MarkLogic with Keycloak Implementation Guide

Author: Martin Warnes, Principal Technical Support Engineer, Progress Software
Document Status: Draft
Last Modified: October 2, 2025

Introduction

This document provides comprehensive methods and procedures for implementing Keycloak as an identity provider for MarkLogic Application Servers. The guide covers the configuration and integration of Keycloak and MarkLogic to support both OAuth2 and SAML authentication protocols, enabling secure single sign-on (SSO) capabilities for MarkLogic applications. By following the procedures outlined in this document, organizations can leverage Keycloak's robust identity and access management features to enhance security, streamline user authentication, and provide a seamless user experience across their MarkLogic-based applications.

About MarkLogic

MarkLogic is an enterprise NoSQL database platform originally developed by MarkLogic Corporation and now owned by Progress Software. It is designed to handle complex, multi-structured data including documents, metadata, collections, and semantic triples. MarkLogic provides a unified platform for data integration, search, analytics, and application development, making it particularly suitable for content-driven applications and enterprise data hub architectures. The platform includes built-in search capabilities, ACID transactions, and enterprise-grade security features.

Website: https://www.progress.com/marklogic

About Keycloak

Keycloak is an open-source identity and access management solution developed by Red Hat. It provides comprehensive authentication and authorization services for applications and services, supporting industry-standard protocols including OAuth 2.0, OpenID Connect, and SAML 2.0. Keycloak offers features such as single sign-on (SSO), identity brokering, user federation, fine-grained authorization policies, and social login capabilities. As a Red Hat project, it benefits from enterprise support and continuous development by a vibrant open-source community.

Website: https://www.keycloak.org

Scope and Assumptions

This guide concentrates specifically on creating a Keycloak client for OAuth2 and SAML authentication with MarkLogic Application Servers. The following assumptions are made:

- Keycloak has already been installed and is running
- A Keycloak realm is already available and configured
- Keycloak users and roles for MarkLogic have been configured
- Basic familiarity with MarkLogic administration
- Administrative access to both Keycloak and MarkLogic systems

The document will focus on the client configuration process rather than the initial Keycloak installation or realm setup procedures.

Table of Contents

OAuth2 Integration

- Creating a Keycloak OpenID Connect Client
  - Keycloak OpenID Client Configuration Settings
  - Keycloak Roles Considerations
- Creating a MarkLogic External Security Profile
  - Obtaining the JWKS URI from Keycloak
  - Configuring the MarkLogic External Security Profile
  - Example Configuration Reference
  - Testing the Configuration
- OAuth2 Troubleshooting
  - MarkLogic Troubleshooting
  - Keycloak Troubleshooting
  - Common OAuth2 Integration Issues
  - Diagnostic Commands
  - Best Practices for Troubleshooting

SAML Integration

- SAML Integration Overview
- Creating a Keycloak SAML Client
  - Step 1: Create SAML Client in Keycloak

    - Step 2: Configure SAML Client Settings
    - Step 3: Configure SAML Attribute Mappings
- Creating MarkLogic SAML External Security Profile
    - Step 1: Obtain Keycloak SAML Metadata
    - Step 2: Configure MarkLogic External Security Profile
    - Step 3: Certificate Management
- SAML Testing and Validation
    - Testing SAML Authentication Flow
- SAML Troubleshooting
- Best Practices for SAML Implementation

Creating a Keycloak OpenID Connect Client

Create a Keycloak client for use by MarkLogic, the client should use "Standard Flow" Authentication.

Note: MarkLogic does not support "Implicit Flow" so the Access settings can be left unpopulated.

Keycloak OpenID Client Configuration Settings

The following screenshot shows an example configuration for a Keycloak client:

Clients are applications and services that can request authentication of a user.

Settings | Keys | Credentials | Roles | Client scopes | Service accounts roles | Sessions | Advanced | Events

Jump to section
- General settings
- Access settings
- Capability config
- Login settings
- Logout settings

General settings

Client ID * : marklogic-oauth

Name :

Description :

Always display in UI : Off

Access settings

Root URL :

Home URL :

Valid redirect URIs : /*
Add valid redirect URIs

Valid post logout redirect URIs :
Add valid post logout redirect URIs

Web origins : /*
Add web origins

Admin URL :

Capability config

Client authentication : On

Authorization : Off

Authentication flow :
- [x] Standard flow
- [x] Direct access grants
- [ ] Implicit flow
- [x] Service accounts roles
- [ ] Standard Token Exchange
- [ ] OAuth 2.0 Device Authorization Grant
- [ ] OIDC CIBA Grant

Login settings

Login theme : Choose...

Consent required : Off

Display client on screen : Off

Consent screen text :

Logout settings

Front channel logout : On

Front-channel logout URL :

Front-channel logout session required : On

Keycloak Roles Considerations

JSON Object Array Compatibility Issue

Keycloak returns user roles as JSON object arrays within the `realm_access` and `resource_access` sections of the OpenID access token. However, MarkLogic cannot directly process these nested JSON object arrays for role extraction, which creates a compatibility challenge.

