

**RIZIV – INAMI**

**Keycloak**

**ADFS v.2 to Keycloak migration guide**

**for .NET 4.x applications**

**Version : 1.3**



**Document Parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **Document version** | **1.3** |  |  |
| **Creation Date** | **07/03/2022** | **Last update date** | **05/09/2022** |
| **Author**  | **Wilke Jansoone** | **By** | **Wilke Jansoone** |

**Document Flags**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tag** | **Flag** | **Issue** | **Date Set** |
| **Content** |   |   |   |
| **Reminder** |   |   |   |
| **Keyword** |   |

**Document Acceptance**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Signature** | **Title** | **Date** | **Version** |
|   |   |   |   |   |
|   |   |   |   |   |

**Document Distribution**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Signature** | **Date** | **Version** |
|   |   |   |   |
|   |   |   |   |

**Document Glossary**

|  |  |
| --- | --- |
| **Word** | **Definition** |
|   |   |

# Table of content

[Table of content 3](#_Toc1948497160)

[Delegate authentication to Keycloak instead of ADFS 3](#_Toc352825060)

[Introduction 3](#_Toc1889582886)

[Concepts 4](#_Toc1366820212)

[Migration strategy 8](#_Toc1555031080)

[Approach 9](#_Toc345650689)

[DevTokenServer (equivalent to local STS) 10](#_Toc518395889)

[How to migrate your application to Keycloak 13](#_Toc1149865274)

[On your development machines 13](#_Toc1253986482)

# Delegate authentication to Keycloak instead of ADFS

## Introduction

At NIHDI there is an initiative going to replace the on premise ADFS servers by Keycloak. Primary reasons for this migration are:

* ADFS on premise has an uncertain roadmap for the future
* At NIHDI there is a mix of several ADFS versions, there is a desire to consolidate the server farm
* ADFS 2016 (the current version at the NIHDI) is not fully OpenID Connect (OIDC) compliant
* Infrastructure as Code is difficult to achieve with ADFS 2016
* ADFS 2.0 is still used for integration with external IDP's (e.g. eHealth or KSZ-BCSS) and the required functionality cannot be upgraded to ADFS 2016 (because ADFS 2019 is not deployed anymore as an IIS application but as HTTP.sys middleware)

Keycloak has been proposed as an alternative to ADFS. Keycloak is an open-source identity management solution that supports authentication based on SAML and OIDC protocol, it has no out of the box support for the WS-\* security standards (e.g. WS\_Federation WS\_Trust). You can read more about Keycloak at <https://www.keycloak.org/>.

At the NIHDI most, if not all, business web applications based on **.NET 4.x** are implemented as web-based applications. Some web applications are only for internal use, others are also available for an external audience.

**Internal web applications** all require that users are authenticated, the users are known in the riziv.org active directory domain and when they access a business web application from NIHDI computers connected to the internal network (either direct or via VPN) they are already authenticated.

**External web applications** require in most cases that the user is authenticated, but some external applications are also available anonymously. Authentication for external users is delegated to identity providers such as eHealth and KSZ-BCSS.

In all aforementioned cases authentication is currently orchestrated by **ADFS**.

The switch from ADFS to Keycloak has several implications. This document focuses on the authentication and authorization aspects of accessing a front-end web application. The proposed approach tries to avoid code changes required to make the switch from ADFS to Keycloak, unfortunately in some cases code changes cannot be avoided.

**Remark:**
**The purpose of this document is to highlight concepts and to provide a recipe for the implementation and operations teams to be able to make the switch from ADFS to Keycloak, it is however not the intention to go into too much low-level detail about ADFS, OWIN or OIDC.**

**General information about Keycloak as a product and how it is implemented and integrated at the NIHDI will be provided in a separate document (location still to be decided).**

## Concepts

Web applications at NIHDI that rely on ADFS to provide authentication services follow a similar pattern. The configuration is performed through settings in the configuration file of the web application (web.config).

The image below shows the configuration of a NIHDI business web application, the highlighted sections contain the basic configuration of the authentication flow. The logic that redirects the user to the ADFS server is the **WSFederationAuthenticationModule**.



