Immagine che contiene testo, schermata, diagramma, Carattere

Descrizione generata automaticamente

**Flusso di Autenticazione di Amazon Cognito con OpenID Connect**

Amazon Cognito utilizza i flussi standard di OpenID Connect (OIDC) per l'autenticazione degli utenti e la gestione degli attributi. OpenID Connect è uno strato di identità basato sul framework di autorizzazione OAuth 2.0. Questo standard estende la funzionalità di un server di autorizzazione OAuth 2.0, includendo i token di autenticazione e le informazioni sugli utenti finali (claims) utilizzando transazioni simili a REST tra domini di sicurezza.

Una volta che un utente è stato autenticato, Amazon Cognito fornisce dei token di pool utenti (user pool tokens) all'applicazione. Questi token possono essere utilizzati per gestire l'accesso alle risorse lato applicazione. I token del pool utenti di Cognito sono semplicemente token OIDC standard. In questo scenario, il pool utenti di Amazon Cognito agisce come Identity Provider (IDP) e server di autorizzazione, mentre l'applicazione funge da resource server in una transazione OIDC standard.

**Descrizione del diagramma**

Il diagramma rappresenta un esempio di flusso di autorizzazione tramite il codice di autorizzazione (Authorization Code) con verifica tramite Proof Key for Code Exchange (PKCE). Questo flusso è uno dei vari flussi standard disponibili in OpenID Connect. Anche se ci sono differenze chiave tra i vari flussi disponibili, i loro casi d'uso ottimali e i token che forniscono, il flusso con codice di autorizzazione è un buon esempio poiché fornisce tutti i tipi di token disponibili.

Nel diagramma vediamo i seguenti passaggi:

1. **L'utente non autenticato** con un account esistente clicca su un link per accedere a un sito web che utilizza un pool utenti di Amazon Cognito per la gestione degli utenti e l'autenticazione.
2. **L'applicazione** genera una richiesta di codice di autorizzazione, inviando un code\_challenge crittograficamente generato tramite un code\_verifier casuale.
3. L'utente viene **reindirizzato alla pagina di login** del pool utenti di Cognito.
4. **L'utente inserisce le sue credenziali** e, eventualmente, fornisce il consenso affinché Cognito condivida le informazioni memorizzate nel pool utenti con l'applicazione.
5. **Il pool utenti verifica le credenziali dell'utente** e reindirizza l'utente all'applicazione con un **codice di autorizzazione** a uso unico e di breve durata. Questo codice è la prova che l'utente ha fornito le credenziali.
6. L'applicazione invia il **codice di autorizzazione** insieme al code\_verifier, precedentemente creato, al server di autorizzazione di Amazon Cognito.
7. Il server di autorizzazione verifica il code\_verifier e il code\_challenge. Se corrispondono crittograficamente, il server di autorizzazione convalida che l'applicazione possa fare questa richiesta di informazioni sull'utente dal pool utenti.
8. **Il server di autorizzazione risponde** all'applicazione con un **ID token firmato** che attesta che il server di autorizzazione ha validato le credenziali dell'utente e contiene le informazioni (claims) sull'utente. Insieme all'ID token, vengono inviati un **access token** e un **refresh token** che l'applicazione può usare per accedere a risorse protette o per richiedere nuovi token senza ulteriore interazione dell'utente.

**Concetti Chiave**

* **PKCE (Proof Key for Code Exchange)**: PKCE è un'estensione di sicurezza usata per proteggere il flusso di autorizzazione del codice quando non si utilizza un client segreto. Viene utilizzato per prevenire attacchi come l'intercettazione del codice (code interception attacks).
* **Token di Accesso (Access Token)**: Viene utilizzato dall'applicazione per fare richieste a risorse protette in nome dell'utente.
* **ID Token**: Fornisce informazioni sull'utente autenticato (claims), solitamente utilizzato dall'applicazione per identificare l'utente.
* **Refresh Token**: Permette all'applicazione di ottenere nuovi token di accesso senza dover chiedere all'utente di autenticarsi nuovamente.