OpenID Access Token Structure

A typical Keycloak OpenID access token contains roles in the following structure:

```
{
  "exp": 1759152574,
  "iat": 1759152274,
  "iss": "https://oauth.warnesnet.com:8443/realms/progress-marklogic",
  "realm_access": {
    "roles": ["marklogic-user", "marklogic-manage", "marklogic-admin"]
  },
  "resource_access": {
    "account": {
      "roles": ["manage-account", "view-profile"]
    }
  },
  "preferred_username": "martin",
  "email": "martin@example.com"
}
```

The Problem

- **`realm_access.roles`** : Contains an array of realm-level roles nested within a JSON object
- **`resource_access.[client].roles`** : Contains client-specific roles in a similar nested structure
- **MarkLogic Limitation**: MarkLogic's OAuth2 implementation cannot parse roles from these nested JSON object arrays

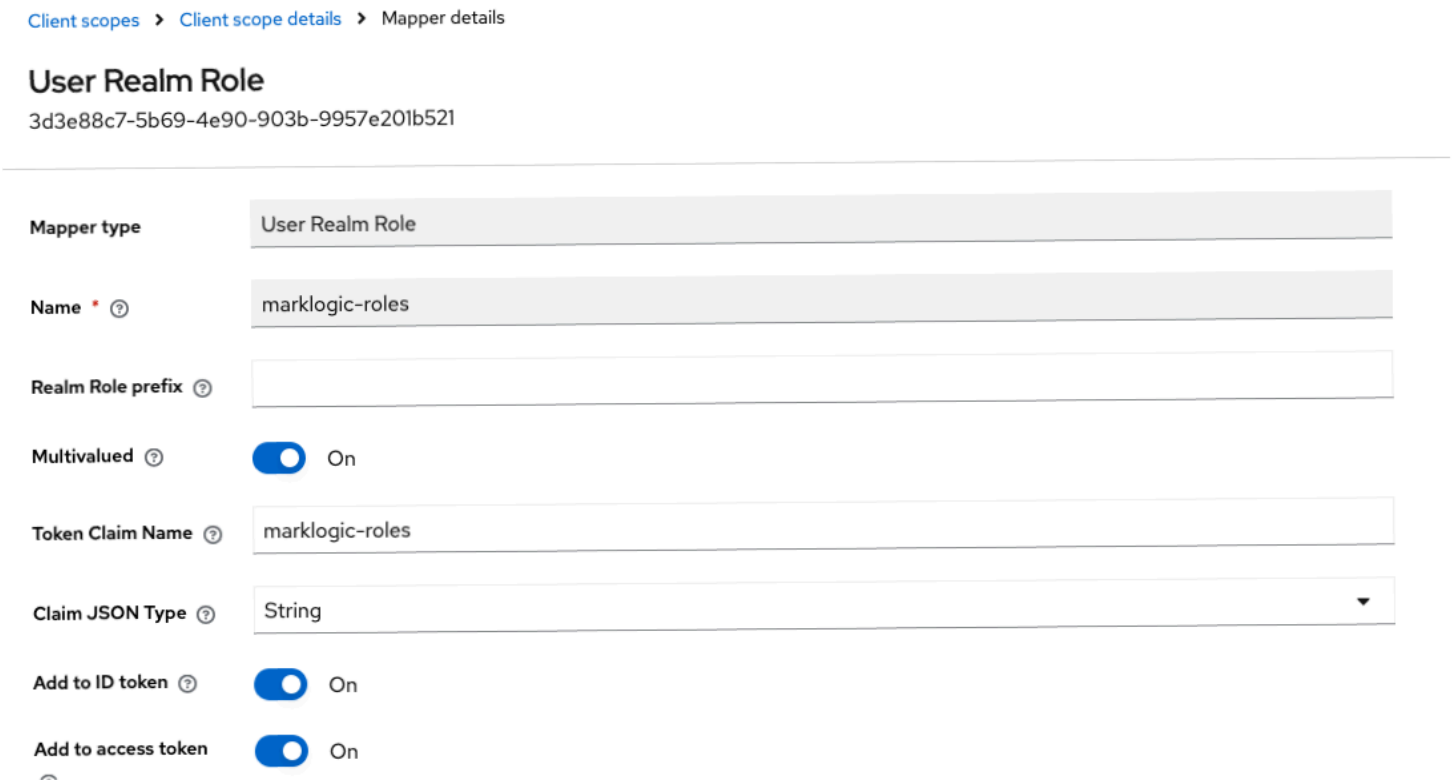
The Solution

To work around this limitation, Keycloak must be configured to provide roles in a format that MarkLogic can process. This typically involves:

1. **Custom Claim Mapping**: Creating a custom mapper that flattens the roles into a simple array
2. **Direct Role Claims**: Configuring a top-level claim (like `marklogic-roles` in the example above) that contains roles as a simple string array

3. **Protocol Mappers**: Using Keycloak's built-in mappers to transform the token structure

The following example demonstrates a Protocol Mapper that has been added to a Profile Scope, which extracts user Realm roles and returns them as a new claim using a string data type and name "marklogic-roles".



