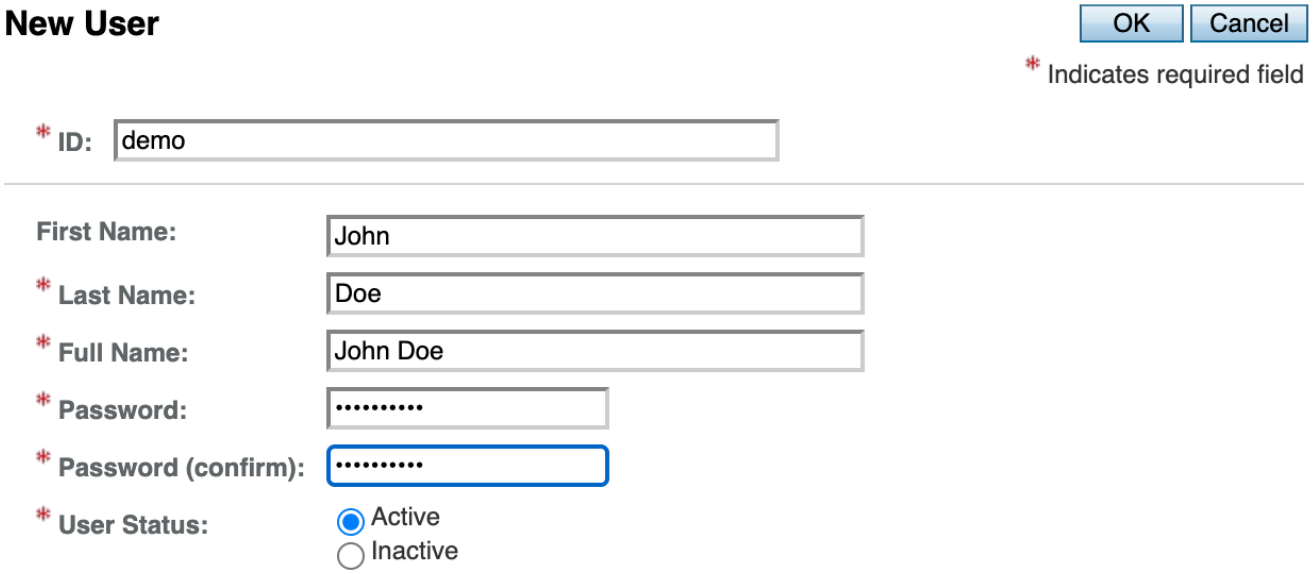
    public OpenAmAuthenticationToken(String username) {
        super(Collections.singleton(new SimpleGrantedAuthority("ROLE_USER")));
        this.username = username;
    }

    @Override
    public Object getCredentials() {
        return "";
    }

    @Override
    public Object getPrincipal() {
        return username;
    }
}
```

Test Solution

Create the `demo` user for the top level realm for testing purposes. Open OpenAM console, open the top level realm. In the left menu click the `Subjects` element. In the list of subject click the `New` button.



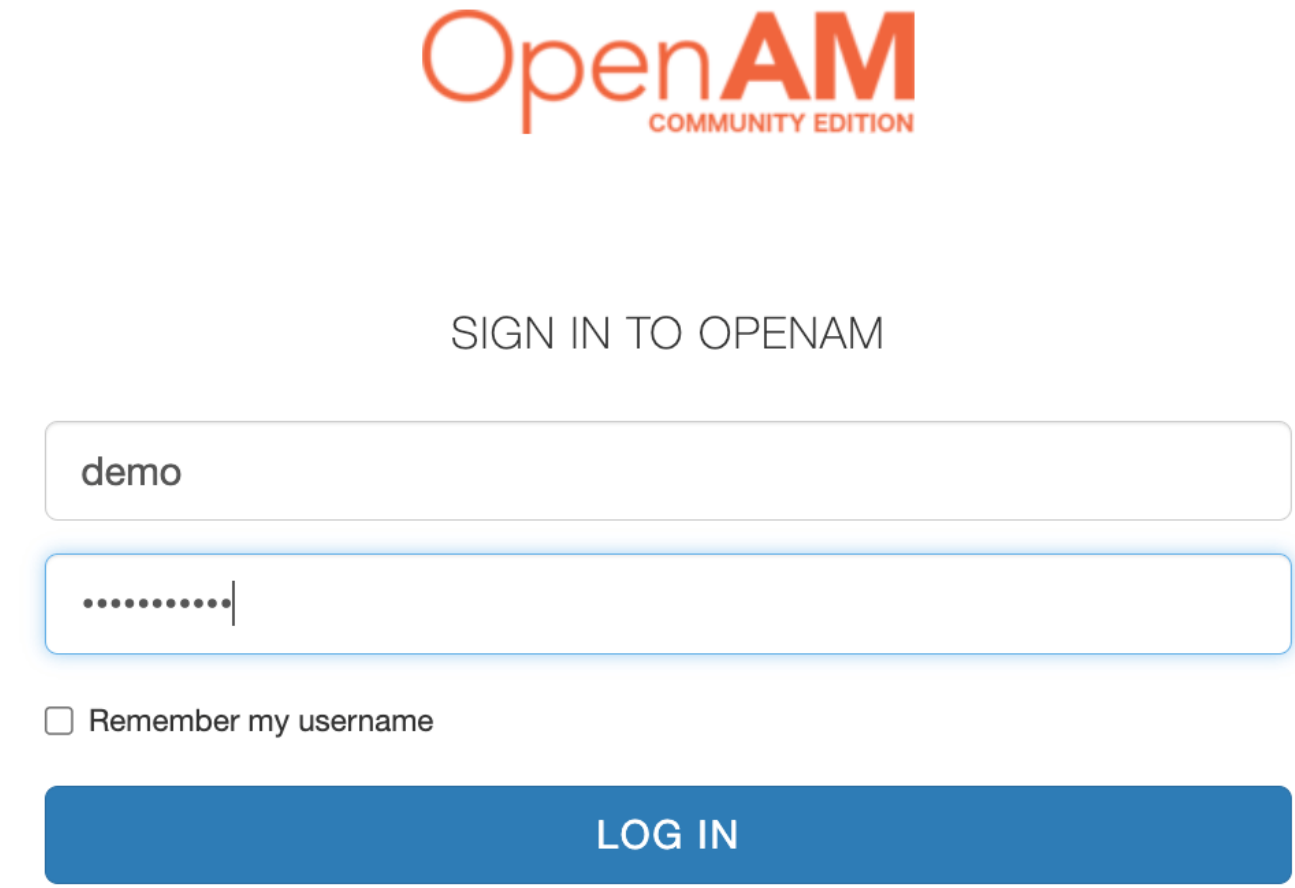
Fill the attributes and press the `OK` button.

Logout form OpenAM.

Run the Spring application an open its URL in a browser: http://app.example.org:8081.



Click the OpenAM Cookie link. You will be redirected to the OpenAM authentication page.



Enter desired credentials and press the Log In button. After successful authentication you will be redirected to the Spring Boot application page as an authenticated user.



PS. Thanks to https://www.tune-it.ru/web/adpashnin/blog/-/blogs/spring-security-i-openam for the idea for this manual.

OpenAM and Spring Boot 3 Integration via SAMLv2 Protocol

Introduction

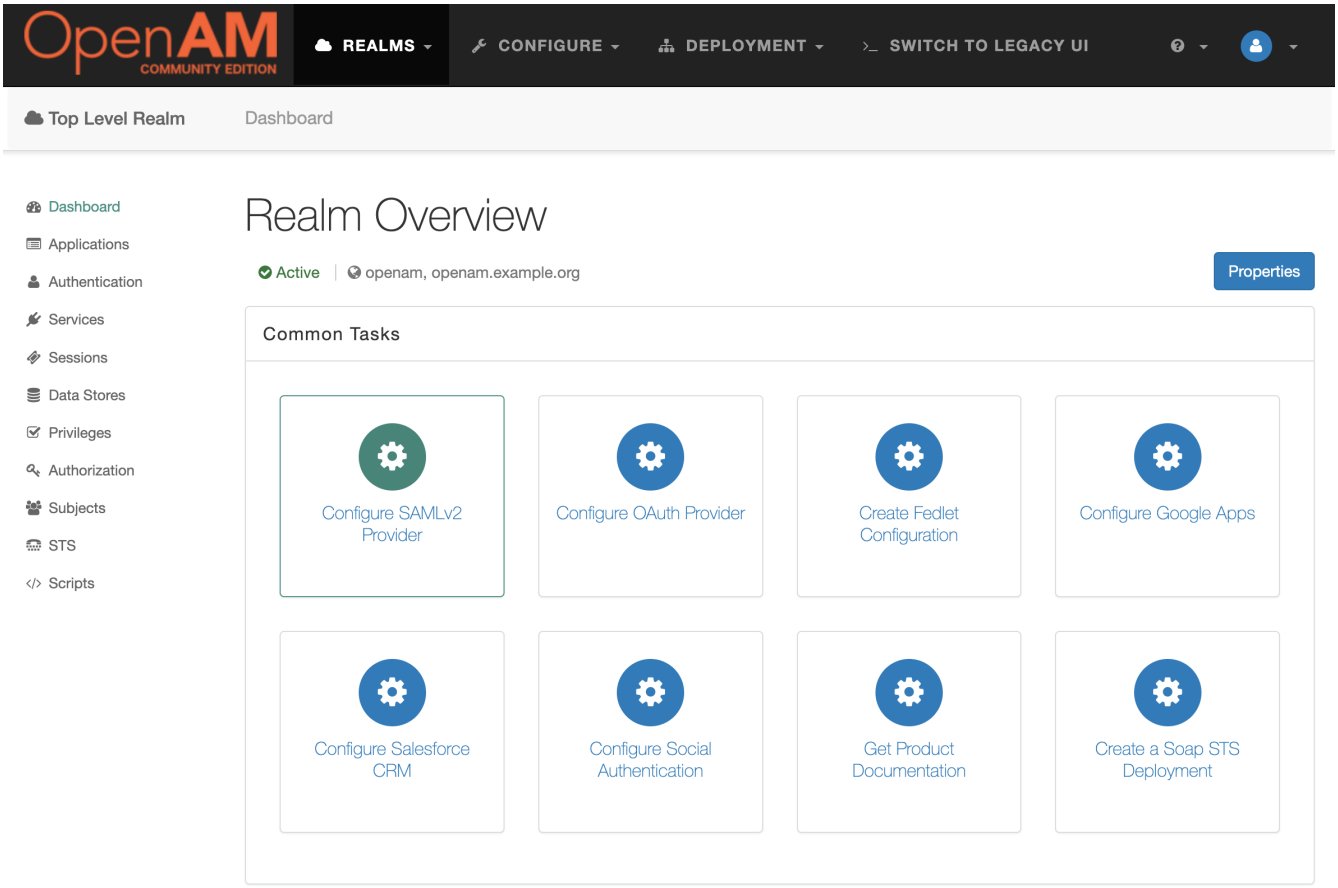
In the following article, we will set up OpenAM as an Identity Provider for a Spring Boot Application. In the previous article, we set up a Spring Boot application authentication via OpenAM with OAuth2/OIDC protocol. In the following article, we will set up authentication via OpenAM as an Identity Provider and Spring Boot application as a service provider using SAMLv2 protocol.

OpenAM Configuration

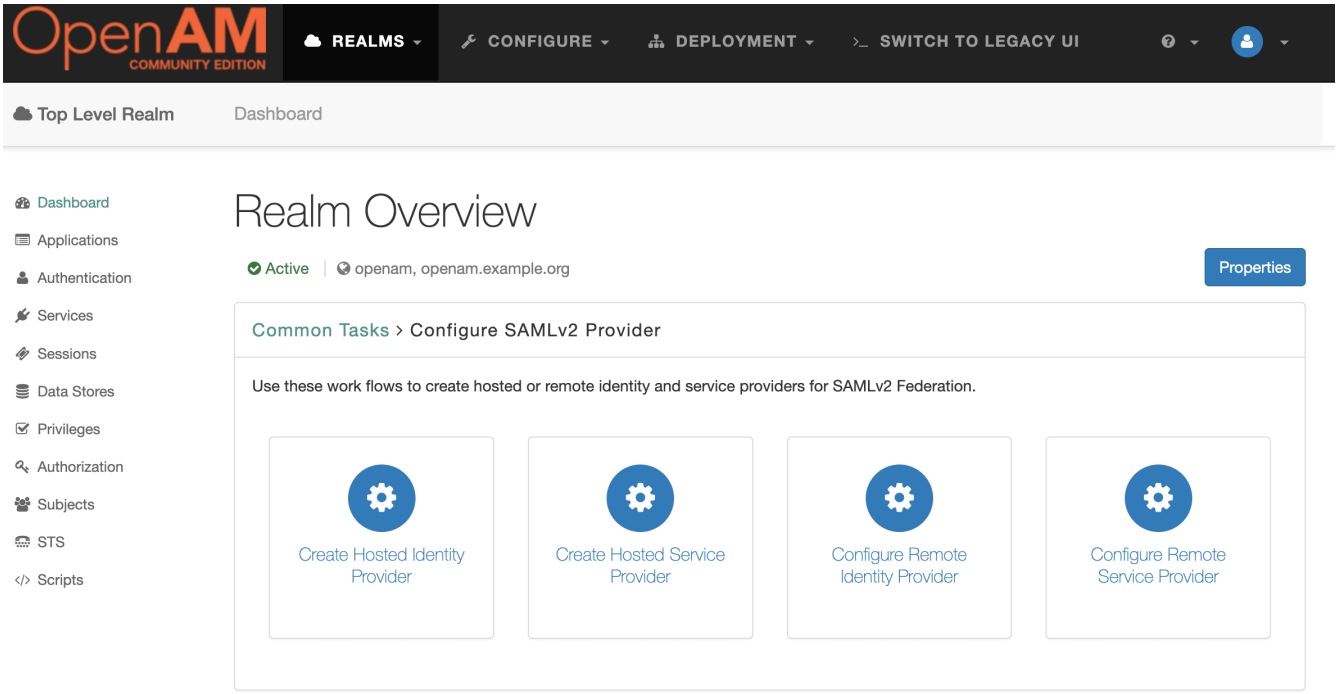
Create Hosted Identity Provider

If you have not installed OpenAM yet, you could run OpenAM as a Docker image. Let's assume the OpenAM instance URL is `http://openam.example.org:8080/openam`

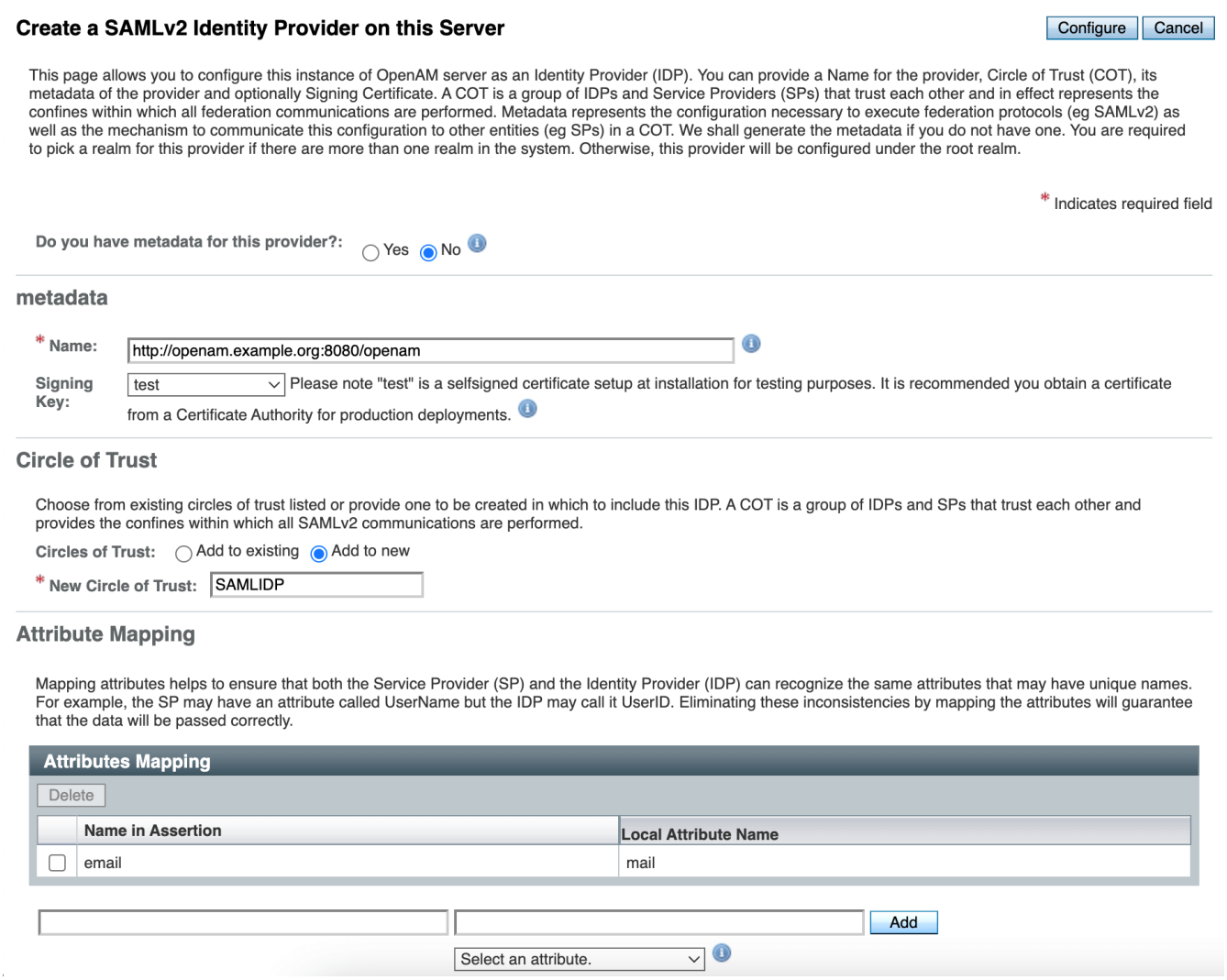
Go to the target realm and in the Common Tasks section. Open OpenAM console in a browser and press the Configure SAMLv2 Provider button.



Then press the Create Hosted Identity Provider button.



Select a Signing Key (for demonstration purposes we will use **test**), enter the Circle of Trust name, and then map user's attributes to OpenAM. For demonstration the OpenAM users will be mapped by email.



Press the Configure button.

Setup User Mapping in OpenAM

Open the administration console, goto desired realm, from the menu on the left. Select the applications section and navigate to SAML 2.0. In the Entity Providers list select `http://openam.example.org:8080/openam` On the Assertion Content tab, under **NameID Format** in the **NameID Value Map** list, add the element `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified=mail` Press the Save button.

Create OpenAM Users

Open the administration console, goto desired realm, create a new user and fill the Email Address attribute

Spring Application Configuration

Create a new Spring Boot application and add the following Maven repositories and dependencies:

```
<repositories>
    <repository>
        <id>releases</id>
        <url>https://repo.maven.apache.org/maven2</url>
    </repository>
    <repository>
        <id>shibboleth</id>
        <url>https://build.shibboleth.net/nexus/content/repositories/releases/</url>
    </repository>
</repositories>
....
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!--security dependencies-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-saml2-service-provider</artifactId>
</dependency>
```

Generate certificate and key files with the OpenSSL tool.

```
openssl req -newkey rsa:2048 -nodes -keyout rp-private.key -x509 -days 365 -out rp-
certificate.crt
```

Put the generated files into `src/main/resources/credentials` directory.

Add the following settings to the `application.yml` file

```
server:
  port: 8081
spring:
  security:
    saml2:
      relyingparty:
        registration:
          openam:
            signing:
              credentials:
              - private-key-location: classpath:credentials/rp-private.key
                certificate-location: classpath:credentials/rp-certificate.crt
            singlelogout:
              binding: POST
              url: "{baseUrl}/logout/saml2/slo"
            assertingparty:
              metadata-uri:
http://openam.example.org:8080/openam/saml2/jsp/exportmetadata.jsp
```

Create a controller and two endpoints: `index` and `protected-saml`. The `index` endpoint will be accessible for anyone and `protected-saml` endpoint will be accessible for authenticated with OpenAM users via SAMLv2 protocol.

```
@Controller
public class SampleController {

    @GetMapping
    public String index() {
        return "index";
    }

    @GetMapping("/protected-saml")
    public String samlProtected(Model model, @AuthenticationPrincipal
Saml2AuthenticatedPrincipal principal) {
        String emailAddress = principal.getFirstAttribute("email");
        model.addAttribute("userName", emailAddress);
        model.addAttribute("method", "SAMLv2");
        return "protected";
    }
```

```
}
```

Create the following templates for controllers:

index.html

```
<!DOCTYPE html><html>
<body>
<h1>OpenAM Spring Security Integration</h1>
<h2>Test Authentication</h2>
<ul>
    <li><a href="/protected-saml">SAMLv2</li>
</ul>
</body>
</html>
```

protected.html

```
<!DOCTYPE html><html xmlns:th="http://www.thymeleaf.org">
<body>
<h1>Protected resource</h1>
<a href="/">Back</a></li>
<p><span th:text="${userName}"/> user authenticated with <span th:text=
"${method}"/></p>
</body>
</html>
```

Create Spring Security configuration

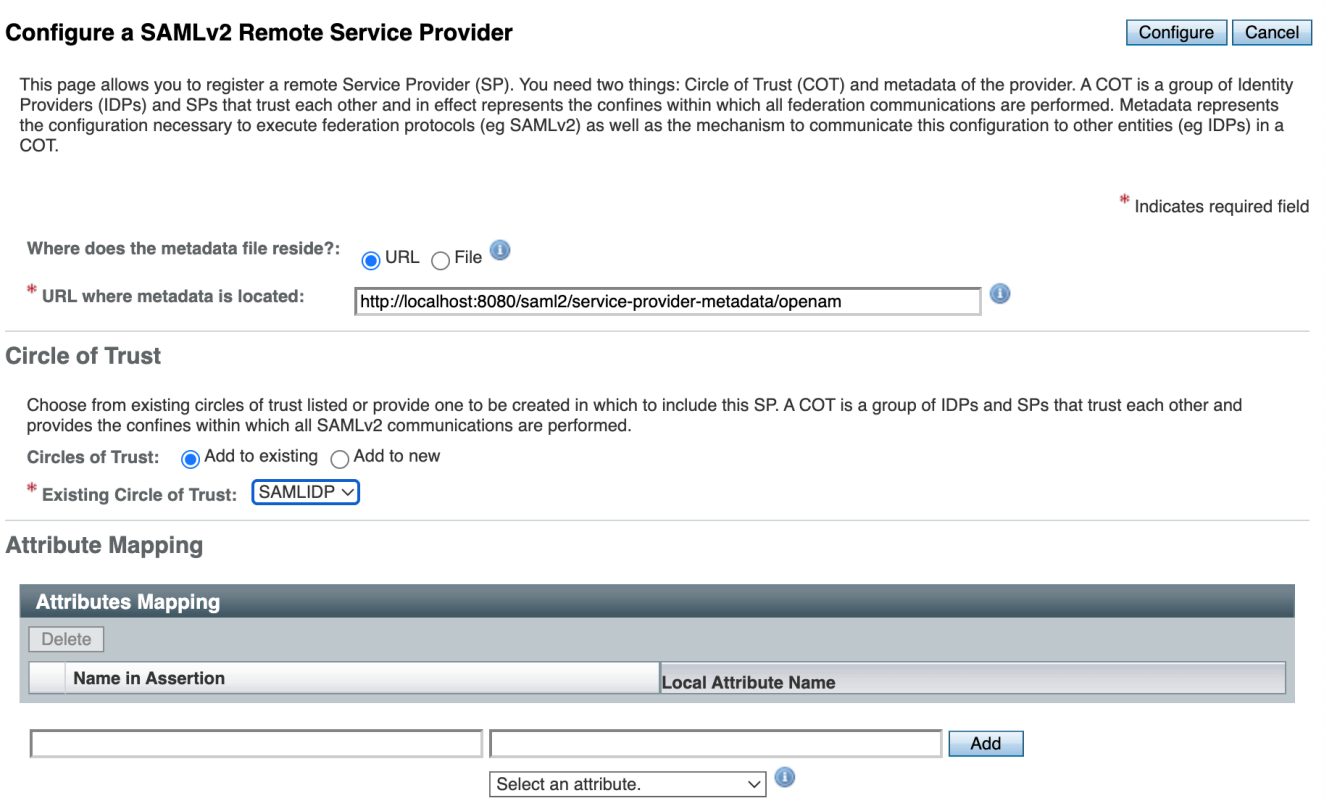
```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration {
    @Bean
    public SecurityFilterChain securitySamlFilterChain(HttpSecurity http) throws
Exception {
        http.authorizeHttpRequests((authorize) -> authorize.requestMatchers("/")
                        .permitAll()
                        .requestMatchers("/protected-saml").authenticated())
                .saml2Metadata(Customizer.withDefaults())
                .saml2Login(Customizer.withDefaults())
                .saml2Logout(Customizer.withDefaults());
        return http.build();
    }
}
```

Setup Service Provider in OpenAM

Run the Spring Boot Project to import the SAML metadata file of the Spring Boot project.

Open the OpenAM administration console, goto desired realm, click **Configure SAMLv2 Provider** in the Common Tasks section, then click **Configure Remote Service Provider.**

Set the metadata URL: `http://localhost:8081/saml2/service-provider-metadata/openam` and select the existing circle of trust created in the Create Hosted Identity Provider Step.



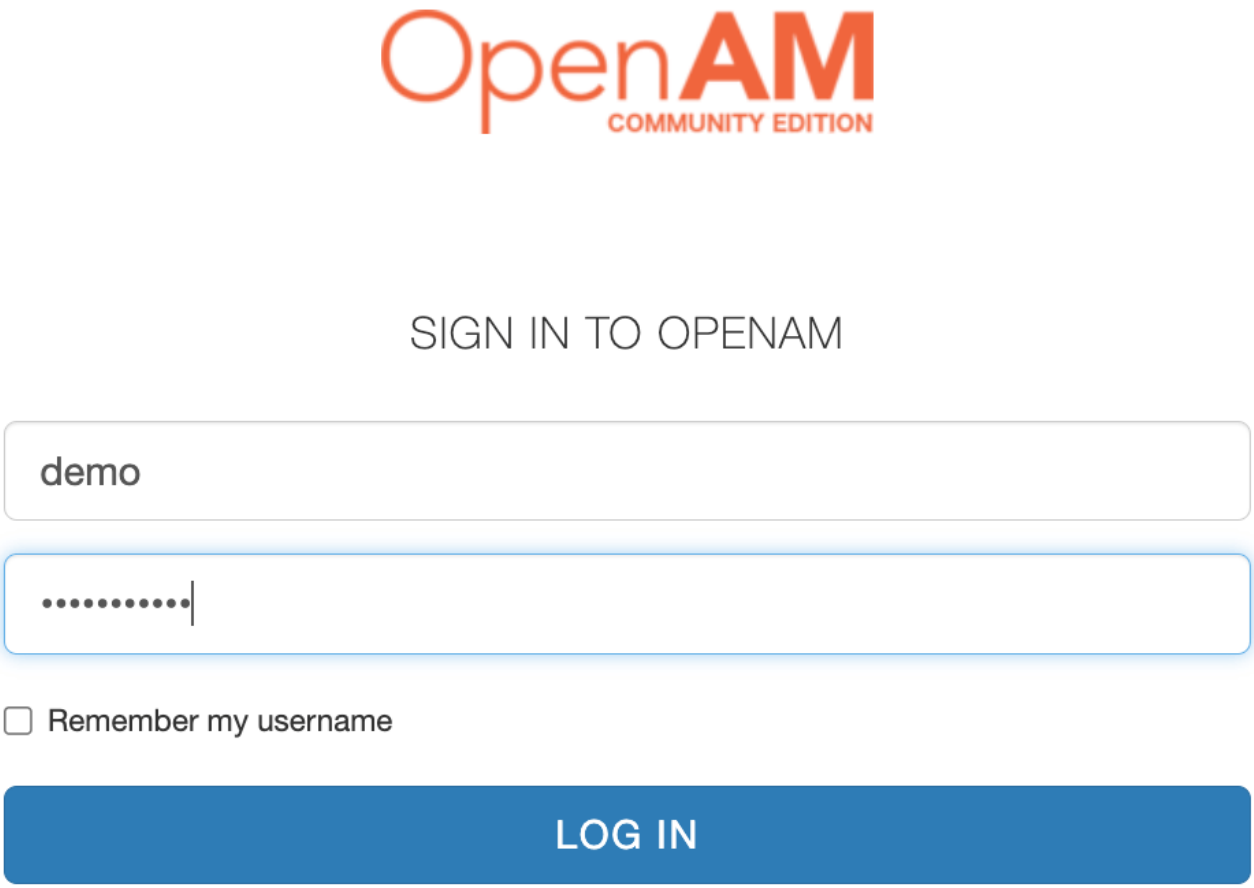
Test the Solution

Logout form OpenAM if you are logged in.

Run the Spring application an open its URL in a browser: http://localhost:8081



Enter the user's login and password and then press the LOG IN button.



After pressing the Log In button, you will be redirected to the Spring Security Application as an authenticated user.



OpenAM and Zero Trust: Confirming Critical Operations

Introduction

One of the principles of Zero Trust states: *Never trust, always verify*. In this article, we will explore how to implement this principle in an authentication system using open-source products OpenAM and OpenIG.

A practical example of this principle can be seen in banking applications. When confirming a payment, banks almost always want to ensure that it is *you* conducting the transaction, not a malicious actor. To verify this, they send a one-time code to a trusted device via push notification or SMS.

Alternatively, the user may be asked to confirm their biometric data, such as a fingerprint, use a hardware token, or rely on a specialized application like Microsoft Authenticator or Google Authenticator.

Solution Design

The solution consists of three components:

- **Protected Application**: Any application (in this case, a simple Node.js application) where users authenticate. It includes two pages: a profile page and a page with sensitive data.
- **Authentication Service (OpenAM)**: Handles user authentication and authorization.
- **Authorization Gateway (OpenIG)**: Sends a request to OpenAM to verify policy compliance. If successful, it grants access to the protected application; otherwise, it redirects the user for authentication. When attempting to access sensitive information, OpenAM will check whether the user has authenticated using a one-time code, ensuring no more than 20 seconds have passed since the last authentication.

As a second authentication factor, one-time passwords generated using the TOTP algorithm will be used, along with mobile applications like Microsoft Authenticator or Google Authenticator.

Preparation

The complete solution code is available at this link: https://github.com/OpenIdentityPlatform/openam-openig-otp-example.

Preparing the Hosts File

Let's assume the hostname for the authentication service will be `openam.example.org` and for the gateway `openig.example.org`. Before running the setup, add these hostnames and IP addresses to the `hosts` file, for example:

```
127.0.0.1 openam.example.org openig.example.org
```

- On **Windows**, the `hosts` file is located at: `C:\Windows\System32\drivers\etc\hosts`
- On **Linux** and **Mac**, it is located at: `/etc/hosts`

Authenticator Application

Install the mobile application Microsoft Authenticator or Google Authenticator on your device.

Docker Compose

For simplicity, all services will be launched using `docker compose`.

Create an empty file named `docker-compose.yml` and add the `services` object:

```
services:
```

Demo Application

For demonstration purposes, we will use a simple Node.js application with two URLs:

- **Profile Page (`/`)** – Displays user profile information
- **Sensitive Data Page (`/sensitive`)** – Contains protected sensitive information

The application code can be found at this link.

```
const express = require("express");
const app = express();
const port = 3000;

app.set("view engine", "ejs");

app.use((req, res, next) => {
    console.log(req.headers)
    const token = req.headers.authorization;
    if (!token) {
        return res.status(401).send("Unauthorized");
    }
    next()
})

app.get("/", (req, res) => {
    const token = req.headers.authorization;
    const jwtPayload = JSON.parse(Buffer.from(token.split('.')[1], 'base64').
toString());
    const user = { name: jwtPayload.sub };
    res.render("profile", { user });
});

app.get("/sensitive", (req, res) => {
```

```
    const sensitiveData = { bankAccount: "1234-5678-9012-3456", secretKey:
"MY_SUPER_SECRET_KEY" };
    res.render("sensitive", { sensitiveData });
});

app.listen(port, () => console.log(`Server running at http://localhost:${port}`));
```

Add a demo application to the `docker-compose.yml` file in the `services` object

```
services:
  demo-app:
    build: ./demo-app
    container_name: demo-app
```

Run the application with the command `docker compose up -d --build demo-app`

OpenAM Authentication Service Configuration

Add the OpenAM seris to the `docker-compose.yml` file in the `services` object:

```
services:
...
  openam:
    image: openidentityplatform/openam:latest
    container_name: openam
    hostname: openam.example.org
    ports:
      - "8080:8080"
```

Start the OpenAM container with the command `docker compose up openam`. Wait for the container to start and perform the initial installation with the command:

```
docker exec -w '/usr/openam/ssoconfiguratortools' openam bash -c \
'echo "ACCEPT_LICENSES=true
SERVER_URL=http://openam.example.org:8080
DEPLOYMENT_URI=/$OPENAM_PATH
BASE_DIR=$OPENAM_DATA_DIR
locale=en_US
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=passw0rd
AMLDAPUSERPASSWD=p@passw0rd
COOKIE_DOMAIN=example.org
ACCEPT_LICENSES=true
DATA_STORE=embedded
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=openam.example.org
```

