Amazon Cognito

Developer Guide

Version Last Updated: 08/26/2017

Amazon Cognito: Developer Guide

Copyright © 2017 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

- What Is Amazon Cognito? 1
  - What's New 1
  - Features of Amazon Cognito 1
  - Accessing Amazon Cognito 2
  - Are You a First-Time Amazon Cognito User? 2
  - Pricing for Amazon Cognito 2
  - SDKs for Amazon Cognito 3
  - SDKs for Amazon Cognito User Pool App Integration and Federation 3
- Getting Started with User Pools App Integration and Federation 4
  - Setting Up an Amazon Cognito User Pool as Your User Directory 4
  - Configuring App Integration 5
  - Configuring Federation with a Social Identity Provider 5
  - Configuring Federation with a SAML 2.0 Identity Provider 6
  - Integrating the Sign-in Flow with Your App 8
  - Where to Find the SDKs and Sample Apps for App Integration and Federation 8
  - Where to Go from Here 8
- Amazon Cognito User Pools 9
  - Getting Started with Amazon Cognito User Pools 10
    - Sign Up for an AWS Account 10
    - Creating a User Pool in Amazon Cognito 10
    - Install the SDK 10
    - Integrate Your User Pool into Your App 11
  - Setting up User Pools 11
    - Creating a New User Pool Using the Console 11
    - Creating a New User Pool Using the AWS CLI 11
    - Creating a New User Pool Using the API 11
    - Console Quickstart 12
    - Step Through Settings 13
  - Setting up the Mobile SDK for Android 30
    - Gradle Dependencies 30
    - Network Permissions 30
    - Using the Mobile SDK for Android in Your Amazon Cognito Application 30
    - Tutorial: Integrating User Pools for Android Apps 31
    - Examples: Using the Mobile SDK for Android 37
    - Example: Handling Users Created Using the Android AdminCreateUser API 43
  - Setting up the Mobile SDK for iOS 44
    - Installing the AWS Mobile SDK for iOS 44
    - Tutorial: Integrating User Pools for iOS Apps 45
    - Examples: Using the iOS SDK 48
  - Setting up the Amazon Cognito Identity SDK for JavaScript 54
    - Tutorial: Integrating User Pools for JavaScript Apps 54
    - Examples: Using the JavaScript SDK 58
    - Example: Handling Users Created Using the JavaScript AdminCreateUser API 66
  - Using Lambda Triggers 67
    - Creating an AWS Lambda Trigger for a Stage 68
    - AWS Lambda Trigger Request and Response Parameters 68
    - AWS Lambda Trigger Examples 76
  - Integrating Apps 83
    - Specifying App Identity Provider Settings 83
    - Assigning a Domain 86
    - Specifying App UI Customization Settings 87
    - Defining Resource Servers 90
  - Using Federation 93
    - Adding Social Identity Providers 93

        - Creating SAML Providers ... 95
        - Integrating Third-Party SAML Providers ... 100
        - Specifying Attribute Mappings ... 101
    - Creating Users as Administrator ... 103
        - Authentication Flow for Users Created by Administrators or Developers ... 104
        - Creating a New User in the AWS Management Console ... 104
    - User Groups ... 106
        - Assigning IAM Roles to Groups ... 106
        - Assigning Precedence Values to Groups ... 107
        - Using Groups to Control Permission with Amazon API Gateway ... 107
        - Limitations on Groups ... 107
        - Creating a New Group in the AWS Management Console ... 107
    - Importing Users ... 108
        - Creating the CloudWatch Logs IAM Role ... 108
        - Creating the User Import .csv File ... 110
        - Creating and Running the User Import Job ... 112
        - Viewing Import Job Results ... 116
        - Requiring Imported Users to Reset Their Passwords ... 117
    - Signing Up and Confirming User Accounts ... 117
        - Overview of User Account Confirmation ... 117
        - Allowing Users to Sign Up and Confirm Themselves and Verify Email or Phone ... 118
        - Allowing Users to Sign Up in Your App but Confirming Them as Administrator ... 119
        - Computing SecretHash Values ... 119
        - Confirming User Accounts Without Verifying Email or Phone Number ... 120
        - Verifying When Users Change Their Email or Phone Number ... 120
        - Confirmation and Verification Processes for User Accounts Created by Administrators or Developers ... 121
        - Confirmation and Verification Processes for Imported User Accounts ... 121
    - Managing and Searching for Users ... 121
        - Viewing User Attributes ... 121
        - Searching User Attributes ... 122
        - Searching for Users Using the AWS Management Console ... 122
        - Searching for Users Using the `ListUsers` API ... 123
        - Examples of Using the `ListUsers` API ... 124
    - Authentication Flow ... 125
        - Custom Authentication Flow ... 125
        - Admin Authentication Flow ... 126
    - Integrating User Pools with Federated Identities ... 126
        - Setting Up a User Pool ... 126
        - Configuring Your Identity Pool Using the AWS Management Console ... 127
        - Using Amazon Cognito User Pools ... 127
    - Using Tokens ... 128
        - Using the ID Token ... 129
        - Using the Access Token ... 129
        - Using the Refresh Token ... 129
        - Structure of ID Tokens ... 129
        - Structure of Access Tokens ... 130
        - Using ID Tokens and Access Tokens in your Web APIs ... 131
        - Revoking All Tokens for a User ... 131
- Amazon Cognito Federated Identities ... 133
    - Getting Started with Amazon Cognito Federated Identities ... 134
        - Sign Up for an AWS Account ... 134
        - Create an Identity Pool in Amazon Cognito ... 134
        - Install the Mobile or JavaScript SDK ... 135
        - Integrate the Identity Providers ... 135
        - Get Credentials ... 135
    - Identity Pools ... 135

- Authenticated and Unauthenticated Identities ... 136
- User IAM Roles ... 136
- Federated Identities Concepts ... 136
  - Authentication Flow ... 136
  - IAM Roles ... 141
  - Role Trust and Permissions ... 144
- Role-Based Access Control ... 145
  - Creating Roles for Role Mapping ... 145
  - Granting Pass Role Permission ... 145
  - Using Tokens to Assign Roles to Users ... 146
  - Using Rule-Based Mapping to Assign Roles to Users ... 147
  - Token Claims to Use in Rule-Based Mapping ... 147
  - Best Practices for Role-Based Access Control ... 148
- Getting Credentials ... 148
  - Android ... 148
  - iOS - Objective-C ... 149
  - iOS - Swift ... 150
  - JavaScript ... 151
  - Unity ... 152
  - Xamarin ... 153
- Accessing AWS Services ... 153
  - Android ... 153
  - iOS - Objective-C ... 154
  - iOS - Swift ... 154
  - JavaScript ... 154
  - Unity ... 154
  - Xamarin ... 154
- External Identity Providers ... 155
  - Facebook ... 155
  - Login with Amazon ... 160
  - Google ... 163
  - Twitter/Digits ... 169
  - Open ID Connect Providers ... 173
  - SAML Identity Provider ... 175
- Developer Authenticated Identities ... 177
  - Understanding the Authentication Flow ... 177
  - Define a Developer Provider Name and Associate it with an Identity Pool ... 177
  - Implement an Identity Provider ... 177
  - Updating the Logins Map (Android and iOS only) ... 183
  - Getting a Token (Server Side) ... 183
  - Connect to an Existing Social Identity ... 184
  - Supporting Transition Between Providers ... 185
- Switching Identities ... 187
  - Android ... 187
  - iOS - Objective-C ... 188
  - iOS - Swift ... 188
  - JavaScript ... 188
  - Unity ... 188
  - Xamarin ... 189

Amazon Cognito Sync ... 190
- Getting Started with Amazon Cognito Sync ... 190
  - Sign Up for an AWS Account ... 190
  - Set Up an Identity Pool in Amazon Cognito ... 191
  - Store and Sync Data ... 191
- Synchronizing Data ... 191
  - Initializing the Amazon Cognito Sync Client ... 191
  - Understanding Datasets ... 193

        - Reading and Writing Data in Datasets 194
        - Synchronizing Local Data with the Sync Store 196
    - Handling Callbacks 198
        - Android 198
        - iOS - Objective-C 199
        - iOS - Swift 202
        - JavaScript 204
        - Unity 206
        - Xamarin 208
    - Push Sync 209
        - Create an Amazon Simple Notification Service (Amazon SNS) App 210
        - Enable Push Sync in the Amazon Cognito console 210
        - Use Push Sync in Your App: Android 210
        - Use Push Sync in Your App: iOS - Objective-C 212
        - Use Push Sync in Your App: iOS - Swift 213
    - Amazon Cognito Streams 215
    - Amazon Cognito Events 217
- Logging Amazon Cognito API Calls with AWS CloudTrail 220
    - Amazon Cognito Information in CloudTrail 220
    - Understanding Amazon Cognito Log File Entries 221
- Limits 223
- Resource Permissions 226
    - Amazon Resource Names (ARNs) 226
    - Example Policies 227
    - Managed Policies 228
    - Signed versus Unsigned APIs 228
- Using the Amazon Cognito Console 230
    - What is the Amazon Cognito Console? 230
    - Create an Identity Pool 230
    - Delete an Identity Pool 230
    - Delete an Identity from an Identity Pool 231
    - Enable or edit authentication providers 231
    - Change the role associated with an identity type 231
    - Enable or disable unauthenticated identities 232
    - Managing Datasets in the Amazon Cognito Console 232
    - Create a Dataset for an Identity 232
    - Delete a Dataset Associated with an Identity 233
    - Set Up Amazon Cognito Streams 233
    - Bulk Publish Data 233
    - Enable Push Synchronization 233
    - Set Up Amazon Cognito Events 234
- Amazon Cognito API Reference 235
    - Amazon Cognito Auth API Reference 235
        - AUTHORIZATION Endpoint 235
        - TOKEN Endpoint 239
        - LOGOUT Endpoint 242
    - Cognito User Pools API Reference 243
    - Cognito Federated Identities API Reference 243
    - Cognito Sync API Reference 243
- Document History 244
- AWS Glossary 247

What Is Amazon Cognito?

Amazon Cognito lets you easily add user sign-up and sign-in and manage permissions for your mobile and web apps. You can create your own user directory within Amazon Cognito. You can also choose to authenticate users through social identity providers such as Facebook, Twitter, or Amazon; with SAML identity solutions; or by using your own identity system. In addition, Amazon Cognito enables you to save data locally on users' devices, allowing your applications to work even when the devices are offline. You can then synchronize data across users' devices so that their app experience remains consistent regardless of the device they use.

With Amazon Cognito, you can focus on creating great app experiences instead of worrying about building, securing, and scaling a solution to handle user management, authentication, and synchronization across devices.

Topics

- What's New (p. 1)
- Features of Amazon Cognito (p. 1)
- Accessing Amazon Cognito (p. 2)
- Are You a First-Time Amazon Cognito User? (p. 2)
- Pricing for Amazon Cognito (p. 2)
- SDKs for Amazon Cognito (p. 3)
- SDKs for Amazon Cognito User Pool App Integration and Federation (p. 3)

What's New

Amazon Cognito User Pools now has a built-in, customizable UI to sign in users and provides built-in federation with Facebook, Google, Login with Amazon, and SAML identity providers. With these new features, you can easily integrate user sign-in into your app and offer your users multiple options for signing in. For more information, see Getting Started with User Pools App Integration and Federation (p. 4).

Features of Amazon Cognito

Amazon Cognito User Pools: You can create and maintain a user directory and add sign-up and sign-in to your mobile app or web application using Amazon Cognito User Pools. You can also sign in users to a user pool through social identity providers such as Google, Facebook, and Amazon, and through SAML-based identity providers. User pools scale to hundreds of millions of users and provide simple, secure, and low-cost options for you as a developer. You can implement enhanced security features, such as email and phone number verification, and multi-factor authentication. In addition, Amazon Cognito User Pools lets you customize workflows through AWS Lambda; for example, by adding app-specific logic to user registration for fraud detection and user validation.

Amazon Cognito is HIPAA-eligible and PCI-compliant.

For more information, see Amazon Cognito User Pools (p. 9).

Amazon Cognito Federated Identities: Amazon Cognito Federated Identities enable you to create unique identities for your users and authenticate them with federated identity providers. With a federated identity, you can obtain temporary, limited-privilege AWS credentials to synchronize data with Amazon Cognito Sync. You can also use these credentials to securely access other AWS services such as Amazon DynamoDB, Amazon S3, and Amazon API Gateway. Amazon Cognito Federated Identities support federated identity providers—including Amazon, Facebook, Google, Twitter, OpenID Connect providers, and SAML identity providers—as well as unauthenticated identities. This feature also supports developer authenticated identities, which let you register and authenticate users via your own backend authentication systems.

For more information, see Amazon Cognito Federated Identities (p. 133).

Amazon Cognito Sync: Amazon Cognito Sync is an AWS service that supports offline access and cross-device syncing of application-related user data. You can use Amazon Cognito Sync to synchronize user profile data across mobile devices and the web without requiring your own backend. The client libraries cache data locally so your app can read and write data regardless of device connectivity status. When the device is online, you can synchronize data, and, if you set up push synchronization, notify other devices immediately that an update is available.

For more information, see Amazon Cognito Sync (p. 190).

Accessing Amazon Cognito

Amazon Cognito can be accessed using the Amazon Cognito console, the AWS Command Line Interface, and the Amazon Cognito APIs.

- The Amazon Cognito User Pool APIs are documented in the User Pools API Reference.
- The Amazon Cognito Federated Identities APIs are documented in the Amazon Cognito Identity API Reference.
- The Amazon Cognito Sync APIs are documented in the Amazon Cognito Sync APIs.

Are You a First-Time Amazon Cognito User?

If you are a first-time user of Amazon Cognito, we recommend that you begin by reading the Getting Started with User Pools App Integration and Federation (p. 4) guide.

You can also find information and links to videos, articles, documentation, and sample apps on our Developer Resources Page.

Pricing for Amazon Cognito

For information on Amazon Cognito pricing, see the Amazon Cognito Pricing Page.

SDKs for Amazon Cognito

There are three types of SDKs for Amazon Cognito:

- Dedicated SDKs and sample apps for integrating the Amazon Cognito User Pools hosted UI with your app. For information, see SDKs for Amazon Cognito User Pool App Integration and Federation (p. 3).
- Higher-level client SDKs for iOS, Android, and JavaScript.
- Standard AWS SDKs, which cover a wider array of languages, including Java, C#, and Ruby.

The standard SDKs cover all of the APIs of the service, while the higher-level SDKs provide additional features that make it easier to perform some functions. One of the key differences between the two types of SDKs is in signing in users. Amazon Cognito uses a Secure Remote Password (SRP) protocol, which requires some calculations and a couple of requests between the client and the service APIs. In the higher-level SDKs, that process is handled for you. The standard SDKs expose the underlying APIs, but they currently do not include built-in support for SRP. For more information about authentication options, see Amazon Cognito User Pool Authentication Flow (p. 125).

Use the following links for the SDKs for Amazon Cognito.

Higher-Level Client SDKs

- Amazon Cognito SDK for iOS
- Amazon Cognito SDK for Android
- Amazon Cognito SDK for JavaScript

Standard AWS SDKs

You can download and find links for documentation for all of the SDKs at Tools for Amazon Web Services.

SDKs for Amazon Cognito User Pool App Integration and Federation

In addition to the Amazon Cognito SDKs, we have dedicated SDKs and sample apps for Amazon Cognito User Pool app integration and federation. You can find them in the following locations:

- **Android**
  - Amazon Cognito Auth SDK for Android
  - Android sample app
- **iOS**
  - The Amazon Cognito Auth SDK for iOS is included in the Mobile SDK for iOS. Make sure to download version 2.5.8 or later.
  - iOS sample app
- **JavaScript**
  - Amazon Cognito Auth SDK for JavaScript (including JavaScript sample app)

Getting Started with User Pools App Integration and Federation

Amazon Cognito User Pools app integration and federation provide a customizable experience to sign in users and built-in integrations with Facebook, Google, and Login with Amazon as well as SAML 2.0 identity providers. With the Amazon Cognito SDK and a few lines of code, you can add sign-up and sign-in pages to your mobile or web app. With these new features, Amazon Cognito User Pools can authenticate and manage both sets of users: those who sign in directly to your user pool and users who sign in through an external identity provider. All users have profiles and tokens provided by Amazon Cognito.

This guide describes how to get started with Amazon Cognito User Pools using the customizable user experience and built-in integrations with identity providers. If you prefer to build your own user experience and connect it to Amazon Cognito User Pools using our SDK, see the Getting Started with Amazon Cognito User Pools (p. 10). If your app needs to get AWS credentials for federated users, but you do not want to link federated users with Amazon Cognito User Pool profiles or tokens, see Getting Started with Amazon Cognito Federated Identities (p. 134).

Setting Up an Amazon Cognito User Pool as Your User Directory

A user pool is a user directory that you can use to sign up and sign in users and to manage user profiles. User pools also provide tokens for your users when they sign in. You can use these tokens to control access for the user (via your app) to resources such as backend APIs. User pools can contain both native users who sign in directly (i.e., with a username and password stored in the user pool) and federated users who sign in via an external identity provider. You can also map external identity provider user attributes (e.g., name or email address) to user pool attribute values. Both native and federated users have a user profile and receive user pool tokens when they sign in, so you can standardize your app to handle all users through Amazon Cognito.

If you do not already have a user pool, you can create and configure one from the Amazon Cognito console. For more information, see Quickstart: Using the Console to Create a New User Pool (p. 12) and Step Through Amazon Cognito User Pool Settings in the AWS Management Console (p. 13).

To try the app integration and federation features, we recommend you create a user pool with the following settings in the Amazon Cognito console:

- On the **Attributes** tab, select **Email address or phone number** and select **Allow email addresses**.

- On the **Policies** tab, select **Allow users to sign themselves up** (default).
- On the **Verifications** tab, under **Do you want to require verification of emails or phone numbers?**, select **Email** (default).
- On the **App clients** tab, create an app client.

Configuring App Integration

After you create a user pool, the Amazon Cognito console displays an **App integration** tab where you can configure settings for the customizable, built-in UI for signing up and signing in users. For more information, see Integrating Mobile and Web Apps into Amazon Cognito User Pools (p. 83).

To use app integration, specify the following settings in the Amazon Cognito console:

1. On the **App client settings** tab:
    1. Select the appropriate boxes to enable the identity providers you want to allow your users to sign in with.

        Note
        To allow your users to sign in with external identity providers such as Facebook or a SAML identity provider, you first configure them as described next and then return to the **App client settings** tab to enable them.
    2. Enter a callback URL for the Amazon Cognito authorization server to call after users are authenticated. For a web app, the URL must start with `https://`. For an iOS or Android app, you can use a callback URL such as `myapp://`. You may want to return to this setting after you've integrated the SDK, as described in the following section.
    3. Unless you specifically want to exclude one, select the boxes for all of the **Allowed OAuth Flows** and **Allowed OAuth scopes**.
2. On the **Domain name** tab, enter a domain prefix that is available.
3. On the **UI customization** tab, you can upload a logo for the hosted end-user pages and edit the CSS values to change the look and feel to match your app and branding.

    Note
    You can view the hosted UI with your customizations by constructing the following URL, with the specifics for your user pool, and typing it into a browser: `https://<your_domain>/login?response_type=code&client_id=<your_app_client_id>&redirect_uri=<your_callback_url>`
    You may have to wait up to one minute to refresh your browser before changes made in the console appear.
    Your domain is shown on the **Domain name** tab. Your app client ID and callback URL are shown on the **App client settings** tab.

Configuring Federation with a Social Identity Provider

Amazon Cognito User Pools provide built-in integrations with social identity providers, such as Facebook, Google, and Login with Amazon. By configuring these social identity providers for an Amazon Cognito user pool, you can quickly and easily add them as choices for your end users to sign in. The user pool becomes a single point of identity management for your application. You can add one or more social identity providers in the Amazon Cognito console and define mappings of user attributes (such as email addresses) from social identity providers to user attributes in your user pool.

You can skip this step if you do not want to enable your end users to sign in through social identity.

For more information, see Adding Social Identity Providers (p. 93). To try this feature, make the following choices in the Amazon Cognito console:

To add a social identity provider, you first create a developer account with the identity provider. After you have your developer account, you register your app with the identity provider. The identity provider creates an app ID and an app secret for your app, and you need to configure those values in your user pool.

Here are some links to get started with social identity providers:

- Google Identity Platform
- Facebook for Developers
- Login with Amazon

You will need to configure your user pool domain or redirect URL with the identity provider. This ensures that the identity provider will accept the redirect URL supplied by Amazon Cognito when it authenticates users.

- For Google, add your Amazon Cognito user pool domain URL (`https://<your-user-pool-domain>/oauth2/idpresponse`) in the Google app's **Authorized redirect URIs** (in the **Credentials** section).
- For Facebook, add your Amazon Cognito user pool domain URL (`https://<your-user-pool-domain>/`) in the Facebook app's **Settings (Basic)**, **Website URL**.
- For Login with Amazon, add your Amazon Cognito user pool domain URL (`https://<your-user-pool-domain>/oauth2/idpresponse`) to the Login with Amazon app's **Allowed Return URLs**.

To configure a social identity provider in Amazon Cognito User Pools

1. On the **Identity providers** tab, choose the button for your social identity provider: **Facebook**, **Google**, or **Login with Amazon**.
2. Enter the app ID and app secret that you received from the identity provider.
3. Enter the names of the scopes that you want to authorize. Scopes define which user attributes (such as `name` and `email`) you want to access with your app. For Facebook, these should be separated by commas (for example, `public_profile, email`). For Google and Login with Amazon, they should be separated by spaces. (Google example: `profile email openid`. Login with Amazon example: `profile postal_code`.)

   The end-user is asked to consent to providing these attributes to your app. For more information about their scopes, see the documentation from Google, Facebook, and Login with Amazon.
4. Choose **Update Facebook** (or **Google** or **Amazon**).
5. On the **Attribute mapping** tab, add mappings for at least the required attributes, typically `email`, as follows:

   1. Select the check box to choose the Facebook, Google, or Amazon attribute name. You can also enter the names of additional attributes that are not listed in the Amazon Cognito console.
   2. Select the destination user pool attribute from the drop-down list.

Configuring Federation with a SAML 2.0 Identity Provider

Amazon Cognito User Pools support SAML 2.0 federation with post-binding endpoints. This eliminates the need for your app to retrieve or parse SAML assertion responses, because the user pool directly receives the SAML response from your identity provider via a user agent. Amazon Cognito acts as an authentication service provider on behalf of your application. The user pool becomes a single point of identity management for your application, so that your application does not need to integrate with multiple SAML identity providers. You can add one or more SAML identity providers in the Amazon Cognito console and define mappings of user attributes (such as email addresses) from the SAML identity provider (via SAML assertion claims) to user attributes in your user pool.

You can skip this step if you do not want to enable your end users to sign in via SAML federation. To set up a SAML identity provider, you will do configuration tasks both in Amazon Cognito and in the SAML identity provider.

You configure your user pool as a relying party or application in your SAML 2.0 identity provider. See the documentation for your identity provider for more information.

You enter a redirect or sign-in URL, which is `https://<yourDomainPrefix>.auth.<region>.amazoncognito.com/saml2/idpresponse`. You can find your domain prefix and the region value for your user pool on the **Domain name** tab of the Amazon Cognito console.

> **Note**
> Any SAML identity providers that you created in a user pool during the public beta before August 10, 2017 have redirect URLs of `https://<yourDomainPrefix>.auth.<region>.amazoncognito.com/login/redirect`. If you have one of these SAML identity providers from the public beta in your user pool, you must either:
>
> - Replace it with a new one that uses the new redirect URL.
> - Update the configuration in your SAML identity provider to accept both the old and new redirect URLs.
>
> All SAML identity providers in Amazon Cognito will switch to the new URLs, and the old ones will stop working on October 31, 2017.

For some SAML Identity providers you must provide the `urn` / Audience URI / SP Entity ID, in the form `urn:amazon:cognito:sp:<yourUserPoolID>`. You can find your user pool ID on the **App client settings** tab in the Amazon Cognito console.

You must also configure your SAML identity provider to provide attributes values for any attributes required in your user pool. Typically `email` is a required attribute for user pools, and in that case the SAML identity provider must provide an `email` value (claim) in the SAML assertion.

To configure a SAML 2.0 identity provider in Amazon Cognito User Pools

For more information, see Using Federation for Amazon Cognito User Pools (p. 93) . To try this feature, make the following choices in the Amazon Cognito console:

1. On the **Identity providers** tab, create a new provider by uploading or entering a URL for the metadata document from your SAML identity provider. For more information about the metadata document, see Using Federation for Amazon Cognito User Pools (p. 93).
2. On the **Attribute mapping** tab, add mappings for at least the required attributes, typically `email`, as follows:
    1. Enter the SAML attribute name as it appears in the SAML assertion from your identity provider. If your identity provider offers sample SAML assertions, that may help you to find the name. Some identity providers use simple names, such as `email`, while others use names similar to `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`.
    2. Select the destination user pool attribute from the drop-down list.

Integrating the Sign-in Flow with Your App

You can use the Amazon Cognito iOS, Android, and JavaScript SDKs to integrate the customizable sign-in flows into your mobile or web app. The SDKs help you display the hosted UI, receive the user pool tokens, and refresh tokens. For more information, see the documentation with each SDK.

Your app can also form requests and call the Amazon Cognito authorization server directly. The following example shows the hosted sign-in page and tests signing in from a browser with an authorize request:

```
https://<yourDomainPrefix>.auth.<region>.amazoncognito.com/authorize?
response_type=code&
client_id=<yourAppClientId>&
redirect_uri=<yourRedirectURL>&
state=STATE&
scope=email+phone+openid+profile+aws.cognito.signin.user.admin
```

For more information, see the Amazon Cognito Auth API Reference (p. 235).

Where to Find the SDKs and Sample Apps for App Integration and Federation

You can find the SDKs and sample apps for Amazon Cognito User Pools app integration and federation in the following locations:

- **Android**
  - Amazon Cognito Auth SDK for Android
  - Android sample app
- **iOS**
  - The Amazon Cognito Auth SDK for iOS is included in the Mobile SDK for iOS. Make sure to download version 2.5.8 or later.
  - iOS sample app
- **JavaScript**
  - Amazon Cognito Auth SDK for JavaScript (including JavaScript sample app)

Where to Go from Here

With the preceding instructions and the SDKs you can add user sign-up and sign-in, with social identity or SAML federation, to your mobile or web app with Amazon Cognito User Pools. After a user signs in, your app can use the user pool profile to manage users and the tokens the user pool provides to authorize access to your APIs and resources. If your app needs to use these user pool tokens to get AWS credentials to access resources, such as in an S3 bucket or DynamoDB table, see Integrating User Pools with Federated Identities (p. 126). For information about using Amazon Cognito user pool tokens directly with API Gateway, see Use Amazon Cognito User Pools in the *API Gateway Developer Guide*.

Amazon Cognito User Pools

Create and maintain a user directory and add sign-up and sign-in to your mobile app or web application using user pools. User pools scale to hundreds of millions of users and are designed to provide simple, secure, and low-cost options for you as a developer.

You can use user pools to add user registration and sign-in features to your apps. Instead of using external identity providers (p. 155) such as Facebook, Google, or Twitter, you can use user pools to let users register with or sign in to an app using an email address, phone number, or a user name. You can also create custom registration fields and store that metadata in your user directory. You can verify email addresses and phone numbers, recover passwords, and enable multi-factor authentication (MFA) with just a few lines of code.

User pools are for mobile and web app developers who want to handle user registration and sign-in directly in their apps. Previously, you needed to implement your own user directory to create user accounts, store user profiles, and implement password recovery flows to support user registration and sign-in.

User pools integrate easily with the existing Amazon Cognito functionality for anonymous and social identities. In addition, a user can start as an anonymous user and then either sign in using a social identity or using user pools to register and sign in using email, phone number, or user name.

Amazon Cognito user pools are PCI- and HIPAA-compliant.

You can get started with user pools by using the AWS Management Console, the AWS Command Line Interface, or the APIs provided in one of our SDKs. For more information, see Setting up User Pools (p. 11).

To learn more about user pool settings, such as attributes, policies, multi-factor authentication, and triggers, see Step Through Amazon Cognito User Pool Settings in the AWS Management Console (p. 13).

Topics

- Getting Started with Amazon Cognito User Pools (p. 10)
- Setting up User Pools (p. 11)
- Setting Up the AWS Mobile SDK for Android to Work with User Pools (p. 30)
- Setting Up the AWS Mobile SDK for iOS to Work with User Pools (p. 44)
- Setting up the Amazon Cognito Identity SDK for JavaScript to Work with User Pools (p. 54)
- Customizing User Pool Workflows by Using AWS Lambda Triggers (p. 67)
- Integrating Mobile and Web Apps into Amazon Cognito User Pools (p. 83)

- Using Federation for Amazon Cognito User Pools (p. 93)
- Creating User Accounts as Administrator in the AWS Management Console and with the Amazon Cognito User Pools API (p. 103)
- User Groups (p. 106)
- Importing Users into User Pools (p. 108)
- Signing Up and Confirming User Accounts (p. 117)
- Managing and Searching for User Accounts in the AWS Management Console and in the Amazon Cognito User Pools API (p. 121)
- Amazon Cognito User Pool Authentication Flow (p. 125)
- Integrating User Pools with Federated Identities (p. 126)
- Using Tokens with User Pools (p. 128)

Getting Started with Amazon Cognito User Pools

With Amazon Cognito user pools, you can create and maintain a user directory and add sign-up and sign-in to your mobile app or web application.

Topics

- Sign Up for an AWS Account (p. 10)
- Creating a User Pool in Amazon Cognito (p. 10)
- Install the SDK (p. 10)
- Integrate Your User Pool into Your App (p. 11)

Sign Up for an AWS Account

To use Amazon Cognito user pools, you need an AWS account. If you don't already have one, use the following procedure to sign up:

To sign up for an AWS account

1. Open https://aws.amazon.com/, and then choose **Create an AWS Account**.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

Creating a User Pool in Amazon Cognito

You can quickly create a user pool through the Amazon Cognito console, or you can use the AWS Command Line Interface (CLI) or the Amazon Cognito APIs. The simplest way is to sign in to the Amazon Cognito console and choose **Manage your User Pools**. You can find more information about these options in the Setting up User Pools (p. 11) page.

Install the SDK

There are two types of SDKs for Amazon Cognito user pools. We have higher-level client SDKs for iOS, Android, and JavaScript, and the standard AWS SDKs that cover a wider array of languages including Java, C#, and Ruby. The standard SDKs cover all of the APIs of the service, while the higher-level SDKs provide additional features that make it easier to perform some functions. One of the key differences between the two types of SDKs is in signing in users. Amazon Cognito uses a Secure Remote Password (SRP) protocol, which requires some calculations and a couple of requests between the client and the service APIs. In the higher-level SDKs, that process is taken care of for you. The standard SDKs expose the underlying APIs, but they currently do not include built-in support for SRP. To learn more about authentication options, see the Amazon Cognito User Pool Authentication Flow (p. 125) page.

Follow the links below to the SDKs for Cognito user pools.

Higher-Level Client SDKs

- Amazon Cognito SDK for Android
- Amazon Cognito SDK for iOS
- Amazon Cognito SDK for JavaScript

Standard AWS SDKs

You can download all the SDKs and find documentation here.

Integrate Your User Pool into Your App

Now that you have created a user pool and installed the SDK, you need to integrate your app with your user pool by creating your user interface and connecting it to the user pool through the SDK. We have example apps for iOS, Android, and JavaScript to help you get started. See our Developer Resources page for more information.

Setting up User Pools

To create a new user pool for Amazon Cognito, you can use the AWS Management Console, the AWS CLI, or the Amazon Cognito API.

Creating a New User Pool Using the Console

You can create a new user pool by choosing **Create new pool** from the Amazon Cognito console and following the instructions. For a step-by-step walkthrough, see Quickstart: Using the Console to Create a New User Pool (p. 12).

Creating a New User Pool Using the AWS CLI

You can create a new user pool using the create-user-pool command in the AWS CLI.

For more information, see the Amazon Cognito Identity AWS CLI Reference.

Creating a New User Pool Using the API

You can create a new user pool using the `CreateUserPool()` API. For more information, see CreateUserPool.

You can also use one of the following SDKs to create and manage new user pools:

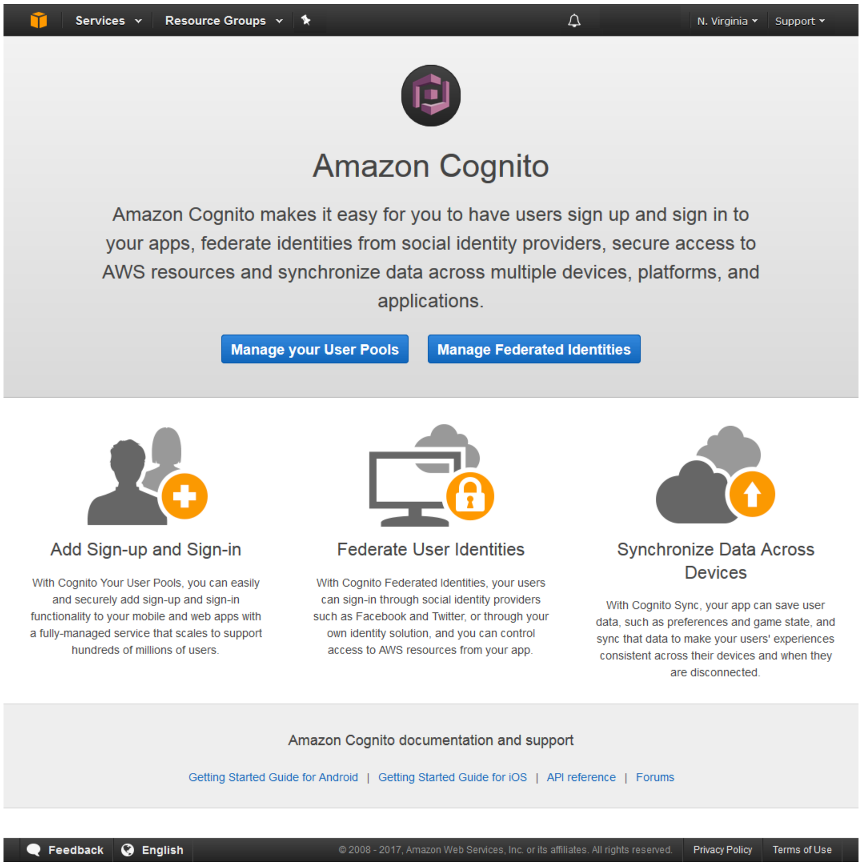
- Setting Up the AWS Mobile SDK for Android to Work with User Pools (p. 30)
- Setting Up the AWS Mobile SDK for iOS to Work with User Pools (p. 44)
- Setting up the Amazon Cognito Identity SDK for JavaScript to Work with User Pools (p. 54)

Quickstart: Using the Console to Create a New User Pool

The following procedure shows how to create a new user pool using the AWS Management Console.

To create a new user pool using the AWS Management Console

1. Open the Amazon Cognito console.
2. On the Amazon Cognito home page, choose **Manage your User Pools**.


Services
Resource Groups
N. Virginia
Support
Amazon Cognito
Manage your User Pools
Manage Federated Identities
Add Sign-up and Sign-in
Federate User Identities
Synchronize Data Across Devices
Amazon Cognito documentation and support
Getting Started Guide for Android | Getting Started Guide for iOS | API reference | Forums
Feedback
English
© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

Amazon Cognito makes it easy for you to have users sign up and sign in to your apps, federate identities from social identity providers, secure access to AWS resources and synchronize data across multiple devices, platforms, and applications.

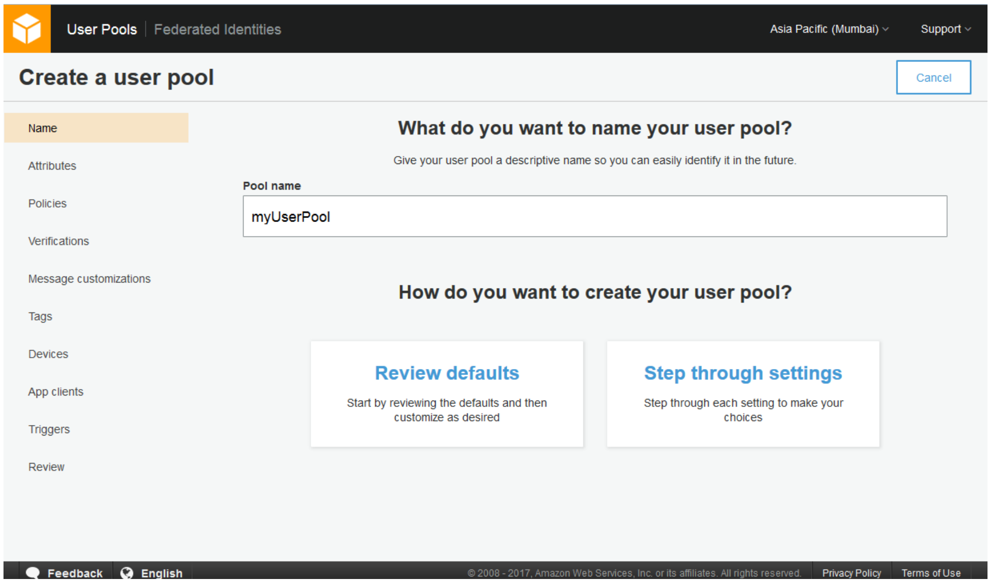
With Cognito Your User Pools, you can easily and securely add sign-up and sign-in functionality to your mobile and web apps with a fully-managed service that scales to support hundreds of millions of users.

With Cognito Federated Identities, your users can sign-in through social identity providers such as Facebook and Twitter, or through your own identity solution, and you can control access to AWS resources from your app.

With Cognito Sync, your app can save user data, such as preferences and game state, and sync that data to make your users' experiences consistent across their devices and when they are disconnected.

3. Choose **Create a User Pool** to get started.
4. Specify a **Pool name**.

   Pool names must be between one and 128 characters long. They can contain uppercase and lowercase letters (a-z, A-Z), numbers (0-9), and the following special characters: + = , . @ and -.
5. Next, decide how you want to create your user pool.