MarkLogic will be able to access these roles by referencing the claim in the External Security profile configuration.


```
{
  "exp": 1759152574,
  "iat": 1759152274,
  "iss": "https://oauth.warnesnet.com:8443/realms/progress-marklogic",
  "realm_access": {
    "roles": ["marklogic-user", "marklogic-manage", "marklogic-admin"]
  },
  "resource_access": {
    "account": {
      "roles": ["manage-account", "view-profile"]
    }
  },
  "marklogic-roles": ["marklogic-user", "marklogic-manage", "marklogic-admin"],
  "preferred_username": "martin",
  "email": "martin@example.com"
}
```

Creating a MarkLogic External Security Profile

Once the Keycloak client is configured with the appropriate role mappers, the next step is to create an External Security profile in MarkLogic that will integrate with the Keycloak identity provider.

Obtaining the JWKS URI from Keycloak

Before configuring the MarkLogic External Security profile, you need to obtain the JWKS (JSON Web Key Set) URI from your Keycloak realm. This URI is required by MarkLogic to validate the JWT tokens issued by Keycloak.

Steps to Find the JWKS URI:

1. **Access Keycloak Admin Console**: Log into your Keycloak admin interface
2. **Navigate to Realm Settings**: Select your realm (e.g., "progress-marklogic")
3. **Find OpenID Endpoint Configuration**: In the Realm Settings, look for the "OpenID Endpoint Configuration" link
4. **Access the Configuration**: Click the link to view the OpenID configuration JSON
5. **Locate JWKS URI**: In the configuration JSON, find the `jwks_uri` field

The JWKS URI typically follows this format:

```
https://your-keycloak-server:port/realms/your-realm-name/protocol/openid-connect/certs
```

Configuring the MarkLogic External Security Profile

Access the MarkLogic AdminUI interface and navigate to the External Security configuration page.

Required Configuration Settings:

1. **External Security Name**: Provide a descriptive name (e.g., "Keycloak-OAuth2")
2. **Authentication Protocol**: Select "oauth2"
3. **Cache Timeout**: Set according to your security requirements (typically 300-3600 seconds)
4. **Authorization**: Select "oauth2"
5. **OAuth2 Settings**:
    - **OAuth Flow Type**: Select "Resource Server"
    - **OAuth Vendor**: Select "Other"
    - **OAuth Client ID**: The client id from your Keycloak client configuration
    - **OAuth Token Type**: Select "JSON Web Tokens"

- **OAuth username attribute**: Enter the claim holding the userid from your Keycloak client configuration
- **OAuth role attribute**; Enter "marklogic-roles" (or the custom claim name you configured in Keycloak) to map Keycloak roles to MarkLogic roles as needed
- **OAuth JWT Algorithm**: Select "RS256"
- **JWKS URI**: The JWKS URI obtained from the Keycloak OpenID configuration

Example Configuration Reference

For reference, the following is an example of a configured OAUTH External Security profile:

Testing the Configuration

After creating the External Security profile, you can test the OAuth2 integration using various methods.

Manual Token Testing with curl

You can test the Keycloak token generation and MarkLogic integration using the following curl command to obtain a JWT token directly from Keycloak:

```
curl --location 'https://oauth.warnesnet.com:8443/realms/progress-marklogic/protocol/openic
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'username=martin' \
--data-urlencode 'password=sterling123' \
--data-urlencode 'client_id=marklogic-oauth' \
--data-urlencode 'client_secret=4UZyJkjWsGV5JtpsWfgkL1qW5vZ5hhmv' \
--data-urlencode 'grant_type=password' \
--data-urlencode 'scope=openid'
```

Parameter Explanations

- **URL**:
  `https://oauth.warnesnet.com:8443/realms/progress-marklogic/protocol/openid-connect/token`
  - The Keycloak token endpoint for your specific realm
- **Content-Type**: `application/x-www-form-urlencoded`
  - Required header for OAuth2 token requests
- **username**: `martin`
  - The Keycloak user account to authenticate
- **password**: `sterling123`
  - The user's password (Note: This is the Resource Owner Password Credentials grant type)
- **client_id**: `marklogic-oauth`
  - The client ID configured in Keycloak for MarkLogic
- **client_secret**: `4UZyJkjWsGV5JtpsWfgkL1qW5vZ5hhmv`
  - The client secret from your Keycloak client configuration
- **grant_type**: `password`
  - OAuth2 grant type (Resource Owner Password Credentials)
- **scope**: `openid`
  - Requested OAuth2 scope to include OpenID Connect claims

Extracting and Storing the Access Token

To extract the access token and store it in an environment variable for subsequent API calls, use the following curl command as an example:

```
# Extract access token and store in environment variable
ACCESS_TOKEN=$(curl --silent --location -k 'https://oauth.warnesnet.com:8443/realms/progres
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'username=martin' \
--data-urlencode 'password=sterling123' \
--data-urlencode 'client_id=marklogic-oauth' \
--data-urlencode 'client_secret=4UZyJkjWsGV5JtpsWfgkL1qW5vZ5hhmv' \
--data-urlencode 'grant_type=password' \
--data-urlencode 'scope=openid' | jq -r '.access_token')

# Verify the token was extracted
echo "Access Token: $ACCESS_TOKEN"

# Export for use in subsequent commands
export ACCESS_TOKEN
```

Prerequisites for token extraction:

- `jq` command-line JSON processor must be installed
- The curl command must receive a successful response from Keycloak