ADFS authentication relies on the traditional HTTP application pipeline based on HTTPModules and HTTPHandlers. Keycloak authentication on the other hand uses a different concept of integration logic, it is based on OWIN middleware.

To delegate the authentication to Keycloak we need to first ensure the WSFederationAuthenticationModule is inactivated (as well as the SessionAuthenticationModule), next we need to provide the required OWIN middleware.

The OWIN middleware executes in the HTTP application pipeline by providing an OWIN startup class containing the required logic and registering this OWIN startup class with the hosting environment (console, IIS express, IIS, …).

There are several ways to register the bootstrap class:

1. **Naming Convention**: OWIN looks for a class named Startup in namespace matching the assembly's name or the global namespace.
2. **OwinStartup Attribute**: The following attribute will set the OWIN startup class to the TestStartup class in the StartupDemo namespace.

C#

[assembly: OwinStartup(typeof(Nihdi.Identity.Owin.Startup))]

VB.NET

<Assembly: OwinStartup(GetType(Nihdi.Identity.Owin.Startup))>

The OwinStartup attribute overrides the naming convention. You can also specify a friendly name with this attribute, however, using a friendly name requires you to also use the appSetting element in the configuration file.

1. **The appSetting element in the Configuration file**: The following appSetting element overrides the OwinStartup attribute and naming convention. You can have multiple startup classes (each using an OwinStartup attribute) and configure which startup class will be loaded in a configuration file using markup similar to the following:

<appSettings>

 <add key="owin:appStartup" value="Nihdi.Identity.Owin.Startup" />

</appSettings>

The startup class contains the logic to redirect the user to Keycloak and use the authentication services (**for the latest version of the code check out the code in Azure Devops**):

public static IAppBuilder UseKeycloakImplicitFlow(this IAppBuilder app)

{

 if (app == null)

 {

 throw new ArgumentNullException("app");

 }

 // Currently at the NIHDI SSL is terminated at the loadbalancer.

 // However redirect should use TLS, following statement enforce

 // https.

 if (ConfigurationHelper.RequireHttps)

 {

 app.Use((ctx, next) =>

 {

 ctx.Request.Scheme = "https";

 return next();

 });

 }

 app.SetDefaultSignInAsAuthenticationType(CookieAuthenticationDefaults.AuthenticationType);

 // Enable the application to use a cookie to store information for the signed in user

 app.UseCookieAuthentication(new CookieAuthenticationOptions());

 var openIdConnectAuthenticationOptions = new OpenIdConnectAuthenticationOptions

 {

 AuthenticationMode = Microsoft.Owin.Security.AuthenticationMode.Active,

 Authority = ConfigurationHelper.Authority,

 ClientId = ConfigurationHelper.ClientId,

 RedirectUri = ConfigurationHelper.RedirectUri,

 ResponseType = OpenIdConnectResponseType.CodeIdToken, // Implicit flow needs to be activated at the level of the KeyCloak client.

 Scope = OpenIdConnectScope.OpenIdProfile,

 UseTokenLifetime = true,

 RequireHttpsMetadata = ConfigurationHelper.RequireHttps,

 TokenValidationParameters = new TokenValidationParameters

 {

 // By default the OWIN middleware adds

 // - The authority to the list of valid issuers

 // - The clientId as a valid audience

 // to the default TokenValidationParameters instance.

 // The values added in this instance are merged with these coming from the metadata.

 // By default, the default OWIN token validators verify these values since the

 // ValidateIssuer and ValidateAudience is set to true.

 SaveSigninToken = true,

 // It is possible for tokens to contain the public key needed to

 // check the signature.

 // For example, X509Data can be hydrated into an X509Certificate, which can be used

 // to validate the signature. In these cases it is important to validate the

 // SigningKey

 // that was used to validate the signature.

 // Ensure the validation of the SecurityKey that signed the securityToken.