User Pools
Federated Identities
Asia Pacific (Mumbai)
Support
Cancel
Name
Attributes
Policies
Verifications
Message customizations
Tags
Devices
App clients
Triggers
Review
Feedback
English
© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

Create a user pool

What do you want to name your user pool?

Give your user pool a descriptive name so you can easily identify it in the future.

Pool name: myUserPool

How do you want to create your user pool?

Review defaults: Start by reviewing the defaults and then customize as desired

Step through settings: Step through each setting to make your choices

To create a user pool with the default settings, choose **Review defaults**, and then choose **Create pool**. You can still customize settings from the default values.

To step through each setting to make your choices, choose **Step through settings** and go to the next step. For more information, see Step Through Amazon Cognito User Pool Settings in the AWS Management Console (p. 13).

6. Review your user pool configuration in the next step, and then choose **Create pool**.

Step Through Amazon Cognito User Pool Settings in the AWS Management Console

You can customize the user pool settings to the needs of your app. This topic describes each category of settings and gives you detailed information about attributes, policies, email and phone verification, multi-factor authentication, apps, triggers, and trusted devices.

Topics

- Specifying a User Pool Name (p. 14)
- Importing and Creating Users and Groups (p. 14)
- Specifying User Pool Attribute Settings (p. 14)
- Specifying User Pool Policy Settings (p. 19)
- Specifying User Pool MFA Setting and Email and Phone Verification Settings (p. 20)
- Customizing SMS and Email Verification Messages and User Invitation Messages (p. 21)
- Adding Cost Allocation Tags to Your User Pool (p. 23)
- Specifying User Pool Device Tracking Settings (p. 24)
- Specifying User Pool App Settings (p. 24)
- Specifying User Pool Lambda Trigger Settings (p. 25)
- Reviewing Your User Pool Creation Settings (p. 26)
- Specifying App Client Settings for Your User Pool (p. 26)
- Specifying a Domain Name for Your User Pool (p. 26)
- Customizing the Built-in App UI to Sign Up and Sign In Users (p. 27)

- Specifying Resource Servers for Your User Pool (p. 27)
- Specifying Identity Providers for Your User Pool (p. 28)
- Specifying Attribute Mapping for Your User Pool (p. 29)

Specifying a User Pool Name

You must specify a **Pool Name** for your Amazon Cognito user pool in the AWS Management Console. The name cannot be changed after the user pool is created.

Pool names must be between one and 128 characters long. They can contain uppercase and lowercase letters (a-z, A-Z), numbers (0-9), and the following special characters: + = , . @ and -.

Importing and Creating Users and Groups

On the **Users and groups** tab, you can import users, create users, create groups and assign users to them, and search for users.

For more information, see:

- Importing Users into User Pools (p. 108)
- User Groups (p. 106)
- Creating User Accounts as Administrator in the AWS Management Console and with the Amazon Cognito User Pools API (p. 103)

Specifying User Pool Attribute Settings

You get a set of default attributes, called "standard attributes," with all user pools. You can also add custom attributes to your user pool definition in the AWS Management Console. This topic describes those attributes in detail and gives you tips on how to set up your user pool.

Attributes are pieces of information that help you identify individual users, such as name, email, and phone number.

Not all information about your users should be stored in attributes. For example, user data that changes frequently, such as usage statistics or game scores, should be kept in a separate data store, such as Amazon Cognito Sync or Amazon DynamoDB.

Standard Attributes

The following are the standard attributes for all users in a user pool. These are implemented following the OpenID Connect specification.

- `address`
- `birthdate`
- `email`
- `family_name`
- `gender`
- `given_name`
- `locale`
- `middle_name`
- `name`

- `nickname`
- `phone_number`
- `picture`
- `preferred_username`
- `profile`
- `timezone`
- `updated_at`
- `website`

These attributes are available as optional attributes for all users. To make an attribute required, select the check box next to the attribute.

> **Note**
> When a standard attribute is marked as required, a user cannot register unless a value for the attribute is provided. Administrators can create users without giving values for required attributes by using the AdminCreateUser API. An attribute cannot be switched between required and not required after a user pool has been created.

Custom attribute names are between one and 20 characters long.

Standard and custom attribute values can be any string up to 2048 characters by default, but some attribute values, such as `updated_at`, have formatting restrictions. Only **email** and **phone** can be verified.

> **Note**
> In the specification, attributes are called *members*.

Here are some additional notes regarding some of the above fields.

email

Email address values can be verified.

An administrator with proper AWS account permissions can change the user's email and also mark it as verified. This can be done by using the AdminUpdateUserAttributes API or the admin-update-user-attributes CLI command to change the `email_verified` attribute to `true`.

phone

A phone number is required if multi-factor authentication (MFA) is enabled. For more information, see Multi-factor Authentication (MFA) (p. 20).

Phone number values can be verified.

An administrator with proper AWS account permissions can change the user's phone number and also mark it as verified. This can be done by using the AdminUpdateUserAttributes API or the admin-update-user-attributes CLI command to change the `phone_verified` attribute to `true`.

> **Important**
> Phone numbers must follow these formatting rules: A phone number must start with a plus (**+**) sign, followed immediately by the country code. A phone number can only contain the **+** sign and digits. You must remove any other characters from a phone number, such as parentheses, spaces, or dashes (**-**) before submitting the value to the service. For example, a United States-based phone number must follow this format: **`+14325551212`**.

preferred_username

The `preferred_username` cannot be selected as both required and as an alias. If the `preferred_username` is an alias, a user can add the attribute value once he or she is confirmed by using the UpdateUserAttributes API.

To edit standard attributes

1. On the **Attributes** tab, choose the attributes you require for user registration. If an attribute is required and a user doesn't provide the required attribute, the user cannot register.

    Important
    You cannot change these requirements after the user pool is created. For more information, see Specifying User Pool Attribute Settings (p. 14).
2. To create an alias for email, phone number, address, or preferred username, choose **Alias**. For more information on aliases, see Overview of Aliases (p. 16).
3. Move on to create Custom Attributes (p. 18).

Usernames and Preferred Usernames

The `username` value is a separate attribute and not the same as the `name` attribute. A `username` is always required to register a user, and it cannot be changed after a user is created.

Developers can use the `preferred_username` attribute to give users a username that they can change. For more information, see Overview of Aliases (p. 16).

If your application does not require a username, you do not need to ask users to provide one. Your app can create a unique username for users in the background. This is useful if, for example, you want users to register and sign in with an email address and password. For more information, see Overview of Aliases (p. 16).

The `username` must be unique within a user pool. A `username` can be reused, but only after it has been deleted and is no longer in use.

Overview of Aliases

You can allow your end users to sign in with multiple identifiers by using aliases.

By default, users sign in with their username and password. The username is a fixed value that users cannot change. If you mark an attribute as an alias, users can sign in using that attribute in place of the username. The email address, phone number, and preferred username attributes can be marked as aliases.

For example, if email and phone are selected as aliases for a user pool, users in that user pool can sign in using their username, email address, or phone number, along with their password.

If email is selected as an alias, a username cannot match a valid email format. Similarly, if phone number is selected as an alias, a username that matches a valid phone number pattern is not accepted by the service for that user pool.

> **Note**
> Alias values must be unique in a user pool. If an alias is configured for an email address or phone number, the value provided can be in a verified state in only one account. During sign-up, if an email address or phone number is supplied as an alias from a different account that has already been used, registration succeeds. Nevertheless, when a user tries to confirm the account with this email (or phone number) and enters the valid code, an `AliasExistsException` error is thrown. The error indicates to the user that an account with this email (or phone number) already exists. At this point, the user can abandon the new account creation and can try to reset the password for the old account. If the user continues creating the new account, your app should call the `ConfirmSignUp` API with the `forceAliasCreation` option. This moves the

alias from the previous account to the newly created account, and it also marks the attribute unverified in the previous account.

Phone numbers and email addresses only become active aliases for a user after the phone numbers and email addresses are verified. We therefore recommend that you choose automatic verification of email addresses and phone numbers if you use them as aliases. The `preferred_username` attribute provides users the experience of changing their username, when in fact the actual username value for a user is not changeable.

If you want to enable this user experience, submit the new `username` value as a `preferred_username` and choose `preferred_username` as an alias. Then users can sign in with the new value they entered.

If `preferred_username` is selected as an alias, the value can be provided only when an account is confirmed. The value cannot be provided during registration.

Using Aliases to Simplify User Sign-Up and Sign-In

In the **Attributes** console tab, you can choose whether to allow your user to sign up with an email address or phone number as their username.

Topics

- Option 1: User Signs Up with Username and Signs In with Username or Alias (p. 17)
- Option 2: User Signs Up and Signs In with Email or Phone Number Instead of Username (p. 17)

Option 1: User Signs Up with Username and Signs In with Username or Alias

In this case, the user signs up with a username. In addition, you can optionally allow users to sign in with one or more of the following aliases:

- a verified email address
- a verified phone number
- a preferred username

These aliases can be changed after the user signs up.

Use the following steps to configure your user pool in the console to allow sign-in with an alias.

To configure a user pool for sign-in with an alias

1. In the **Attributes** tab, under **How do you want your end users to sign-up and sign-in?**, select **Username**.
2. Choose one of the following options:
   - **Also allow sign in with verified email address**: This allows users to sign in with their email address.
   - **Also allow sign in with verified phone number**: This allows users to sign in with their phone number.
   - **Also allow sign in with preferred username**: This allows users to sign in with a preferred username. This is a username that the user can change.

Option 2: User Signs Up and Signs In with Email or Phone Number Instead of Username

In this case, the user signs up with an email address or phone number as their username. You can choose whether to allow sign-up with only email addresses, only phone numbers, or either one.

The email or phone number must be unique, and it must not already be in use by another user. It does not have to be verified. After the user has signed up using an email or phone number, the user cannot create a new account with the same email or phone number; the user can only reuse the existing account (and reset the password if needed). However, the user can change the email or phone number to a new email or phone number; if it is not already in use, it becomes the new username.

> **Note**
> If users sign up with an email address as their username, they can change the username to another email address; they cannot change it to a phone number. If they sign up with a phone number, they can change the username to another phone number; they cannot change it to an email address.

Use the following steps to configure your user pool in the console to allow sign-up and sign-in with email or phone number.

To configure a user pool for sign-up and sign-in with email or phone number

1. In the **Attributes** tab, under **How do you want your end users to sign-up and sign-in?**, select **Email address or phone number**.
2. Choose one of the following options:
   - **Allow email addresses**: This allows your user to sign up with email as the username.
   - **Allow phone numbers**: This allows your user to sign up with phone number as the username.
   - **Allow both email addresses and phone number (users can choose one)**: This allows your user to use either an email address or a phone number as the username during sign-up.

> **Note**
> You do not need to mark email or phone number as required attributes for your user pool.

To implement option 2 in your app

1. Call the `CreateUserPool` API to create your user pool. Set the `UserNameAttributes` parameter to `phone_number`, `email`, or `phone_number | email`.
2. Call the `SignUp` API and pass an email address or phone number in the `username` parameter of the API. This API does the following:
   - If the `username` string is in valid email format, the user pool automatically populates the `email` attribute of the user with the `username` value.
   - If the `username` string is in valid phone number format, the user pool automatically populates the `phone_number` attribute of the user with the `username` value.
   - If the `username` string format is not in email or phone number format, the `SignUp` API throws an exception.
   - The `SignUp` API generates a persistent UUID for your user, and uses it as the immutable username attribute internally. This UUID has the same value as the `sub` claim in the user identity token.
   - If the `username` string contains an email address or phone number that is already in use, the `SignUp` API throws an exception.

You can use an email address or phone number as an alias in place of the username in all APIs except the `ListUsers` API. When you call `ListUsers`, you can search by the `email` or `phone_number` attribute; if you search by `username`, you must supply the actual username, not an alias.

Custom Attributes

You can add up to 25 custom attributes. These custom attributes can be defined as strings or numbers. You can specify a minimum and/or maximum length for the custom attributes. However, the maximum length can be no more than 2048 characters.

Custom attributes cannot be required.

Custom attribute names can be any string from one to 20 characters.

Custom attributes cannot be removed or changed once they are added to the user pool.

> **Note**
> In your code and in rules settings for Role-Based Access Control (p. 145), custom attributes require the `custom:` prefix to distinguish them from standard attributes.

To add a custom attribute

1. If you want to add custom attributes, open the **Do you want to use custom attributes?** section and choose **Add custom attribute**.
2. Provide properties for each custom attribute, such as the data **Type** (string or number), the **Name**, **Min length**, and **Max length**.
3. If you want to allow the user to change the value of a custom attribute after the value has been provided by the user, choose **Mutable**.
4. To add more attributes, choose **Add another attribute**.

Attribute Permissions and Scopes

You can set per-app read and write permissions for each user attribute. This gives you the ability to control which applications can see and/or modify each of the attributes that are stored for your users. For example, you could have a custom attribute that indicates whether a user is a paying customer or not. Your apps could see this attribute but could not modify it directly. Instead, you would update this attribute using an administrative tool or a background process. Permissions for user attributes can be set from the Amazon Cognito console, API, or CLI.

Attributes can be marked as readable or writable for each app. This is true for both standard and custom attributes. An app can read an attribute that is marked as readable and can write an attribute that is marked as writable. If an app tries to update an attribute that is not writable, the app gets a `NotAuthorizedException` exception. An app calling GetUser only receives the attributes that are readable for that app. The ID token issued post-authentication only contains claims corresponding to the readable attributes. Required attributes on a user pool are always writable. If you, using CLI or the admin API, set a writable attribute and do not provide required attributes, then an `InvalidParameterException` exception is thrown.

You can change attribute permissions and scopes after you have created your user pool.

Specifying User Pool Policy Settings

Password Policy

The following characters are allowed in passwords, regardless of the policy you set: uppercase and lowercase letters, numbers, and the special characters listed below.

You can specify the following password requirements in the AWS Management Console:

- **Minimum length**, which must be at least 6 characters but fewer than 99 characters
- **Require numbers**
- **Require special character**, which includes the following set:

  ```
  ^ $ * . [ ] { } ( ) ? - " ! @ # % & / \ , > < ' : ; | _ ~ `
  ```
- **Require uppercase letters**

- **Require lowercase letters**

Note
Specifying a minimum password length of at least 8 characters, as well as requiring uppercase, numeric, and special characters, increases password complexity for users in your user pool. The increased complexity helps protect users from the security risks of guessing attacks or common-pattern attacks. It is generally considered a best practice to require users to create strong passwords by using these options.

Admin Create User Policy

You can specify the following policies for Admin Create User:

- Specify whether to allow users to sign themselves up. This option is set by default. If it is not set, only administrators can create users in this pool and calls to the SignUp API fail with `NotAuthorizedException`.
- Specify the user account expiration time limit (in days) for new accounts. The default setting is 7 days, measured from the time when the user account is created. The maximum setting is 90 days. After the account expires, the user cannot log in to the account until the administrator updates the user's profile by updating an attribute or by resending the password to the user.

  Note
  Once the user has logged in, the account never expires.

Specifying User Pool MFA Setting and Email and Phone Verification Settings

Important
To ensure that SMS messages are sent to verify phone numbers and for MFA, you must request an increased spend limit from Amazon SNS.
Amazon Cognito uses Amazon SNS for sending SMS messages to users. The number of SMS messages Amazon SNS delivers is subject to spend limits. Spend limits can be specified for an AWS account and for individual messages, and the limits apply only to the cost of sending SMS messages.
The default spend limit per account (if not specified) is 1.00 USD per month. If you want to raise the limit, submit an SNS Limit Increase case in the AWS Support Center. For **New limit value**, enter your desired monthly spend limit. In the **Use Case Description** field, explain that you are requesting an SMS monthly spend limit increase.

In the **Verifications** tab, you can choose settings for multi-factor authentication (MFA) and for email and phone verification.

Note
SMS for MFA or for verifying phone numbers is charged separately. (There is no charge for sending verification codes to email addresses.) For information about Amazon SNS pricing, see Worldwide SMS Pricing. For the current list of countries where SMS messaging is available, see Supported Regions and Countries.

Multi-factor Authentication (MFA)

Multi-factor authentication (MFA) increases security for your app by requiring the user to receive and enter an authorization code when signing in to your app, in addition to their username (or alias) and password.

The following MFA settings are available:

- **Required**: All users must use MFA. This setting can only be specified when the user pool is created.

- **Optional**: Individual users can choose whether to enable MFA for their own user accounts.
- **Off**: MFA is disabled for all users.

When MFA is required on a user pool, you must mark either the email attribute or the phone number attribute as required.

When a user signs in with MFA turned on, he or she first enters and submits his or her username and password. The client app will receive a `getMFA` response indicating where the authorization code was sent. The client app should tell the user where to look for the code (such as which phone number the code was sent to), provide a form for entering the code, and then submit the code to complete the sign-in process. The destination is masked (e.g., only last 4 digits of the phone number are displayed).

The authorization code is valid for 3 minutes.

If a user no longer has access to his or her device where MFA codes are sent, he or she must request help from your customer service office. An administrator with necessary AWS account permissions can change the user's phone number, but only via the AWS Command Line Interface or the API.

When a user successfully goes through the MFA flow, his or her phone number is also marked as verified.

Requiring Email and Phone Number Verification

Verification requires users to retrieve a code from their email or phone to confirm ownership. Verification of a phone or email is necessary to automatically confirm users and enable recovery from forgotten passwords.

Amazon Cognito can automatically verify email addresses and mobile phone numbers by sending a verification code—or, in the case of email, a verification link. For email addresses, the code or link is sent in an email message. For phone numbers, the code is sent in an SMS text message.

The verification code or link is valid for 24 hours.

If verification is selected as required for email or phone, the verification code or link is automatically sent when a user signs up.

> **Note**
> The forgotten password flow requires either the user's email or the user's phone number to be verified.

> **Note**
> If a user signs up with both a phone number and an email address, and your user pool settings require verification of both attributes, a verification code is sent via SMS to the phone. The email address is not verified, so your app needs to call GetUser to see if an email address is awaiting verification. If it is, the app should call GetUserAttributeVerificationCode to initiate the email verification flow and then submit the verification code by calling VerifyUserAttribute.

Authorizing Amazon Cognito to Send SMS Messages on Your Behalf

To send SMS messages to your users on your behalf, Amazon Cognito needs your permission. To grant that permission, you create an AWS Identity and Access Management (IAM) role by choosing **Create role** to create the role.

Customizing SMS and Email Verification Messages and User Invitation Messages

In the **Message Customizations** tab, you can customize:

- Your SMS MFA message

- Your SMS and email verification messages
- The verification type for email—code or link
- Your user invitation messages
- From and Reply-To email addresses for emails going through your user pool

> **Note**
> The SMS and email verification message templates only appear if you have chosen to require phone number and email verification in the **Verifications** tab. Similarly, the SMS MFA message template only appears if the MFA setting is REQUIRED or OPTIONAL.

Customizing the SMS Message

You can customize the SMS message for MFA authentication by editing the template under the **Do you want to customize your SMS messages?** heading.

> **Important**
> Your custom message must contain the `{####}` placeholder, which is replaced with the authentication code before the message is sent.

The maximum length for the message is 140 UTF-8 characters, including the authentication code.

Customizing SMS Verification Messages

You can customize the SMS message for phone number verifications by editing the template under the **Do you want to customize your SMS verification messages?** heading.

> **Important**
> Your custom message must contain the `{####}` placeholder, which is replaced with the verification code before the message is sent.

The maximum length for the message is 140 UTF-8 characters, including the verification code.

Customizing Email Verification Messages

You can choose the verification type for email verifications: code or link.

You can customize the email subject and message for email address verifications by editing the template under the **Do you want to customize your email verification messages?** heading.

> **Important**
> If you have chosen code as the verification type, your custom message must contain the `{####}` placeholder, which is replaced with the verification code before the message is sent.

The maximum length for the message is 20,000 UTF-8 characters, including the verification code (if present). HTML tags can be used in these emails.

Customizing User Invitation Messages

You can customize the user invitation message that Amazon Cognito sends to new users via SMS or email by editing the templates under the **Do you want to customize your user invitation messages?** heading.

> **Important**
> Your custom message must contain the `{username}` and `{####}` placeholders, which are replaced with the user's username and password before the message is sent.

For SMS, the maximum length is 140 UTF-8 characters, including the verification code. For email, the maximum length for the message is 20,000 UTF-8 characters, including the verification code. HTML tags can be used in these emails.

Customizing Your Email Address

By default, the email messages that Amazon Cognito sends to users in your user pools come from **no-reply@verificationemail.com**. You can specify custom FROM email addresses and REPLY-TO email addresses to be used instead of **no-reply@verificationemail.com**.

To customize the FROM email address, choose **Add custom FROM address** and follow the instructions to verify your Amazon Simple Email Service identity. Choose an AWS Region and an Amazon SES verified identity. For more information, see Verifying Email Addresses and Domains in Amazon SES in the *Amazon Simple Email Service Developer Guide*.

To customize the REPLY-TO email address, choose **Add custom REPLY-TO address** and enter a valid email address.

Authorizing Amazon Cognito to Send Amazon SES Email on Your Behalf (from a Custom FROM Email Address)

If you want to send email from a custom FROM email address instead of the default, Amazon Cognito needs your permission to send email messages to your users on behalf of your Amazon SES verified identity. To grant that permission, create a sending authorization policy. For more information, see Using Sending Authorization with Amazon SES in the *Amazon Simple Email Service Developer Guide*.

The following is an example of an Amazon SES sending authorization policy for Amazon Cognito User Pools. For more examples, see Amazon SES Sending Authorization Policy Examples in the *Amazon Simple Email Service Developer Guide*.

> **Note**
> In this example, the "Sid" value is an arbitrary string that uniquely identifies the statement. For more information about policy syntax, see Amazon SES Sending Authorization Policies in the *Amazon Simple Email Service Developer Guide*.

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Sid": "stmnt1234567891234",
            "Effect": "Allow",
            "Principal": {
                "Service": "cognito-idp.amazonaws.com"
            },
            "Action": [
                "ses:SendEmail",
                "ses:SendRawEmail"
            ],
            "Resource": "<your SES identity ARN>"
        }
    ]
}
```

The Amazon Cognito console adds this policy for you when you select an Amazon SES identity from the drop-down menu. If you use the CLI or API to configure the user pool, you must attach this policy to your Amazon SES Identity.

Adding Cost Allocation Tags to Your User Pool

In the **Tags** tab, you can add cost allocation tags to categorize and track your AWS costs. When you apply tags to your AWS resources (such as Amazon Cognito user pools), your AWS cost allocation report includes usage and costs aggregated by tags. You can apply tags that represent business categories (such as cost centers, application names, and owners) to organize your costs across multiple services. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

To add a tag, choose **Add tag**. Specify a **Tag key** and **Tag value**, following the restrictions listed in Tag Restrictions. Choose **Save changes** to save your tag.

> **Important**
> In order for tags to appear on your billing reports, you must activate your applied tags in the billing console. For more information, see Activating User-Defined Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Specifying User Pool Device Tracking Settings

As a way of providing additional security, you can track devices that users have logged in to. This topic describes how to add device tracking to your Amazon Cognito user pool in the AWS Management Console.

Setting Up Remembered Devices

With Amazon Cognito user pools, you can choose to have Amazon Cognito remember devices used to access your application and associate these remembered devices with your application's users in a user pool. You can also choose to use remembered devices to stop sending codes to your users when you have set up multi-factor authentication (MFA).

When setting up the remembered devices functionality through the Amazon Cognito console, you have three options: **Always**, **User Opt-In**, and **No**.

- **No** (default) – Devices are not remembered.
- **Always** – Every device used by your application's users is remembered.
- **User Opt-In** – Your user's device is only remembered if that user opts to remember the device.

If either **Always** or **User Opt-In** is selected, a device identifier (key and secret) will be assigned to each device the first time a user signs in with that device. This key will not be used for anything other than identifying the device, but it will be tracked by the service.

If you select **Always**, Amazon Cognito will use the device identifier (key and secret) to authenticate the device on every user sign-in with that device as part of the user authentication flow.

If you select **User Opt-In**, you can remember devices only when your application's users opt to do so. When a user signs in with a new device, the response from the request to initiate tracking indicates whether the user should be prompted about remembering their device. You must create the user interface to prompt users. If the user opts to have the device remembered, the device status is updated with a 'remembered' state.

The AWS Mobile SDKs have additional APIs to see remembered devices (ListDevices, GetDevice), mark a device as remembered or not remembered (UpdateDeviceStatus), and stop tracking a device (ForgetDevice). In the REST API, there are additional administrator versions of these APIs that have elevated privileges and work on any user. They have API names such as AdminListDevices, AdminGetDevice, and so on. They are not exposed through the SDKs.

Using Remembered Devices to Suppress Multi Factor Authentication (MFA)

If you have selected either **Always** or **User Opt-In**, you also can suppress MFA challenges on remembered devices for the users of your application. To use this feature, you must enable MFA for your user pool. For more information, see Multi-factor Authentication (MFA) (p. 20).

Specifying User Pool App Settings

An app is an entity within a user pool that has permission to call unauthenticated APIs (APIs that do not have an authenticated user), such as APIs to register, sign in, and handle forgotten passwords. To call these APIs, you need an app client ID and an optional client secret. It is your responsibility to secure any app client IDs or secrets so that only authorized client apps can call these unauthenticated APIs.

You can create multiple apps for a user pool, and generally an app corresponds to the platform of an app. For example, you may create an app for a server-side application and a different Android app. Each app has its own app client ID.

When you create an app, you can optionally choose to create a secret for that app. If a secret is created for the app, the secret must be provided to use the app. Browser-based applications written in JavaScript may not need an app with a secret.

Secrets cannot be changed after an app is created. You can create a new app with a new secret if you want to rotate the secret that you are using. You can also delete an app to block access from apps that use that app client ID.

To create an app

1. On the **Apps** tab in **Create a user pool**, choose **Add an app client**.
2. Specify an **App client name**.
3. Specify the app's **Refresh token expiration (days)**. The default value is 30. You can change it to any value between 1 and 3650.
4. By default, user pools generate a client secret for your app. If you don't want that to happen, clear **Generate client secret**.
5. If your app is a server app that requires developer credentials (using Signature Version 4) and doesn't use Secure Remote Protocol (SRP) authentication, select **Enable sign-in API for server-based authentication (ADMIN_NO_SRP_AUTH)** to enable server-side authentication. For more information, see Admin Authentication Flow (p. 126).
6. By default, user pools allow your app to read and write all attributes. If you want to set different permissions for your app, perform the following steps.
    1. Choose **Set attribute read and write permissions**.
    2. You can set read and write permissions in both of the following ways:
        - By choosing one or more scopes. Each scope is a set of standard attributes. For more information, see the list of standard OIDC scopes.
        - By choosing individual standard or custom attributes.

    Note
    You cannot remove required attributes from write permissions in any app.
7. Choose **Create app client**.
8. If you want to create another app, choose **Add an app**.
9. Once you've created all the apps you want, choose **Save changes**.

You can also use the CLI commands, **create-user-pool-client** and **update-user-pool-client**, and the APIs, CreateUserPoolClient and UpdateUserPoolClient, to create and update app clients on a user pool.

Specifying User Pool Lambda Trigger Settings

You can use AWS Lambda triggers to customize workflows and the user experience with Amazon Cognito. You can create the following Lambda triggers: **Pre sign-up**, **Pre authentication**, **Custom message**, **Post authentication**, **Post confirmation**, **Define Auth Challenge**, **Create Auth Challenge**, and **Verify Auth Challenge Response**. For examples of each Lambda trigger, see Customizing User Pool Workflows by Using AWS Lambda Triggers (p. 67).

Note

The **Custom message** AWS Lambda trigger is an advanced way to customize messages for email and SMS. For more information, see Customizing User Pool Workflows by Using AWS Lambda Triggers (p. 67).

Reviewing Your User Pool Creation Settings

Before you create your user pool, you can review the different settings and edit them in the AWS Management Console. Amazon Cognito validates the user pool settings and warns you if something needs to be changed. For example:

> **Warning**
>
> This user pool does not have an IAM role defined to allow Amazon Cognito to send SMS messages, so it will not be able to confirm phone numbers or for MFA after August 31, 2016. You can define the IAM role by selecting a role on the **Verifications** panel.

If you see a message, follow the instructions to fix them before choosing **Create pool**.

Specifying App Client Settings for Your User Pool

Note

The **App client settings** tab appears only when you're editing an existing user pool.

On the **App client settings** tab, you must configure at least one identity provider (IdP) for your apps if you want to use the built-in hosted pages to sign up and sign in users or you want to use OAuth2.0 flows. For more information, see Specifying Identity Provider Settings for Your User Pool App (p. 83).

To specify app client settings for your user pool

1. In **Enabled Identity Providers**, select the identity providers you want for the apps that you configured in the **App Clients** tab.
2. Enter the **Callback URLs** you want, separated by commas. These URLs apply to all of the selected identity providers.

   Note

   You must register the URLs in the console, or by using the CLI or API, before you can use them in your app.
3. Enter the **Sign out URLs** you want, separated by commas.

   Note

   You must register the URLs in the console, or by using the CLI or API, before you can use them in your app.
4. Under **OAuth 2.0**, select the from the following options. For more information, see About App Identity Settings (p. 83) and the OAuth 2.0 specification.

   - For **Allowed OAuth Flows**, select **Authorized code grant** and **Implicit grant**. Select **Client credentials** only if your app needs to request access tokens on its own behalf, not on behalf of a user.
   - For **Allowed OAuth Scopes**, select the scopes you want. Each scope is a set of one or more standard attributes.
   - For **Allowed Custom Scopes**, select the scopes you want from any custom scopes that you have defined. Custom scopes are defined in the **Resource Servers** tab. For more information, see Defining Resource Servers for Your User Pool (p. 90).

Specifying a Domain Name for Your User Pool

Note

The **Domain name** tab appears only when you're editing an existing user pool.

On the **Domain name** tab, you can enter your own prefix domain name. The domain for your app is `https://<domain_prefix>.auth.<region>.amazoncognito.com`.

The full URL for your app will look like this example: `https://example.auth.us-east-1.amazoncognito.com/login?redirect_uri=https://www.google.com&response_type=code&client_id=<client_id_value>`

For more information, see Assigning a Domain to Your User Pool (p. 86).

> **Important**
> Before you can access the URL for your app, you must specify app client settings such as callback and redirect URLs. For more information, see Specifying App Client Settings for Your User Pool (p. 26).

To specify a domain name for your user pool

1. Enter the domain name you want in the **Prefix domain name** box.
2. Choose **Check availability** as needed.
3. Choose **Save changes**.

Customizing the Built-in App UI to Sign Up and Sign In Users

> **Note**
> The **UI customization** tab appears only when you're editing an existing user pool.

On the **UI customization** tab, you can add your own customizations to the default app UI.

For detailed information about each of the customization fields, see Specifying App UI Customization Settings for Your User Pool (p. 87).

> **Note**
> You can view the hosted UI with your customizations by constructing the following URL, with the specifics for your user pool, and typing it into a browser: `https://<your_domain>/login?response_type=code&client_id=<your_app_client_id>&redirect_uri=<your_callback_url>`
> You may have to wait up to one minute to refresh your browser before changes made in the console appear.
> Your domain is shown on the **Domain name** tab. Your app client ID and callback URL are shown on the **App client settings** tab.

To customize the built-in app UI

1. Under **App client to customize**, choose the app you want to customize from the dropdown menu of app clients that you previously created in the **App clients** tab.
2. To add a logo to the default app UI, choose **Choose a file** or drag a file onto the **Logo** box.
3. Under **CSS customizations (optional)**, you can customize the appearance of the app by changing various properties from their default values.
4. Choose **Save changes**.

Specifying Resource Servers for Your User Pool

> **Note**
> The **Resource Servers** tab appears only when you're editing an existing user pool.

A *resource server* is a server for access-protected resources. It handles authenticated requests from an app that has an access token. A *scope* is a level of access that an app can request to a resource.

In the **Resource Servers** tab, you can define custom resource servers and scopes for your user pool. For more information, see Defining Resource Servers for Your User Pool (p. 90).

To define a custom resource server

1. Choose **Add a resource server**.
2. Enter the name of your resource server, for example, `Photo Server`.
3. Enter the identifier of your resource server, for example, `com.example.photos`.
4. Enter the names of the custom scopes for your resources, such as `read` and `write`.
5. For each of the scope names, enter a description, such as `view your photos` and `update your photos`.

Each of the custom scopes that you define appears on the **App client settings** tab, under **OAuth2.0 Allowed Custom Scopes**; for example `com.example.photos/read`.

Specifying Identity Providers for Your User Pool

> **Note**
> The **Identity providers** tab appears only when you're editing an existing user pool.

In the **Identity providers** tab, you can specify identity providers (IdPs) for your user pool. For more information, see Using Federation for Amazon Cognito User Pools (p. 93).

Allowing Users to Sign in Using a Social Identity Provider

You can use federation for Amazon Cognito User Pools to integrate with social identity providers such as Facebook, Google, and Login with Amazon.

To add a social identity provider, you first create a developer account with the identity provider. Once you have your developer account, you register your app with the identity provider. The identity provider creates an app ID and an app secret for your app, and you configure those values in your Amazon Cognito user pool.

Here are links to help you get started with social identity providers:

- Google Identity Platform
- Facebook for Developers
- Login with Amazon

To allow users to sign in using a social identity provider

1. Choose a social identity provider such as **Facebook**, **Google**, or **Login with Amazon**.
2. For the Facebook (or Google or Amazon) app ID, enter the app ID that you received when you created your Facebook, Google, or Login with Amazon client app.
3. For **App secret**, enter the app secret that you received when you created your client app.
4. For **Authorize scopes**, enter the names of the social identity provider scopes that you want to map to user pool attributes. Scopes define which user attributes (such as `name` and `email`) you want to access with your app. For Facebook, these should be separated by commas (for example, `public_profile, email`). For Google and Login with Amazon, they should be separated by spaces. (Google example: `profile email openid`. Login with Amazon example: `profile postal_code`.)

   The end-user is asked to consent to providing these attributes to your app. For more information about their scopes, see the documentation from Google, Facebook, and Login with Amazon.
5. Choose **Enable Facebook** (or **Enable Google** or **Enable Login with Amazon**).

Allowing Users to Sign in Using Corporate ID via SAML

You can use federation for Amazon Cognito User Pools to integrate with a SAML identity provider (IdP). You supply a metadata document, either by uploading the file or by entering a metadata document endpoint URL. For information about obtaining metadata documents for third-party SAML IdPs, see Integrating Third-Party SAML Identity Providers with Amazon Cognito User Pools (p. 100).

To allow users to sign in using corporate ID via SAML

1. Choose **Corporate ID via SAML** to display the SAML identity provider options.
2. To upload a metadata document, choose **Select file**, or enter a metadata document endpoint URL. The metadata document must be a valid XML file.
3. Enter your SAML **Provider name**, for example, `"SAML_provider_1"`, and any **Identifiers** you want. The provider name is required; the identifiers are optional. For more information, see Creating SAML Identity Providers for Your User Pool (p. 95).
4. Choose **Create provider**.
5. To create additional providers, repeat the previous steps.

Note
If you see `InvalidParameterException` while creating a SAML identity provider with an HTTPS metadata endpoint URL, for example, "Error retrieving metadata from *`<metadata endpoint>`*," make sure that the metadata endpoint has SSL correctly set up and that there is a valid SSL certificate associated with it.

Specifying Attribute Mapping for Your User Pool

Note
The **Attribute mapping** tab appears only when you're editing an existing user pool.

On the **Attribute mapping** tab, you can map identity provider (IdP) attributes or assertions to user pool attributes. For more information, see Specifying Identity Provider Attribute Mappings for Your User Pool (p. 101).

Note
Currently, only the Facebook `id`, Google `sub`, and Login with Amazon `user_id` attributes can be mapped to the Amazon Cognito user pools `username` attribute.

Note
The attribute in the user pool must be large enough to fit the values of the mapped identity provider attributes, or an error occurs when users sign in. Custom attributes should be set to the maximum 2048 character size if mapped to identity provider tokens.
You must create mappings for any attributes that are required for your user pool.

To specify a social identity provider attribute mapping for your user pool

1. Choose the **Facebook**, **Google**, or **Amazon** tab.
2. For each attribute you need to map, perform the following steps:
    1. Choose the **Capture** check box.
    2. In the **User pool attribute** field, choose the user pool attribute to map the social identity provider attribute to from the drop-down list.
    3. If you need more attributes, choose **Add Facebook attribute** (or **Add Google attribute** or **Add Amazon attribute**) and perform the following steps:
        1. In the **Facebook attribute** (or **Google attribute** or **Amazon attribute**) field, enter the name of the attribute to be mapped.

    2. For **User pool attribute**, choose the user pool attribute you want to map to the social identity provider attribute to from the drop-down list.
  4. Choose **Save changes**.

To specify a SAML provider attribute mapping for your user pool

1. Choose the **SAML** tab.
2. For each attribute you need to map, perform the following steps:
    1. Choose **Add SAML attribute**.
    2. In the **SAML attribute** field, enter the name of the SAML attribute to be mapped.
    3. For **User pool attribute**, choose the user pool attribute you want to map to the SAML attribute from the drop-down list.
    4. Choose **Save changes**.

Setting Up the AWS Mobile SDK for Android to Work with User Pools

This section provides information about setting up the AWS Mobile SDK for Android to so you can use the Amazon Cognito Identity Provider in your app. These instructions are written for Android application development in Android Studio.

For more information about setting up the AWS Mobile SDK for Android, see Set Up the AWS Mobile SDK for Android.

Gradle Dependencies

Add the these dependencies in your app's Gradle file:

- **AWS Android Core SDK (aws-android-sdk-core-x.x.x.jar)**: Add the latest version of the AWS Android Core as the **aws-android-sdk-core-2.2.8.jar** as a dependency library to your project's build Gradle.
- **AWS Cognito Identity Provider Android SDK (aws-android-sdk-cognitoidentityprovider:2.3.8.jar)**: Add the latest version of the Android SDK for Cognito Identity Provider in your app's build Gradle.

Network Permissions

To allow your app to make network calls to talk to the AWS Cognito Identity Provider, you need to enable your app to perform network operations.

Include the following permissions in your app's manifest file:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Using the Mobile SDK for Android in Your Amazon Cognito Application

In this section, you will learn about using the Mobile SDK for Android to use Amazon Cognito user pools in your apps. This SDK provides APIs to register (sign up), confirm, and authenticate users, and more.

Key Concepts

Executing in the Current Thread or a Background Thread

All APIs that perform network calls to interact with the Amazon Cognito Identity Provider service have two methods. One of these methods executes the task and the network operations in the current thread (e.g., `signUp()`) and the other method (postfixed with `InBackground`, e.g., `signUpInBackground()`), executes the task in a background thread but invokes the callback methods in the thread where the `InBackground` method was invoked.

Caching

The Mobile SDK for Android caches the last successfully authenticated user and the user's tokens locally on the device, in `SharedPreferences`. The SDK also provides methods to get the last successfully authenticated user.

App Id and App Secret

The App ID (also called Client ID) and App Secret (also called Client Secret) are generated at the Amazon Cognito User Pools console. App Id is necessary to use the Mobile SDK for Android. The App Secret is optional. However, if an App Id has an associated App Secret, then you must use the App Id and App Secret in the SDK.

Main Classes

CognitoUserPool

Represents an abstraction of a user pool. Provides methods to register a new user and create a new instance `CognitoUser` for a user belonging to this pool.

CognitoUser

Represents a single user in a user pool. Through this class you can perform all possible operations on a user, including authentication (Sign-In), managing user attributes and settings, and much more. You can create an instance of this class from the `CognitoUserPool` object.

CognitoUserSession

Encapsulates the tokens issued by Amazon Cognito (ID , access, and refresh token) and provides methods to read ID and access tokens.

CognitoUserDetails

Encapsulates `CognitoUserAttributes` and `CognitoUserSettings`.

CognitoUserAttributes

Encapsulates all user attributes and provides methods to read and write attributes. For more information about attributes, see Specifying User Pool Attribute Settings (p. 14)

CognitoUserSettings

Encapsulates all user settings and provides methods to read and write attributes.

Tutorial: Integrating User Pools for Android Apps

This tutorial outlines the key steps to integrate Amazon Cognito Your User Pools with an Android application. For a complete sample application that shows how to use user pools in your application, see the Amazon Cognito Your User Pools sample on the GitHub website.

Topics

- Step 1: Creating a User Pool for Your App by Using the Console (p. 32)
- Step 2: Creating a User Pool Object (p. 33)
- Step 3: Signing up Users for Your App (p. 33)
- Step 4: Confirming Users for Your App (p. 34)
- Step 5: Resolving Alias Value Conflicts (p. 34)
- Step 6: Signing Users in to Your App (p. 35)
- Step 7: Getting User Details (p. 36)
- Step 8: Getting Credentials to Access AWS Resources for an App User (p. 36)
- Step 9: Setting IAM Permissions to Enable Access to AWS Resources (p. 37)

Step 1: Creating a User Pool for Your App by Using the Console

The following procedure describes how to create a user pool and use it in your app. This procedure creates a pool ID, an app client ID, and an app client secret using default settings. For information on customizing these settings, see Step Through Amazon Cognito User Pool Settings in the AWS Management Console (p. 13).

To create a user pool for your app

1. Sign in to the Amazon Cognito console
2. Choose **Manage your User Pools**.
3. Choose **Create a User Pool**.
4. In **Pool name**, type a name for the pool and then choose **Review defaults**. This creates the pool with the default settings.
5. From the left navigation pane, choose **Attributes** to specify which attributes are required and which attributes to use as aliases. After you set the following attributes and after users in the pool verify their email addresses, they can sign in with their usernames or email addresses.

    a. For **email**, choose **Required** and **Alias**.
    b. For **phone number**, choose **Required** and **Alias**.
    c. For **given name**, choose **Required**.
    d. Choose **Save changes**.
6. From the left navigation pane, choose **Policies** to specify the password policy. For this tutorial, use the default settings.
7. From the left navigation pane, choose **Verifications**. On this page, you can customize the messages that are sent to the users in your pool to deliver verification codes. For this tutorial, use the default settings.
8. From the left navigation pane, choose **Apps** and then choose **Add an app**. You can create multiple app clients for a user pool and you can create one app per platform.
9. For **App name**, type a name for your app. Keep **Generate app client secret** selected, and then choose **Set attribute read and write permissions**. Select the attributes that require write permissions. Required attributes always have write permissions.
10. Choose **Create app** and then choose **Save changes**.
11. From the left navigation bar, choose **Review** and then choose **Create pool**.
12. Note the **Pool ID**, **Pool ARN**, **App client ID**, and **App client secret**. You can find the app client ID and app client secret under **Apps** on the left navigation bar. To view the client secret, choose **Show details**.

Step 2: Creating a User Pool Object

To create a user pool object, you need the pool ID, client ID, and client secret. The following example shows how to create a `ClientConfiguration` object and a `CognitoUserPool` object. The `CognitoUserPool` object is the entry point for all interactions with your user pool from your application. In the sample application the `userPool` is created in `AppHelper.java`.

