**Custom OpenID Connect (OIDC) Harvester for GeoNetwork**

This document walks through building a Java-based custom GeoNetwork harvester that uses OpenID Connect (OIDC) for authentication, handles token refresh, ingests metadata into the GeoNetwork database upon successful authentication, and exposes an Admin UI via XML configuration and an Angular template. Robust error handling is included in each component.

**1. Prerequisites & Dependencies**

1. **GeoNetwork 4.4.6** (or later) installed.
2. **Maven** build tool.
3. **Java 11+** SDK.
4. **OIDC library**: e.g., [Nimbus OAuth 2.0 SDK & OpenID Connect SDK](https://connect2id.com/products/nimbus-oauth-openid-connect-sdk).
5. **Angular 10+** (GeoNetwork UI uses Angular).
6. **Logging**: SLF4J + Logback.

Add to your pom.xml:

<dependencies>

<!-- GeoNetwork Harvest API -->

<dependency>

<groupId>org.geonetwork</groupId>

<artifactId>geonetwork-api</artifactId>

<version>${geonetwork.version}</version>

</dependency>

<!-- Nimbus OIDC -->

<dependency>

<groupId>com.nimbusds</groupId>

<artifactId>oauth2-oidc-sdk</artifactId>

<version>9.25</version>

</dependency>

<!-- SLF4J + Logback -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.32</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.6</version>

</dependency>

</dependencies>

**2. Project Structure**

custom-oidc-harvester/

├── pom.xml

├── src/main/java/org/geonetwork/harvester/oidc/

│ ├── OidcAuthService.java

│ ├── TokenManager.java

│ ├── OidcHarvester.java

│ └── OidcHarvesterFactory.java

├── src/main/resources/config/services/harvesters/

│ └── h.harvester.oidc.xml

└── src/main/resources/catalog-ui/src/app/harvest-oidc/

├── harvest-oidc.component.ts

├── harvest-oidc.component.html

└── harvest-oidc.module.ts

**3. OIDC Authentication Service**

**OidcAuthService.java**: Handles initial authentication and token refresh.

package org.geonetwork.harvester.oidc;

import com.nimbusds.oauth2.sdk.TokenRequest;

import com.nimbusds.oauth2.sdk.TokenResponse;

import com.nimbusds.oauth2.sdk.TokenErrorResponse;

import com.nimbusds.oauth2.sdk.token.Tokens;

import com.nimbusds.oauth2.sdk.token.BearerAccessToken;

import com.nimbusds.oauth2.sdk.id.ClientID;

import com.nimbusds.oauth2.sdk.auth.ClientSecretBasic;

import com.nimbusds.oauth2.sdk.http.HTTPResponse;

import com.nimbusds.oauth2.sdk.token.RefreshToken;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.net.URI;

import java.time.Instant;

public class OidcAuthService {

private static final Logger log = LoggerFactory.getLogger(OidcAuthService.class);

private final URI tokenEndpoint;

private final ClientID clientId;

private final ClientSecretBasic clientAuth;

public OidcAuthService(URI tokenEndpoint, String clientId, String clientSecret) {

this.tokenEndpoint = tokenEndpoint;

this.clientId = new ClientID(clientId);

this.clientAuth = new ClientSecretBasic(this.clientId, new com.nimbusds.oauth2.sdk.auth.Secret(clientSecret));

}

public Tokens authenticate(String username, String password) throws OidcException {

try {

TokenRequest request = new TokenRequest(

tokenEndpoint,

clientAuth,

new com.nimbusds.oauth2.sdk.ResourceOwnerPasswordCredentialsGrant(username, new com.nimbusds.oauth2.sdk.id.Secret(password))

);

HTTPResponse httpResponse = request.toHTTPRequest().send();

TokenResponse tokenResponse = TokenResponse.parse(httpResponse);

if (tokenResponse instanceof TokenErrorResponse) {

throw new OidcException("Authentication failed: " + ((TokenErrorResponse) tokenResponse).getErrorObject());

}

return ((com.nimbusds.oauth2.sdk.AccessTokenResponse) tokenResponse).getTokens();

} catch (Exception e) {

log.error("Error during OIDC authentication", e);

throw new OidcException("OIDC authentication error", e);

}

}

public Tokens refresh(RefreshToken refreshToken) throws OidcException {

try {

TokenRequest request = new TokenRequest(

tokenEndpoint,

clientAuth,

new com.nimbusds.oauth2.sdk.RefreshTokenGrant(refreshToken)

);

HTTPResponse httpResponse = request.toHTTPRequest().send();

TokenResponse tokenResponse = TokenResponse.parse(httpResponse);

if (tokenResponse instanceof TokenErrorResponse) {

throw new OidcException("Token refresh failed: " + ((TokenErrorResponse) tokenResponse).getErrorObject());

}

return ((com.nimbusds.oauth2.sdk.AccessTokenResponse) tokenResponse).getTokens();

} catch (Exception e) {

log.error("Error refreshing OIDC token", e);

throw new OidcException("OIDC token refresh error", e);