```
DIRECTORY_PORT=50389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1689
ROOT_SUFFIX=dc=openam,dc=example,dc=org
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=passw0rd" > conf.file && java -jar openam-configurator-tool*.jar
--file conf.file'
```

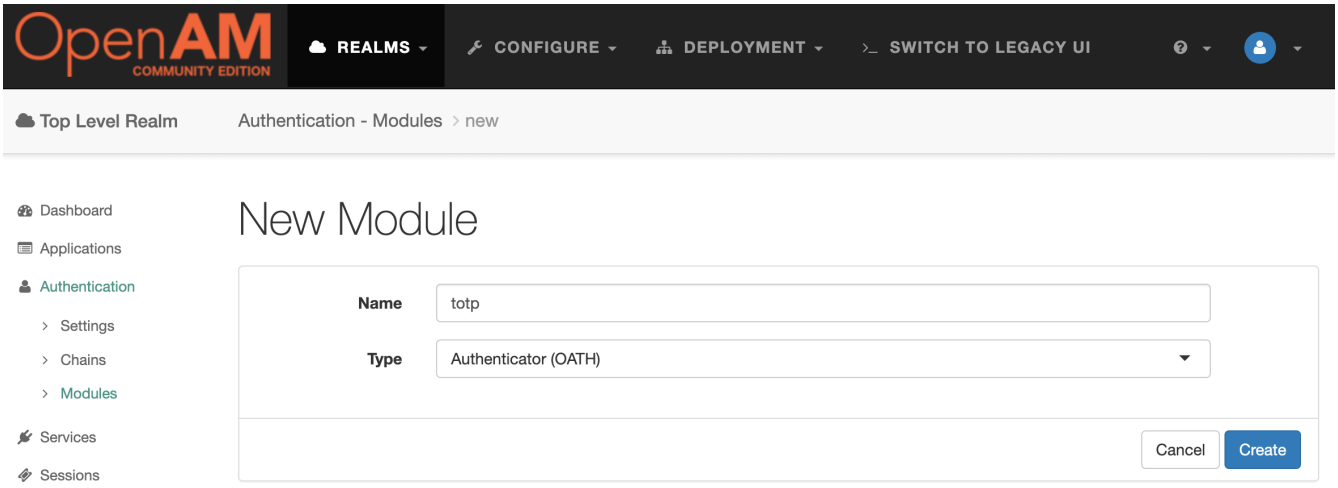
Wait until the installation is complete.

Configuring MFA in OpenAM

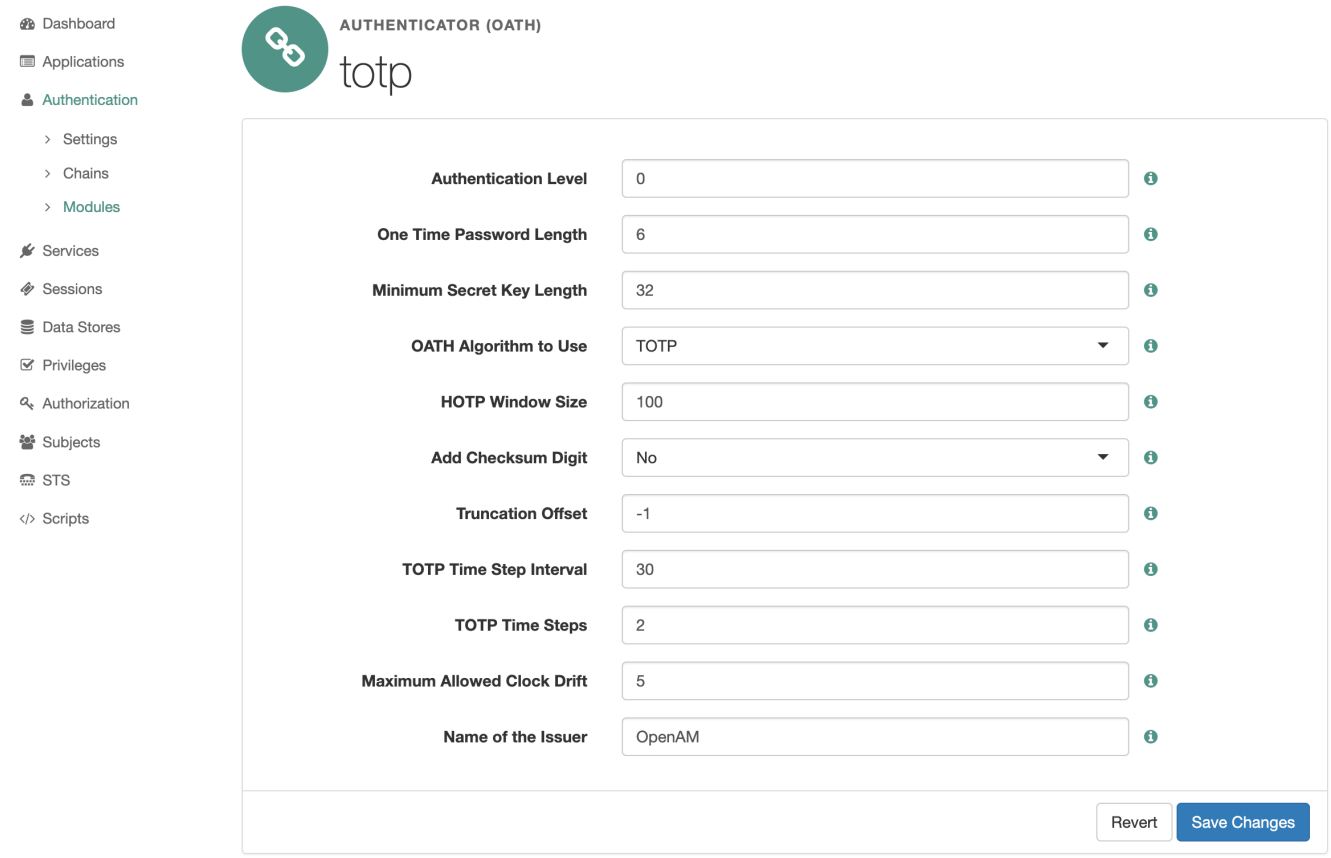
Open the OpenAM administrator console at http://openam.example.org:8080/openam.

Enter the administrator login and password. In this case, it is `amadmin` and `passw0rd` respectively.

In the console, select **Top Level Realm**. In the left menu, select **Authentication** → **Modules** and create a new module `totp` with type `Authenticator (OATH)`.

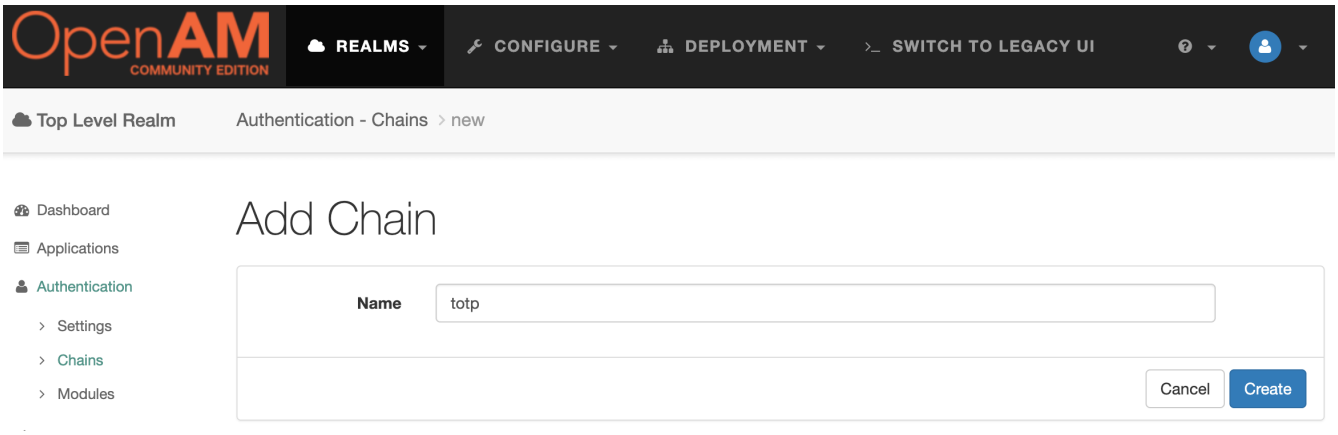


In the settings, select **OATH Algorithm to Use: TOTP**, also specify **Name of the Issuer**, e.g. **OpenAM**. The rest of the settings can be left unchanged. Save the `totp` module settings.

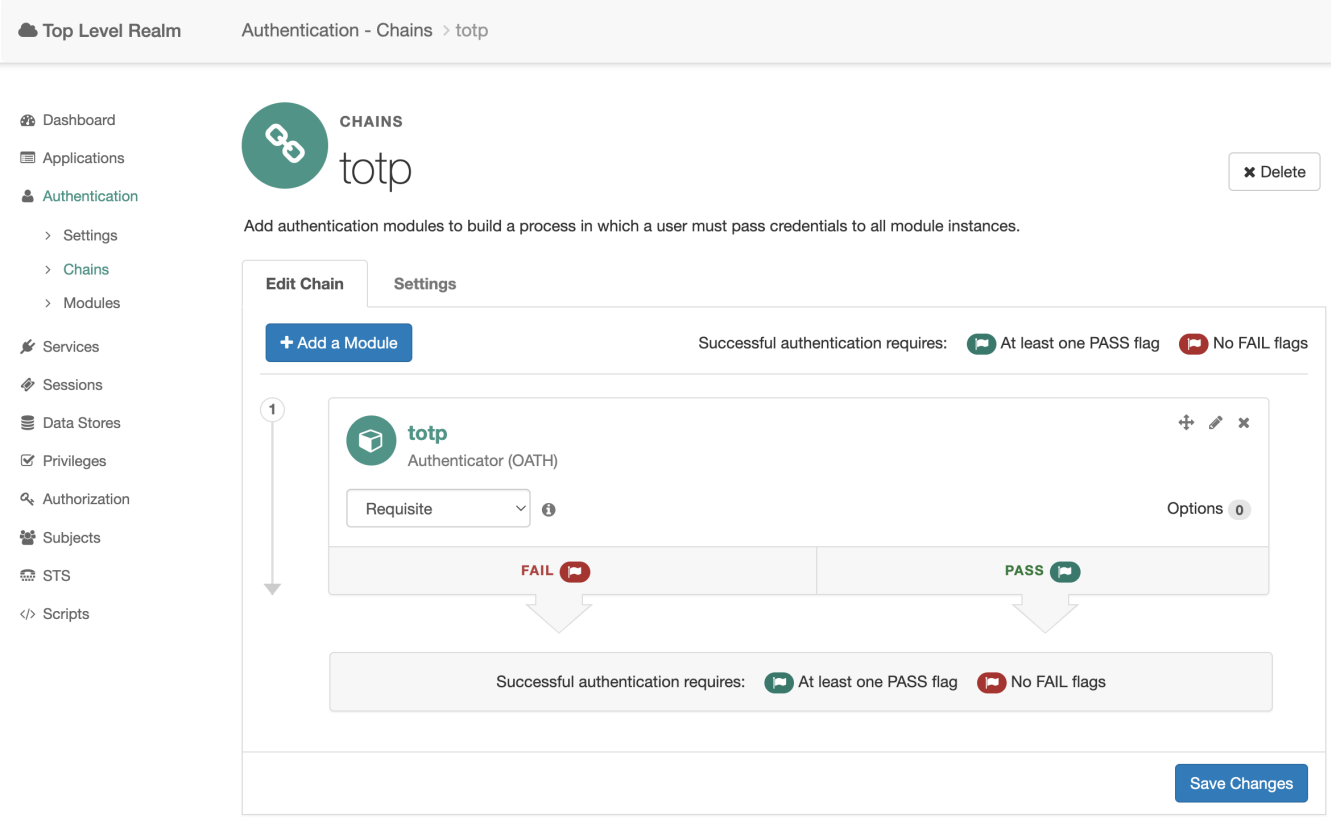


Next, let's configure the authentication chain

In the administrator console, open the Top Level Realm, then go to **Authentication** → **Chains** in the left menu and create a new authentication chain `totp`.



Add the created `totp` module to the chain and save the changes.

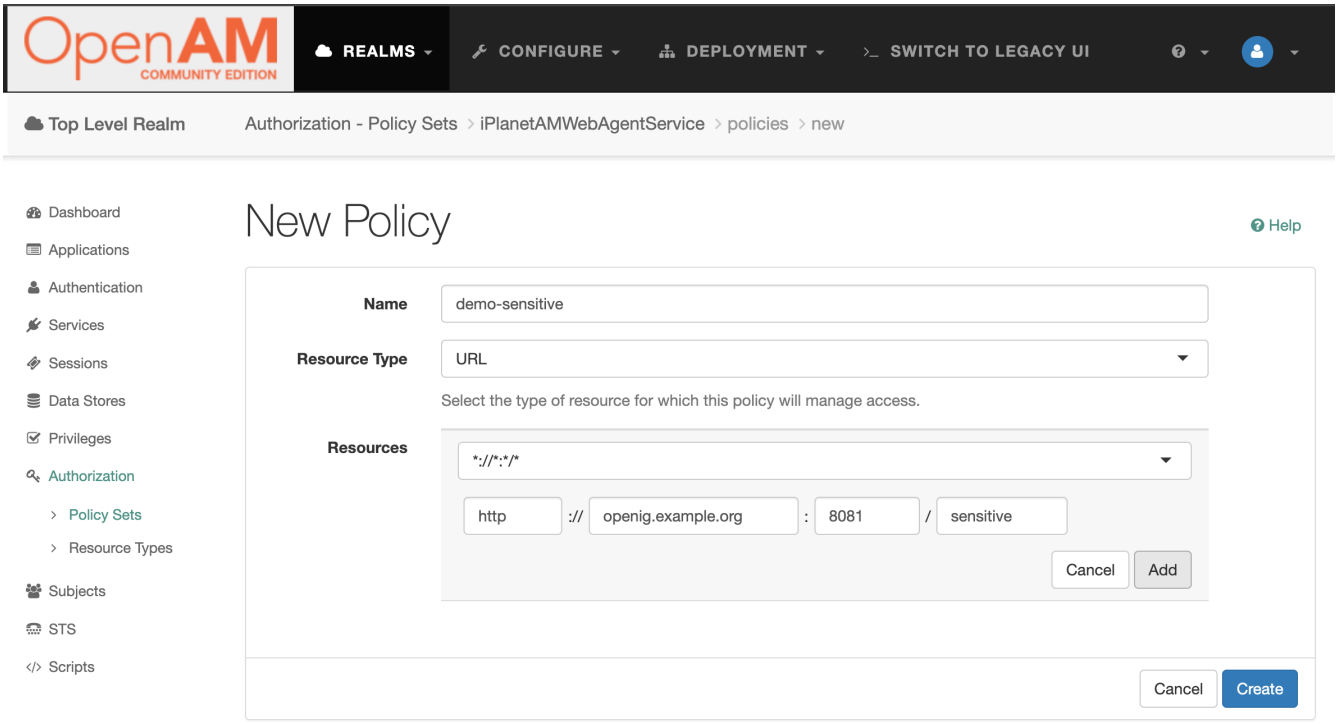


Configuring OpenAM Authorization Policy

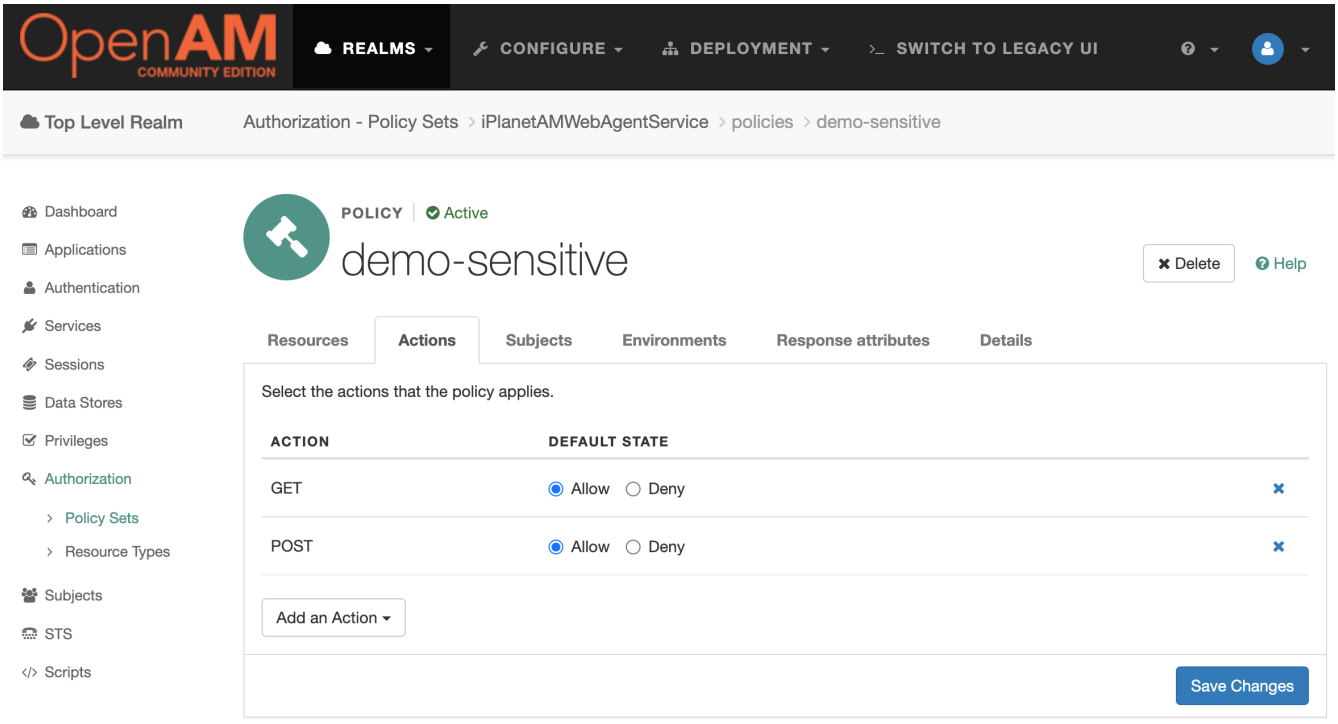
Now let's move on to configuring the authorization policy in OpenAM for the `/sensitive` endpoint of the demo application. The policy will be configured so that the user is required to authenticate with a one-time code in the `totp` authentication chain, but the authentication will only be valid for 20 seconds.

Open the OpenAM administrator console. Open Top Level Realm. From the menu on the left, select **Authorization → Policy Sets**. Select the **Default Policy Set**. Create a new `demo-sensitive` policy.

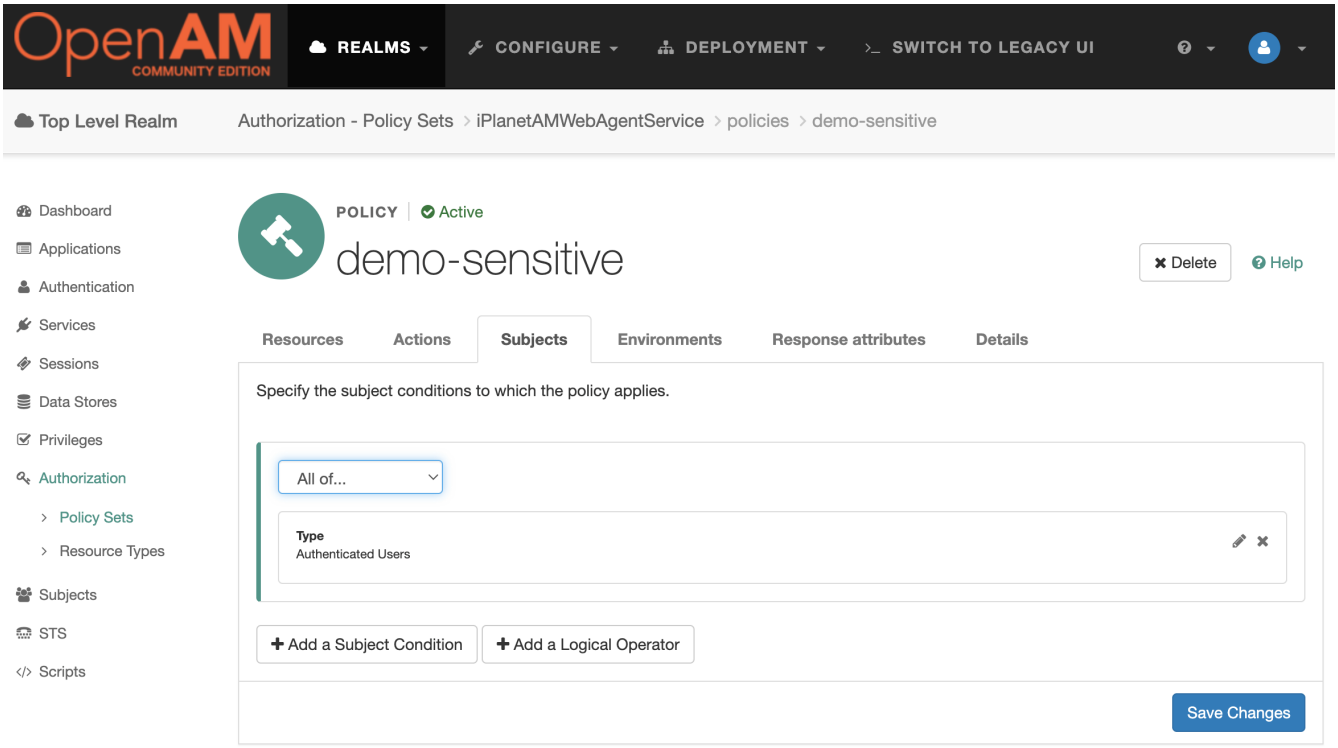
Select URL as the resource type and specify the resource as shown in the example in the figure below. Click **Add** and then **Create**.



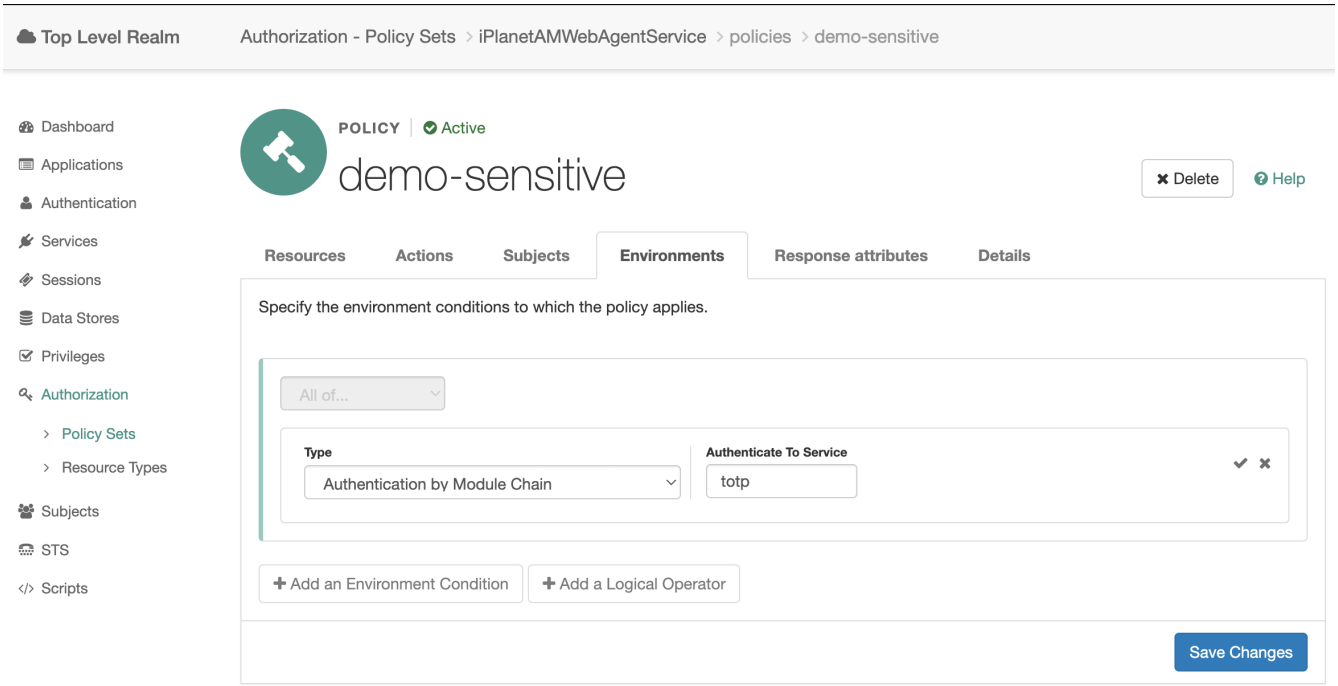
For the created policy, on the Resources tab, allow GET and POST requests.



On the Subjects tab, add the Authenticated Users type.

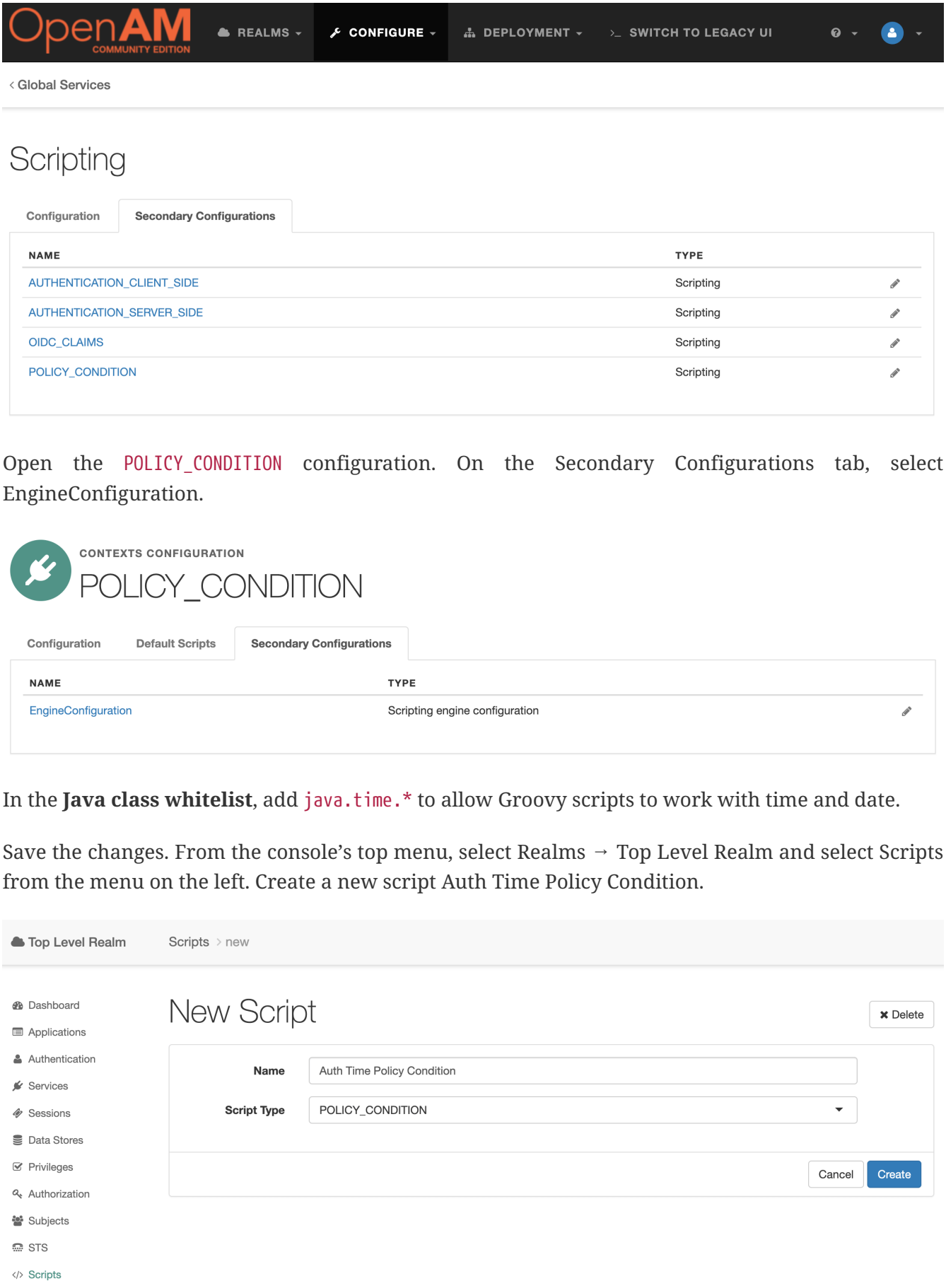


On the Environments tab, add the Authentication by Module Chain condition and add the `totp` chain.



Save the changes. This policy will authorize requests authenticated by the `totp` chain.

Now configure the policy so that access is only valid for 20 seconds. OpenAM doesn't have such a policy out of the box, so we'll configure a policy script. But first let's prepare OpenAM to work with the time from the script. In the top menu, go to Configure → Global Services. In the list that opens, select Scripting. Click the Secondary Configuration tab.

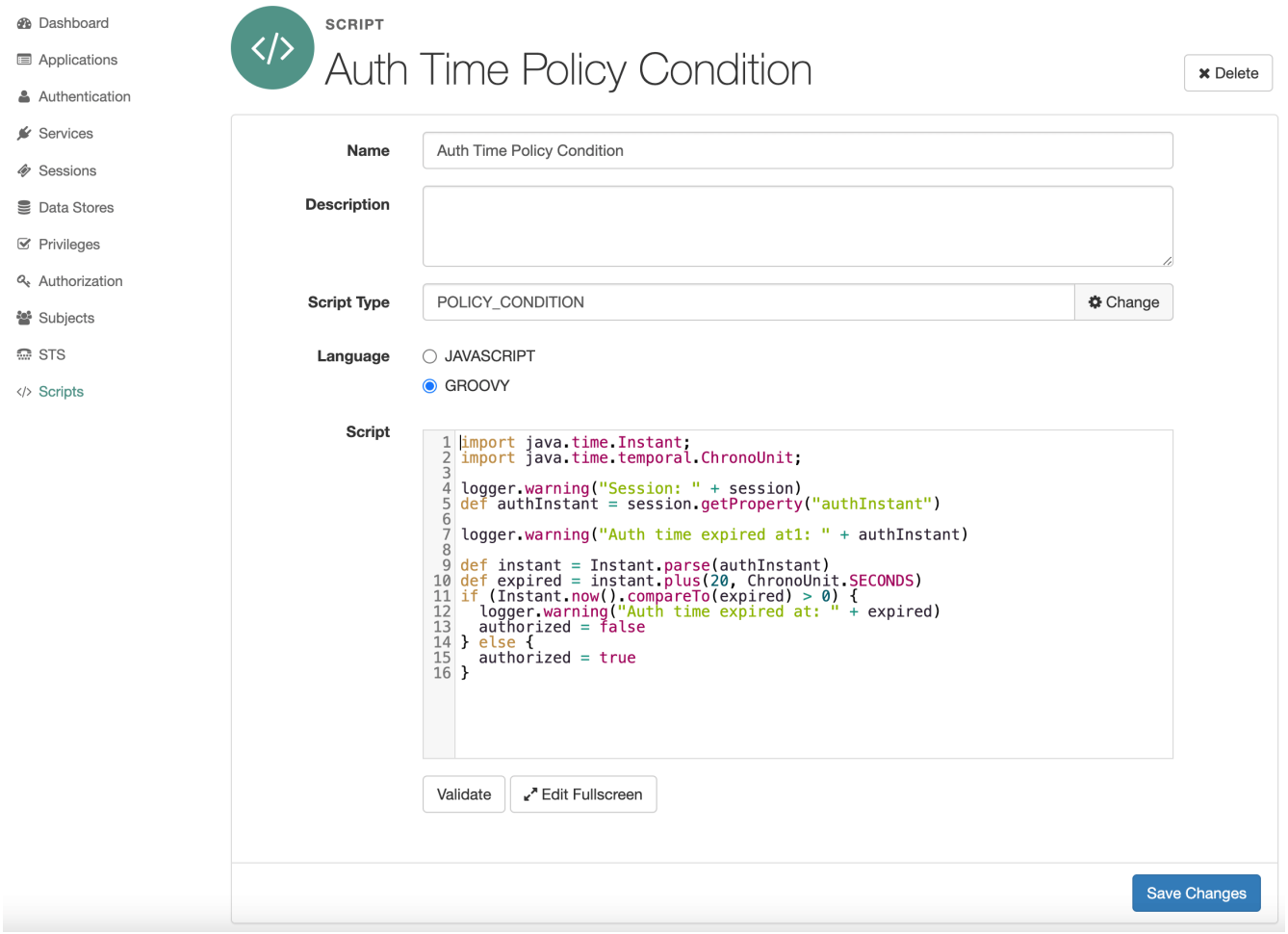


Open the `POLICY_CONDITION` configuration. On the Secondary Configurations tab, select EngineConfiguration.

In the **Java class whitelist**, add `java.time.*` to allow Groovy scripts to work with time and date.

Save the changes. From the console's top menu, select Realms → Top Level Realm and select Scripts from the menu on the left. Create a new script Auth Time Policy Condition.

Script type - POLICY_CONDITION. Language - Groovy.




```
import java.time.Instant;
import java.time.temporal.ChronoUnit;

logger.warning("Session: " + session)
def authInstant = session.getProperty("authInstant")

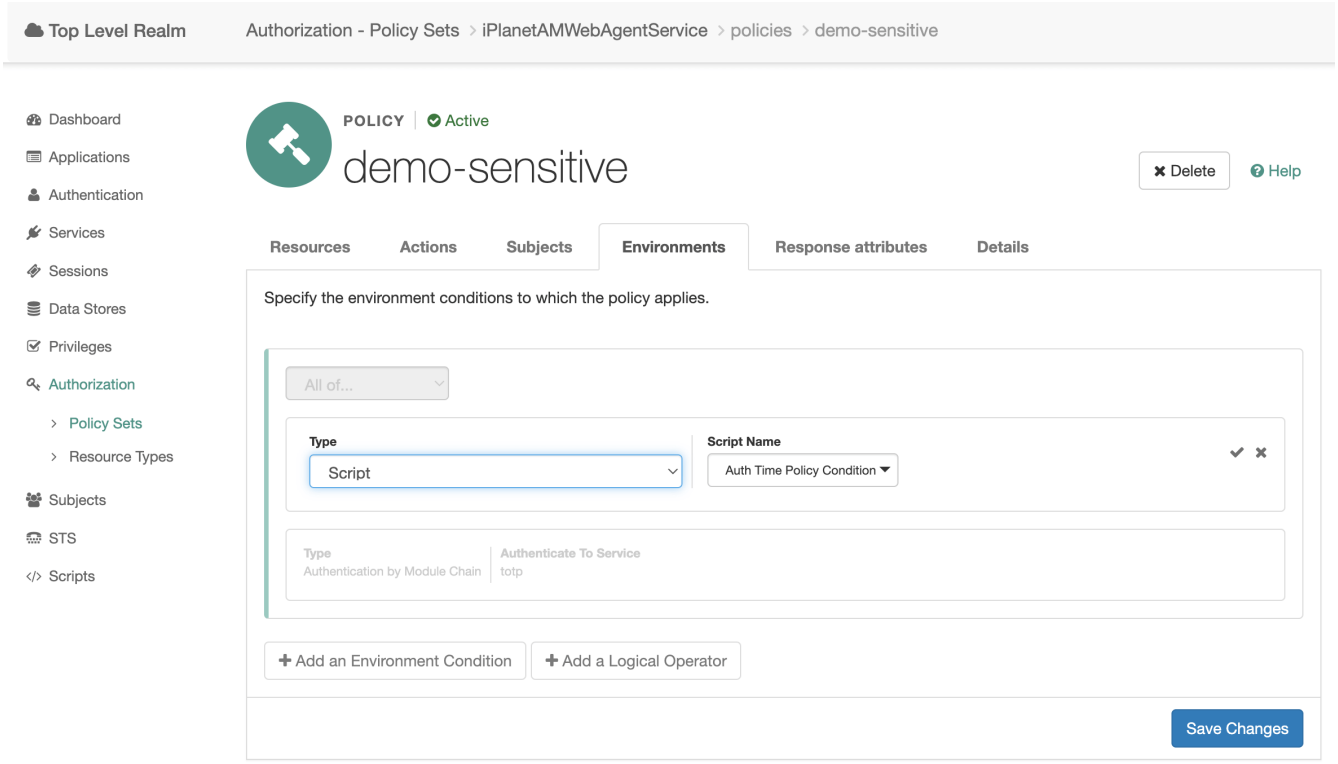
logger.warning("Auth time expired at1: " + authInstant)

def instant = Instant.parse(authInstant)
def expired = instant.plus(20, ChronoUnit.SECONDS)
if (Instant.now().compareTo(expired) > 0) {
  logger.warning("Auth time expired at: " + expired)
  authorized = false
} else {
  authorized = true
}
```

Save the policy changes.

Now let's configure the use of the script in the authorization policy. In the left menu, go to **Authorization → Policy Sets → Default Policy Set → demo-sensitive**.

On the Environments tab, add a condition with the Script type and the value Auth Time Policy Condition.



Save the changes.

Now let's configure the use of MFA for the `demo` user. This account was created when OpenAM was installed.

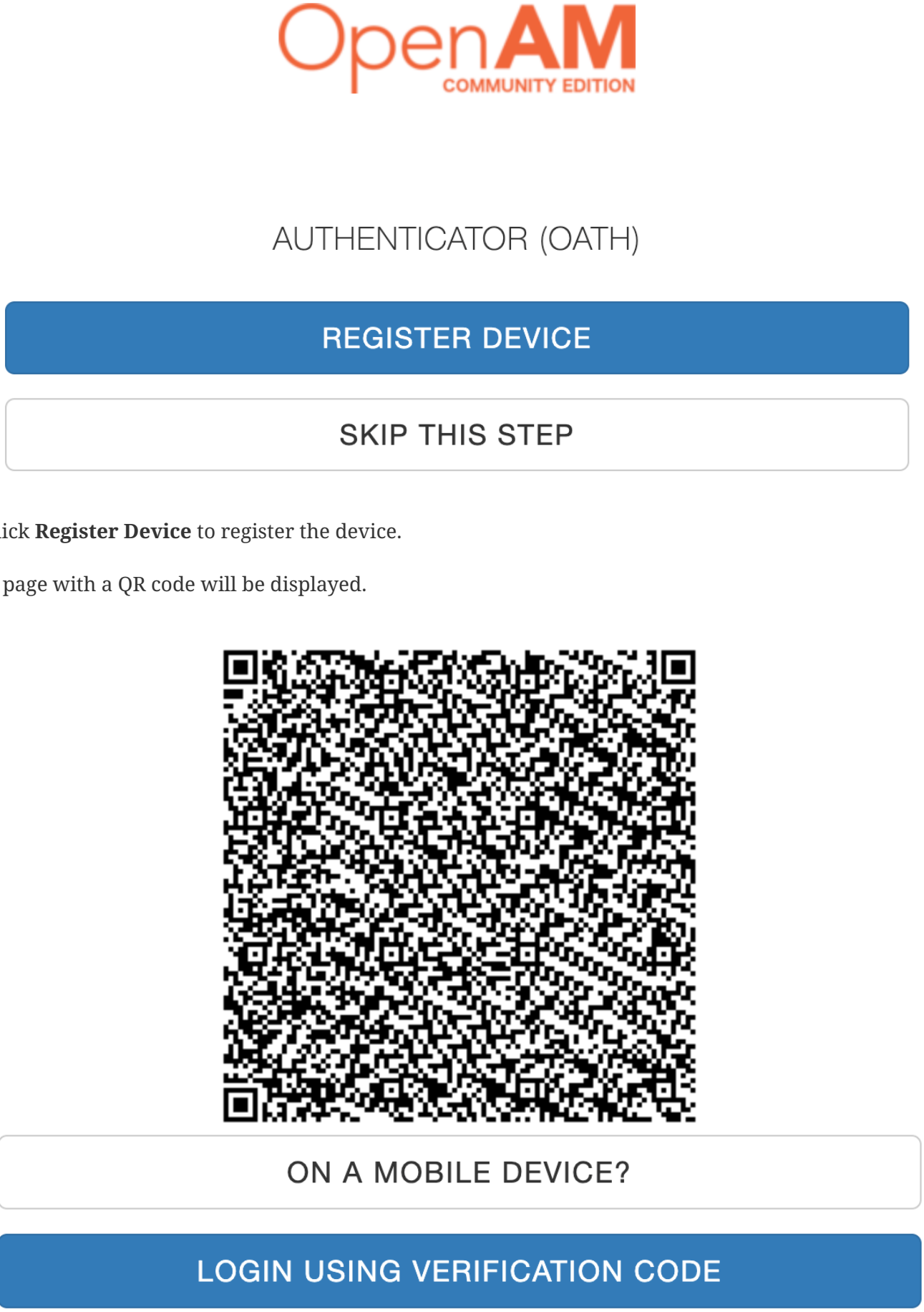
OpenAM User Setup

Log out of the admin console, or open a browser in Incognito mode and go to http://openam.example.org:8080/openam/XUI/#login

In the login and password fields, enter `demo` and `changeit` respectively. This will open the user profile.

Now, start the authentication process on the `totp` chain. For this, open the link http://openam.example.org:8080/openam/XUI/#login/&service=totp&ForceAuth=true

A window will open asking you to register a new device

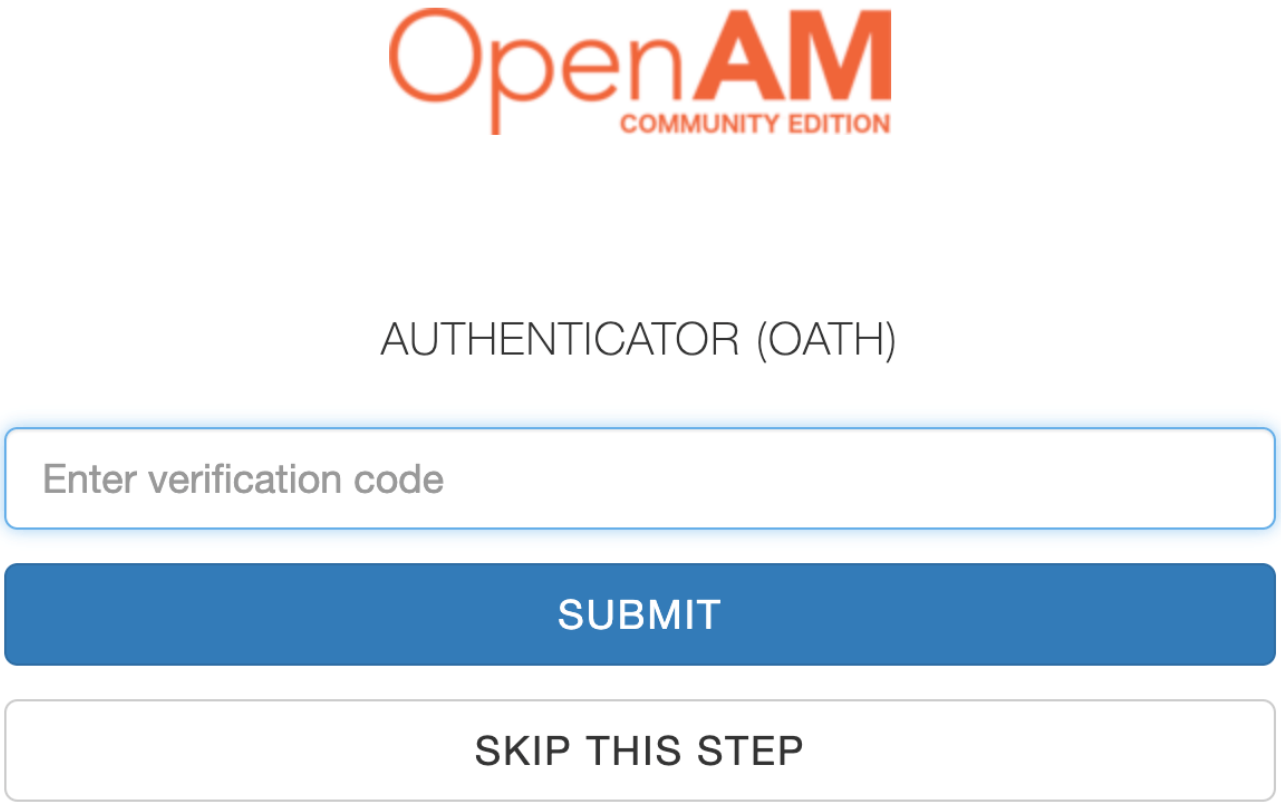


Click **Register Device** to register the device.

A page with a QR code will be displayed.

Open the authenticator app on your mobile device and scan the issued QR code in it. Click the **Login Using Verification Code** button.

Enter the code from the mobile app and click **Submit**.



`MFA` for user `demo` is configured

Configuring Authentication Token Conversion to JWT

After successful authentication, OpenAM creates a session and writes the session ID, which is a random character set, in a cookie to the browser. We will configure OpenAM so that third-party applications, such as OpenIG, can exchange the authentication token to the JWT to make it easier for third-party applications to work with.

Open the OpenAM admin console as described earlier. Select **Top Level Realm**. From the menu on the left, select **STS**. In the list that opens, create a new **Rest STS** instance. Fill in the settings

Setting	Value
Supported Token Transforms	OPENAM->OPENIDCONNECT;don't invalidate interim OpenAM session
Deployment Url Element	jwt
The id of the OpenID Connect Token Provider	https://openam.example.org/openam
Client secret	changeme
Confirm client secret	changeme

Setting	Value
The audience for issued tokens	https://openam.example.org/openam

Save the Rest STS instance conversion settings.

You can read more about installing and configuring OpenAM in the documentation: OpenAM documentation

Configure the OpenIG authorization gateway

Add the OpenIG service to the `docker-compose.yml` file

```
services:
...
  openig:
    image: openidentityplatform/openig:latest
    container_name: openig
    hostname: openig.example.org
    volumes:
      - ./openig:/usr/local/openig-config:ro
    environment:
      CATALINA_OPTS: -Dopenig.base=/usr/local/openig-config -Ddemo.app=http://demo
-app:3000 -Dopenam=http://openam.example.org:8080/openam
    ports:
      - "8081:8080"
```

Note the arguments in the `CATALINA_OPTS` environment variable:

- `openig.base` - the path to the OpenIG configuration files
- `demo.app` - URL of the demo application to which OpenIG will proxy the request
- `openam` - the OpenAM URL to which OpenIG will redirect the user to authenticate and receive the JWT.

General Settings

Create a `openig-config` folder and in it another `config` folder. In the `config` folder, create an `admin.json` file with the following contents:

```
{
    "prefix": "openig",
    "mode": "PRODUCTION"
}
```

In the same folder, create a `config.json` file.