**Amazon Cognito Identity Pools**

Gli **Identity Pools** di Amazon Cognito consentono agli sviluppatori di applicazioni di collegare l'accesso degli utenti e delle applicazioni alle risorse AWS. Simile al funzionamento degli utenti federati IAM di AWS, gli utenti degli Identity Pools possono ottenere credenziali temporanee tramite l'AWS Security Token Service (STS). Tuttavia, non esiste un account utente IAM corrispondente per questi utenti; l'accesso è gestito tramite una **trust policy** IAM.

Un Identity Pool richiede un **identity provider federato** per fornire le identità degli utenti, che può essere un pool utenti di Cognito, un provider sociale come Facebook o Twitter, un provider di identità basato su SAML2 o OIDC, o una combinazione di essi. Gli Identity Pools autorizzano l'accesso alle risorse AWS utilizzate da un'applicazione, spesso ospitata sulla piattaforma AWS stessa.

Sebbene possano sembrare ridondanti rispetto ad altri servizi di identità di AWS, gli Identity Pools sono progettati appositamente per le applicazioni che devono essere distribuite su AWS. Offrono SDK e codice di esempio per facilitarne l'adozione da parte degli sviluppatori.

**Casi d'Uso di Amazon Cognito**

**1 caso d’Uso**

Immagine che contiene testo, diagramma, linea, design

Descrizione generata automaticamente

Il pattern di design più semplice da implementare quando si utilizza Amazon Cognito è la gestione completamente esternalizzata degli account utente e l'autenticazione. In questo pattern, il pool utenti di Cognito funge da **Identity Provider (IDP)** e archivio degli utenti per l'applicazione. Le applicazioni possono sfruttare la gestione degli account ospitata da Amazon Cognito, insieme ai processi di registrazione e verifica, per creare nuovi account utente nel pool se un utente non ha credenziali per accedere all'applicazione. Una volta che l'utente registra il proprio account e imposta le sue credenziali, l'applicazione si affida a Amazon Cognito come provider di identità. Utilizzando un flusso di autenticazione basato su standard, l'applicazione riceve conferma da Cognito che l'utente è stato autenticato tramite l'emissione di un **ID token** e un **access token** firmati da Amazon Cognito.

**2 caso d’Uso**

Immagine che contiene diagramma, linea, testo, origami

Descrizione generata automaticamente

Secondo questo modello, l'unica differenza significativa è la possibilità di scegliere dove avviene l'autenticazione dell'utente, così come dove vengono gestite le sue credenziali. Il pool utenti di Amazon Cognito può fare riferimento agli account gestiti localmente all'interno della sua directory o a un provider federato al momento dell'autenticazione dell'utente; tuttavia, l'applicazione continuerà a considerare solo Amazon Cognito come fonte di identità dell'utente. Anche se gli utenti federati delegano l'autenticazione a un provider di identità, essi esistono comunque come record all'interno del pool utenti, e qualsiasi informazione fornita all'applicazione su questi utenti proviene da Amazon Cognito utilizzando gli attributi del pool stesso.

**3 caso d’Uso**

Immagine che contiene testo, schermata, diagramma, Carattere

Descrizione generata automaticamente

**Autenticazione e autorizzazione degli utenti per l'accesso alle risorse dell'applicazione**: Questo modello è simile al precedente, con una differenza chiave: l'applicazione che utilizza Amazon Cognito come provider di identità funge anche da server di risorse. Oltre alle funzionalità di autenticazione e registrazione degli utenti, l'applicazione utilizza il pool utenti di Amazon Cognito come server di autorizzazione secondo il modello OAuth2. Nel diagramma precedente, l'applicazione si affida a Cognito per l'autenticazione dell'utente, la registrazione di nuovi utenti e la verifica degli account. Il pool utenti di Amazon Cognito può opzionalmente fare riferimento a un provider di identità federato per l'autenticazione. Tuttavia, in questo caso, l'applicazione richiede che gli utenti possano accedere a una o più delle risorse disponibili una volta autenticati. I pool utenti di Amazon Cognito possono facilitare questo attraverso l'assegnazione di gruppi all'interno della sua directory utenti. Possiamo assegnare autorizzazioni, come "i membri del gruppo 1 possono accedere alla risorsa 1", che mappano gli ambiti (scopes) che Amazon Cognito (come server di autorizzazione) e l'applicazione (come server di risorse) utilizzano per limitare l'accesso, che viene concesso tramite il token di accesso rilasciato al momento dell'autenticazione dell'utente. Con tutte queste discussioni su ambiti, server di risorse e server di autorizzazione, si potrebbe pensare che anche le API siano coinvolte in questo modello. In effetti, potrebbero esserlo come risorse disponibili sul server dell'applicazione/risorse. Tuttavia, poiché esporre endpoint API non è una buona pratica di sicurezza, AWS consiglia di associare il pool utenti di Amazon Cognito con AWS API Gateway.