 ValidateIssuerSigningKey = true

 },

 Notifications = new OpenIdConnectAuthenticationNotifications

 {

 RedirectToIdentityProvider = (context) =>

 {

 // The OWIN middleware by default does not distinguish following scenarios and

 // always returns a 401 response:

 // 1 - The user is not authenticated => should return 401

 // 2 - The user is authenticated but does not have the required

 // role => should return 403

 // The following statements enforce the return of a 403 response id the user is

 // authenticated but not authorized.

 if (context.OwinContext.Response.StatusCode == 401 &&

 context.OwinContext.Authentication.User.Identity.IsAuthenticated)

 {

 context.OwinContext.Response.StatusCode = 403;

 context.HandleResponse();

 }

 return Task.FromResult(0);

 },

 },

 CookieManager = new SystemWebCookieManager()

 };

 if (!ConfigurationHelper.RequireHttps)

 {

 // System.Web response cookie integration issues

 // https://github.com/aspnet/AspNetKatana/wiki/System.Web-response-cookie-integration-issues

 openIdConnectAuthenticationOptions.CookieManager = new OwinCookieManager();

 }

 app.UseOpenIdConnectAuthentication(openIdConnectAuthenticationOptions)

 .UseStageMarker(PipelineStage.Authenticate);

 return app;

}

OWIN configuration specifics are registered in the appSettings sections of the web application configuration file.

The middleware is invoked when an unauthenticated request is received by the endpoint. ASPNET provides different approaches to mark a resource to be available only for authenticated users:

* Use aspnet URL authorization
* Use the Authorize (or derived class thereof) attribute on controllers
* Use an authorization filter
* Use an authorization policy

The logic mentioned above has been implemented in a NuGet package: **Riziv.Common.Identity.Owin.**

The package will make following modifications:

* Adds following assemblies:

|  |  |
| --- | --- |
| **Assembly name** | **Version** |
| Microsoft.IdentityModel.JsonWebTokens  | 5.3.0 |
| Microsoft.IdentityModel.Logging | 5.3.0 |
| Microsoft.IdentityModel.Protocols | 5.3.0 |
| Microsoft.IdentityModel.Protocols.OpenIdConnect | 5.3.0 |
| Microsoft.IdentityModel.Tokens | 5.3.0 |
| Microsoft.Owin | 4.1.0 |
| Microsoft.Owin.Host.SystemWeb | 4.1.0 |
| Microsoft.Owin.Security | 4.1.0 |
| Microsoft.Owin.Security.Cookies | 4.1.0 |
| Microsoft.Owin.Security.OpenIdConnect | 4.1.0 |
| Newtonsoft.Json | 10.0.1 |
| Owin | 1.0 |
| System.IdentityModel.Tokens.Jwt | 5.3.0 |

* Adds following appsettings to the web.config:
	+ <add key="owin:appStartup" value="[Owin startup class]" />
	+ <add key="Authority" value="[Authority]" />
	+ <add key="ClientId value="[ClientId]" /
	+ <add key="RedirectUri" value="[RedirectUri]" />
	+ <add key="RequireHttps" value="[RequireHttps]" />

## Migration strategy

### Approach

There are about 100 .NET 4.x web applications that need to be included in the switch trajectory from ADFS to Keycloak. Some of these web applications already use the OWIN middleware via the bootstrap mechanism (in most cases via a solution wide assembly attribute).

Initially we imagined two different migration tracts, one for web applications that do not yet use OWIN and those that already do. The applications that already used OWIN would be migrated by the implementation team, the others by the operations team.

We performed a high-level risk assessment of migrating the web applications via the operations team. This risk assessment matrix documents the risks the migration might impose on an application in validation or production. The risk assessment results are based on the premise that the state of the applications is the same in TST, VAL and PRD environments.

|  |  |  |  |
| --- | --- | --- | --- |
| **Event description** | **Inherent risk (\*) (\*\*)** | **Risk response** | **Residual risk (\*)** |
| TFS is not correctly updated. Consecutive release provokes error. | 3/2 | Releases are always first executed in the TST environment. Errors will be catched and corrective actions will be taken. | High |
| Migration patch is not correctly applied. Application may be unavailable. | 1/3 | Several rollback actions are available to bring the application to its original state (undo actions or re-release). | Medium |
| Application is not sufficiently tested. Users might not be able to use all the application's functionality. | 2/2 | Patch is first applied in the TST environment. Testing is executed in this environment. | Medium |
| Keycloak is not correctly configured. Users might not be able to use all the application's functionality. | 2/2 | By testing the application sufficiently in the TST environment we can reduce the likelihood that the issue presents itself in VAL or PRD. | Medium |

(\*) Risk is expressed as low, medium or high, either as numeric value 1, 2 or 3 or as color green, orange or red.