```
{
```

```
  "heap": [
    {
      "name": "EndpointHandler",
      "type": "DispatchHandler",
      "config": {
        "bindings": [
          {
            "handler": "ClientHandler",
            "capture": "all",
            "baseURI": "${system['demo.app']}"
          }
        ]
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "STSFilter",
          "type": "ConditionalFilter",
          "config": {
            "condition": "${empty contexts.sts.issuedToken and not empty
request.cookies['iPlanetDirectoryPro'][0].value}",
            "delegate": {
              "type": "TokenTransformationFilter",
              "config": {
                "openamUri": "${system['openam']}",
                "realm": "/",
                "instance": "jwt",
                "from": "OPENAM",
                "to": "OPENIDCONNECT",
                "idToken": "${request.cookies['iPlanetDirectoryPro'][0].value}"
              }
            }
          }
        },
        {
          "name": "AuthorizationHeaderFilter",
          "type": "ConditionalFilter",
          "config": {
            "condition": "${not empty contexts.sts.issuedToken}",
            "delegate": {
              "type": "HeaderFilter",
              "config": {
                "messageType": "REQUEST",
                "remove": [
                  "Authorization",
                  "JWT"
                ],
```

```
                "add": {
                  "Authorization": [
                    "Bearer ${contexts.sts.issuedToken}"
                  ]
                }
              }
            }
          }
        },
        {
          "name": "AuthenticationRedirectionFilter",
          "type": "ConditionEnforcementFilter",
          "config": {
            "condition": "${not empty contexts.sts.issuedToken}",
            "failureHandler": {
              "type": "StaticResponseHandler",
              "config": {
                "status": 302,
                "reason": "Found",
                "headers": {
                  "Content-Type": [
                    "application/json"
                  ],
                  "Location": [

"${system['openam']}/XUI/#login&goto=${urlEncode(contexts.router.originalUri)}"
                  ]
                },
                "entity": "{ \"Redirect\":
\"${system['openam']}/XUI/#login&goto=${urlEncode(contexts.router.originalUri)}\"}"
              }
            }
          }
        }
      ],
      "handler": {
        "type": "Router",
        "name": "_router",
        "capture": "all"
      }
    }
  }
}
```

The `config.json` file defines a filter chain for each request to the demo application:

- `STSFilter` - if the HTTP request contains a cookie from OpenAM, the filter gets a JWT from this cookie, which is written to the context for further use
- `AuthorizationHeaderFilter` - adds the JWT received from OpenAM to the request in the `Authorization` header for use in the protected application

- `AuthenticationRedirectionFilter` - if the JWT is not present in the request context, redirects the user to authenticate with OpenAM.

An `EndpointHandler` handler is defined in the `heap` object that proxies requests in OpenIG to the demo application.

Configure Routes to the Demo Application

In the `config` folder, create a `routes` folder and add a `10-home.json` route

```
{
  "name": "${matches(request.uri.path, '^/$')}",
  "condition": "${matches(request.uri.path, '^/$')}",
  "monitor": true,
  "timer": true,
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [],
      "handler": "EndpointHandler"
    }
  },
  "heap": [

  ]
}
```

The route simply proxies requests to the demo application using the `EndpointHandler` defined in the `config.json` configuration file.

We will add the route to a URL with sensitive information of the demo application. We will then configure a filter for the route so that it uses the authorization policy from OpenAM.

Add the `20-sensitive.json` route to the `routes` folder

```
{
  "name": "${matches(request.uri.path, '^/sensitive')}",
  "condition": "${matches(request.uri.path, '^/sensitive')}",
  "monitor": true,
  "timer": true,
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "MFAPEPFilter",
          "type": "PolicyEnforcementFilter",
          "config": {
            "openamUrl": "${system['openam']}",
```

```
            "pepUsername": "amadmin",
            "pepPassword": "ampassword",
            "ssoTokenSubject": "${request.cookies['iPlanetDirectoryPro'][0].value}",
            "failureHandler": {
              "type": "StaticResponseHandler",
              "config": {
                "status": 403,
                "headers": {
                  "Content-Type": [
                    "application/json"
                  ]
                },
                "entity": "{ \"attributes\":
\"${system['openam']}/XUI/#login&service=totp&ForceAuth=true&goto=${urlEncode(contexts
.router.originalUri)}\"}"
              }
            }
          },
          "handler": "ClientHandler"
        }
      ],
      "handler": "EndpointHandler"
    }
  },
  "heap": []
}
```

The route uses `MFAPEPFilter` to get the result of the authorization policy from OpenAM. And, if the policy check fails, redirects to authentication with a one-time code.

You can read more about installing and configuring OpenIG in the documentation: https://doc.openidentityplatform.org/openig/

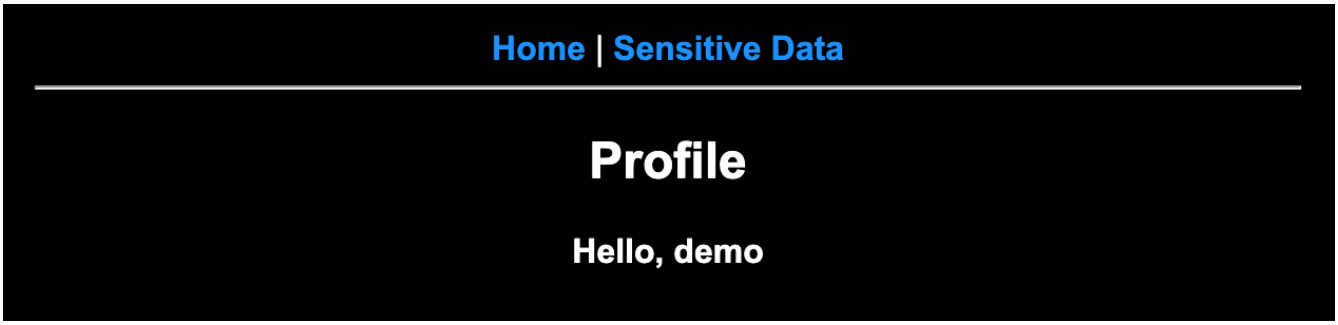
Test the Solution

Start OpenIG with the `docker compose ui openig` command.

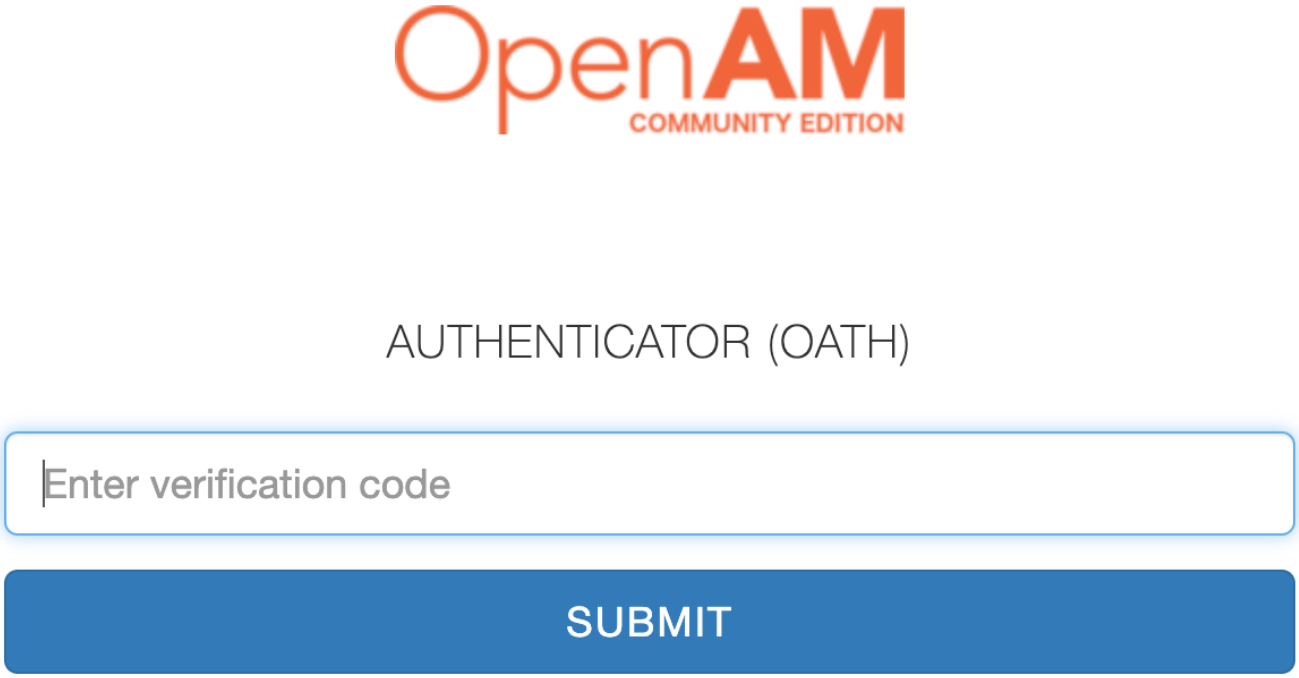
Log out of OpenAM if you are still authenticated.

Open the URL of the OpenIG protected demo application: http://openig.example.org:8081/. You will be redirected to authenticate to OpenAM. Enter the test user login and password: `demo` and `changeit`.

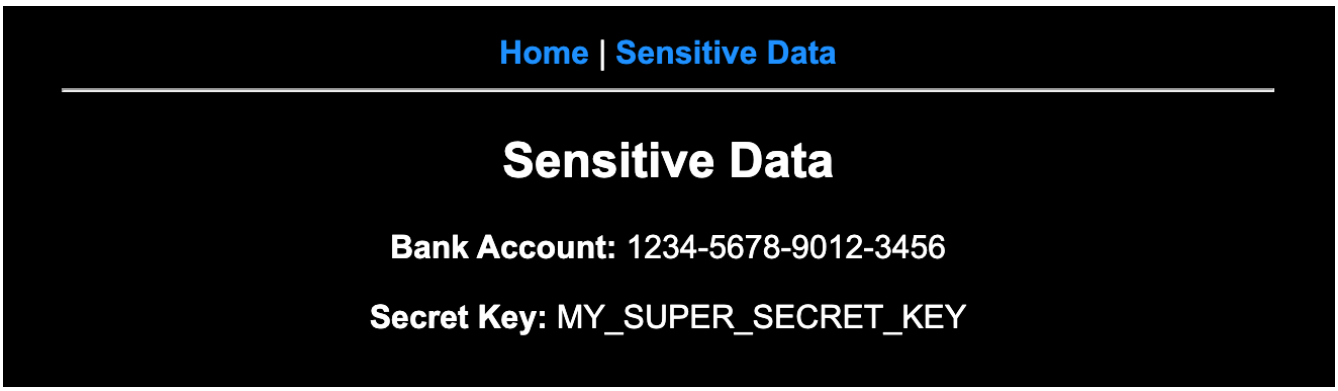
After authentication, you will be redirected to the main screen of the demo application



Click the **Sensitive data** link. You will be redirected to additional authentication with a one-time code in OpenAM.



Enter the code from the authenticator application and click Submit. If successful, you will be redirected back to the page with sensitive data



Wait 30 seconds and reload the page. You will be redirected to authentication with a one-time code again.

OpenAM vs Keycloak

General Information

	OpenAM	Keycloak
Initial release	2008	2014
Maintainer	Open Identity Platform Community	Red Hat
Current version	16.0.4	26.4.7
Release and patches	Regular	Regular
Open Source	✓	✓
Programming Language	Java	Java
Supported Java version	11+ LTS	17+ LTS
License	CDDL	Apache License 2.0
Multiple languages supported	✓	✓

Single Sign On (SSO) and Federation

	OpenAM	Keycloak
Applications deployment behind an auth gateway	✓ doc	✓ doc
API gateway control	✓ doc	✗✗
Apache Web Server Policy Agent	✓ doc	✗✗
IIS Policy Agent	✓ doc	✗✗
J2EE Policy Agent (for Tomcat, Jetty, etc)	✓ doc	✗✗
OAuth 2.0/OIDC federation	✓ Google example	✓ GitHub example
SAMLv2 federation	✓ WordPress example	✓ doc

Authentication

Supported authentication methods:

Authentication method	OpenAM	Keycloak
Login and password authentication	✓ doc	✓ doc

Authentication method	OpenAM	Keycloak
Microsoft Active Directory authentication	□ doc	□ doc
Authentication for demo access	□ doc	□□
Adaptive authentication	□ doc	□□
Authentication in an LDAPv3-compatible directory	□ doc	□ doc
Persistent cookie authentication	□ doc	□ doc
RDBMS authentication	□ doc	□ doc
Self-registration	□ doc	□ doc
HTTP Header enrichment authentication	□ doc	□□
Windows NT authentication	□ doc	□□
OAuth 2.0/OIDC authentication	□ doc	□ doc
Kerberos authentication	□ doc	□ doc
OIDC id_token authentication	□ doc	□□
RADIUS authentication	□ doc	□□
HOTP via SMS or email	□ doc	□□
One time password with HOTP or TOTP authentication	□ doc	□ doc
Custom scripted authentication provider	□ doc	□□
SAMLv2 authentication	□ doc	□ doc
ReCaptcha	□ doc	□ doc
QR-code authentication	□ doc	□□
NTLM authentication	□ doc	□□
Docker HTTP Basic Authentication	□□	□ doc
HTTP Basic Authentication	□ doc	□ doc
Recovery codes authentication	□ doc	□ doc
WebAuthn	□ doc	□ doc
X509 certificate authentication	□ doc	□ doc
Custom authentication provider	□ doc	□ doc

Isolation (realms)

Realm support for isolation of identities and authentication processes

	OpenAM	Keycloak
Realms support	□ doc	□ doc
Realm hierarchy	□	□□

Interfaces

Interface	OpenAM	Keycloak
Administrator GUI	□	□
Admin REST API	□	□
Admin UI customization	□□	□ doc
Authentication GUI	□	□
Authentication GUI customization	□ doc	□ doc
Authentication REST API	□ doc	□□
Authentication XML-RPC API	□ doc	□□

Authentication Sessions

	OpenAM	Keycloak
Stateful	Random session ID	JWT
Stateless	JWT	JWT
REST Security Token Service	□	□
SOAP Security Token Service	□	□□

Identity Repositories

Repository type	OpenAM	Keycloak
LDAP (OpenDJ, OpenLDAP, etc.)	□	□
Active Directory	□	□
Apache Cassandra	□	□□
MariaDB Server	□	□
Microsoft SQL Server	□	□
MySQL	□	□
Oracle Database	□	□
PostgreSQL	□	□
Flat file	□	□□

Repository type	OpenAM	Keycloak
Custom identity repository	□ doc	□ doc

Audit logging and Monitoring

	OpenAM	Keycloak
Audit Logging	□ doc	□ doc
HTTP-based Monitoring	□ doc	□ doc
SNMP Monitoring	□ doc	□□
JMX Monitoring	□ doc	□□

Useful Links

OpenAM:

- Official website
- GitHub repository
- Documentation

Keycloak:

- Official website
- GitHub repository
- Documentation

Passwordless Authentication Methods

- Intro
- Passwordless Authenticaion Methods
- Conclusion

Intro

Nowadays, people use more and more different online services in everyday life, and each service requires authentication. So, for each service, you need to remember your username and password. And, even the login for each service can be the same (for example, the email), then the password must be unique for each service. It is much better to have a complex password, i.e: have a length of 8 characters and more, contain uppercase and lowercase letters, numbers, and special characters.

Of course, it is impossible to remember the complex password for each service, so users either use simple passwords or use the same password for each service. Some users even write their own passwords on a piece of paper and put it under the keyboard (sic!). Of course, it compromises user accounts.

Passwordless authentication could solve this problem. In the following article, I will try to consider main passwordless authentication methods, as well as their advantages and disadvantages.

There are the following passwordless authentication methods:

- One-time link sent to the e-mail
- One-time password sent by SMS or Push-notification
- HOTP and TOTP (HMAC and Time-based one-time password)
- Persistent Cookie
- Third-party Identity provider (for example, log in via Facebook or via Google)
- USB token device
- Mobile application with biometric authentication.

Passwordless Authenticaion Methods

One-time Authentication Link Sent to the E-mail

During authentication, the user enters his email, the service generates a one-time link and sends it to the specified email. Then the user must open the mail application, receive an email from the service, and follow the link.

Pros:

- Low cost - sending e-mail is almost free

Cons:

- The need for the user to open an additional email client application
- If the attacker has access to the user's e-mail, then authentication can be compromised.
- There is a risk of receiving an email with a phishing link to enter a malicious resource

One-time password via SMS or Push

The most widely used passwordless authentication method. During authentication, the user enters his phone number, then he receives an SMS or pushes notification with a one-time confirmation code, which has a limited validity period. The user enters the received one-time code in the service and authenticates.

Pros:

- Relative reliability - to fake a SIM card or steal a phone seems to be a rather non-trivial task for an attacker. In addition, the mobile phone can determine the location of the attacker.

Cons:

- Users must manually enter the code from an SMS, every time they authenticate, which can be annoying.
- For receiving a push notification users should install a mobile application.

HMAC and Time-based one-time password

HMAC-based one-time password (HOTP) is generating a one-time password algorithm based on authentication attempts and a shared secret between user server and client. A time-based one-time password - is an improvement of HOTP and generates passwords based on system time. These algorithms generate passwords on both a server and a client each time user authenticates the system.

Pros:

- You can use third-party trusted software to implement this algorithm (for example Google authenticator)

Cons:

- For TOTP there is a need to synchronize time between server and client
- The shared secret can be stolen and attackers can generate their own TOTP values to authenticate

Persistent Cookie

One of the simplest and widely used way to authenticate without a password. After authentication, a special cookie is set in the user's browser, which is then used to authenticate the user.

Pros:

- Further authentications do not require entering any data from the user

Cons:

- Works on a single device (browser)
- If an attacker steals an users cookie, he could gain access to the user's account
- The cookie should expire. When the cookie expires, the user should authenticate again.

Using third-party Identity Provides (via Social Networks)

During authentication, the user is prompted to authenticate using an existing account of a third-party Identity Provider (Google, Facebook, LinkedIn)

Pros:

- Very easy to use, if the user has already authenticated to the identity provider.

Cons:

- If the user lost his Identity Provider account, access to the service can also be lost.
- Users may not have profiles in the Identity Providers list supported by the service.

USB Token Device

Users can be authenticated using a USB token device. There is a cryptographic key, that uniquely identifies the device holder.

Pros:

- High security - it is almost not possible to forge the token

Cons:

- The user need to carry an extra device
- Sometimes, there is a need to install special software to authenticate
- The token device can be lost or stolen

Mobile Phone Biometrics

While authenticating, the user receives notifications on his mobile phone application, asking for confirmation via fingerprint, face recognition, and so on.

Pros:

- High security, because mobile phone manufacturers are focusing on mobile phone security and protecting them from unauthorized access.
- Almost everyone has a mobile phone

Cons:

- User need to install and setup additional application on his phone

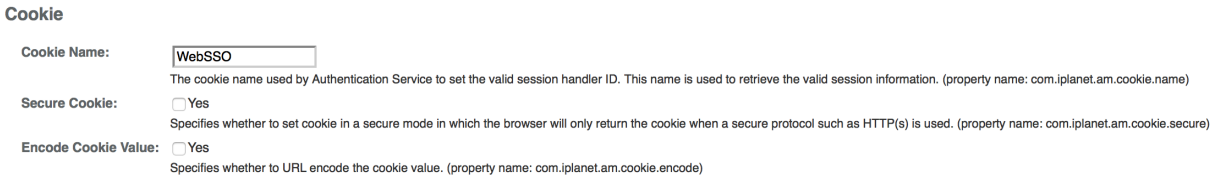
Conclusion

All of the methods above have advantages and disadvantages. But for better user experience, is the usage of a combination of several methods. For example, to provide the ability to authenticate using OAuth or OpenID and a saved cookie. Also, a promising and secure approach is authentication using mobile phone biometrics, such as fingerprint or face recognition authentication.

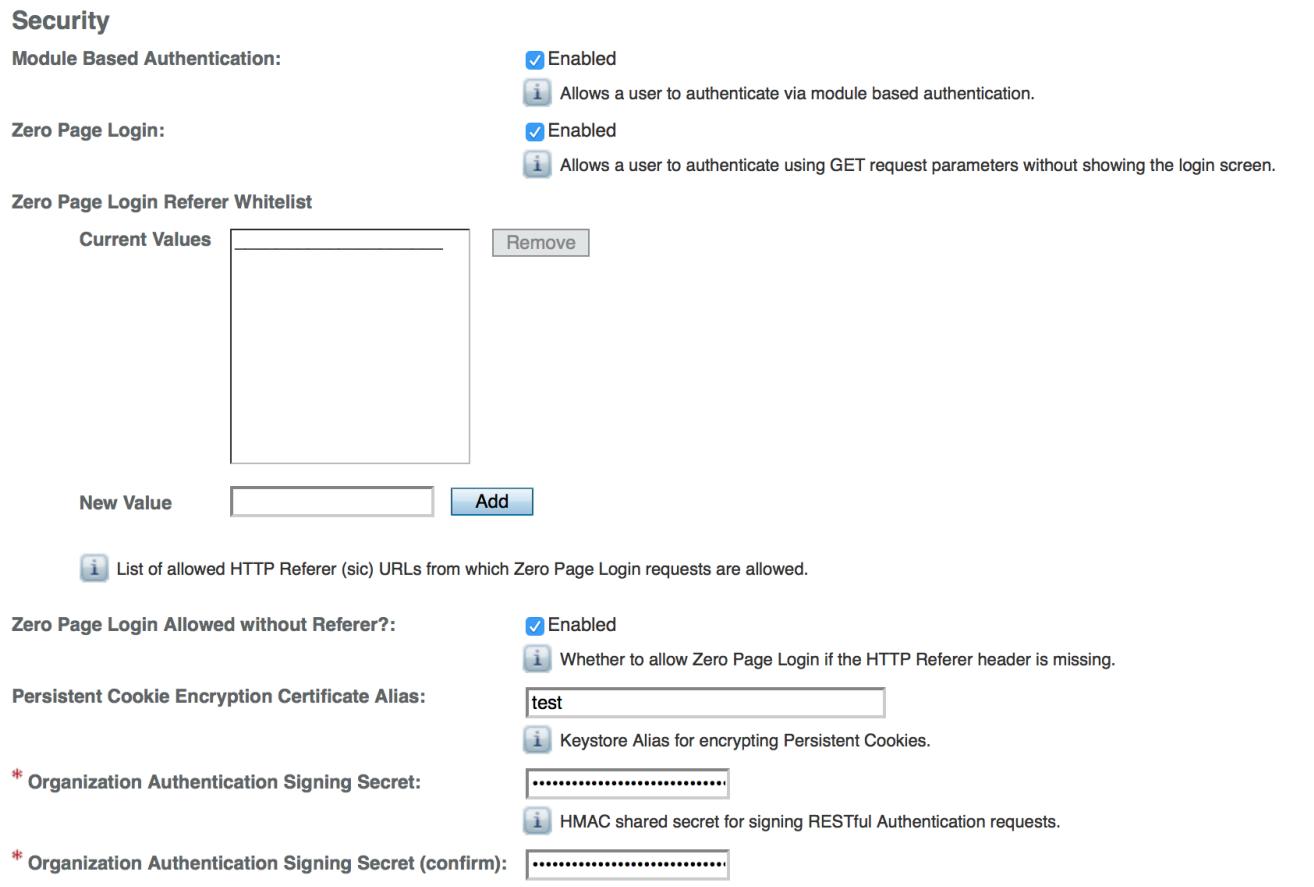
Persistent Cookie module

The Persistent Cookie module supports the configuration of cookie lifetimes based on requests and a maximum time. Note that by default, the persistent cookie is called **session-jwt**.

> If Secure Cookie is enabled Configuration > Servers and Sites > Default Server Settings > **Security** > **Cookie**, the Persistent Cookie module only works over HTTPS.

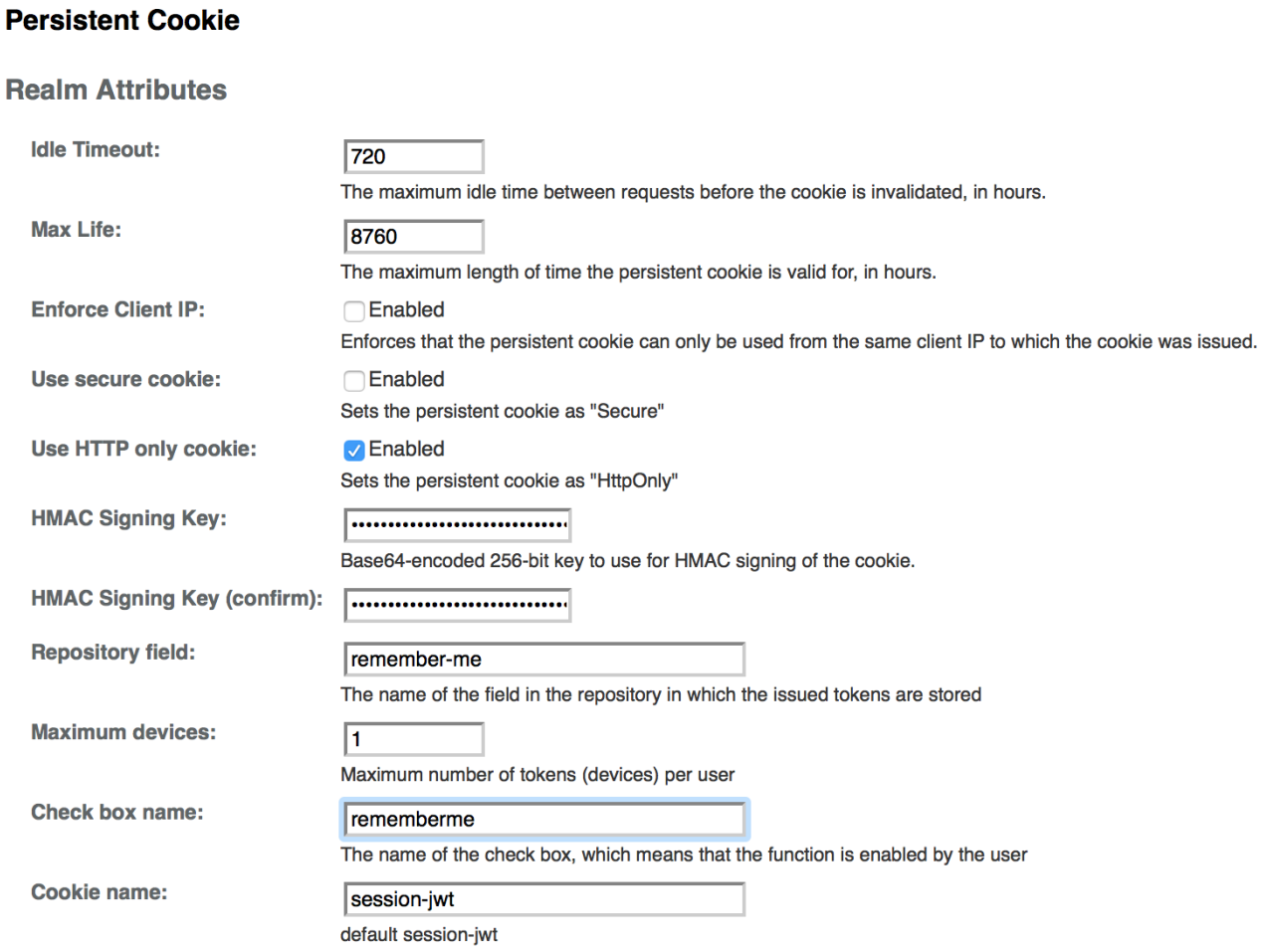


Before you begin, make sure a public key alias is defined in AM. The Persistent Cookie module encrypts a JSON Web Token (JWT) using a public key from the AM keystore. The keystore must be configured under Realms > **Realm Name** > Authentication > All Core Settings > Security > **Persistent Cookie Encryption Certificate Alias**. If the keystore changes and the default test key is no longer present, the public key alias must be updated to reflect the change, otherwise the module will fail. Similarly, in multi-instance deployments, the keypair must be available on all AM instances.



To configure the Persistent Cookie module globally in the AM console, navigate to Configuration > Authentication, and then click **Persistent Cookie**. In the window that appears you should see the

following attributes:

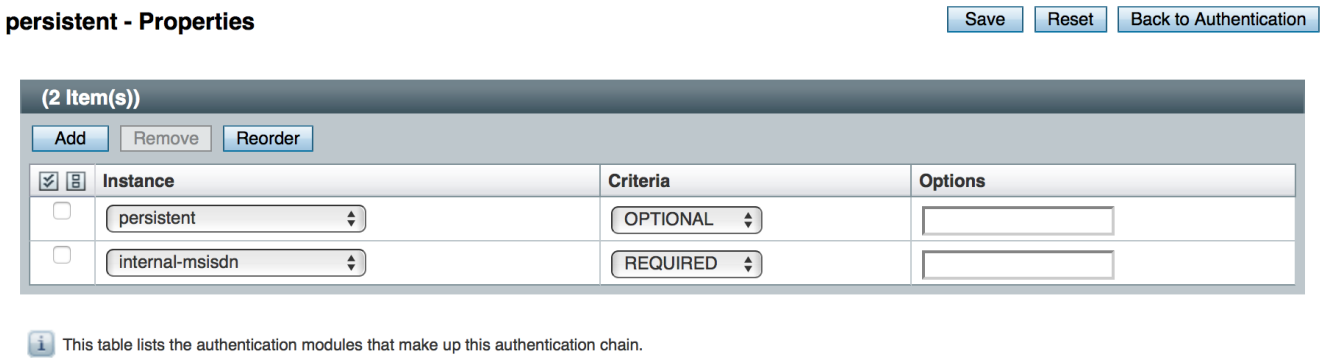


> For example, to generate an HMAC signing key, run the following:
>
> ```
> openssl rand -base64 32
> ```
>
> or
>
> ```
> cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 32 | head -n 1|base64
> ```
>
> Default: a random 256-bit secret key.

When the Persistent Cookie module enforces the client IP address, and AM lies behind a load balancer or proxy layer, configure the load balancer or proxy to send the address by using the **X-Forwarded-For** header, and configure OpenAM to consume and forward the header as necessary. For details, see Handling HTTP Request Headers in the Installation Guide.

The Persistent Cookie module belongs with a second module in an Authentication Chaining. To see how this works, navigate to Realms > **Realm Name** > Authentication > Authentication Chaining. Create a new chain and add modules as shown in the figure. The following example shows how a Persistent Cookie module is sufficient. If the persistent cookie does not yet exist, authentication

relies on LDAP:



You should now be able to authenticate automatically, as long as the cookie exists for the associated domain.

ssoadm service name: iPlanetAMAuthPersistentCookieService

Idle Timeout

Specifies the maximum idle time between requests in hours. If that time is exceeded, the cookie is no longer valid.

ssoadm attribute: **openam-auth-persistent-cookie-idle-time**

Max Life

Specifies the maximum life of the cookie in hours.

ssoadm attribute: **openam-auth-persistent-cookie-max-life**

Enforce Client IP

When enabled, enforces that the persistent cookie can only be used from the same client IP to which the cookie was issued.

ssoadm attribute: **openam-auth-persistent-cookie-enforce-ip**

Use secure cookie

When enabled, adds the "Secure" attribute to the persistent cookie.

ssoadm attribute: **openam-auth-persistent-cookie-secure-cookie**

Use HTTP only cookie

When enabled, adds the HttpOnly attribute to the persistent cookie.

ssoadm attribute: **openam-auth-persistent-cookie-http-only-cookie**

HMAC Signing Key

Specifies a key to use for HMAC signing of the persistent cookie. Values must be base64-encoded and at least 256 bits (32 bytes) long.

ssoadm attribute: **openam-auth-persistent-cookie-hmac-key**

Check box name

The name of the check box, which means that the function is enabled by the user

ssoadm attribute: **openam-auth-persistent-cookie-input**

Repository field

The name of the field in the repository in which the issued tokens are stored

ssoadm attribute: **openam-auth-persistent-cookie-field**

Maximum devices

Maximum number of tokens (devices) per user

ssoadm attribute: **openam-auth-persistent-cookie-field-max**

Cookie name

default session-jwt

ssoadm attribute: **openam-auth-persistent-cookie-name**

other Authentication modules

Quick Start Guide

OpenAM Getting Started

> OpenAM Getting Started Guide

OpenAM 13.5 Getting Started

> https://github.com/OpenIdentityPlatform/OpenAM/wiki/old/OpenAM-13.5-Getting-Started.pdf

Preparations

Install Docker

Install Docker for your platform, if you did not do this before, from https://docs.docker.com/install/#supported-platforms.

Prepare Hosts File

First, you need to add to your *hosts* file the local IP and FQDN mappings. *openam.example.org* - for OpenAm and *example.org* for Apache Http Server. Your hosts file should contain the following line:

```
127.0.0.1    openam.example.org example.org
```

To add the above line to the hosts file on Windows:

1. Press the Windows key.
2. Type Notepad in the search field.
3. In the search results, right-click Notepad and select Run as administrator.
4. From Notepad, open the following file: c:\Windows\System32\Drivers\etc\hosts.
5. Make the necessary changes to the file.
6. Select File > Save to save your changes

OpenAM Configuration

Running OpenAM Image

Create Docker network for OpenAM