Using the Token with MarkLogic

Once you have the access token, you can test it against a MarkLogic endpoint:

```
# Test the token against a MarkLogic endpoint
curl --location 'http://oauth.warnesnet.com:8002/manage/LATEST/' \
--header "Authorization: Bearer $ACCESS_TOKEN"
```

Verification Steps

After obtaining and using the token:

1. **Save the Configuration**: Ensure all settings are saved in MarkLogic
2. **Apply to App Server**: Associate the External Security profile with your target App Server
3. **Verify Role Mapping**: Confirm that roles from Keycloak are properly mapped in MarkLogic via the Roles External Name field
4. **Test Token Generation**: Use the curl command above to obtain a valid JWT token
5. **Test MarkLogic Authentication**: Use the token to authenticate against MarkLogic endpoints
6. **Check Logs**: Review MarkLogic error logs for any authentication issues
7. **Validate Role Assignment**: Verify that the user receives appropriate roles and permissions in MarkLogic

Security Note: The Resource Owner Password Credentials grant type should only be used for testing purposes. In production environments, use the Authorization Code flow in your application for better security.

OAuth2 Troubleshooting

When OAuth2 authentication between Keycloak and MarkLogic isn't working as expected, systematic troubleshooting of both systems is essential. This section provides guidance on reviewing logs and enabling debug traces to identify and resolve integration issues.

MarkLogic Troubleshooting

Enabling Debug Trace Events

To increase the level of debug messages in MarkLogic logs, enable the following trace event:

1. **Access MarkLogic Admin Console**: Navigate to `http://your-marklogic-server:8001`
2. **Go to Groups**: Select "Configure" > "Groups" > "Default"
3. **Navigate to Diagnostics**: Click on "Diagnostics" in the left navigation
4. **Add Trace Event**: In the "Trace Events" section, add the following trace events:
    - `JSON Status`
    - `JSON Communications`
    - `JSON Processing`
    - `External Authenticate`

This trace event will provide detailed information about JSON processing and OAuth2 token validation in the MarkLogic logs.

Key MarkLogic Log Messages to Look For

When troubleshooting OAuth2 issues, watch for these types of log entries:

- **Token Validation Errors**: Messages about JWT signature verification failures
- **JWKS URI Issues**: Problems accessing or parsing the Keycloak JWKS endpoint
- **Role Mapping Problems**: Issues with extracting or mapping roles from tokens
- **JSON Processing Issues**: Token parsing or claim extraction failures

Common MarkLogic OAuth2 Error Patterns

```
# JWT signature validation failure
XDMP-OAUTH2: Invalid JWT signature

# JWKS URI access issues
XDMP-OAUTH2: Unable to retrieve JWKS from URI

# Role claim not found
XDMP-OAUTH2: Role attribute not found in token

# Token expiration
XDMP-OAUTH2: JWT token has expired
```

Keycloak Troubleshooting

Keycloak Log Locations

Keycloak logs are typically located in:

Standalone Mode:

```
# Server logs
$KEYCLOAK_HOME/standalone/log/server.log

# Access logs (if enabled)
$KEYCLOAK_HOME/standalone/log/access.log
```

Docker Deployments:

```
# View container logs
docker logs keycloak-container-name

# Follow logs in real-time
docker logs -f keycloak-container-name
```

Enabling Debug Logging in Keycloak

To enable more detailed logging in Keycloak:

1. **Access Admin Console**: Log into Keycloak admin interface
2. **Navigate to Events**: Go to "Realm Settings" > "Events"

3. **Enable Event Logging**:
    - Turn on "Save Events"
    - Turn on "Save Admin Events"
    - Set appropriate expiration times
4. **Configure Log Levels** (via CLI or configuration):

```
# Enable debug logging for OAuth2/OIDC
/subsystem=logging/logger=org.keycloak.protocol.oidc:add(level=DEBUG)
/subsystem=logging/logger=org.keycloak.services.resources.LoginActionsService:add(level=DEE
/subsystem=logging/logger=org.keycloak.authentication:add(level=DEBUG)
```

Key Keycloak Log Messages to Monitor

- **Authentication Events**: User login attempts and results
- **Token Generation**: OAuth2/OIDC token creation and signing
- **Client Authentication**: Client credential validation
- **Mapper Execution**: Protocol mapper processing and claim generation
- **CORS Issues**: Cross-origin request problems
- **Client Configuration Errors**: Invalid redirect URIs or client settings

Common OAuth2 Integration Issues

1. Token Signature Verification Failures

Symptoms:

- MarkLogic rejects valid tokens
- "Invalid JWT signature" errors in MarkLogic logs

Solutions:

- Verify JWKS URI is accessible from MarkLogic server
- Check network connectivity and firewall rules
- Ensure MarkLogic system time is synchronized
- Validate the JWKS URI in External Security profile

2. Role Mapping Problems

Symptoms:

- Users authenticate but receive insufficient permissions
- Role-related errors in MarkLogic logs

Solutions:

- Verify the custom role claim name in both Keycloak mapper and MarkLogic profile
- Check that the Protocol Mapper is correctly configured and enabled
- Ensure the claim appears in the token (decode JWT to verify)
- Validate MarkLogic role mapping configuration

