**AWS Cognito**

Ref: <https://coderjony.com/blogs/authenticating-users-in-aspnet-core-mvc-using-amazon-cognito>

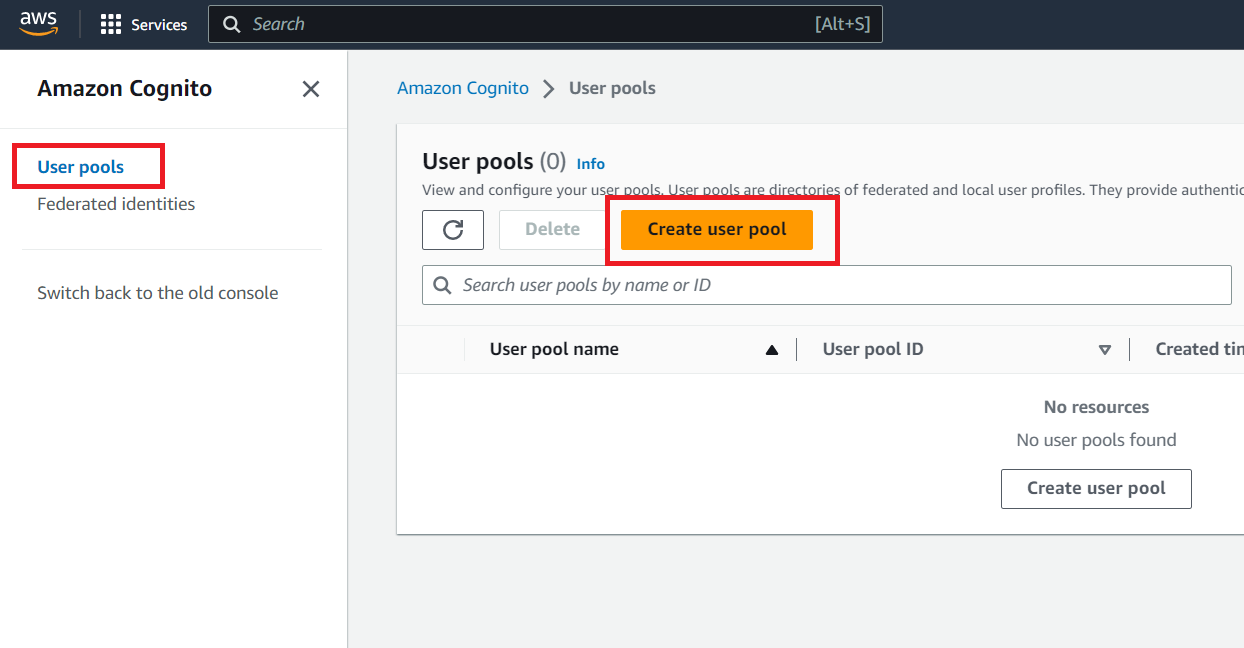
Amazon Cognito as an Identity Provider for your ASP.NET Core MVC application. This means that you will store your users in Amazon Cognito and configure your web application to authenticate users through a Cognito Hosted UI.

**Pre-requisites**

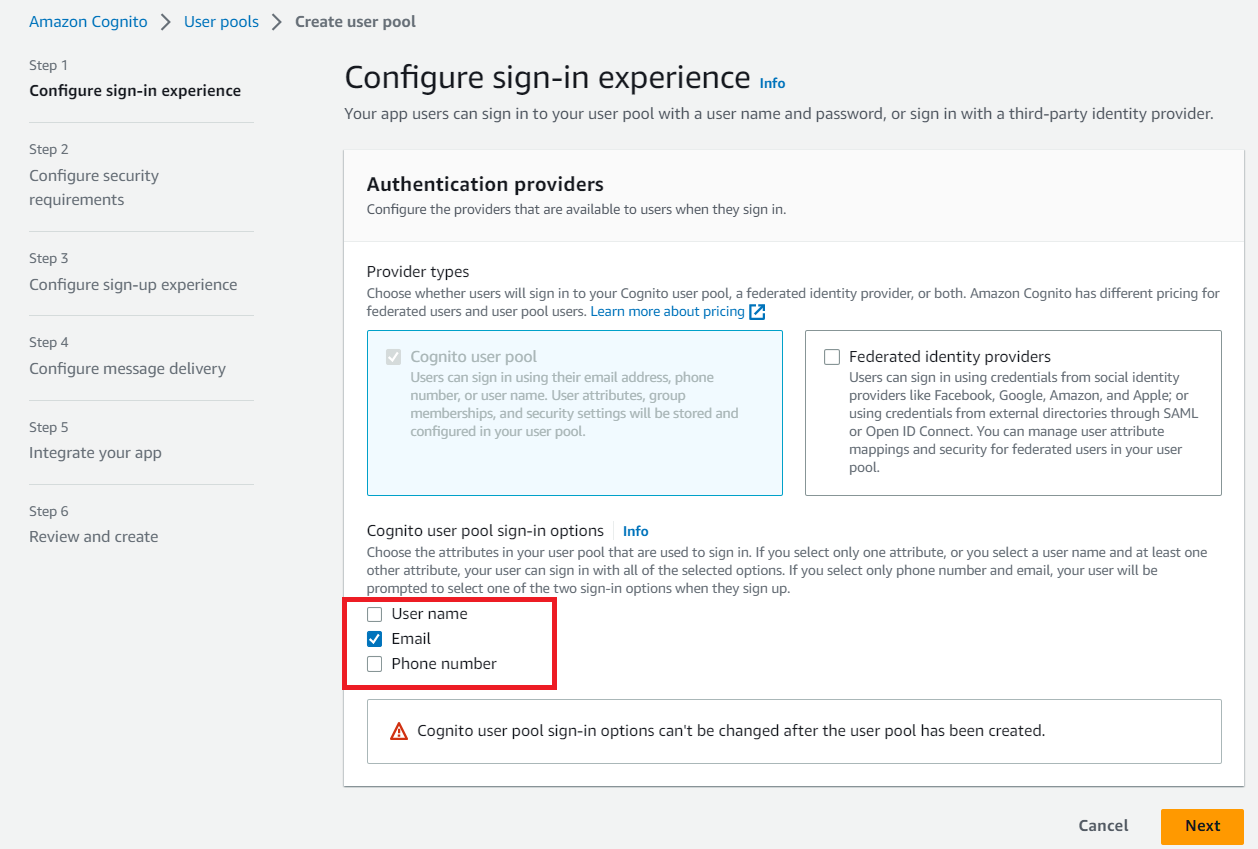
* An active AWS account
* Visual Studio
* ASP.NET Core 6 or higher

Follow the instructions below to create a user pool in Amazon Cognito.