```
docker network create openam-quickstart
```

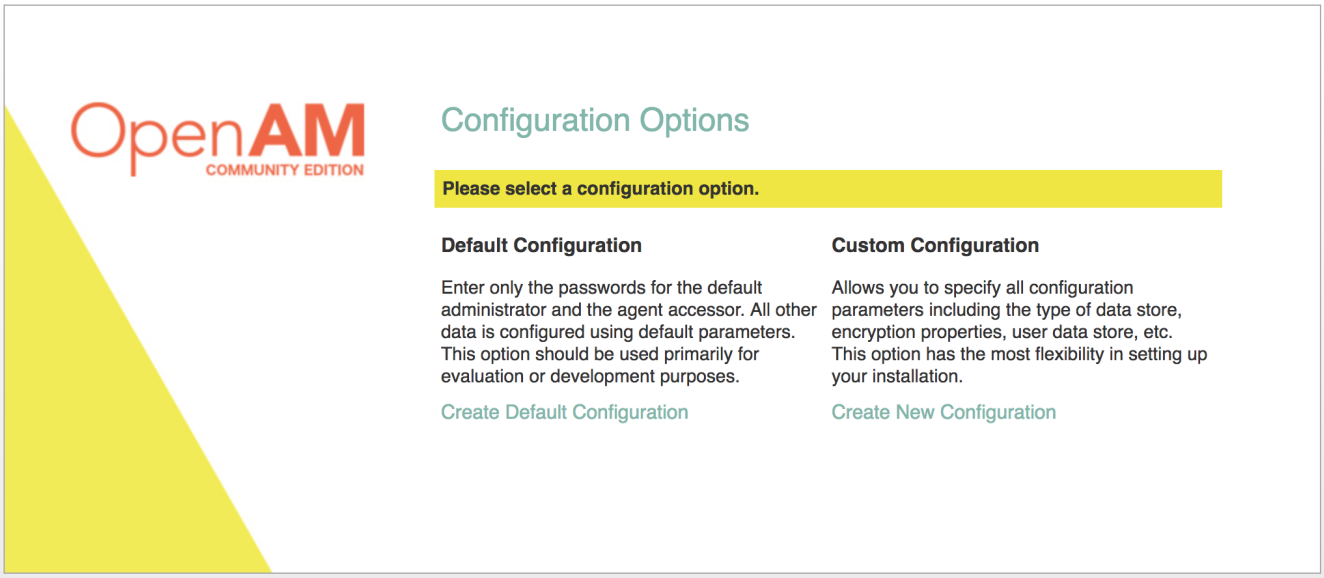
Run OpenAM image

```
docker run -h openam.example.org -p 8080:8080 --network openam-quickstart --name
openam openidentityplatform/openam
```

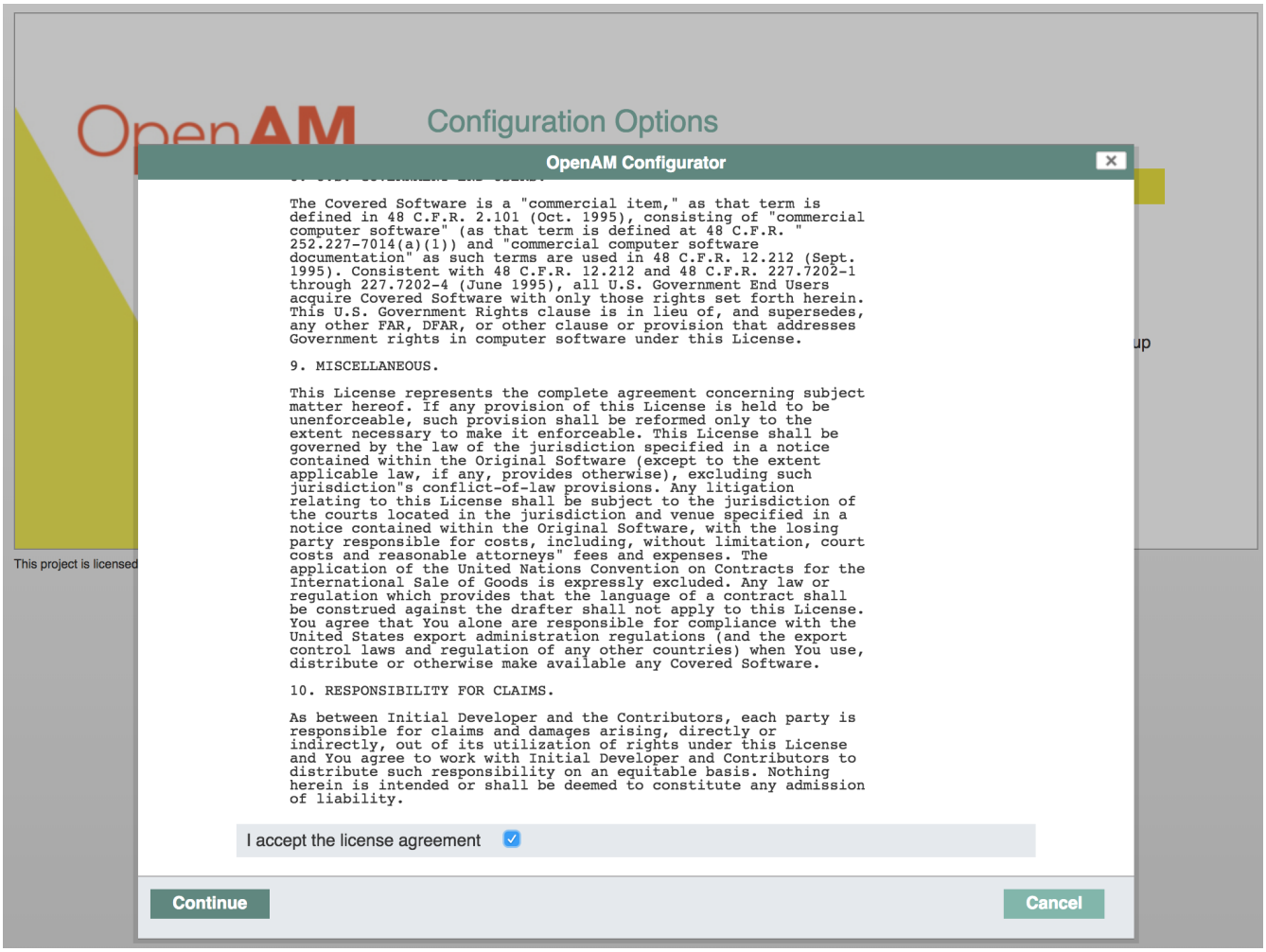
Change the name in the above command if you run into errors saying the container is already in use.

Basic OpenAM Setup

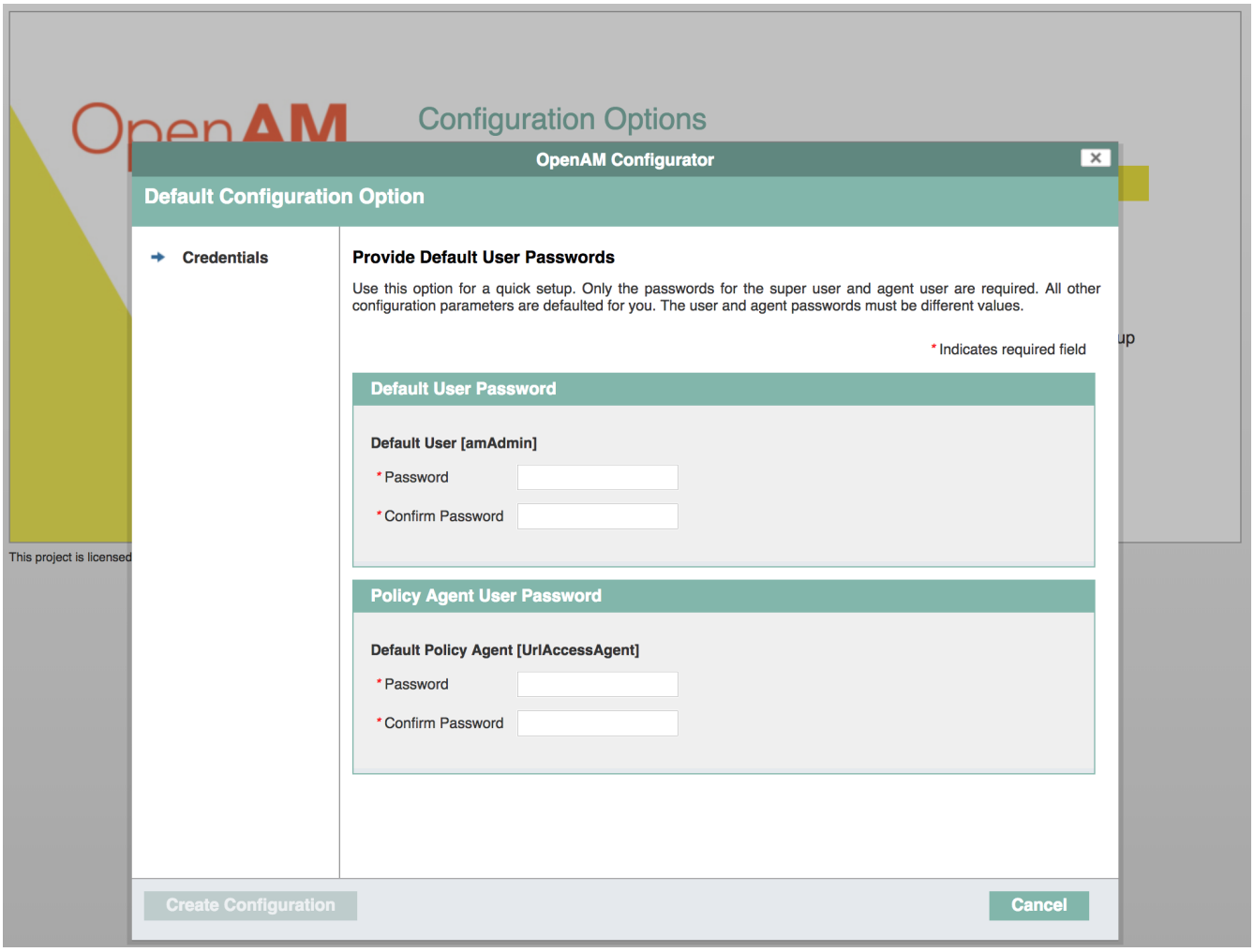
Open your browser, goto url http://openam.example.org:8080/openam.



Click **Create Default Configuration**.



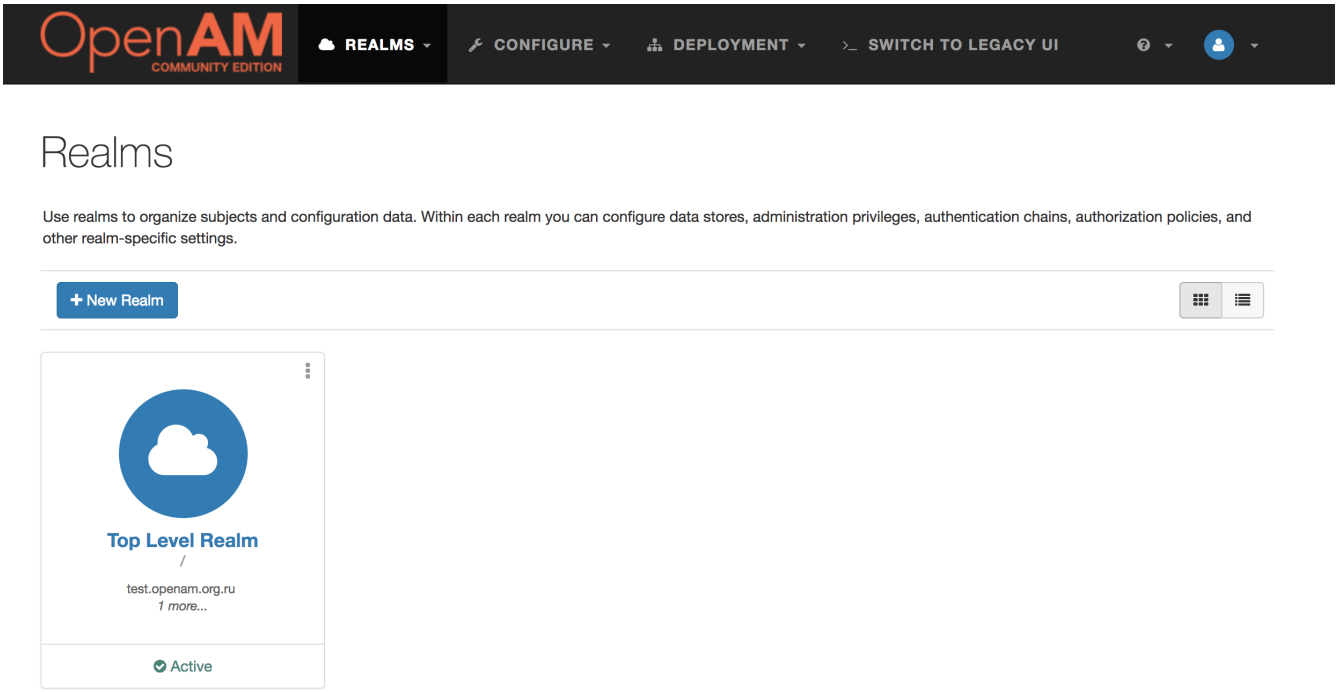
Accept License Agreement



Set password for default admin user and policy agent

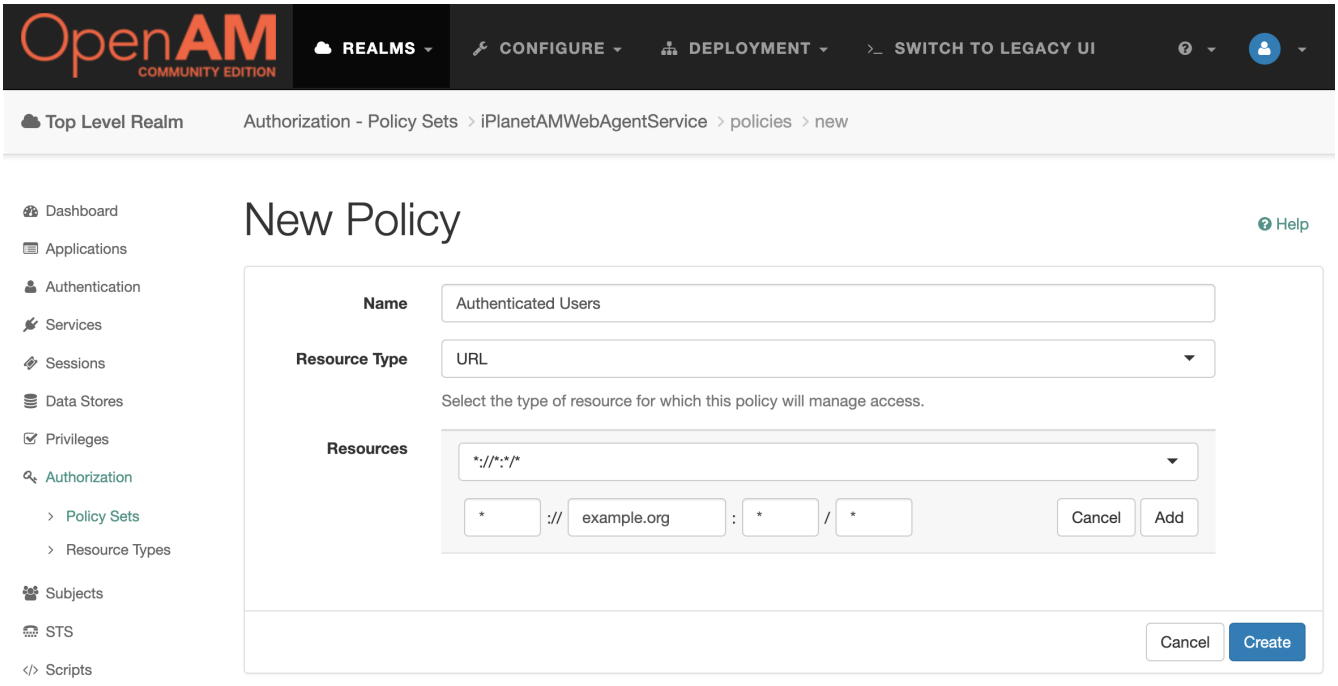
Press **Create Configuration**. After configuration successfully created, press **Proceed to Login** or open http://openam.example.com:8080/openam/console link in browser.

Policy Configuration

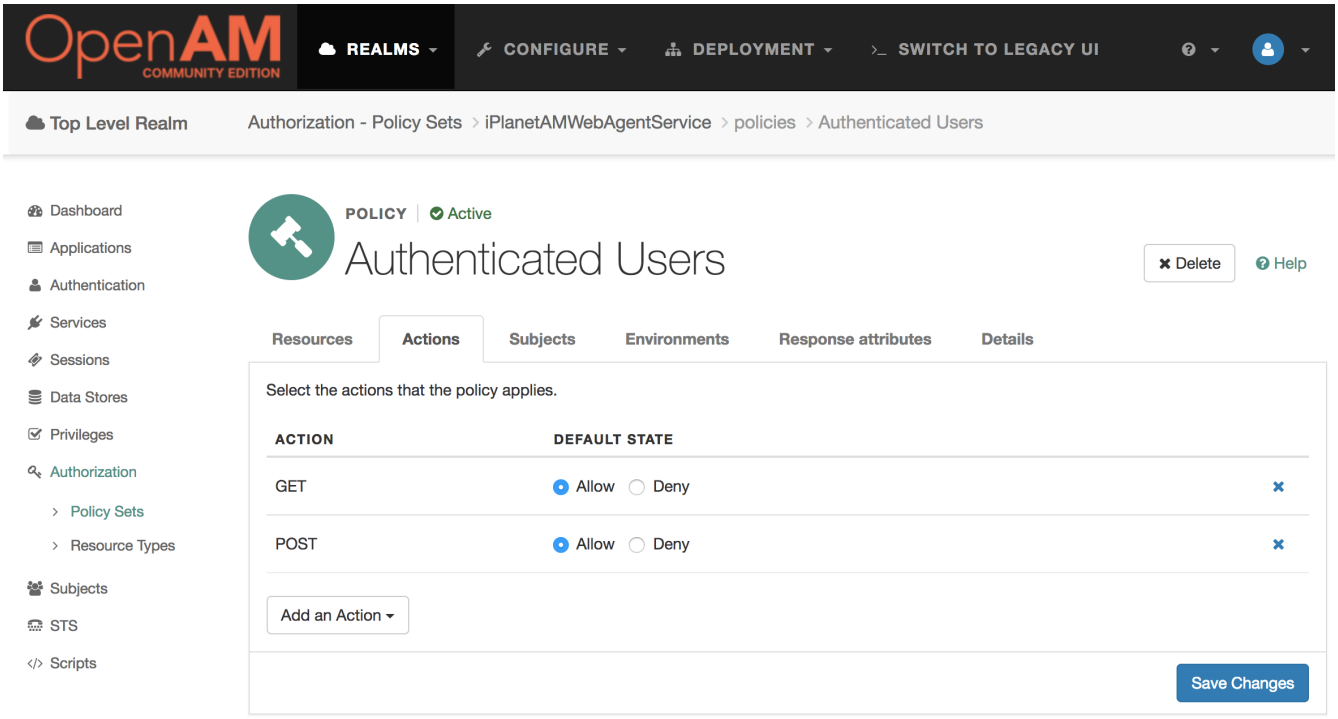


In administration console select realm, then go to **Authorization -> Policy Sets**, select **Default Policy Set** and add new Policy

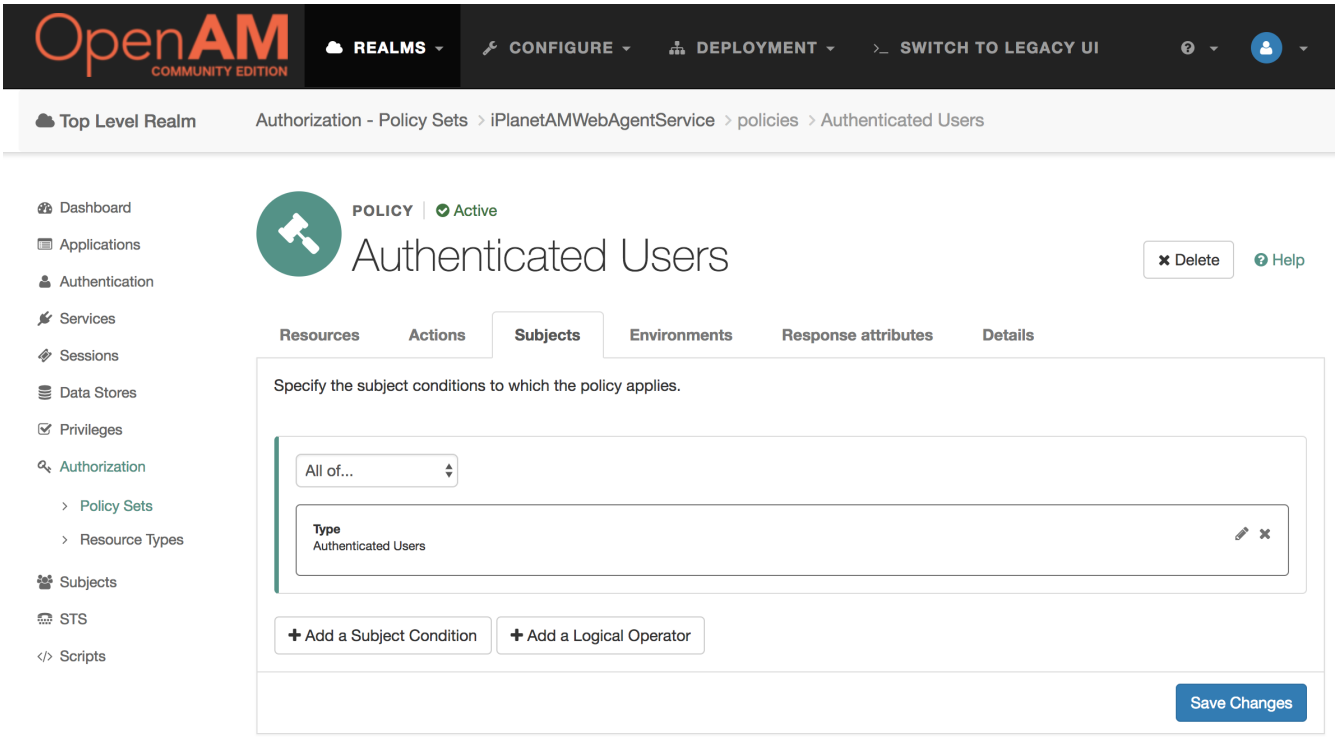
Set Policy Name as you wish, Resource Type set URL, and add new Resource **://example.org:*/** and click **Create** to save new policy.



On new policy settings select Actions tab and add two actions **GET** and **POST**



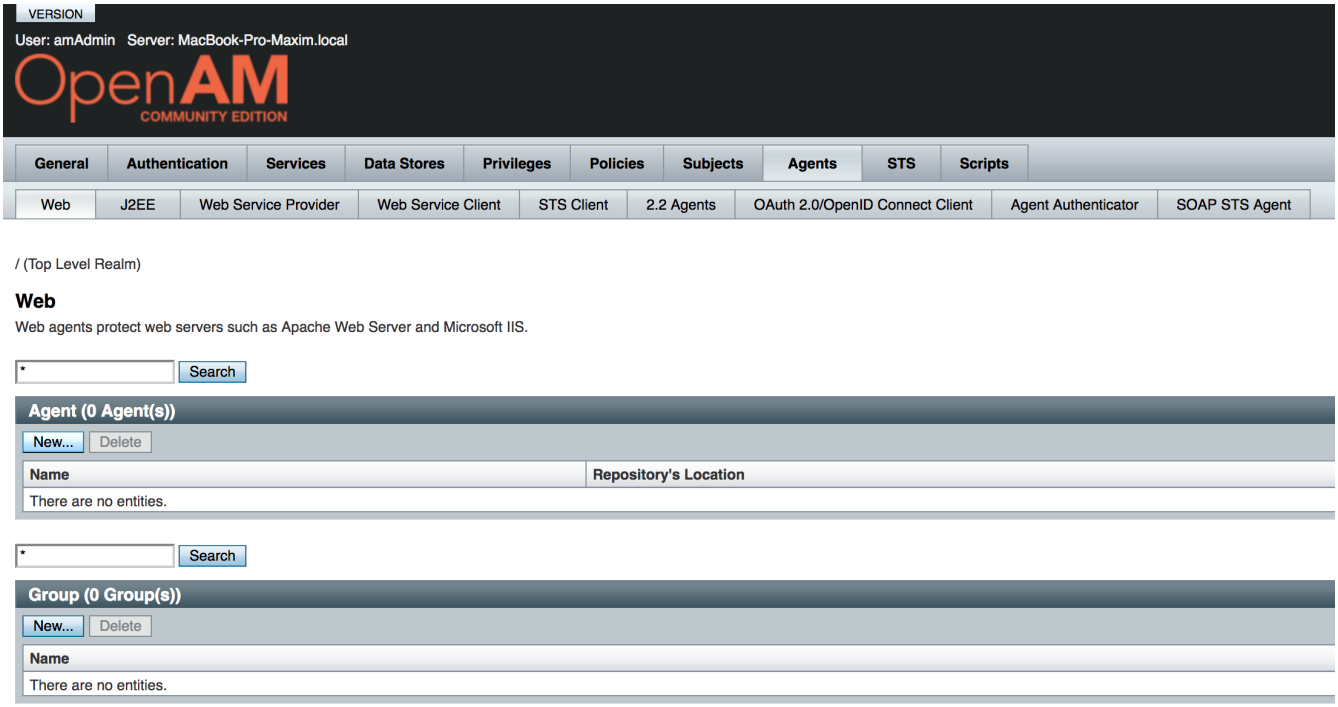
Then select Subjects tab and set type to **Authenticated Users**



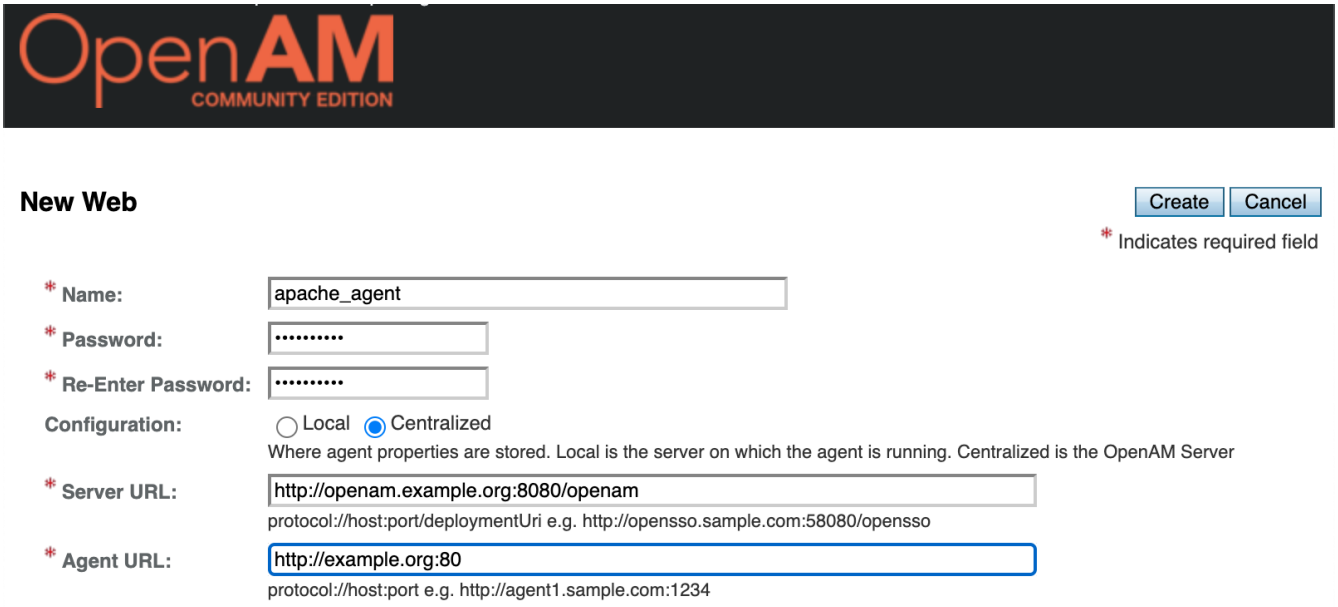
Click **Save Changes** to save your policy

Agent Configuration

Then, in left menu, go to **Applications -> Web Agents** and create new Agent



Set name as you wish, for example `apache_agent`, set agent password.



Server URL set http://openam.example.org:8080/openam

Agent URL set http://example.org:80

Click **Create** to save new Web Agent

Click **General** tab to return to main menu.

Cookie Domain Configuration

Navigate to **Configure -> Global Services -> Platform -> Cookie Domain**.

Set cookie domain to *.example.org*, save your settings.

Apache HTTP Server Configuration

Create Dockerfile in your */home/user/openam-quickstart/apache/* folder with following content

```
FROM httpd:2.4.34

ENV PA_PASSWORD password

RUN echo "deb [trusted=yes] http://archive.kernel.org/debian-archive/debian/ jessie
main" >> /etc/apt/sources.list

RUN apt-get update || true

RUN apt-get install -y curl unzip

RUN curl -L -o /tmp/Apache_v24_Linux_64bit_4.1.1.zip
https://github.com/OpenIdentityPlatform/OpenAM-Web-
Agents/releases/download/4.1.1/Apache_v24_Linux_64bit_4.1.1.zip

RUN unzip /tmp/Apache_v24_Linux_64bit_4.1.1.zip -d /usr/

RUN rm /tmp/Apache_v24_Linux_64bit_4.1.1.zip

RUN echo $PA_PASSWORD > /tmp/pwd.txt

RUN cat /tmp/pwd.txt

RUN cat /etc/issue

RUN /usr/web_agents/apache24_agent/bin/agentadmin --s
"/usr/local/apache2/conf/httpd.conf" "http://openam.example.org:8080/openam"
"http://example.org:80" "/" "apache_agent" "/tmp/pwd.txt" --acceptLicence
--changeOwner
```

Set ENV PA_PASSWORD as you previously set for your WebAgent in OpenAm

Build Apache Docker image

```
docker build --network=host -t apache_agent -f /home/user/openam-
quickstart/apache/Dockerfile /home/user/openam-quickstart/apache/
```

And then run image

```
docker run -it --name apache_agent -p 80:80 -h example.org --shm-size 2G --network
openam-quickstart  apache_agent
```

Change apache_agent to httpd in the above command. apache_agent is not available on docker hub.

Open in browser link http://example.org, and you will be redirected to OpenAM Authentication. After authentication you should see default Apache HTTP Server page



It works!

SAML Authentication in WordPress via OpenAM

Introduction

SAMLv2, despite its venerable age, is the de facto standard for SSO (Single Sign On) in an enterprise environment. And in the following article, we will set up a SAML login to WordPress using OpenAM authentication. That is, when authenticating to WordPress, users will be redirected to OpenAM and, after authenticating to OpenAM will be automatically authenticated to WordPress. Given OpenAM's flexibility in customizing authentication methods, you can customize WordPress login not only by login and password but also, for example, using built-in Windows authentication (NTLMv2 or Kerberos), adding a second authentication factor (biometrics or one-time password) or even by scanning a QR code in a special mobile app.

Instead of WordPress, it can be almost any application that supports SAMLv2. The OpenAM configuration will be nearly identical. Only the settings for the application itself will differ.

Some terminology

Service Provider (SP) - The application whose services users will use after authentication.

Identity Provider (IdP) - the application that authenticates users and provides the Service Provider with information about the authenticated accounts.

In our case, the Identity Provider is OpenAM and the Service Provider is WordPress.

Test environment

For demonstration purposes, all applications will be run in Docker containers via the Docker Compose utility.

Add the OpenAM and WordPress hostnames `127.0.0.1 openam.example.org wordpress.example.org` to the `hosts` file.

On systems running Windows, the `hosts` file is `C:\Windows\System32\drivers\etc\hosts`. On Linux and Mac systems, `/etc/hosts`.

Create a `docker-compose.yml` file with the following contents:

```
services:
  openam:
    image: openidentityplatform/openam:latest
    restart: always
    hostname: openam.example.org
    ports:
      - "8080:8080"
```

```
    volumes:
      - openam-data:/usr/openam/config
      - ./openam-config.properties:/usr/openam/openam-config.properties:ro
      - ./openam-init.sh:/usr/local/tomcat/bin/openam-init.sh:ro
    command: |
      bash /usr/local/tomcat/bin/openam-init.sh

  wordpress:
    image: wordpress
    restart: always
    hostname: wordpress.example.org
    ports:
      - 8081:80
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: exampleuser
      WORDPRESS_DB_PASSWORD: examplepass
      WORDPRESS_DB_NAME: exampledb
    volumes:
      - wordpress:/var/www/html

  db:
    image: mysql:8.0
    restart: always
    environment:
      MYSQL_DATABASE: exampledb
      MYSQL_USER: exampleuser
      MYSQL_PASSWORD: examplepass
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
    volumes:
      - db:/var/lib/mysql

volumes:
  wordpress:
  db:
  openam-data:
```

To configure OpenAM immediately configured at startup, create an OpenAM `openam-config.properties` configuration file:

```
ACCEPT_LICENSES=true
SERVER_URL=http://openam.example.org:8080
DEPLOYMENT_URI=/openam
BASE_DIR=/usr/openam/config
locale=en_US
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=passw0rd
AMLDAPUSERPASSWD=p@passw0rd
COOKIE_DOMAIN=openam.example.org
```

```
ACCEPT_LICENSES=true
DATA_STORE=embedded
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=openam.example.org
DIRECTORY_PORT=50389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1689
ROOT_SUFFIX=dc=openam,dc=example,dc=org
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=passw0rd
```

and an initial configuration script `openam-init.sh`

```
#!/bin/bash

/usr/local/tomcat/bin/catalina.sh run &
SERVER_PID=$!