Immagine che contiene testo, schermata, diagramma, linea

Descrizione generata automaticamente

In questa relazione, oltre ai rapporti e ai servizi precedentemente stabiliti tra i pool utenti di Amazon Cognito e l'applicazione, dobbiamo anche creare una relazione tra il pool utenti e il servizio Amazon API Gateway, in modo che Amazon API Gateway sappia che Amazon Cognito può fungere da server di autorizzazione per le API dell'applicazione che sta proxyando. Proprio come nella variante precedente di questo modello, i gruppi di utenti all'interno del pool utenti di Amazon Cognito sono mappati ad ambiti (scopes) che determinano quali API il token di accesso è autorizzato a chiamare. Questo modello è molto specifico per la piattaforma e fa riferimento ai servizi AWS coinvolti.

**4 caso d’Uso**

Immagine che contiene testo, diagramma, Piano

Descrizione generata automaticamente

**Autenticazione degli utenti e accesso ai servizi AWS esposti tramite un'applicazione**

Il modello successivo è il primo a includere sia i pool di identità di Amazon Cognito che un caso d'uso in cui un'applicazione e i suoi utenti devono accedere alle risorse AWS. Immaginiamo un'applicazione che utilizza Amazon Cognito per la gestione delle identità e parte della funzionalità dell'app prevede il caricamento e il download di file da un repository, che in realtà è solo un bucket S3:

Tutte le capacità e le funzionalità dei pool utenti di Amazon Cognito rimangono le stesse come nei modelli precedenti. La novità qui è che l'applicazione chiama il pool di identità per scambiare i token del pool utenti di Amazon Cognito dell'utente con credenziali AWS IAM, che consentiranno l'accesso alle risorse AWS in conformità con una policy di fiducia predefinita. L'applicazione delega l'autorizzazione ad AWS IAM. Una volta che l'utente assume tali credenziali – più specificamente, un ruolo definito e delimitato per consentire l'accesso da quel pool di identità – può interagire con la risorsa AWS tramite l'applicazione.

**5 caso d’Uso**

Immagine che contiene testo, schermata, diagramma, Biglietto Post-it

Descrizione generata automaticamente

Autenticazione federata degli utenti e accesso ai servizi AWS esposti tramite un'applicazione

Questo modello è quasi identico al precedente, con la differenza che non utilizza affatto i pool utenti di Amazon Cognito, affidandosi esclusivamente ai provider di identità federati. Questo potrebbe creare confusione, poiché un pool utenti di Amazon Cognito può anche utilizzare un provider di identità federato per l'autenticazione; la differenza principale è che un pool utenti fornisce sia l'autenticazione che la gestione degli utenti per l'applicazione, indipendentemente dal fatto che il pool utenti stesso si federi o meno con un provider di identità esterno. D'altra parte, un pool di identità di Amazon Cognito è indifferente al provider di identità utilizzato dall'applicazione per l'autenticazione degli utenti. I pool di identità si concentrano solo sui token emessi dal provider di identità autorevole, poiché questi attributi possono essere utilizzati per determinare l'autorizzazione e la mappatura dei privilegi:

Molte delle stesse meccaniche dell'architettura precedente sono in gioco anche qui, ma questa volta, al posto di un token del pool utenti di Amazon Cognito, viene utilizzato il token del provider federato per identificare l'utente, raccogliere le mappature critiche degli attributi e allineare l'utente alla policy di fiducia appropriata che regola il ruolo assunto durante lo scambio di token.

Queste architetture comuni possono aiutarci a concettualizzare come i pool utenti e i pool di identità di Amazon Cognito possano essere utilizzati per risolvere le sfide di identità applicativa, ma non ci danno ancora un'idea delle opzioni disponibili all'interno di questi servizi. Nella sezione successiva, impareremo come creare un pool utenti di Amazon Cognito e acquisire una comprensione migliore del servizio.