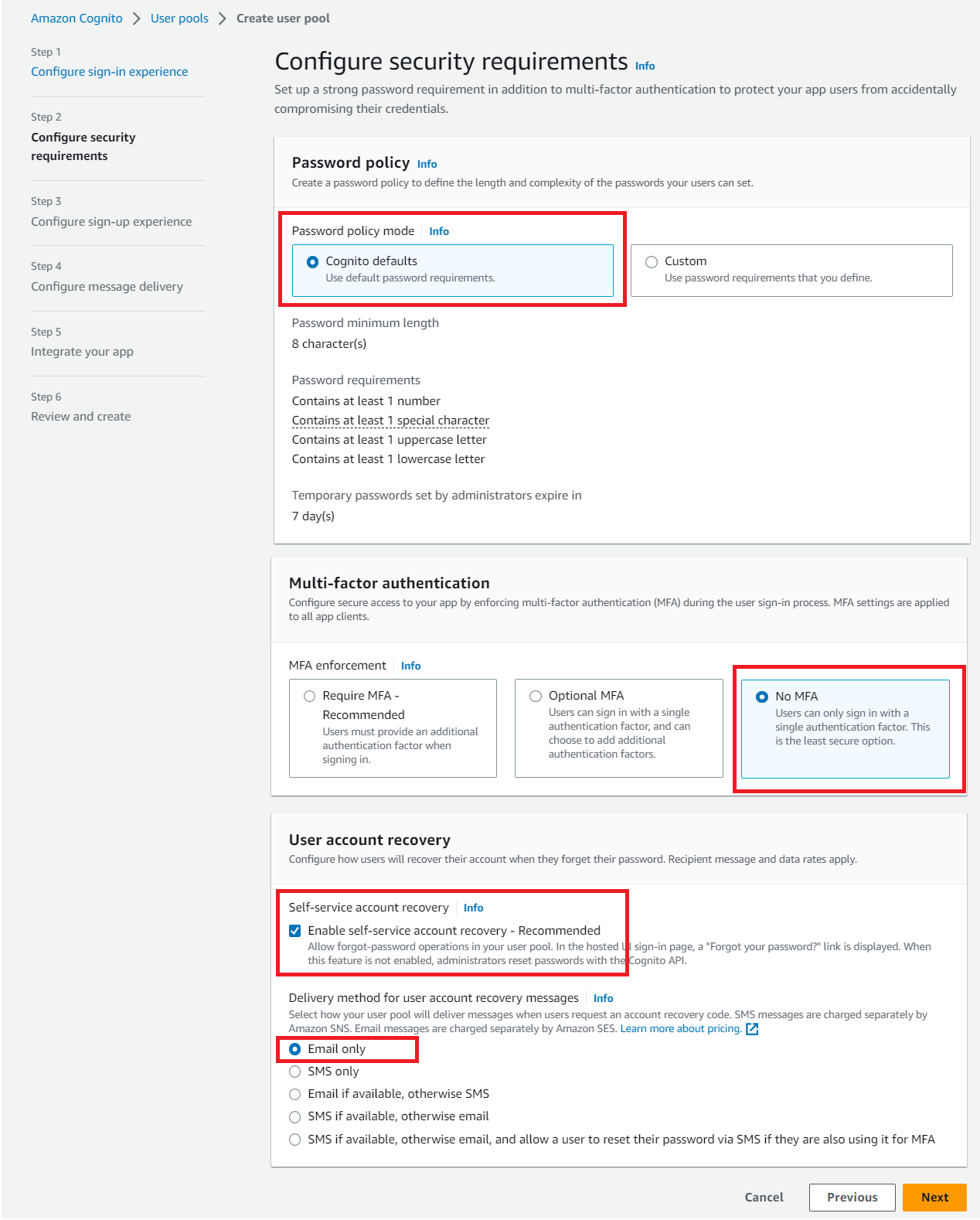
1. Go to the [Amazon Cognito console](https://console.aws.amazon.com/cognito/home). If prompted, enter your AWS credentials.
2. Choose **User Pools** from the left panel, and then click on **Create use pool** button.



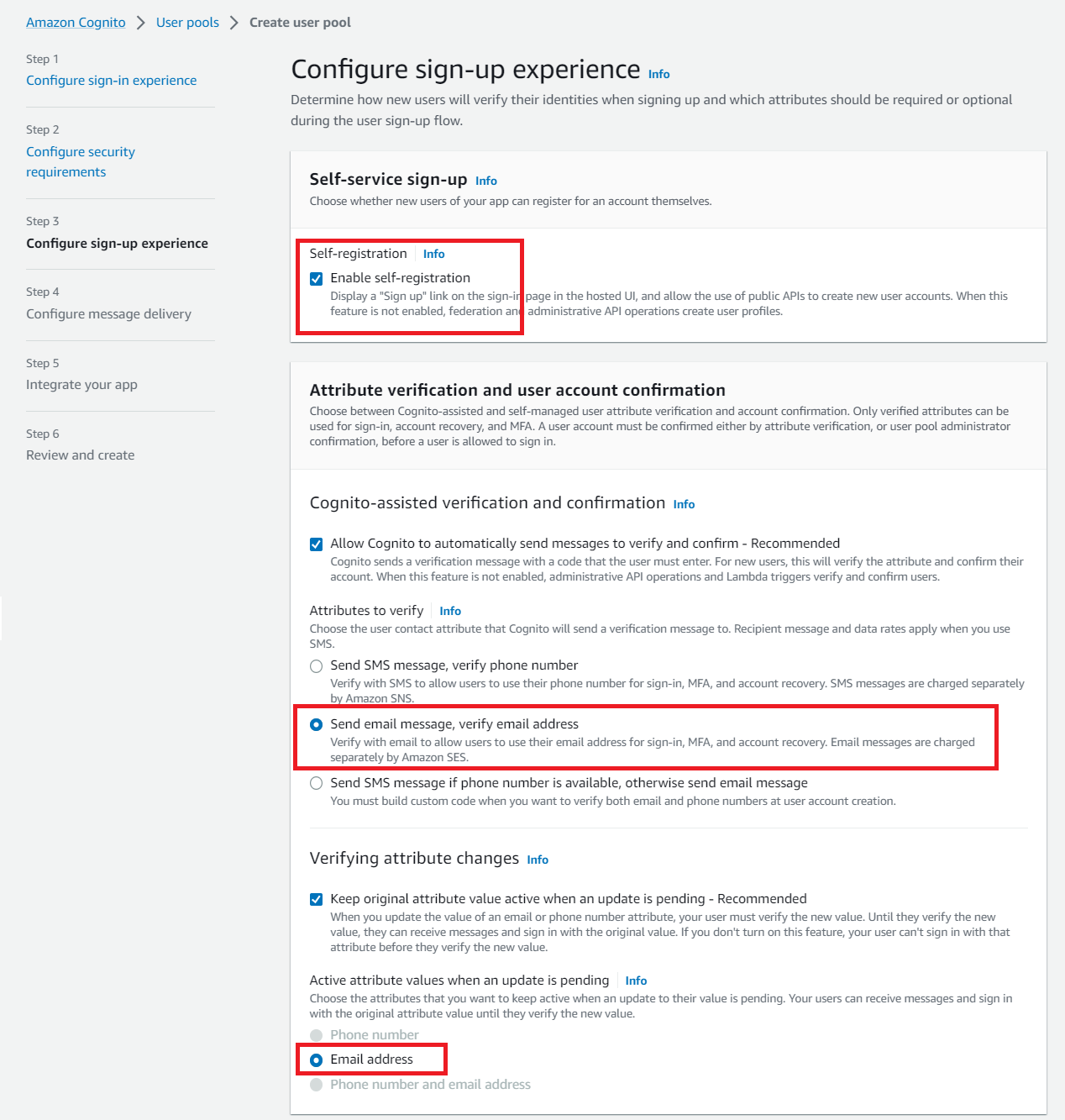
1. In **Configure sign-in experience**, choose **Email** for **Cognito user pool sign-in options**. Although you can choose **User name** and **Phone number** options as well, but just to keep this blog post simple, I am choosing only Email as a sign-in option.