}

}

}

// Custom exception

public class OidcException extends Exception {

public OidcException(String msg) { super(msg); }

public OidcException(String msg, Throwable cause) { super(msg, cause); }

}

**4. Token Manager**

**TokenManager.java**: Caches tokens, checks expiry, refreshes as needed.

package org.geonetwork.harvester.oidc;

import com.nimbusds.oauth2.sdk.token.Tokens;

import com.nimbusds.oauth2.sdk.token.AccessToken;

import java.time.Instant;

public class TokenManager {

private Tokens tokens;

private Instant expiry;

private final OidcAuthService authService;

public TokenManager(OidcAuthService authService) {

this.authService = authService;

}

public synchronized AccessToken getAccessToken() throws OidcException {

if (tokens == null || Instant.now().isAfter(expiry.minusSeconds(30))) {

if (tokens == null) {

throw new OidcException("No tokens available; authenticate first.");

}

tokens = authService.refresh(tokens.getRefreshToken());

computeExpiry(tokens.getAccessToken());

}

return tokens.getAccessToken();

}

public synchronized void authenticate(String user, String pass) throws OidcException {

tokens = authService.authenticate(user, pass);

computeExpiry(tokens.getAccessToken());

}

private void computeExpiry(AccessToken token) {

this.expiry = Instant.now().plusSeconds(token.getLifetime());

}

}

**5. Harvester Implementation**

**OidcHarvesterFactory.java**: Registers the harvester to GeoNetwork’s harvesting framework.

package org.geonetwork.harvester.oidc;

import org.geonetwork.harvest.api.AbstractHarvesterFactory;

import org.fao.geonet.kernel.harvest.HarvesterManager;

import org.springframework.stereotype.Component;

@Component

public class OidcHarvesterFactory extends AbstractHarvesterFactory {

public OidcHarvesterFactory(HarvesterManager hm) {

super("oidc", "OIDC Harvester", hm);

}

@Override

public OidcHarvester create(String id) {

return new OidcHarvester(id);

}

}

**OidcHarvester.java**: Core logic.

package org.geonetwork.harvester.oidc;

import org.fao.geonet.kernel.harvest.harvester.Harvester;

import org.fao.geonet.kernel.harvest.HarvesterInput;

import org.fao.geonet.kernel.harvest.HarvesterOutput;

import org.fao.geonet.kernel.harvest.HarvestResult;

import org.fao.geonet.repository.MetadataRepository;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class OidcHarvester implements Harvester {

private static final Logger log = LoggerFactory.getLogger(OidcHarvester.class);

private final String id;

private final TokenManager tokenManager;

public OidcHarvester(String id) {

this.id = id;

this.tokenManager = new TokenManager(

new OidcAuthService(

URI.create("https://keycloak.example.com/auth/realms/geo/protocol/openid-connect/token"),

"client-id", "client-secret"

)

);

}

@Override

public HarvestResult harvest(HarvesterInput input) {

HarvestResult result = new HarvestResult();

try {

tokenManager.authenticate(input.getUsername(), input.getPassword());

String endpoint = input.getUrl();

String metadataXml = fetchMetadata(endpoint, tokenManager.getAccessToken().getValue());

ingestMetadata(metadataXml);

result.setSuccess();

} catch (OidcException e) {

log.error("Authentication or token error", e);

result.addError("Authentication failed: " + e.getMessage());

} catch (Exception e) {

log.error("General harvesting error", e);

result.addError("Harvesting failed: " + e.getMessage());

}

return result;

}

private String fetchMetadata(String url, String bearerToken) throws IOException {

// HTTP GET with Authorization: Bearer <token>

HttpURLConnection conn = (HttpURLConnection) new URL(url).openConnection();

conn.setRequestProperty("Authorization", "Bearer " + bearerToken);

if (conn.getResponseCode() != 200) {

throw new IOException("Metadata fetch HTTP " + conn.getResponseCode());

}

try (InputStream in = conn.getInputStream()) {

return new String(in.readAllBytes(), StandardCharsets.UTF\_8);

}

}

private void ingestMetadata(String xml) throws Exception {

// Use GeoNetwork’s MetadataRepository to save

MetadataRepository repo = ...; // inject via Spring

// parse xml into Metadata entity

// repo.save(parsedMetadata);

}

@Override public String getName() { return "OIDC Harvester"; }

@Override public String getId() { return id; }

@Override public void stop() { /\* no-op \*/ }

}

**6. Service Registration XML**

Place under config/services/harvesters/h.harvester.oidc.xml:

<service name="harvest.oidc" class="org.geonetwork.harvester.oidc.OidcHarvesterFactory"/>

**7. Admin UI Integration (Angular)**

**Module**: harvest-oidc.module.ts

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { HarvestOidcComponent } from './harvest-oidc.component';

import { FormsModule } from '@angular/forms';

@NgModule({

declarations: [HarvestOidcComponent],

imports: [CommonModule, FormsModule],

exports: [HarvestOidcComponent]

})

export class HarvestOidcModule {}

**Component**: harvest-oidc.component.ts

import { Component } from '@angular/core';

import { HttpClient } from '@angular/common/http';