3. Client Configuration Mismatches

Symptoms:

- Authentication flows fail at various stages
- Client authentication errors in Keycloak logs

Solutions:

- Verify client ID matches between Keycloak and MarkLogic
- Check client secret if using confidential client
- Ensure redirect URIs are correctly configured
- Validate OAuth2 flow settings (Standard Flow enabled)

4. Network and Connectivity Issues

Symptoms:

- Intermittent authentication failures
- Timeout errors in logs

Solutions:

- Test network connectivity between MarkLogic and Keycloak
- Check DNS resolution for Keycloak hostnames
- Verify SSL/TLS certificate validity
- Ensure proper firewall rules are in place

Diagnostic Commands

Test JWKS URI Accessibility

```
# Test JWKS URI from MarkLogic server
curl -v "https://oauth.warnesnet.com:8443/realms/progress-marklogic/protocol/openid-connect
```

Decode JWT Token for Analysis

```
# Decode JWT token to inspect claims (requires jwt-cli or online decoder)
echo "$ACCESS_TOKEN" | jwt decode -

#Note: Requires the jwt-cli tools package

# Or using online tool: https://jwt.io/
```

Test Token Validity

```
# Validate token against Keycloak userinfo endpoint
curl -H "Authorization: Bearer $ACCESS_TOKEN" \
"https://oauth.warnesnet.com:8443/realms/progress-marklogic/protocol/openid-connect/userinf
```

Best Practices for Troubleshooting

1. **Enable Logging Early**: Turn on debug logging before testing
2. **Test Components Separately**: Verify Keycloak token generation before testing MarkLogic integration
3. **Use Network Tools**: Employ tools like curl, nslookup, and telnet for connectivity testing
4. **Monitor Both Systems**: Watch logs on both Keycloak and MarkLogic simultaneously
5. **Document Configurations**: Keep detailed records of all configuration settings
6. **Test with Simple Scenarios**: Start with basic authentication before adding complex role mappings

SAML Integration with Keycloak and MarkLogic

This section covers the configuration and integration of Keycloak as a SAML 2.0 identity provider for MarkLogic Application Servers. SAML (Security Assertion Markup Language) provides an alternative authentication method to OAuth2, particularly suited for enterprise environments with existing SAML infrastructure.

SAML Integration Overview

SAML authentication flow between MarkLogic and Keycloak involves:

1. **User Access**: User attempts to access a MarkLogic application
2. **SAML Request**: MarkLogic generates and sends a SAML authentication request to Keycloak
3. **User Authentication**: Keycloak authenticates the user (if not already authenticated)
4. **SAML Response**: Keycloak generates a SAML response with user attributes and roles
5. **Token Validation**: MarkLogic validates the SAML response and extracts user information
6. **Access Granted**: User gains access to MarkLogic resources based on their roles

Creating a Keycloak SAML Client

Step 1: Create SAML Client in Keycloak

1. **Access Keycloak Admin Console**: Navigate to your Keycloak admin interface
2. **Select Realm**: Choose your realm (e.g., "progress-marklogic")
3. **Navigate to Clients**: Click "Clients" in the left sidebar
4. **Create Client**: Click "Create client"
5. **Configure Basic Settings**:
   - **Client type**: Select "SAML"
   - **Client ID**: Enter a unique identifier (e.g., "marklogic-saml")
   - **Name**: Provide a descriptive name
   - **Description**: Optional description for documentation

Step 2: Configure SAML Client Settings

General Settings

- **Client ID**: `marklogic-saml` (or your chosen identifier)
- **Name**: `MarkLogic SAML Client`
- **Description**: `SAML client for MarkLogic Application Server authentication`
- **Always display in UI**: Off (typically)
- **Client authentication**: On (recommended for production)

Important Note: Make a note of the Client ID chosen as this must match excactlythe `SAML Issuer` field in the MarkLogic SAML configuration.

SAML Settings

Basic SAML Configuration:

- **Include AuthnStatement**: On
- **Include OneTimeUse Condition**: Off

- **Sign Documents**: On
- **Sign Assertions**: On
- **Name ID Format**: username or email
- **Home URL**: `http://your-marklogic-server:8002/`
- **Valid Redirect URIs**: `http://your-marklogic-server:8002/`
- **Master SAML Processing URL**: `http://your-marklogic-server:8002/`

The following screenshot shows an example configuration for a Keycloak client:

Clients › Client details

marklogic-saml SAML

Clients are applications and services that can request authentication of a user.