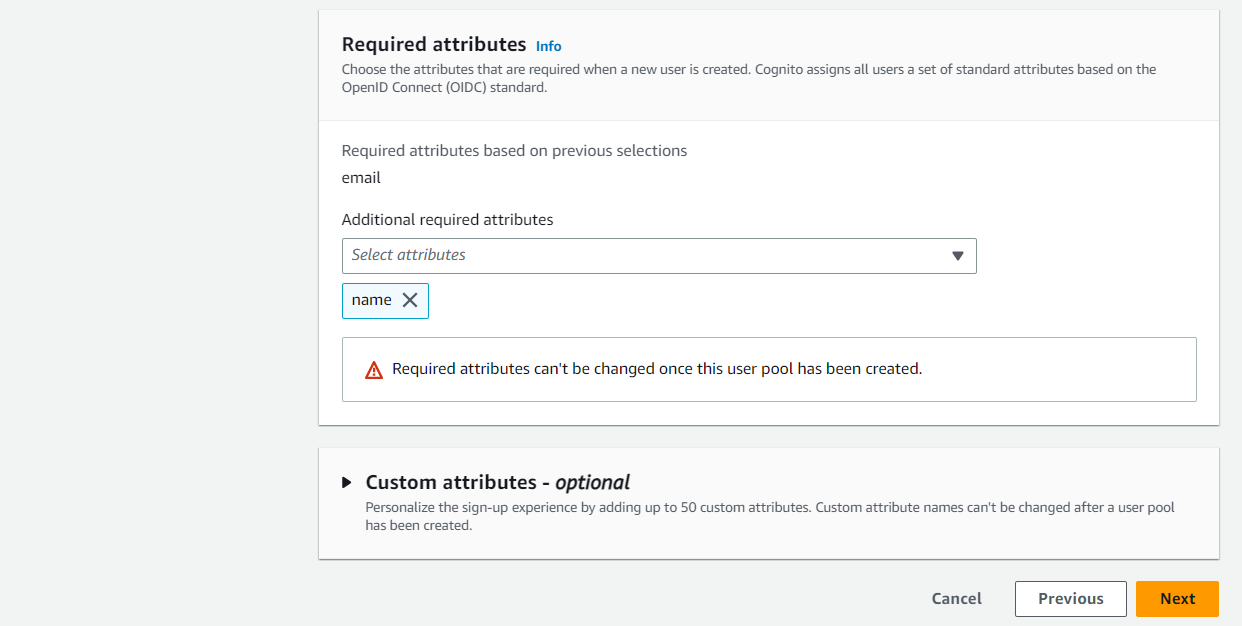
1. In **Configure security requirements**, choose **Cognito defaults** for **Password policy**, **No MFA** for **Multi-factor authentication**, and default values (as shown below) for **User account recovery** sections.



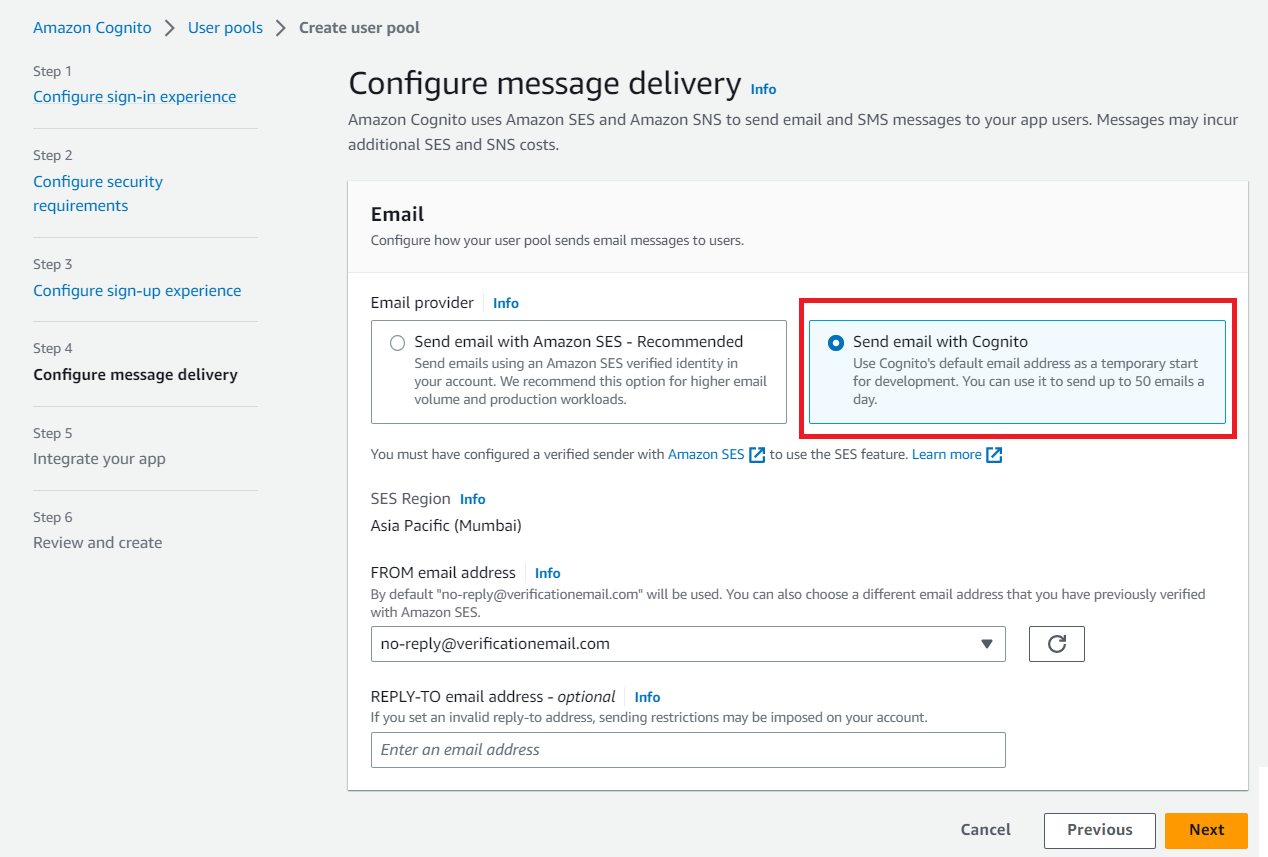
1. In **Configure sign-up experience**, go with the default selection for **Self-service sign-up** and **Attribute verification and user account confirmation** sections.