(\*\*) Inherent risk is expressed using two values, the first refers to the likelihood of the event, the second one to the impact or severity of the event.

After analysis and evaluation, we decided it was better to let all the applications be migrated by the implementation team.

### DevTokenServer (equivalent to local STS)

Currently applications relying on Active Directory Federation Service authentication can install a NuGet package that includes a local security token service with support for WS-Fed, named Local STS.

A similar tool has been developed and packaged for applications relying on Keycloak. This tool has been called DevTokenServer, and allows any developer to run a local OpenID Connect server to test an application.

The NuGet package, named Nihdi.IdentityModel.OpenIdConnect.DevTokenServer, is available on the RIZIV feed. The goal was to deliver something similar to the local STS.

This tool has been presented in a video. More about this in the WiKi post:

[Presentation of the Nihdi.IdentityModel.OpenIdConnect.DevTokenServer NuGet package in Wiki](https://teams.microsoft.com/l/entity/com.microsoft.teamspace.tab.wiki/tab%3A%3Af7a2bf1c-0d9b-4938-a68d-2c12bb985e1a?context=%7B%22subEntityId%22%3A%22%7B%5C%22pageId%5C%22%3A6%2C%5C%22sectionId%5C%22%3A12%2C%5C%22origin%5C%22%3A2%7D%22%2C%22channelId%22%3A%2219%3Ac454cd15ca0e49f8a04303736ca1463d%40thread.tacv2%22%7D&tenantId=66c008a4-b565-49a9-93c9-c1e64cad2e11)

There is also a short movie (animated GIF) explaining how to use the tool:

[Local Dev Token Server.gif](https://gcloudbelgium.sharepoint.com/%3Ai%3A/r/teams/GRP-NIHDI-PIdentityGov.andAdmin/Shared%20Documents/Keycloak/Recordings/Local%20Dev%20Token%20Server.gif?csf=1&web=1&e=VzaejD)

If you need to test your DevTokenServer configuration, there is also a tool available to do this, named OIDC test client:

[OIDCTestClient v1.0.8.zip](https://gcloudbelgium.sharepoint.com/%3Au%3A/r/teams/GRP-NIHDI-PIdentityGov.andAdmin/Shared%20Documents/Keycloak/Tools/OIDCTestClient%20v1.0.8.zip?csf=1&web=1&e=ZwBlaX)

**Important**

**The dev token server uses a local development certificate when exposing the endpoints over TLS. If this certificate is not present, the following exception will be thrown when launching the dev token server:**



In order to install a local development certificate, you need to execute following command on your development virtual machine:

dotnet dev-certs https --trust

This command will install a development certificate in the CurrentUser/my store, this can be checked by executing the command:

dotnet dev-certs https --check --verbose



**If the web application that calls the dev token server is hosted in IIS and the application pool is running with the apppool identity, you must also copy the developer certificate to the  Local Computer\Trusted Root Certification Authorities. If this is not the case the following exception will be thrown:**

* **Could not establish trust relationship for the SSL/TLS secure channel**
	+ **The remote certificate is invalid according to the validation procedure**

## How to migrate your application to Keycloak

**Important: Before an application can be migrated to using Keycloak, the configuration of the required attribute set (claims set) must be done by the operations team (for .NET 6 applications the configuration of Keycloak is currently integrated within the release pipeline).**

**When an implementation team is invited to start the migration from ADFS to Keycloak this configuration is already done. A successful functional test ensures that the correct set of attributes (claims) have been configured in the Keycloak administration console.**