Settings | Keys | Roles | Client scopes | Sessions | Advanced | Events

Jump to section
- General settings
- Access settings
- SAML capabilities
- Signature and Encryption
- Login settings
- Logout settings

General settings

Client ID * : marklogic-saml

Name :

Description :

Always display in UI : Off

Access settings

Root URL :

Home URL : http://oauth.warnesnet.com:9002

Valid redirect URIs : http://oauth.warnesnet.com:9002/*
Add valid redirect URIs

Valid post logout redirect URIs :
Add valid post logout redirect URIs

IDP-Initiated SSO URL name :

IDP Initiated SSO Relay State :

Master SAML Processing URL : http://oauth.warnesnet.com:9002

SAML capabilities

Name ID format : email

Force name ID format : Off

Force POST binding : On

Force artifact binding : Off

Include AuthnStatement : On

Include OneTimeUse Condition : Off

Optimize REDIRECT signing key lookup : Off

Allow ECP flow : Off

Signature and Encryption

Sign documents : On

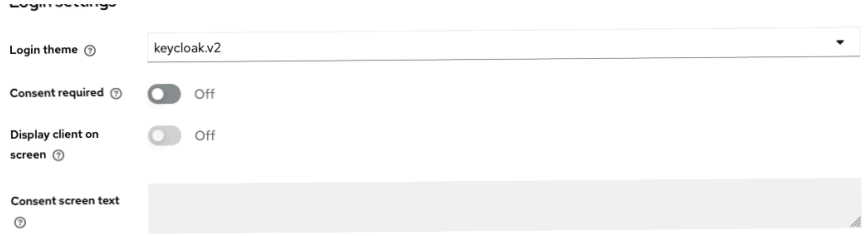
Sign assertions : On

Signature algorithm : RSA_SHA256

SAML signature key name : KEY_ID

Canonicalization method : EXCLUSIVE

Login settings

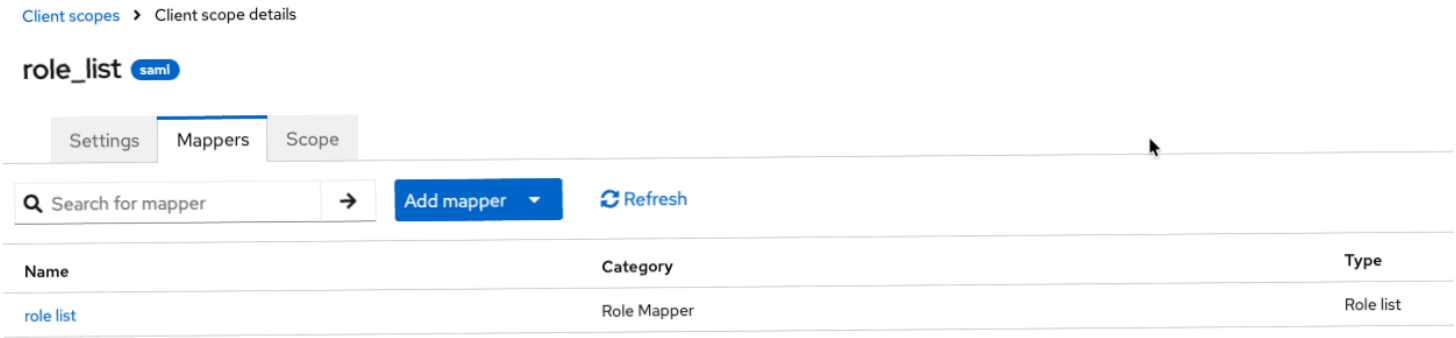


Step 3: Configure SAML Attribute Mappings

Similar to OAuth2, you need to map user attributes and roles for MarkLogic consumption.

Create Role Mapper

1. **Navigate to Client Scopes**: Go to "Client scopes" in Keycloak
2. **Select Client Scope**: Choose the appropriate scope or create a new one
3. **Add Mapper**: Click "Add predefined mapper"
4. **Configure Role Mapper**:
   - **Name**: `marklogic-roles`
   - **Mapper Type**: `Role list`
   - **Role Attribute Name**: `marklogic-roles`
   - **Friendly Name**: `MarkLogic Roles`
   - **SAML Attribute Name**: `marklogic-roles`
   - **SAML Attribute NameFormat**: `Basic`



Creating MarkLogic SAML External Security Profile

Step 1: Obtain Keycloak SAML Metadata

First, obtain the SAML metadata from your Keycloak realm:

SAML Metadata URL:

```
https://oauth.warnesnet.com:8443/realms/progress-marklogic/protocol/saml/descriptor
```

Download or access this metadata as it contains essential configuration information for MarkLogic.

Example IdP Metadata

```
<?xml version="1.0" encoding="UTF-8"?>
<md:EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
    xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#" entityID="https://oauth.warnesnet.com:844
    <md:IDPSSODescriptor WantAuthnRequestsSigned="true" protocolSupportEnumeration="urn:oas
        <md:KeyDescriptor use="signing">
            <ds:KeyInfo>
                <ds:KeyName>9xJihMI2qDH9Aon014odONQLByhF553_8Wu_zcM2StM</ds:KeyName>
                <ds:X509Data>
                    <ds:X509Certificate>MIICszCCAZsCBgGXq8D+0jANBgkqhkiG9w0BAQsFADAdMRswGQY
                </ds:X509Data>
            </ds:KeyInfo>
        </md:KeyDescriptor>
        <md:ArtifactResolutionService Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP" L
        <md:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" Lo
        <md:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect
        <md:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact
        <md:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP" Locatic
        <md:NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:persistent</md:NameIDFor
        <md:NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:transient</md:NameIDForm
        <md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</md:NameIDFo
        <md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress</md:NameIDF
        <md:SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" Lo
        <md:SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect
        <md:SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP" Locatic
        <md:SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact
    </md:IDPSSODescriptor>
</md:EntityDescriptor>
```