Integrazione con asp.net framework

**Scenario 1: Integrazione di AWS Cognito con applicazioni .NET Framework 4.0 utilizzando AWSSDK.CognitoIdentityProvider**

**Immagine che contiene testo, diagramma, schermata, Parallelo

Descrizione generata automaticamente**

**Descrizione del contesto**

L'applicazione legacy utilizza attualmente un database locale per la gestione degli utenti tramite **ASP.NET Membership Provider**. L'obiettivo è integrare **Amazon Cognito** come provider di identità esterno, pur mantenendo il database locale per la sincronizzazione degli utenti.

**Opzioni di Integrazione:**

1. **Gestione diretta degli utenti Cognito** all'interno dell'applicazione legacy utilizzando il pacchetto **AWSSDK.CognitoIdentityProvider**.
2. **Sincronizzazione tra il database locale degli utenti e Cognito**, consentendo agli utenti di esistere sia localmente che su Cognito e gestendo il login tramite Cognito come IDP (Identity Provider).

**Step 1: Aggiunta del pacchetto AWSSDK.CognitoIdentityProvider**

Per prima cosa, dovrai aggiungere il pacchetto **AWSSDK.CognitoIdentityProvider** (versione compatibile con **.NET Framework 4.0**). Puoi farlo tramite **NuGet Package Manager** in Visual Studio.

Ecco come aggiungere il pacchetto:

Install-Package AWSSDK.CognitoIdentityProvider -Version 3.7.403.14

**Step 2: Configurazione delle credenziali AWS**

Prima di interagire con **Amazon Cognito**, è necessario configurare le credenziali AWS per poter accedere ai servizi. Puoi farlo utilizzando il file app.config o web.config dell'applicazione.

Esempio di configurazione delle credenziali nel web.config:

<configuration>

<appSettings>

<add key="AWSAccessKey" value="your-access-key"/>

<add key="AWSSecretKey" value="your-secret-key"/>

<add key="AWSRegion" value="your-region"/>

</appSettings>

</configuration>

**Step 3: Interazione con Amazon Cognito per la gestione degli utenti**

Con il package **AWSSDK.CognitoIdentityProvider**, puoi gestire gli utenti direttamente all'interno della tua applicazione legacy. Qui mostriamo come autenticare un utente con Cognito e sincronizzarlo con il database locale.

**Esempio di codice per l'autenticazione tramite Cognito:**

using Amazon.CognitoIdentityProvider;

using Amazon.CognitoIdentityProvider.Model;

public class CognitoService

{

private readonly AmazonCognitoIdentityProviderClient \_cognitoClient;

private readonly string \_clientId = "your-cognito-client-id";

private readonly string \_userPoolId = "your-user-pool-id";

public CognitoService()

{

var region = Amazon.RegionEndpoint.USEast1; // Sostituisci con la tua regione

\_cognitoClient = new AmazonCognitoIdentityProviderClient(region);

}

public async Task<AdminInitiateAuthResponse> AuthenticateUser(string username, string password)

{

var authRequest = new AdminInitiateAuthRequest

{

UserPoolId = \_userPoolId,

ClientId = \_clientId,

AuthFlow = AuthFlowType.ADMIN\_NO\_SRP\_AUTH,

AuthParameters = new Dictionary<string, string>

{

{ "USERNAME", username },

{ "PASSWORD", password }

}

};

var authResponse = await \_cognitoClient.AdminInitiateAuthAsync(authRequest);

// Controlla se l'autenticazione ha avuto successo

if (authResponse.AuthenticationResult != null)

{

// Sincronizza l'utente nel database locale se necessario

SyncUserWithLocalDb(username);

}

return authResponse;

}

private void SyncUserWithLocalDb(string username)

{

// Logica per sincronizzare gli utenti di Cognito con il database locale

// Controlla se l'utente esiste nel database locale e se non esiste, crea l'utente

}

}

**Sincronizzazione utenti con il database locale:**

* Se un utente si autentica tramite Cognito ma non esiste ancora nel database locale, puoi aggiungere l'utente nel database.
* In alternativa, puoi aggiornare i dati degli utenti locali con i nuovi attributi ottenuti da Cognito (ad esempio email o nome).