#### On your development machines

* Make sure the framework version of the application is 4.6.1 or higher.
* Add NuGet package **Riziv.Common.Identity.Owin** (2.4.293.1) from Riziv feed to the web application that requires authentication features previously provided by ADFS.
* Update or add Keycloak startup code.
	+ In case existing Owin startup code exists:
		- Add statement to configure Keycloak OIDC middleware

using Riziv.Common.Identity.Owin;

public class Startup

{

public void Configuration(IAppBuilder app)

{

**app.UseKeycloakImplicitFlow();** // => Keycloak OIDC middleware statement

// Existing Owin startup logic

}

}

* + In case existing Owin startup code does not exist
		- Add new public class Startup.cs
		- Add method Configuration:

using Riziv.Common.Identity.Owin;

public class Startup

{

public void Configuration(IAppBuilder app)

{

**app.UseKeycloakImplicitFlow();** // => Keycloak OIDC middleware statement

}

}

 **Remark: The Katana OWIN middleware does not provide extensibility support for invoking a custom ClaimsAuthenticationManager by default (in contrast with the ASP.NET identity model). This means that if the application authorization logic relies on claims augmentation using a custom ClaimsAuthenticationManager you need to indicate this explicitly when configuring the middleware like so:**

 using Riziv.Common.Identity.Owin;

public class Startup

{

public void Configuration(IAppBuilder app)

{

**app.UseKeycloakImplicitFlow(new [your custom ClaimsAuthenticationManager type]());** // => Keycloak OIDC middleware statement

}

}

* Update web.config
	+ Disable WS-Fed http modules (it is recommended to just comment out the WSFederationAuthentiocationModule and the SessionAuthenticationModule configuration entries, reverting back to using WSFed is easier in this way).
		- **WSFederationAuthenticationModule**
		- **SessionAuthenticationModule**
	+ Update Keycloak OIDC configuration appsetting parameters:

		- owin:appStartup: If the class that contains the OIDC bootstrap code is not called **Startup**, you can use this parameter to give the name of that class. If this class is called **Startup**, you can remove this parameter**.**
		- Authority: the base URL of the authorization server:

|  |  |
| --- | --- |
| LOCAL (DevTokenServer) | https://localhost:5001/auth/realms/dev/ |
| DEV | https://rias-t.riziv-inami.fgov.be/auth/realms/dev/ |
| LEGACY (TST) | https://rias-t.riziv-inami.fgov.be/auth/realms/legacy/ |
| LEGACY (VAL) | https://rias-v.riziv-inami.fgov.be/auth/realms/legacy/ |
| LEGACY (PRD) | https://rias.riziv-inami.fgov.be/auth/realms/legacy/ |
| TST | https://rias-t.riziv-inami.fgov.be/auth/realms/tst/ |
| VAL | https://rias-v.riziv-inami.fgov.be/auth/realms/val/ |
| PRD | https://rias.riziv-inami.fgov.be/auth/realms/prd/ |

**The LOCAL authority is applied when using the local DevTokenServer. The default port is 5001, if you use a different port, the value for the authority needs to be adapted.**
**DEV and TST realms both are present in the rias-t environment, the difference is that the DEV realm is integrated with RIZIV.TSTDEV, and the TST realm is integrated with RIZIV.ORG. This means that if you want to authenticate with RIZIV.ORG accounts (for example tests with business users) you need to refer to the TST realm.**