# Wait for OpenAM to respond to isAlive.jsp
until curl -f -s -o /dev/null http://localhost:8080/openam/isAlive.jsp; do
    echo "Waiting for OpenAM to fully initialize..."
    sleep 5
done

if [[ -f /usr/openam/config/boot.json ]]; then
    echo "OpenAM has already been configured."
else
    echo "Setting up OpenAM..."
    java -jar /usr/openam/ssoconfiguratortools/openam-configurator-tool*.jar --file
/usr/openam/openam-config.properties
fi

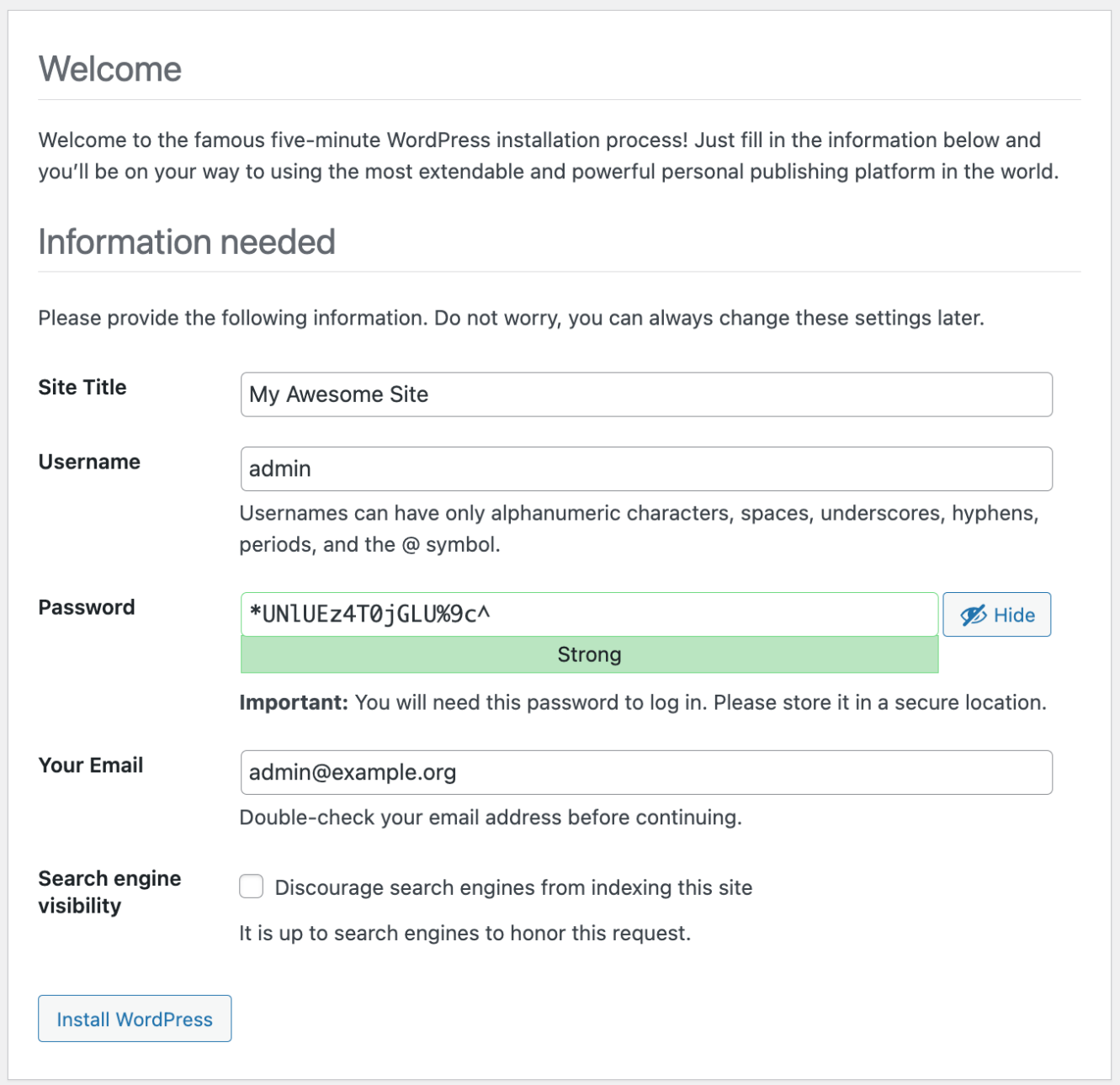
wait $SERVER_PID
```

Start the containers with the `docker compose up` command.

WordPress Configuration

Initial WordPress Configuration

If you already have WordPress configured, you can skip this section. If not, open the http://wordpress.example.org:8081/wp-admin/install.php link in your browser. Select the language and go to settings. Fill in the settings and be sure to remember the generated password. You will need it to log in to the administrator console.

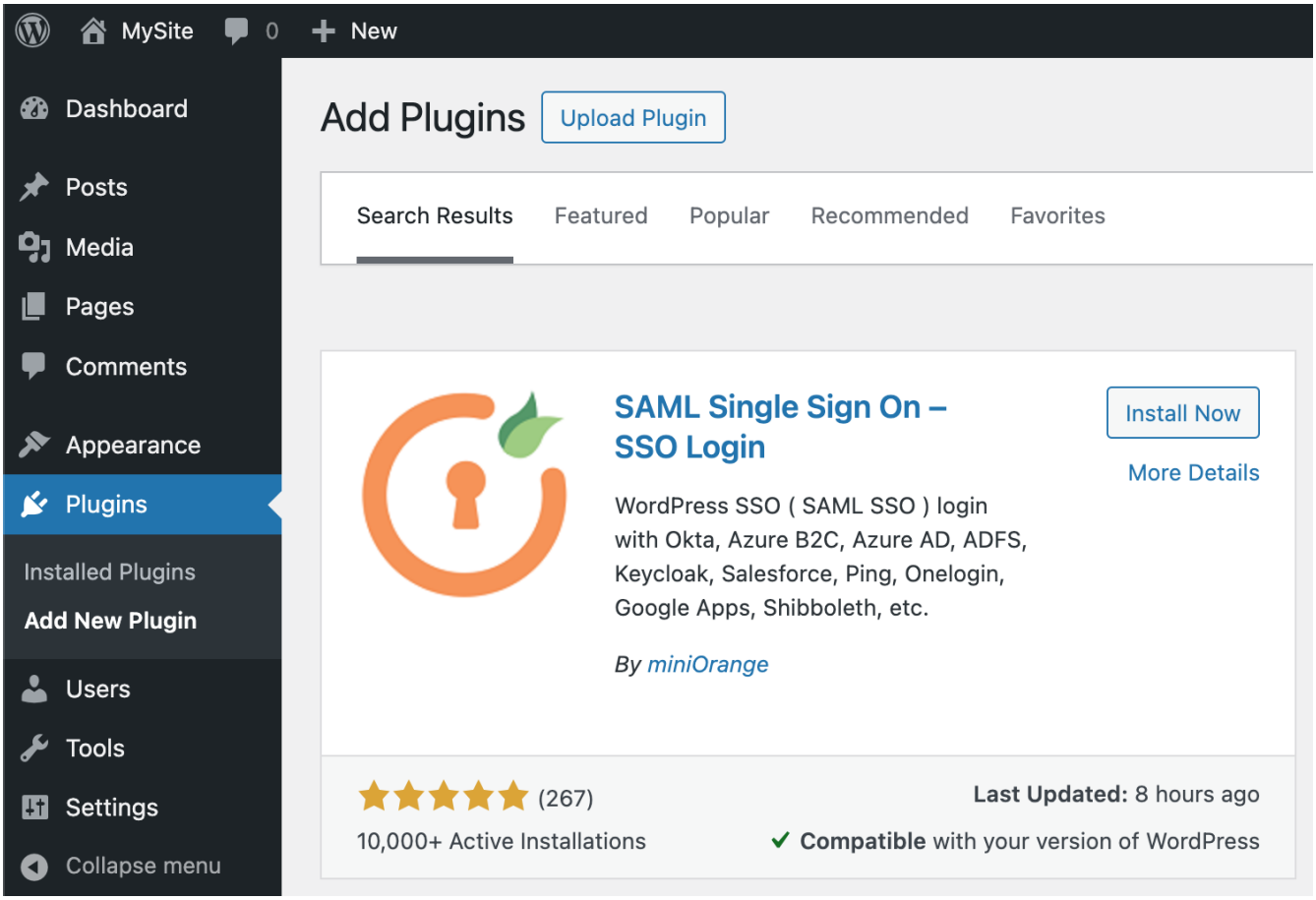


Press the `Install WordPress` button.

After completing the configuration, navigate to the login form and enter the administrator username and password specified during installation. The administrator console will open.

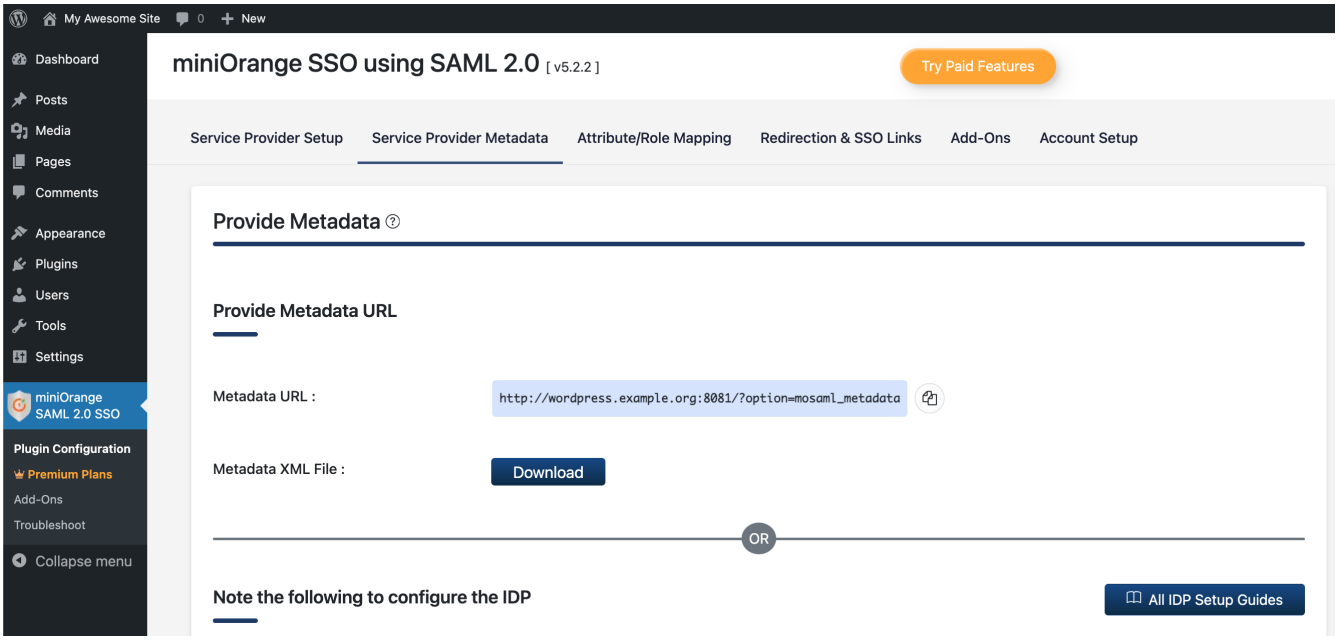
Install the SAMLv2 Plugin

In the administrator console, select Plugins. Click the `Add New Plugin` button and install the miniOrange SAML Single Sign On - SSO Login plugin.



Once installed, activate the plugin by clicking the Activate button in the plugin window. In the left panel, you will see the settings item for the installed plugin.

Go to the `Service Provider Metadata` section in the plugin settings panel and copy the Metadata URL. Change the port from 8081 to 80 because OpenAM will load metadata from the WordPress Docker container, available on port 80 within the Docker environment: `http://wordpress.example.org:80/?option=mosaml_metadata`.



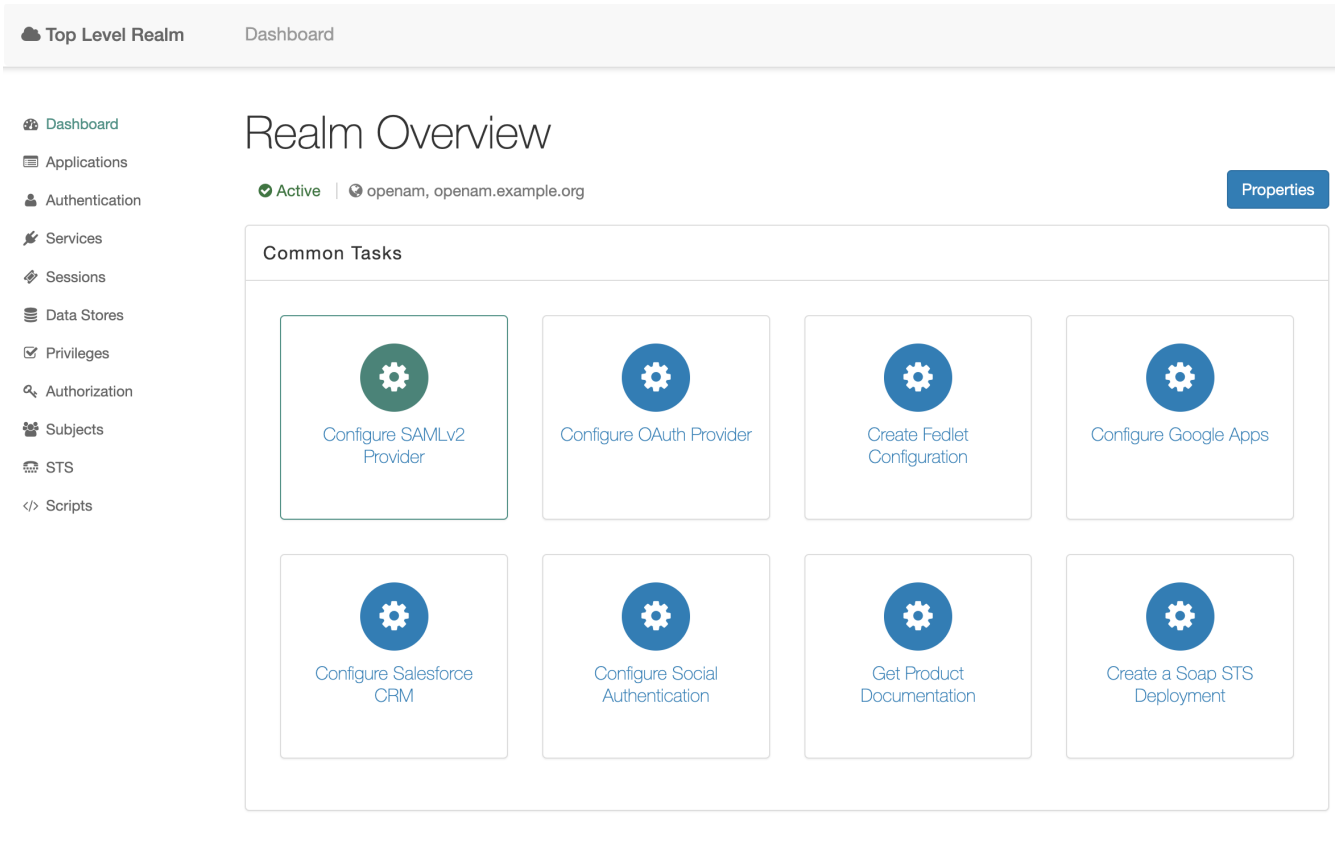
OpenAM Setup

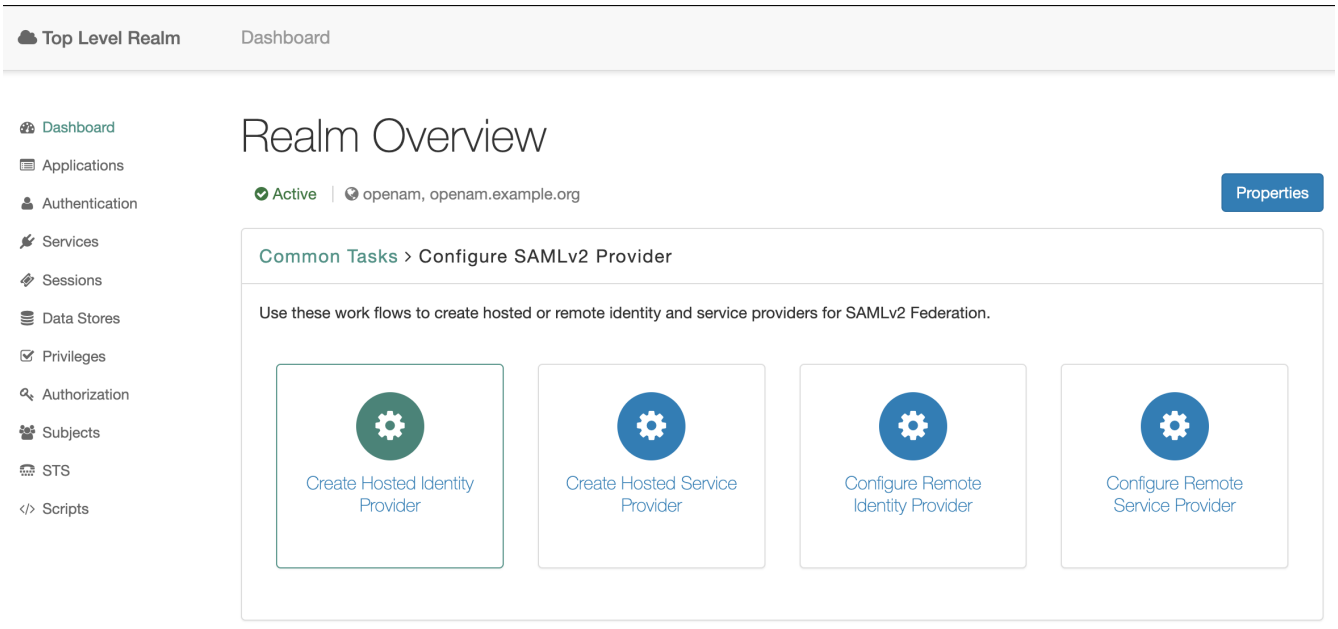
Create a Hosted Identity Provider

Go to the OpenAM administrator console at http://openam.example.org:8080/openam/XUI/#login/.

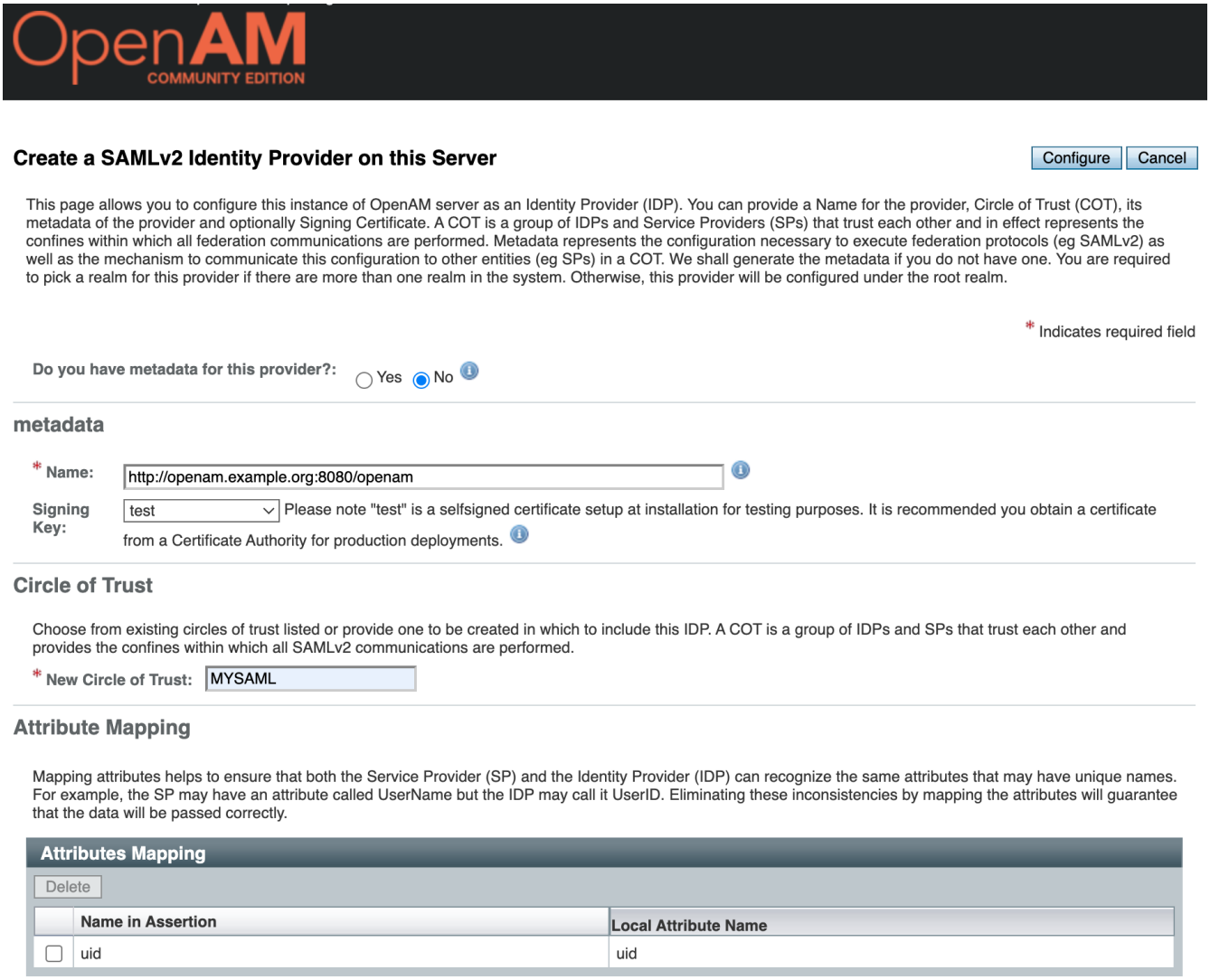
Enter the OpenAM administrator login and password. In our case, it will be `amadmin` and `passw0rd` respectively.

In the console, open `Top Level Realm`, click `Configure SAMLv2 Provider` → `Create Hosted Identity Provider`.

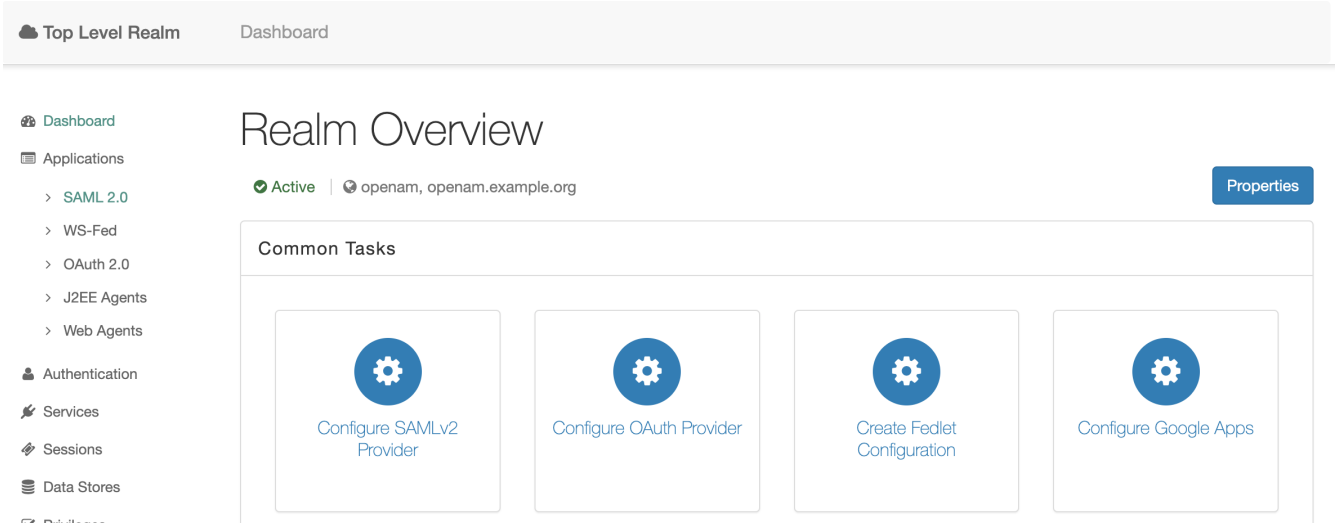




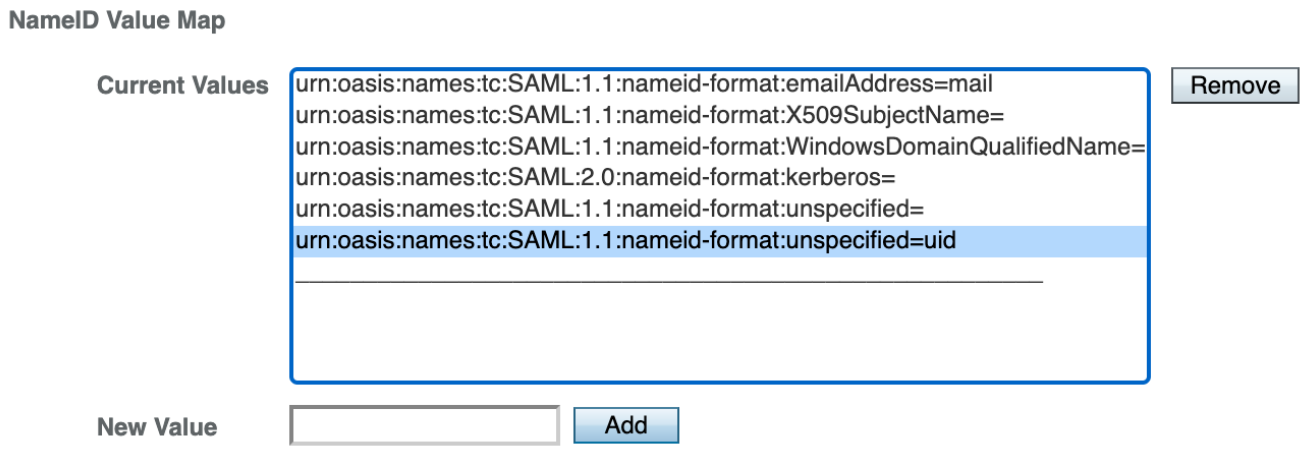
Complete the settings as shown in the screenshot and click the "Configure" button.



In the administrator console click `Top Level Realm` and in the left menu select `Applications` → `SAML 2.0`



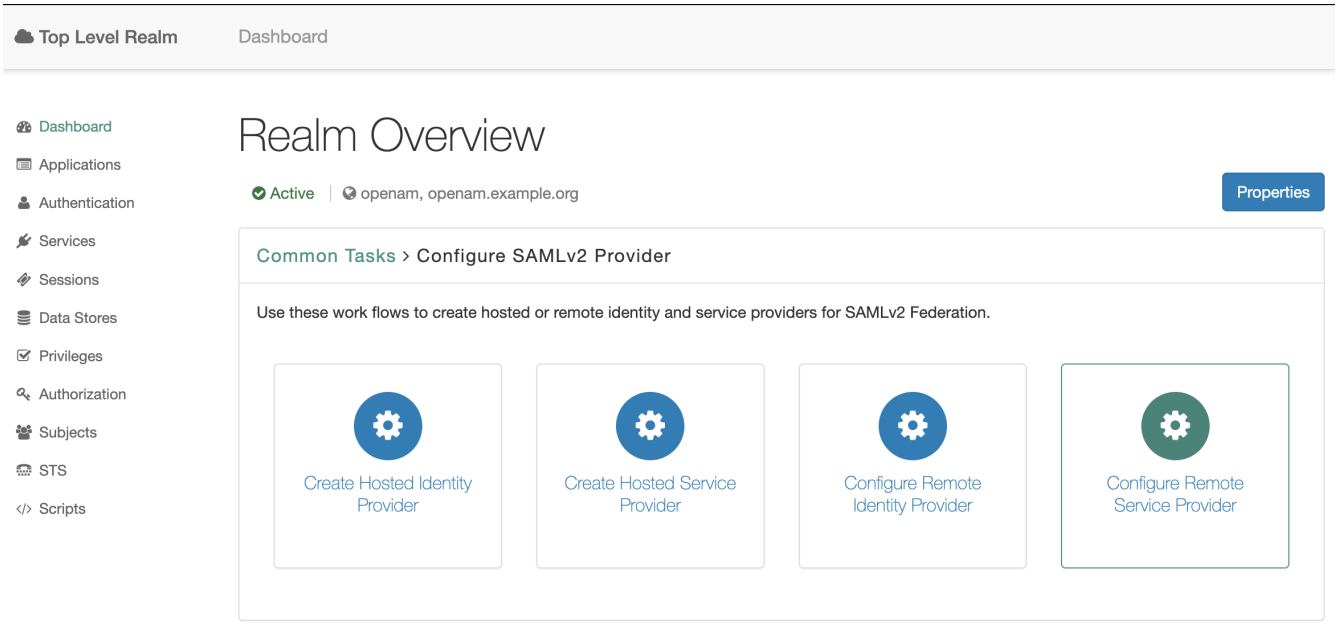
In the `Entity Providers` section, open the settings for Identity Provider `http://openam.example.org:8080/openam`. On the `Assertion Content` tab, go to `Name ID Format` → `NameID Value Map` and add the value `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified=uid`



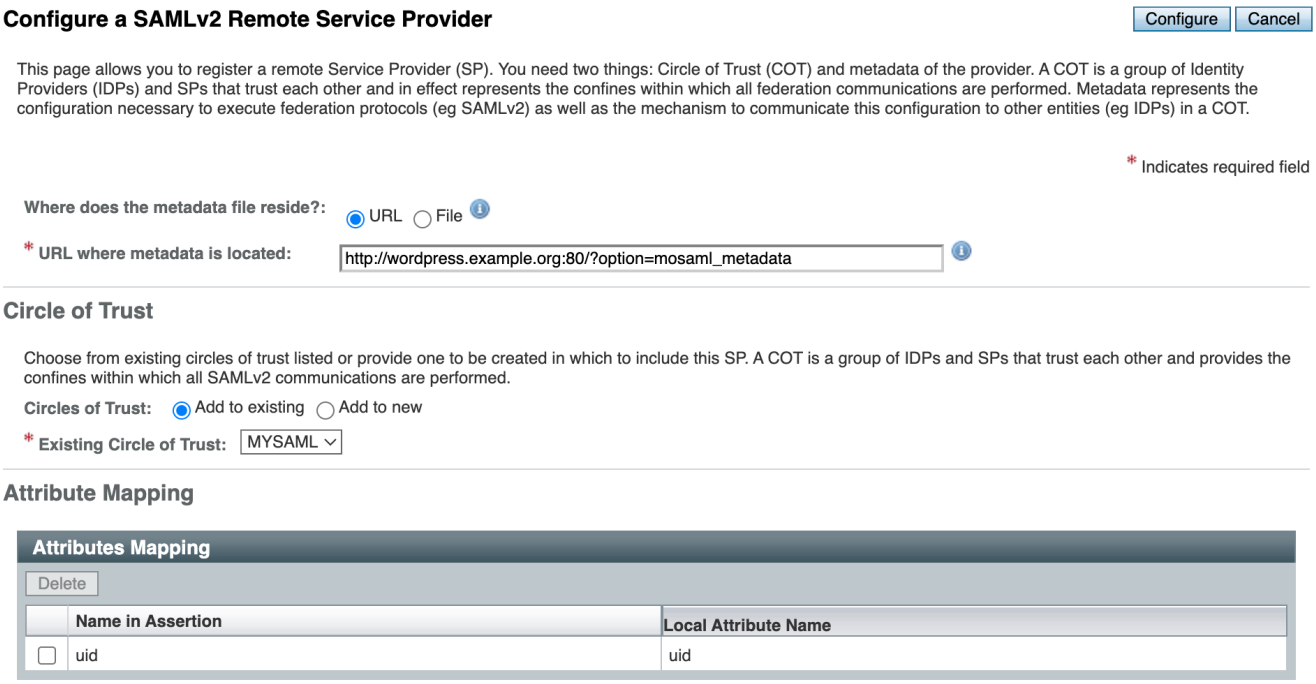
Press the Save button.

Configure Remote Service Provider

Next, register WordPress as a remote service provider. In the OpenAM administrator console, select Top Level Realm, then Configure SAMLv2 Provider → Configure Remote Service Provider.



Fill out the Remote Service Provider settings as shown in the screenshot. The metadata URL should be as in the SAMLv2 WordPress plugin configuration step: http://wordpress.example.org:80/?option=mosaml_metadata.



WordPress OpenAM Identity Provider Configuration

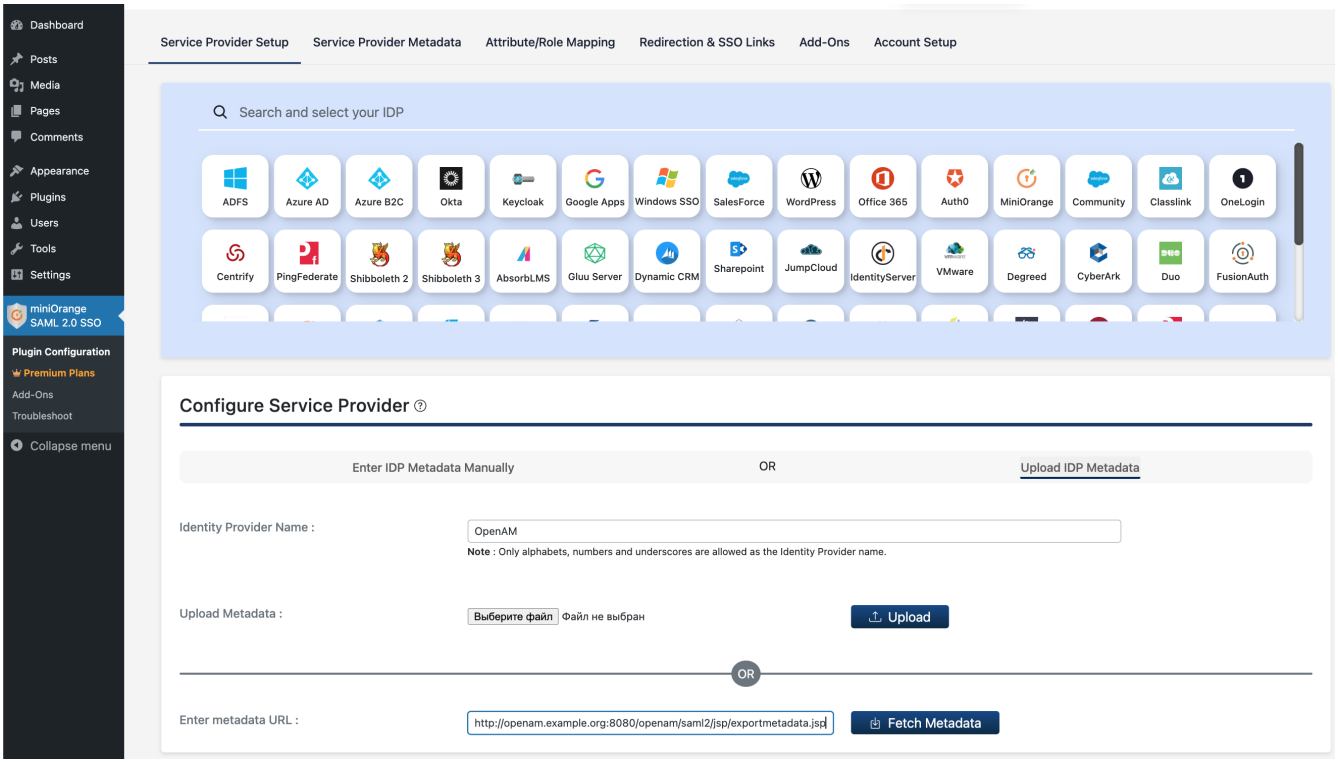
Go back to the WordPress administration console.

Open the miniOrange SAML plugin settings.

Click the `Service Provider Setup` tab.

In the `Configure Service Provider` section, click the `Upload IDP Metadata` tab.

Fill in the fields as shown in the screenshot below. The SAML metadata URL for OpenAM will be:`http://openam.example.org:8080/openam/saml2/jsp/exportmetadata.jsp`

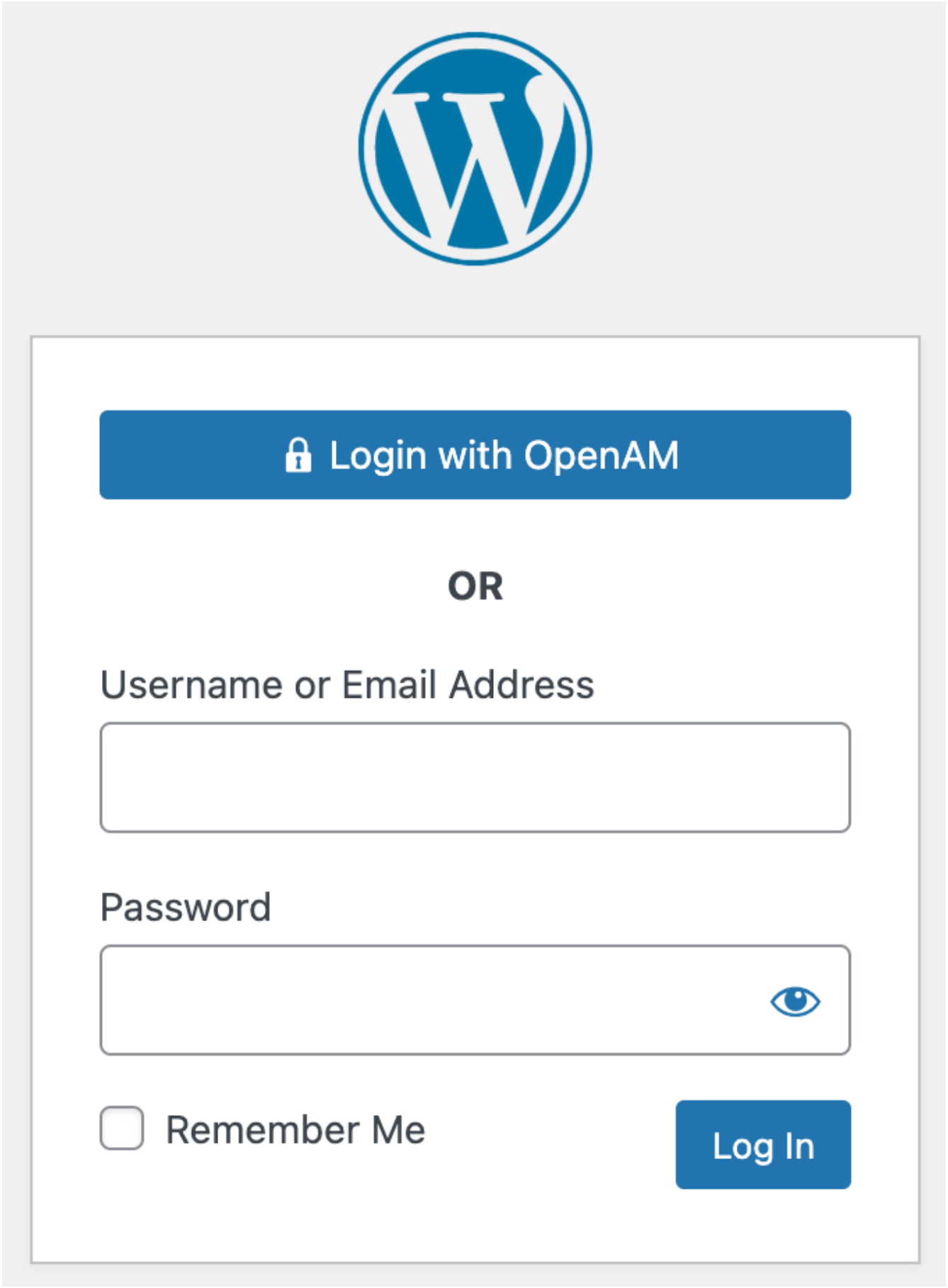


Press the `Fetch Metadata` button.

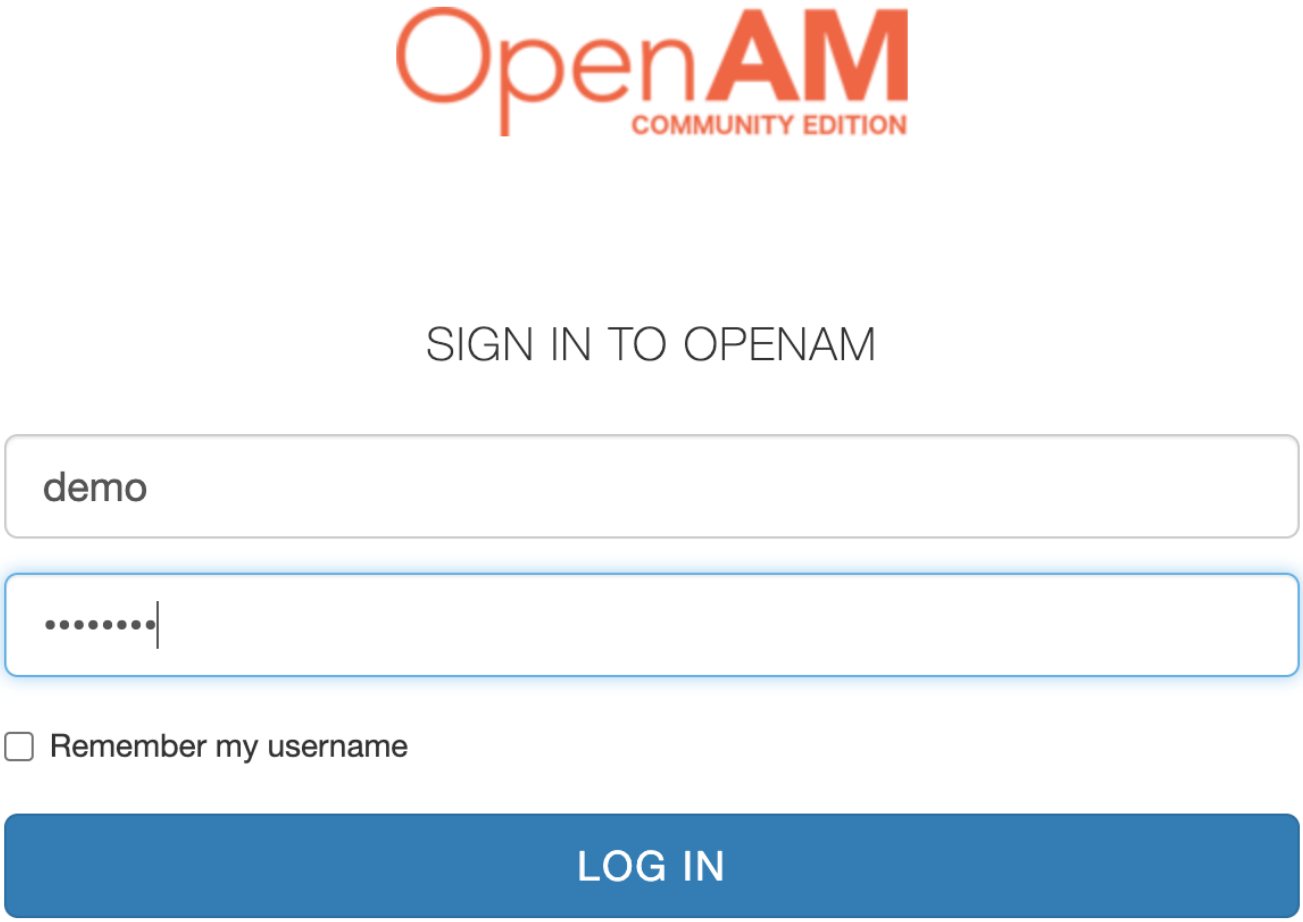
The configuration is now complete.

Test the Solution

Now let's test the solution. Log out of the WordPress administration console, OpenAM administration console or open a browser window in Incognito mode. Open the http://wordpress.example.org:8081/wp-admin/ link. The OpenAM login button will appear in the WordPress login window.

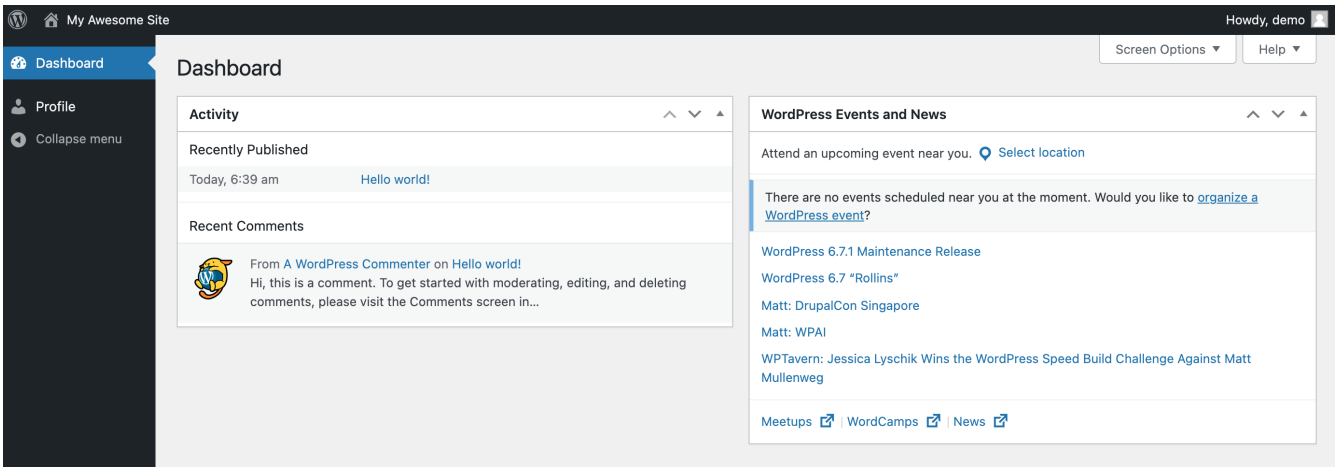


Click this button. You will be redirected to OpenAM authentication. Enter the user name `demo` and password `changeit`. The `demo` user is created during the initial installation of OpenAM. In a production environment, you should delete this user or change it's the default password.



Press the `LOG IN` button.

You will be authenticated to WordPress with the `demo` user account.



SAML Authentication in Yandex Cloud via OpenAM

Russian version

Getting started

To configure SAML authentication, there are the following prerequisites:

1. Docker platform installed and running Docker Engine
2. Locally running OpenAM instance. To launch OpenAM, run the following command.

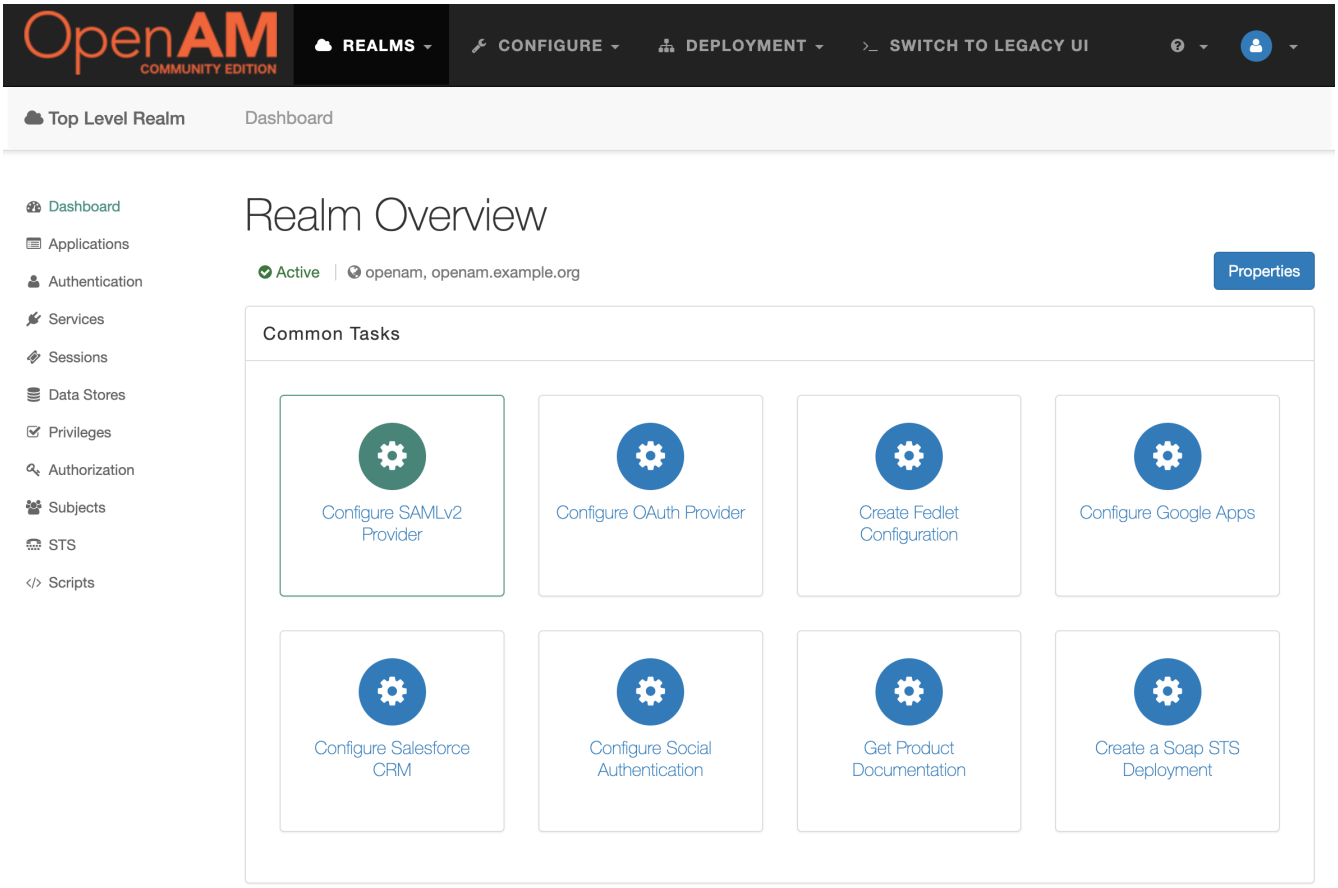
```
docker run -h openam.example.org -p 8080:8080 --name openam
openidentityplatform/openam
```

Detailed description of the installation and initial configuration of OpenAM at this link

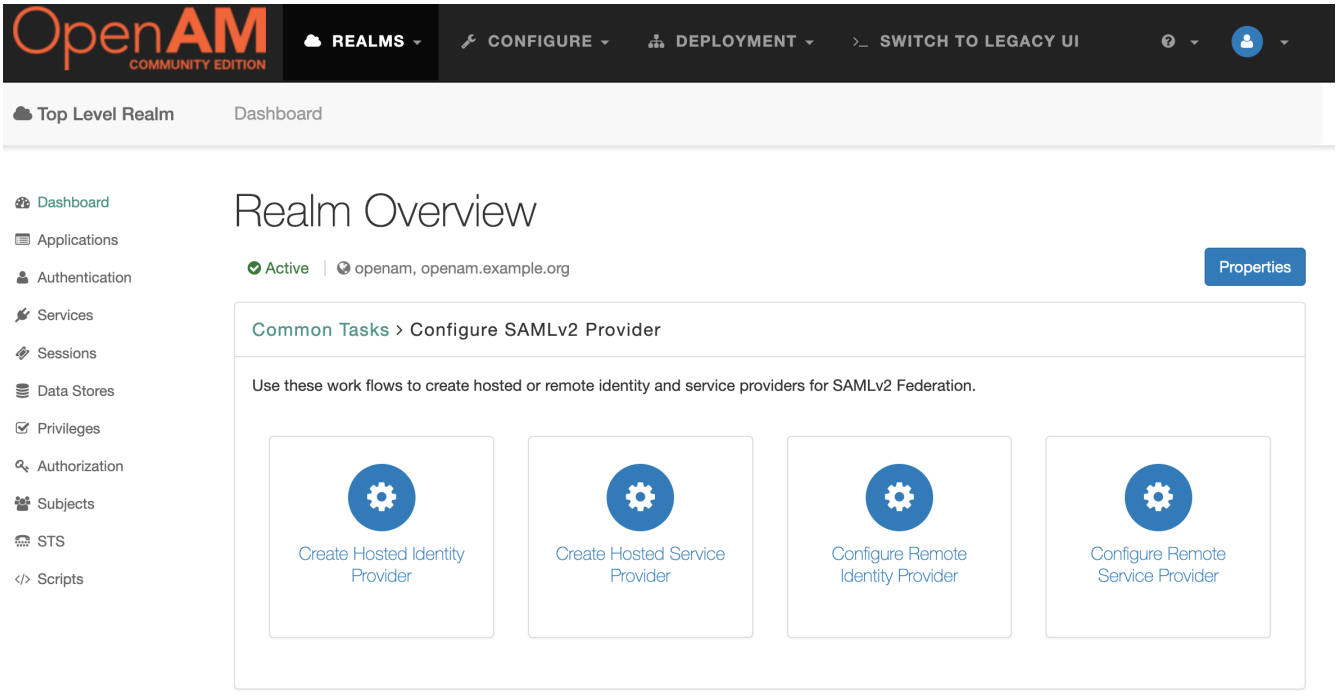
Setting up Federation in OpenAM

Create and Configure Identity Provider in OpenAM

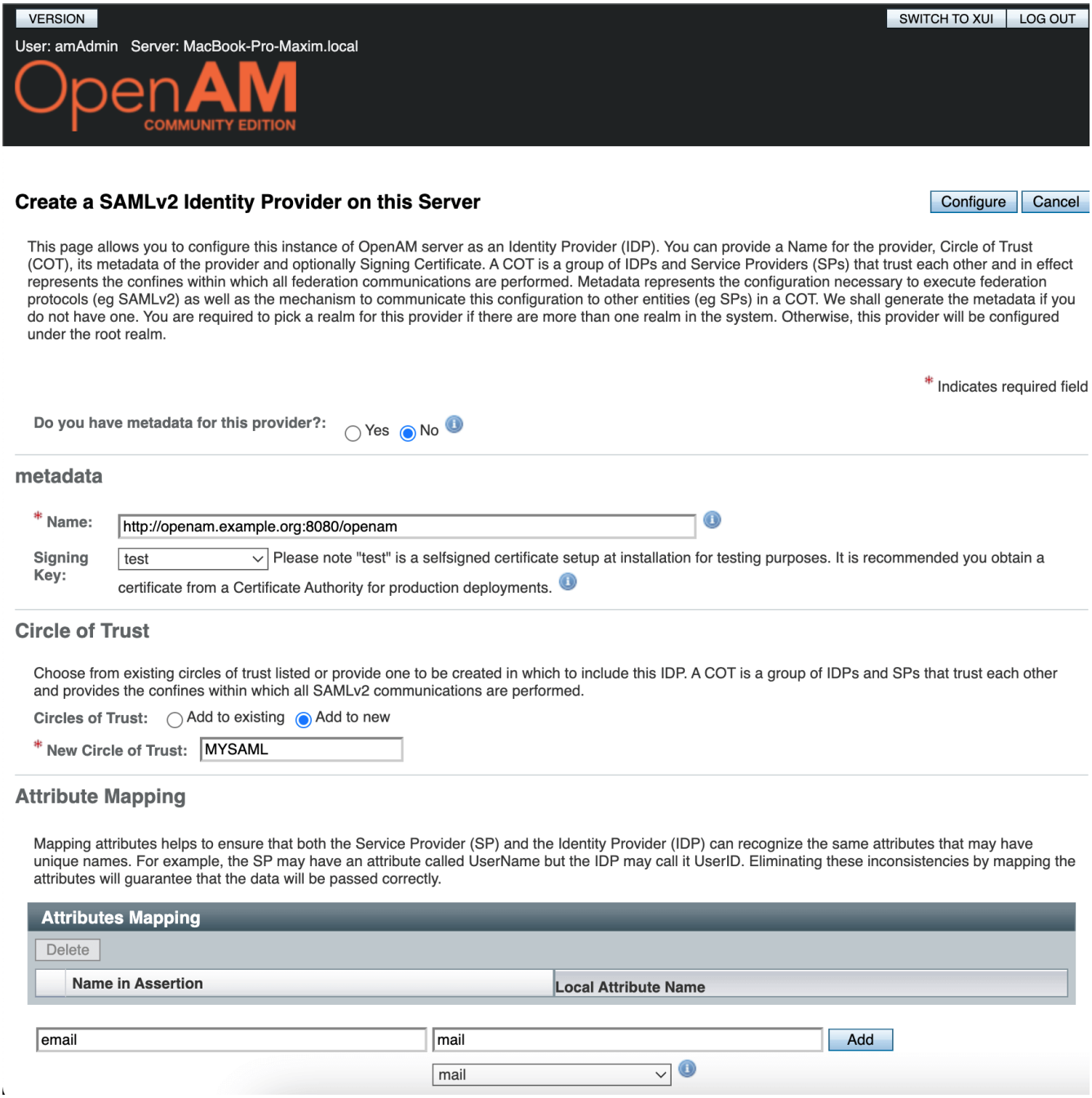
1. Open the administration console
2. Goto desired realm
3. Select **Configure SAMLv2 Provider** in the Common Tasks section



4. Then **Create Hosted Identity Provider**



5. Select a Signing Key (for demonstration purposes we will use **test**), enter the Circle of Trust name, and then map user's attributes to OpenAM. For demonstration users will be mapped by email.



6. Press the Configure button.

Setting up the User Mapping in OpenAM

1. Open the administration console
2. Goto desired realm
3. From the menu on the left, select the Applications section and navigate to SAML 2.0
4. In the Entity Providers list select `http://openam.example.org:8080/openam`
5. On the Assertion Content tab, under **NameID Format** in the **NameID Value Map** list, add the element `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified=mail`
6. Press the Save button

Creating OpenAM Users

1. Open the administration console
2. Goto desired realm
3. Create a new user and fill the Email Address attribute

Creating and Setting up a Federation in Yandex Cloud Organization

Create a Federation

1. Gotot Yandex Cloud Organization.
2. On the left panel choose Federations .
3. Click the **Create Federation** button.
4. Give your federation a name. It must be unique within the folder.
5. You can also add a description, if required.
6. In the Cookie lifetime field, specify the time before the browser asks the user to re-authenticate.
7. In the **IdP Issuer** field, enter a link to the OpenAM: `http://openam.example.org:8080/openam`
8. In the Link to the **IdP login page** field, enter a link: `http://openam.example.org:8080/openam/SSORedirect/metaAlias/idp`
9. Click the **Create federation** button.

Add Certificates

While authenticating, the Cloud Organization service should be able to verify the IdP server certificate. To enable this, add the certificate to the federation:

1. Open the OpenAM link `http://openam.example.org:8080/openam/saml2/jsp/exportmetadata.jsp` and copy the `ds:X509Certificate` tag value
2. Save the certificate in a text file with a `.cer` extension in the following format:

```
----BEGIN CERTIFICATE-----
<value ds:X509Certificate>
----END CERTIFICATE-----
```

1. In the left panel, select Federations.
2. Click the the federation for which you want to add the certificate.
3. At the bottom of the page, click **Adding a certificate**.
4. Enter the certificate name and description.
5. Click **Choose a file** and specify the path to the certificate file.

6. Click the **Add** button.

Creating a Remote Service Provider в OpenAM

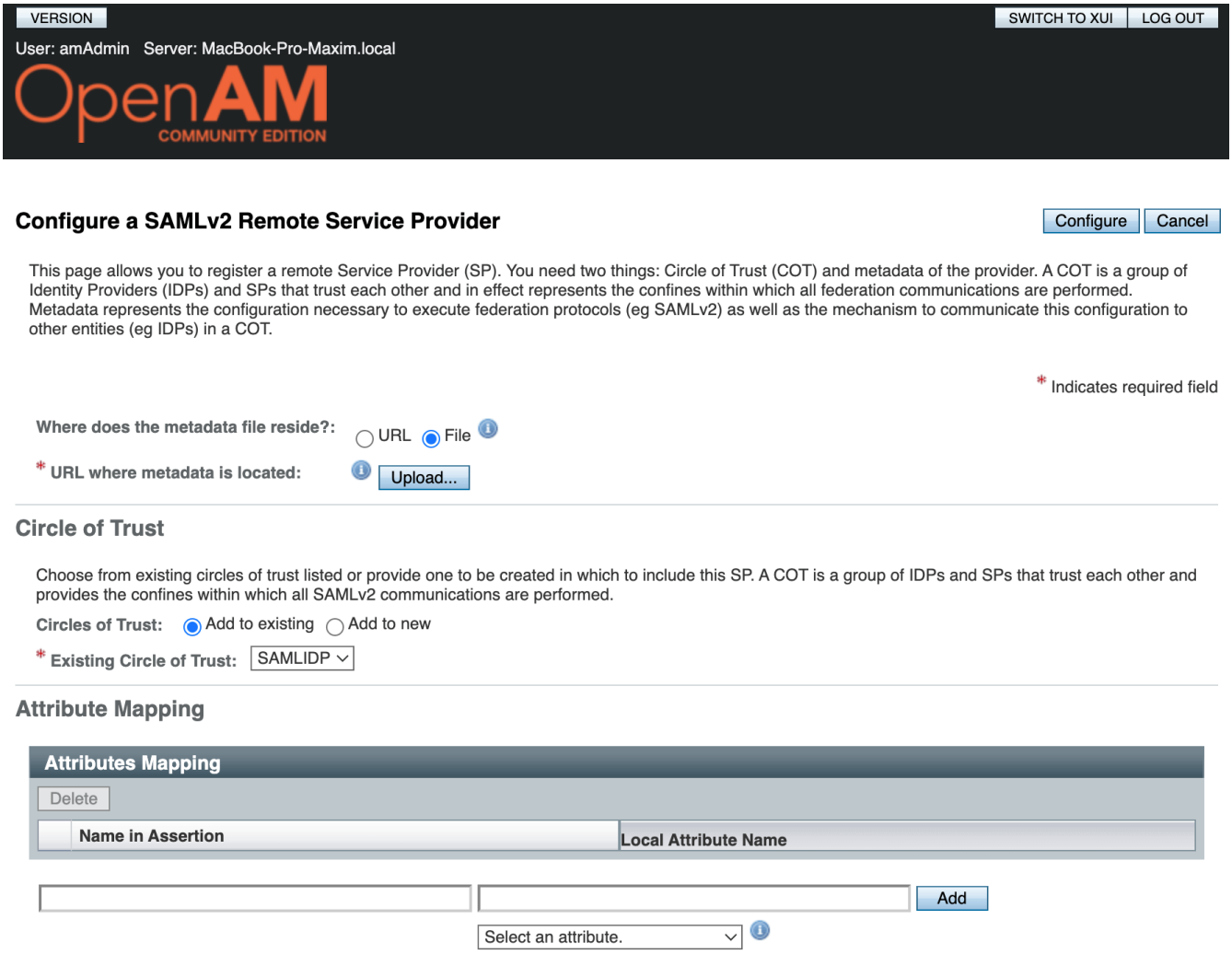
Creating a Metadata file for OpenAM

1. Create a `metadata.xml` file with the following contents where federation_ID - is a federation identifier in Yandex Cloud Organization, Federations section

```
<?xml version="1.0" encoding="UTF-8"?><md:EntityDescriptor
xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
entityID="https://console.cloud.yandex.ru/federations/<federation_ID>">
    <md:SPSSODescriptor
protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
        <md:AssertionConsumerService
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
Location="https://console.cloud.yandex.ru/federations/<federation_ID>" index="1"/>
    </md:SPSSODescriptor>
</md:EntityDescriptor>
```

Creating a Service Provider in OpenAM

1. Open the administration console
2. Goto desired realm
3. Choose **Configure SAMLv2 Provider** in the Common Tasks section
4. Then click **Configure Remote Service Provider**



1. Load the `metadata.xml` file created in the previous step as metadata.
2. Select the existing Circle Of Trust, created on the **Create and Configure Identity Provider in OpenAM** stage
3. Нажмите Configure

Adding Users to Yandex Cloud Organization

If the **Automatically create users** option is enabled, a federation will only add users logging in to a cloud for the first time. You can only add a federated user again manually after deleting them from a federation.

1. Login to account that belongs to an organization administrator or owner.
2. Goto Yandex Cloud Organization service.
3. In the left panel, select Users.
4. In the top-right corner, click **Add federated users**.
5. Select the identity federation to add users from.
6. Enter emails of the OpenAM users
7. Click the **Add** button. This will give the OpenAM users access to the organization.

Testing Authentication

When you finish setting up SSO, test that everything works properly:

1. Open your browser in guest or private browsing mode.
2. Follow the URL to log in to the management console:

   `https://console.cloud.yandex.ru/federations/<federation_ID>`

3. Enter OpenAM user credentials and click Login.

On successful authentication, the IdP server will redirect you to URL `https://console.cloud.yandex.ru/federations/<federation_ID>`, which you have set in the `metadata.xml` file for the Service Provider in the OpenAM settings and then to the management console home page. In the top-right corner, you will be able to see you are logged in to the console as a federated user.

SSO Configuration: OpenIG as SAML Service Provider for OpenAM

Introduction

The SAML 2.0 protocol is the standard for Single Sign-On (SSO) in enterprise environments. In this guide, we will show you how to use OpenIG as a proxy and service provider to easily add SAML authentication to any of your applications without changing its code.

Preparation

1. To simplify service deployment, we will use OpenAM and OpenIG's Docker images. Therefore, you must have Docker installed.
2. Enter the host names for OpenAM and OpenIG in the `hosts` file. On Windows systems, the hosts file is located in the `C:\Windows/System32/drivers/etc/hosts` directory, and on Linux or Mac OS, it is located in `/etc/hosts`.

   ```
   127.0.0.1    openam.example.org openig.example.org
   ```

Configuring OpenAM

Installing OpenAM

Deploy the OpenAM container with the command:

```
docker run -h openam.example.org -p 8080:8080 --name openam
openidentityplatform/openam
```

And perform the initial configuration:

```
docker exec -w '/usr/openam/ssoconfiguratortools' openam bash -c \
'echo "ACCEPT_LICENSES=true
SERVER_URL=http://openam.example.org:8080
DEPLOYMENT_URI=/$OPENAM_PATH
BASE_DIR=$OPENAM_DATA_DIR
locale=en_US
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=passw0rd
AMLDAPUSERPASSWD=p@passw0rd
COOKIE_DOMAIN=example.org
ACCEPT_LICENSES=true
DATA_STORE=embedded
```

```
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=openam.example.org
DIRECTORY_PORT=50389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1689
ROOT_SUFFIX=dc=openam,dc=example,dc=org
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=passw0rd" > conf.file && java -jar openam-configurator-tool*.jar
--file conf.file'
```

OpenAM Identity Provider Setup

1. Access the administrator console via the link http://openam.example.org:8080/openam. Use the login `amadmin` and password `passw0rd`.
2. Select Top Level Realm
3. Go to **Create SAMLv2 Providers → Create Hosted Identity Provider**
4. Set Metadata Name: `openam`
5. In the Signing Key setting, select `test`
6. Enter the name of the Circle of Trust, for example, `cot`
7. In the Attribute Mapping section, add the mapping uid → uid, mail → mail
8. Click **Configure**

OpenAM Fedlet Configuration

1. Open the administrator console
2. Select Top Level Realm
3. Go to **Create Fedlet Configuration**
4. Enter the fedlet name in the Name field, for example, openig.
5. Set the **Destination URL of the Service Provider which will include the Fedlet** to the URL that points to OpenIG: `http://openig.example.org:8081/saml`
6. In the Attribute Mapping section, add the mapping uid → uid, mail → mail
7. Click **Create**

   The Fedlet settings will be saved in the container in the directory `/usr/openam/config/myfedlets/openig/Fedlet.zip`.

   Copy the settings to the host machine using the command:

   ```
   docker cp openam:/usr/openam/config/myfedlets/openig/Fedlet.zip .
   ```

Preparing a test user

1. Open the administrator console
2. Select Top Level Realm
3. In the left pane, select Subjects
4. In the list of accounts, open the `demo` account
5. In the Email Address field, enter `demo@example.org` or another valid email
6. Click **Save**

OpenIG Setup

Preparing OpenIG Configuration Files

1. Create a directory for OpenIG configuration files `openig-saml`
2. Add a directory `config` to it
3. In the `config` directory, create the files `admin.json` and `config.json`:

   `admin.json`:

   ```
   {
     "prefix" : "openig",
     "mode": "PRODUCTION"
   }
   ```

   `config.json`:

   ```
   {
     "heap": [
       {
         "name": "JwtSession",
         "type": "JwtSession"
       },
       {
         "name": "capture",
         "type": "CaptureDecorator",
         "config": {
           "captureEntity": true,
           "_captureContext": true
         }
       }
     ],
     "handler": {
       "type": "Chain",
       "config": {
         "filters": [],
   ```

```
      "handler": {
        "type": "Router",
        "name": "_router",
        "capture": "all"
      }
    }
  }
}
```

4. Add a directory for routes named `routes` to the `config` directory.
5. Add the default route `99-default.json` to the `routes` directory. OpenIG will serve static content on this route and will not require authentication:

```
{
  "handler": {
    "type": "DispatchHandler",
    "config": {
      "bindings": [
        {
          "handler": {
            "type": "StaticResponseHandler",
            "config": {
              "status": 200,
              "reason": "OK",
              "entity":
"<!doctype html>
<html>
<head>
  <title>Home</title>
  <meta charset='UTF-8'>
</head>
<body>
  <h1><a href='/app'>Login</a></h1>
</body>
</html>"
            }
          }
        }
      ]
    }
  },
  "session": "JwtSession"
}
```

6. Start the OpenIG Docker container with the command. Note the mounted directory `/app-saml`.

```
docker run -h openig.example.org -p 8081:8080 --name openig \
  -v ./app-saml:/usr/local/app-saml:ro \
```

```
  -e "CATALINA_OPTS=-Dopenig.base=/usr/local/app-saml" \
  openidentityplatform/openig
```

7. Check if the application is working:

```
$ curl -v http://openig.example.org:8081
*   Trying 127.0.0.1:8081...
* Connected to openig.example.org (127.0.0.1) port 8081 (#0)
> GET / HTTP/1.1
> Host: openig.example.org:8081
> User-Agent: curl/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Content-Length: 146
< Date: Mon, 24 Nov 2025 12:46:56 GMT
<
<!doctype html>
<html>
<head>
  <title>Home</title>
  <meta charset='UTF-8'>
</head>
<body>
  <h1><a href='/app'>Login</a></h1>
</body>
```

Configuring SAML Fedlet in OpenIG

1. In the `openig-saml` directory, create a directory named `SAML`.
2. Copy the contents of the Fedlet.zip archive that you received from OpenAM into this directory.

```
unzip Fedlet.zip
cp conf/* app-saml/SAML/
```

3. Create a route for obtaining credentials from SAML assertions `05-saml.json`

```
{
 "handler": {
   "type": "SamlFederationHandler",
   "config": {
     "assertionMapping": {
       "uid": "uid",
       "mail": "mail"
     },
     "redirectURI": "/app"
```

```
    }
  },
  "condition": "${matches(request.uri.path, '^/saml')}",
  "session": "JwtSession"
}
```

4. Create a route for the application requiring SAML authentication `05-app.json`:

```
{
  "handler": {
    "type": "DispatchHandler",
    "config": {
      "bindings": [
        {
          "condition": "${empty session.uid}",
          "handler": {
            "type": "StaticResponseHandler",
            "config": {
              "status": 302,
              "reason": "Found",
              "headers": {
                "Location": [
                  "http://openig.example.org:8081/saml/SPInitiatedSSO"
                ]
              }
            }
          }
        },
        {
          "handler": {
          "handler": {
            "type": "StaticResponseHandler",
            "config": {
              "status": 200,
              "reason": "OK",
              "entity":
"<!doctype html>
<html>
<head>
  <title>OpenID Connect Discovery</title>
  <meta charset='UTF-8'>
</head>
<body>
  <h1>User: ${session.uid}, email: ${session.mail} </h1>
</body>
</html>"
            }
          }
        }
        }
```

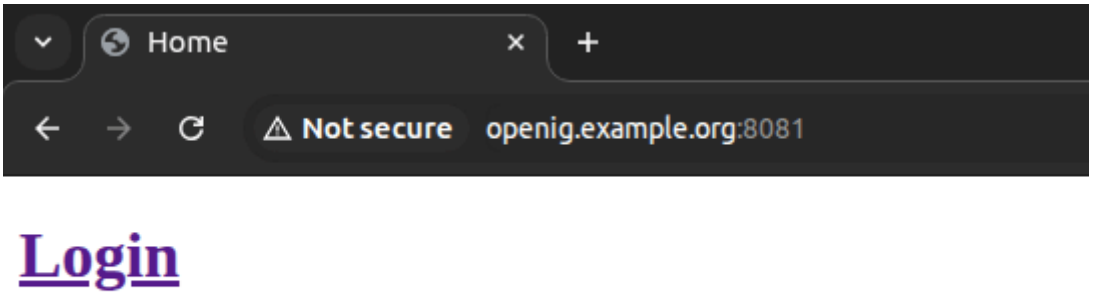
```
        ]
      }
    },
    "condition": "${matches(request.uri.path, '^/app')}",
    "session": "JwtSession"
}
```

5. Start the OpenIG container:

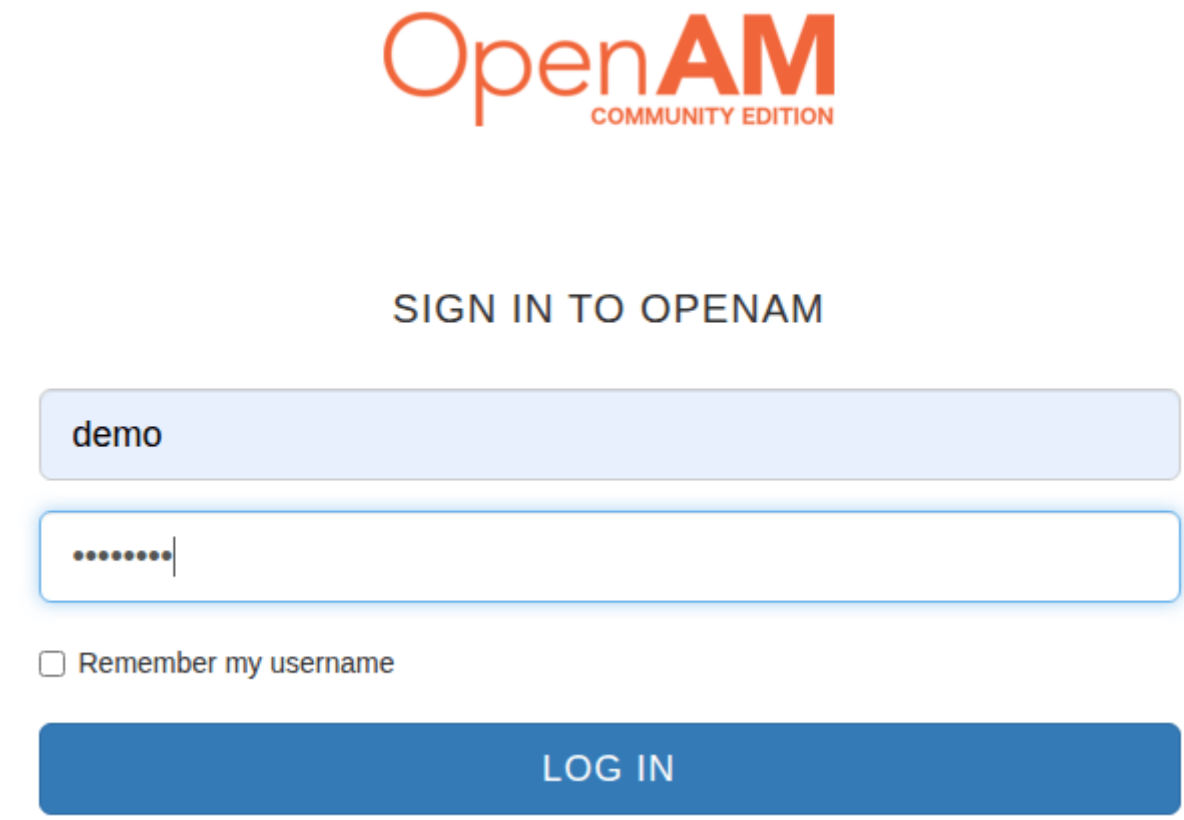
```
docker run -h openig.example.org -p 8081:8080 --name openig \
  -v ./app-saml:/usr/local/app-saml:ro \
  -e CATALINA_OPTS=-Dopenig.base=/usr/local/app-saml \
  openidentityplatform/openig
```

Test the Solution

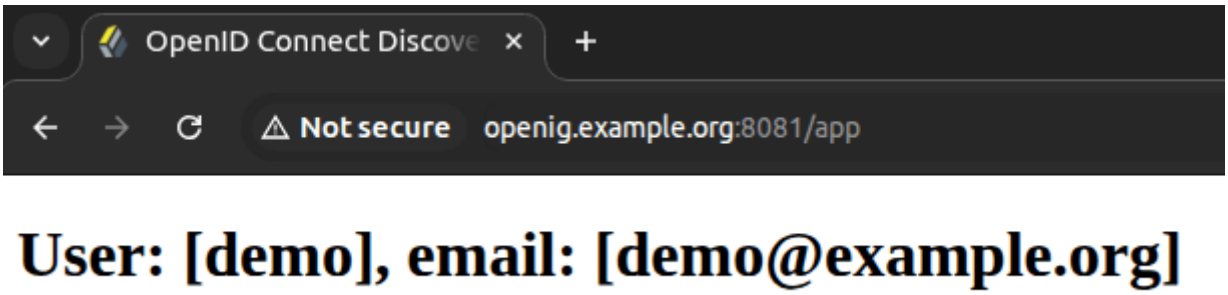
1. Exit the OpenAM console or open your browser in incognito mode.
2. Open the link to the OpenIG application, which does not require authentication: http://openig.example.org:8081/.



3. Click on the `Login` link.
4. The OpenAM authentication form will open.
5. Enter the demo user credentials. Login: `demo`, password: `changeit`, and click the **Login** button.



6. You will be redirected to the application with the demo user credentials:



Conclusion.

We have successfully configured OpenIG as a Service Provider and implemented SAML 2.0 authentication via OpenAM. You can now use this approach to secure any applications in your infrastructure. The next step could be to configure Log Out.

For more details on configuring OpenAM and OpenIG, please refer to the documentation:

- OpenAM documentation
- http://doc.openidentityplatform.org/openig

Setup Tracing with OpenTelemetry and Jaeger

Preconditions

Download the OpenTelemetry agent file opentelemetry-javaagent.jar by link https://github.com/open-telemetry/opentelemetry-java-instrumentation/releases/latest each OpenAM и OpenIG host

If Jaeger is not installed it can be run as a Docker container:

```
docker run -d --name jaeger \
  -e COLLECTOR_ZIPKIN_HOST_PORT=:9411 \
  -e COLLECTOR_OTLP_ENABLED=true \
  -p 6831:6831/udp \
  -p 6832:6832/udp \
  -p 5778:5778 \
  -p 16686:16686 \
  -p 4317:4317 \
  -p 4318:4318 \
  -p 14250:14250 \
  -p 14268:14268 \
  -p 14269:14269 \
  -p 9411:9411 \
  jaegertracing/all-in-one:1.43
```

OpenAM

Set environment variables

Variable	Example	Description
OTEL_TRACES_EXPORTER	jaeger	An exporter of traces
OTEL_SERVICE_NAME	openam	Service name for monitoring
OTEL_EXPORTER_JAEGER_TIMEOUT	10000	Jaeger connection timeout in milliseconds
OTEL_EXPORTER_JAEGER_ENDPOINT	http://localhost:14250	gRPC Jaeger URL endpoint
JAVA_TOOL_OPTIONS	-javaagent:/path/to/opentelemetry-javaagent.jar	Path to OpenTelemetry Java agent

Restart OpenAM instance

OpenIG

Set environment variables

Variable	Example	Description
OTEL_TRACES_EXPORTER	jaeger	An exporter of traces
OTEL_SERVICE_NAME	openig	Service name for monitoring
OTEL_EXPORTER_JAEGER_TIMEOUT	10000	Jaeger connection timeout in milliseconds
OTEL_EXPORTER_JAEGER_ENDPOINT	http://localhost:14250	gRPC Jaeger URL endpoint
JAVA_TOOL_OPTIONS	-javaagent:/path/to/opentelemetry-javaagent.jar	Path to OpenTelemetry Java agent

Restart OpenIG instance

Stateful vs Stateless Authentication

Preface

Authentication is a process that exists in almost every application to Identify the application client whether it is a user or another application. While authentication client sends credentials to the application. The application ensures that credentials are correct, generates an authentication token, and sends it back to the client. To access the application as an identified client, the client uses the received authentication token.

Authentication tokens could be Stateless and Stateful.

Stateful Authentication

After successful authentication, the application generates a random token to send back to the client and then creates a client authenticated session in memory or an internal database. When a client tries to access the application with a given token, the application tries to retrieve session data from session storage, checks if the session valid, and then decides whether the client has access to the desired resource or not.

Stateless Authentication

After successful authentication, the application generates a token with all necessary data, signs it with a public key, and sends it back to a client. There is a standard for token generation, it is JWT (JSON Web Token). The process is described in OpenID Connect (OIDC) specification. When a client tries to access the application with a token, the application verifies the token sign with a private key, checks if the token is expired, retrieves all session data from the token, and makes a decision if a client has access to the desired resource.

	Stateful	**Stateless**
Session information could be stolen	✅ It is impossible to steal session information from the session identifier because it is just an identifier associated with the session	❌Session identifier contains all authentication information and it is possible to steal sensitive information, it is not encrypted.
Resource consuming	❌When retrieving session information, the service always gets access to session storage which causes additional resource consumption.	❌The session identifier contains all session information.
Easy to implement	❌When session information is stored in an external database, there is a need to implement session database persistence.	❌Session identifier contains all session information, there is no need to implement additional functionality

	Stateful	Stateless
Easy to scale	While adding new instances, there is a need to implement additional scale to session storage as well	Adding new service instances does not require additional effort
Possibility to compromise session data.	Only the authentication system is able to retrieve session information from an authentication token, so there are no more vulnerabilities.	To decrypt session information from a token, all parts of the system should share the same key. And, if at least one system is compromised, all parts of the system are under the threat.
Authentication token size	An authentication token is just an identifier, so session data does not affect its size.	If an authentication session contains a large amount of data, the authentication token also becomes large, which can cause additional load on a network.
Restrict access among different parts of an application	It is possible to configure the system so different parts of the system will only have access to the data necessary for their work	All parts of the system have access to all session data
Possibility to revoke session	It is possible to revoke a session at any time	Since the session token contains an expiration date, it is impossible to revoke the authentication session
Possibility to modify session data	It is possible to modify any session data in session data storage.	Since the session token contains all session data, it is not possible to modify it
SSO implementation	The integration of different parts of the system is possible without modification of the source code, session information can come through the authentication system gateway.	Changes must be made to each part of the system to retrieve data from an authentication token

Conclusion

Both approaches make sense, both have their advantages and disadvantages. Stateless authentication is easier to implement and scale, but stateful authentication is more secure and easier to manage.

Bootstrap OpenAM and OpenDJ Docker Containers

Create `docker-compose.yml` file with OpenAM and OpenDJ services with the following contents:

```
services:
  opendj:
    image: openidentityplatform/opendj:latest
    container_name: opendj
    environment:
      - BASE_DN=dc=example,dc=com
      - LDAP_PORT=1389
      - ROOT_PASSWORD=admin_password
      - REPLICATION_PORT=8989
    ports:
      - "1389:1389"
      - "1636:1636"
      - "4444:4444"
    volumes:
      - opendj-data:/opt/opendj/data
    restart: always

  openam:
    image: openidentityplatform/openam:latest
    container_name: openam
    depends_on:
      - opendj
    ports:
      - "8080:8080"
    volumes:
      - openam-data:/usr/openam/config
      - ./openam-config.properties:/usr/openam/openam-config.properties:ro
      - ./openam-init.sh:/usr/local/tomcat/bin/openam-init.sh:ro
    command: |
      bash /usr/local/tomcat/bin/openam-init.sh
    restart: always

volumes:
  opendj-data:
  openam-data:
```

Create `openam-config.properties` file with the following contents:

```
# General Settings
SERVER_URL=http://localhost:8080
DEPLOYMENT_URI=/openam
BASE_DIR=/usr/openam/config
```

```
locale=en_US
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=amadmin_password
AMLDAPUSERPASSWD=admin_password
COOKIE_DOMAIN=localhost
ACCEPT_LICENSES=true

# Data Store (OpenDJ)
DATA_STORE=dirServer
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=opendj
DIRECTORY_PORT=1389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1636
ROOT_SUFFIX=dc=example,dc=com
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=admin_password

# User Store (OpenDJ)
USERSTORE_TYPE=LDAPv3ForOpenDS
USERSTORE_SSL=SIMPLE
USERSTORE_HOST=opendj
USERSTORE_PORT=1389
USERSTORE_SUFFIX=dc=example,dc=com
USERSTORE_MGRDN=cn=Directory Manager
USERSTORE_PASSWD=admin_password
```

Create `openam-init.sh` file. This script waits for OpenDJ, then for OpenAM to start, and if OpenAM is not configured runs `openam-configurator-tool` with `openam-config.properties` file.