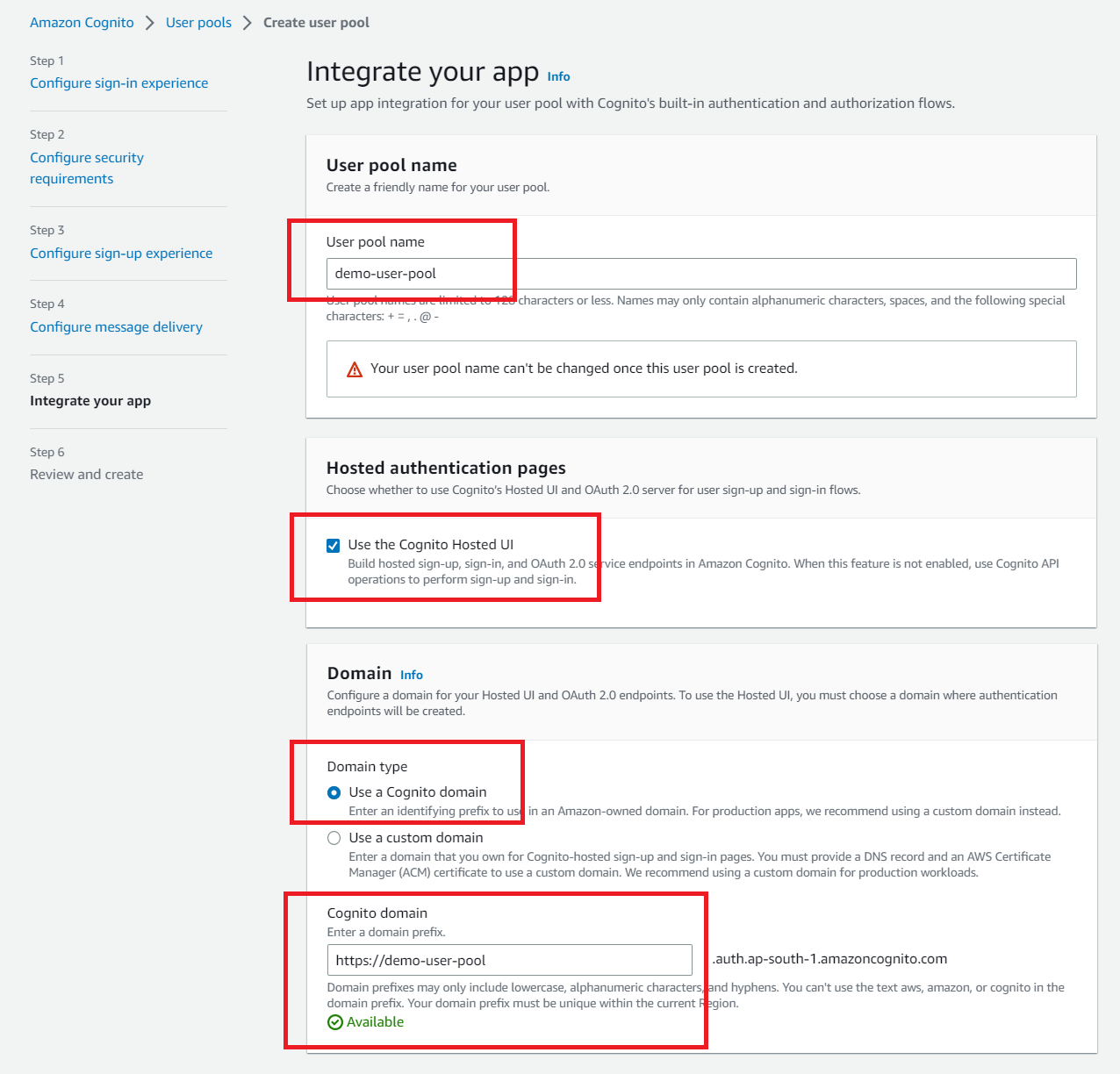
For **Required attributes** section, choose only **name** as a required field.



1. In **Configure message delivery**, choose how the mails should be sent for events such as Account verification, Forgot password, and so on. Choose **Send email with Cognito** option here.