@Component({

selector: 'gn-harvest-oidc',

templateUrl: './harvest-oidc.component.html'

})

export class HarvestOidcComponent {

url = '';

username = '';

password = '';

result = '';

error = '';

constructor(private http: HttpClient) {}

runHarvest() {

this.result = this.error = '';

this.http.post('/api/harvest/oidc', {

url: this.url,

username: this.username,

password: this.password

}).subscribe(

(res: any) => this.result = 'Success: ' + res.message,

err => this.error = 'Error: ' + err.error.message

);

}

}

**Template**: harvest-oidc.component.html

<div class="gn-admin-panel">

<h3>OIDC Harvester</h3>

<label>Endpoint URL: <input [(ngModel)]="url"/></label>

<label>Username: <input [(ngModel)]="username"/></label>

<label>Password: <input type="password" [(ngModel)]="password"/></label>

<button (click)="runHarvest()">Run Harvest</button>

<div \*ngIf="result" class="gn-success">{{ result }}</div>

<div \*ngIf="error" class="gn-error">{{ error }}</div>

</div>

**Registration**: add to GeoNetwork’s Admin UI XML (e.g., config/catalog-ui/src/app/app.module.ts) to import and register the module and its route under /admin/harvest-oidc.

**8. Deployment & Testing**

1. Build the JAR: mvn clean package.
2. Drop the JAR into WEB-INF/lib of your GeoNetwork webapp.
3. Restart GeoNetwork.
4. In Admin UI, navigate to **Harvesters → OIDC Harvester**.
5. Enter credentials and endpoint; click **Run Harvest**.
6. Verify metadata appears in **Catalog → Metadata**.

*All code above includes explicit error handling in try-catch blocks, SLF4J logging, and user-facing error messages in the Admin UI.*

**9. Usage Examples: CSW API Calls**

Below are two illustrative scenarios showing how the OIDC-enabled harvester can be used to fetch records from a CSW endpoint requiring bearer‑token authentication.

**9.1. Java Code: Direct CSW GetRecords**

// 1. Authenticate via OIDC

TokenManager tokenManager = new TokenManager(

new OidcAuthService(

URI.create("https://keycloak.example.com/auth/realms/geo/protocol/openid-connect/token"),

"client-id", "client-secret"

)

);

try {

tokenManager.authenticate("user@example.com", "password123");

String accessToken = tokenManager.getAccessToken().getValue();

// 2. Build CSW GetRecords XML payload

String getRecordsRequest = """

<csw:GetRecords service="CSW" version="2.0.2" resultType="results"

outputSchema="http://www.isotc211.org/2005/gmd"

xmlns:csw="http://www.opengis.net/cat/csw/2.0.2">

<csw:Query typeNames="csw:Record">

<csw:ElementSetName>full</csw:ElementSetName>

</csw:Query>

</csw:GetRecords>

""";

// 3. Send HTTP POST with Authorization header

HttpURLConnection conn = (HttpURLConnection) new URL("https://remote-geonet.example.com/csw").openConnection();

conn.setRequestMethod("POST");

conn.setRequestProperty("Authorization", "Bearer " + accessToken);

conn.setRequestProperty("Content-Type", "application/xml");

conn.setDoOutput(true);

try(OutputStream out = conn.getOutputStream()) {

out.write(getRecordsRequest.getBytes(StandardCharsets.UTF\_8));

}

// 4. Read and process response

if (conn.getResponseCode() == 200) {

String responseXml = new String(conn.getInputStream().readAllBytes(), StandardCharsets.UTF\_8);

// parse responseXml and ingest individual metadata records as in ingestMetadata()

ingestMetadata(responseXml);

} else {

throw new IOException("CSW request failed: HTTP " + conn.getResponseCode());

}

} catch (OidcException | IOException e) {

log.error("CSW harvest error", e);

// report error in HarvestResult

}

**9.2. Using the Harvester Framework**

Configure a new OIDC harvester record in GeoNetwork with these parameters:

<harvester id="remoteCswOidc" type="oidc">

<url>https://remote-geonet.example.com/csw</url>

<username>user@example.com</username>

<password>password123</password>

<cswQuery><![CDATA[

<csw:GetRecords service="CSW" version="2.0.2" resultType="results"

outputSchema="http://www.isotc211.org/2005/gmd"

xmlns:csw="http://www.opengis.net/cat/csw/2.0.2">

<csw:Query typeNames="csw:Record">

<csw:ElementSetName>brief</csw:ElementSetName>

</csw:Query>

</csw:GetRecords>

]]></cswwQuery>

</harvester>

When you run this harvester, it:

1. Performs an OIDC login to obtain a bearer token.
2. Uses that token in an Authorization: Bearer <token> header on the HTTP POST to /csw with your cswQuery payload.
3. Parses the CSW XML response and ingests each <csw:Record> into GeoNetwork’s database.

These examples show how you can leverage the OIDC harvester to interact with any CSW‑compliant service secured by bearer‑token authentication.