```
#!/bin/bash

until cat < /dev/null > /dev/tcp/opendj/1389; do
    echo "Waiting for OpenDJ..."
    sleep 5
done

/usr/local/tomcat/bin/catalina.sh run &
SERVER_PID=$!

# Wait for OpenAM to respond to isAlive.jsp
until curl -f -s -o /dev/null http://localhost:8080/openam/isAlive.jsp; do
    echo "Waiting for OpenAM to fully initialize..."
    sleep 5
done

if [[ -f /usr/openam/config/boot.json ]]; then
    echo "OpenAM has already been configured."
else
```

```
    echo "Setting up OpenAM..."
    java -jar /usr/openam/ssoconfiguratortools/openam-configurator-tool*.jar --file
/usr/openam/openam-config.properties
fi

wait $SERVER_PID
```

Run the `docker compose up` command. OpenAM and OpenDJ will be configured according to the `openam-config.properties` file

Test OpenAM with the following command:

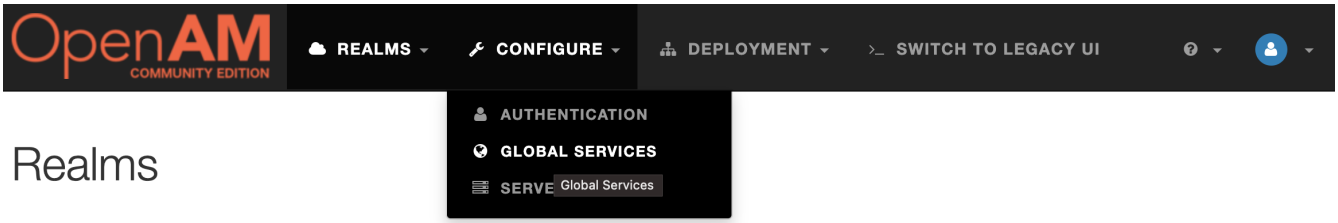
```
curl \
 --request POST \
 --header "Content-Type: application/json" \
 --header "X-OpenAM-Username: amadmin" \
 --header "X-OpenAM-Password: amadmin_password" \

http://localhost:8080/openam/json/authenticate
{
   "realm" : "/",
   "successUrl" : "/openam/console",
   "tokenId" :
"AQIC5wM2LY4SfcxC_WOLjtSC7DdP5nIkhXU2DPuIWjHNlSE.*AAJTSQACMDEAAlNLABIxMjY1NTY0NTA4NDU3
ODQyNTAAAlMxAAA.*"
}
```

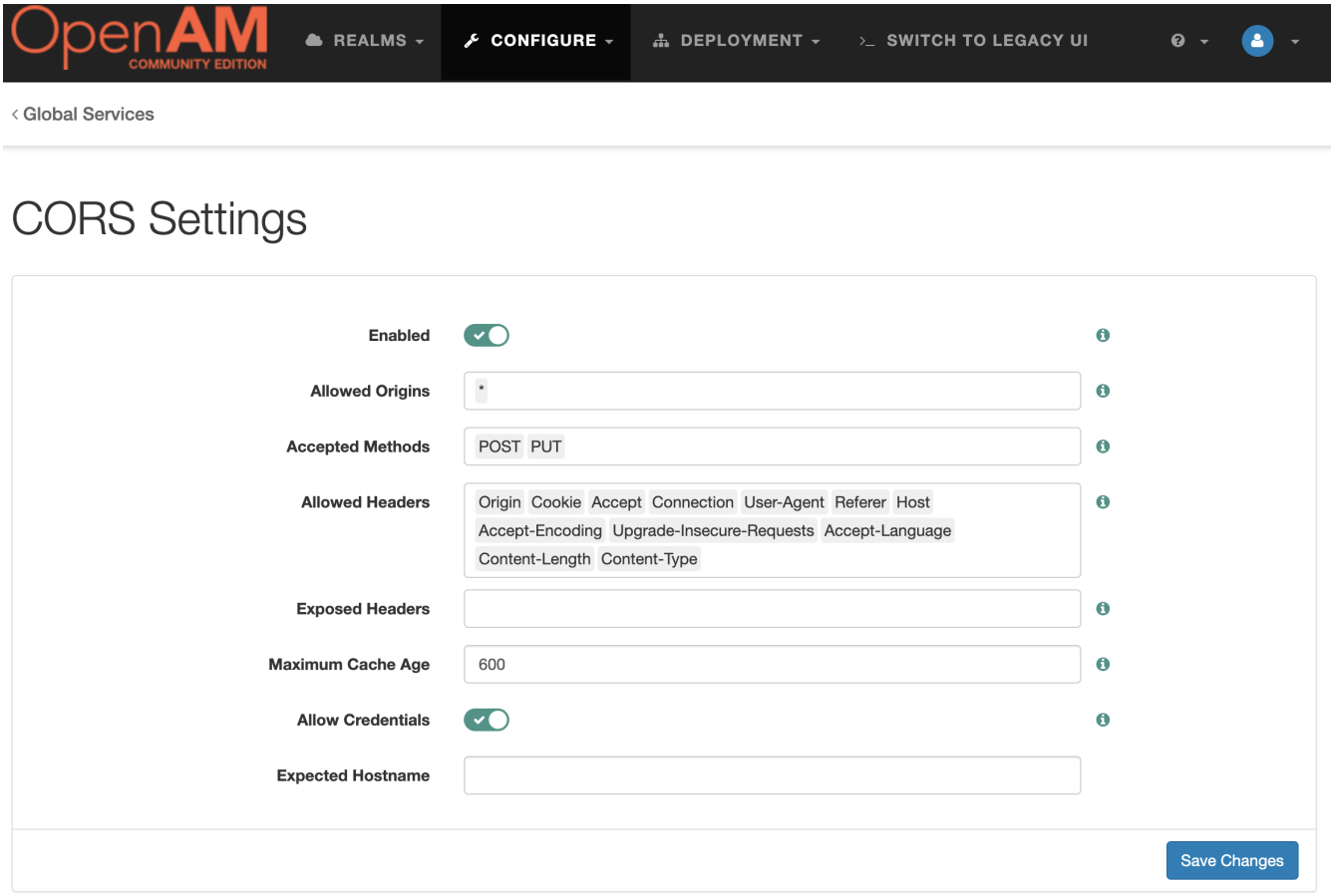
Thanks to @Lewiscowles1986. See #817

How to Enable CORS

Open the administrator console. In the top menu, select Configure → Global Services.



Next, go to CORS Settings and enable CORS support



Pres the `Save Changes` button.

How to Pass Additional System Property to CATALINA_OPTS to OpenAM Docker Container

Sometimes it is necessary to append a new system property to existing CATALINA_OTPS environment variable, for example -DXUI.enable=false

You could do this with the following command:

```
docker run --rm -h openam.example.org -p 8080:8080 --network openam --name openam
openidentityplatform/openam \
 "sh" "-c" "CATALINA_OPTS=\"\$CATALINA_OPTS -DXUI.enable=false\" &&
/usr/local/tomcat/bin/catalina.sh run"
```

How to Persist OpenAM Docker Container Data Between Restarts

Docker containers are stateless. So, if you want persist the OpenAM's configuration data, you need to specify a volume that maps `/usr/openam/config` container directory to a local directory. Example with `docker-compose.yaml`

```
services:
  openam:
    image: openidentityplatform/openam:latest
    ports:
      - 8080:8080
      - 8085:8085
    volumes:
      - ./openam-data:/usr/openam/config
```

With `docker run` command:

```
docker run -v ./openam-data:/usr/openam/config \
    -h openam.example.org \
    -p 8080:8080 \
    --name openam \
    openidentityplatform/openam
```

If you are facing permisson issues in Lunux, use a named volume

```
docker volume create openam-data
```

```
docker run -v openam-data:/usr/openam/config \
    -h openam.example.org \
    -p 8080:8080 \
    --name openam \
    openidentityplatform/openam
```

Quick OpenAM Docker Configuration From a Command Line

Let OpenAM hostname is `openam.example.org` Before starting, add the hostname and IP address to the `hosts` file, for example `127.0.0.0.1 openam.example.org`.

On Windows systems, the `hosts` file is located at `C:\Windows/System32/drivers/etc/hosts` , on Linux and Mac it is located at `/etc/hosts`.

Next, start the OpenAM Docker container. Run the following command:

```
docker run -h openam.example.org -p 8080:8080 --name openam
openidentityplatform/openam
```

There is no need to configure OpenAM manually via UI. The following command helps to automate the process:

```
 docker exec -w '/usr/openam/ssoconfiguratortools' openam bash -c \
'echo "ACCEPT_LICENSES=true
SERVER_URL=http://openam.example.org:8080
DEPLOYMENT_URI=/$OPENAM_PATH
BASE_DIR=$OPENAM_DATA_DIR
locale=en_US
PLATFORM_LOCALE=en_US
AM_ENC_KEY=
ADMIN_PWD=passw0rd
AMLDAPUSERPASSWD=p@passw0rd
COOKIE_DOMAIN=example.org
ACCEPT_LICENSES=true
DATA_STORE=embedded
DIRECTORY_SSL=SIMPLE
DIRECTORY_SERVER=openam.example.org
DIRECTORY_PORT=50389
DIRECTORY_ADMIN_PORT=4444
DIRECTORY_JMX_PORT=1689
ROOT_SUFFIX=dc=openam,dc=example,dc=org
DS_DIRMGRDN=cn=Directory Manager
DS_DIRMGRPASSWD=passw0rd" > conf.file && java -jar openam-configurator-tool*.jar
--file conf.file'
```

For external LDAP server OpenDJ use:

```
docker exec -w '/usr/openam/ssoconfiguratortools' openam bash -c \
          'echo "ACCEPT_LICENSES=true
          SERVER_URL=http://openam.example.org:8080
          DEPLOYMENT_URI=/$OPENAM_PATH
```