* + - ClientId: public identifier for applications, ideally defined based on the [naming convention](https://gcloudbelgium.sharepoint.com/%3Aw%3A/r/teams/GRP-NIHDI-PIdentityGov.andAdmin/Shared%20Documents/Keycloak/General%20documentation/Keycloak%20ClientId%20naming%20convention.docx?d=w9afee03057ba4215bfea58a59ecc214f&csf=1&web=1&e=MDFppM). The value of this field is found in [this file](https://gcloudbelgium.sharepoint.com/%3Ax%3A/r/teams/GRP-NIHDI-PIdentityGov.andAdmin/Shared%20Documents/Keycloak/Support%20documents/Web%20applications%20overview.xlsx?d=w1349ca0327e8429baadfb711b70ff588&csf=1&web=1&e=7ylbaJ).
		***Important note: the applications currently using ADFS and migrated to Keycloak will be configured on Keycloak in a dedicated realm where the naming convention IS NOT followed. To ease the migration, the ClientId has been retained from the ADFS configuration.***
		- RedirectUri: the URI the authorization server will redirect the user back to after a successful authentication. If the attribute RequireHttps is true, then the URL must use the https scheme. For new applications using .NET 6, the RedirectUri ONLY allows lowercase characters (for legacy application we still allow a mix of upper and lowercase).
		This new URI will starts with:
			* for internal applications:
			[https://webappsint-t.riziv-inami.fgov.be/<applicationpath>/](https://eur03.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwebappsint-t.riziv-inami.fgov.be%2F%253capplicationpath%253e%2F&data=05%7C01%7CEmmanuel.Hankenne%40riziv-inami.fgov.be%7C6588fae6d2354e09714608da5f31d497%7C66c008a4b56549a993c9c1e64cad2e11%7C0%7C0%7C637926964872468556%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=A2dAFPYjfozaKBdhfsG4LrVJFIVYY7lj%2F5YWVH2nOTw%3D&reserved=0) in TST
			[https://webappsint-v.riziv-inami.fgov.be/<applicationpath>/](https://eur03.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwebappsint-v.riziv-inami.fgov.be%2F%253capplicationpath%253e%2F&data=05%7C01%7CEmmanuel.Hankenne%40riziv-inami.fgov.be%7C6588fae6d2354e09714608da5f31d497%7C66c008a4b56549a993c9c1e64cad2e11%7C0%7C0%7C637926964872468556%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=sKnMV9Y54drNoxcYep6ekR1Cv067G%2F0RLLqzuv8vjMo%3D&reserved=0) in VAL
			[https://webappsint.riziv-inami.fgov.be/<applicationpath>/](https://eur03.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwebappsint.riziv-inami.fgov.be%2F%253capplicationpath%253e%2F&data=05%7C01%7CEmmanuel.Hankenne%40riziv-inami.fgov.be%7C6588fae6d2354e09714608da5f31d497%7C66c008a4b56549a993c9c1e64cad2e11%7C0%7C0%7C637926964872468556%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=YoUicXMmWZLDNj5GfKicVgzlPYJiGoNtKbZjQHdQ4SQ%3D&reserved=0) in PRD
			* for external “public” anomymous applications
			[https://webappsa-t.riziv-inami.fgov.be/<applicationpath>/](https://eur03.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwebappsa-t.riziv-inami.fgov.be%2F%253capplicationpath%253e%2F&data=05%7C01%7CEmmanuel.Hankenne%40riziv-inami.fgov.be%7C6588fae6d2354e09714608da5f31d497%7C66c008a4b56549a993c9c1e64cad2e11%7C0%7C0%7C637926964872468556%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=JGU5rlgZ3weNmsp2P%2BPmiUSGXULVKZvYC1du19u4Rr0%3D&reserved=0) in TST
			[https://webappsa-v.riziv-inami.fgov.be/<applicationpath>/](https://webappsa-v.riziv-inami.fgov.be/%3Capplicationpath%3E/) in VAL
			[https://webappsa.riziv-inami.fgov.be/<applicationpath>/](https://eur03.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwebappsa.riziv-inami.fgov.be%2F%253capplicationpath%253e%2F&data=05%7C01%7CEmmanuel.Hankenne%40riziv-inami.fgov.be%7C6588fae6d2354e09714608da5f31d497%7C66c008a4b56549a993c9c1e64cad2e11%7C0%7C0%7C637926964872468556%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=KwbBEbLFqqqwB5C8t4FPbWMpx%2FsL2ACZ3oWmukcpWjA%3D&reserved=0) in PRD
			* for external “authorized” applications
			[https://webapps-t.riziv-inami.fgov.be/<applicationpath>/](https://eur03.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwebapps-t.riziv-inami.fgov.be%2F%253capplicationpath%253e%2F&data=05%7C01%7CEmmanuel.Hankenne%40riziv-inami.fgov.be%7C6588fae6d2354e09714608da5f31d497%7C66c008a4b56549a993c9c1e64cad2e11%7C0%7C0%7C637926964872468556%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=0d3Gz767vtp1P6zZlYUyqkKUBPDAhQLZeReJ%2FEGkm5c%3D&reserved=0) in TST
			[https://webapps-v.riziv-inami.fgov.be/<applicationpath>/](https://eur03.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwebapps-v.riziv-inami.fgov.be%2F%253capplicationpath%253e%2F&data=05%7C01%7CEmmanuel.Hankenne%40riziv-inami.fgov.be%7C6588fae6d2354e09714608da5f31d497%7C66c008a4b56549a993c9c1e64cad2e11%7C0%7C0%7C637926964872468556%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=1rG1lkbVXIMOpP8k9lWZpUHG2ZPE1a2Zcyua83%2BIwF4%3D&reserved=0) in VAL
			[https://webapps.riziv-inami.fgov.be/<applicationpath>/](https://webapps.riziv-inami.fgov.be/%3Capplicationpath%3E/) in PRD