**Step 4: Uso dei token di accesso e ID token**

Dopo l'autenticazione, **Cognito** emette un **ID token** e un **access token**. Questi token possono essere utilizzati per autorizzare l'accesso alle risorse dell'applicazione.

**Esempio di come recuperare i token dall'autenticazione:**

var authResponse = await AuthenticateUser(username, password);

var idToken = authResponse.AuthenticationResult.IdToken;

var accessToken = authResponse.AuthenticationResult.AccessToken;

// Utilizza questi token per autorizzare le chiamate successive o per memorizzarli in sessione

**Benefici dell'integrazione diretta con Cognito:**

1. **Riduzione del carico gestionale**: Con Cognito come IDP, la gestione delle credenziali e delle policy di sicurezza viene esternalizzata.
2. **Sicurezza migliorata**: Cognito gestisce la crittografia dei token e le policy di autenticazione multi-fattore (MFA).
3. **Facilità di sincronizzazione**: L'integrazione può essere graduale, consentendo una sincronizzazione trasparente con il database locale degli utenti esistente.

**Considerazioni:**

* È importante definire una logica robusta per la sincronizzazione dei dati tra il database locale e Cognito.
* Potrebbe essere necessario gestire eventuali conflitti di dati se gli utenti esistono sia nel database locale che su Cognito.

**Scenario 2: Creazione di un'applicazione ASP.NET Core come proxy per un'app legacy .NET Framework 4.0**

**Immagine che contiene testo, schermata, diagramma, Carattere

Descrizione generata automaticamente**

**Descrizione del contesto**

In questo scenario, l'applicazione legacy non viene modificata direttamente. Invece, si crea una nuova applicazione ASP.NET Core che si occupa della gestione dell'autenticazione degli utenti tramite **Amazon Cognito** e quindi inoltra le richieste all'app legacy. Questo approccio offre maggiore flessibilità e separazione delle preoccupazioni.

**Step 1: Creazione di un nuovo progetto ASP.NET Core**

Puoi creare un nuovo progetto ASP.NET Core utilizzando il modello **Web Application** in Visual Studio o tramite la CLI .NET.

Esempio di comando per creare un nuovo progetto:

dotnet new webapp -n ProxyApp

Perfetto! Passiamo al secondo scenario, in cui creiamo una nuova applicazione **ASP.NET Core** che funge da proxy per l'applicazione legacy **ASP.NET WebForms in .NET Framework 4.0**. Questa nuova applicazione utilizzerà i pacchetti **Amazon.AspNetCore.Identity.Cognito** e **Amazon.Extensions.CognitoAuthentication** per autenticare gli utenti di **Cognito** e gestire l'accesso all'app legacy.

**Scenario 2: Creazione di un'applicazione ASP.NET Core come proxy per un'app legacy .NET Framework 4.0**

**Descrizione del contesto**

In questo scenario, l'applicazione legacy non viene modificata direttamente. Invece, si crea una nuova applicazione ASP.NET Core che si occupa della gestione dell'autenticazione degli utenti tramite **Amazon Cognito** e quindi inoltra le richieste all'app legacy. Questo approccio offre maggiore flessibilità e separazione delle preoccupazioni.

**Step 1: Creazione di un nuovo progetto ASP.NET Core**

Puoi creare un nuovo progetto ASP.NET Core utilizzando il modello **Web Application** in Visual Studio o tramite la CLI .NET.

Esempio di comando per creare un nuovo progetto:

bash

Copia codice

dotnet new webapp -n ProxyApp

**Step 2: Aggiunta dei pacchetti NuGet necessari**

All'interno del progetto ASP.NET Core, aggiungi i seguenti pacchetti NuGet:

dotnet add package Amazon.AspNetCore.Identity.Cognito

dotnet add package Amazon.Extensions.CognitoAuthentication

Questi pacchetti forniscono un'integrazione facile con **Amazon Cognito** e **ASP.NET Core Identity**

**Step 3: Configurazione di AWS Cognito nell'app ASP.NET Core**

Apri il file appsettings.json e configura le impostazioni per **Cognito**:

{

"AWS": {

"Region": "your-region",

"Cognito": {

"UserPoolId": "your-user-pool-id",

"ClientId": "your-cognito-client-id",

"IdentityPoolId": "your-identity-pool-id"

}

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

**Step 4: Configurazione di Identity e Cognito nel Startup.cs**

Aggiungi la configurazione per **Cognito** nel metodo ConfigureServices:

using Amazon.Extensions.CognitoAuthentication;

public class Startup

{

public void ConfigureServices(IServiceCollection services)

{

services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = CookieAuthenticationDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = CookieAuthenticationDefaults.AuthenticationScheme;

})

.AddCookie()

.AddCognitoIdentityProvider(options =>

{

options.UserPoolId = Configuration["AWS:Cognito:UserPoolId"];

options.ClientId = Configuration["AWS:Cognito:ClientId"];

});

services.AddControllersWithViews();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

});

}

}

**Step 5: Implementazione del controller per la gestione dell'autenticazione**

Crea un controller che gestisca il login e l'accesso a Cognito. Ecco un esempio di controller AccountController:

using Microsoft.AspNetCore.Mvc;

using Amazon.AspNetCore.Identity.Cognito;

using Amazon.CognitoIdentityProvider;

using System.Threading.Tasks;

public class AccountController : Controller

{

private readonly UserManager<CognitoUser> \_userManager;

private readonly SignInManager<CognitoUser> \_signInManager;

public AccountController(UserManager<CognitoUser> userManager, SignInManager<CognitoUser> signInManager)

{

\_userManager = userManager;

\_signInManager = signInManager;

}

[HttpGet]

public IActionResult Login(string returnUrl = null)

{

ViewData["ReturnUrl"] = returnUrl;

return View();

}

[HttpPost]

public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)

{

if (ModelState.IsValid)

{

var result = await \_signInManager.PasswordSignInAsync(model.Username, model.Password, model.RememberMe, lockoutOnFailure: false);

if (result.Succeeded)

{

return LocalRedirect(returnUrl ?? Url.Action("Index", "Home"));

}

ModelState.AddModelError(string.Empty, "Invalid login attempt.");

}

return View(model);

}

[HttpPost]

public async Task<IActionResult> Logout()

{

await \_signInManager.SignOutAsync();

return RedirectToAction("Index", "Home");

}

}

**Modello di Login:**

Assicurati di avere un modello di vista per il login (LoginViewModel):

public class LoginViewModel

{

public string Username { get; set; }

public string Password { get; set; }

public bool RememberMe { get; set; }

}

**Step 6: Inoltro delle richieste all'applicazione legacy**

Dopo l'autenticazione, puoi inoltrare le richieste all'app legacy, passando i token o le informazioni necessarie per l'autenticazione. Puoi farlo utilizzando **HttpClient**.

Esempio di come inoltrare una richiesta:

using System.Net.Http;

public async Task<IActionResult> CallLegacyApi()

{

using (var client = new HttpClient())

{

client.BaseAddress = new Uri("https://legacy-app-url/api/");

// Aggiungi i token di accesso o altre intestazioni necessarie

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);

var response = await client.GetAsync("your-legacy-endpoint");

if (response.IsSuccessStatusCode)

{

var data = await response.Content.ReadAsStringAsync();

// Elaborare i dati

}

return View();

}

}

**Benefici di questo approccio:**

1. **Separazione delle preoccupazioni**: L'applicazione legacy rimane intatta e si evita di modificare il codice esistente.
2. **Scalabilità**: Consente di sfruttare le nuove funzionalità di ASP.NET Core senza dover riscrivere completamente l'applicazione legacy.
3. **Flessibilità**: È possibile implementare nuove funzionalità o aggiornamenti nell'app proxy senza influenzare l'app legacy.

**Considerazioni:**

* Potresti dover gestire la sessione e i token di accesso in modo sicuro nel contesto dell'app proxy.
* Assicurati di proteggere adeguatamente le comunicazioni tra l'app proxy e l'app legacy, specialmente se si utilizzano informazioni sensibili.