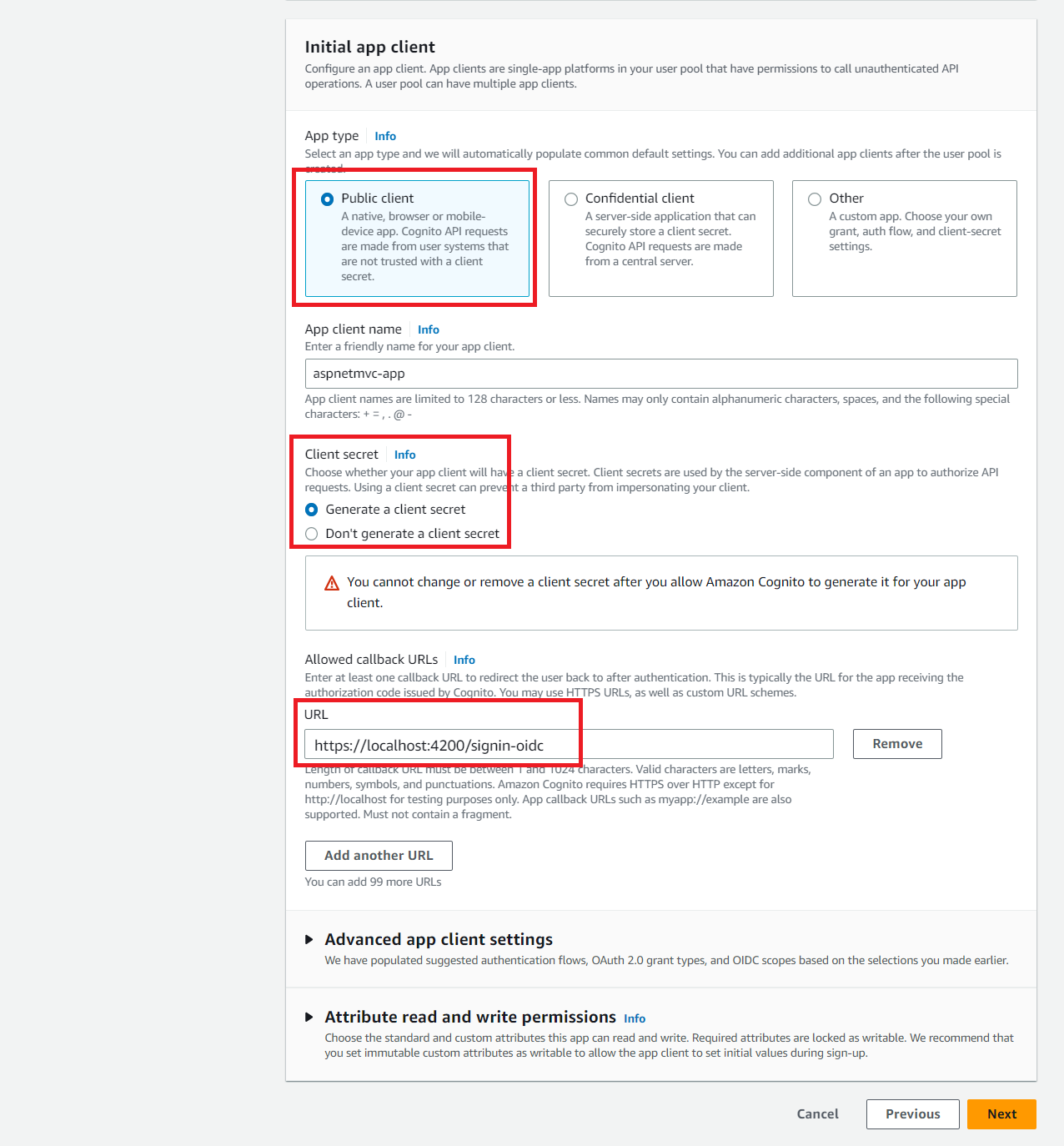


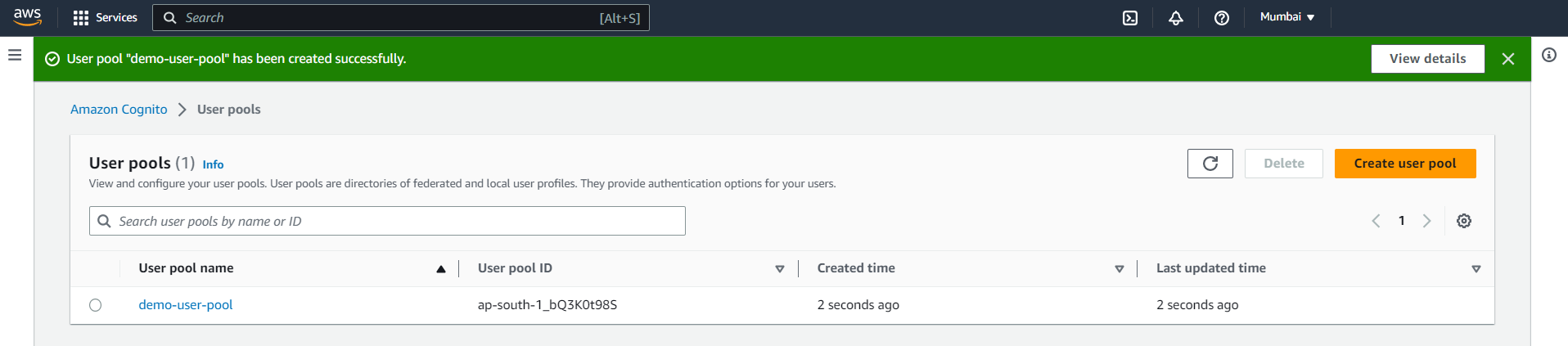
1. In **Integrate your app**, enter a name for your user pool, select **Use the Cognito Hosted UI**, choose **Use a Cognito domain** option for **Domain type**, and then enter a name for your Cognito domain(https://demo-user-pool-adi).



Next, create an **app client** for your web application. Choose **Public client** for the **App type**, choose **Generate a client secret** for **Client Secret**, and https://localhost:4200/signin-oidc as a callback URL.

☞ Remember that the callback URL must be HTTPS or you will encounter authentication errors.



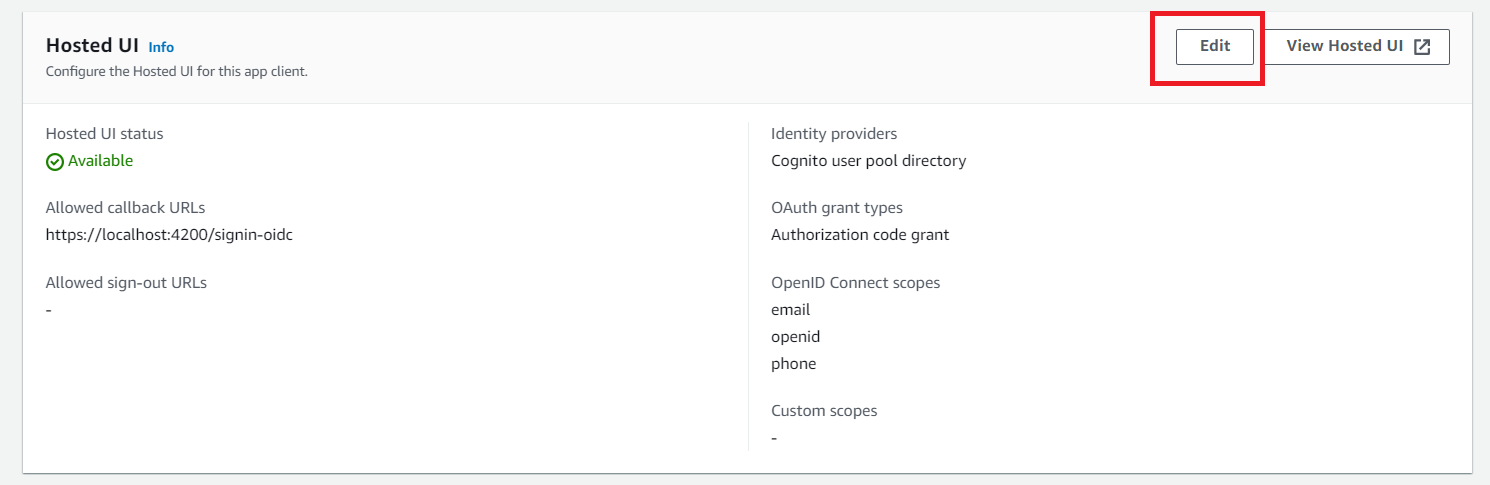
1. In **Review and create**, scroll down and select **Create user pool** to proceed.
2. Once the user pool has been created, it should look like this. 