***Note:***
***On top of the ADFS -> Keycloak migration, we will also migrate applications from UAG/TMG to F5 Big-IP. Please take contact with*** ***the Nework & Telecom team*** ***before you release for the first time the application to inform them about your release, so that the needed action can be performed on their side. Please mention the correct URL for each environment in your mail to*** ***the Nework & Telecom team******.***

***Important: The RedirectUri parameter is case sensitive, it must be identical to what has been registered in the Keycloak admin console. To avoid issues, we decided on the convention to write all RedirectUri in lowercase only (again for legacy application derogation is allowed).***

* + - Scopes: A string containing the list of requested scopes, the default value for this parameter is **openid**. Normally the scope **profile** is also present, the purpose of this scope is to pass generic user information as JWT tokens. For legacy applications however, we need to pass the information using xml namespace formatting. Therefore we removed the default **profile** scope from the default configuration and replaced it by **legacy-profile** scope. This **legacy- profile** scope passes the same information but uses xml namespace formatting for backwards compatibility with WS-Fed clients, it does not need to be explicitly requested as it is added by default through Keycloak configuration.
		- RequireHttps: application (client) endpoint uses TLS (this attribute does not apply to the communication with the authorization server, in most cases the authorization server require HTTPS anyway).

***Note: Because the*** [***Implicit flow***](https://www.keycloak.org/docs/latest/securing_apps/index.html#_javascript_implicit_flow) ***has been enabled on the Keycloak server for all legacy applications migrating from ADFS to Keycloak, there is NO secret required.***

* Run and test these modifications on your local development machine (use the DevTokenServer).
* If local tests are successful make the necessary modifications for release in TST (use the **legacy** realm).
	+ Update the SetParameters file with entry values for the specific environment (TTS/VAL/PRD):
		- <setParameter name="Authority" value="[Authority value]" />
		- <setParameter name="ClientId" value="[ClientId value]" />
		- <setParameter name="RedirectUri " value=" RedirectUri value"/>
		- <setParameter name="Scopes" value="[Scopes value]" />
		- <setParameter name="RequireHttps" value="[RequireHttps value]" />

**Important: The RedirectUri parameter is case sensitive, it must be identical to what has been registered in the Keycloak admin console.**

* Release in TST
* Perform required technical (authentication) and functional (authorization) tests.
	+ Technical test verifies whether user can authenticate using Keycloak
	+ Functional tests verify the authorization part, can the users still access the required application functions with the set of claims present in the ID token received from Keycloak.
* If tests are successful and **you do not need to use ADFS anymore for authentication**:
	+ WS-Fed related configuration sections can be removed from web.config:
		- **modules/WSFederationAuthenticationModule**
		- **modules/SessionAuthenticationModule**
		- **system.identityModel**
		- **system.identityModel.services**
		- WS-Fed related settings can be removed from the SetParameters file.
* Release in LEGACY (VAL) and LEGACY (PRD) environments

Congratulations, once you have completed those steps, you have successfully updated the application to use Keycloak instead of ADFS 2.
Thanks for your collaboration.

\* \* \* \* \*

\* \* \*

\*