```
ClientConfiguration clientConfiguration = new ClientConfiguration();

// Create a CognitoUserPool object to refer to your user pool
CognitoUserPool userPool = new CognitoUserPool(context, poolId, clientId, clientSecret,
 clientConfiguration);
```

Step 3: Signing up Users for Your App

The following steps describe how to sign up users for your app.

To sign up users for your app

1. Collect the following information from the user:
   - user-id: This is used by the user to log in and must be unique within the pool.
   - password: This is the user's password.
   - user attributes: You must specify the required attributes (email, given name, and phone number) for your pool.
2. Use the pool instance to sign up the user.

```
// Create a CognitoUserAttributes object and add user attributes
CognitoUserAttributes userAttributes = new CognitoUserAttributes();

// Add the user attributes. Attributes are added as key-value pairs
// Adding user's given name.
// Note that the key is "given_name" which is the OIDC claim for given name
userAttributes.addAttribute("given_name", userGivenName);

// Adding user's phone number
userAttributes.addAttribute("phone_number", phoneNumber);

// Adding user's email address
userAttributes.addAttribute("email", emailAddress);
```

3. Create a callback handler for sign-up. The `onSuccess` method is called when the sign-up is successful.

```
SignUpHandler signupCallback = new SignUpHandler() {

    @Override
    public void onSuccess(CognitoUser cognitoUser, boolean userConfirmed,
 CognitoUserCodeDeliveryDetails cognitoUserCodeDeliveryDetails) {
        // Sign-up was successful

        // Check if this user (cognitoUser) needs to be confirmed
        if(!userConfirmed) {
            // This user must be confirmed and a confirmation code was sent to the user
            // cognitoUserCodeDeliveryDetails will indicate where the confirmation code
 was sent
            // Get the confirmation code from user
        }
        else {
            // The user has already been confirmed
        }
    }

    @Override
    public void onFailure(Exception exception) {
        // Sign-up failed, check exception for the cause
    }
};
```

4. Call the sign-up API.

```
userPool.signUpInBackground(userId, password, userAttributes, null, signupCallback);
```

Step 4: Confirming Users for Your App

Users may need to be confirmed after they sign up before they can sign in. Users can confirm through email or phone. After a successful sign-up, if the user needs to be confirmed, a confirmation code is sent to the user's email address or phone number. You can also automatically confirm a user after sign-up by using Lambda triggers.

If a user provides an email address or phone number during sign-up, and you selected automatic verification for your user pool, a confirmation code is sent to the user's phone number as a text message or to the user's email address. The `cognitoUserCodeDeliveryDetails` object, which was delivered to the callback handler after successful sign-up, indicates where this confirmation code was sent. You can use this to let the user know how he or she will get confirmation code.

The following steps describe how to confirm user information before users can sign in to your app.

To confirm a user for your app

1. Create a callback handler to confirm the user. This callback handler is used by the SDK to communicate the results of the confirmation API call.

```
 // Callback handler for confirmSignUp API
GenericHandler confirmationCallback = new GenericHandler() {

    @Override
    public void onSuccess() {
    // User was successfully confirmed
    }

    @Override
    public void onFailure(Exception exception) {
    // User confirmation failed. Check exception for the cause.
    }
};
```

2. When a new user is confirmed, the user's attribute through which the confirmation code was sent (email address or phone number) is marked as verified. If this attribute is also set to be used as an alias, then the user can sign in with that attribute (email address or phone number) instead of the username.

Step 5: Resolving Alias Value Conflicts

Alias values must be unique in a pool. When you confirm a new user, if that user's email address or phone number are used as an alias, and that email or phone number are already in use for an existing user in the pool, you must resolve this conflict. To ensure uniqueness, you can do either of the following:

- Set the `forcedAliasCreation` parameter to false. This resolves the conflict by allowing the user confirmation to fail. The attribute remains verified for the existing user and continues to be an alias for the existing user. The new user remains un-confirmed, as shown in the following example.

```
 // This will cause confirmation to fail if the user attribute has been verified for
 another user in the same pool
boolean forcedAliasCreation = false;

// Call API to confirm this user
cognitoUser.confirmSignUpInBackground(confirmationCode, forcedAliasCreation,
 confirmationCallback);
```

- Setting the `forcedAliasCreation` parameter to true resolves the conflict by marking the attribute (email or phone number) as verified for the new user, and consequently marking it as not-verified for the existing user. This attribute is no longer an alias for the existing user.

All confirmed users can sign in. On successful sign-in, access and ID tokens are returned. These tokens are in a `CognitoUserSession` object.

Step 6: Signing Users in to Your App

To sign a user in to your app, you must first create a callback handler for authentication. The following example shows how the SDK interacts with your application through this callback handler.

```
// Callback handler for the sign-in process
AuthenticationHandler authenticationHandler = new AuthenticationHandler() {

    @Override
    public void onSuccess(CognitoUserSession cognitoUserSession) {
        // Sign-in was successful, cognitoUserSession will contain tokens for the user
    }

    @Override
    public void getAuthenticationDetails(AuthenticationContinuation
 authenticationContinuation, String userId) {
        // The API needs user sign-in credentials to continue
        AuthenticationDetails authenticationDetails = new AuthenticationDetails(userId,
 password, null);

        // Pass the user sign-in credentials to the continuation
        authenticationContinuation.setAuthenticationDetails(authenticationDetails);

        // Allow the sign-in to continue
        authenticationContinuation.continueTask();
    }

    @Override
    public void getMFACode(MultiFactorAuthenticationContinuation
 multiFactorAuthenticationContinuation) {
        // Multi-factor authentication is required; get the verification code from user
        multiFactorAuthenticationContinuation.setMfaCode(mfaVerificationCode);
        // Allow the sign-in process to continue
        multiFactorAuthenticationContinuation.continueTask();
    }

    @Override
    public void onFailure(Exception exception) {
        // Sign-in failed, check exception for the cause
    }
};

// Sign in the user
cognitoUser.getSessionInBackground(authenticationHandler);
```

Step 7: Getting User Details

After authenticating a user, you can retrieve other information about the user in the user pool, as shown in the following example.

```
// Implement callback handler for getting details
GetDetailsHandler getDetailsHandler = new GetDetailsHandler() {
    @Override
    public void onSuccess(CognitoUserDetails cognitoUserDetails) {
        // The user detail are in cognitoUserDetails
    }

    @Override
    public void onFailure(Exception exception) {
        // Fetch user details failed, check exception for the cause
    }
};

// Fetch the user details
cognitoUser.getDetailsInBackground(getDetailsHandler);
```

Step 8: Getting Credentials to Access AWS Resources for an App User

To get credentials to access AWS resources for your user, first create an identity pool and associate your user pool with that identity pool.

To get AWS credentials to access AWS resources

1. Sign in to the Amazon Cognito console.
2. Choose **Manage Federated Identities**.
3. Choose **Create new identity pool**. Type a name for your identity pool in **Identity pool name**.
4. Expand the **Authentication providers** section. On the **Cognito** tab, type the **User Pool ID** and the **App Client ID** for the user pool you just created.
5. Choose **Create Pool**.
6. In your application code, add the ID tokens, received after successful authentication, to your credentials provider, as follows.

```
// Get id token from CognitoUserSession.
   String idToken = cognitoUserSession.getIdToken().getJWTToken();

// Create a credentials provider, or use the existing provider.
   CognitoCachingCredentialsProvider credentialsProvider = new
CognitoCachingCredentialsProvider(context, IDENTITY_POOL_ID, REGION);

// Set up as a credentials provider.
   Map<String, String> logins = new HashMap<String, String>();
   logins.put("cognito-idp.us-east-1.amazonaws.com/us-east-1_123456678",
cognitoUserSession.getIdToken().getJWTToken());
   credentialsProvider.setLogins(logins);
```

7. Use the credentials provider to access AWS resources, such as a Amazon DynamoDB table, as follows.

```
AmazonDynamoDBClient ddbClient = new AmazonDynamoDBClient(credentialsProvider);
```

Step 9: Setting IAM Permissions to Enable Access to AWS Resources

When you create an identity pool, Amazon Cognito creates two IAM roles: `Cognito`*`<identity pool name>`*`Auth_Role` and `Cognito`*`<identity pool name>`*`Unauth_Role`. By default, these roles only allow access to Amazon Cognito Identity and Amazon Cognito Sync. To allow your application to access AWS services such as Amazon DynamoDB, you must attach the appropriate managed policy to the role. For example, if your application needs to read and write to a DynamoDB database you must attach the `AmazonDynamoDBFullAccess` managed policy to the role, as described in the following procedure.

To set IAM permissions to enable access to AWS resources

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose the authenticated role for your policy from the list of roles, and then choose **Attach Policy**.
3. Choose the required policy for the list of managed policies (**AmazonDynamoDBFullAccess**, for example), and then choose **Attach Policy**.

You application can now perform create, read, update, and delete operations in DynamoDB.

Examples of Using User Pools with the Mobile SDK for Android

This topic provides code examples that perform basic tasks using the Mobile SDK for Android. Since the SDK makes network calls, all API calls should be made from a non-activity thread.

Create a CognitoUserPool

```
CognitoUserPool userPool = new CognitoUserPool(context, userPoolId, clientId,
 clientSecret);

// user pool can also be created with client app configuration:
CognitoUserPool userPool = new CognitoUserPool(context, userPoolId, clientId, clientSecret,
 clientConfiguration);
```

Register a New User

```
// create a handler for registration
SignUpHandler handler = new SignUpHandler() {
    @Override
    public void onSuccess(CognitoUser user, CognitoUserCodeDeliveryDetails
 codeDeliveryDetails) {
        // If the sign up was successful, "user" is a CognitoUser object of the user who
 was signed up.
        // "codeDeliveryDetails" will contain details about where the confirmation codes
 will be delivered.
}

    @Override
    public void onFailure(Exception exception) {
        // Sign up failed, code check the exception for cause and perform remedial actions.
    }
}
```

Get the Cached User

```
CognitoUser user = userPool.getCurrentUser();
```

Create a User Object with a UserId

```
CognitoUser user = userPool.getUser(userId);
```

Confirm a User

```
// create a callback handler for confirm
GenericHandler handler = new GenericHandler() {

    @Override
    public void onSuccess() {
        // User was successfully confirmed!
    }
    @Override
    public void onFailure(Exception exception) {
        // Confirmation failed, probe exception for details
    }
}

user.confirmSignUp(code, handler);
```

Request a Confirmation Code

```
// create a callback handler for the confirmation code request
GenericHandler handler = new GenericHandler() {

    @Override
    public void onSuccess() {
        // Confirmation code was successfully sent!
    }
    @Override
    public void onFailure(Exception exception) {
        // Confirmation code request failed, probe exception for details
    }
}

user.resendConfirmationCode(handler);
```

Forgot Password: Get Code to Set New Password

```
ForgotPasswordHandler handler = new ForgotPasswordHandler() {
    @Override
    public void onSuccess() {
        // Forgot password process completed successfully, new password has been
 successfully set

    }

    @Override
    public void getResetCode(ForgotPasswordContinuation continuation) {
        // A code will be sent, use the "continuation" object to continue with the forgot
 password process

        // This will indicate where the code was sent
        String codeSentHere = continuation.getParameters();

        // Code to get the code from the user - user dialogs etc.

        // If the program control has to exit this method, take the "continuation" object.
        // "continuation" is the only possible way to continue with the process



        // When the code is available

        // Set the new password
        continuation.setPassword(newPassword);

        // Set the code to verify
        continuation.setVerificationCode(code);

        // Let the forgot password process proceed
        continuation.continueTask();

    }

    /**
     * This is called for all fatal errors encountered during the password reset process
     * Probe {@exception} for cause of this failure.
     * @param exception
     */
    public void onFailure(Exception exception) {
        // Forgot password processing failed, probe the exception for cause
    }
}

user.forgotPassword(handler);
```

Authentication Handler: Get Tokens

```
// Implement authentication handler,
AuthenticationHandler handler = new AuthenticationHandler() {
    @Override
    public void onSuccess(CognitoUserSession userSession, CognitoDevice newDevice) {
        // Authentication was successful, the "userSession" will have the current valid
 tokens
        // Time to do awesome stuff
    }

    @Override
    public void getAuthenticationDetails(final AuthenticationContinuation continuation,
 final String userID) {
        // User authentication details, userId and password are required to continue.
        // Use the "continuation" object to pass the user authentication details

        // After the user authentication details are available, wrap them in an
 AuthenticationDetails class
        // Along with userId and password, parameters for user pools for Lambda can be
 passed here
        // The validation parameters "validationParameters" are passed in as a Map<String,
 String>
        AuthenticationDetails authDetails = new AuthenticationDetails(userId, password,
 validationParameters);
```

```
        // Now allow the authentication to continue
        continuation.setAuthenticationDetails(authDetails);
        continuation.continueTask();
    }

    @Override
    public void getMFACode(final MultiFactorAuthenticationContinuation continuation) {
        // Multi-factor authentication is required to authenticate
        // A code was sent to the user, use the code to continue with the authentication


        // Find where the code was sent to
        String codeSentHere = continuation.getParameter()[0];

        // When the verification code is available, continue to authenticate
        continuation.setMfaCode(code);
        continuation.continueTask();
    }

    @Override
    public void authenticationChallenge(final ChallengeContinuation continuation) {
        // A custom challenge has to be solved to authenticate

        // Set the challenge responses

        // Call continueTask() method to respond to the challenge and continue with
 authentication.
    }



    @Override
    public void onFailure(final Exception exception) {
        // Authentication failed, probe exception for the cause

    }
};
user.getSession(handler);
```

Get User Details

```
GetDetailsHandler handler = new GetDetailsHandler() {
    @Override
    public void onSuccess(final CognitoUserDetails list) {
        // Successfully retrieved user details
    }

    @Override
    public void onFailure(final Exception exception) {
        // Failed to retrieve the user details, probe exception for the cause
    }
};
user.getDetails(handler);
```

Get Attribute Verification Code

```
GenericHandler handler = new GenericHandler() {

    @Override
    public void onSuccess() {
        // Attribute verification code was successfully sent!
    }
    @Override
    public void onFailure(Exception exception) {
        // Attribute verification code request failed, probe exception for details
    }
};
user.getAttibuteVerificationCode(attributeName, handler);
```

Verify Attribute

```
GenericHandler handler = new GenericHandler() {

    @Override
    public void onSuccess() {
        // Attribute verification was successful!
    }

    @Override
    public void onFailure(Exception exception) {
        // Attribute verification failed, probe exception for details
    }
};
user.verifyAttribute(attributeName, code, handler);
```

Delete Attribute

```
GenericHandler handler = new GenericHandler() {

    @Override
    public void onSuccess() {
        // Attribute deletion was successful!
    }

    @Override
    public void onFailure(Exception exception) {
        // Attribute deletion failed, probe exception for details
    }
};
user.deleteAttribute(attributeName, handler);
```

Change Password

```
GenericHandler handler = new GenericHandler() {

    @Override
    public void onSuccess() {
        // Password change was successful!
    }

    @Override
    public void onFailure(Exception exception) {
        // Password change failed, probe exception for details
    }
};
user.changePassword(oldPassword, newPassword, handler);
```

Change or Set User Settings

```
GenericHandler handler = new GenericHandler() {
    @Override
    public void onSuccess() {
        // Successfully changed settings!
    }

    @Override
    public void onFailure(Exception exception) {
        // Change failed, probe exception for details
    }
};

// userSettings is an object of the type CognitoUserSettings,
CognitoUserSettings userSettings = new CognitoUserSettings();

// Set the user settings
userSettings.setSettings(settingName, settingValue);

// Now update the new settings to the Amazon Cognito Identity Provider Service
user.setUserSettings(userSettings, handler);
```

Delete User

```
GenericHandler handler = new GenericHandler() {

    @Override
    public void onSuccess() {
        // Delete was successful!
    }

    @Override
    public void onFailure(Exception exception) {
        // Delete failed, probe exception for details
    }
};
user.deleteUser(handler);
```

Sign Out User

```
// This has cleared all tokens and this user will have to go through the authentication
 process to get tokens.
user.signOut();
```

Get Access and ID Tokens from CognitoUserSession

```
// Session is an object of the type CognitoUserSession
String accessToken = session.getAccessToken().getJWT();
String idToken = session.getIdToken().getJWTToken();
```

List All Devices for a User

```
DevicesHandler devicesHandler = new DevicesHandler() {
    @Override
    public void onSuccess(List<CognitoDevice> devices) {
        // devices will contain a list of all remembered devices
    }

    @Override
    public void onFailure(Exception e) {
        // List devices failed, probe exception for details

    }
};
user.listDevicesInBackground(10, null, devicesHandler);
```

Remember a Device

```
GenericHandler handler = new GenericHandler() {

    @Override
    public void onSuccess() {
        // Successful!
    }

    @Override
    public void onFailure(Exception exception) {
        // Failed, probe exception for details
    }
};
cognitoDevice.rememberThisDeviceInBackground(handler)
```

Do Not Remember a Device

```
GenericHandler handler = new GenericHandler() {

    @Override
    public void onSuccess() {
        // Successful!
    }

    @Override
    public void onFailure(Exception exception) {
        // Failed, probe exception for details
    }
};
cognitoDevice.doNotRememberThisDeviceInBackground(handler)
```

Example: Handling Users Created Using the AdminCreateUser API in the Mobile SDK for Android

Amazon Cognito Your User Pools allows administrators to create new users and invite the users to sign in. The user must set his or her password during the first sign-in. Also during the first sign-in, the user must provide values for any required attributes that don't already have values.

The Mobile SDK for Android (version 2.3.2 and later) supports this feature. To support this feature in your apps, you must implement the `AuthenticationChallenge` callback method. The user authentication process for these users has not changed. However, after the initial password verification, the SDK invokes the `AuthenticationChallenge` callback, which you can implement to read the new password from the user. You can then allow the user to set required attributes and change user attributes that were already set by the administrator.

The continuation object passed to the `AuthenticationChallenge` callback method is of the type `NewPasswordContinuation`. The `NewPasswordContinuation` class is a child of `ChallengeContinuation`. The `ChallengeContinuation` class provides easier access to the challenge attributes.

When the `AuthenticationChallenge` callback is invoked during the user authentication process, first check the `Challenge name`. The challenge name, `NEW_PASSWORD_REQUIRED`, indicates that the user is trying to sign in for the first time after the administrator created the user's account. To get the challenge name, call `continuation.getChallengeName`.

To complete the sign-in process, the user must set a new password and provide any missing values for user attributes that were marked as required when the user pool was created or updated. To get the list of all required attributes, call `continuation.getRequiredAttributes`. To get the attributes and the values that were already set by the administrator, call `continuation.getCurrentUserAttributes`.

Call `continuation.setPassword` and `continuation.setUserAttribute`, respectively, to set the user's new password and attributes (including required attributes).

Call `continuation.continueTask` to complete the sign-in process.

```
@Override
    public void authenticationChallenge(final ChallengeContinuation continuation) {
        // Check the challenge name
 if("NEW_PASSWORD_REQUIRED".equals(continuation.getChallengeName()) {
  // A new user is trying to sign in for the first time after
  // admin has created the user’s account

  // Cast to NewPasswordContinuation for easier access to challenge parameters
  NewPasswordContinuation newPasswordContinuation = (NewPasswordContinuation) continuation;

  // Get the list of required parameters
  List<String> requiredAttributes = newPasswordContinuation.getRequiredAttributes()

  // Get the current user attributes
  Map<String, String> currUserAttributes =
 newPasswordContinuation.getCurrentUserAttributes();

  // Prompt user to set a new password and values for required attributes

  // Set new user password
  newPasswordContinuation.setPassword();

  // Set user attributes
  newPasswordContinuation.setUserAttribute(attributeName, attributeValue);

  // Set user attributes
  newPasswordContinuation.setUserAttribute(anotherAttribute, valueOfAnotherAttribute);

  // Allow the sign-in to complete
  newPasswordContinuation.continueTask();
 }
        // Set the challenge responses

        // Call continueTask() method to respond to the challenge and continue with
 authentication.
    }
```

Setting Up the AWS Mobile SDK for iOS to Work with User Pools

Amazon Cognito Identity provides a Mobile SDK for iOS. The following topic provides set-up instructions and examples for common tasks while working with user pools.

Installing the AWS Mobile SDK for iOS

The following procedure describes how to set up the SDK.

To set up the Mobile SDK for iOS

1. Follow the instructions here: Set Up the Mobile SDK for iOS.
2. If you are using CocoaPods, add pod **AWSCognitoIdentityProvider** to your PodSpec and `#import AWSCognitoIdentityProvider.h` in the classes you want to use it in.
3. If you are using Frameworks, add **AWSCognitoIdentityProvider.framework** and `#import <AWSCognitoIdentityProvider/AWSCognitoIdentityProvider.h>` into the classes you want to use it.

Tutorial: Integrating User Pools for iOS Apps

This tutorial helps you get started with user pools.

Topics

- Step 1: Creating a User Pool for Your App by Using the Console (p. 45)
- Step 2: Creating a UserPool Object (p. 45)
- Step 3: Signing up Users for Your App (p. 46)
- Step 4: Confirming Users for Your App (p. 46)
- Step 5: Authenticating Users for Your App (p. 47)
- Step 6: Getting User Details (p. 47)
- Step 7: Getting Credentials to Access AWS Resources For an App User (p. 48)
- Next Steps (p. 48)

Step 1: Creating a User Pool for Your App by Using the Console

The following procedure describes how to create a user pool and use it in your app. This procedure creates a pool ID, an app client ID, and an app client secret using default settings. For information on customizing these settings, see Step Through Amazon Cognito User Pool Settings in the AWS Management Console (p. 13).

To create a user pool for your app

1. Sign in to the Amazon Cognito console
2. Choose **Manage your User Pools**.
3. Choose **Create a User Pool**.
4. In **Pool name**, type a name for the pool and then choose **Review defaults**.
5. From the left navigation bar, choose **Apps** and then choose **Add an app**. You can create multiple app clients for a user pool and you can create one app per platform.
6. For **App name**, type a name for your app. Keep **Generate client secret** selected, choose **Create app**, and then choose **Save changes**.
7. From the left navigation bar, choose **Review** and then choose **Create pool**.
8. Note the pool ID. You can find the app client ID and app client secret under **Apps** on the left navigation bar.

Step 2: Creating a UserPool Object

You must create a `UserPool` object for your client app. Using the user pool ID, app client ID, and app client secret you obtained in step 1, create an `AWSCognitoIdentityUserPool`.

```
  //setup service config
  AWSServiceConfiguration *serviceConfiguration = [[AWSServiceConfiguration alloc]
 initWithRegion:AWSRegionUSEast1 credentialsProvider:nil];

  //create a pool
  AWSCognitoIdentityUserPoolConfiguration *configuration =
 [[AWSCognitoIdentityUserPoolConfiguration alloc] initWithClientId:@"CLIENT_ID"

                  clientSecret:@"CLIENT_SECRET"

                        poolId:@"USER_POOL_ID"];
  [AWSCognitoIdentityUserPool
 registerCognitoIdentityUserPoolWithConfiguration:serviceConfiguration
 userPoolConfiguration:configuration forKey:@"UserPool"];
  AWSCognitoIdentityUserPool *pool = [AWSCognitoIdentityUserPool
 CognitoIdentityUserPoolForKey:@"UserPool"];
```

Step 3: Signing up Users for Your App

To sign up users, your app's registration UI must collect information from users and call `signUp`.

```
NSMutableArray * attributes = [NSMutableArray new];

//Set user attributes by retrieving them from your UI.  These values are hardcoded for this
 example
AWSCognitoIdentityUserAttributeType * phone = [AWSCognitoIdentityUserAttributeType new];

phone.name = @"phone_number";
//All phone numbers require +country code as a prefix
phone.value = @"+15555555555";

AWSCognitoIdentityUserAttributeType * email = [AWSCognitoIdentityUserAttributeType new];
email.name = @"email";
email.value = @"email@mydomain.com";

[attributes addObject:phone];
[attributes addObject:email];

//set username and password by retrieving them from your UI.  They are hardcoded in this
 example.
AWSCognitoIdentityUser *user = [[pool signUp:@"username" password:@"password"
 userAttributes:attributes validationData:nil] continueWithSuccessBlock:^id
 _Nullable(AWSTask<AWSCognitoIdentityUser *> * _Nonnull task) {
    NSLog(@"Successfully registered user: %@",task.result.username);
    return nil;
}];
```

Step 4: Confirming Users for Your App

Users are confirmed when either their email address or phone number is verified. In the following example, users receive a verification code at their email address or via SMS on their mobile phone during the registration flow and must input the code to complete sign-up. After obtaining the verification code from your end user, call `confirmSignUp`.

```
//replace VERIFICATION_CODE with the value the user inputs
[[user confirmSignUp:@"VERIFICATION_CODE"] continueWithSuccessBlock:^id
 _Nullable(AWSTask<AWSCognitoIdentityProviderConfirmSignUpResponse *> * _Nonnull task) {
    NSLog(@"Successfully confirmed user: %@",user.username);
    return nil;
}];
```

Step 5: Authenticating Users for Your App

To authenticate the confirmed user, implement the `AWSCognitoIdentityInteractiveAuthenticationDelegate` protocol, as shown next, and set the delegate for the pool. This protocol manages your custom login UI and accepts username and password information from your end user. The protocol's methods are only invoked if the user has never authenticated, if the user has signed out, or if the user's refresh token (which is valid for 30 days) has expired.

```
    //This code goes in your AppDelegate
    pool.delegate = self;

    -(id<AWSCognitoIdentityPasswordAuthentication>) startPasswordAuthentication{
        //implement code to instantiate and display login UI here
    //return something that implements the AWSCognitoIdentityPasswordAuthentication
 protocol
    return loginUI;
    }

    //This code goes in your Login UI
    -(void) getPasswordAuthenticationDetails:
 (AWSCognitoIdentityPasswordAuthenticationInput *) authenticationInput
 passwordAuthenticationCompletionSource: (AWSTaskCompletionSource *)
 passwordAuthenticationCompletionSource {
        //using inputs from login UI create an
 AWSCognitoIdentityPasswordAuthenticationDetails object.
        //These values are hardcoded for this example.
        AWSCognitoIdentityPasswordAuthenticationDetails * result =
 [[AWSCognitoIdentityPasswordAuthenticationDetails alloc] initWithUsername:@"USERNAME"
 password:@"PASSWORD"];
        //set the result to continue the sign-in process
        passwordAuthenticationDetails.result = result;
    };

    -(void) didCompletePasswordAuthenticationStepWithError:(NSError*) error {
    dispatch_async(dispatch_get_main_queue(), ^{
        //present error to end user
        if(error){
            [[[UIAlertView alloc] initWithTitle:error.userInfo[@"__type"]
                                        message:error.userInfo[@"message"]
                                       delegate:nil
                              cancelButtonTitle:nil
                              otherButtonTitles:@"Ok", nil] show];
        }else{
            //dismiss view controller
            [self dismissViewControllerAnimated:YES completion:nil];
        }
    });
}
```

Step 6: Getting User Details

To get user details, call `getDetails`, as shown next.

```
[[user getDetails] continueWithSuccessBlock:^id
 _Nullable(AWSTask<AWSCognitoIdentityUserGetDetailsResponse *> * _Nonnull task) {
     AWSCognitoIdentityUserGetDetailsResponse *response = task.result;
     for (AWSCognitoIdentityUserAttributeType *attribute in response.userAttributes) {
         //print the user attributes
         NSLog(@"Attribute: %@ Value: %@", attribute.name, attribute.value);
     }
     return nil;
}];
```

Step 7: Getting Credentials to Access AWS Resources For an App User

To get credentials to access AWS resources for your user, first associate your user pool with an identity pool, and then provide `AWSCognitoIdentityUserPool` to your `AWSCognitoCredentialsProvider`. The following procedure describes how to get an identity pool.

To create an identity pool

1. Sign in to the Amazon Cognito console.
2. Choose **Manage Federated Identities**.
3. Choose **Create new identity pool**. Type a name for your identity pool in **Identity pool name**.
4. Expand the **Authentication providers** section.
5. On the Cognito tab, specify your **User Pool ID** and **App Client ID**.
6. After you configure the identity pool association, get AWS credentials into your app by providing `AWSCognitoIdentityUserPool` to your `AWSCognitoCredentialsProvider`.

```
AWSCognitoCredentialsProvider *credentialsProvider = [[AWSCognitoCredentialsProvider
 alloc] initWithRegionType:AWSRegionUSEast1 identityPoolId:@"IDENTITY_POOL_ID"
 identityProviderManager:pool];
AWSServiceConfiguration *defaultServiceConfiguration = [[AWSServiceConfiguration alloc]
 initWithRegion:AWSRegionUSEast1
        credentialsProvider:credentialsProvider];
AWSServiceManager.defaultServiceManager.defaultServiceConfiguration =
 defaultServiceConfiguration;
```

Next Steps

For a working example demonstrating the functionality described in this tutorial, see the Objective-C sample on Github.

Examples: Using User Pools with the iOS SDK

This topic provides details about registering, confirming, and authenticating users, as well as getting user attributes, when using user pools with the AWS Mobile SDK for iOS.

Creating an AWSCognitoIdentityUserPool Object

The following procedure describes how to create an **AWSCognitoIdentityUserPool** object to interact with.

1. Set up your service config.

```
//setup service config
AWSServiceConfiguration *serviceConfiguration = [[AWSServiceConfiguration alloc]
 initWithRegion:AWSRegionUSEast1 credentialsProvider:nil];
```

2. Create a user pool configuration.

```
//create a pool
AWSCognitoIdentityUserPoolConfiguration *configuration =
 [[AWSCognitoIdentityUserPoolConfiguration alloc] initWithClientId:@"CLIENT_ID"
```

```
                          clientSecret:@"CLIENT_SECRET"

                                poolId:@"USER_POOL_ID"];
[AWSCognitoIdentityUserPool
 registerCognitoIdentityUserPoolWithConfiguration:serviceConfiguration
 userPoolConfiguration:configuration forKey:@"UserPool"];
AWSCognitoIdentityUserPool *pool = [AWSCognitoIdentityUserPool
 CognitoIdentityUserPoolForKey:@"UserPool"];
```

Example: Register a User

Use **pool.signUp:password:userAttributes:validationData** to register a user.

```
AWSCognitoIdentityUserAttributeType * phone = [AWSCognitoIdentityUserAttributeType new];
phone.name = @"phone_number";
//phone number must be prefixed by country code
phone.value = @"+15555555555";
AWSCognitoIdentityUserAttributeType * email = [AWSCognitoIdentityUserAttributeType new];
email.name = @"email";
email.value = @"email@mydomain.com";

//register the user
[[pool signUp:@"username" password:@"password" userAttributes:@[email,phone]
 validationData:nil] continueWithBlock:^id
 _Nullable(AWSTask<AWSCognitoIdentityUserPoolSignUpResponse *> * _Nonnull task) {
    dispatch_async(dispatch_get_main_queue(), ^{
        if(task.error){
            [[[UIAlertView alloc] initWithTitle:task.error.userInfo[@"__type"]
                                        message:task.error.userInfo[@"message"]
                                       delegate:self
                              cancelButtonTitle:@"Ok"
                              otherButtonTitles:nil] show];
        }else {
            AWSCognitoIdentityUserPoolSignUpResponse * response = task.result;
            if(!response.userConfirmed){
                //need to confirm user using user.confirmUser:
            }
        }});
    return nil;
}];
```

Example: Get a User

You can either get a user by registering or by using one of these methods on the pool.

```
//get the last logged in user
[pool currentUser];

//get a user without a username
[pool getUser];

//get a user with a specific username
[pool getUser:@"username"];
```

Example: Sign in a User

There are two ways to sign in: explicitly or when credentials are needed via a delegate.

To sign in explicitly, use the following:

```
[user getSession:@"username" password:@"password" validationData:nil scopes:nil];
```

To implement the delegate, implement **AWSCognitoIdentityInteractiveAuthenticationDelegate** and set the delegate on the pool:

```
pool.delegate = self;
```

In your implementation, write code to instantiate your authentication user interfaces if they weren't created and display them.

```
//set up password authentication ui to retrieve username and password from the user
-(id) startPasswordAuthentication {
      //write code to instantiate your sign in ui if it wasn't created here
      dispatch_async(dispatch_get_main_queue(), ^{
            //write code to display your ui
      });

      //return your sign in ui which implements the
 AWSCognitoIdentityPasswordAuthentication protocol
      return signInViewController;
}

//set up mfa ui to retrieve mfa code from end user
//this is optional and only necessary if you turn on multifactor authentication on your
 pool
-(id) startMultiFactorAuthentication {
      //write code to instantiate your multifactor authentication ui if it wasn’t created
 here
      dispatch_async(dispatch_get_main_queue(), ^{
            //write code to display your ui
      });

      //return your sign in ui which implements the
 AWSCognitoIdentityMultiFactorAuthentication protocol
      return mfaViewController;
}

//set up new password required ui to retrieve new password and any required user profile
 from end user
//this is optional and only necessary if you use the AdminCreateUser feature on the pool
-(id) startNewPasswordRequired {
      //write code to instantiate your new password required ui if it wasn’t created here
      dispatch_async(dispatch_get_main_queue(), ^{
            //write code to display your ui
      });

      //return your new password required ui which implements the
 AWSCognitoIdentityNewPasswordRequired protocol
      return newPasswordRequiredController;
}
```

In your password authentication UI, implement the **AWSCognitoIdentityPasswordAuthentication** protocol.