Step 2: Configure MarkLogic External Security Profile

Access the MarkLogic Admin interface and create a new External Security profile:

Required Configuration Settings

1. **External Security Name**: `Keycloak-SAML`
2. **Description**: `SAML integration with Keycloak identity provider`
3. **Authentication Protocol**: `saml`
4. **Cache Timeout**: `300` (or as required)

5. **Authorization**: `saml`

SAML-Specific Settings

1. **SAML Settings**:
   - **SAML Entity ID**: `YOUR-SAML-SP`
   - **SAML Destination**: `https://oauth.warnesnet.com:8443/realms/progress-marklogic/protocol/saml` (Matches SingleSignOnService value for HTTP-POST from Keycloak SAML metadata)
   - **SAML Issuer**: `marklogic-saml` (matches Keycloak Client ID)
2. **Certificate Configuration**:
   - **IdP Certificate**: Copy the X.509 certificate from Keycloak SAML metadata

Important Note: The certificate value from the Keycloak SAML metadata is a Base64 encoded string. MarkLogic requires this certificate to be converted to PEM format before it can be used in the External Security configuration.

Base64 to PEM Conversion Process:
The certificate in the metadata appears as a single Base64 string without headers. You must add the appropriate PEM headers (`-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----`) and ensure proper line formatting (64 characters per line) for MarkLogic to accept it.

3. **Attribute Mapping**:
   - **Username Attribute**: `username`
   - **Role Attribute**: `marklogic-roles`
   - **Email Attribute**: `email` (optional)

Step 3: Certificate Management

Obtaining Keycloak Certificate

Extract the X.509 certificate from the SAML metadata:

```
<ds:X509Certificate>
MIICmzCCAYMCBgF/... (certificate content)
</ds:X509Certificate>
```

Converting Base64 Certificate to PEM Format

The certificate content from the SAML metadata is in Base64 format and must be converted to PEM format for MarkLogic. Use the following script to perform this conversion:

```
#!/bin/bash

# Certificate conversion script
# Usage: ./convert-cert.sh <base64-certificate-string>

if [ $# -eq 0 ]; then
    echo "Usage: $0 <base64-certificate-string>"
    echo "Example: $0 'MIICmzCCAYMCBgF/...'"
    exit 1
fi

BASE64_CERT="$1"

# Create PEM formatted certificate
cat << EOF > keycloak-idp-certificate.pem
-----BEGIN CERTIFICATE-----
$(echo "$BASE64_CERT" | fold -w 64)
-----END CERTIFICATE-----
EOF

echo "Certificate converted and saved as 'keycloak-idp-certificate.pem'"
echo "You can now copy the contents of this file into MarkLogic External Security configura

# Optionally display the certificate info
echo ""
echo "Certificate information:"
openssl x509 -in keycloak-idp-certificate.pem -text -noout | head -20
```

Alternative one-liner conversion:

```
# Quick conversion command
echo "-----BEGIN CERTIFICATE-----" > keycloak-cert.pem
echo "YOUR_BASE64_CERT_HERE" | fold -w 64 >> keycloak-cert.pem
echo "-----END CERTIFICATE-----" >> keycloak-cert.pem
```

Manual conversion steps:

1. Copy the Base64 certificate string from the SAML metadata (content between `<ds:X509Certificate>` tags)
2. Create a new file with `.pem` extension
3. Add the header: `-----BEGIN CERTIFICATE-----`
4. Add the Base64 certificate content with line breaks every 64 characters
5. Add the footer: `-----END CERTIFICATE-----`

6. Save the file and copy its contents into MarkLogic External Security configuration

Example Configuration Reference

For reference, the following is an example of a configured SAML External Security profile:

External Security

An external authentication and authorization config.

External Security Name
SAML-Keycloak
External security name (unique)

Description

An object's description.

Authentication
saml
Authentication

Cache Timeout
300
The login cache timeout, in seconds.

Authorization
saml
An authorization scheme.

SAML Server

An SAML server configuration.

SAML Entity ID
MarkLogic-SP
SAML entity id. Required if authorization is SAML.

SAML Destination
https://oauth.warnesnet.com:8443/realms/progress-marklogic/protocol/saml
SAML destination.

SAML Issuer
marklogic-saml
SAML issuer.

SAML Assertion Host

SAML assertion host.