**Scenario 3: Integrazione di AWS Cognito in un'applicazione ASP.NET Framework 4.5 utilizzando OWIN**

**Immagine che contiene diagramma, testo, linea, Piano

Descrizione generata automaticamente**

**Descrizione del contesto**

**OWIN** (Open Web Interface for .NET) è un'interfaccia standard che consente di separare le applicazioni ASP.NET dalle pipeline di hosting. È stato progettato per consentire un'architettura più flessibile e modulare. In questo scenario, utilizzeremo OWIN per integrare **AWS Cognito** come provider di autenticazione esterna utilizzando **OpenID Connect** (OIDC).

**Step 1: Aggiunta dei pacchetti NuGet necessari**

Apri il tuo progetto ASP.NET Framework 4.5 e aggiungi i seguenti pacchetti NuGet:

Install-Package Microsoft.Owin

Install-Package Microsoft.Owin.Security

Install-Package Microsoft.Owin.Security.Cookies

Install-Package Microsoft.Owin.Security.OpenIdConnect

Install-Package AWSSDK.CognitoIdentityProvider

Questi pacchetti forniscono gli strumenti necessari per configurare OWIN e autenticare gli utenti tramite **AWS Cognito**.

**Step 2: Configurazione di OWIN**

Crea un file chiamato Startup.cs nella radice del tuo progetto e aggiungi la seguente configurazione:

using Microsoft.Owin;

using Microsoft.Owin.Security.Cookies;

using Microsoft.Owin.Security.OpenIdConnect;

using Owin;

[assembly: OwinStartup(typeof(YourNamespace.Startup))]

namespace YourNamespace

{

public class Startup

{

public void Configuration(IAppBuilder app)

{

app.UseCookieAuthentication(new CookieAuthenticationOptions

{

AuthenticationType = "ApplicationCookie",

LoginPath = new PathString("/Account/Login")

});

app.UseOpenIdConnectAuthentication(new OpenIdConnectAuthenticationOptions

{

ClientId = "your-cognito-client-id",

Authority = "https://cognito-idp.{region}.amazonaws.com/{user-pool-id}",

RedirectUri = "https://your-app-url/signin-oidc",

ResponseType = "code",

Scope = "openid profile",

SignInAsAuthenticationType = "ApplicationCookie",

Notifications = new OpenIdConnectAuthenticationNotifications

{

AuthenticationFailed = notification =>

{

// Gestisci eventuali errori di autenticazione

notification.HandleResponse();

notification.Response.Redirect("/Error");

return Task.FromResult(0);

}

}

});

}

}

}

**Spiegazione della configurazione**

* **CookieAuthenticationOptions**: configura il middleware di autenticazione dei cookie. Quando un utente effettua il login, il middleware gestisce la creazione e la gestione dei cookie di autenticazione.
* **OpenIdConnectAuthenticationOptions**: imposta le opzioni per OIDC. Qui, definisci il tuo **ClientId**, l'**Authority** (che punta al pool di utenti Cognito), l'**URI di reindirizzamento** e altre impostazioni.

**Step 3: Creazione del controller per la gestione dell'autenticazione**

Crea un controller chiamato AccountController per gestire le operazioni di login e logout:

using Microsoft.AspNet.Identity;

using Microsoft.Owin.Security;

using System.Security.Claims;

using System.Threading.Tasks;

using System.Web.Mvc;

public class AccountController : Controller

{

private readonly IAuthenticationManager \_authenticationManager;

public AccountController(IAuthenticationManager authenticationManager)

{

\_authenticationManager = authenticationManager;

}

[HttpGet]

public ActionResult Login(string returnUrl)

{

ViewBag.ReturnUrl = returnUrl;

return View();

}

[HttpGet]

public ActionResult Logout()

{

\_authenticationManager.SignOut(DefaultAuthenticationTypes.ApplicationCookie);

return RedirectToAction("Index", "Home");

}

}

**Step 4: Creazione della vista di login**

Crea una vista Login.cshtml per consentire agli utenti di effettuare il login. In questo caso, la vista può essere molto semplice poiché il login avviene tramite la redirezione a Cognito.