```
        BASE_DIR=$OPENAM_DATA_DIR
        locale=en_US
        PLATFORM_LOCALE=en_US
        AM_ENC_KEY=
        ADMIN_PWD=ampassword
        AMLDAPUSERPASSWD=password
        COOKIE_DOMAIN=example.org
        DATA_STORE=dirServer
        DIRECTORY_SSL=SIMPLE
        DIRECTORY_SERVER=opendj
        DIRECTORY_PORT=1389
        DIRECTORY_ADMIN_PORT=4444
        DIRECTORY_JMX_PORT=1689
        ROOT_SUFFIX=dc=example,dc=com
        DS_DIRMGRDN=cn=Directory Manager
        DS_DIRMGRPASSWD=password
        USERSTORE_TYPE=LDAPv3ForOpenDS
        USERSTORE_SSL=SIMPLE
        USERSTORE_HOST=opendj
        USERSTORE_PORT=1389
        USERSTORE_SUFFIX=dc=example,dc=com
        USERSTORE_MGRDN=cn=Directory Manager
        USERSTORE_PASSWD=password
        " > conf.file && java -jar openam-configurator-tool*.jar --file conf.file'
```

Using Google Authenticator with OpenAM

This article is intended for technical specialists or security system architects who want to implement two-factor authentication (2FA) in an access control system to enhance the security of user accounts.

Adding a second factor makes it significantly more difficult for attackers to compromise accounts.

Stack Used

OpenAM is an open-source access management system. It is designed for centralized management of authentication, authorization, and user accounts.

Google Authenticator is a mobile application designed to be used as an additional authentication factor. It supports one-time passwords (TOTP), and biometric authentication.

In this article, we will deploy OpenAM, configure modules and authentication chains for use with Google Authenticator, and show you how to add a second factor authentication for a user.

We will use authentication with one-time passwords generated using the TOTP (time-based one-time password) protocol. Such passwords do not need to be sent to the client device via SMS or PUSH notifications. These passwords are generated using a specific cryptographic algorithm directly on the device.

Installing OpenAM

If you do not have OpenAM installed yet, you can deploy a Docker container as described in the OpenAM wiki article

Configuring OpenAM

We will configure the module and authentication chain.

The authentication module in OpenAM is responsible for a specific authentication method. This can be authentication with a username and password, via the Kerberos protocol, or using biometrics. Modules can be organized into chains. This allows you to build chains of modules to authenticate users in multiple stages or using different methods. For example, if seamless authentication via the Kerberos protocol fails, you can ask the user for their login and password.

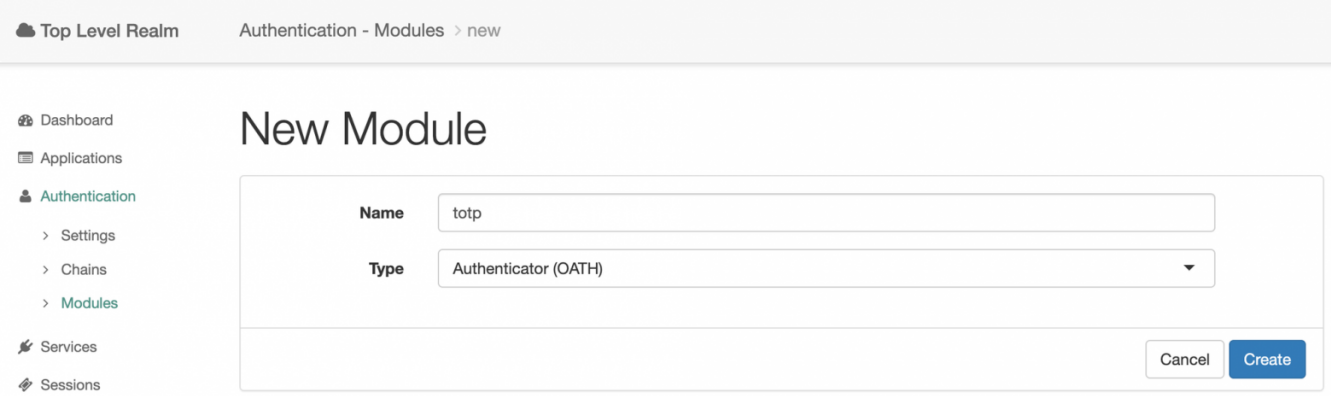
Configuring the TOTP Authentication Module

Open the administrator console at http://openam.example.org:8080/openam/console

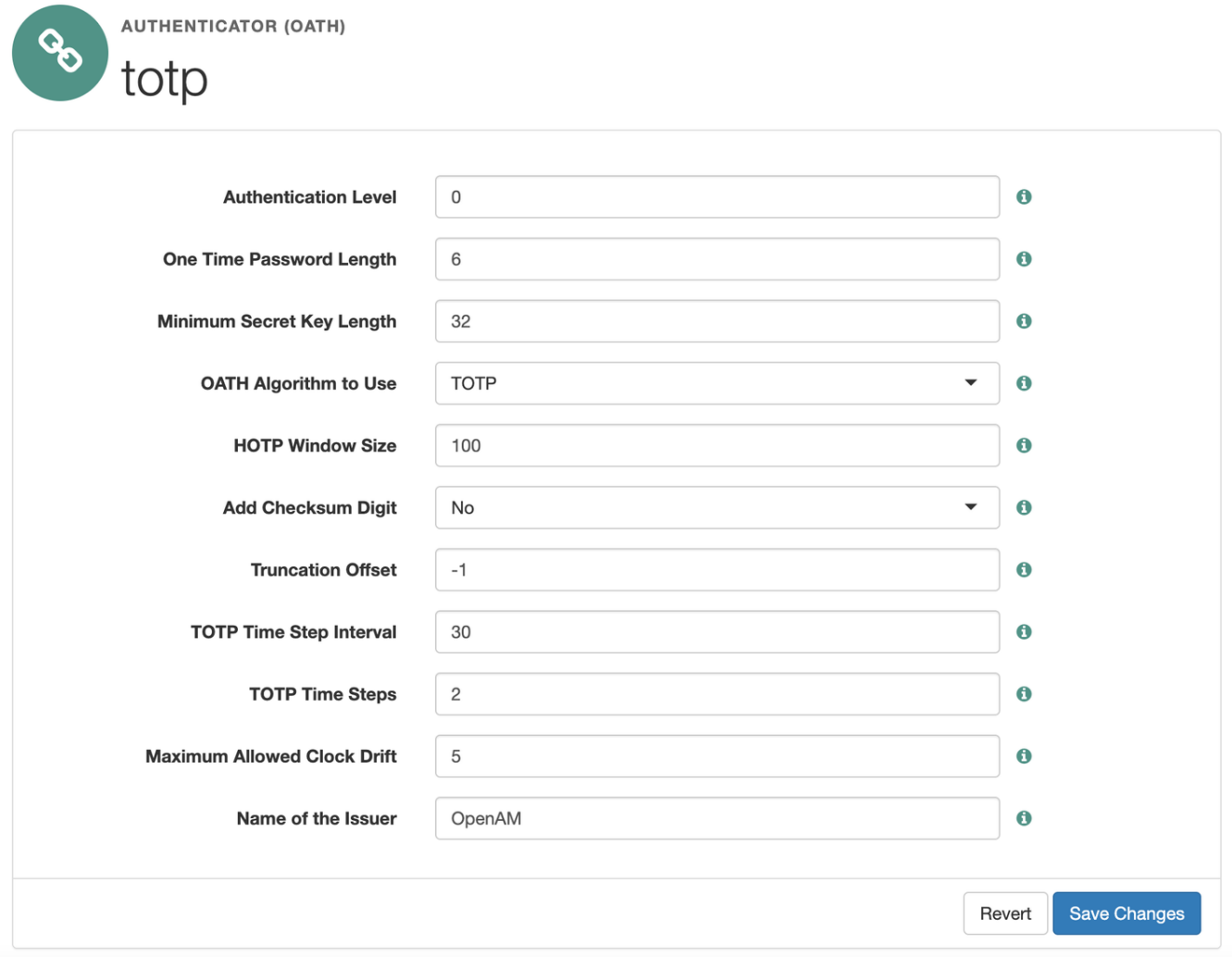
In the login field, enter `amadmin`. In the password field, enter the administrator password specified during installation.

Open the root realm, select Authentication → Modules in the left menu, and click the `Add Module` button. In the form that appears, enter the module name, for example `totp`, and the module type -

`Authenticator (OATH)`. Click the `Create` button.



Set the `OATH Algorithm to Use` setting to `TOTP`, enter any non-empty value in the `Name of the Issuer` field, for example `OpenAM`, and click `Save Changes`.



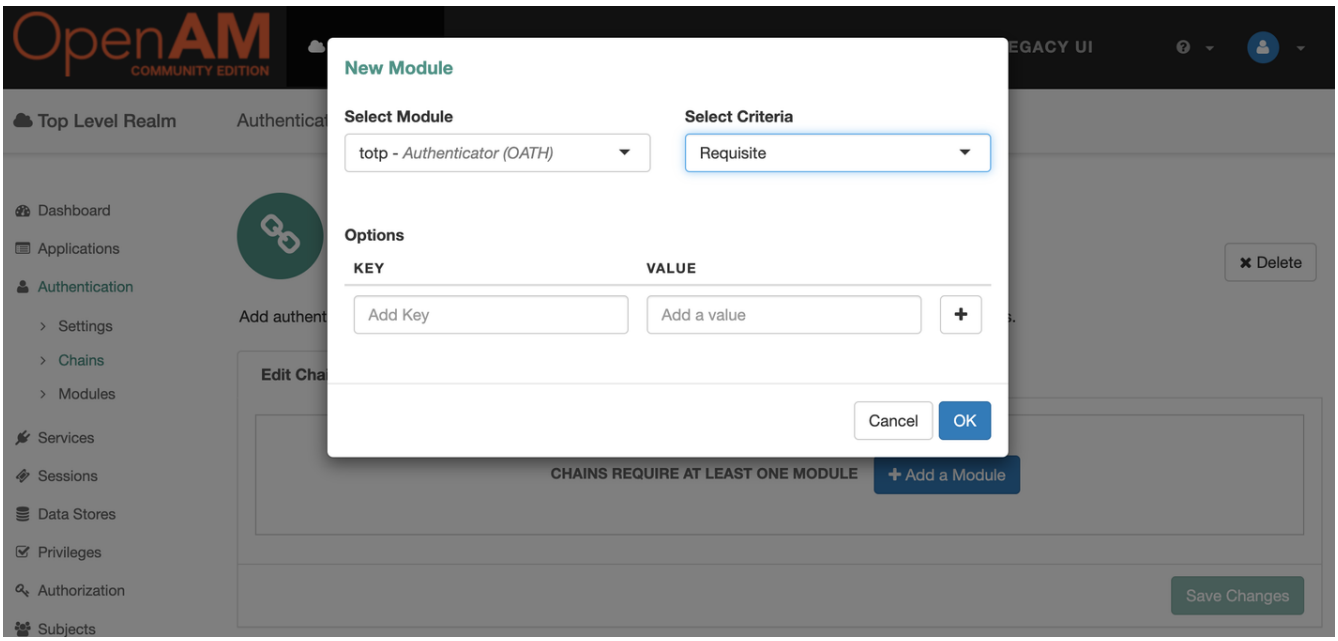
Configuring the Device Registration Chain

The registration chain is required so that authenticated users can add a second authentication factor using Google Authenticator.

In the administrator console, in the realm settings in the left menu, select Authentication → Chains and click the `Add Chain` button in the list that opens.

Enter the chain name `totp-register` and click the `Create` button.

In the chain settings, click the `Add a Module` button and add the created `totp` authentication module as shown in the figure. Click the `OK` button and then `Save Changes`.



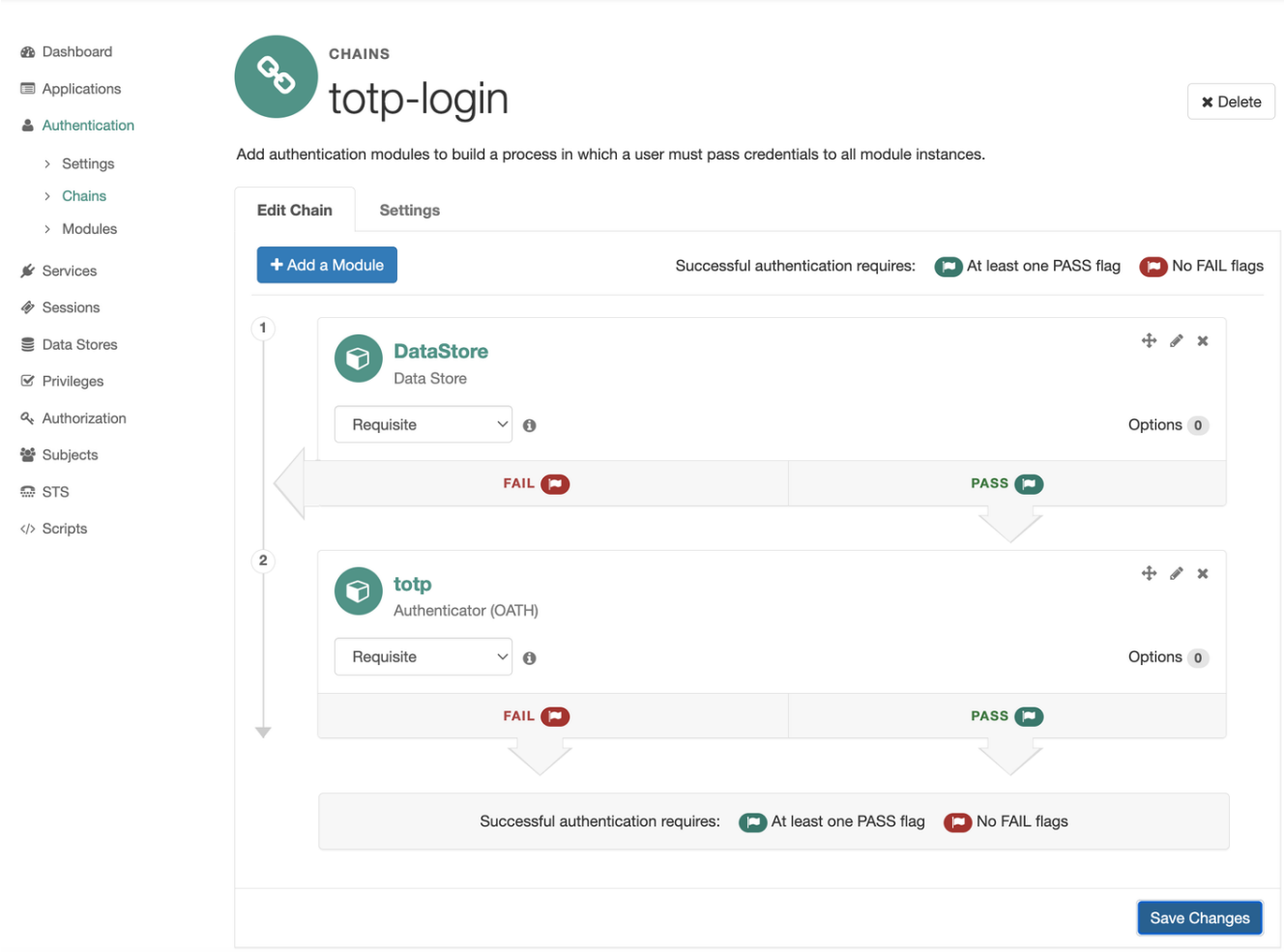
Configuring the Authentication Chain

In this chain, we will configure authentication so that after authenticating with a username and password, the user is required to enter a one-time password from the Google Authenticator mobile app.

In the administrator console, in the realm settings in the left menu, select Authentication → Chains and click the `Add Chain` button in the list that opens.

Enter the chain name `totp-login` and click the `Create` button. First, add the `DataStore` login and password authentication module. Then add the `totp` one-time code authentication module.

Click `Save Changes`.



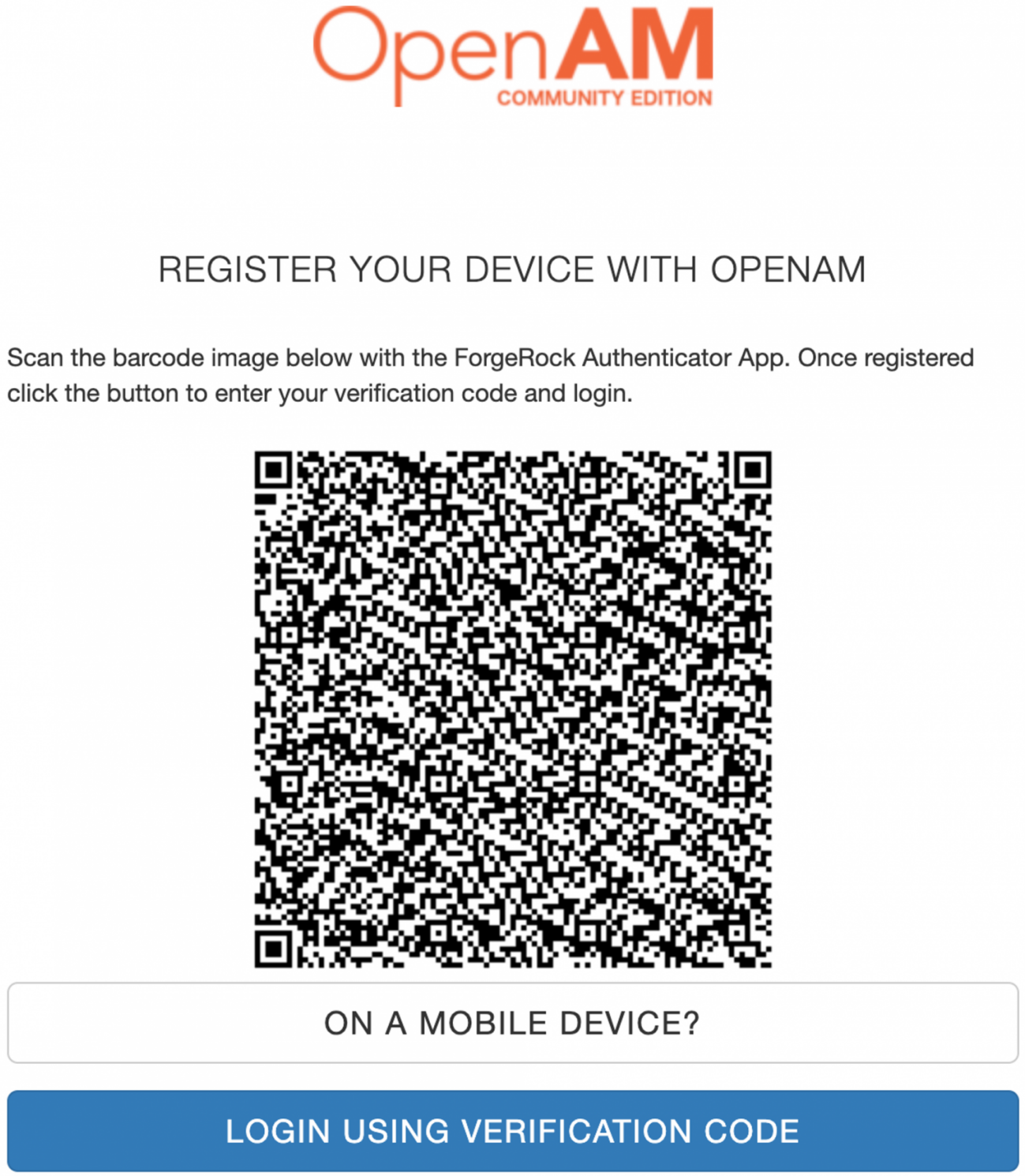
Setting up Google Authenticator.

Download the Google Authenticator app from the appropriate app store for your device.

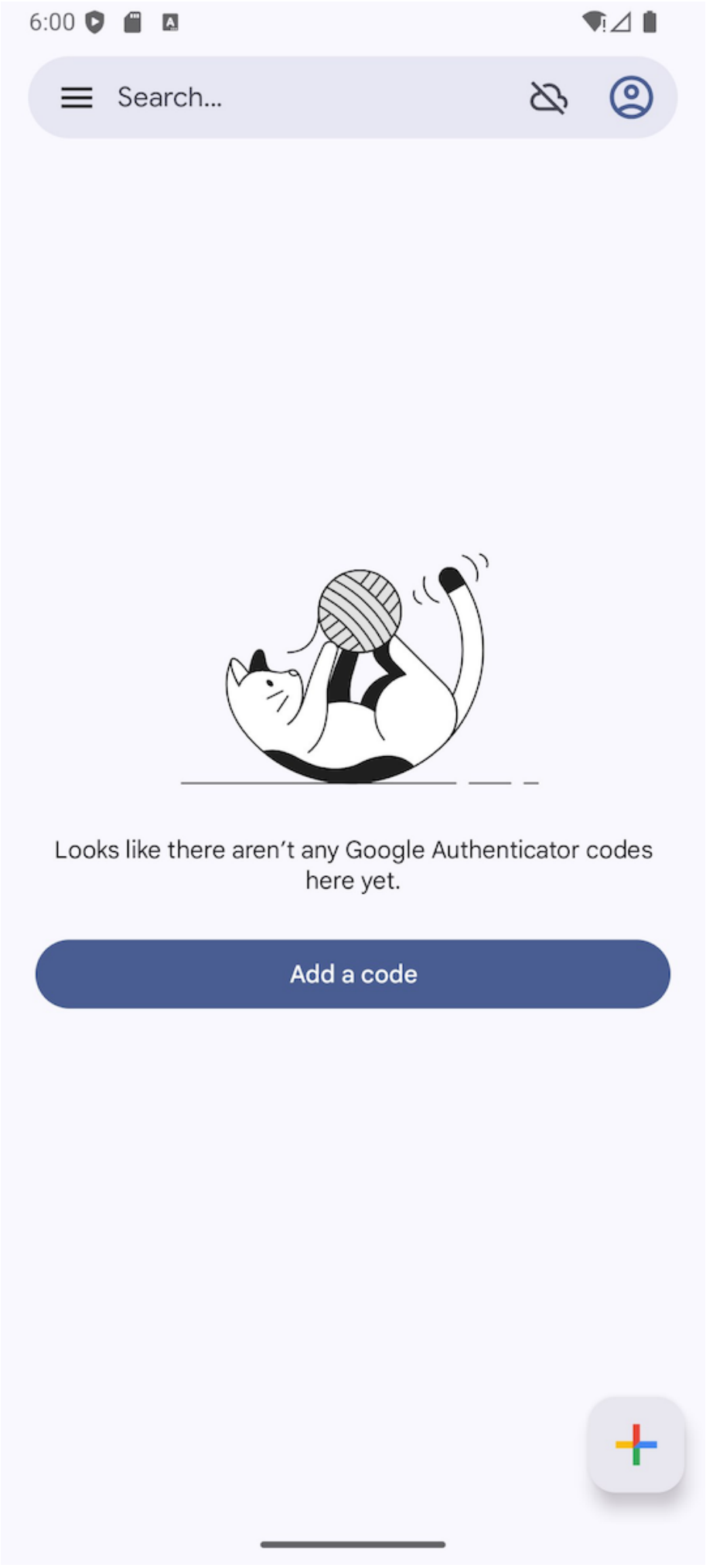
Registering a Device

Sign in to the console with a test user account. To do this, either sign out of the administrator console or open your browser in incognito mode. Go to the URL http://openam.example.org:8080/openam/XUI/#login/ and log in to OpenAM with the `demo` account. The default password is `changeit`.

After successful authentication, open the device registration chain link in your browser: http://openam.example.org:8080/openam/XUI/#login&service=totp-register.

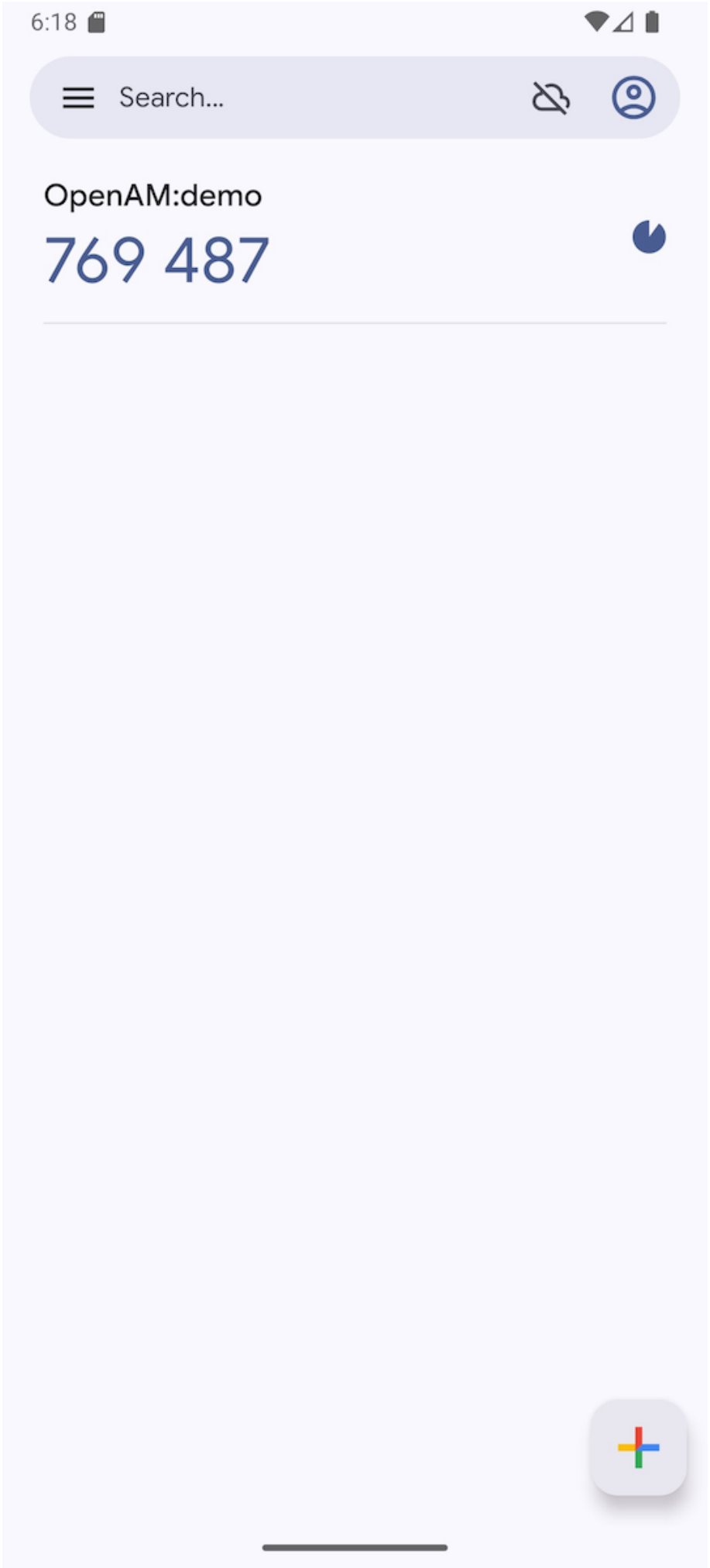


Open the Goole Authenticator app and tap the `Add code` button.


6:00
Search...
Looks like there aren't any Google Authenticator codes here yet.
Add a code

You will be prompted to scan a QR code. Scan the QR code displayed in the OpenAM browser window. After scanning, the OpenAM account will be added to the Google Authenticator app.

Locate the newly added account. A one-time password will be displayed.


6:18
Search...
OpenAM:demo
769 487

In your browser, click the `Login Using Verification Code` button.

Enter the one-time password from the mobile app and click the `Submit` button.

Authentication with One-Time Password

Exit the OpenAM console or open your browser in incognito mode. Follow the link http://openam.example.org:8080/openam/XUI/#login&service=totp-login. Enter the username and password for the `demo` account. After entering the username and password, OpenAM will request a one-time password from the mobile app. Open the mobile app, select the `demo` user account. Enter the one-time password from the mobile app in the browser, and click the `Submit` button. If the one-time password is correct, you will be authenticated successfully.

Using Microsoft Authenticator with OpenAM

This article is intended for technical specialists or security system architects who want to implement two-factor authentication (2FA) in an access control system to enhance the security of user accounts.

Adding a second factor makes it significantly more difficult for attackers to compromise accounts.

Stack Used

OpenAM is an open-source access management system. It is designed for centralized management of authentication, authorization, and user accounts.

Microsoft Authenticator is a mobile application designed to be used as an additional authentication factor. It supports push notifications, one-time passwords (TOTP), and biometric authentication.

In this article, we will deploy OpenAM, configure modules and authentication chains for use with Microsoft Authenticator, and show you how to add a second factor authentication for a user.

We will use authentication with one-time passwords generated using the TOTP (time-based one-time password) protocol. Such passwords do not need to be sent to the client device via SMS or PUSH notifications. These passwords are generated using a specific cryptographic algorithm directly on the device.

Installing OpenAM

If you do not have OpenAM installed yet, you can deploy a Docker container as described in the OpenAM wiki article

Configuring OpenAM

We will configure the module and authentication chain.

The authentication module in OpenAM is responsible for a specific authentication method. This can be authentication with a username and password, via the Kerberos protocol, or using biometrics. Modules can be organized into chains. This allows you to build chains of modules to authenticate users in multiple stages or using different methods. For example, if seamless authentication via the Kerberos protocol fails, you can ask the user for their login and password.

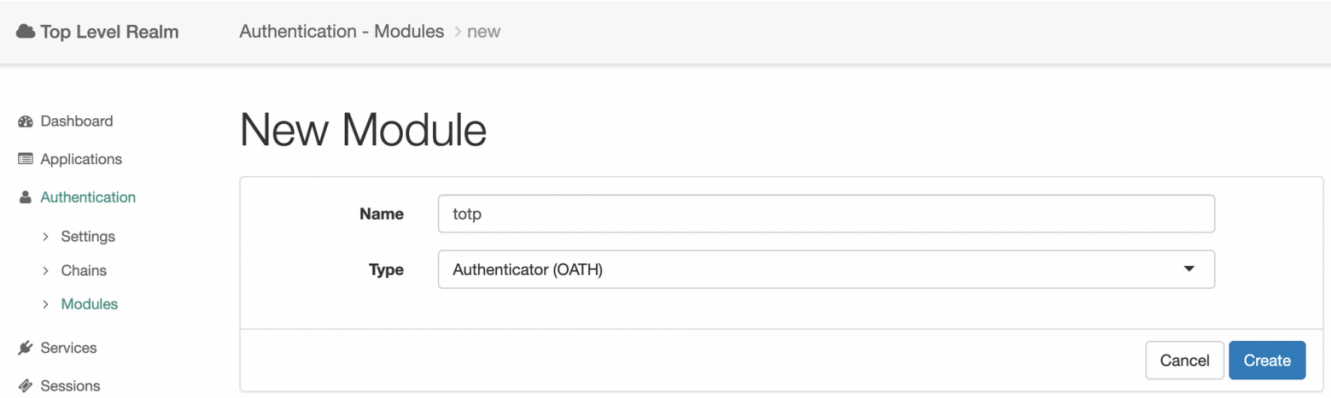
Configuring the TOTP Authentication Module

Open the administrator console at http://openam.example.org:8080/openam/console

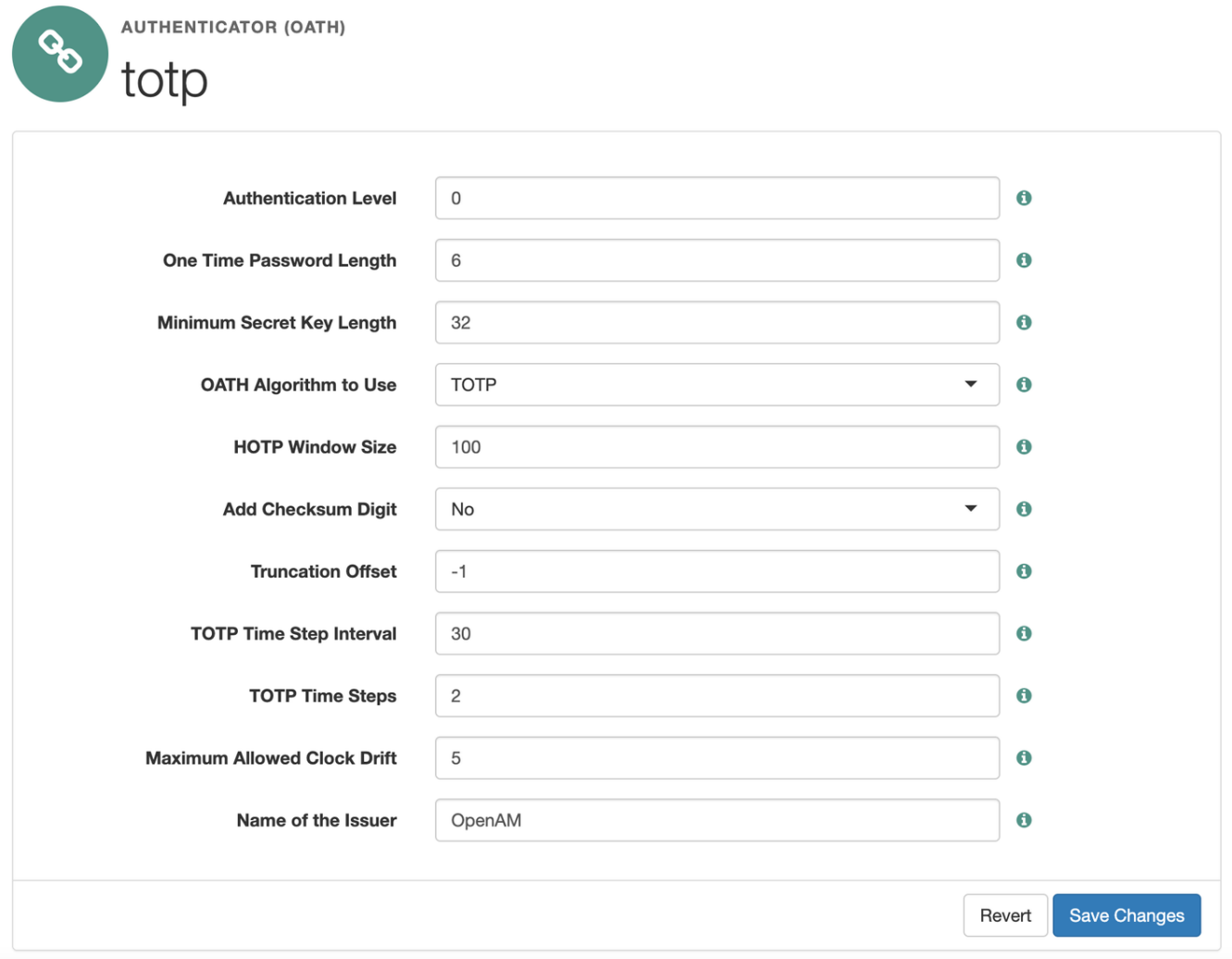
In the login field, enter `amadmin`. In the password field, enter the administrator password specified during installation.

Open the root realm, select Authentication → Modules in the left menu, and click the `Add Module`

button. In the form that appears, enter the module name, for example `totp`, and the module type - `Authenticator (OATH)`. Click the `Create` button.



Set the `OATH Algorithm to Use` setting to `TOTP`, enter any non-empty value in the `Name of the Issuer` field, for example `OpenAM`, and click `Save Changes`.



Configuring the Device Registration Chain

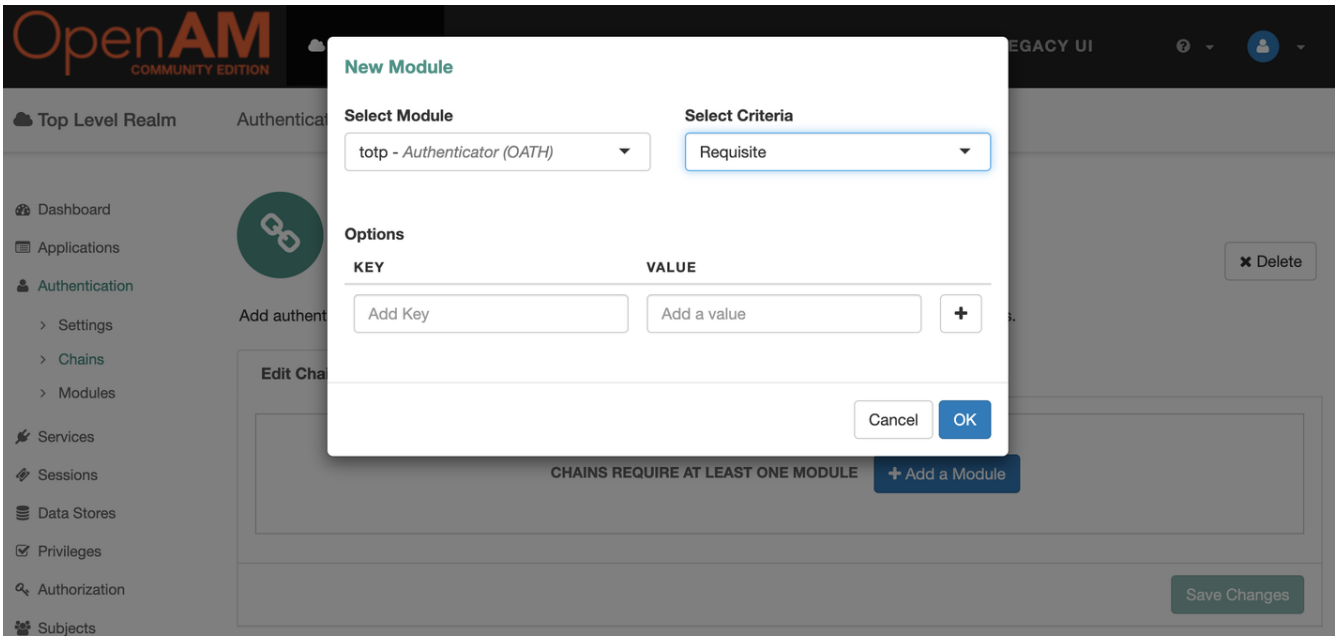
The registration chain is required so that authenticated users can add a second authentication factor using Microsoft Authenticator.

In the administrator console, in the realm settings in the left menu, select Authentication → Chains

and click the `Add Chain` button in the list that opens.

Enter the chain name `totp-register` and click the `Create` button.

In the chain settings, click the `Add a Module` button and add the created `totp` authentication module as shown in the figure. Click the `OK` button and then `Save Changes`.



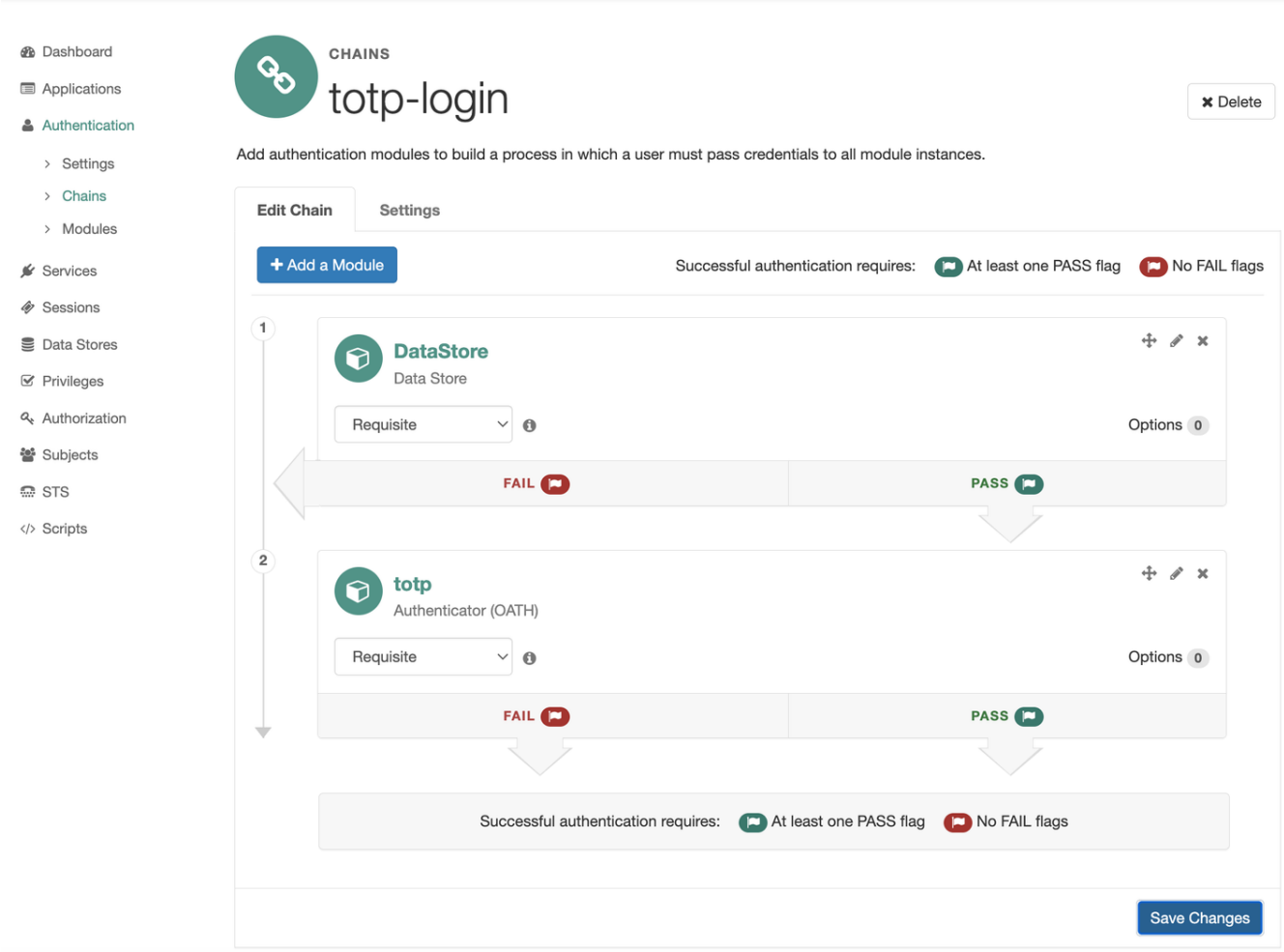
Configuring the Authentication Chain

In this chain, we will configure authentication so that after authenticating with a username and password, the user is required to enter a one-time password from the Microsoft Authenticator mobile app.

In the administrator console, in the realm settings in the left menu, select Authentication → Chains and click the `Add Chain` button in the list that opens.

Enter the chain name `totp-login` and click the `Create` button. First, add the `DataStore` login and password authentication module. Then add the `totp` one-time code authentication module.

Click `Save Changes`.



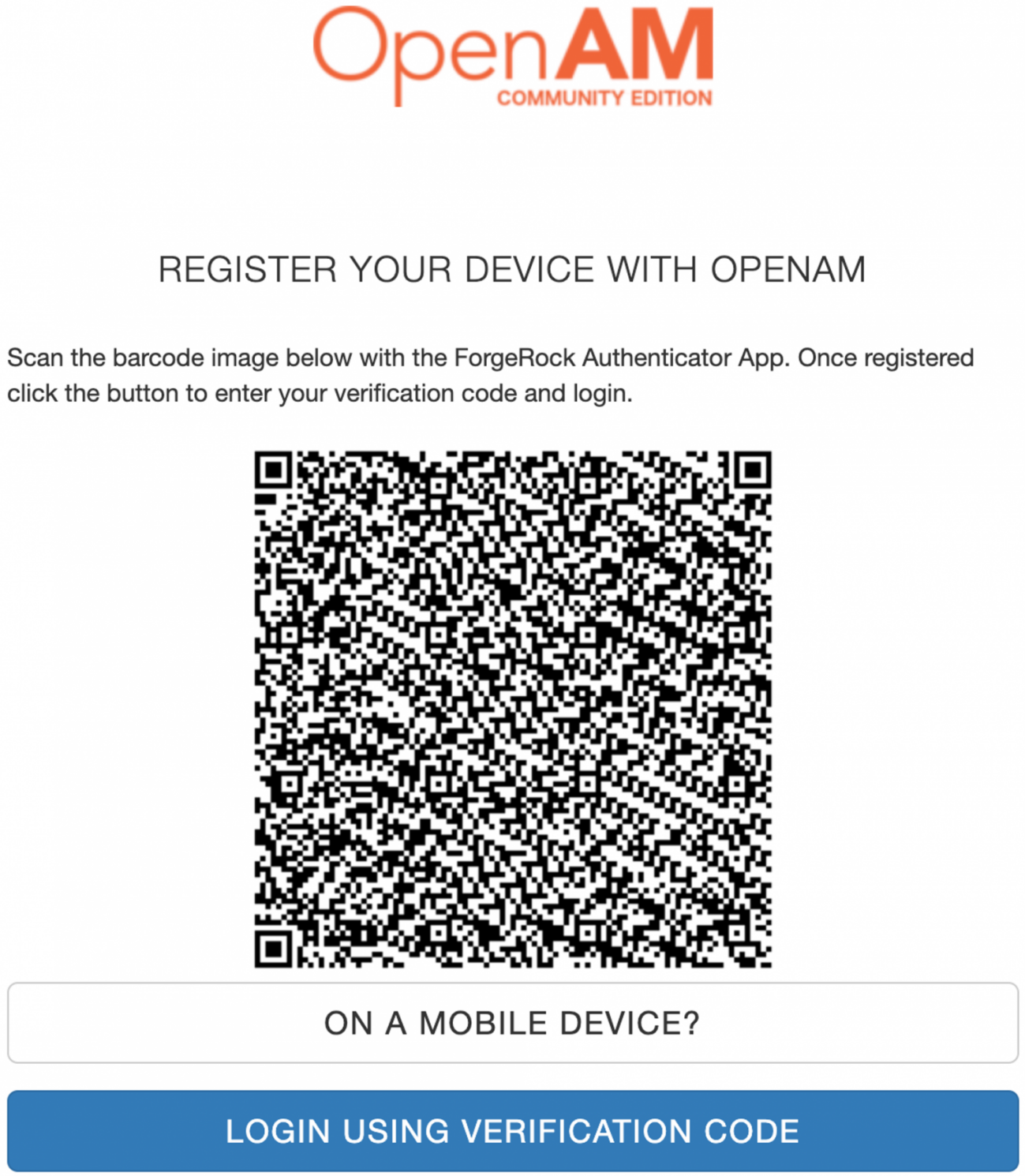
Setting up Microsoft Authenticator.

Download the Microsoft Authenticator app from the appropriate app store for your device.

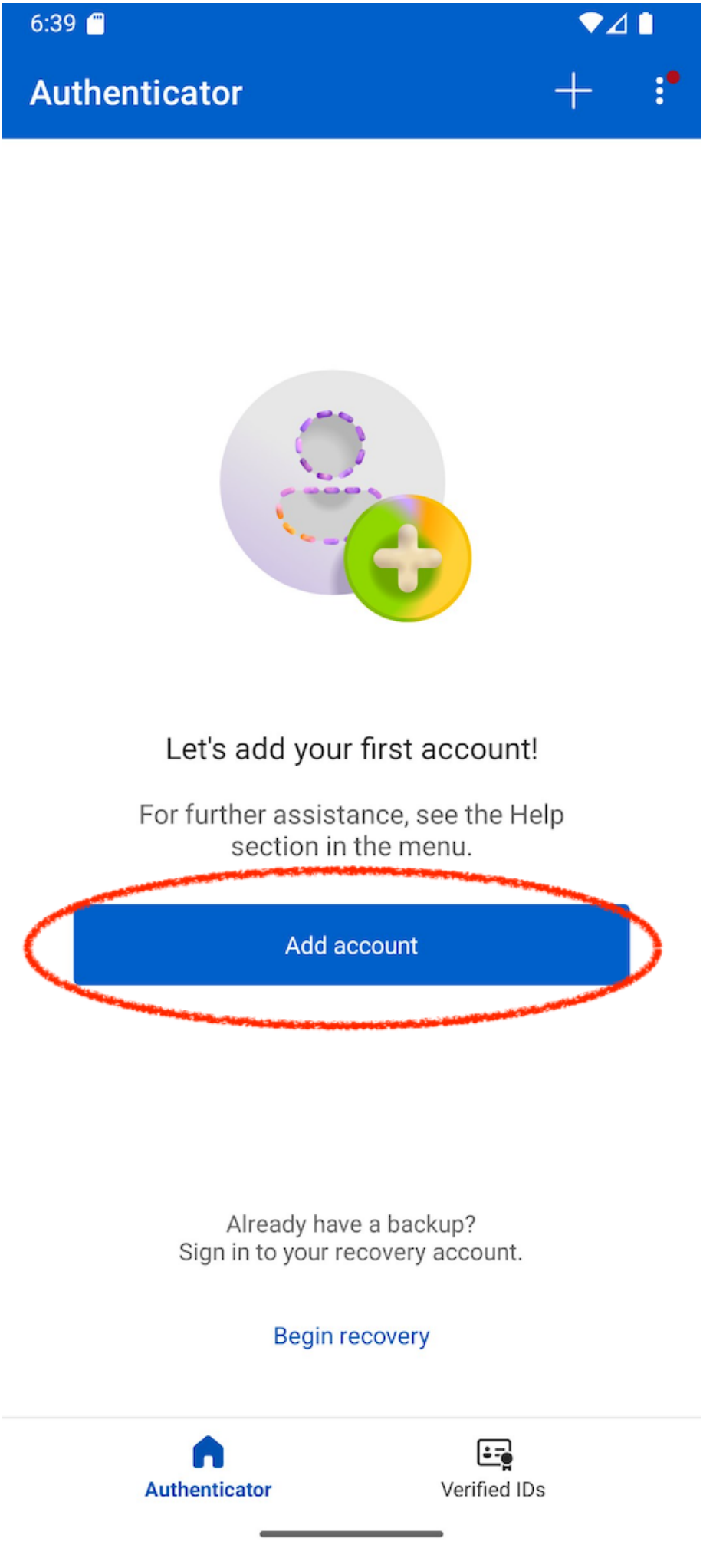
Registering a Device

Sign in to the console with a test user account. To do this, either sign out of the administrator console or open your browser in incognito mode. Go to the URL http://openam.example.org:8080/openam/XUI/#login/ and log in to OpenAM with the `demo` account. The default password is `changeit`.

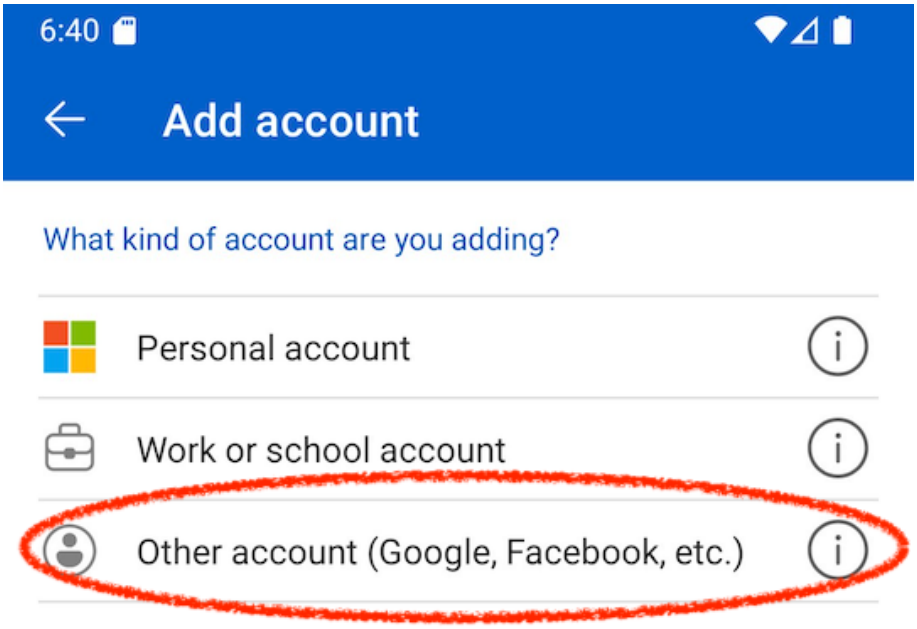
After successful authentication, open the device registration chain link in your browser: http://openam.example.org:8080/openam/XUI/#login&service=totp-register.



Open the Microsoft Authenticator app and tap the `Add account` button.


6:39
Authenticator
Let's add your first account!
For further assistance, see the Help section in the menu.
Add account
Already have a backup?
Sign in to your recovery account.
Begin recovery
Authenticator
Verified IDs

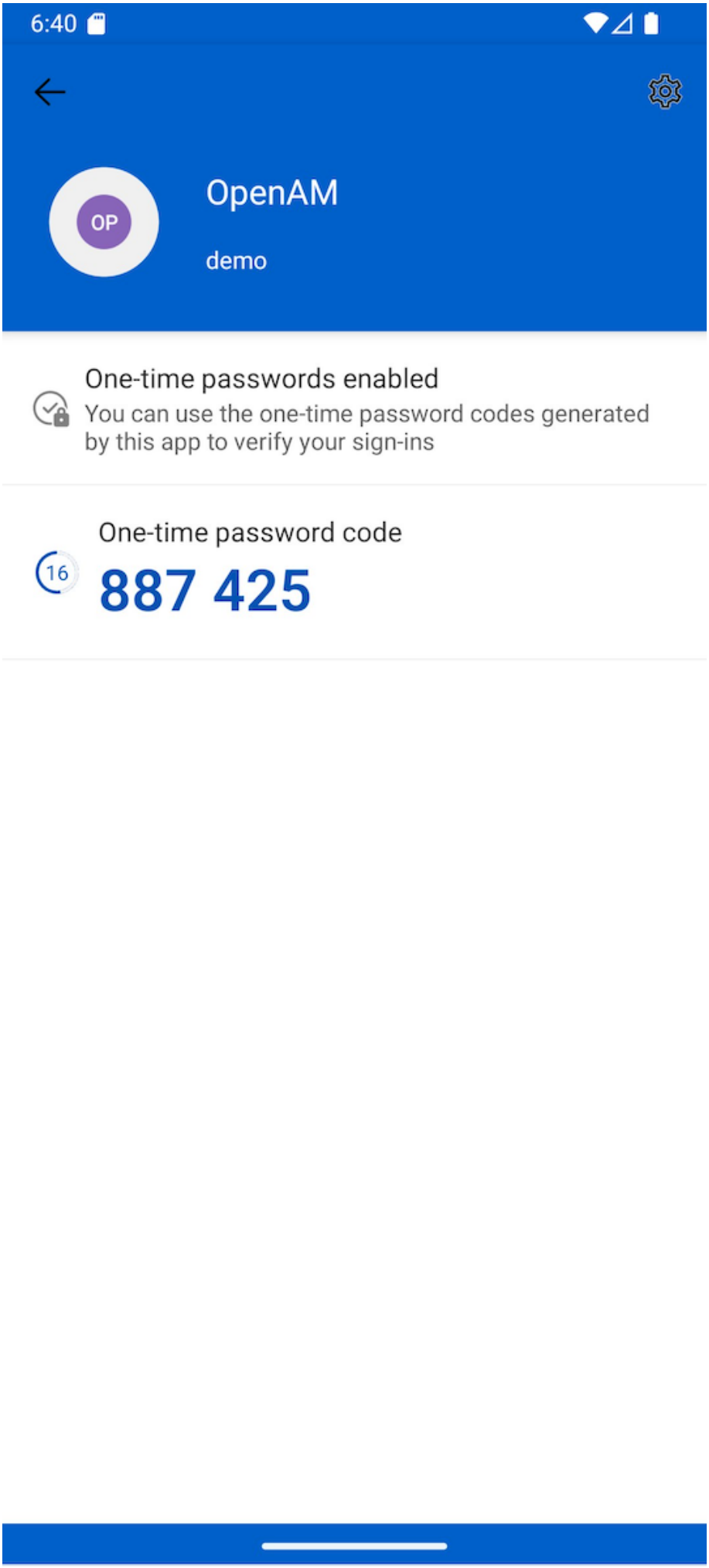
Select `Other account`.


6:40
Add account
What kind of account are you adding?
Personal account
Work or school account
Other account (Google, Facebook, etc.)

You will be prompted to scan a QR code. Scan the QR code displayed in the OpenAM browser window. After scanning, the OpenAM account will be added to the Microsoft Authenticator app.


Authenticator
OP
OpenAM
demo
Account added successfully
Authenticator
Verified IDs

Locate the newly added account. A one-time password will be displayed.


6:40
OP
OpenAM
demo
One-time passwords enabled
You can use the one-time password codes generated by this app to verify your sign-ins
One-time password code
16
887 425

In your browser, click the `Login Using Verification Code` button.

Enter the one-time password from the mobile app and click the `Submit` button.

Authentication with One-Time Password

Exit the OpenAM console or open your browser in incognito mode. Follow the link http://openam.example.org:8080/openam/XUI/#login&service=totp-login. Enter the username and password for the `demo` account. After entering the username and password, OpenAM will request a one-time password from the mobile app. Open the mobile app, select the `demo` user account. Enter the one-time password from the mobile app in the browser, and click the `Submit` button. If the one-time password is correct, you will be authenticated successfully.

What is Single Sign On and How does it Work

- What is Single Sign-On
- How does Single Sign On Work

What is Single Sign-On

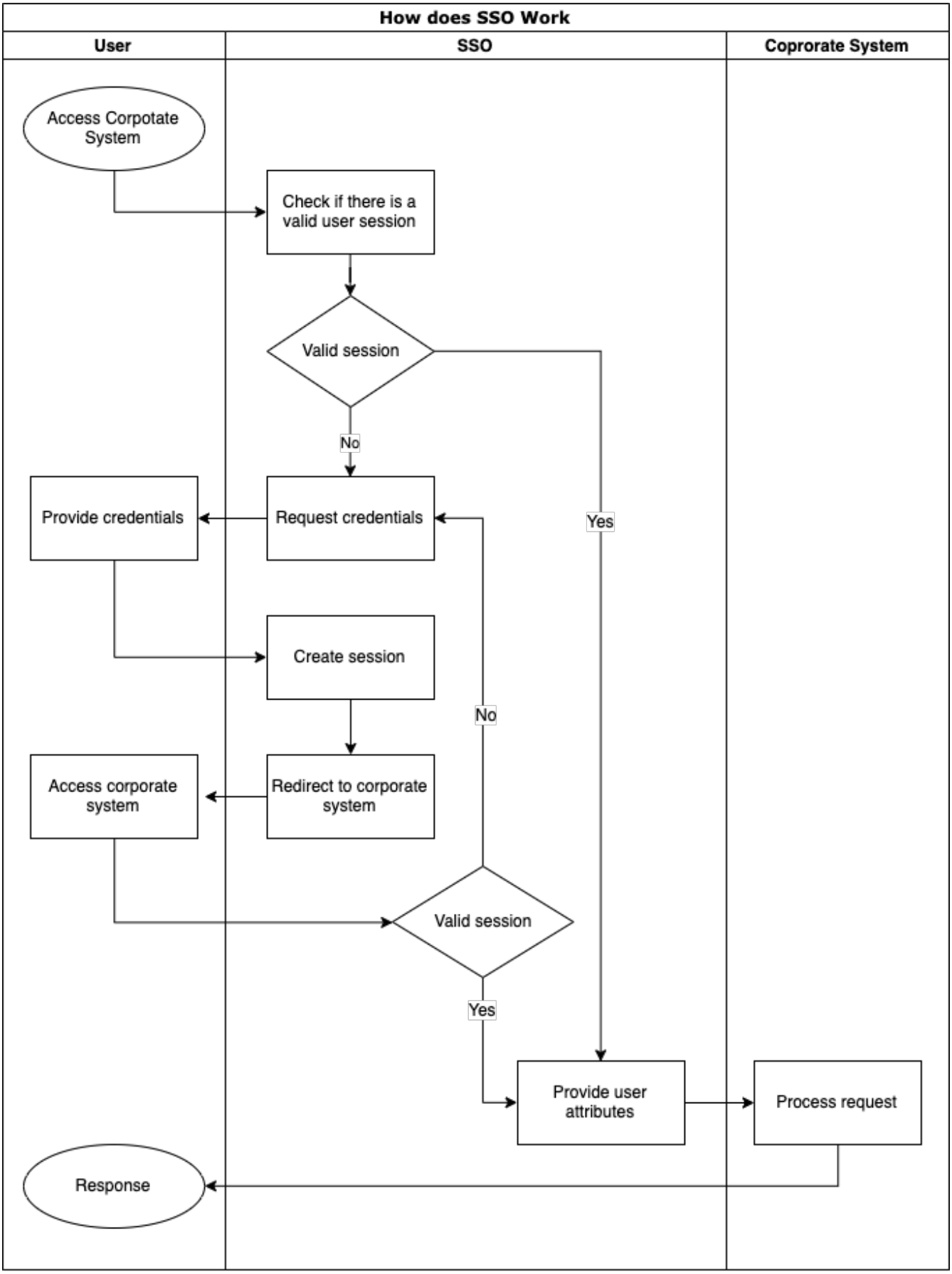
Single Sign-On (SSO) is a technology that allows users to authenticate into various systems using single credentials.

For example, Acme Corp uses various information systems and each system has its user database and its login processes. So, when Acme Corp hires a new employee, they should create an account for each system. To use each system, an employee should enter his credentials into each system as well. If the employee getting fired or promoted to a new position, all user accounts for the employee should also be changed. With an increasing number of employees and a number of information systems in the organization, user account management becomes more difficult and expensive.

To simplify user account management, Acme Corp starts to use the SSO system. So, when an employee authenticates the operating system, he could use his operating system account to access all other information systems. If the employee got fired or promoted or just wants to change his password, only one user account should be modified. When Acme Corp stars using new software, they just set up it to use SSO for authentication and there's no need to create user database in this software.

How does Single Sign On Work

When an employee tries to login into an information system, SSO checks if he already logged in in SSO. If the employee is not logged in, SSO asks the employee for authentication. After authentication, SSO creates a session for the employee and redirects him to the desired system. If the employee is authenticated in SSO, SSO provides account information to the desired system, the employee tries to access.


How does SSO Work
User
SSO
Coprorate System
Access Corpotate System
Check if there is a valid user session
Valid session
No
Yes
Provide credentials
Request credentials
Create session
No
Access corporate system
Redirect to corporate system
Valid session
Yes
Provide user attributes
Process request
Response