## 2. Update sign-out URL for app client

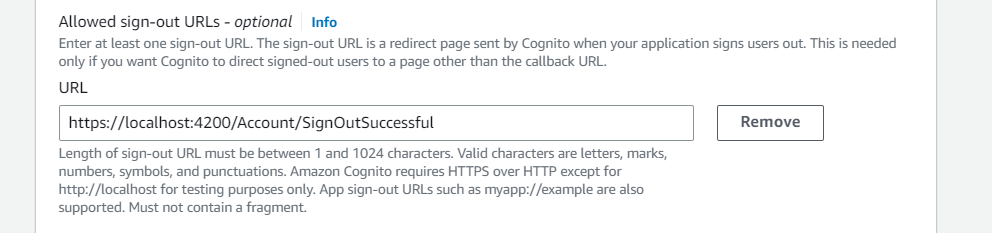
A sign-out URL indicates where your user is to be redirected after signing out.

Follow the steps below to update the sign-out URL.

1. Open the recently created user pool.
2. Navigate to **App Integration** tab, and scroll down to the bottom.
3. Choose the **app client** you created earlier in this post.
4. Click on **Edit** button in **Hosted UI** section.

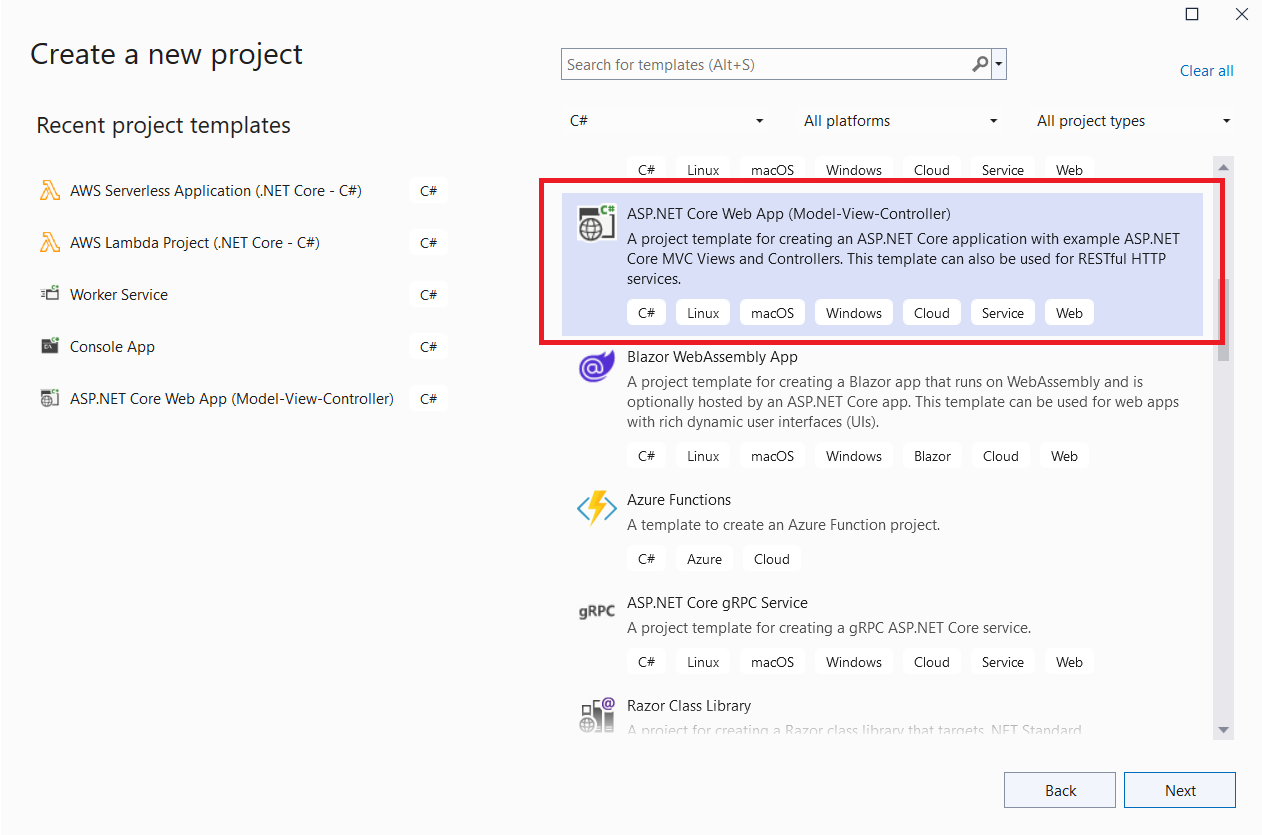


1. Click on **Add sign-out URL** button, and add https://localhost:4200/Account/SignOutSuccessful as a sign-out URL.



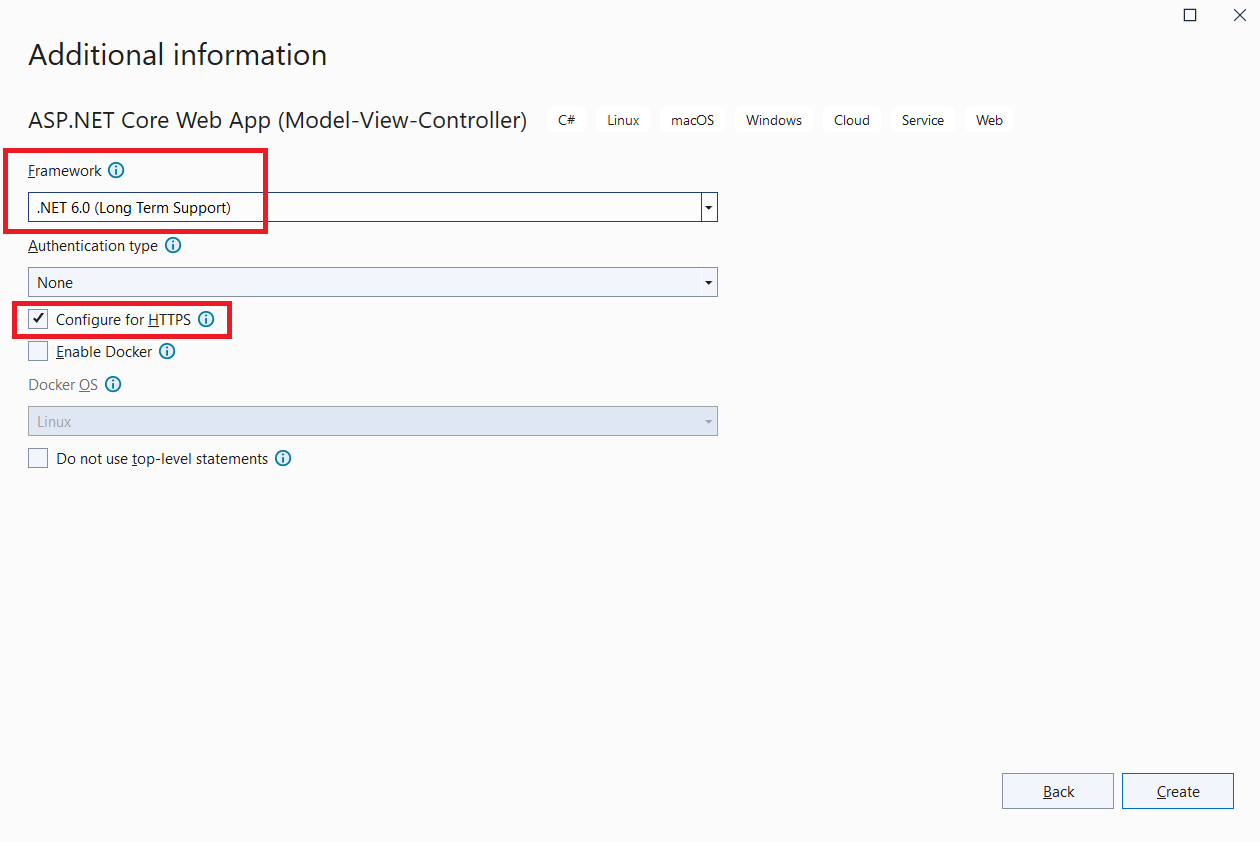
## 3. Creating a new ASP.NET Core MVC application

Let's create a new **ASP.NET Core MVC** application from Visual Studio.



Choose **Next**, enter the project and solution name, and then press **Next** again.

On **Additional information** dialog, make sure **Framework** is **.NET 6.0**, and **Configure for HTTPS** checkbox is selected.

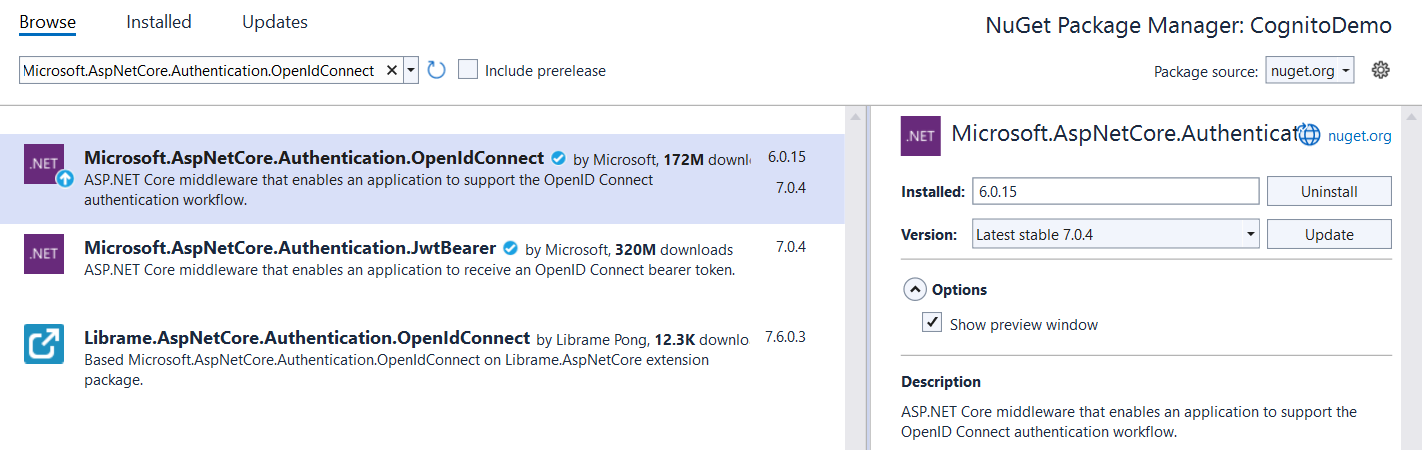


Your MVC app is now created successfully. In the next step, you will configure this app to use Cognito authentication.

## 4. Configure MVC application to use Cognito authentication

Install the Microsoft.AspNetCore.Authentication.OpenIdConnect NuGet package.

☞ Note that the latest version of this NuGet package may not support .NET 6, so install a version that supports your current .NET version.



Next, navigate to the appsettings.json file and replace the existing content with the following. Also, replace the placeholders with their actual values.

{

"Logging": {

"LogLevel": {

"Default": "Warning"

}

},

"AllowedHosts": "\*",

"Authentication": {

"Cognito": {

"ClientId": "<app client id from AWS Cognito>",

"ClientSecret": "<client-secret>",

"MetadataAddress": "https://cognito-idp.<your region>.amazonaws.com/<your-pool id>/.well-known/openid-configuration",

"ResponseType": "code",

"SaveToken": true,

"AppSignOutUrl": "/Account/SignOutSuccessful",

"CognitoDomain": "https://<custom domain name>.auth.<your region>.amazoncognito.com"

}

}

}

JSON

Copy

Next, Go to the **Program.cs** file, and make the changes as per the instructions given below.