SAML IDP Certificate Authority

```
cmVzcy1tYXJrbG9naWMwHhcNMjUwNjI2MTAxOTAxWhcNMzUwNjI2MTAyMDQxWjAd
MRswGQYDVQQDDBJwcm9ncmVzcy1tYXJrbG9naWMwggEiMA0GCSqGSIb3DQEBAQUA
A4IBDwAwggEKAoIBAQDwjlL7ZifNR5ixXQrRvwD9dDiQFa6Fm2iPOnpxwhvr+k9c
2hRpbCvo0r43I5lNGE5cDab31AHfO8gfz0nJlpzLAfRlUZ1H6ikcZD09ctRE3+TM
QYkyMcPC7rMpoyPr2BoPzqVbMoE+84sUbsDfrbfp51PKh51MvlLDjfWGCyOw1J5+
gLspT9woBQULMPTdS41omT8JtOCqEUmi4+04fQgadano3ruhom8QSRXuUZOTpwpD
TxYRjFli//wywRSRRlJbFFw58c1ec+8JSz9XQqh0yahKdX+O53zylhT0HakAkNUZ
+cSU6czJHXrAtEZNg4dy2AHG9wpl7deC9DlsZmUhAgMBAAEwDQYJKoZIhvcNAQEL
BQADggEBAE+fJ24YCSAw0/C0tdj8meNoeypkODUc9KYUHZdjnjJT+Efcs7bzV3Jz
Xk/ZHTNWRewCJtfle7dOYb50EqjZ+g5ToIgYfh9UgoKH4IrZF7caha/xcm3ffvIq
Iv67llATDolRPtTZMgJkhnTozeUajh2sDVYXHV78Qh4NsgtGVjPPxPdebx4cx4Qs
5v0aiWdNEsp1ranvElFzcuVBbTe6wpp62IGLWawhiVele4Lkll4Qj/CGTPnx2sWi
tFZZ7jX9olJjh4i2OtM0at3Uc+fj3lnOTONL4Q2J6gf/1YfyYMygjcdsX5TSZmF7
Olhf8qsDUeE9l1yAtUkTKDV/FevmFDE=
-----END CERTIFICATE-----
```

The PEM encoded X509 certificate authority for SAML IDP.

SAML SP Certificate

The PEM encoded X509 certificate for SAML SP.

SAML SP Private Key

The PEM encoded private key for SAML SP.

SAML Authn Signature
sha256

SAML Testing and Validation

Testing SAML Authentication Flow

1. **Configure App Server**: Associate the SAML External Security profile with your MarkLogic App Server
2. **Test SSO**: Access a protected MarkLogic resource
3. **Verify Redirect**: Confirm redirection to Keycloak login page
4. **Complete Authentication**: Log in with valid Keycloak credentials
5. **Validate Response**: Verify successful return to MarkLogic with proper roles

SAML Response Validation

Use browser developer tools to inspect the SAML response:

1. **Network Tab**: Monitor network requests during authentication
2. **SAML Response**: Look for POST requests to the ACS endpoint
3. **Response Content**: Decode base64 SAML response to verify attributes

Example SAML assertion structure:

```
<saml:Assertion>
  <saml:AttributeStatement>
    <saml:Attribute Name="Role"
        NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string" >mar
    </saml:Attribute>
  </saml:AttributeStatement>
</saml:Assertion>
```

SAML Troubleshooting

Common SAML Issues

1. Certificate Validation Failures

Symptoms:

- SAML signature validation errors
- Certificate-related error messages in MarkLogic logs

Solutions:

- Verify IdP certificate is correctly copied from Keycloak metadata
- Ensure certificate format is correct (PEM format)
- Check certificate expiration dates
- Validate certificate chain if using intermediate CAs

2. Attribute Mapping Problems

Symptoms:

- Users authenticate but receive no roles
- Missing user attributes in MarkLogic

Solutions:

- Verify attribute mapper configuration in Keycloak
- Check SAML attribute names match MarkLogic configuration
- Inspect SAML response for correct attribute format
- Ensure role mappers are enabled and properly configured

3. URL and Endpoint Mismatches

Symptoms:

- SAML redirect failures
- ACS endpoint not found errors

Solutions:

- Verify ACS URL matches in both Keycloak and MarkLogic configurations
- Check redirect URIs are correctly configured
- Ensure network accessibility between systems

- Validate SLO endpoints for logout functionality

SAML Debug Logging

MarkLogic SAML Trace Events

Enable the following trace events in MarkLogic for SAML debugging:

1. **Navigate to Diagnostics**: MarkLogic Admin > Groups > Default > Diagnostics
2. **Add Trace Events**:
   - `SAML`
   - `SAML Response`
   - `External Authenticate`

Keycloak SAML Logging

Enable SAML-specific logging in Keycloak:

```
# Enable SAML protocol logging
/subsystem=logging/logger=org.keycloak.protocol.saml:add(level=DEBUG)

# Enable SAML processing logging
/subsystem=logging/logger=org.keycloak.saml:add(level=DEBUG)

# Enable XML signature logging
/subsystem=logging/logger=org.keycloak.saml.common:add(level=DEBUG)
```

SAML Diagnostic Tools

Decode SAML Response

Use online tools or command-line utilities to decode SAML responses:

```
# Decode base64 SAML response
echo "BASE64_SAML_RESPONSE" | base64 -d | xmllint --format -
```

Alternativel use a Browser Extension such as `SAML Tracer` to track SAML Requests and Responses

Validate SAML Metadata

Test Keycloak SAML metadata accessibility:

```
# Test metadata endpoint
curl -v "https://oauth.warnesnet.com:8443/realms/progress-marklogic/protocol/saml/descriptc
```

Best Practices for SAML Implementation

1. **Certificate Management**: Implement proper certificate rotation procedures
2. **Secure Communication**: Always use HTTPS for SAML endpoints
3. **Attribute Minimization**: Only include necessary attributes in SAML responses
4. **Regular Testing**: Periodically test both SSO and SLO flows
5. **Documentation**: Maintain detailed documentation of all SAML configurations
6. **Monitoring**: Set up monitoring for SAML authentication success/failure rates