```
-(void) getPasswordAuthenticationDetails: (AWSCognitoIdentityPasswordAuthenticationInput
 *) authenticationInput  passwordAuthenticationCompletionSource:
 (AWSTaskCompletionSource<AWSCognitoIdentityPasswordAuthenticationDetails *> *)
 passwordAuthenticationCompletionSource {
   //keep a handle to the completion, you'll need it continue once you get the inputs from
 the end user
   self.passwordAuthenticationCompletion = passwordAuthenticationCompletionSource;
   //authenticationInput has details about the last known username if you need to use it
}

-(void) didCompletePasswordAuthenticationStepWithError:(NSError*) error {
    dispatch_async(dispatch_get_main_queue(), ^{
        //on completion, either display the error or dismiss the ui
        if(error){
            [[[UIAlertView alloc] initWithTitle:error.userInfo[@"__type"]
                                        message:error.userInfo[@"message"]
                                       delegate:nil
                              cancelButtonTitle:nil
                              otherButtonTitles:@"Retry", nil] show];
        }else{
            [self dismissViewControllerAnimated:YES completion:nil];
        }
    });
}
```

When the end user has entered his or her username and password, set the result on **passwordAuthenticationCompletion**.

```
self.passwordAuthenticationCompletion.result =
 [[AWSCognitoIdentityPasswordAuthenticationDetails alloc] initWithUsername:@"Username"
 password:@"Password"];
```

If you support multi-factor authentication (MFA), you can implement the **AWSCognitoIdentityMultiFactorAuthentication** protocol.

```
-(void) getMultiFactorAuthenticationCode:
 (AWSCognitoIdentityMultifactorAuthenticationInput )authenticationInput
 mfaCodeCompletionSource: (AWSTaskCompletionSource<NSString > *) mfaCodeCompletionSource {
//keep a handle to the completion, you’ll need it continue once you get the inputs from the
 end user
self.mfaCodeCompletion = mfaCodeCompletionSource;
//authenticationInput has details about where the mfa code was sent if you need to display
 them in your ui
}
-(void) didCompleteMultifactorAuthenticationStepWithError:(NSError*) error {
dispatch_async(dispatch_get_main_queue(), ^{
//on completion, either display the error or dismiss the ui
if(error){
[[[UIAlertView alloc] initWithTitle:error.userInfo[@"__type"]
message:error.userInfo[@"message"]
delegate:nil
cancelButtonTitle:nil
otherButtonTitles:@"Retry", nil] show];
}else{
[self dismissViewControllerAnimated:YES completion:nil];
}
});
}
```

When the end user has entered his or her code, set the result on **mfaCodeCompletion**.

```
self.mfaCodeCompletion.result = @"mfaCodeFromUser";
```

If you support sign-up using **AdminCreateUser**, you can implement the **AWSCognitoIdentityNewPasswordRequired** protocol.

```
-(void) getNewPasswordDetails: (AWSCognitoIdentityNewPasswordRequiredInput
          *) newPasswordRequiredInput
          newPasswordRequiredCompletionSource:
(AWSTaskCompletionSource<AWSCognitoIdentityNewPasswordRequiredDetails *> *)
 newPasswordRequiredCompletionSource {
      //keep a handle to the completion, you'll need it continue once you get the inputs
 from the end user
      self.newPasswordRequiredCompletionSource = newPasswordRequiredCompletionSource;
      //AWSCognitoIdentityNewPasswordRequiredDetails has details about the existing user
 attributes and required fields if you need to display them in your ui
}

-(void) didCompleteNewPasswordStepWithError:(NSError* _Nullable) error {
      dispatch_async(dispatch_get_main_queue(), ^{
            //on completion, either display the error or dismiss the ui
            if(error){
                  [[[UIAlertView alloc] initWithTitle:error.userInfo[@"__type"]
                  message:error.userInfo[@"message"]
                  delegate:nil
                  cancelButtonTitle:nil
                  otherButtonTitles:@"Retry", nil] show];
            }else{
                  [self dismissViewControllerAnimated:YES completion:nil];
            }
      });
}
```

When the end user has entered their proposed password and any required attributes, set the result on **newPasswordRequiredCompletionSource**.

```
NSDictionary<NSString *, NSString *> *userAttributes = @{@"name":@"My new name",
 @"email":@"mynewemail@myemail.com"};
AWSCognitoIdentityNewPasswordRequiredDetails *details =
 [[AWSCognitoIdentityNewPasswordRequiredDetails alloc]
 initWithProposedPassword:@"newPassword" userAttributes:userAttributes];
self.newPasswordRequiredCompletionSource.result = details;
```

Example: Forgot Password

```
[[user forgotPassword] continueWithSuccessBlock:^id
 _Nullable(AWSTask<AWSCognitoIdentityUserForgotPasswordResponse*> * _Nonnull task) {
    //success
    return nil;
}];

[[user confirmForgotPassword:@"code" password:@"newPassword"] continueWithSuccessBlock:^id
 _Nullable(AWSTask<AWSCognitoIdentityUserConfirmForgotPasswordResponse *> * _Nonnull task)
 {
    //success
    return nil;
}];
```

Authenticated Example: Get User Attributes

```
[[user getDetails] continueWithBlock:^id
 _Nullable(AWSTask<AWSCognitoIdentityUserGetDetailsResponse *> * _Nonnull task) {
    dispatch_async(dispatch_get_main_queue(), ^{
        if(task.error){
            [[[UIAlertView alloc] initWithTitle:task.error.userInfo[@"__type"]
                                        message:task.error.userInfo[@"message"]
                                       delegate:self
                              cancelButtonTitle:nil
```

```
                               otherButtonTitles:@"Retry", nil] show];
        }else{
            AWSCognitoIdentityUserGetDetailsResponse *response = task.result;
            //do something with response.userAttributes
        }
    });
    return nil;
}];
```

Authenticated Example: Verify User Attributes

```
[[user getAttributeVerificationCode:@"phone_number"] continueWithSuccessBlock:^id
 _Nullable(AWSTask<AWSCognitoIdentityUserGetAttributeVerificationCodeResponse *> * _Nonnull
 task) {
    //success
    return nil;
}];

[[user verifyAttribute:@"phone_number"code:@"code"] continueWithSuccessBlock:^id
 _Nullable(AWSTask<AWSCognitoIdentityUserVerifyAttributeResponse *> * _Nonnull task) {
    //success
    return nil;
}];
```

Authenticated Example: Update User Attributes

```
AWSCognitoIdentityUserAttributeType * attribute = [AWSCognitoIdentityUserAttributeType
 new];
attribute.name = @"name";
attribute.value = @"John User";
[[user updateAttributes:@[attribute]] continueWithSuccessBlock:^id
 _Nullable(AWSTask<AWSCognitoIdentityUserUpdateAttributesResponse *> * _Nonnull task) {
    //success
    return nil;
}];
```

Authenticated Example: Change Password

```
[[user changePassword:@"currentPassword" proposedPassword:@"proposedPassword"]
 continueWithSuccessBlock:^id
 _Nullable(AWSTask<AWSCognitoIdentityUserChangePasswordResponse *> * _Nonnull task) {
        //success
        return nil;
}];
```

Authenticated Example: Turning on MFA

```
AWSCognitoIdentityUserSettings * settings = [AWSCognitoIdentityUserSettings new];
AWSCognitoIdentityUserMFAOption * mfaOptions = [AWSCognitoIdentityUserMFAOption new];
mfaOptions.attributeName = @"phone_number";
mfaOptions.deliveryMedium = AWSCognitoIdentityProviderDeliveryMediumTypeSms;
settings.mfaOptions = @[mfaOptions];
[[user setUserSettings:settings] continueWithSuccessBlock:^id
 _Nullable(AWSTask<AWSCognitoIdentityUserSetUserSettingsResponse *> * _Nonnull task) {
    //success
    return nil;
}];
```

Setting up the Amazon Cognito Identity SDK for JavaScript to Work with User Pools

The Amazon Cognito Identity SDK for JavaScript allows JavaScript-enabled applications to register users; authenticate users; and view, delete, and update user attributes in Amazon Cognito user pools. Other functionality includes password changes for authenticated users and initiating and completing forgotten password flows for unauthenticated users.

To set up the Amazon Cognito Identity SDK for JavaScript, follow the instructions here: Amazon Cognito Identity SDK for JavaScript.

Tutorial: Integrating User Pools for JavaScript Apps

This tutorial helps you use the Amazon Cognito SDK for JavaScript to get started with user pools.

Topics

- Step 1: Creating a User Pool for your JavaScript App by Using the Console (p. 54)
- Step 2: Creating a User Pool Object in Your App (p. 55)
- Step 3: Signing up Users for Your App (p. 55)
- Step 4: Confirming Users for Your App (p. 56)
- Step 5: Signing Users in to Your App (p. 56)
- Step 6: Getting User Details (p. 57)
- Step 7: Getting Credentials to Access AWS Resources for an App User (p. 57)
- Next Steps (p. 57)

Step 1: Creating a User Pool for your JavaScript App by Using the Console

The following procedure describes how to create a user pool and use it in your app. This procedure creates a user pool ID and an app client ID. For information on customizing these settings, see Step Through Amazon Cognito User Pool Settings in the AWS Management Console (p. 13).

To create a user pool for your app

1. Sign in to the Amazon Cognito console
2. Choose **Manage your User Pools**.
3. Choose **Create a User Pool**.
4. In **Pool name**, type a name for the pool and then choose **Review defaults**. This creates the pool with the default settings.
5. From the left navigation pane, choose **Attributes** to specify which attributes are required and which attributes to use as aliases. After you set the following attributes and after users in the pool verify their email addresses, they can sign in with their usernames or email addresses.

    a. For **email**, choose **Required** and **Alias**.
    b. For **phone number**, choose **Required** and **Alias**.
    c. For **given name**, choose **Required**.
    d. Choose **Save changes**.
6. From the left navigation pane, choose **Policies** to specify the password policy. For this tutorial, use the default settings.

7. From the left navigation pane, choose **Verifications**. On this page, you can customize the messages that are sent to the users in your pool to deliver verification codes. For this tutorial, use the default settings.
8. From the left navigation pane, choose **Apps** and then choose **Add an app**. You can create multiple app clients for a user pool and you can create one app per platform.
9. For **App name**, type a name for your app. Ensure that the **Generate client secret** check box is cleared, and then choose **Set attribute read and write permissions**. Select the attributes that require write permissions. Required attributes always have write permissions.

   Note
   The Amazon Cognito JavaScript SDK does not use the app client secret. If you configure your user pool app client with an app client secret, the SDK will throw exceptions.
10. Choose **Create app** and then choose **Save changes**.
11. From the left navigation bar, choose **Review** and then choose **Create pool**.
12. Note the pool ID and client ID. You can find the app client ID under **Apps** on the left navigation bar.

Step 2: Creating a User Pool Object in Your App

To create a user pool object, you need the user pool ID and client ID that you obtained in step 1. The following example shows how to create a `CognitoUserPool` object. The JavaScript SDK does not support the app client secret. If you configure your user pool app client with an app client secret, the SDK will throw exceptions.

```
AWSCognito.config.region = 'us-east-1';

var poolData = {
    UserPoolId : '...', // your user pool id here
    ClientId : '...' // your app client id here
};
var userPool =
new AWSCognito.CognitoIdentityServiceProvider.CognitoUserPool(poolData);
var userData = {
    Username : '...', // your username here
    Pool : userPool
};
```

Step 3: Signing up Users for Your App

After creating a user pool object, users can be signed up for the app. The user's information can be collected through the web UI and used to populate `CognitoUserAttribute` objects that are passed in the `signUp` call.

```
var attributeList = [];

var dataEmail = {
    Name : 'email',
    Value : '...' // your email here
};
var dataPhoneNumber = {
    Name : 'phone_number',
    Value : '...' // your phone number here with +country code and no delimiters in front
};
var attributeEmail =
new AWSCognito.CognitoIdentityServiceProvider.CognitoUserAttribute(dataEmail);
var attributePhoneNumber =
new AWSCognito.CognitoIdentityServiceProvider.CognitoUserAttribute(dataPhoneNumber);

attributeList.push(attributeEmail);
attributeList.push(attributePhoneNumber);

var cognitoUser;
userPool.signUp('username', 'password', attributeList, null, function(err, result){
    if (err) {
        alert(err);
        return;
    }
    cognitoUser = result.user;
    console.log('user name is ' + cognitoUser.getUsername());
});
```

Step 4: Confirming Users for Your App

After signing up, the user confirms the sign-up by entering a code sent either through SMS or email (based on the user pool settings). Alternatively, you can use a `PreSignUp` AWS Lambda function to automatically confirm your users. To confirm sign-up, you must collect the code ('123456' in the following example) received by the user and use it as follows.

```
 cognitoUser.confirmRegistration('123456', true, function(err, result) {
    if (err) {
        alert(err);
        return;
    }
    console.log('call result: ' + result);
});
```

The registration code can be resent by using the `resendConfirmationCode` method of a `cognitoUser` object. This is an unauthenticated call and only the username, the client ID, and the user pool information are needed.

Step 5: Signing Users in to Your App

A confirmed user signs in to obtain a session. The session contains an ID token that contains user claims, an access token that is used internally to perform authenticated calls, and a refresh token that is used internally to refresh the session after it expires each hour. For more information about tokens, see Using Tokens with User Pools (p. 128). If sign in is successful, the `onSuccess` callback is called. If sign in fails, the `onFailure` callback is called. If sign in requires MFA, the `mfaRequired` callback is called and you must invoke `sendMFACode` on the `cognitoUser` object. The verification code that is received must be passed and the user is then signed in.

```


    var authenticationData = {
        Username : '...', // your username here
        Password : '...', // your password here
    };
    var authenticationDetails =
new AWSCognito.CognitoIdentityServiceProvider.AuthenticationDetails(authenticationData);

    var cognitoUser =
new AWSCognito.CognitoIdentityServiceProvider.CognitoUser(userData);
    cognitoUser.authenticateUser(authenticationDetails, {
        onSuccess: function (result) {
            console.log('access token + ' + result.getAccessToken().getJwtToken());
        },

        onFailure: function(err) {
            alert(err);
        },
        mfaRequired: function(codeDeliveryDetails) {
            var verificationCode = prompt('Please input verification code' ,'');
            cognitoUser.sendMFACode(verificationCode, this);
        }
    });
```

Step 6: Getting User Details

After signing in, a user can perform authorized operations such as retrieving user attributes, verifying user attributes (such as an unverified email address), deleting user attributes, updating user attributes, changing the user password, and deleting the user account. For user pools that have an optional MFA setting, users can enable or disable MFA for themselves. Signing out from the app clears the local user session and the user must sign in again to establish a new session.

If users forget their passwords, they can initiate a forgotten password flow. A code will be sent to the user. The user uses this code together with a new password to complete the flow. The relevant call is `forgotPassword` on a `cognitoUser` object that is unauthenticated; the relevant callbacks are shown in the following example.

```
cognitoUser.forgotPassword({
    onSuccess: function (result) {
        console.log('call result: ' + result);
    },
    onFailure: function(err) {
        alert(err);
    },
    inputVerificationCode() {
        var verificationCode = prompt('Please input verification code ' ,'');
        var newPassword = prompt('Enter new password ' ,'');
        cognitoUser.confirmPassword(verificationCode, newPassword, this);
    }
});
```

Step 7: Getting Credentials to Access AWS Resources for an App User

If you want to work with other AWS services, you must first create an Amazon Cognito identity pool (p. 135). After you create this identity pool, you can get AWS credentials by passing the identity pool ID and the ID token (which were obtained earlier) when signing in the user. The following example shows how to populate `IdentityPoolId` and pass the ID token through the `Logins` map.

```
 AWS.config.credentials = new AWS.CognitoIdentityCredentials({
    IdentityPoolId: 'us-east-1:XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX',
    Logins: {
        'cognito-idp.us-east-1.amazonaws.com/us-east-1_XXXXXXXXX':
 result.getIdToken().getJwtToken()
    }
});

AWS.config.credentials.get(function(err){
    if (err) {
        alert(err);
    }
});
```

Next Steps

For more examples and an overview of the code used in this tutorial, see the Amazon Cognito Identity JavaScript GitHub repository.

Examples: Using the JavaScript SDK

Register a User with the Application

You need to create a `CognitoUserPool` object by providing a `UserPoolId` and a `ClientId`, and registering by using a username, password, attribute list, and validation data.

```
   AWSCognito.config.region = 'us-east-1'; //This is required to derive the endpoint

   var poolData = { UserPoolId : 'us-east-1_TcoKGbf7n',
       ClientId : '4pe2usejqcdmhi0a25jp4b5sh3'
   };
   var userPool = new AWSCognito.CognitoIdentityServiceProvider.CognitoUserPool(poolData);

   var attributeList = [];

   var dataEmail = {
       Name : 'email',
       Value : 'email@mydomain.com'
   };
   var dataPhoneNumber = {
       Name : 'phone_number',
       Value : '+15555555555'
   };
   var attributeEmail = new
AWSCognito.CognitoIdentityServiceProvider.CognitoUserAttribute(dataEmail);
   var attributePhoneNumber = new
AWSCognito.CognitoIdentityServiceProvider.CognitoUserAttribute(dataPhoneNumber);

   attributeList.push(attributeEmail);
   attributeList.push(attributePhoneNumber);

   userPool.signUp('username', 'password', attributeList, null, function(err, result){
       if (err) {
           alert(err);
           return;
       }
       cognitoUser = result.user;
       console.log('user name is ' + cognitoUser.getUsername());
   });
```

Delete an Authenticated User

```
   cognitoUser.deleteUser(function(err, result) {
       if (err) {
           alert(err);
           return;
       }
       console.log('call result: ' + result);
   });
```

Retrieve the current user from local storage

```
   var data = { UserPoolId : 'us-east-1_Iqc12345',
       ClientId : '12345du353sm7khjj1q'
   };
   var userPool = new AWSCognito.CognitoIdentityServiceProvider.CognitoUserPool(data);
   var cognitoUser = userPool.getCurrentUser();

   if (cognitoUser != null) {
       cognitoUser.getSession(function(err, session) {
           if (err) {
               alert(err);
               return;
           }
           console.log('session validity: ' + session.isValid());
       });
   }
```

Authenticate a User

The following example authenticates a user and establishes a user session with the Amazon Cognito service.

> **Note**
> If the user was created by the administrator in the console, and not through the preceding JavaScript examples, see Example: Handling Users Created Using the AdminCreateUser API in the Mobile SDK for Android (p. 43) and Creating User Accounts as Administrator in the AWS Management Console and with the Amazon Cognito User Pools API (p. 103).

```
   var authenticationData = {
       Username : 'username',
       Password : 'password',
   };
   var authenticationDetails = new
AWSCognito.CognitoIdentityServiceProvider.AuthenticationDetails(authenticationData);
   var poolData = { UserPoolId : 'us-east-1_TcoKGbf7n',
       ClientId : '4pe2usejqcdmhi0a25jp4b5sh3'
   };
   var userPool = new AWSCognito.CognitoIdentityServiceProvider.CognitoUserPool(poolData);
   var userData = {
       Username : 'username',
       Pool : userPool
   };
   var cognitoUser = new AWSCognito.CognitoIdentityServiceProvider.CognitoUser(userData);
   cognitoUser.authenticateUser(authenticationDetails, {
       onSuccess: function (result) {
           console.log('access token + ' + result.getAccessToken().getJwtToken());
           /*Use the idToken for Logins Map when Federating User Pools with Cognito
Identity or when passing through an Authorization Header to an API Gateway Authorizer*/
           console.log('idToken + ' + result.idToken.jwtToken);
       },

       onFailure: function(err) {
           alert(err);
       },

   });
```

Enable MFA for a User Pool

The following example enables multi-factor authentication (MFA) for a user pool that has an optional MFA setting for an authenticated user.

```
   cognitoUser.enableMFA(function(err, result) {
       if (err) {
           alert(err);
           return;
       }
       console.log('call result: ' + result);
   });
```

Disable MFA for a User Pool

The following example disables multi-factor authentication (MFA) for a user pool that has an optional MFA setting for an authenticated user.

```
cognitoUser.disableMFA(function(err, result) {
    if (err) {
        alert(err);
        return;
    }
    console.log('call result: ' + result);
});
```

Create a User Pool Object

```
var data = { UserPoolId : 'us-east-1_q2Y6U8uuY',
            ClientId : '224kjog47ojnt9ov773erj7qn9'
};

var userPool = new AWSCognito.CognitoIdentityServiceProvider.CognitoUserPool(data);
```

Sign Up For the Application

```
           var attribute = {
               Name : 'email',
               Value : 'email@mydomain.com'
           };

           var attributeEmail = new
AWSCognito.CognitoIdentityServiceProvider.CognitoUserAttribute(attribute);
           var attributeList = [];

           attributeList.push(attributeEmail);
           var cognitoUser;

           userPool.signUp('username', 'password', attributeList, null, function(err,
result) {
               if (err) {
                   alert(err);
                   return;
               }
               cognitoUser = result.user;
           });
```

Sign in With MFA Enabled

```
           var userData = {
               Username : 'username',
               Pool : userPool
           };

           cognitoUser = new
AWSCognito.CognitoIdentityServiceProvider.CognitoUser(userData);

           var authenticationData = {
               Username : 'username',
               Password : 'password',
           };
           var authenticationDetails = new
AWSCognito.CognitoIdentityServiceProvider.AuthenticationDetails(authenticationData);

           cognitoUser.authenticateUser(authenticationDetails, {
               onSuccess: function (result) {
                   alert('authentication successful!')
               },

               onFailure: function(err) {
                   alert(err);
               },

               mfaRequired: function(codeDeliveryDetails) {
                   var verificationCode = prompt('Please input verification code' ,'');
                   cognitoUser.sendMFACode(verificationCode, this);
               }

           });
```

Update Attributes

The following example updates user attributes for an authenticated user.

```
   var attributeList = [];
   var attribute = {
       Name : 'nickname',
       Value : 'joe'
   };
   var attribute = new
AWSCognito.CognitoIdentityServiceProvider.CognitoUserAttribute(attribute);
   attributeList.push(attribute);

   cognitoUser.updateAttributes(attributeList, function(err, result) {
       if (err) {
           alert(err);
           return;
       }
       console.log('call result: ' + result);
   });
```

Delete Attributes

The following example deletes user attributes for an authenticated user.

```
   var attributeList = [];
   attributeList.push('nickname');

   cognitoUser.deleteAttributes(attributeList, function(err, result) {
       if (err) {
           alert(err);
           return;
       }
       console.log('call result: ' + result);
   });
```

Verify an Attribute

The following example verifies user attributes for an authenticated user.

```
   cognitoUser.getAttributeVerificationCode('email', {
       onSuccess: function (result) {
           console.log('call result: ' + result);
       },
       onFailure: function(err) {
           alert(err);
       },
       inputVerificationCode: function() {
           var verificationCode = prompt('Please input verification code: ' ,'');
           cognitoUser.verifyAttribute('email', verificationCode, this);
       }
   });
```

Retrieve Attributes

The following example retrieves user attributes for an authenticated user.

```
   cognitoUser.getUserAttributes(function(err, result) {
       if (err) {
           alert(err);
           return;
       }
       for (i = 0; i < result.length; i++) {
           console.log('attribute ' + result[i].getName() + ' has value ' +
result[i].getValue());
       }
   });
```

Resend a Confirmation Code

The following example resends a confirmation code via SMS that confirms the registration for an unauthenticated user.

```
           cognitoUser.resendConfirmationCode(function(err, result) {
               if (err) {
                   alert(err);
                   return;
                  }
                  alert(result);
           });
```

Confirm Registration

```
           cognitoUser.confirmRegistration('123456', true, function(err, result) {
               if (err) {
                   alert(err);
                   return;
               }
               alert(result);
           });
```

Change a Password

The following example changes the current password of an authenticated user.

```
   cognitoUser.changePassword('oldPassword', 'newPassword', function(err, result) {
       if (err) {
           alert(err);
           return;
       }
       console.log('call result: ' + result);
```

```
    });
```

Forgotten Password Flow

The following example starts and completes a forgotten password flow for an unauthenticated user.

```
    cognitoUser.forgotPassword({
        onSuccess: function (result) {
            console.log('call result: ' + result);
        },
        onFailure: function(err) {
            alert(err);
        },
        inputVerificationCode() {
            var verificationCode = prompt('Please input verification code ' ,'');
            var newPassword = prompt('Enter new password ' ,'');
            cognitoUser.confirmPassword(verificationCode, newPassword, this);
        }
    });
```

Delete a User

The following example deletes an authenticated user.

```
    cognitoUser.deleteUser(function(err, result) {
        if (err) {
            alert(err);
            return;
        }
        console.log('call result: ' + result);
    });
```

Sign a User Out

The following example signs the current user out from the application.

```
    if (cognitoUser != null) {
          cognitoUser.signOut();
        }
```

Sign a User Out Globally

The following example signs the current user out globally by invalidating all issued tokens.

```
    cognitoUser.globalSignOut();
```

Get the Current User

The following example retrieves the current user from local storage.

```
    var data = {
        UserPoolId : '...', // Your user pool id here
        ClientId : '...' // Your client id here
    };
    var userPool = new AWSCognito.CognitoIdentityServiceProvider.CognitoUserPool(data);
    var cognitoUser = userPool.getCurrentUser();
```

```
   if (cognitoUser != null) {
       cognitoUser.getSession(function(err, session) {
           if (err) {
              alert(err);
               return;
           }
           console.log('session validity: ' + session.isValid());

           AWS.config.credentials = new AWS.CognitoIdentityCredentials({
               IdentityPoolId : '...' // your identity pool id here
               Logins : {
                   // Change the key below according to the specific region your user pool
is in.
                   'cognito-idp.<region>.amazonaws.com/<YOUR_USER_POOL_ID>' :
session.getIdToken().getJwtToken()
               }
           });

           // Instantiate aws sdk service objects now that the credentials have been
updated.
           // example: var s3 = new AWS.S3();

       });
   }
```

Integrate a User in a User Pool with an Identity Pool

The following example integrates the current user in a user pool with the specified identity pool.

```
   var cognitoUser = userPool.getCurrentUser();

   if (cognitoUser != null) {
       cognitoUser.getSession(function(err, result) {
           if (result) {
               console.log('You are now logged in.');

               // Add the User's Id Token to the Cognito credentials login map.
               AWS.config.credentials = new AWS.CognitoIdentityCredentials({
                   IdentityPoolId: 'YOUR_IDENTITY_POOL_ID',
                   Logins: {
                       'cognito-idp.<region>.amazonaws.com/<YOUR_USER_POOL_ID>':
result.getIdToken().getJwtToken()
                   }
               });
           }
       });
   }
   //call refresh method in order to authenticate user and get new temp credentials
   AWS.config.credentials.refresh((error) => {
       if (error) {
           console.error(error);
       } else {
           console.log('Successfully logged!');
       }
       });
```

List All Devices for a User

The following example lists all devices for an authenticated user. In this case, we need to pass a limit on the number of devices retrieved at a time. In the first call, the pagination token should be null. The first call returns a pagination token, which should be passed in all subsequent calls.

```
cognitoUser.listDevices(limit, paginationToken, {
    onSuccess: function (result) {
        console.log('call result: ' + result);
    },
    onFailure: function(err) {
        alert(err);
    }
});
```

List Device Information

The following example lists information about the current device.

```
cognitoUser.listDevices(limit, paginationToken, {
    onSuccess: function (result) {
        console.log('call result: ' + result);
    },
    onFailure: function(err) {
        alert(err);
    }
});
```

Remember a Device

The following example remembers a device.

```
cognitoUser.setDeviceStatusRemembered({
    onSuccess: function (result) {
        console.log('call result: ' + result);
    },
    onFailure: function(err) {
        alert(err);
    }
});
```

Do Not Remember a Device

The following example marks a device as not to be remembered.

```
cognitoUser.setDeviceStatusNotRemembered({
    onSuccess: function (result) {
        console.log('call result: ' + result);
    },
onFailure: function(err) {
        alert(err);
    }
});
```

Do Not Remember a Device

The following example forgets the current device.

```
cognitoUser.forgetDevice({
    onSuccess: function (result) {
        console.log('call result: ' + result);
    },
        onFailure: function(err) {
            alert(err);
        }
    });
```

Confirm a Registered, Unauthenticated User

The following example confirms a registered, unauthenticated user using a confirmation code received via SMS message.

```
    var poolData = {
        UserPoolId : 'us-east-1_TcoKGbf7n',
        ClientId : '4pe2usejqcdmhi0a25jp4b5sh3'
    };

    var userPool = new AWSCognito.CognitoIdentityServiceProvider.CognitoUserPool(poolData);
    var userData = {
        Username : 'username',
        Pool : userPool
    };

    var cognitoUser = new AWSCognito.CognitoIdentityServiceProvider.CognitoUser(userData);
    cognitoUser.confirmRegistration('123456', true, function(err, result) {
        if (err) {
            alert(err);
            return;
        }
        console.log('call result: ' + result);
    });
```

Example: Authenticate and Set a New Password for a User Created Using the AdminCreateUser API

To support the user sign-in flow for users created by administrators (using the `AdminCreateUser` API), implement a `newPasswordRequired` callback method to set the new password when the user first signs in. The user first attempts to sign in with the temporary password he or she received in the invitation and the SDK calls your `newPasswordRequired` callback. Gather the required inputs, including the new password and required attributes, and then call the `completeNewPasswordChallenge` method, which is available in the `CognitoUser` class.

The `newPasswordRequired` callback takes two parameters: `userAttributes` and `requiredAttributes`.

```
cognitoUser.authenticateUser(authenticationDetails, {
        onSuccess: function (result) {
            // User authentication was successful
        },

        onFailure: function(err) {
            // User authentication was not successful
        },

        mfaRequired: function(codeDeliveryDetails) {
            // MFA is required to complete user authentication.
            // Get the code from user and call
            cognitoUser.sendMFACode(mfaCode, this)
        },

        newPasswordRequired: function(userAttributes, requiredAttributes) {
            // User was signed up by an admin and must provide new
            // password and required attributes, if any, to complete
            // authentication.

            // userAttributes: object, which is the user's current profile. It will list all attributes that are associated with the user.
            // Required attributes according to schema, which don’t have any values yet, will have blank values.
            // requiredAttributes: list of attributes that must be set by the user along with new password to complete the sign-in.


            // Get these details and call
            // newPassword: password that user has given
            // attributesData: object with key as attribute name and value that the user has given.
            cognitoUser.completeNewPasswordChallenge(newPassword, attributesData, this)
        }
    });
```

Customizing User Pool Workflows by Using AWS Lambda Triggers

You can use AWS Lambda triggers with Amazon Cognito user pools to customize workflows at various stages in the lifecycle of a user account. For more information, see the *AWS Lambda Developer Guide*.

> **Important**
> When called, your Lambda function must respond within 5 seconds. If it does not, Amazon Cognito retries the call. After 3 unsuccessful attempts, Amazon Cognito times out. This 5-second timeout value cannot be changed.

The following Lambda triggers are available for you to implement:

Pre sign-up

Amazon Cognito invokes this trigger when a user attempts to register (sign up), allowing you to perform custom validation to accept or deny the registration request.

Pre authentication

Amazon Cognito invokes this trigger when a user attempts to authenticate (sign in), allowing you to perform custom validation to accept or deny the authentication request.

Custom message

Amazon Cognito invokes this trigger before sending an email or phone verification message or a multi-factor authentication (MFA) code, allowing you to customize the message dynamically. Static custom messages can be edited in the **Message Customizations** tab of the Amazon Cognito console.

Post authentication

Amazon Cognito invokes this trigger after authenticating a user, allowing you to add custom logic.

Post confirmation

Amazon Cognito invokes this trigger after a user is confirmed, allowing you to send custom messages or to add custom logic. For example, you may want to implement analytics in your app.

Define Auth Challenge

Amazon Cognito invokes this trigger to initiate the custom authentication flow.

Create Auth Challenge

Amazon Cognito invokes this trigger after **Define Auth Challenge** if a custom challenge has been specified as part of the **Define Auth Challenge** trigger.

Verify Auth Challenge Response

Amazon Cognito invokes this trigger to verify if the response from the end user for a custom Auth Challenge is valid or not.

Creating an AWS Lambda Trigger for a Stage

You can create an AWS Lambda function and then associate that function with one of the user account life cycle stages to create a Lambda trigger.

To add a Lambda function to a user stage

1. If you haven't done so already, create a Lambda function using the Lambda console.
2. Navigate to the Amazon Cognito console, choose **Manage User Pools**, and then choose the user pool to add the Lambda function to.
3. In your user pool, choose the **Triggers** tab.
4. Associate a user stage with a Lambda function by choosing the function from the drop-down menu for that stage, and then save your changes.

AWS Lambda Trigger Request and Response Parameters

This section describes the AWS Lambda trigger request and response parameters.

Topics

- AWS Lambda Trigger Common Parameters (p. 68)
- Pre Sign-up Lambda Parameters (p. 70)
- Pre-Authentication Lambda Parameters (p. 71)
- Custom Message Lambda Parameters (p. 72)
- Post-Authentication Lambda Parameters (p. 72)
- Post-Confirmation Lambda Parameters (p. 73)
- Define Auth Challenge Lambda Parameters (p. 73)
- Create Auth Challenge Lambda Parameters (p. 74)
- Verify Auth Challenge Response Lambda Parameters (p. 76)

AWS Lambda Trigger Common Parameters

The event information passed to the invoked Lambda function contains the parameters that were passed from the Amazon Cognito service. The general format of the event is shown next. The request and the response parameters depend on the Lambda trigger.

```
{
    "version": number,
    "triggerSource": "string",
    "region": AWSRegion,
    "userPoolId": "string",
    "callerContext":
        {
            "awsSdkVersion": "string",
            "clientId": "string"
        },
    "request":
        {
            "userAttributes": {
                "string": "string",
                ....
            }
        },
    "response": {}
}
```

version

The version number of your Lambda function.

triggerSource

The name of the event that triggered the Lambda function. The following table shows the `triggerSource` values and the triggering event for each value.

`triggerSource` value	Triggering event
`PreSignUp_SignUp`	Pre-sign up
`PostConfirmation_ConfirmSignUp`	Post sign-up confirmation
`PostConfirmation_ConfirmForgotPassword`	Post Forgot Password confirmation
`PreAuthentication_Authentication`	Pre authentication
`PostAuthentication_Authentication`	Post authentication
`CustomMessage_SignUp`	Custom message – To send the confirmation code post sign-up
`CustomMessage_AdminCreateUser`	Custom message – To send the temporary password to a new user
`CustomMessage_ResendCode`	Custom message – To resend the confirmation code to an existing user
`CustomMessage_ForgotPassword`	Custom message – To send the confirmation code for Forgot Password request
`CustomMessage_UpdateUserAttribute`	Custom message – When a user's email or phone number is changed, this trigger sends a verification code automatically to the user. Cannot be used for other attributes.
`CustomMessage_VerifyUserAttribute`	Custom message – This trigger sends a verification code to the user when the they manually request it for a new email or phone number.
`CustomMessage_Authentication`	Custom message – To send MFA code during authentication

`triggerSource` value	Triggering event
`DefineAuthChallenge_Authentication`	Define Auth Challenge
`CreateAuthChallenge_Authentication`	Create Auth Challenge
`VerifyAuthChallengeResponse_Authentication`	Verify Auth Challenge Response

region

The AWS Region, as an `AWSRegion` instance.

userPoolId

The user pool ID for the user pool.

callerContext

The caller context, which consists of the following:

awsSdkVersion

The AWS SDK version number.

clientId

The ID of the client associated with the user pool.

request

The request from the Amazon Cognito service. This request must include:

userAttributes

One or more pairs of user attribute names and values. Each pair is in the form "*`name`*": "*`value`*".

response

The response from your Lambda trigger. The return parameters depend on the triggering event.

Pre Sign-up Lambda Parameters

The request includes validation data from the client.

```
"request": {
    "userAttributes": {
        "string": "string",
        ....
    },
    "validationData": {<validation data as key-value (String,
                String) pairs, from the client>}
}
```

userAttributes

One or more name-value pairs representing user attributes. The attribute names are the keys.

validationData

One or more name-value pairs containing the validation data in the request to register a user. The validation data is set and then passed from the client in the request to register a user.

In the response, you can set `autoConfirmUser` to `true` if you want to auto-confirm the user. You can set `autoVerifyEmail` to `true` to auto-verify the user's email. You can set `autoVerifyPhone` to `true` to auto-verify the user's phone number.

```
"response": {
    "autoConfirmUser": boolean
    "autoVerifyEmail": boolean
    "autoVerifyPhone": boolean
}
```

autoConfirmUser

Set to `true` to auto-confirm the user, or `false` otherwise.

autoVerifyEmail

Set to `true` to set as verified the email of a user who is signing up, or `false` otherwise. If `autoVerifyEmail` is set to `true`, the `email` attribute must have a valid, non-null value. Otherwise an error will occur and the user will not be able to complete sign-up.

If the `email` attribute is selected as an alias, an alias will be created for the user's email when `autoVerifyEmail` is set. If an alias with that email already exists, the alias will be moved to the new user and the previous user's email will be marked as unverified. For more information, see Overview of Aliases (p. 16).

autoVerifyPhone

Set to `true` to set as verified the phone number of a user who is signing up, or `false` otherwise. If `autoVerifyPhone` is set to `true`, the `phone_number` attribute must have a valid, non-null value. Otherwise an error will occur and the user will not be able to complete sign-up.

If the `phone_number` attribute is selected as an alias, an alias will be created for the user's phone number when `autoVerifyPhone` is set. If an alias with that phone number already exist, the alias will be moved to the new user and the previous user's phone number will be marked as unverified. For more information, see Overview of Aliases (p. 16).

Pre-Authentication Lambda Parameters

The request includes validation data from the client.

```
"request": {
    "userAttributes": {
        "string": "string",
        ....
    },
    "validationData": {<validation data as key-value (String,
                String) pairs, from the client>}
}
```

userAttributes

One or more name-value pairs representing user attributes.

validationData

One or more key-value pairs containing the validation data in the user's sign-in request.

No return information is expected in the response.