1. Add the following namespaces on the top.

using Microsoft.AspNetCore.Authentication.Cookies;

using Microsoft.AspNetCore.Authentication.OpenIdConnect;

C#

1. Use the following code to add Cookie and OpenID authentications to your application.

builder.Services.AddAuthentication(item =>

{

item.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;

item.DefaultChallengeScheme = OpenIdConnectDefaults.AuthenticationScheme;

})

.AddCookie()

.AddOpenIdConnect(OpenIdConnectDefaults.AuthenticationScheme, options =>

{

options.ResponseType = builder.Configuration["Authentication:Cognito:ResponseType"];

options.MetadataAddress = builder.Configuration["Authentication:Cognito:MetadataAddress"];

options.ClientId = builder.Configuration["Authentication:Cognito:ClientId"];

options.ClientSecret = builder.Configuration["Authentication:Cognito:ClientSecret"];

options.Events = new OpenIdConnectEvents()

{

// This method enables logout from Amazon Cognito, and it is invoked before redirecting to the identity provider to sign out

OnRedirectToIdentityProviderForSignOut = OnRedirectToIdentityProviderForSignOut

};

options.Scope.Clear();

options.Scope.Add("openid");

options.Scope.Add("email");

options.Scope.Add("phone");

options.SaveTokens = Convert.ToBoolean(builder.Configuration["Authentication:Cognito:SaveToken"]);

});

C#

1. Make sure, both **Authentication** and **Authorization** middlewares are configured.

app.UseAuthentication();

app.UseAuthorization();

C#

1. Lastly, add the definition for OnRedirectToIdentityProviderForSignOut method.

// This method performs a Cognito sign-out, and then redirects user back to the application

Task OnRedirectToIdentityProviderForSignOut(RedirectContext context)

{

context.ProtocolMessage.Scope = "openid";

context.ProtocolMessage.ResponseType = "code";

var cognitoDomain = builder.Configuration["Authentication:Cognito:CognitoDomain"];

var clientId = builder.Configuration["Authentication:Cognito:ClientId"];

var logoutUrl = $"{context.Request.Scheme}://{context.Request.Host}{builder.Configuration["Authentication:Cognito:AppSignOutUrl"]}";

context.ProtocolMessage.IssuerAddress = $"{cognitoDomain}/logout?client\_id={clientId}&logout\_uri={logoutUrl}";

return Task.CompletedTask;

}

C#

After making the changes listed above, your Program.cs file should look like this.



Now, let's understand the changes you made in your **Program.cs** file.

1. First, you added the required namespaces.
2. Second, you added Cookie based authentication to your application. You also registered another authentication scheme named OpenIdConnect with AddOpenIdConnect method to authenticate users from Amazon Cognito.

Since you have set DefaultChallengeScheme to OpenIdConnectDefaults.AuthenticationScheme, the application redirects users to the configured Cognito hosted login page for unauthneticated requests, rather than redirecting them to the local /Account/Login page.

1. Third, you also configured OnRedirectToIdentityProviderForSignOut event for OpenIdConnect authentication scheme to perform Cognito signout. That event is invoked once the user performs HttpContext.SignOutAsync(OpenIdConnectDefaults.AuthenticationScheme) in the application. This method redirects the user to the Cognito logout endpoint with a logout\_uri and client\_id parameter. Cognito performs a sign-out by clearing the user's session cookies and then redirecting the user back to a logout\_uri that belongs to your application.

## 5. Modifications in MVC application to support Authentication

Now that you've configured Cognito authentication in your app, it's time to make some changes to see if authentication is working as expected.

Follow the instructions below to make the changes to your application.

1. Add [Authorize] attribute on HomeController class.

[Authorize]

public class HomeController : Controller

{

// controller code

}

C#

1. Update the content of ~\View\Home\Index.cshtml file with the following.

@using System.Security.Claims;

@{

ViewData["Title"] = "Home Page";

}

@if (User.Identity.IsAuthenticated)

{

<p>

Authenticated!

</p>

<table class="table table-bordered">

<tr>

<th>Claim Type</th>

<th>Value</th>

</tr>

@foreach (var item in User.Claims)

{

<tr>

<td>

@item.Type

</td>

<td>

@item.Value

</td>

</tr>

}

</table>

}

HTML

1. Create a new AccountController and add the following code there.

public class AccountController : Controller

{

public async Task SignOut()

{

// Note: To sign out the current user and delete their cookie, call SignOutAsync

// 1. Initiate signout for cookie based authentication scheme

await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);

// 2. Initiate signout for OpenID authentication scheme

await HttpContext.SignOutAsync(OpenIdConnectDefaults.AuthenticationScheme);

}

public ActionResult SignOutSuccessful()

{

return View();

}

}

C#

Copy

1. Add ~\View\Account\SignOutSuccessful.cshtml file with the following content.

<h1>SignOut</h1>

<p>You have been logged-out successfully from the application.</p>

HTML

Copy

1. In \_Layout.cshtml, add a sign-out link to the navigation bar.

@if (User.Identity.IsAuthenticated)

{

<li class="nav-item">

<a class="nav-link text-dark" asp-area="" asp-controller="Account" asp-action="SignOut">SignOut</a>

</li>

}

HTML

1. Finally, open the launchSettings.json file and configure the app to launch at the URL https://localhost:4200.

{

"iisSettings": {

...

},

"profiles": {

"CognitoDemo": {

"commandName": "Project",

"dotnetRunMessages": true,

"launchBrowser": true,

"applicationUrl": "https://localhost:4200;http://localhost:5026", // <-- This must include: https://localhost:4200

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

},

"IIS Express": {

...

}

}

}

JSON

## 6. Run the application and Test Cognito Sign-in

## 7. Add Claims-based authorization support

To enable [Claims-based authorization](https://learn.microsoft.com/en-us/aspnet/core/security/authorization/claims) with Cognito, you will leverage **Cognito Groups**. This requires you to create a Cognito group first.

### Create a new Cognito group

1. Go to the Amazon Cognito console, and choose the user pool that you created earlier.
2. Choose the **Groups** tab, and then choose **Create a group**.
3. On the **Create a group** page, in Group name, enter **Admin** for your new group.
4. Leave everything with defaults.
5. Choose **Create** to confirm.

### Add user to the Cognito Group

1. Choose the **Users** tab, and then choose a user.
2. On the user detail page, scroll down to the bottom, and click on **Add user to group** button in **Group memberships** section.
3. Add the user to the previously created **Admin** group.

After that, go to your **Program.cs** file, add the instructed code block in your file.

var builder = WebApplication.CreateBuilder(args);

// ------ Add this code block -----

builder.Services.AddAuthorization(options =>

{

options.AddPolicy("AdminOnly", policy => policy.RequireClaim("cognito:groups", "Admin"));

});

C#

Copy

Finally, modify the [Authorize] attribute to include the policy name.

[Authorize(Policy = "AdminOnly")]

public class HomeController : Controller

{

...

}

C#

That's all. You're done!

## 8. Obtaining Cognito Access Token

Because you are authenticating your users through Cognito, you may also require the Access Token in order to call your APIs that are secured through the same Cognito instance.

To retrieve the access token for your current user, use the code below in your MVC application.

string accessToken = HttpContext.GetTokenAsync("access\_token").Result;