@{

ViewBag.Title = "Login";

}

<h2>Login</h2>

<p>

<a href="@Url.Action("Login", "Account", new { returnUrl = ViewBag.ReturnUrl })">Login con AWS Cognito</a>

</p>

**Step 5: Sincronizzazione degli account locali con Cognito**

Se desideri sincronizzare gli account esistenti nel database locale con **AWS Cognito**, puoi creare una funzionalità per registrare gli utenti in Cognito quando vengono creati o modificati nel tuo database. Puoi farlo utilizzando l'SDK di Cognito in .NET.

Esempio di codice per registrare un utente:

using Amazon.CognitoIdentityProvider;

using Amazon.CognitoIdentityProvider.Model;

// Metodo per registrare un utente

public async Task RegisterUser(string username, string password, string email)

{

using (var client = new AmazonCognitoIdentityProviderClient())

{

var request = new SignUpRequest

{

ClientId = "your-cognito-client-id",

Username = username,

Password = password,

UserAttributes = new List<AttributeType>

{

new AttributeType { Name = "email", Value = email }

}

};

var response = await client.SignUpAsync(request);

// Gestisci la risposta di registrazione

}

}

**Benefici di questo approccio:**

1. **Flessibilità**: L'utilizzo di OWIN consente di integrare facilmente Cognito come provider di autenticazione senza dover modificare la logica esistente dell'app.
2. **Standardizzazione**: OWIN fornisce un'interfaccia standard che può essere utilizzata per altri provider di autenticazione in futuro.
3. **Sincronizzazione degli account**: Puoi facilmente gestire la sincronizzazione degli account utente tra il tuo database locale e **Cognito**.

**Considerazioni:**

* Assicurati di gestire le informazioni sensibili come le credenziali degli utenti in modo sicuro.
* Verifica le regole di sicurezza e le politiche di accesso su **Cognito** per garantire che solo gli utenti autorizzati possano accedere alle risorse.

**Opzione B: Integrazione in un'applicazione ASP.NET Web Forms in .NET Framework 4.5 con OWIN**

Se l'applicazione Web Forms è basata su **.NET Framework 4.5**, puoi implementare una soluzione più moderna utilizzando **OWIN** per gestire l'autenticazione.

1. **Installazione dei pacchetti NuGet** Aggiungi i seguenti pacchetti al tuo progetto:

Install-Package Microsoft.Owin

Install-Package Microsoft.Owin.Security

Install-Package Microsoft.Owin.Security.Cookies

Install-Package Microsoft.Owin.Security.OpenIdConnect

**Configurazione di OWIN** Crea un file Startup.cs e configura OWIN per utilizzare **AWS Cognito** come provider di autenticazione:

using Microsoft.Owin;

using Microsoft.Owin.Security.Cookies;

using Microsoft.Owin.Security.OpenIdConnect;

using Owin;

[assembly: OwinStartup(typeof(YourNamespace.Startup))]

namespace YourNamespace

{

public class Startup

{

public void Configuration(IAppBuilder app)

{

app.UseCookieAuthentication(new CookieAuthenticationOptions

{

AuthenticationType = "ApplicationCookie",

LoginPath = new PathString("/Login.aspx")

});

app.UseOpenIdConnectAuthentication(new OpenIdConnectAuthenticationOptions

{

ClientId = "your-cognito-client-id",

Authority = "https://cognito-idp.{region}.amazonaws.com/{user-pool-id}",

RedirectUri = "https://your-app-url/signin-oidc",

ResponseType = "code",

Scope = "openid profile",

SignInAsAuthenticationType = "ApplicationCookie",

Notifications = new OpenIdConnectAuthenticationNotifications

{

AuthenticationFailed = notification =>

{

// Gestisci eventuali errori di autenticazione

notification.HandleResponse();

notification.Response.Redirect("/Error.aspx");

return Task.FromResult(0);

}

}

});

}

}

}

**Gestione della Login in Web Forms** Nella tua pagina di login, puoi semplicemente reindirizzare a Cognito per l'autenticazione:

protected void btnLogin\_Click(object sender, EventArgs e)

{

// Reindirizza a Cognito per il login

HttpContext.Current.GetOwinContext().Authentication.Challenge(new AuthenticationProperties

{

RedirectUri = "/Default.aspx"

}, "OpenIdConnect");

}

**Sincronizzazione degli account** Analogamente all'opzione precedente, puoi registrare e sincronizzare gli utenti esistenti nel database locale con Cognito, come mostrato sopra.