```
"response": {
}
```

Custom Message Lambda Parameters

The request includes `codeParameter`, which is a string that acts as a placeholder for the code that's being delivered to the user. Insert the `codeParameter` string into the message body, at the position where you want the verification code to be inserted. On receiving this response, the Amazon Cognito service replaces the `codeParameter` string with the actual verification code.

```
"request": {
    "userAttributes": {
        "string": "string",
        ....
    },
    "codeParameter": "string"
}
```

userAttributes

One or more name-value pairs representing user attributes.

codeParameter

A string for you to use as the placeholder for the verification code in the custom message.

In the response, you specify the custom text to use in messages to your users.

```
"response": {
    "smsMessage": "string",
    "emailMessage": "string",
    "emailSubject": "string";
}
```

smsMessage

The custom SMS message to be sent to your users. Must include the `codeParameter` value received in the request.

emailMessage

The custom email message to be sent to your users. Must include the `codeParameter` value received in the request.

emailSubject

The subject line for the custom message.

Post-Authentication Lambda Parameters

The request includes the following:

- `newDeviceUsed` flag – indicates if the user has signed in on a new device
- `challengeSession` structure – indicates what authentication challenges the user faced
- `groupConfiguration` structure – contains group-related information that can override group-related claims in the identity token

The `newDeviceUsed` flag is set only if the remembered devices value of the user pool is set to `Always` or `User Opt-In`.

```
"request": {
    "userAttributes": {
        "string": "string",
        ....
    },
    "newDeviceUsed": boolean
}
```

userAttributes

One or more name-value pairs representing user attributes.

validationData

One or more key-value pairs containing the validation data in the request to register a user.

No return information is expected in the response.

```
"response": {
}
```

Post-Confirmation Lambda Parameters

The request contains the current attributes for the confirmed user.

```
"request": {
    "userAttributes": {
        "string": "string",
        ....
    }
}
```

userAttributes

One or more name-value pairs representing user attributes.

No return information is expected in the response.

```
"response": {
}
```

Define Auth Challenge Lambda Parameters

The request contains `session`, which is an array containing all of the challenges that are presented to the user in the authentication process that is underway, along with the corresponding result. The challenge details (`ChallengeResult`) are stored in chronological order in the `session` array, with `session[0]` representing the first challenge that is presented to the user.

```
"request": {
    "userAttributes": {
        "string": "string",
        ....
    },
    "session": {
        [
            ChallengeResult
        ]
    }
}
```

userAttributes

One or more name-value pairs representing user attributes.

session

The session element is an array of `ChallengeResult` elements, each of which contains the following elements:

challengeName

The challenge type. One of: `"CUSTOM_CHALLENGE"`, `"PASSWORD_VERIFIER"`, `"SMS_MFA"`, `"DEVICE_SRP_AUTH"`, `"DEVICE_PASSWORD_VERIFIER"`, or `"ADMIN_NO_SRP_AUTH"`.

challengeResult

Set to `true` if the user successfully completed the challenge, or `false` otherwise.

challengeMetaData

Your name for the custom challenge. Used only if `challengeName` is `"CUSTOM_CHALLENGE"`.

In the response you can return the next stage of the authentication process.

```
"response": {
    "challengeName": "string",
    "issueTokens": boolean,
    "failAuthentication": boolean
}
```

challengeName

A string containing the name of the next challenge. If you want to present a new challenge to your user, specify the challenge name here.

issueTokens

Set to `true` if you determine that the user has sufficiently authenticated by completing the challenges, or `false` otherwise.

failAuthentication

Set to `true` if you want to terminate the current authentication process, or `false` otherwise.

Create Auth Challenge Lambda Parameters

This Lambda trigger is invoked to create a challenge to present to the user. The request for this Lambda trigger includes the `challengeName` and `session`. The `challengeName` is a string and is the name of the next challenge to the user. The value of this attribute is set in the Define Auth Challenge Lambda trigger.

```
"request": {
    "userAttributes": {
        "string": "string",
        ....
    },
    "challengeName": "string",
    "session": {
        [
            ChallengeResult
        ]
    }
}
```

userAttributes

One or more name-value pairs representing user attributes.

challengeName

The name of the new challenge.

session

The session element is an array of `ChallengeResult` elements, each of which contains the following elements:

challengeName

The challenge type. One of: `"CUSTOM_CHALLENGE"`, `"PASSWORD_VERIFIER"`, `"SMS_MFA"`, `"DEVICE_SRP_AUTH"`, `"DEVICE_PASSWORD_VERIFIER"`, or `"ADMIN_NO_SRP_AUTH"`.

challengeResult

Set to `true` if the user successfully completed the challenge, or `false` otherwise.

challengeMetaData

Your name for the custom challenge. Used only if `challengeName` is `"CUSTOM_CHALLENGE"`.

The challenge parameters for the new challenge are added to the response.

```
"response": {
    "publicChallengeParameters": {
        "string": "string",
        ....
    },
    "privateChallengeParameters": {
        "string": "string",
        ....
    },
    "challengeMetadata": "string"
}
```

publicChallengeParameters

One or more key-value pairs for the client app to use in the challenge to be presented to the user. This parameter should contain all of the necessary information to accurately present the challenge to the user.

privateChallengeParameters

This parameter is only used by the Verify Auth Challenge Response Lambda trigger. This parameter should contain all of the information that is required to validate the user's response to the challenge. In other words, the `publicChallengeParameters` parameter contains the question that is presented to the user and `privateChallengeParameters` contains the valid answers for the question.

challengeMetadata

Your name for the custom challenge, if this is a custom challenge.

Verify Auth Challenge Response Lambda Parameters

The request for this trigger contains the `privateChallengeParameters` and `challengeAnswer` parameters. The `privateChallengeParameters` values are returned by the Create Auth Challenge Lambda trigger and will contain the expected response from the user. The `challengeAnswer` parameter contains the user's response for the challenge.

```
"request": {
    "userAttributes": {
        "string": "string",
        ....
    },
    "privateChallengeParameters": {
        "string": "string",
        ....
    },
    "challengeAnswer": {
        "string": "string",
        ....
    }
}
```

userAttributes

One or more name-value pairs representing user attributes.

privateChallengeParameters

This parameter is only used by the Verify Auth Challenge Response Lambda trigger. This parameter should contain all of the information that is required to validate the user's response to the challenge. In other words, the `publicChallengeParameters` parameter contains the question that is presented to the user and `privateChallengeParameters` contains the valid answers for the question.

challengeAnswer

The answer in the user's response to the challenge.

The response contains the `answerCorrect` attribute, which is set to `true` if the user successfully completed the challenge, or `false` otherwise.

```
"response": {
    "answerCorrect": boolean
}
```

answerCorrect

Set to `true` if the user has successfully completed the challenge, or `false` otherwise.

AWS Lambda Trigger Examples

This section gives code examples for each type of AWS Lambda trigger.

Topics

- Pre Sign-up Example #1: Auto-Confirm the User (p. 77)
- Pre Sign-up Example #2: Auto-Confirm and Auto-Verify the User (p. 78)
- Pre Authentication Example (p. 78)
- Custom Message Example (p. 79)
- Post Authentication Example (p. 80)
- Post Confirmation Example (p. 81)
- Define Auth Challenge Example (p. 82)
- Create Auth Challenge Example (p. 82)
- Verify Auth Challenge Response Example (p. 83)

Pre Sign-up Example #1: Auto-Confirm the User

The following example initializes just before the service starts the new user registration process. With this Lambda function, you can add custom logic to validate, filter, or restrict the types of user accounts that can be registered. For example, you may only want to allow users to register if they have been invited to join the service. This example uses the `autoConfirmUser` flag to indicate whether to auto-confirm a user to the user pool.

```
exports.handler = function(event, context) {
    // This Lambda function returns a flag to indicate if a user should be auto-confirmed.

    // Perform any necessary validations.

    // Impose a condition that the minimum length of the username of 5 is imposed on all
 user pools.
    if (event.userName.length < 5) {
        var error = new Error('failed!');
        context.done(error, event);
    }

    // Access your resource which contains the list of emails of users who were invited to
 sign up

    // Compare the list of email IDs from the request to the approved list
    if(event.userPoolId === "yourSpecialUserPool") {
        if (event.request.userAttributes.email in listOfEmailsInvited) {
                event.response.autoConfirmUser = true;
        }
    }
    // Return result to Cognito
    context.done(null, event);
};


{
  "version": 1,
  "triggerSource": "PreSignUp_SignUp",
  "region": "<region>",
  "userPoolId": "<userPoolId>",
  "userName": "<userName>",
  "callerContext": {
      "awsSdk": "<calling aws sdk with version>",
      "clientId": "<apps client id>",
      ...
  },
  "request": {
      "userAttributes": {
          "email": "<email>",
            "phone_number": "<phone_number>",
            ...
         },
        "validationData": {
            "k1": "v1",
            "k2": "v2",
            ...
         }
    },
    "response": {
        "autoConfirmUser": false
    }
}
```

Pre Sign-up Example #2: Auto-Confirm and Auto-Verify the User

The following example is similar to Example #1, except that it automatically confirms the user and automatically sets the user's `email` and `phone_number` attributes to verified if the attribute is present.

If aliasing is enabled, aliases will be created for `phone_number` and `email` when auto-verify is set. If an alias with the same phone number already exists, the alias will be moved to the new user, and the previous user's `phone_number` will be marked as unverified. The same is true for email addresses. If this is not the desired behavior, you can use the `ListUsers` API to see if there is an existing user who is already using the new user's phone number or email address as an alias.

```
exports.handler = (event, context, callback) => {
    //console.log('Received event:', JSON.stringify(event, null, 2));

    // TODO Confirm this is a user that should be auto-confirmed, e.g., has a valid token
 in event.request.validationData
    var autoConfirm = true;

    if (autoConfirm) {
        event.response.autoConfirmUser = true;
        console.log('Auto confirmed user');

        if (event.request.userAttributes.hasOwnProperty("email")) {
            event.response.autoVerifyEmail = true;
            console.log('Set email verified');
        }

        if (event.request.userAttributes.hasOwnProperty("phone_number")) {
            event.response.autoVerifyPhone = true;
            console.log('Set phone_number verified');
        }
    }

    callback(null, event);
};
```

Pre Authentication Example

This sample function restricts users from a specific app client ID from authenticating.

```
exports.handler = function(event, context) {
    if (event.callerContext.clientId === "<client id to be blocked>") {
        var error = new Error('Cannot authenticate users from this client');
        context.done(error, event);
    }
    context.done(null, event);
};
```

Sample event parameter:

```
{
  "version": 1,
  "triggerSource": "PreAuthentication_Authentication",
  "region": "<region>",
  "userPoolId": "<userPoolId>",
  "userName": "<userName>",
  "callerContext": {
      "awsSdkVersion": "<calling AWS sdk with version>",
      "clientId": "<apps client id>",
      ...
  },
  "request": {
      "userAttributes": {
          "phone_number_verified": false,
          "email_verified": false,
          ... // All custom attributes
      },
      "validationData": {
          "k1": "v1",
          "k2": "v2",
          ...
       }
  },
  "response": {}
}
```

Custom Message Example

This Lambda function is invoked when the service requires an app to send a verification code to the user. This function is used to customize the messages that are sent to deliver the code.

This Lambda trigger is invoked at multiple points: post-registration; resending a verification code; forgotten password; or verifying a user attribute. The response includes messages for both SMS and email. The message must include the code parameter, `{####}`, which is the placeholder for the verification code that is delivered to the user.

For email, the maximum length for the message is 20,000 UTF-8 characters, including the verification code. HTML tags can be used in these emails. For SMS, the maximum length is 140 UTF-8 characters, including the verification code.

```
exports.handler = function(event, context) {
    //
    if(event.userPoolId === "theSpecialUserPool") {
        // Identify why was this function invoked
        if(event.triggerSource === "CustomMessage_SignUp") {
            // Ensure that your message contains event.request.codeParameter. This is the
 placeholder for code that will be sent
            event.response.smsMessage = "Welcome to the service. Your confirmation code is
 " + event.request.codeParameter;
            event.response.emailSubject = "Welcome to the service";
            event.response.emailMessage = "Thank you for signing up. " +
 event.request.codeParameter + " is your verification code";
        }
        // Create custom message for other events
    }
    // Customize messages for other user pools
    //

    // Return result to Cognito
    context.done(null, event);
};
```

Sample event parameter:

```
{
  "version": 1,
  "triggerSource": "CustomMessage_SignUp/CustomMessage_ResendCode/
CustomMessage_ForgotPassword/CustomMessage_VerifyUserAttribute",
  "region": "<region>",
  "userPoolId": "<userPoolId>",
  "userName": "<userName>",
  "callerContext": {
      "awsSdk": "<calling aws sdk with version>",
      "clientId": "<apps client id>",
      ...
  },
  "request": {
      "userAttributes": {
          "phone_number_verified": false,
          "email_verified": true,
           ...
      },
      "codeParameter": "####"
  },
  "response": {
      "smsMessage": "<custom message to be sent in the message with code parameter>"
      "emailMessage": "<custom message to be sent in the message with code parameter>"
      "emailSubject": "<custom email subject>"
  }
}
```

Post Authentication Example

This function is invoked after a user is successfully authenticated. This sample function logs in to the console after a user is authenticated.

```
exports.handler = (event, context, callback) => {
    console.log('User authenticated: User-Pool', event.userPoolId+", UserId:" +
 event.userName);
    // Return result to Amazon Cognito
    context.done(null, event);
};
```

Sample event parameter:

```
{
  "version": 1,
  "triggerSource": "PostAuthentication_Authentication",
  "region": "<region>",
  "userPoolId": "<userPoolId>",
  "userName": "<userName>",
  "callerContext": {
      "awsSdk": "<calling aws sdk with version>",
      "clientId": "<apps client id>",
      ...
  },
  "request": {
      "userAttributes": {
          "phone_number_verified": true,
          "email_verified": true,
          ... //all custom attributes
      }
  },
  "response": {}
};
```

Post Confirmation Example

The example sends an email message to inform the user that he or she has been confirmed.

```
var aws = require('aws-sdk');

var ses = new aws.SES();

exports.handler = function(event, context) {
    console.log(event);

    if (event.request.userAttributes.email) {
            sendEmail(event.request.userAttributes.email, "Congratulations "+event.userName
+", you have been confirmed: ", function(status) {
            context.done(null, event);
        });
    } else {
        // Nothing to do, the user's email ID is unknown
        context.done(null, event);
    }
};

function sendEmail(to, body, completedCallback) {
    var eParams = {
        Destination: {
            ToAddresses: [to]
        },
        Message: {
            Body: {
                Text: {
                    Data: body
                }
            },
            Subject: {
                Data: "Cognito Identity Provider registration completed"
            }
        },
        Source: "<source_email>"
    };

    var email = ses.sendEmail(eParams, function(err, data){
        if (err) {
            console.log(err);
        } else {
            console.log("===EMAIL SENT===");
        }
        completedCallback('Email sent');
    });
    console.log("EMAIL CODE END");
};
```

Sample event parameter:

```
{
  "version": 1,
  "triggerSource": "PostConfirmation_ConfirmSignUp",
  "region": "<region>",
  "userPoolId": "<userPoolId>",
  "userName": "<userName>",
  "callerContext": {
      "awsSdk": "<calling aws sdk with version>",
      "clientId": "<apps client id>",
      ...
  },
  "request": {
      "userAttributes" : {
          "email": "<email>",
          "phone_number": "<phone_number>",
          ...
      }
  },
  "response": {}
}
```

Define Auth Challenge Example

This example defines a series of challenges for authentication and issues tokens only if all of the challenges are successfully completed.

```
exports.handler = function(event, context) {
    if (event.request.session.length == 1 && event.request.session[0].challengeName ==
 'SRP_A') {
        event.response.issueTokens = false;
        event.response.failAuthentication = false;
        event.response.challengeName = 'PASSWORD_VERIFIER';
    } else if (event.request.session.length == 2 && event.request.session[1].challengeName
 == 'PASSWORD_VERIFIER' && event.request.session[1].challengeResult == true) {
        event.response.issueTokens = false;
        event.response.failAuthentication = false;
        event.response.challengeName = 'CUSTOM_CHALLENGE';
    } else if (event.request.session.length == 3 && event.request.session[2].challengeName
 == 'CUSTOM_CHALLENGE' && event.request.session[2].challengeResult == true) {
        event.response.issueTokens = true;
        event.response.failAuthentication = false;
    } else {
        event.response.issueTokens = false;
        event.response.failAuthentication = true;
    }
    context.done(null, event);
}
```

Create Auth Challenge Example

A CAPTCHA is created as a challenge to the user. The URL for the CAPTCHA image is added to the public challenge parameters as "`captchaUrl`", and the expected answer is added to the private challenge parameters.

```
exports.handler = function(event, context) {
    if (event.request.challengeName == 'CUSTOM_CHALLENGE') {
        event.response.publicChallengeParameters = {};
        event.response.publicChallengeParameters.captchaUrl = 'url/123.jpg'
        event.response.privateChallengeParameters = {};
        event.response.privateChallengeParameters.answer = '5';
        event.response.challengeMetadata = 'CAPTCHA_CHALLENGE';
    }
    context.done(null, event);
}
```

Verify Auth Challenge Response Example

In this example, the Lambda function checks whether the user's response to a challenge matches the expected response. The `answerCorrect` parameter is set to `true` if the user's response matches the expected response.

```
exports.handler = function(event, context) {
    if (event.request.privateChallengeParameters.answer == event.request.challengeAnswer) {
        event.response.answerCorrect = true;
    } else {
        event.response.answerCorrect = false;
    }

    context.done(null, event);
}
```

Integrating Mobile and Web Apps into Amazon Cognito User Pools

Topics

- Specifying Identity Provider Settings for Your User Pool App (p. 83)
- Assigning a Domain to Your User Pool (p. 86)
- Specifying App UI Customization Settings for Your User Pool (p. 87)
- Defining Resource Servers for Your User Pool (p. 90)

Specifying Identity Provider Settings for Your User Pool App

You can use the AWS Management Console, or the AWS CLI or API, to set up an identity provider for your user pool's existing app client. Or you can specify the identity provider settings when you create a new app client.

About App Identity Settings

Choose the following app client settings:

Enabled Identity Providers

Before you can use an identity provider in your app client, you need to enable it first. You can enable more than one identity provider for an app, but you must enable at least one.

Callback URL(s)

A callback URL specifies where the user is to be redirected upon successful sign-in. You must specify at least one callback URL.

Note
This URL must match the `app_redirect` value that you specify in your app.

Sign out URL(s)

A sign-out URL specifies where the user is to be redirected when he or she signs out. You must specify at least one sign-out URL.

Allowed OAuth Flows

The **Authorization code grant** flow initiates a code grant flow, which provides an authorization code as the response. This code can be exchanged for access tokens with the TOKEN Endpoint (p. 239).

Note
For security reasons, we highly recommend that you use only the **Authorization code grant** flow, together with PKCE, for mobile apps.

The **Implicit grant** flow allows the client to get the access token (and, optionally, ID token, based on scopes) directly from the AUTHORIZATION Endpoint (p. 235). Choose this flow if your app cannot initiate the **Authorization code grant** flow. For more information, see the OAuth 2.0 specification.

The **Client credentials** flow is used when the client requests an access token to access its own resources. Use this flow when your app is requesting the token on its own behalf, not on behalf of a user.

Allowed OAuth Scopes

Choose one or more of the following `OAuth` scopes to specify the access privileges that can be requested for access tokens.

- The `phone` scope grants access to the `phone_number` and `phone_number_verified` claims. This scope can only be requested with the `openid` scope.
- The `email` scope grants access to the `email` and `email_verified` claims. This scope can only be requested with the `openid` scope.
- The `openid` scope returns all user attributes in the ID token that are readable by the client. The ID token is not returned if the `openid` scope is not requested by the client.
- The `aws.cognito.signin.user.admin` scope grants access to Amazon Cognito User Pool API operations that require access tokens, such as UpdateUserAttributes and VerifyUserAttribute.
- The `profile` scope grants access to all user attributes that are readable by the client.

Allowed Custom Scopes

A custom scope is one that you define for your own resource server in the **Resource Servers** tab. The format is *`resource-server-identifier`*/*`scope`*.

For more information about OAuth scopes, see the list of standard OIDC scopes.

Specifying App Identity Provider Settings for Your User Pool (AWS Management Console)

You can use the AWS Management Console to specify app identity provider (IdP) settings for your user pool. You must specify at least one IdP per app.

To specify app identity provider settings

1. Sign in to the Amazon Cognito console.
2. In the navigation pane, choose **Manage your User Pools**, and choose the user pool you want to edit.
3. Choose the **App client settings** tab.

4. In **Enabled Identity Providers**, select the identity provider to be enabled for the app client that you previously configured in the **App clients** tab.
5. In the **Sign in and sign out URLs** section, enter the **Callback URLs** you want to use, separated by commas.

   Note
   You must register the URLs, either in the console or by using the CLI or API, before you can use them in your app.
6. Enter the **Sign out URLs** you want to use, separated by commas.

   Note
   You must register the URLs, either in the console or by using the CLI or API, before you can use them in your app.
7. Under **OAuth 2.0**, select from the following options:
   - For **Allowed OAuth Flows**, select `Authorized code grant` and `Implicit grant`.
   - For **Allowed OAuth Scopes**, select the scopes you want. Each scope is a set of one or more standard attributes. For more information, see About App Identity Settings (p. 83).

Specifying App Identity Provider Settings for Your User Pool (AWS CLI and AWS API)

Use the following commands to specify app identity provider settings for your user pool.

To add or change identity provider settings for your user pool's existing client app

- AWS CLI: `aws cognito-idp update-user-pool-client`

  Example: `update-user-pool-client --user-pool-id <user_pool_id> --client-id <client_id> --allowed-oauth-flows-user-pool-client true --allowed-oauth-scopes "code" "token" --allowed-oauth-scopes "openid" "cognito" --callback-urls https://example.com --supported-identity-providers ["saml_provider_1", "saml_provider_2"]`
- AWS API: UpdateUserPoolClient

To create a user pool client for your app and IdP

- AWS CLI: `aws cognito-idp create-user-pool-client`

  Example: `aws cognito-idp create-user-pool-client --user-pool-id <user_pool_id> --client-name myApp`
- AWS API: CreateUserPoolClient

To get information about your user pool's client app identity provider settings

- AWS CLI: `aws cognito-idp describe-user-pool-client`

  Example: `aws cognito-idp describe-user-pool-client --user-pool-id <user_pool_id> --client-id <client_id>`
- AWS API: DescribeUserPoolClient

To list information about all clients for your user pool

- AWS CLI: `aws cognito-idp list-user-pool-clients`

Example: `aws cognito-idp list-user-pool-clients --user-pool-id <user_pool_id> --max-results 3`

- AWS API: ListUserPoolClients

To delete a user pool client

- AWS CLI: `aws cognito-idp delete-user-pool-client`

  Example: `aws cognito-idp delete-user-pool-client --user-pool-id <user_pool_id> --client-id <client_id>`
- AWS API: DeleteUserPoolClient

Assigning a Domain to Your User Pool

You can host the pages for your user pool on a subdomain of the Amazon Cognito domain by specifying your own prefix domain name.

The domain for your app will be `https://<domain_prefix>.auth.<region>.amazoncognito.com`.

The full URL for your app will look like this example: `https://example.auth.us-east-1.amazoncognito.com/login?redirect_uri=https://www.google.com&response_type=code&client_id=<client_id_value>`

> **Important**
> Before you can access the URL for your app, you must specify app client settings. For more information, see Specifying App Client Settings for Your User Pool (p. 26).

Assigning a Domain to Your User Pool (AWS Management Console)

You can use the AWS Management Console to assign a domain to your user pool.

To assign a domain

1. Sign in to the Amazon Cognito console.
2. In the navigation pane, choose **Manage your User Pools**, and choose the user pool you want to edit.
3. Choose the **Domain name** tab.
4. On the **Domain name** tab, enter the domain name you want to use in the **Prefix domain name** box.
5. Choose **Check availability** as needed.
6. Choose **Save changes**.

Assigning a Domain to Your User Pool (AWS CLI and AWS API)

Use the following commands to create a custom domain name and assign it to your user pool.

To assign a domain

- AWS CLI: `aws cognito-idp create-user-pool-domain`

  Example: `aws cognito-idp create-user-pool-domain --user-pool-id <user_pool_id> --domain <domain_name>`

- AWS API: CreateUserPoolDomain

To get information about a domain

- AWS CLI: `aws cognito-idp describe-user-pool-domain`

  Example: `aws cognito-idp describe-user-pool-domain --domain <domain_name>`
- AWS API: DescribeUserPoolDomain

To delete a domain

- AWS CLI: `aws cognito-idp delete-user-pool-domain`

  Example: `aws cognito-idp delete-user-pool-domain --domain <domain_name>`
- AWS API: DeleteUserPoolDomain

Specifying App UI Customization Settings for Your User Pool

You can use the AWS Management Console, or the AWS CLI or API, to specify customization settings for the built-in app UI experience. You can upload a custom logo image to be displayed in the app. You can also choose many CSS customizations.

You can specify app UI customization settings for a single client (with a specific `clientId`) or for all clients (by setting the `clientId` to `ALL`). If you specify `ALL`, the default configuration will be used for every client that has no UI customization set previously. If you specify UI customization settings for a particular client, it will no longer fall back to the `ALL` configuration.

> **Note**
> To use this feature, your user pool must have a domain associated with it.

Specifying a Custom Logo for the App

The maximum allowable size for a logo image file is 100 KB.

Specifying CSS Customizations for the App

You can customize the CSS for the hosted app pages, with the following restrictions:

- The CSS class names can only be from the following list:
  - `banner-customizable`
  - `errorMessage-customizable`
  - `idpButton-customizable`
  - `idpButton-customizable:hover`
  - `inputField-customizable`
  - `inputField-customizable:focus`
  - `label-customizable`
  - `legalText-customizable`
  - `logo-customizable`
  - `submitButton-customizable`

  - `submitButton-customizable:hover`
  - `textDescription-customizable`
- Property values cannot contain HTML, `@import`, `@supports`, `@page`, or `@media` statements or Javascript.

You can customize the following CSS properties.

Labels

- **font-weight** is a multiple of 100 from 100 to 900.

Input fields

- **width** is the width as a percentage of the containing block.
- **height** is the height of the input field in pixels (px).
- **color** is the text color. It can be any standard CSS color value.
- **background-color** is the background color of the input field. It can be any standard color value.
- **border** is a standard CSS border value that specifies the width, transparency, and color of the border of your app window. Width can be any value from 1px to 100px. Transparency can be solid or none. Color can be any standard color value.

Text descriptions

- **padding-top** is the amount of padding above the text description.
- **padding-bottom** is the amount of padding below the text description.
- **display** can be `block` or `inline`.
- **font-size** is the font size for text descriptions.

Submit button

- **font-size** is the font size of the button text.
- **font-weight** is the font weight of the button text: `bold`, `italic`, or `normal`.
- **margin** is a string of 4 values indicating the top, right, bottom, and left margin sizes for the button.
- **font-size** is the font size for text descriptions.
- **width** is the width of the button text in percent of the containing block.
- **height** is the height of the button in pixels (px).
- **color** is the button text color. It can be any standard CSS color value.
- **background-color** is the background color of the button. It can be any standard color value.

Banner

- **padding** is a string of 4 values indicating the top, right, bottom, and left padding sizes for the banner.
- **background-color** is the banner's background color. It can be any standard CSS color value.

Submit button hover

- **color** is the foreground color of the button when you hover over it. It can be any standard CSS color value.
- **background-color** is the background color of the button when you hover over it. It can be any standard CSS color value.

Identity provider button hover

- **color** is the foreground color of the button when you hover over it. It can be any standard CSS color value.
- **background-color** is the background color of the button when you hover over it. It can be any standard CSS color value.

Password check not valid

- **color** is the text color of the `"Password check not valid"` message. It can be any standard CSS color value.

Background

- **background-color** is the background color of the app window. It can be any standard CSS color value.

Error messages

- **margin** is a string of 4 values indicating the top, right, bottom, and left margin sizes.
- **padding** is the padding size.
- **font-size** is the font size.
- **width** is the width of the error message as a percentage of the containing block.
- **background** is the background color of the error message. It can be any standard CSS color value.
- **border** is a string of 3 values specifying the width, transparency, and color of the border.
- **color** is the error message text color. It can be any standard CSS color value.
- **box-sizing** is used to indicate to the browser what the sizing properties (width and height) should include.

Identity provider buttons

- **height** is the height of the button in pixels (px).
- **width** is the width of the button text as a percentage of the containing block.
- **text-align** is the text alignment setting. It can be `left`, `right`, or `center`.
- **margin-bottom** is the bottom margin setting.
- **color** is the button text color. It can be any standard CSS color value.
- **background-color** is the background color of the button. It can be any standard color value.
- **border-color** is the color of the button border. It can be any standard color value.

Identity provider descriptions

- **padding-top** is the amount of padding above the description.
- **padding-bottom** is the amount of padding below the description.
- **display** can be `block` or `inline`.
- **font-size** is the font size for descriptions.

Legal text

- **color** is the text color. It can be any standard CSS color value.
- **font-size** is the font size.

Logo

- **max-width** is the maximum width as a percentage of the containing block.
- **max-height** is the maximum height as a percentage of the containing block.

Input field focus

- **border-color** is the color of the input field. It can be any standard CSS color value.
- **outline** is the border width of the input field, in pixels.

Social button

- **height** is the height of the button in pixels (px).
- **text-align** is the text alignment setting. It can be `left`, `right`, or `center`.
- **width** is the width of the button text as a percentage of the containing block.
- **margin-bottom** is the bottom margin setting.

Password check valid

- **color** is the text color of the `"Password check valid"` message. It can be any standard CSS color value.

Specifying App UI Customization Settings for Your User Pool (AWS Management Console)

You can use the AWS Management Console to specify UI customization settings for your app.

> **Note**
> You can view the hosted UI with your customizations by constructing the following URL, with the specifics for your user pool, and typing it into a browser: `https://<your_domain>/login?response_type=code&client_id=<your_app_client_id>&redirect_uri=<your_callback_url>`
> You may have to wait up to one minute to refresh your browser before changes made in the console appear.
> Your domain is shown on the **Domain name** tab. Your app client ID and callback URL are shown on the **App client settings** tab.

To specify app UI customization settings

1. Sign in to the Amazon Cognito console.
2. In the navigation pane, choose **Manage your User Pools**, and choose the user pool you want to edit.
3. Choose the **UI customization** tab.
4. Under **App client to customize**, choose the app you want to customize from the dropdown menu of app clients that you previously created in the **App clients** tab.
5. To upload your own logo image file, choose **Choose a file** or drag a file into the **Logo (optional)** box.
6. Under **CSS customizations (optional)**, you can customize the appearance of the app by changing various properties from their default values.

Specifying App UI Customization Settings for Your User Pool (AWS CLI and AWS API)

Use the following commands to specify app UI customization settings for your user pool.

To get the UI customization settings for a user pool's built-in app UI

- AWS CLI: `aws cognito-idp get-ui-customization`
- AWS API: GetUICustomization

To set the UI customization settings for a user pool's built-in app UI

- AWS CLI: `aws cognito-idp set-ui-customization --user-pool-id <your-user-pool-id> --client-id <your-app-client-id> --image-file <path-to-logo-image-file> --css ".label-customizable{ color: <color>;}"`
- AWS API: SetUICustomization

Defining Resource Servers for Your User Pool

A *resource server* is a server for access-protected resources. It handles authenticated requests from an app that has an access token. Typically the resource server provides a CRUDL API for making these access requests. This API can be hosted in Amazon API Gateway or outside of AWS. The app passes the access token in the API call to the resource server. The app should treat the access token as opaque when it passes the token in the access request. The resource server inspects the access token to determine if access should be granted.

> **Note**
> Your resource server must verify the access token signature and expiration date before processing any claims inside the token. For more information about verifying and using user pool tokens, see this blog post. Amazon API Gateway is a good option for inspecting access tokens and protecting your resources. For more about API Gateway custom authorizers, see Use API Gateway Custom Authorizers.

A *scope* is a level of access that an app can request to a resource. For example, if you have a resource server for photos, it might define two scopes: one for read access to the photos and one for write/delete access. When the app makes an API call to request access and passes an access token, the token will have one or more scopes embedded in inside it.

Overview

Amazon Cognito allows app developers to create their own OAuth2.0 resource servers and define custom scopes in them. Custom scopes can then be associated with a client, and the client can request them in OAuth2.0 authorization code grant flow, implicit flow, and client credentials flow. Custom scopes are added in the `scope` claim in the access token. A client can use the access token against its resource server, which makes the authorization decision based on scopes present in the token. For more information about access token scope, see Using Tokens with User Pools (p. 128).

> **Note**
> Your resource server must verify the access token signature and expiration date before processing any claims inside the token.

> **Note**
> An app client can only use the client credentials flow if the app client has a client secret.

Managing the Resource Server and Custom Scopes

When creating a resource server, you must provide a resource server name and a resource server identifier. For each scope you create in the resource server, you must provide the scope name and description.

Example:

- `Name`: a friendly name for the resource server, such as `Weather API` or `Photo API`.
- `Identifier`: Unique identifier for the resource server. This could be an HTTPS endpoint where your resource server is located. For example, `https://my-weather-api.example.com`
- `Scope Name`: The scope name. For example, `weather.read`
- `Scope Description`: A brief description the scope. For example, `Retrieve weather information`.

When a client app requests a custom scope in any of the OAuth2.0 flows, it must request the full identifier for the scope, which is *`resourceServerIdentifier`*`/`*`scopeName`*. For example, if the resource server identifier is `https://myphotosapi.example.com` and the scope name is `photos.read`, the client app must request `https://myphotosapi.example.com/photos.read` at runtime.

Deleting a scope from a resource server does not delete its association with all clients; deleting the scope makes it inactive. So if a client app requests the deleted scope at runtime, the scope is ignored and is not included in the access token. If the scope is re-added later, then it is again included in the access token.

If a scope is removed from a client, the association between client and scope is deleted. If a client requests a disallowed scope at runtime, this results in an error, and an access token is not issued.

You can use the AWS Management Console, API, and CLI to define resource servers and scopes for your user pool.

Defining a Resource Server for Your User Pool (AWS Management Console)

You can use the AWS Management Console to define a resource server for your user pool.

To define a resource server

1. Sign in to the Amazon Cognito console.
2. In the navigation pane, choose **Manage your User Pools**, and choose the user pool you want to edit.
3. Choose the **Resource servers** tab.
4. Choose **Add a resource server**.
5. Enter the name of your resource server, for example, `Photo Server`.
6. Enter the identifier of your resource server, for example, `com.example.photos`.
7. Enter the names of the custom scopes for your resources, such as `read` and `write`.
8. For each of the scope names, enter a description, such as `view your photos` and `update your photos`.
9. Choose **Save changes**.

Each of the custom scopes that you define appears on the **App client settings** tab, under **OAuth2.0 Allowed Custom Scopes**; for example `com.example.photos/read`.

Defining a Resource Server for Your User Pool (AWS CLI and AWS API)

Use the following commands to specify resource server settings for your user pool.

To create a resource server

- AWS CLI: `aws cognito-idp create-resource-server`
- AWS API: CreateResourceServer

To get information about your resource server settings

- AWS CLI: `aws cognito-idp describe-resource-server`
- AWS API: DescribeResourceServer

To list information about all resource servers for your user pool

- AWS CLI: `aws cognito-idp list-resource-servers`
- AWS API: ListResourceServers

To delete a resource server

- AWS CLI: `aws cognito-idp delete-resource-server`
- AWS API: DeleteResourceServer

To update the settings for a resource server

- AWS CLI: `aws cognito-idp update-resource-server`
- AWS API: UpdateResourceServer

Using Federation for Amazon Cognito User Pools

Federation enables users to sign in to an Amazon Cognito user pool using an external identity provider. Currently, users can sign up and sign in through:

- Corporate identity providers (IdPs), such as Microsoft Active Directory Federation Services, via SAML
- Social identity providers, such as Facebook, Google, and Login with Amazon

Federation for Amazon Cognito user pools simplifies user management. It provides a unified user directory so that user profiles for all users from federated IdPs can be managed in your user pool. It simplifies authentication and authorization by providing a common set of tokens for all authenticated users from all IdPs, so backend systems can standardize on one set of tokens. It provides built-in integrations with IdPs, so developers can direct users to Amazon Cognito for sign-up or sign-in, and Amazon Cognito can manage the rest of the process. Amazon Cognito also provides a single sign-on (SSO) experience, allowing users to sign in once for access to multiple apps when those apps use the same Amazon Cognito user pool.

Topics

- Adding Social Identity Providers (p. 93)
- Creating SAML Identity Providers for Your User Pool (p. 95)
- Integrating Third-Party SAML Identity Providers with Amazon Cognito User Pools (p. 100)
- Specifying Identity Provider Attribute Mappings for Your User Pool (p. 101)

Adding Social Identity Providers

You can add a social identity provider (such as Facebook, Google, or Login with Amazon) as an identity provider in the AWS Management Console, or with Amazon Cognito CLI or Amazon Cognito API calls.

When you add a social identity provider, you specify authorization scopes for requesting permission to access user information in the social networking site. The user must consent to sharing the requested scopes with your app, after authenticating to the identity provider. Amazon Cognito invokes only the identity provider's profile API, so only scopes that are valid for the profile API should be included in your authorization scopes.

Adding Facebook as an Identity Provider

1. Create an app in the Facebook app site at https://developers.facebook.com/apps/ and note the app ID and app secret.
2. Sign in to the Amazon Cognito console.
3. In the navigation pane, choose **Manage your User Pools**, and choose the user pool you want to edit.
4. Choose the **Identity provider** tab.
5. Choose **Facebook**.
6. For the Facebook app ID, enter the app ID that you received when you created your Facebook app.
7. For **App secret**, enter the app secret that you received when you created your Facebook app.

8. For **Authorize scopes**, enter the names of the Facebook scopes that you want to map to user pool attributes, separated by commas, for example, `email,public_profile`. For more information, see https://developers.facebook.com/docs/facebook-login/permissions/#reference-public_profile.
9. Choose **Update Facebook**.
10. In the Facebook apps dashboard:
    1. Choose your app.
    2. Choose **Facebook Login** in the left navigation bar.
    3. In the **Settings** section, enable **Client OAuth Login** and **Web OAuth Login**.
    4. In **Valid OAuth Redirect URIs**, add your domain redirect URL, for example, `https://your-user-pool-domain/oauth2/idpresponse`.

Adding Google as an Identity Provider

1. Create an app in the Google API dashboard at https://console.developers.google.com/apis/dashboard.
2. Create Credentials for the app in the **Credentials** section.
3. Sign in to the Amazon Cognito console.
4. In the navigation pane, choose **Manage your User Pools**, and choose the user pool you want to edit.
5. Choose the **Identity provider** tab.
6. Choose **Google**.
7. For the Google app ID, enter the app ID that you received when you created your Google app.
8. For **App secret**, enter the app secret that you received when you created your Google app.
9. For **Authorize scopes**, enter the names of the Google scopes that you want to map to user pool attributes, separated by spaces, for example, `openid email profile`. For more information, see https://developers.facebook.com/docs/facebook-login/permissions/#reference-public_profile.
10. Choose **Update Google**.

Enable the Google People API for your app by clicking the **Enable API** menu at the top of the **Google APIs** dashboard.

Developers must also add their Amazon Cognito User Pool domain URL `https://your-user-pool-domain/oauth2/idpresponse` in the Google app's Authorized redirect URIs (in the Credentials section). This ensures that Google will accept the redirect URL supplied by Amazon Cognito when it authenticates users. For more information, see https://developers.google.com/identity/protocols/OAuth2WebServer.

Adding Login with Amazon as an Identity Provider

1. Create an app in the Login With Amazon App Console at http://login.amazon.com/manageApps. Note the client ID and client secret (shown in the **Web Settings** section).
2. Sign in to the Amazon Cognito console.
3. In the navigation pane, choose **Manage your User Pools**, and choose the user pool you want to edit.
4. Choose the **Identity provider** tab.
5. Choose **Amazon**.
6. For the Amazon app ID, enter the app ID that you received when you created your Amazon app.
7. For **App secret**, enter the app secret that you received when you created your Amazon app.
8. For **Authorize scopes**, enter the names of the Login with Amazon scopes that you want to map to user pool attributes, separated by spaces, for example, `profile postal_code`.

9. Choose **Update Amazon**.

Add your Amazon Cognito User Pool domain URL `https://your-user-pool-domain/oauth2/idpresponse` to the Login with Amazon app's Allowed Return URLs. This ensures that Login with Amazon will accept the redirect URL supplied by Amazon Cognito when it authenticates users. For more information, see http://login.amazon.com/website.

Adding Social Identity Providers (AWS Management Console)

You can use the AWS Management Console to specify attribute mappings for your user pool's identity providers.

To add a social identity provider

1. Sign in to the Amazon Cognito console.
2. In the navigation pane, choose **Manage your User Pools**, and choose the user pool you want to edit.
3. Choose the **Identity providers** tab.
4. Choose a social identity provider such as **Facebook**, **Google**, or **Amazon**.
5. For the Facebook (or Google or Amazon) app ID, enter the app ID that you received when you created your Facebook, Google, or Login with Amazon client app.
6. For **App secret**, enter the app secret that you received when you created your client app.
7. For **Authorize scopes**, enter the names of the social identity provider scopes, such as `email`, that you want to map to user pool attributes.
8. Choose **Update Facebook** (or **Update Google** or **Update Amazon**).

Creating SAML Identity Providers for Your User Pool

A SAML 2.0 identity provider is an entity in Amazon Cognito that describes an identity provider (IdP) that supports the SAML 2.0 standard. You can create and manage a SAML IdP in the AWS Management Console, or with Amazon Cognito CLI or Amazon Cognito API calls.

On the **Identity provider** tab, you can create SAML 2.0 identity providers (IdP) directly for use with your user pool.

> **Note**
> SAML federation support in Amazon Cognito User Pools is independent of Amazon Cognito Federated Identities.

Amazon Cognito User Pools acts as a service provider (SP) on behalf of your application. Amazon Cognito supports SP-initiated single sign-on (SSO) as described in section 5.1.2 of the SAML V2.0 Technical Overview. The redirect endpoint for the `POST` binding is `https://<domain_prefix>.auth.<region>.amazoncognito.com/saml2/idpresponse`.

> **Note**
> Any SAML identity providers that you created in a user pool during the public beta before August 10, 2017 have redirect URLs of `https://<yourDomainPrefix>.auth.<region>.amazoncognito.com/login/redirect`. If you have one of these SAML identity providers from the public beta in your user pool, you must either:
>
> - Replace it with a new one that uses the new redirect URL.
> - Update the configuration in your SAML identity provider to accept both the old and new redirect URLs.

All SAML identity providers in Amazon Cognito will switch to the new URLs, and the old ones will stop working on October 31, 2017.

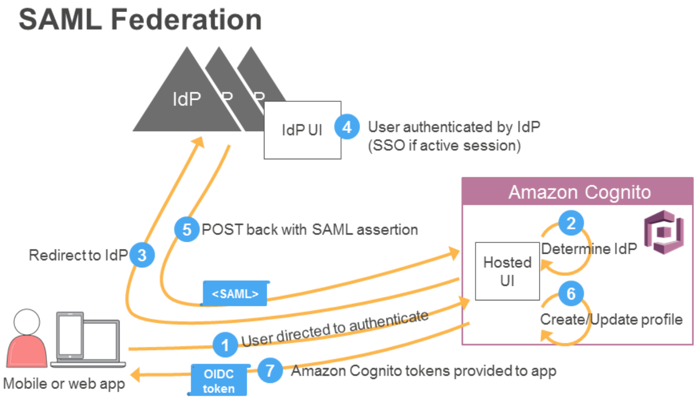
SAML IdP Authentication Flow

Developers can easily integrate the SAML-based IdPs of their customers into their apps.

1. The app starts the sign-up and sign-in process by directing the user to the UI hosted by AWS. A mobile app uses a web view to show the pages hosted by AWS.
2. Typically Amazon Cognito User Pools determines the identity provider for the user from the user's email address.

   Alternatively, if the app gathered information before directing the user to Amazon Cognito User Pools, it provides that information to Amazon Cognito through a query parameter.
3. The user is redirected to the identity provider.
4. The IdP authenticates the user if necessary. If the IdP recognizes that the user has an active session, the IdP skips the authentication to provide a single sign-in (SSO) experience.
5. The IdP POSTs the SAML assertion to Amazon Cognito User Pools.
6. The user's profile can be created or updated in the AWS Management Console, or the Amazon Cognito CLI or API.
7. After verifying the SAML assertion and collecting the user attributes (claims) from the assertion, Amazon Cognito User Pools returns OIDC tokens to the app for the now signed-in user.

The following diagram shows the authentication flow for this process:


SAML Federation
IdP
P
P
IdP UI
4 User authenticated by IdP
(SSO if active session)
Amazon Cognito
2
Determine IdP
Hosted UI
6
Create/Update profile
5 POST back with SAML assertion
Redirect to IdP 3
<SAML>
1 User directed to authenticate
OIDC token
7 Amazon Cognito tokens provided to app
Mobile or web app

When a user authenticates, the user pool provides ID, access, and refresh tokens. The ID token is a standard OIDC token for identity management, and the access token is a standard OAuth 2.0 token. The ID and access tokens expire after one hour, but your app can use the refresh token to get new tokens without having the user re-authenticate. As a developer, you can choose the expiration time of refresh tokens, and therefore how frequently users need to reauthenticate. If the user has authenticated through an external IdP (i.e., they are a federated user), your app still uses the Amazon Cognito tokens with the refresh token to determine how long until the user reauthenticates, regardless of when the external IdP's token expires. The user pool automatically uses the refresh token to get new ID and access tokens when they expire. If the refresh token has also expired, the server automatically initiates authentication through the pages in your app that are hosted by AWS.

Creating and Managing a SAML Identity Provider (AWS Management Console)

You can use the AWS Management Console to create and delete SAML identity providers.

Before you create a SAML identity provider, you need the SAML metadata document that you get from the third-party identity provider (IdP). For instructions on how to get or generate the required SAML metadata document, see Integrating Third-Party SAML Identity Providers with Amazon Cognito User Pools (p. 100).

You need to choose names for your SAML providers. The string format is [\w\s+=.@-]+ and can be up to 40 characters long.

You can also optionally choose identifiers for your SAML providers. An identifier uniquely resolves to an identity provider associated with your user pool. Typically each identifier corresponds to a domain that belongs to the company that the IdP represents. For a multitenant app that can be used by different companies, identifiers can be used to redirect users to the correct IdP. Since there can be multiple domains owned by the same company, you can provide multiple identifiers.

You can associate up to 50 identifiers with each SAML provider. Identifiers must be unique across the identity provider.

For example, suppose that you built an app that can be used by employees of two different companies, A and B. Company A owns `domainA.com` and `domainA.co.uk`; Company B owns `domainB.com`. Suppose further that you set up two IdPs, one for each company:

- For IdP A, you can define identifiers `DomainA.com` and `DomainA.co.uk`.
- For IdP B, you can define identifier `DomainB.com`.

In your application, you can prompt users to enter their email addresses. By deriving the domain from the email address, you can redirect the user to the correct IdP by providing the domain in the `IdPIdentifier` in the call to the `/authorize` endpoint. For example, if a user enters `bob@domain1.co.uk`, the user is redirected to IdP A.

The sign-in page hosted by Amazon Cognito parses the email address automatically to derive the information. It parses the email domain from email and uses it as `IdPIdentifier` when calls the `/authorize` endpoint.

- If you have multiple SAML IdPs and you specify an `IdPIdentifier` value for any one of them, you will see a box to enter an email address on the hosted page.
- If you have multiple IdPs, and you do not specify an `IdPIdentifier` value for any of them, the hosted page will show a list of IdPs.

If you're building your own UI, you should parse the domain name so that it matches the `IdPIdentifiers` that are provided during the IdP setup. For more information about IdP setup, see Specifying Identity Providers for Your User Pool (p. 28).

To create a SAML provider

1. Sign in to the Amazon Cognito console.
2. In the navigation pane, choose **Manage your User Pools**, and choose the user pool you want to edit.
3. Choose the **Identity provider** tab.
4. Choose **Corporate ID via SAML**.
5. To attach your own custom metadata document, choose **Select file**. Or enter a metadata document endpoint URL. The metadata document must be a valid XML file.

   Note
   We recommend that you provide the endpoint URL if it is a public endpoint, rather than uploading a file, because this allows Amazon Cognito to refresh the metadata automatically. Typically metadata refresh happens every 6 hours or before the metadata expires, whichever is earlier.

6. Enter your SAML **Provider name** and any **Identifiers** that you want. The provider name is required; the identifiers are optional.
7. Choose **Create provider**.

> **Note**
> If you see `InvalidParameterException` while creating a SAML identity provider with an HTTPS metadata endpoint URL, for example, "Error retrieving metadata from *`<metadata endpoint>`*," make sure that the metadata endpoint has SSL correctly set up and that there is a valid SSL certificate associated with it.

To set up the SAML IdP to add Amazon Cognito User Pools as a relying party

- The Amazon Cognito User Pools service provider URN is: `urn:amazon:cognito:sp:<user_pool_id>`. Amazon Cognito User Pools issues the `AuthnRequest` to SAML IdP to issue a SAML assertion with audience restriction to this URN. Your IdP uses the following POST binding endpoint for the IdP-to-SP response message: `https://<domain_prefix>.auth.<region>.amazoncognito.com/saml2/idpresponse`.
- You need to make sure that your SAML IdP populates `NameID` and any required attributes for your user pool in the SAML assertion. `NameID` is used for uniquely identifying your SAML federated user in the user pool.

To delete a SAML provider

1. Sign in to the Amazon Cognito console.
2. In the navigation pane, choose **Manage your User Pools**, and choose the user pool you want to edit.
3. Choose the **Identity provider** tab.
4. Choose **Corporate ID via SAML** to display the SAML identity providers.
5. Select the check box next to the provider to be deleted.
6. Choose **Delete provider**.

Creating and Managing a SAML Identity Provider (AWS CLI and AWS API)

Use the following commands to create and manage a SAML provider.

To create an identity provider and upload a metadata document

- AWS CLI: `aws cognito-idp create-identity-provider`

  Example with metadata file: `aws cognito-idp create-identity-provider --user-pool-id <user_pool_id> --provider-name=SAML_provider_1 --provider-type SAML --provider-details file:///details.json --attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`

  Where `details.json` contains:

```
{
    "MetadataFile": "<SAML metadata XML>"
}
```

> **Note**
> If the *`<SAML metadata XML>`* contains any quotations (`"`), they must be escaped (`\"`).

Example with metadata URL: `aws cognito-idp create-identity-provider --user-pool-id <user_pool_id> --provider-name=SAML_provider_1 --provider-type SAML --provider-details MetadataURL=<metadata_url> --attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`

- AWS API: CreateIdentityProvider

To upload a new metadata document for an identity provider

- AWS CLI: `aws cognito-idp update-identity-provider`

  Example with metadata file: `aws cognito-idp update-identity-provider --user-pool-id <user_pool_id> --provider-name=SAML_provider_1 --provider-details file:///details.json --attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`

  Where `details.json` contains:

```
{
    "MetadataFile": "<SAML metadata XML>"
}
```

> **Note**
> If the `<SAML metadata XML>` contains any quotations (`"`), they must be escaped (`\"`).

  Example with metadata URL: `aws cognito-idp update-identity-provider --user-pool-id <user_pool_id> --provider-name=SAML_provider_1 --provider-details MetadataURL=<metadata_url> --attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`

- AWS API: UpdateIdentityProvider

To get information about a specific identity provider

- AWS CLI: `aws cognito-idp describe-identity-provider`

  `aws cognito-idp describe-identity-provider --user-pool-id <user_pool_id> --provider-name=SAML_provider_1`

- AWS API: DescribeIdentityProvider

To list information about all identity providers

- AWS CLI: `aws cognito-idp list-identity-providers`

  Example: `aws cognito-idp list-identity-providers --user-pool-id <user_pool_id> --max-results 3`

- AWS API: ListIdentityProviders

To delete an IdP

- AWS CLI: `aws cognito-idp delete-identity-provider`

  `aws cognito-idp delete-identity-provider --user-pool-id <user_pool_id> --provider-name=SAML_provider_1`

- AWS API: DeleteIdentityProvider

Integrating Third-Party SAML Identity Providers with Amazon Cognito User Pools

To configure third-party SAML 2.0 identity provider solutions to work with federation for Amazon Cognito User Pools, you must enter a redirect or sign-in URL, which is `https://<yourDomainPrefix>.auth.<region>.amazoncognito.com/saml2/idpresponse`. You can find your domain prefix and the region value for your user pool on the **Domain name** tab of the Amazon Cognito console.

> **Note**
> Any SAML identity providers that you created in a user pool during the public beta before August 10, 2017 have redirect URLs of `https://<yourDomainPrefix>.auth.<region>.amazoncognito.com/login/redirect`. If you have one of these SAML identity providers from the public beta in your user pool, you must either:
>
> - Replace it with a new one that uses the new redirect URL.
> - Update the configuration in your SAML identity provider to accept both the old and new redirect URLs.
>
> All SAML identity providers in Amazon Cognito will switch to the new URLs, and the old ones will stop working on October 31, 2017.

For some SAML Identity providers you must provide the `urn` / Audience URI / SP Entity ID, in the form `urn:amazon:cognito:sp:<yourUserPoolID>`. You can find your user pool ID on the **App client settings** tab in the Amazon Cognito console.

You must also configure your SAML identity provider to provide attributes values for any attributes required in your user pool. Typically, `email` is a required attribute for user pools, and in that case the SAML identity provider will need to provide an `email` value (claim) in the SAML assertion.

The following links help you configure third-party SAML 2.0 identity provider solutions to work with federation for Amazon Cognito User Pools.

> **Note**
> Identity provider support is built in to Amazon Cognito, so you only need to go to the following provider sites to get the SAML metadata document. You may see further instructions on the provider website about integrating with AWS, but you won't need those.

Solution	More information
Microsoft Active Directory Federation Services (AD FS)	You can download the SAML metadata document for your ADFS federation server from the following address: `https://<yourservername>/FederationMetadata/2007-06/FederationMetadata.xml`.
Okta	Once you have configured your Amazon Cognito User Pool as an application in Okta, you can find the metadata document in the **Admin** section of the Okta dashboard. Choose the application, select the **Sign On** section, and look under the **Settings for SAML**. The URL should look like `https://<app-domain>.oktapreview.com/app/<application-ID>/sso/saml/metadata`.
Auth0	The metadata download document is obtained from the Auth0 dashboard. Choose **Clients**,

Solution	More information
	and then choose **Settings**. Scroll down, choose **Show Advanced Settings**, and then look for your **SAML Metadata URL**. It should look like `https://<your-domain-prefix>.auth0.com/samlp/metadata/<your-Auth0-client-ID>`.
Ping Identity	For PingFederate, you can find instructions for downloading a metadata XML file in Provide general SAML metadata by file.

Specifying Identity Provider Attribute Mappings for Your User Pool

You can use the AWS Management Console, or the AWS CLI or API, to specify attribute mappings for your user pool's identity providers.

Specifying Identity Provider Attribute Mappings for Your User Pool (AWS Management Console)

You can use the AWS Management Console to specify attribute mappings for your user pool's identity providers.

> **Note**
> Currently, only the Facebook `id`, Google `sub`, and Login with Amazon `user_id` attributes can be mapped to the Amazon Cognito user pools `username` attribute.

> **Note**
> Make sure the attribute in the Amazon Cognito User Pool is large enough to fit the values of the mapped identity provider attributes, or an error occurs when users sign in. Custom attributes should be set to the maximum 2048 character size if mapped to identity provider tokens.
> Make sure you create mappings for any attributes that are required for your user pool.

To specify a social identity provider attribute mapping

1. Sign in to the Amazon Cognito console.
2. In the navigation pane, choose **Manage your User Pools**, and choose the user pool you want to edit.
3. Choose the **Attribute mapping** tab.
4. Choose the **Facebook**, **Google**, or **Amazon** tab.
5. For each attribute you need to map, perform the following steps:
    1. Select the **Capture** check box.
    2. For **User pool attribute**, choose the user pool attribute you want to map to the social identity provider attribute to from the drop-down list.
    3. If you need more attributes, choose **Add Facebook attribute** (or **Add Google attribute** or **Add Amazon attribute**) and perform the following steps:
        1. In the **Facebook attribute** (or **Google attribute** or **Amazon attribute**) field, enter the name of the attribute to be mapped.
        2. In the **User pool attribute** field, choose the user pool attribute to map the social identity provider attribute to from the drop-down list.
    4. Choose **Save changes**.

To specify a SAML provider attribute mapping

1. Sign in to the Amazon Cognito console.
2. In the navigation pane, choose **Manage your User Pools**, and choose the user pool you want to edit.
3. Choose the **Attribute mapping** tab.
4. Choose the **SAML** tab.
5. Select the **Capture** box for all attributes for which you want to capture values. If you clear the **Capture** box for an attribute and save your changes, that attribute's mapping is removed.
6. Choose the identity provider from the drop-down list.
7. For each attribute you need to map, perform the following steps:
    1. Choose **Add SAML attribute**.
    2. In the **SAML attribute** field, enter the name of the SAML attribute to be mapped.
    3. In the **User pool attribute** field, choose the user pool attribute to map the SAML attribute to from the drop-down list.
8. Choose **Save changes**.

Specifying Identity Provider Attribute Mappings for Your User Pool (AWS CLI and AWS API)

Use the following commands to specify identity provider attribute mappings for your user pool.

To specify attribute mappings at provider creation time

- AWS CLI: `aws cognito-idp create-identity-provider`

  Example with metadata file: `aws cognito-idp create-identity-provider --user-pool-id <user_pool_id> --provider-name=SAML_provider_1 --provider-type SAML --provider-details file:///details.json --attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`

  Where `details.json` contains:

```
{
    "MetadataFile": "<SAML metadata XML>"
}
```

  Note
  If the `<SAML metadata XML>` contains any quotations (`"`), they must be escaped (`\"`).

  Example with metadata URL: `aws cognito-idp create-identity-provider --user-pool-id <user_pool_id> --provider-name=SAML_provider_1 --provider-type SAML --provider-details MetadataURL=<metadata_url> --attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`
- AWS API: CreateIdentityProvider

To specify attribute mappings for an existing identity provider

- AWS CLI: `aws cognito-idp update-identity-provider`

  Example: `aws cognito-idp update-identity-provider --user-pool-id <user_pool_id> --provider-name <provider_name> --attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`

- AWS API: UpdateIdentityProvider

To get information about attribute mapping for a specific identity provider

- AWS CLI: `aws cognito-idp describe-identity-provider`

  Example: `aws cognito-idp describe-identity-provider --user-pool-id <user_pool_id> --provider-name <provider_name>`
- AWS API: DescribeIdentityProvider

Creating User Accounts as Administrator in the AWS Management Console and with the Amazon Cognito User Pools API

After you create your user pool, you can create users using the AWS Management Console, as well as the AWS Command Line Interface or the Amazon Cognito API. You can create a profile for a new user in a user pool and send a welcome message with sign-up instructions to the user via SMS or email.

Developers and administrators can perform the following tasks:

- Create a new user profile by using the AWS Management Console or by calling the `AdminCreateUser` API.
- Specify the temporary password or allow Amazon Cognito to automatically generate one.
- Specify whether provided email addresses and phone numbers are marked as verified for new users.
- Specify custom SMS and email invitation messages for new users via the AWS Management Console or a Custom Message Lambda trigger. For more information, see Customizing User Pool Workflows by Using AWS Lambda Triggers (p. 67).
- Specify whether invitation messages are sent via SMS, email, or both.
- Resend the welcome message to an existing user by calling the `AdminCreateUser` API, specifying `RESEND` for the `MessageAction` parameter.

  > **Note**
  > This action cannot currently be performed using the AWS Management Console.
- Suppress the sending of the invitation message when the user is created.
- Specify an expiration time limit for the user account (up to 90 days).
- Allow users to sign themselves up or require that new users only be added by the administrator.

For code examples, see the following topics:

- Example: Handling Users Created Using the AdminCreateUser API in the Mobile SDK for Android (p. 43)
- Examples: Using User Pools with the iOS SDK (p. 48)
- Example: Authenticate and Set a New Password for a User Created Using the AdminCreateUser API (p. 66)

Authentication Flow for Users Created by Administrators or Developers

The authentication flow for these users includes the extra step to submit the new password and provide any missing values for required attributes. The steps are outlined next; steps 5, 6, and 7 are specific to these users.

1. The user starts to sign in for the first time by submitting the username and password provided to him or her.
2. The SDK calls `InitiateAuth(Username, USER_SRP_AUTH)`.
3. Amazon Cognito returns the `PASSWORD_VERIFIER` challenge with Salt & Secret block.
4. The SDK performs the SRP calculations and calls `RespondToAuthChallenge(Username, <SRP variables>, PASSWORD_VERIFIER)`.
5. Amazon Cognito returns the `NEW_PASSWORD_REQUIRED` challenge along with the current and required attributes.
6. The user is prompted and enters a new password and any missing values for required attributes.
7. The SDK calls `RespondToAuthChallenge(Username, <New password>, <User attributes>)`.
8. If the user requires a second factor for MFA, Amazon Cognito returns the SMS_MFA challenge and the code is submitted.
9. After the user has successfully changed his or her password and optionally provided attributed values or completed MFA, the user is signed in and tokens are issued.

When the user has satisfied all challenges, the Amazon Cognito service marks the user as confirmed and issues ID, access, and refresh tokens for the user. For more information, see Using Tokens with User Pools (p. 128).

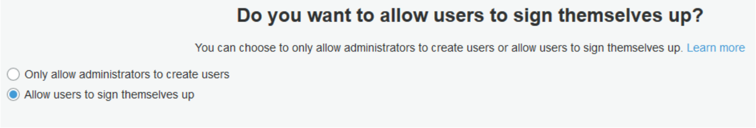
Creating a New User in the AWS Management Console

The Amazon Cognito console for managing user pools has been updated to support this feature, as shown next.

Policies Tab

The **Policies** tab has these related settings:

- Specify whether to allow users to sign themselves up. This option is set by default.



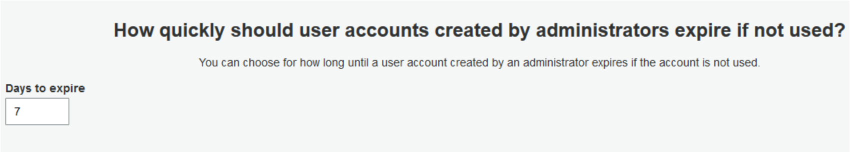
Do you want to allow users to sign themselves up?

You can choose to only allow administrators to create users or allow users to sign themselves up. Learn more

○ Only allow administrators to create users
● Allow users to sign themselves up

- Specify user account expiration time limit (in days) for new accounts. The default setting is 7 days, measured from the time when the user account is created. The maximum setting is 90 days. After the account expires, the user cannot log in to the account until the administrator updates the user's profile.

  Note
  Once the user has logged in, the account never expires.



How quickly should user accounts created by administrators expire if not used?

You can choose for how long until a user account created by an administrator expires if the account is not used.

Days to expire

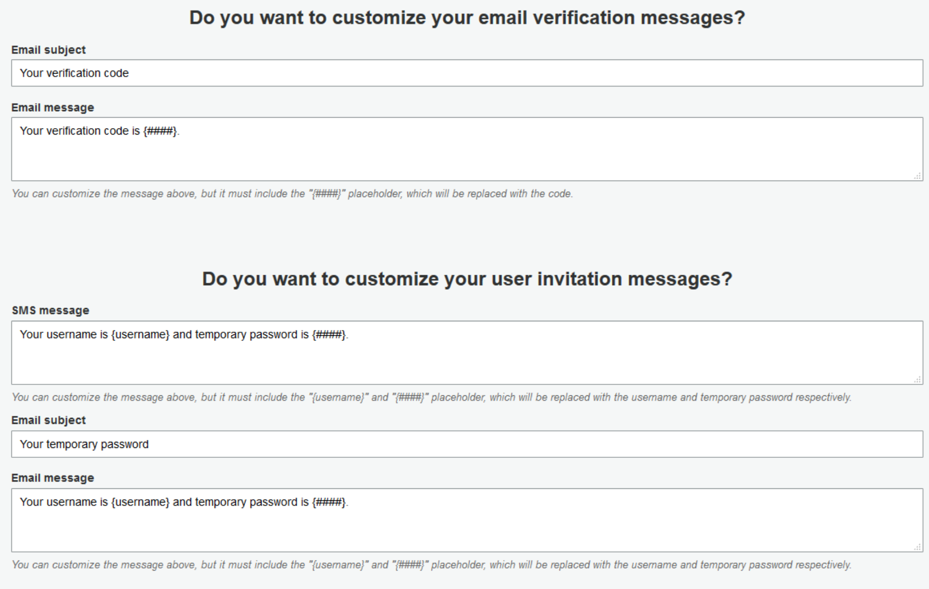
7

Message Customizations Tab

The **Message Customizations** tab includes templates for specifying custom email verification messages and custom user invitation messages.

For email (verification messages or user invitation messages), the maximum length for the message is 2048 UTF-8 characters, including the verification code or temporary password. For SMS, the maximum length is 140 UTF-8 characters, including the verification code or temporary password.

Verification codes are valid for 24 hours.



Do you want to customize your email verification messages?

Email subject

Your verification code

Email message

Your verification code is {####}.

You can customize the message above, but it must include the "{####}" placeholder, which will be replaced with the code.

Do you want to customize your user invitation messages?

SMS message

Your username is {username} and temporary password is {####}.

You can customize the message above, but it must include the "{username}" and "{####}" placeholder, which will be replaced with the username and temporary password respectively.

Email subject

Your temporary password

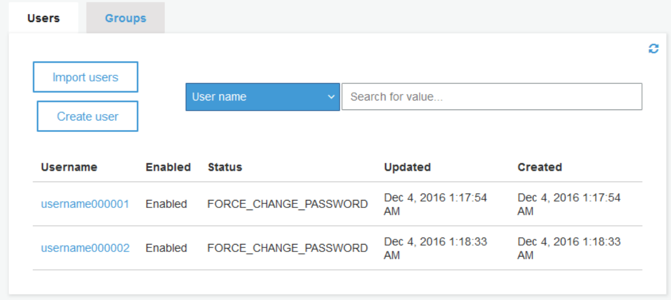
Email message

Your username is {username} and temporary password is {####}.

You can customize the message above, but it must include the "{username}" and "{####}" placeholder, which will be replaced with the username and temporary password respectively.

Users Tab

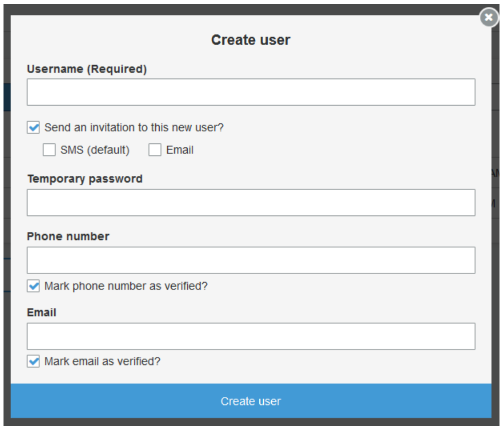
The **Users** tab in the **Users and groups** tab has a **Create user** button.



Username	Enabled	Status	Updated	Created
username000001	Enabled	FORCE_CHANGE_PASSWORD	Dec 4, 2016 1:17:54 AM	Dec 4, 2016 1:17:54 AM
username000002	Enabled	FORCE_CHANGE_PASSWORD	Dec 4, 2016 1:18:33 AM	Dec 4, 2016 1:18:33 AM

When you choose **Create user**, a **Create user** form appears, which you can use to enter information about the new user. Only the **Username** field is required.

Note
For user accounts that you create by using the **Create user** form in the AWS Management Console, only the attributes shown in the form can be set in the AWS Management Console. Other attributes must be set by using the AWS Command Line Interface or the Amazon Cognito API, even if you have marked them as required attributes.



User Groups

Support for groups in Amazon Cognito user pools enables you to create and manage groups, add users to groups, and remove users from groups. Use groups to create collections of users to manage their permissions or to represent different types of users. You can assign an AWS Identity and Access Management (IAM) role to a group to define the permissions for members of a group.

You can use groups to create a collection of users in a user pool, which is often done to set the permissions for those users. For example, you can create separate groups for users who are readers, contributors, and editors of your website and app. Using the IAM role associated with a group, you can also set different permissions for those different groups so that only contributors can put content into Amazon S3 and only editors can publish content through an API in Amazon API Gateway.

You can create and manage groups in a user pool from the AWS Management Console, the APIs, and the CLI. As a developer (using AWS credentials), you can create, read, update, delete, and list the groups for a user pool. You can also add users and remove users from groups.

There is no additional cost for using groups within a user pool. See Amazon Cognito Pricing for more information.

You can see this feature used in the SpaceFinder reference app.

Assigning IAM Roles to Groups

You can use groups to control permissions to access your resources in AWS by assigning an IAM role for the users within a group. When you create a group, you can specify the IAM role for users in that group by providing a role ARN for the group. IAM roles have associated policies that define the resources and actions that are allowed and denied for users. IAM roles and their permissions are tied to the temporary AWS credentials that Amazon Cognito identity pools provide for authenticated users. Users in a group are automatically assigned the IAM role for the group when AWS credentials are provided by Amazon Cognito Federated Identities using the **Choose role from token** option.

Individual users can be in multiple groups. As a developer, you have the following options for automatically choosing the IAM role when a user is in multiple groups:

- You can assign precedence values to each group. The group with the better (lower) precedence will be chosen and its associated IAM role will be applied.
- Your app can also choose from among the available roles when requesting AWS credentials for a user through an identity pool, by specifying a role ARN in the GetCredentialsForIdentity `CustomRoleARN` parameter. The specified IAM role must match a role that is available to the user.

Assigning Precedence Values to Groups

A user can belong to more than one group. In the user's ID token, the `cognito:groups` claim contains the list of all the groups a user belongs to. The `cognito:roles` claim contains the list of roles corresponding to the groups.

Because a user can belong to more than one group, each group can be assigned a precedence. This is a non-negative number that specifies the precedence of this group relative to the other groups that a user can belong to in the user pool. Zero is the top precedence value. Groups with lower precedence values take precedence over groups with higher or null precedence values. If a user belongs to two or more groups, it is the group with the lowest precedence value whose IAM role is applied to the `cognito:preferred_role` claim in the user's ID token.

Two groups can have the same precedence value. If this happens, neither group takes precedence over the other. If two groups with the same precedence value have the same role ARN, that role is used in the `cognito:preferred_role` claim in ID tokens for users in each group. If the two groups have different role ARNs, the `cognito:preferred_role` claim is not set in users' ID tokens.

Using Groups to Control Permission with Amazon API Gateway

You can use groups in a user pool to control permission with Amazon API Gateway. The groups that a user is a member of are included in the ID token provided by a user pool when a user signs in. You can submit those ID tokens with requests to Amazon API Gateway, use a custom authorizer Lambda function to verify the token, and then inspect which groups a user belongs to. See this blog post for an example of using user pool tokens with an Amazon API Gateway custom authorizer.

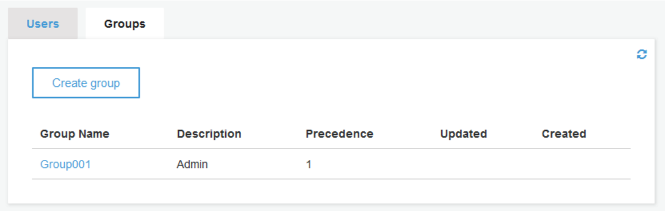
Limitations on Groups

User groups are subject to the following limitations:

- You can create up to 25 groups per user pool.
- Groups cannot be nested.
- You cannot search for users in a group.
- You cannot search for groups by name, but you can list groups.
- Only groups with no members can be deleted.

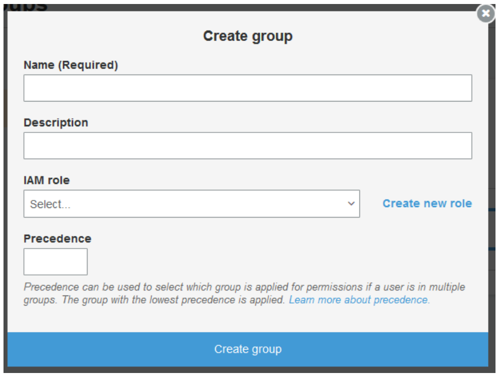
Creating a New Group in the AWS Management Console

The **Groups** tab in the **Users and groups** tab has a **Create group** button.



Group Name	Description	Precedence	Updated	Created
Group001	Admin	1		

When you choose **Create group**, a **Create group** form appears. This form is where you enter information about the new group. Only the **Name** field is required. If you are integrating a user pool with an identity pool, the **IAM role** setting determines which role is assigned in the user's ID token if the identity pool is configured to choose the role from the token. If you don't have roles already defined, choose **Create new role**. If you have more than one group, and your users can be assigned to more than one group, you can set a **Precedence** value for each group. The precedence value can be any non-negative integer. Zero is the top precedence value.



Precedence can be used to select which group is applied for permissions if a user is in multiple groups. The group with the lowest precedence is applied. Learn more about precedence.

Importing Users into User Pools

You can import users into an Amazon Cognito user pool. The user information is imported from a specially formatted .csv file. The import process sets values for all user attributes except **password**. Password import is not supported, because security best practices require that passwords are not available as plain text, and we don't support importing hashes. This means that your users must change their passwords the first time they sign in. The creation date for each user is the time when that user was imported into the user pool. (Creation date is not one of the imported attributes.)

The basic steps are:

1. Create an Amazon CloudWatch Logs role in the AWS Identity and Access Management (IAM) console.
2. Create the user import .csv file.
3. Create and run the user import job.
4. Upload the user import .csv file.
5. Start and run the user import job.
6. Use CloudWatch to check the event log.
7. Require the imported users to reset their passwords.

Creating the CloudWatch Logs IAM Role

If you're using the Amazon Cognito CLI or API, then you need to create a CloudWatch IAM role. The following procedure describes how to enable Amazon Cognito to record information in CloudWatch Logs about your user pool import job.

Note

You don't need to use this procedure if you are using the Amazon Cognito console, because the console creates the role for you.

To create the CloudWatch Logs IAM Role for user pool import

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles**.
3. Choose **Create New Role**.
4. Type a role name and choose **Next Step**.
5. In **Select Role Type**, choose **Amazon EC2**. You can choose any role type; you'll change this setting in a later step. This is because you can't create an IAM role from scratch; you can only use an existing IAM role as a template and overwrite it to make the role you need.
6. In **Attach Policy**, choose **Next Step**.
7. In **Review**, choose **Create Role**.
8. In **Roles**, choose the role you just created.
9. In **Summary**, choose **Permissions**.
10. On the **Permissions** tab, choose **Inline Policies**, and then choose **click here**.
11. In **Set Permissions**, choose **custom policy**, and then choose **select**.
12. In **Review Policy**, type a policy name (no spaces) and copy/paste the following text as your role access policy, replacing any existing text:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:REGION:ACCOUNT:log-group:/aws/cognito/*"
            ]
        }
    ]
}
```

13. Choose **Apply Policy**.
14. In **Summary**, choose the **Trust Relationships** tab.
15. Choose **Edit Trust Relationship**.
16. Copy/paste the following trust relationship text into the **Policy Document** text box, replacing any existing text:

```
{
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": "cognito-idp.amazonaws.com"
                },
                "Action": "sts:AssumeRole"
            }
        ]
    }
```

17. Choose **Update Trust Policy**. You are now finished creating the role.
18. Note the role ARN. You need this later when you're creating an import job.

Creating the User Import .csv File

Before you can import your existing users into your user pool, you must create a .csv file that serves as the input. To do this, you download the user import .csv header information, and then you edit the file to match the formatting requirements outlined in Formatting the .csv File (p. 111).

Downloading the .csv File Header By Using the AWS Management Console

1. Navigate to the Amazon Cognito console, choose **Manage User Pools**, and then choose the user pool that you are importing the users into.
2. Choose the **Users** tab.
3. Choose **Import users**.
4. Choose **Download CSV header** to get a .csv file containing the header row that you must include in your .csv file.

Downloading the .csv File Header By Using the CLI

To get a list of the correct headers, run the following CLI command, where *USER_POOL_ID* is the user pool identifier for the user pool you'll import users into:

```
aws cognito-idp get-csv-header --user-pool-id "USER_POOL_ID"
```

Sample response:

```
{
    "CSVHeader": [
        "name",
        "given_name",
        "family_name",
        "middle_name",
        "nickname",
        "preferred_username",
        "profile",
        "picture",
        "website",
        "email",
        "email_verified",
        "gender",
        "birthdate",
        "zoneinfo",
        "locale",
        "phone_number",
        "phone_number_verified",
        "address",
        "updated_at",
        "cognito:mfa_enabled",
        "cognito:username"
    ],
    "UserPoolId": "USER_POOL_ID"
}
```

Formatting the .csv File

The downloaded user import .csv header file looks like this:

```
cognito:username,name,given_name,family_name,middle_name,nickname,preferred_username,profile,picture,we
```

You'll need to edit your .csv file so that it includes this header and the attribute values for your users and is formatted according to the following rules.

> **Note**
> For more information about attribute values, such as proper format for phone numbers, see Specifying User Pool Attribute Settings (p. 14).

- The first line in the file is the downloaded header row that contains the user attribute names.
- The order of columns in the .csv file doesn't matter.
- Each line after the first line contains the attribute values for a user.
- All columns in the header must be present, but you don't need to provide values in every column.
- The following attributes are required:
  - **cognito:username**
  - **cognito:mfa_enabled**
  - **email_verified** or **phone_number_verified**
  - **email** (if **email_verified** is `true`)
  - **phone_number** (if **phone_number_verified** is `true`)
  - Any attributes that you marked as required when you created the user pool
- The user pool must have at least one auto-verified attribute, either **email_verified** or **phone_number_verified**. At least one of the auto-verified attributes must be `true` for each user. If the user pool has no auto-verified attributes, the import job will not start. If the user pool only has one auto-verified attribute, that attribute must be verified for each user. For example, if the user pool has only **phone_number** as an auto-verified attribute, the **phone_number_verified** value must be `true` for each user.

  > **Note**
  > In order for users to reset their passwords, they must have a verified email or phone number. Amazon Cognito sends a message containing a reset password code to the email or phone number specified in the .csv file. If the message is sent to the phone number, it is sent via SMS.
- Attribute values that are strings should *not* be in quotation marks.
- If an attribute value contains a comma, you must put a backslash (\) before the comma. This is because the fields in a .csv file are separated by commas.
- The .csv file contents should be in UTF-8 format without byte order mark.
- The **cognito:username** field is required and must be unique within your user pool. It can be any Unicode string. However, it cannot contain spaces or tabs.
- The **birthdate** values, if present, must be in the format *mm/dd/yyyy*. This means, for example, that a birthdate of February 1, 1985 must be encoded as 02/01/1985.
- The **cognito:mfa_enabled** field is required. If you've set multi-factor authentication (MFA) to be required in your user pool, this field must be `true` for all users. If you've set MFA to be off, this field must be `false` for all users. If you've set MFA to be optional, this field can be either `true` or `false`, but it cannot be empty.

- The maximum line length is 16,000 characters.
- The maximum .csv file size is 100 MB.
- The maximum number of lines (users) in the file is 500,000, not including the header.
- The **updated_at** field value is expected to be epoch time in seconds, for example: 1471453471.
- Any leading or trailing white space in an attribute value will be trimmed.

A complete sample user import .csv file looks like this:

```
cognito:username,name,given_name,family_name,middle_name,nickname,preferred_username,profile,picture,we
John,,John,Doe,,,,,,,johndoe@example.com,TRUE,,02/01/1985,,,+12345550100,TRUE,123 Any
 Street,,FALSE
Jane,,Jane,Roe,,,,,,,janeroe@example.com,TRUE,,01/01/1985,,,+12345550199,TRUE,100 Main
 Street,,FALSE
```

Creating and Running the Amazon Cognito User Pool Import Job

This section describes how to create and run the user pool import job by using the Amazon Cognito console and the AWS Command Line Interface.

Topics

- Importing Users from a .csv File By Using the Amazon Cognito Console (p. 112)
- Importing Users By Using the AWS CLI (p. 112)

Importing Users from a .csv File By Using the Amazon Cognito Console

The following procedure describes how to import the users from the .csv file.

To import users from the .csv file by using the Amazon Cognito console

1. Choose **Create import job**.
2. Type a **Job name**. Job names can contain uppercase and lowercase letters (a-z, A-Z), numbers (0-9), and the following special characters: + = , . @ and -.
3. If this is your first time creating a user import job, the AWS Management Console will automatically create an IAM role for you. Otherwise, you can choose an existing role from the **IAM Role** list or let the AWS Management Console create a new role for you.
4. Choose **Upload CSV** and select the .csv file to import users from.
5. Choose **Create job**.
6. To start the job, choose **Start**.

Importing Users By Using the AWS CLI

The following CLI commands are available for importing users into a user pool:

- `create-user-import-job`
- `get-csv-header`
- `describe-user-import-job`
- `list-user-import-jobs`

- `start-user-import-job`
- `stop-user-import-job`

To get the list of command-line options for these commands, use the `help` command-line option. For example:

```
aws cognito-idp get-csv-header help
```

Creating a User Import Job

After you create your .csv file, create a user import job by running the following CLI command, where *`JOB_NAME`* is the name you're choosing for the job, *`USER_POOL_ID`* is the same user pool ID as before, and *`ROLE_ARN`* is the role ARN you received in Creating the CloudWatch Logs IAM Role (p. 108):

```
aws cognito-idp create-user-import-job --job-name "JOB_NAME" --user-pool-id "USER_POOL_ID"
 --cloud-watch-logs-role-arn "ROLE_ARN"
```

The *`PRE_SIGNED_URL`* returned in the response is valid for 15 minutes. After that time, it will expire and you must create a new user import job to get a new URL.

Sample response:

```
{
    "UserImportJob": {
        "Status": "Created",
        "SkippedUsers": 0,
        "UserPoolId": "USER_POOL_ID",
        "ImportedUsers": 0,
        "JobName": "JOB_NAME",
        "JobId": "JOB_ID",
        "PreSignedUrl": "PRE_SIGNED_URL",
        "CloudWatchLogsRoleArn": "ROLE_ARN",
        "FailedUsers": 0,
        "CreationDate": 1470957431.965
    }
}
```

Status Values for a User Import Job

In the responses to your user import commands, you'll see one of the following `Status` values:

- "Created" - The job was created but not started.
- "Pending" - A transition state. You have started the job, but it has not begun importing users yet.
- "InProgress" - The job has started, and users are being imported.
- "Stopping" - You have stopped the job, but the job has not stopped importing users yet.
- "Stopped" - You have stopped the job, and the job has stopped importing users.
- "Succeeded" - The job has completed successfully.
- "Failed" - The job has stopped due to an error.
- "Expired" - You created a job, but did not start the job within 24-48 hours. All data associated with the job was deleted, and the job cannot be started.

Uploading the .csv File

Use the following `curl` command to upload the .csv file containing your user data to the pre-signed URL that you obtained from the response of the `create-user-import-job` command.

```
curl -v -T "PATH_TO_CSV_FILE" -H
        "x-amz-server-side-encryption:aws:kms" "PRE_SIGNED_URL"
```

In the output of this command, look for the phrase `"We are completely uploaded and fine"`. This phrase indicates that the file was uploaded successfully.

Describing a User Import Job

To get a description of your user import job, use the following command, where *`USER_POOL_ID`* is your user pool ID, and *`JOB_ID`* is the job ID that was returned when you created the user import job.

```
aws cognito-idp describe-user-import-job --user-pool-id "USER_POOL_ID" --job-id "JOB_ID"
```

Sample response:

```
{
    "UserImportJob": {
        "Status": "Created",
        "SkippedUsers": 0,
        "UserPoolId": "USER_POOL_ID",
        "ImportedUsers": 0,
        "JobName": "JOB_NAME",
        "JobId": "JOB_ID",
        "PreSignedUrl": "PRE_SIGNED_URL",
        "CloudWatchLogsRoleArn":"ROLE_ARN",
        "FailedUsers": 0,
        "CreationDate": 1470957431.965
    }
}
```

In the preceding sample output, the *`PRE_SIGNED_URL`* is the URL that you uploaded the .csv file to. The *`ROLE_ARN`* is the CloudWatch Logs role ARN that you received when you created the role.

Listing Your User Import Jobs

To list your user import jobs, use the following command:

```
aws cognito-idp list-user-import-jobs --user-pool-id "USER_POOL_ID" --max-results 2
```

Sample response:

```
{
    "UserImportJobs": [
        {
            "Status": "Created",
            "SkippedUsers": 0,
            "UserPoolId": "USER_POOL_ID",
            "ImportedUsers": 0,
            "JobName": "JOB_NAME",
            "JobId": "JOB_ID",
            "PreSignedUrl":"PRE_SIGNED_URL",
            "CloudWatchLogsRoleArn":"ROLE_ARN",
            "FailedUsers": 0,
            "CreationDate": 1470957431.965
        },
        {
            "CompletionDate": 1470954227.701,
            "StartDate": 1470954226.086,
            "Status": "Failed",
            "UserPoolId": "USER_POOL_ID",
            "ImportedUsers": 0,
            "SkippedUsers": 0,
            "JobName": "JOB_NAME",
            "CompletionMessage": "Too many users have failed or been skipped during the
 import.",
            "JobId": "JOB_ID",
            "PreSignedUrl":"PRE_SIGNED_URL",
            "CloudWatchLogsRoleArn":"ROLE_ARN",
            "FailedUsers": 5,
            "CreationDate": 1470953929.313
        }
    ],
    "PaginationToken": "PAGINATION_TOKEN"
}
```

Jobs are listed in chronological order from last created to first created. The `PAGINATION_TOKEN` string after the second job indicates that there are additional results for this list command. To list the additional results, use the `--pagination-token` option as follows:

```
aws cognito-idp list-user-import-jobs --user-pool-id "USER_POOL_ID" --max-results 10 --
pagination-token "PAGINATION_TOKEN"
```

Starting a User Import Job

To start a user import job, use the following command:

```
aws cognito-idp start-user-import-job --user-pool-id "USER_POOL_ID" --job-id "JOB_ID"
```

Only one user import job can be active at a time for a given user pool.

Sample response:

```
{
    "UserImportJob": {
        "Status": "Pending",
        "StartDate": 1470957851.483,
        "UserPoolId": "USER_POOL_ID",
        "ImportedUsers": 0,
        "SkippedUsers": 0,
        "JobName": "JOB_NAME",
        "JobId": "JOB_ID",
        "PreSignedUrl":"PRE_SIGNED_URL",
        "CloudWatchLogsRoleArn": "ROLE_ARN",
        "FailedUsers": 0,
        "CreationDate": 1470957431.965
    }
}
```

Stopping a User Import Job

To stop a user import job while it is in progress, use the following command. After you stop the job, it cannot be restarted.

```
aws cognito-idp stop-user-import-job --user-pool-id "USER_POOL_ID" --job-id "JOB_ID"
```

Sample response:

```
{
    "UserImportJob": {
        "CompletionDate": 1470958050.571,
        "StartDate": 1470958047.797,
        "Status": "Stopped",
        "UserPoolId": "USER_POOL_ID",
        "ImportedUsers": 0,
        "SkippedUsers": 0,
        "JobName": "JOB_NAME",
        "CompletionMessage": "The Import Job was stopped by the developer.",
        "JobId": "JOB_ID",
        "PreSignedUrl":"PRE_SIGNED_URL",
        "CloudWatchLogsRoleArn": "ROLE_ARN",
        "FailedUsers": 0,
        "CreationDate": 1470957972.387
    }
}
```

Viewing the User Pool Import Results in the CloudWatch Console

You can view the results of your import job in the Amazon CloudWatch console.

Topics

- Viewing the Results (p. 116)
- Interpreting the Results (p. 116)

Viewing the Results

The following steps describe how to view the user pool import results.

To view the results of the user pool import

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Logs**.
3. Choose the log group for your user pool import jobs. The log group name is in the form `/aws/cognito/userpools/USER_POOL_ID/USER_POOL_NAME`.
4. Choose the log for the user import job you just ran. The log name is in the form *`JOB_ID`*/*`JOB_NAME`*. The results in the log refer to your users by line number. No user data is written to the log. For each user, a line similar to the following appears:

   - `[SUCCEEDED] Line Number 5956 - The import succeeded.`
   - `[SKIPPED] Line Number 5956 - The user already exists.`
   - `[FAILED] Line Number 5956 - The User Record does not set any of the auto verified attributes to true. (Example: email_verified to true).`

Interpreting the Results

Successfully imported users have their status set to "PasswordReset".

In the following cases, the user will not be imported, but the import job will continue:

- No auto-verified attributes are set to `true`.
- The user data doesn't match the schema.
- The user couldn't be imported due to an internal error.

In the following cases, the import job will fail:

- The Amazon CloudWatch Logs role cannot be assumed, doesn't have the correct access policy, or has been deleted.
- The user pool has been deleted.
- Amazon Cognito is unable to parse the .csv file.

Requiring Imported Users to Reset Their Passwords

The first time each imported user signs in, he or she is required to enter a new password as follows:

Requiring imported users to reset their passwords

1. The user attempts to sign in, providing username and password (via `GetAuthenticationDetails` or `InitiateAuth`).
2. Amazon Cognito returns `PasswordResetRequiredException`.
3. The app should direct the user into the `ForgotPassword` flow as outlined in the following procedure:
    1. The app calls `ForgotPassword(username)`.
    2. Amazon Cognito sends a code to the verified email or phone number (depending on what you have provided in the .csv file for that user) and indicates to the app where the code was sent in the response to the `ForgotPassword` request.

        Note
        For sending reset password codes, it is important that your user pool has phone number or email verification turned on.
    3. The app indicates to the user that a code was sent and where the code was sent, and the app provides a UI to enter the code and a new password.
    4. The user enters the code and new password in the app.
    5. The app calls `ConfirmForgotPassword(code, password)`, which, if successful, sets the new password.
    6. The app should now direct the user to a sign-in page.

Signing Up and Confirming User Accounts

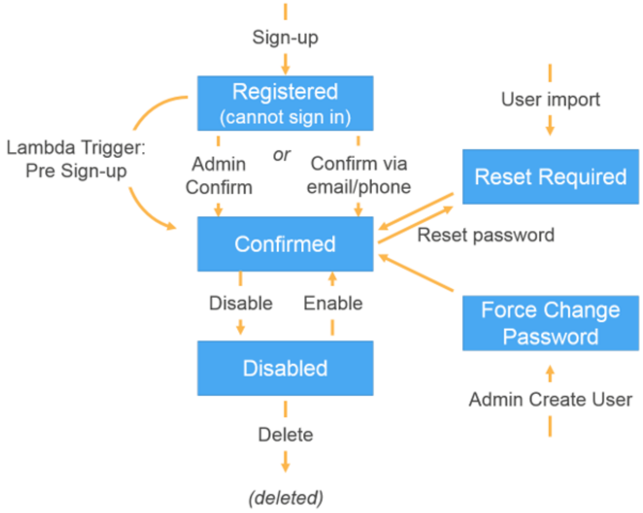
User accounts are added to your user pool in one of the following ways:

- The user signs up in your user pool's client app, which can be a mobile or web app.
- You can import the user's account into your user pool. For more information, see Importing Users into User Pools (p. 108).
- You can create the user's account in your user pool and invite the user to sign in. For more information, see Creating User Accounts as Administrator in the AWS Management Console and with the Amazon Cognito User Pools API (p. 103).

Users who sign themselves up need to be confirmed before they can sign in. Imported and created users are already confirmed, but they need to create their password the first time they sign in. The following sections explain the confirmation process and email and phone verification.

Overview of User Account Confirmation

The following diagram illustrates the confirmation process:


Sign-up
Registered (cannot sign in)
Lambda Trigger: Pre Sign-up
Admin Confirm
or
Confirm via email/phone
User import
Reset Required
Confirmed
Reset password
Disable
Enable
Force Change Password
Disabled
Admin Create User
Delete
(deleted)

A user account can be in any of the following states:

Registered (Unconfirmed)

The user has successfully signed up, but cannot sign in until the user account is confirmed. The user is enabled but not confirmed in this state.

New users who sign themselves up start in this state.

Confirmed

The user account is confirmed and the user can sign in. If the user confirmed the user account by entering a confirmation code that was received via email or phone (SMS)—or, in the case of email, by clicking a confirmation link—that email or phone number is automatically verified. If the user account was confirmed by the administrator or a Pre Sign-up Lambda trigger, there might not be a verified email or phone number associated with the account.

Password Reset Required

The user account is confirmed, but the user must request a code and reset his or her password before he or she can sign in.

User accounts that are imported by an administrator or developer start in this state.

Force Change Password

The user account is confirmed and the user can sign in using a temporary password, but on first sign-in, the user must change his or her password to a new value before doing anything else.

User accounts that are created by an administrator or developer start in this state.

Disabled

Before a user account can be deleted, it must be disabled.

Allowing Users to Sign Up and Confirm Themselves and Verify Email or Phone

1. A user signs up in your app by entering a username, phone number and/or email address, and possibly other attributes.

2. The Amazon Cognito service receives the sign-up request from the app. After verifying that the request contains all attributes required for sign-up, the service completes the sign-up process and sends a confirmation code to the user's phone (via SMS) or email.

    Note
    If a user signs up with both a phone number and an email address, and your user pool settings require verification of both attributes, a verification code is sent via SMS to the phone. *The email address is not verified.* Your app can call GetUser to see if an email address is awaiting verification. If it is, the app should call GetUserAttributeVerificationCode to initiate the email verification flow and then submit the verification code by calling VerifyUserAttribute.

3. The service returns to the app that sign-up is complete and that the user account is pending confirmation. The response contains information about where the confirmation code was sent. At this point the user's account is in an unconfirmed state, and the user's email address and phone number are unverified.
4. The app can now prompt the user to enter the confirmation code. It is not necessary for the user to enter the code immediately. However, the user will not be able to sign in until after they enter the confirmation code.
5. The user enters the confirmation code in the app.
6. The app calls ConfirmSignUp to send the code to the Amazon Cognito service, which verifies the code and, if the code is correct, sets the user's account to the confirmed state. After successfully confirming the user account, the Amazon Cognito service automatically marks the attribute that was used to confirm (email or phone number) as verified. Unless the value of this attribute is changed, the user will not have to verify it again.
7. At this point the user's account is in a confirmed state, and the user can sign in.

Allowing Users to Sign Up in Your App but Confirming Them as Administrator

1. A user signs up in your app by entering a username, phone number and/or email address, and possibly other attributes.
2. The Amazon Cognito service receives the sign-up request from the app. After verifying that the request contains all attributes required for sign-up, the service completes the sign-up process and returns to the app that sign-up is complete, pending confirmation. At this point the user's account is in an unconfirmed state. The user cannot sign in until the account is confirmed.
3. The administrator confirms the user's account, either in the Amazon Cognito console (by finding the user account in the **Users** tab and choosing the **Confirm** button) or in the CLI (by using the `admin-confirm-sign-up` command). Both the **Confirm** button and the `admin-confirm-sign-up` command use the AdminConfirmSignUp API to perform the confirmation.
4. At this point the user's account is in a confirmed state, and the user can sign in.

Computing SecretHash Values

The following Amazon Cognito User Pools APIs have a `SecretHash` parameter:

- ConfirmForgotPassword
- ConfirmSignUp
- ForgotPassword
- ResendConfirmationCode
- SignUp

The `SecretHash` value is a Base 64-encoded keyed-hash message authentication code (HMAC) calculated using the secret key of a user pool client and username plus the client ID in the message. The following pseudocode shows how this value is calculated. In this pseudocode, `+` indicates concatenation, `HMAC_SHA256` represents a function that produces an HMAC value using HmacSHA256, and `Base64` represents a function that produces Base-64-encoded version of the hash output.

```
Base64 ( HMAC_SHA256 ( "Client Secret Key", "Username" + "Client Id" ) )
```

Alternatively, you can use the following code example in your server-side Java application code:

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

public static String calculateSecretHash(String userPoolClientId, String
 userPoolClientSecret, String userName) {
    final String HMAC_SHA256_ALGORITHM = "HmacSHA256";

    SecretKeySpec signingKey = new SecretKeySpec(
            userPoolClientSecret.getBytes(StandardCharsets.UTF_8),
            HMAC_SHA256_ALGORITHM);
    try {
        Mac mac = Mac.getInstance(HMAC_SHA256_ALGORITHM);
        mac.init(signingKey);
        mac.update(userName.getBytes(StandardCharsets.UTF_8));
        byte[] rawHmac = mac.doFinal(userPoolClientId.getBytes(StandardCharsets.UTF_8));
        return Base64.getEncoder().encodeToString(rawHmac);
    } catch (Exception e) {
        throw new RuntimeException("Error while calculating ");
    }
}
```

Confirming User Accounts Without Verifying Email or Phone Number

The Pre-Sign Up Lambda trigger can be used to auto-confirm user accounts at sign-up time, without requiring a confirmation code or verifying email or phone number. Users who are confirmed this way can immediately sign in without having to receive a code.

You can also mark a user's email or phone number verified through this trigger.

> **Note**
> While this approach is convenient for users when they're getting started, we recommend auto-verifying at least one of email or phone number. Otherwise the user can be left unable to recover if they forget their password.

If you don't require the user to receive and enter a confirmation code at sign-up and you don't auto-verify email and phone number in the Pre-Sign Up Lambda trigger, you risk not having a verified email address or phone number for that user account. The user can verify the email address or phone number at a later time. However, if the user forgets his or her password and doesn't have a verified email address or phone number, the user is locked out of the account, because the Forgot Password flow requires a verified email or phone number in order to send a verification code to the user.

Verifying When Users Change Their Email or Phone Number

When a user changes his or her email address or phone number in your app, that attribute is marked as unverified. If auto-verification was enabled for the attribute being updated, the service immediately sends the user a message containing a verification code, which the user should enter to verify the change. You can use a Custom Message Lambda trigger to customize this message. For more information, see Customizing User Pool Workflows by Using AWS Lambda Triggers (p. 67). Whenever the user's email address or phone number is unverified, your app should display the unverified status and provide a button or link for users to verify their new email or phone number.

Confirmation and Verification Processes for User Accounts Created by Administrators or Developers

User accounts that are created by an administrator or developer are already in the confirmed state, so users aren't required to enter a confirmation code. The invitation message that the Amazon Cognito service sends to these users includes the username and a temporary password. The user is required to change the password before signing in. For more information, see the Message Customizations Tab (p. 105) in Creating User Accounts as Administrator in the AWS Management Console and with the Amazon Cognito User Pools API (p. 103) and the Custom Message trigger in Customizing User Pool Workflows by Using AWS Lambda Triggers (p. 67).

Confirmation and Verification Processes for Imported User Accounts

User accounts that are created by using the user import feature in the AWS Management Console, CLI, or API (see Importing Users into User Pools (p. 108)) are already in the confirmed state, so users aren't required to enter a confirmation code. No invitation message is sent. However, imported user accounts require users to first request a code by calling the `ForgotPassword` API and then create a password using the delivered code by calling `ConfirmForgotPassword` API before they sign in. For more information, see Requiring Imported Users to Reset Their Passwords (p. 117).

Either the user's email or phone number must be marked as verified when the user account is imported, so no verification is required when the user signs in.

Managing and Searching for User Accounts in the AWS Management Console and in the Amazon Cognito User Pools API

Once you create your user pool, you can view and manage users using the AWS Management Console, as well as the AWS Command Line Interface or the Amazon Cognito API. This topic describes how you can view and search for users using the AWS Management Console.

Viewing User Attributes

There are a number of operations you can perform in the AWS Management Console:

- You can view the **Pool details** and edit user pool attributes, password policies, MFA settings, apps, and triggers. For more information, see Step Through Amazon Cognito User Pool Settings in the AWS Management Console (p. 13).
- You can view the users in your user pool and drill down for more details.
- You can also view the details for an individual user in your user pool.
- You can also search for a user in your user pool.

To manage user pools using the AWS Management Console

1. From the Amazon Cognito home page in the AWS Management Console, choose **Manage your user identities**.
2. Choose your user pool from the **Your User Pools** page.
3. Choose **User and Groups** to view user information.
4. Choose a user name to show more information about an individual user. From this screen, you can perform any of the following actions:

    - **Add user to group**
    - **Reset user password**
    - **Confirm user**
    - **Enable or disable MFA**
    - **Delete user**

    The **Reset user password** action results in a confirmation code being sent to the user immediately and disables the user's current password by changing the user state to RESET_REQUIRED. The **Enable MFA** action results in a confirmation code being sent to the user when the user tries to log in. The **Reset user password** code is valid for 1 hour. The MFA code is valid for 3 minutes.

Searching User Attributes

If you have already created a user pool, you can search from the **Users** panel in the AWS Management Console. You can also use the Amazon Cognito ListUsers API, which accepts a **Filter** parameter.

You can search for any of the following standard attributes. Custom attributes are not searchable.

- username (case-sensitive)
- email
- phone_number
- name
- given_name
- family_name
- preferred_username
- cognito:user_status (called **Status** in the Console) (case-insensitive)
- status (called **Enabled** in the Console) (case-sensitive)

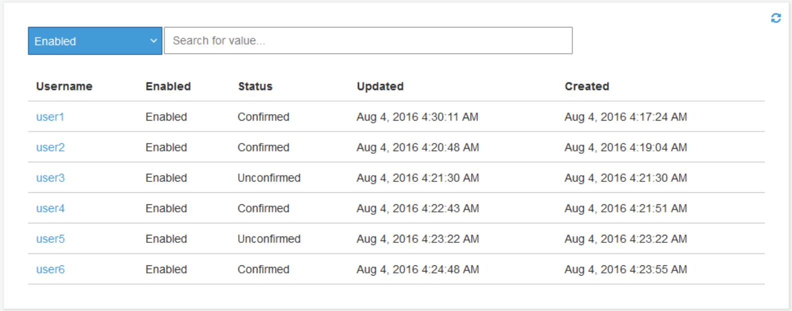
Searching for Users Using the AWS Management Console

If you have already created a user pool, you can search from the **Users** panel in the AWS Management Console.

AWS Management Console searches are always prefix ("starts with") searches.

All of the following examples use the same user pool.

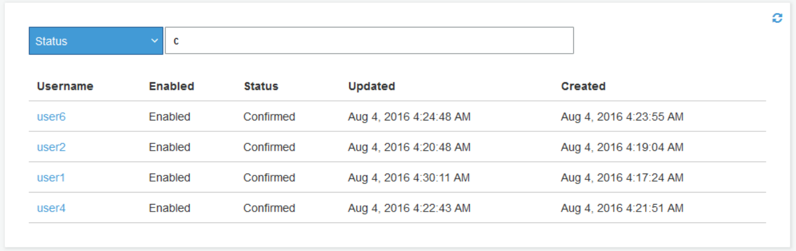
For example, if you want to list all users, leave the search box empty.



Enabled

Username	Enabled	Status	Updated	Created
user1	Enabled	Confirmed	Aug 4, 2016 4:30:11 AM	Aug 4, 2016 4:17:24 AM
user2	Enabled	Confirmed	Aug 4, 2016 4:20:48 AM	Aug 4, 2016 4:19:04 AM
user3	Enabled	Unconfirmed	Aug 4, 2016 4:21:30 AM	Aug 4, 2016 4:21:30 AM
user4	Enabled	Confirmed	Aug 4, 2016 4:22:43 AM	Aug 4, 2016 4:21:51 AM
user5	Enabled	Unconfirmed	Aug 4, 2016 4:23:22 AM	Aug 4, 2016 4:23:22 AM
user6	Enabled	Confirmed	Aug 4, 2016 4:24:48 AM	Aug 4, 2016 4:23:55 AM

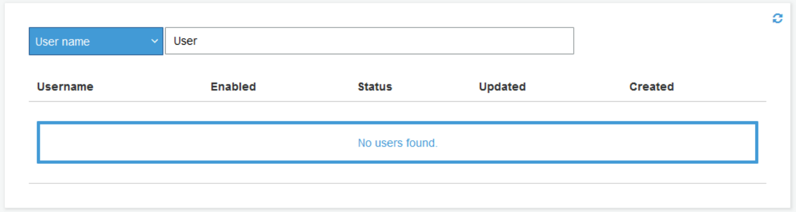
If you want to search for all confirmed users, choose **Status** from the drop-down menu. In the search box, type the first letter of the word "confirmed."



Status
c

Username	Enabled	Status	Updated	Created
user6	Enabled	Confirmed	Aug 4, 2016 4:24:48 AM	Aug 4, 2016 4:23:55 AM
user2	Enabled	Confirmed	Aug 4, 2016 4:20:48 AM	Aug 4, 2016 4:19:04 AM
user1	Enabled	Confirmed	Aug 4, 2016 4:30:11 AM	Aug 4, 2016 4:17:24 AM
user4	Enabled	Confirmed	Aug 4, 2016 4:22:43 AM	Aug 4, 2016 4:21:51 AM

Note that some attribute values are case-sensitive, such as **User name**.



User name
User

Username	Enabled	Status	Updated	Created

No users found.

Searching for Users Using the `ListUsers` API

To search for users from your app, use the Amazon Cognito ListUsers API. This API uses the following parameters:

- `AttributesToGet`: An array of strings, where each string is the name of a user attribute to be returned for each user in the search results. If the array is empty, all attributes are returned.
- `Filter`: A filter string of the form "`AttributeName` `Filter-Type` "`AttributeValue`"". Quotation marks within the filter string must be escaped using the backslash (\) character. For example, `"family_name = \"Reddy\""`. If the filter string is empty, `ListUsers` returns all users in the user pool.
  - `AttributeName`: The name of the attribute to search for. You can only search for one attribute at a time.

    Note
    You can only search for standard attributes. Custom attributes are not searchable. This is because only indexed attributes are searchable, and custom attributes cannot be indexed.
  - `Filter-Type`: For an exact match, use `=`, for example, `given_name = "Jon"`. For a prefix ("starts with") match, use `^=`, for example, `given_name ^= "Jon"`.
  - `AttributeValue`: The attribute value that must be matched for each user.
- `Limit`: Maximum number of users to be returned.
- `PaginationToken`: A token to get more results from a previous search.
- `UserPoolId`: The user pool ID for the user pool on which the search should be performed.

All searches are case-insensitive. Search results are sorted by the attribute named by the `AttributeName` string, in ascending order.

Examples of Using the `ListUsers` API

The following example returns all users and includes all attributes.

```
{
    "AttributesToGet": [],
    "Filter": "",
    "Limit": 10,
    "UserPoolId": "us-east-1_samplepool"
}
```

The following example returns all users whose phone numbers start with "+1312" and includes all attributes.

```
{
    "AttributesToGet": [],
    "Filter": "phone_number ^= \"+1312\"",
    "Limit": 10,
    "UserPoolId": "us-east-1_samplepool"
}
```

The following example returns the first 10 users whose family name is "Reddy". For each user, the search results include the user's given name, phone number, and email address. If there are more than 10 matching users in the user pool, the response includes a pagination token.

```
{
    "AttributesToGet": [
        "given_name", "phone_number", "email"
    ],
    "Filter": "family_name = \"Reddy\"",
    "Limit": 10,
    "UserPoolId": "us-east-1_samplepool"
}
```

If the previous example returns a pagination token, the following example returns the next 10 users that match the same filter string.

```
{
    "AttributesToGet": [
        "given_name", "phone_number", "email"
    ],
    "Filter": "family_name = \"Reddy\"",
    "Limit": 10,
    "PaginationToken": "pagination_token_from_previous_search",
    "UserPoolId": "us-east-1_samplepool"
}
```

Amazon Cognito User Pool Authentication Flow

Custom Authentication Flow

The custom authentication flow is designed to allow for a series of challenge and response cycles that can be customized to meet different requirements. The flow starts with a call to the **InitiateAuth** API that indicates the type of authentication that will be used and provides any initial authentication parameters. Amazon Cognito will respond to the **InitiateAuth** call with either:

- ID, access, and refresh tokens if the user is signed in
- A challenge for the user along with a session and parameters
- An error if the user fails to authenticate

If Amazon Cognito responds to the **InitiateAuth** call with a challenge, the app will gather more input and call the **RespondToAuthChallenge** API, providing the challenge responses and passing back the session. Amazon Cognito responds to the **RespondToAuthChallenge** call similarly to the **InitiateAuth** call, providing tokens if the user is signed in, another challenge, or an error. If another challenge is returned, the sequence repeats with the app calling **RespondToAuthChallenge** until the user is signed in or an error is returned. More details are provided in the API documentation for the **InitiateAuth** and **RespondToAuthChallenge** APIs.

Amazon Cognito has some built-in `AuthFlow` and `ChallengeName` values for a standard authentication flow to validate username and password through the Secure Remote Password (SRP) protocol. This flow is built into the iOS, Android, and JavaScript SDKs for Amazon Cognito. At a high level, the flow starts by sending `USER_SRP_AUTH` as the `AuthFlow` to **InitiateAuth** along with `USERNAME` and `SRP_A` values in **AuthParameters**. If the **InitiateAuth** call is successful, the response will include `PASSWORD_VERIFIER` as the `ChallengeName` and `SRP_B` in the challenge parameters. The app will then call **RespondToAuthChallenge** with the `PASSWORD_VERIFIER` `ChallengeName` and the necessary parameters in **ChallengeResponses**. If the call to **RespondToAuthChallenge** is successful and the user is signed in, the tokens will be returned. If multi-factor authentication (MFA) is enabled for the user, a `ChallengeName` of `SMS_MFA` will be returned, and the app can provide the necessary code through another call to **RespondToAuthChallenge**.

An app can initiate a custom authentication flow by calling **InitiateAuth** with `CUSTOM_AUTH` as the `Authflow`. With a custom authentication flow, the challenges and verification of the responses are controlled through three AWS Lambda triggers. The **DefineAuthChallenge** Lambda trigger takes as input a session array of previous challenges and responses and outputs the next challenge name and booleans indicating if the user is authenticated (and should be granted tokens) or if the authentication has failed. This Lambda trigger is a state machine that controls the user's path through the challenges. The **CreateAuthChallenge** Lambda trigger takes a challenge name as input and generates the challenge and parameters to evaluate the response. **CreateAuthChallenge** is called when **DefineAuthChallenge** returns `CUSTOM_CHALLENGE` as the next challenge, and the next type of challenge is passed in the challenge metadata parameter. The **VerifyAuthChallengeResponse** Lambda function evaluates the response and returns a boolean to indicate if the response was valid.

A custom authentication flow can also use a combination of built-in challenges such as SRP password verification and MFA via SMS, and custom challenges such as CAPTCHA or secret questions. If you want to include SRP in a custom authentication flow, you need to start with it. To initiate SRP password verification, the **DefineAuthChallenge** Lambda trigger returns `SRP_A` as the challenge name and `SRP_A` in the authentication parameters map. Once the password is verified the **DefineAuthChallenge** Lambda trigger will be called again with `PASSWORD_VERIFIER` in the previous challenges array. MFA will be done automatically if it is enabled for a user.

For more information about the Lambda triggers, including sample code, see Customizing User Pool Workflows by Using AWS Lambda Triggers (p. 67).

Admin Authentication Flow

The APIs described Custom Authentication Flow (p. 125) with the use of SRP for password verification is the recommended approach for authentication. The iOS, Android, and JavaScript SDKs are based on that approach and make it easy to use SRP. However, there is an alternative set of admin APIs designed for use on secure backend servers if you want to avoid the SRP calculations. For these back-end admin implementations, **AdminInitiateAuth** is used in place of **InitiateAuth**, and **AdminRespondToAuthChallenge** is used in place of **RespondToAuthChallenge**. When using these APIs, the password can be submitted as plain text so the SRP calculations are not needed. For example,

```
AdminInitiateAuth Request {
    "AuthFlow":"ADMIN_NO_SRP_AUTH",
    "AuthParameters":{
        "USERNAME":"<username>",
        "PASSWORD":"<password>"
        },
    "ClientId":"<clientId>",
    "UserPoolId":"<userPoolId>"
}
```

These admin authentication APIs require developer credentials and use the AWS Signature Version 4 (SigV4) signing process. These APIs are available in standard AWS SDKs including Node.js, which is convenient for use in Lambda functions. In order to use these APIs and have them accept passwords in plain text, you must enable them for the app in the console or by passing `ADMIN_NO_SRP_AUTH` for the `ExplicitAuthFlow` parameter in calls to **CreateUserPoolClient** or **UpdateUserPoolClient**. The `ADMIN_NO_SRP_AUTH AuthFlow` is not accepted for the **InitiateAuth** and **RespondToAuthChallenge** APIs.

In the **AdminInitiateAuth** response `ChallengeParameters`, the `USER_ID_FOR_SRP` attribute, if present, will contain the user's actual username, not an alias (such as email address or phone number). In your call to **AdminRespondToAuthChallenge**, in the `ChallengeResponses`, you'll need to pass this username in the `USERNAME` parameter.

> **Note**
> Because it's designed for back-end admin implementations, admin authentication flow doesn't support device tracking. When device tracking is enabled, admin authentication succeeds, but any call to refresh the access token will fail.

Integrating User Pools with Federated Identities

Amazon Cognito user pools represent an identity provider that you manage. To enable users in your user pool to access AWS resources through your client apps, you must configure Amazon Cognito Federated Identities to accept users that are federated with your user pool.

Setting Up a User Pool

Create an Amazon Cognito user pool and make a note of the **User Pool ID** and **App Client ID** for each of your client apps. For more information about creating user pools, see Amazon Cognito User Pools (p. 9). For more information about creating apps (to get app client IDs) for your client apps, see Specifying User Pool App Settings (p. 24).

You can create multiple user pools, and each user pool can have multiple apps.

Configuring Your Identity Pool Using the AWS Management Console

The following procedure describes how to use the AWS Management Console to integrate an identity pool with one or more user pools and client apps.

To configure your identity pool

1. Open the Amazon Cognito console.
2. Choose **Manage Federated Identities**.
3. Choose the name of the identity pool for which you want to enable Amazon Cognito user pools as a provider.
4. On the **Dashboard** page, choose **Edit identity pool**.
5. Expand the **Authentication providers** section.
6. Choose **Cognito**.
7. Type the **User Pool ID**.
8. Type the **App Client ID**. This must be the same client app ID that you received when you created the app in the **Your User Pools** section of the AWS Management Console for Amazon Cognito.
9. If you have additional apps or user pools, choose **Add Another Provider** and type the **User Pool ID** and **App Client ID** for each app in each user pool.
10. When you have no more apps or user pools to add, choose **Save changes**.

    If successful, you will see **Changes saved successfully.** on the **Dashboard** page.

Using Amazon Cognito User Pools

Follow the instructions in Authentication Flow (p. 136) to authenticate users.

After the user is authenticated, add that user's identity token to the logins map in the credentials provider. The provider name will depend on your Amazon Cognito user pool ID. It will have the following structure:

```
cognito-idp.<region>.amazonaws.com/<YOUR_USER_POOL_ID>
```

The value for *<region>* will be the same as the region in the **User Pool ID**. For example, `cognito-idp.us-east-1.amazonaws.com/us-east-1_123456789`.

iOS - Objective-C

```
AWSServiceConfiguration *serviceConfiguration = [[AWSServiceConfiguration alloc]
 initWithRegion:AWSRegionUSEast1 credentialsProvider:nil];
AWSCognitoIdentityUserPoolConfiguration *userPoolConfiguration =
 [[AWSCognitoIdentityUserPoolConfiguration alloc] initWithClientId:@"YOUR_CLIENT_ID"
 clientSecret:@"YOUR_CLIENT_SECRET" poolId:@"YOUR_USER_POOL_ID"];
[AWSCognitoIdentityUserPool
 registerCognitoIdentityUserPoolWithConfiguration:serviceConfiguration
 userPoolConfiguration:userPoolConfiguration forKey:@"UserPool"];
AWSCognitoIdentityUserPool *pool = [AWSCognitoIdentityUserPool
 CognitoIdentityUserPoolForKey:@"UserPool"];
AWSCognitoCredentialsProvider *credentialsProvider = [[AWSCognitoCredentialsProvider
 alloc] initWithRegionType:AWSRegionUSEast1 identityPoolId:@"YOUR_IDENTITY_POOL_ID"
 identityProviderManager:pool];
```

iOS - Swift

```
let serviceConfiguration = AWSServiceConfiguration(region: .USEast1, credentialsProvider:
 nil)
let userPoolConfiguration = AWSCognitoIdentityUserPoolConfiguration(clientId:
 "YOUR_CLIENT_ID", clientSecret: "YOUR_CLIENT_SECRET", poolId: "YOUR_USER_POOL_ID")
AWSCognitoIdentityUserPool.registerCognitoIdentityUserPoolWithConfiguration(serviceConfiguration,
 userPoolConfiguration: userPoolConfiguration, forKey: "UserPool")
let pool = AWSCognitoIdentityUserPool(forKey: "UserPool")
let credentialsProvider = AWSCognitoCredentialsProvider(regionType: .USEast1,
 identityPoolId: "YOUR_IDENTITY_POOL_ID", identityProviderManager:pool)
```

Android

```
cognitoUser.getSessionInBackground(new AuthenticationHandler() {
 @Override
 public void onSuccess(CognitoUserSession session) {
  String idToken = session.getIdToken().getJWTToken();

  Map<String, String> logins = new HashMap<String, String>();
  logins.put(cognito-idp.<region>.amazonaws.com/<YOUR_USER_POOL_ID>,
 session.getIdToken().getJWTToken());
  credentialsProvider.setLogins(logins);
 }

});
```

JavaScript

```
var cognitoUser = userPool.getCurrentUser();

if (cognitoUser != null) {
 cognitoUser.getSession(function(err, result) {
  if (result) {
   console.log('You are now logged in.');

   // Add the User's Id Token to the Cognito credentials login map.
   AWS.config.credentials = new AWS.CognitoIdentityCredentials({
    IdentityPoolId: 'YOUR_IDENTITY_POOL_ID',
    Logins: {
     'cognito-idp.<region>.amazonaws.com/<YOUR_USER_POOL_ID>':
 result.getIdToken().getJwtToken()
    }
   });
  }
 });
}
```

Using Tokens with User Pools

After successful authentication of a user, Amazon Cognito issues three tokens to the client:

- ID token
- Access token
- Refresh token

Important
We strongly recommended that you secure all three tokens in transit and storage in the context of your application.

Using the ID Token

The ID token is represented as a JSON Web Key Token (JWT). The token contains claims about the identity of the authenticated user. For example, it includes claims such as `name`, `family_name`, `phone_number`, etc. For more information about standard claims, see the OpenID Connect specification. A client app can use this identity information inside the application. The ID token can also be used to authenticate users against your resource servers or server applications. When an ID token is used outside of the application against your web APIs, you must verify the signature of the ID token before you can trust any claims inside the ID token.

The ID token expires one hour after the user authenticates. You should not process the ID token in your client or web API after it has expired.

Using the Access Token

The access token is also represented as a JSON Web Key Token (JWT). It contains claims about the authenticated user, but unlike the ID token, it does not include all of the user's identity information. The primary purpose of the access token is to authorize operations in the context of the user in the user pool. For example, you can use the access token against Amazon Cognito Identity to update or delete user attributes. The access token can also be used with any of your web APIs to make access control decisions and authorize operations in the context of the user. As with the ID token, you must first verify the signature of the access token in your web APIs before you can trust any claims inside the access token.

The access token expires one hour after the user authenticates. It should not be processed after it has expired.

Using the Refresh Token

The refresh token can only be used against Amazon Cognito to retrieve a new access or ID token.

Note
If you're using the Mobile SDK for iOS or Android, your access token and ID token will be refreshed automatically. In these SDKs, the expiration limit for these tokens is 5 minutes by default, but you can increase that value up to 30 minutes.

By default, the refresh token expires 30 days after the user authenticates. When you create an app for your user pool, you can set the app's **Refresh token expiration (days)** to any value between 1 and 3650.

Note
The user account itself never expires, as long as the user has logged in at least once before the `UnusedAccountValidityDays` time limit for new accounts.

To use the refresh token to get new tokens, use the AdminInitiateAuth API, passing `REFRESH_TOKEN_AUTH` for the `AuthFlow` parameter and the refresh token for the `AuthParameters` parameter with key "REFRESH_TOKEN". This initiates the token refresh process with the Amazon Cognito server and returns new ID and access tokens.

Structure of ID Tokens

ID tokens are JSON Web Key Tokens (JWT) and can be broken down into three parts: a header, a payload, and a signature.

Header

The header contains two pieces of information: the `kid` and the `alg`. A `kid` value is used to locate the public key. The public key should verify the ID token signature. The `alg` value represents the cryptographic algorithm used to secure `IdToken`. Currently, user pools only use RS256 as the cryptographic algorithm. For more information, see JSON Web Key Token (JWT).

For example, the header will look like this:

```
{

"alg" : "RS256",
"kid" : "samplekid****"

}
```

Payload

The payload contains claims as per the JWT specification. For more information, see RFC7519. The following are details of some specific claims:

- `iss` : The issuer. It has the following format: `https://cognito-idp.{region}.amazonaws.com/{userPoolId}`. For example, if you created a user pool in the `us-east-1` region and its ID is `u123456`, the ID token issued for users of your user pool have an `iss` claim value of `https://cognito-idp.us-east-1.amazonaws.com/u123456`.
- `sub` : The UUID of the authenticated user. This is not the same as `username`.
- `aud` : Contains the `client_id` with which the user authenticated.
- `token_use` : The intended purpose of this token. Its value is always `id` in the case of the ID token.

Additionally, the ID token contains standard claims defined in the OIDC Core spec, Section 5.1. It also contains the custom attributes that you define in your user pool. The custom attributes are always prefixed with the `custom:` prefix.

Signature

The signature of the ID token is calculated based on the header and payload of the ID token. When used outside of an application in your web APIs, you must always verify this signature before processing the ID token.

Structure of Access Tokens

Access tokens are also JSON Web Tokens (JWT) and can be broken down into three parts: a header, a payload, and a signature.

Header

The header for the access token will be the same structure as the ID token, but the `kid` will be different because different keys are used to sign ID tokens and access tokens.

Payload

The payload contains claims as per the JWT specification. For more information, see RFC7519. The following are details of some specific claims:

- `iss` : The issuer. It has the following format: `https://cognito-idp.{region}.amazonaws.com/{userPoolId}`. For example, if you created a user pool in the `us-east-1` region and its ID is `u123456`, the ID token issued for users of your user pool have an `iss` claim value of `https://cognito-idp.us-east-1amazonaws.com/u123456`.
- `client_id` : The client app that was issued this access token.

- `username` : The user name of the authenticated user.
- `sub` : The UUID of the authenticated user. This is not the same as `username`.
- `token_use` : The intended purpose of this token. Its value is always `access` in the case of the access token.

Signature

The signature of the access token is calculated based on the header and payload of the access token. You should always verify this signature if you use access tokens in your web APIs.

Using ID Tokens and Access Tokens in your Web APIs

Since both the ID token and the access token are JSON Web Tokens (JWT), you may use any of the available JWT libraries to decode the JWT and verify the signature. For example, if your platform is Java, you could use the Nimbus JOSE and JWT library. The following procedure describes the high level steps you must implement to process the ID token and the access token on the server side.

To verify a signature for ID and access tokens

1. Download and store the JSON Web Token (JWT) Set for your user pool. You can locate them at `https://cognito-idp.{region}.amazonaws.com/{userPoolId}/.well-known/jwks.json`.

   Each JWT should be stored against its `kid`.

   Note
   This is a one time step before your web APIs can process the tokens. Now you can perform the following steps each time the ID token or the access token are used against your web APIs.
2. Decode the token string into JWT format.
3. Check the `iss` claim. It should match your user pool. For example, a user pool created in the `us-east-1` region will have an `iss` value of `https://cognito-idp.us-east-1.amazonaws.com/{userPoolId}`.
4. Check the `token_use` claim.

   If you are only accepting the access token in your web APIs, its value must be `access`.

   If you are only using the ID token, its value must be `id`.

   If you are using both tokens, the value is either `id` or `access`.
5. Get the `kid` from the JWT token header and retrieve the corresponding JSON Web Key that was stored in step 1.
6. Verify the signature of the decoded JWT token.
7. Check the `exp` claim and make sure the token is not expired.

   You can now trust the claims inside the token and use it as it fits your requirements.

Revoking All Tokens for a User

Users can sign out from all devices where they are currently signed in when you revoke all of the user's tokens by using the `GlobalSignOut` and `AdminUserGlobalSignOut` APIs. After the user has been signed out:

- The user's refresh token cannot be used to get new tokens for the user.
- The user's access token cannot be used against the user pools service.

- The user must reauthenticate to get new tokens.

An app can use the `GlobalSignOut` API to allow individual users to sign themselves out from all devices. Typically an app would present this option as a choice, such as **Sign out from all devices**. The app must call this method with the user's valid, nonexpired, nonrevoked access token. This method cannot be used to allow a user to sign out another user.

An administrator app can use the `AdminUserGlobalSignOut` API to allow administrators to sign out a user from all devices. The administrator app must call this method with AWS developer credentials and pass the user pool ID and the user's username as parameters. The `AdminUserGlobalSignOut` API can sign out any user in the user pool.

Amazon Cognito Federated Identities

Amazon Cognito Federated Identities enable you to create unique identities for your users and federate them with identity providers. With an identity, you can obtain temporary, limited-privilege AWS credentials to synchronize data with Amazon Cognito Sync, or directly access other AWS services. Amazon Cognito Identity supports the following identity providers:

- Public providers: Login with Amazon (p. 160), Facebook (p. 155), Google (p. 163), Twitter/Digits (p. 169)
- Amazon Cognito User Pools (p. 9)
- Open ID Connect Providers (p. 173)
- SAML Identity Provider (p. 175)
- Developer Authenticated Identities (p. 177)

For information about Amazon Cognito Identity Region availability, see AWS Service Region Availability.

For more information about Amazon Cognito Identity, see the following topics.

- Identity Pools (p. 135)
- Getting Credentials (p. 148)
- Accessing AWS Services (p. 153)
- External Identity Providers (p. 155)
- Developer Authenticated Identities (p. 177)
- Switching Identities (p. 187)

Topics

- Getting Started with Amazon Cognito Federated Identities (p. 134)
- Identity Pools (p. 135)
- Federated Identities Concepts (p. 136)
- Role-Based Access Control (p. 145)
- Getting Credentials (p. 148)
- Accessing AWS Services (p. 153)
- External Identity Providers (p. 155)
- Developer Authenticated Identities (p. 177)

- Switching Identities (p. 187)

Getting Started with Amazon Cognito Federated Identities

Amazon Cognito Federated Identities enable you to create unique identities and assign permissions for users. Your identity pool can include:

- Users in an Amazon Cognito user pool
- Users who authenticate with federated identity providers such as Facebook, Google, or a SAML-based identity provider
- Users authenticated via your own existing authentication process

With an identity pool, you can obtain temporary AWS credentials with permissions you define to directly access other AWS services or to access resources through Amazon API Gateway.

Topics

- Sign Up for an AWS Account (p. 134)
- Create an Identity Pool in Amazon Cognito (p. 134)
- Install the Mobile or JavaScript SDK (p. 135)
- Integrate the Identity Providers (p. 135)
- Get Credentials (p. 135)

Sign Up for an AWS Account

To use Amazon Cognito identity pools, you need an AWS account. If you don't already have one, use the following procedure to sign up:

To sign up for an AWS account

1. Open https://aws.amazon.com/, and then choose **Create an AWS Account**.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

Create an Identity Pool in Amazon Cognito

You can quickly create an identity pool through the Amazon Cognito console, or you can use the AWS Command Line Interface (CLI) or the Amazon Cognito APIs.

To create a new identity pool in the console

1. Sign in to the Amazon Cognito console, choose **Manage Federated Identities**, and then choose **Create new identity pool**.
2. Type a name for your identity pool, select **Enable access to unauthenticated identities**.
3. If desired, configure an authentication provider in the **Authentication providers** section. For more information, see Integrate the Identity Providers (p. 135) below.
4. Choose **Create Pool**.

5. Choose **Allow** to create the two default roles associated with your identity pool–one for unauthenticated users and one for authenticated users. These default roles provide your identity pool access to Amazon Cognito Sync. You can modify the roles associated with your identity pool in the IAM console. For additional instructions on working with the Amazon Cognito console, see Using the Amazon Cognito Console (p. 230).

Install the Mobile or JavaScript SDK

To use Amazon Cognito identity pools, you must install and configure the AWS Mobile or JavaScript SDK. For more information, see the following topics:

- Set Up the AWS Mobile SDK for Android
- Set Up the AWS Mobile SDK for iOS
- Set Up the AWS SDK for JavaScript
- Set Up the AWS Mobile SDK for Unity
- Set Up the AWS Mobile SDK for .NET and Xamarin

Integrate the Identity Providers

Amazon Cognito Federated Identities support user authentication through Amazon Cognito user pools, federated identity providers—including Amazon, Facebook, Google, and SAML identity providers—as well as unauthenticated identities. This feature also supports Developer Authenticated Identities (p. 177), which lets you register and authenticate users via your own back-end authentication process.

To learn more about using an Amazon Cognito user pool to create your own user directory, see Amazon Cognito User Pools (p. 9) and Integrating User Pools with Federated Identities (p. 126).

To learn more about using external identity providers, see External Identity Providers (p. 155).

To learn more about integrating your own back-end authentication process, see Developer Authenticated Identities (p. 177).

Get Credentials

Amazon Cognito identity pools provide temporary AWS credentials for users who are guests (unauthenticated) and for users who have authenticated and received a token. With those AWS credentials your app can securely access a back end in AWS or outside AWS through Amazon API Gateway. See Getting Credentials (p. 148).

Identity Pools

To use Amazon Cognito Federated Identities in your app, you'll need to create an identity pool. An identity pool is a store of user identity data specific to your account. Using Amazon Cognito Sync (p. 190), you can retrieve the data across client platforms, devices, and operating systems, so that if a user starts using your app on a phone and later switches to a tablet, the persisted app information is still available for that user.

To create a new identity pool for your application:

1. Log in to the Amazon Cognito console, choose **Manage Federated Identities**, and choose **Create new identity pool**.

2. Enter a name for your identity pool, select the checkbox to enable access to unauthenticated identities or configure an identity provider, and then choose **Create Pool.**
3. Choose **Allow** to create the two default roles associated with your identity pool—one for unauthenticated users and one for authenticated users. These default roles provide your identity pool access to Amazon Cognito Sync. You can modify the roles associated with your identity pool in the IAM console.

For additional instructions on working with the Amazon Cognito console, see Using the Amazon Cognito Console (p. 230).

Authenticated and Unauthenticated Identities

Amazon Cognito identity pools support both authenticated and unauthenticated identities. Authenticated identities belong to users who are authenticated by any supported identity provider. Unauthenticated identities typically belong to guest users.

- To configure authenticated identities with a public login provider, see External Identity Providers (p. 155).
- To configure your own backend authentication process, see Developer Authenticated Identities (p. 177).

User IAM Roles

An IAM role defines the permissions for your users to access AWS resources, like Amazon Cognito Sync (p. 190). Users of your application will assume the roles you create. You can specify different roles for authenticated and unauthenticated users. To learn more about IAM roles, see IAM Roles (p. 141).

Federated Identities Concepts

Amazon Cognito Identity enables you to create unique identities for your users and authenticate them with identity providers. With an identity, you can obtain temporary, limited-privilege AWS credentials to synchronize data with Amazon Cognito Sync, or directly access other AWS services. Amazon Cognito Identity supports public identity providers—Amazon, Facebook, and Google—as well as unauthenticated identities. It also supports developer authenticated identities, which let you register and authenticate users via your own back-end authentication process.

For information about Amazon Cognito Identity Region availability, see AWS Service Region Availability. For more information about Amazon Cognito Identity concepts, see the following topics.

Topics

- Authentication Flow (p. 136)
- IAM Roles (p. 141)
- Role Trust and Permissions (p. 144)

Authentication Flow

Amazon Cognito helps you create unique identifiers for your end users that are kept consistent across devices and platforms. Amazon Cognito also delivers temporary, limited-privilege credentials to your application to access AWS resources. This page covers the basics of how authentication in Amazon Cognito works and explains the life cycle of an identity inside your identity pool.

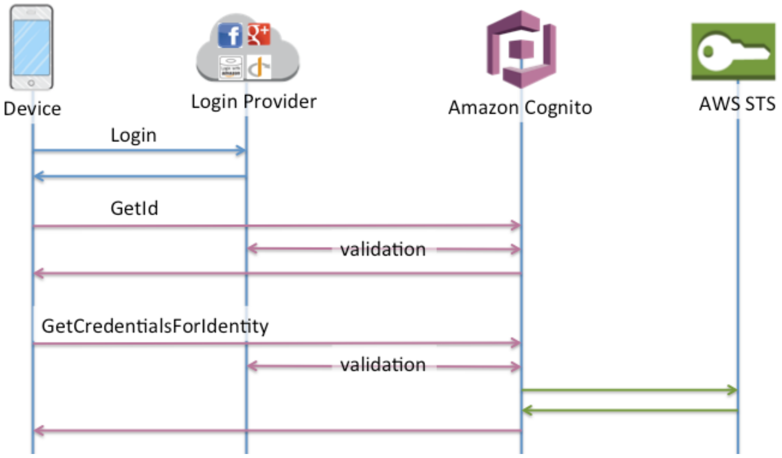
External Provider Authflow

A user authenticating with Amazon Cognito will go through a multi-step process to bootstrap their credentials. Amazon Cognito has two different flows for authentication with public providers: enhanced and basic.

Once you complete one of these flows, you can access other AWS services as defined by your role's access policies. By default, the Amazon Cognito console will create roles with access to the Amazon Cognito Sync store and to Amazon Mobile Analytics. For more information on how to grant additional access see IAM Roles (p. 141).

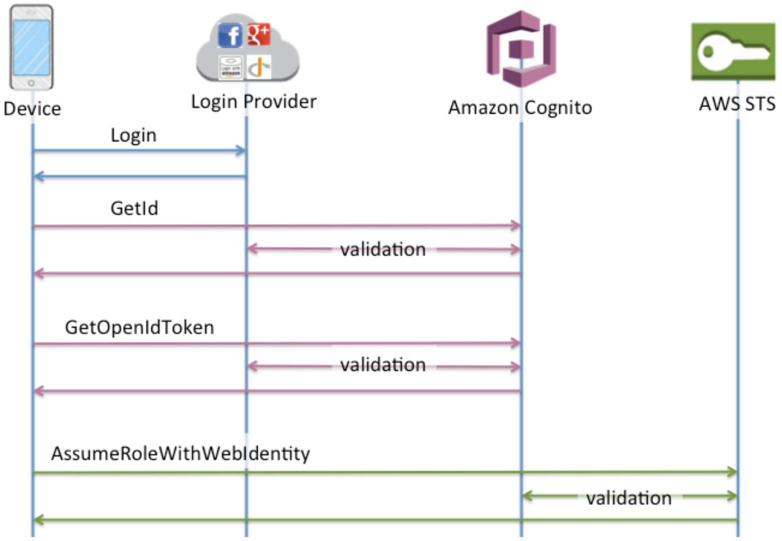
Enhanced (Simplified) Authflow

1. `GetId`
2. `GetCredentialsForIdentity`


Device
Login Provider
Amazon Cognito
AWS STS
Login
GetId
validation
GetCredentialsForIdentity
validation

Basic (Classic) Authflow

1. `GetId`
2. `GetOpenIdToken`
3. `AssumeRoleWithWebIdentity`

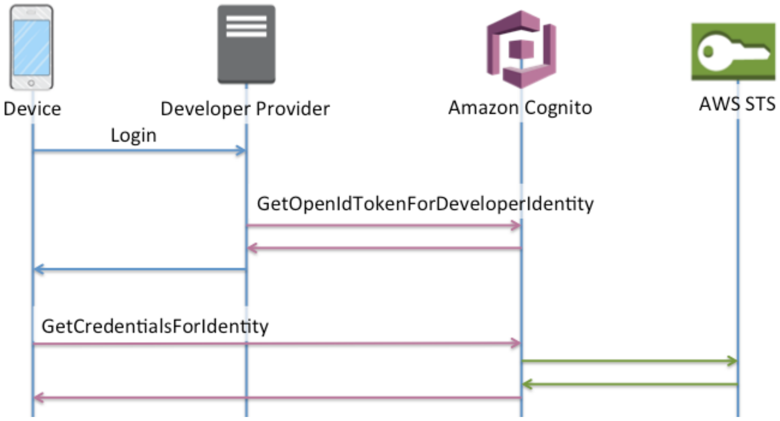

Device
Login Provider
Amazon Cognito
AWS STS
Login
GetId
validation
GetOpenIdToken
validation
AssumeRoleWithWebIdentity
validation

Developer Authenticated Identities Authflow

When using Developer Authenticated Identities (p. 177), the client will use a different authflow that will include code outside of Amazon Cognito to validate the user in your own authentication system. Code outside of Amazon Cognito is indicated as such.

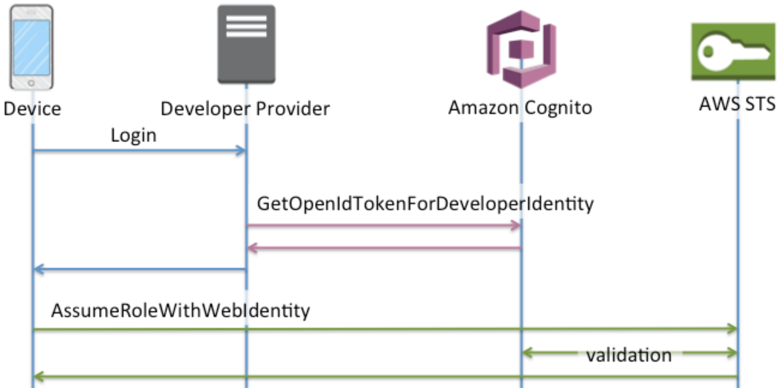
Enhanced Authflow

1. Login via Developer Provider (code outside of Amazon Cognito)
2. Validate the user's login (code outside of Amazon Cognito)
3. GetOpenIdTokenForDeveloperIdentity
4. GetCredentialsForIdentity


Device
Developer Provider
Amazon Cognito
AWS STS
Login
GetOpenIdTokenForDeveloperIdentity
GetCredentialsForIdentity

Basic Authflow

1. Login via Developer Provider (code outside of Amazon Cognito)
2. Validate the user's login (code outside of Amazon Cognito)
3. GetOpenIdTokenForDeveloperIdentity
4. GetCredentialsForIdentity
5. AssumeRoleWithWebIdentity


Device
Developer Provider
Amazon Cognito
AWS STS
Login
GetOpenIdTokenForDeveloperIdentity
AssumeRoleWithWebIdentity
validation

Which Authflow Should I Use?

For most customers, the Enhanced Flow is the correct choice, as it offers many benefits over the Basic Flow:

- One fewer network call to get credentials on the device.
- All calls are made to Amazon Cognito, meaning it is also one less network connection.
- Roles no longer need to be embedded in your application, only an identity pool id and region are necessary to start bootstrapping credentials.

Since February 2015, the Amazon Cognito console displayed example code that used the Enhanced Flow. Additionally, the console will display a notification if your identity pool does not have the role association necessary to use the Enhanced Flow.

The following are the minimum SDK versions where the Enhanced Flow is supported:

SDK (Minimum Version)

- AWS SDK for iOS (2.0.14)
- AWS SDK for Android (2.1.8)
- AWS SDK for JavaScript (2.1.7)
- AWS SDK for Unity (1.0.3)
- AWS SDK for Xamarin (3.0.0.5)

You may still wish to use the Basic Flow if you want to use more than the two default roles configured when you create a new identity pool in the console.

API Summary

GetId

The GetId API call is the first call necessary to establish a new identity in Amazon Cognito.

Unauthenticated Access

Amazon Cognito has the ability to allow unauthenticated guest access in your applications. If this feature is enabled in your identity pool, users can request a new identity ID at any time via the GetId API. The application is expected to cache this identity ID to make subsequent calls to Amazon Cognito. The AWS Mobile SDKs as well as the AWS SDK for JavaScript in the Browser have credentials providers that handle this caching for you.

Authenticated Access

When you've configured your application with support for a public login provider (Facebook, Google +, Login with Amazon), users will also be able to supply tokens (OAuth or OpenID Connect) that identify them in those providers. When used in a call to GetId, Amazon Cognito will either create a new authenticated identity or return the identity already associated with that particular login. Amazon Cognito does this by validating the token with the provider and ensuring that:

- The token is valid and from the configured provider
- The token is not expired
- The token matches the application identifier created with that provider (e.g., Facebook app ID)
- The token matches the user identifier

GetCredentialsForIdentity

The GetCredentialsForIdentity API can be called after you establish an identity ID. This API is functionally equivalent to calling GetOpenIdToken followed by AssumeRoleWithWebIdentity.

In order for Amazon Cognito to call AssumeRoleWithWebIdentity on your behalf, your identity pool must have IAM roles associated with it. You can do this via the Amazon Cognito Console or manually via the SetIdentityPoolRoles operation (see the API reference)

GetOpenIdToken

The GetOpenIdToken API call is called after you establish an identity ID. If you have a cached identity ID, this can be the first call you make during an app session.

Unauthenticated Access

To obtain a token for an unauthenticated identity, you only need the identity ID itself. It is not possible to get an unauthenticated token for authenticated or disabled identities.

Authenticated Access

If you have an authenticated identity, you must pass at least one valid token for a login already associated with that identity. All tokens passed in during the GetOpenIdToken call must pass the same validation mentioned earlier; if any of the tokens fail, the whole call fails. The response from the GetOpenIdToken call also includes the identity ID. This is because the identity ID you pass in may not be the one that is returned.

Linking Logins

If you pass in a token for a login that is not already associated with any identity, the login is considered to be "linked" to the associated identity. You may only link one login per public provider. Attempts to link more than one login with a public provider will result in a ResourceConflictException. If a login is merely linked to an existing identity, the identity ID returned from GetOpenIdToken will be the same as what was passed in.

Merging Identities

If you pass in a token for a login that is not currently linked to the given identity, but is linked to another identity, the two identities are merged. Once merged, one identity becomes the parent/owner of all associated logins and the other is disabled. In this case, the identity ID of the parent/owner is returned. You are expected to update your local cache if this value differs (this is handled for you if you are using the providers in the AWS Mobile SDKs or AWS SDK for JavaScript in the Browser).

GetOpenIdTokenForDeveloperIdentity

The GetOpenIdTokenForDeveloperIdentity API replaces the use of GetId and GetOpenIdToken from the device when using developer authenticated identities. Because this API call is signed by your AWS credentials, Amazon Cognito can trust that the user identifier supplied in the API call is valid. This replaces the token validation Amazon Cognito performs with external providers.

The payload for this API includes a logins map which must contain the key of your developer provider and a value as an identifier for the user in your system. If the user identifier isn't already linked to an existing identity, Amazon Cognito will create a new identity and return the new identity id and an OpenId Connect token for that identity. If the user identifier is already linked, Amazon Cognito will return the pre-existing identity id and an OpenId Connect token.

Linking Logins

As with external providers, supplying additional logins that are not already associated with an identity will implicitly link those logins to that identity. It is important to note that if you link an external provider login to an identity, the user can use the external provider authflow with that provider, but they cannot use your developer provider name in the logins map when calling GetId or GetOpenIdToken.

Merging Identities

With developer authenticated identities, Amazon Cognito supports both implicit merging as well as explicit merging via the MergeDeveloperIdentities API. This explicit merging allows you to mark two identities with user identifiers in your system as a single identity. You simply supply the source and destination user identifiers and Amazon Cognito will merge them. The next time you request an OpenId Connect token for either user identifier, the same identity id will be returned.

AssumeRoleWithWebIdentity

Once you have an OpenID Connect token, you can then trade this for temporary AWS credentials via the AssumeRoleWithWebIdentity API call in AWS Security Token Service (STS). This call is no different than if you were using Facebook, Google+, or Login with Amazon directly, except that you are passing an Amazon Cognito token instead of a token from one of the other public providers.

Because there's no restriction on the number of identities that can be created, it's important to understand the permissions that are being granted to your users. We recommend having two different roles for your application: one for unauthenticated users, and one for authenticated users. The Amazon Cognito console will create these for you by default when you first set up your identity pool. The access policy for these two roles will be exactly the same: it will grant users access to Amazon Cognito Sync as well as to submit events to Amazon Mobile Analytics. You are welcome and encouraged to modify these roles to meet your needs.

Learn more about Role Trust and Permissions (p. 144).

IAM Roles

In the process of creating an identity pool, you'll be prompted to update the IAM roles that your users assume. IAM roles work like this: When a user logs in to your app, Amazon Cognito generates temporary AWS credentials for the user. These temporary credentials are associated with a specific IAM role. The IAM role lets you define a set of permissions to access your AWS resources.

You can specify default IAM roles for authenticated and unauthenticated users. In addition, you can define rules to choose the role for each user based on claims in the user's ID token. For more information, see Role-Based Access Control (p. 145).

By default, the Amazon Cognito Console creates IAM roles that provide access to Amazon Mobile Analytics and to Amazon Cognito Sync. Alternatively, you can choose to use existing IAM roles.

To modify IAM roles, thereby allowing or restricting access to other services, log in to the IAM Console. Then click Roles and select a role. The policies attached to the selected role are listed in the Permissions tab. You can customize an access policy by clicking the corresponding Manage Policy link. To learn more about using and defining policies, see Overview of IAM Policies. For Amazon Cognito to work, the IAM policy must at least enable access to the Amazon Cognito store for each identity, as in the following example:

```
{
  "Version": "2012-10-17",
  "Statement":[{
      "Effect":"Allow",
      "Action":"cognito-sync:*",
      "Resource":["arn:aws:cognito-sync:us-east-1:123456789012:identitypool/${cognito-
identity.amazonaws.com:aud}/identity/${cognito-identity.amazonaws.com:sub}/*"]
      }]
  }
```

The following policy provides access to the entire Amazon Cognito Sync store:

```
{
  "Version": "2012-10-17",
  "Statement":[{
      "Effect":"Allow",
      "Action":"cognito-sync:*",
      "Resource":["arn:aws:cognito-sync:us-east-1:123456789012:identitypool/*"]
      }]
  }
```

Role Trust and Permissions

Amazon Cognito leverages IAM roles to generate temporary credentials for your application's users. Access to permissions is controlled by a role's trust relationships. Learn more about Role Trust and Permissions (p. 144).

Reuse Roles Across Identity Pools

To reuse a role across multiple identity pools, because they share a common permission set, you can include multiple identity pools, like this:

```
"StringEquals": {
    "cognito-identity.amazonaws.com:aud": [
        "us-east-1:12345678-abcd-abcd-abcd-123456790ab",
        "us-east-1:98765432-dcba-dcba-dcba-123456790ab"
    ]
}
```

Limit Access to Specific Identities

To create a policy limited to a specific set of app users, check the value of `cognito-identity.amazonaws.com:sub`:

```
"StringEquals": {
    "cognito-identity.amazonaws.com:aud": "us-east-1:12345678-abcd-abcd-abcd-123456790ab",
    "cognito-identity.amazonaws.com:sub": [
        "us-east-1:12345678-1234-1234-1234-123456790ab",
        "us-east-1:98765432-1234-1234-1243-123456790ab"
    ]
}
```

Limit Access to Specific Providers

To create a policy limited to users who have logged in with a specific provider (perhaps your own login provider), check the value of `cognito-identity.amazonaws.com:amr`:

```
"ForAnyValue:StringLike": {
    "cognito-identity.amazonaws.com:amr": "login.myprovider.myapp"
}
```

For example, an app that trusts only Facebook would have the following amr clause:

```
"ForAnyValue:StringLike": {
    "cognito-identity.amazonaws.com:amr": "graph.facebook.com"
}
```

Access Policies

The permissions attached to a role are effective across all users that assume that role. If you want to partition your users' access, you can do so via policy variables. Be careful when including your users' identity IDs in your access policies, particularly for unauthenticated identities as these may change if the user chooses to login.

For additional security protection, Amazon Cognito applies a scope-down policy to credentials vended by `GetCredentialForIdentity` to prevent access to services other than these to your unauthenticated users:

- CloudWatch
- Amazon Cognito Identity
- Amazon Cognito Sync
- DynamoDB
- Amazon Kinesis Firehose

- GameLift
- AWS IoT
- Amazon Kinesis Streams
- AWS KMS
- AWS Lambda
- Amazon Lex
- Amazon Machine Learning
- Amazon Mobile Analytics
- Amazon Polly
- Amazon Rekognition
- Amazon S3
- Amazon SimpleDB
- Amazon SES
- Amazon SNS
- Amazon SQS

If you need access to something other than these services for your unauthenticated users, you must use the basic authentication flow. If you are getting `NotAuthorizedException` and you have enabled access to the service in your unauthenticated role policy, this is likely the reason.

S3 Prefix

You can give a user a specific prefix "folder" in an S3 bucket by mapping the prefix to the `${cognito-identity.amazonaws.com:sub}` variable:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket"],
      "Condition": {"StringLike": {"s3:prefix": ["${cognito-identity.amazonaws.com:sub}/
*"]}}
    },
    {
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket/${cognito-identity.amazonaws.com:sub}/*"]
    }
  ]
}
```

Fine-Grained Access to Amazon DynamoDB

You can use Amazon Cognito variables to provide fine-grained access control to Amazon DynamoDB resources. Just grant access to items in DynamoDB by identity ID:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/MyTable"
      ],
      "Condition": {
        "ForAllValues:StringEquals": {
          "dynamodb:LeadingKeys":  ["${cognito-identity.amazonaws.com:sub}"]
        }
      }
    }
  ]
}
```

Role Trust and Permissions

The way these roles differ is in their trust relationships. Let's take a look at an example trust policy for an unauthenticated role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Federated": "cognito-identity.amazonaws.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "cognito-identity.amazonaws.com:aud": "us-east-1:12345678-dead-beef-
cafe-123456790ab"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "unauthenticated"
        }
      }
    }
  ]
}
```

This policy defines that we want to allow federated users from `cognito-identity.amazonaws.com` (the issuer of the OpenID Connect token) to assume this role. Additionally, we make the restriction that the aud of the token, in our case the identity pool ID, matches our identity pool. Finally, we specify that the amr of the token contains the value unauthenticated.

When Amazon Cognito creates a token, it will set the `amr` of the token to be either "unauthenticated" or "authenticated" and in the authenticated case will include any providers used during authentication. This means you can create a role that trusts only users that logged in via Facebook, simply by changing the amr clause to look like the following:

```
"ForAnyValue:StringLike": {
  "cognito-identity.amazonaws.com:amr": "graph.facebook.com"
}
```

Be careful when changing your trust relationships on your roles, or when trying to use roles across identity pools. If your role is not configured to correctly trust your identity pool, you will see an exception from STS like the following:

```
AccessDenied -- Not authorized to perform sts:AssumeRoleWithWebIdentity
```

If you see this, double check that you are using an appropriate role for your identity pool and authentication type.

Role-Based Access Control

Amazon Cognito Federated Identities assigns your authenticated users a set of temporary, limited privilege credentials to access your AWS resources. The permissions for each user are controlled through IAM roles that you create. You can define rules to choose the role for each user based on claims in the user's ID token. You can define a default role for authenticated users. You can also define a separate IAM role with limited permissions for guest users who are not authenticated.

Creating Roles for Role Mapping

It is important to add the appropriate trust policy for each role so that it can only be assumed by Amazon Cognito Identity for authenticated users in your identity pool. Here is an example of such a trust policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Federated": "cognito-identity.amazonaws.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "cognito-identity.amazonaws.com:aud": "us-east-1:12345678-dead-beef-
cafe-123456790ab"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "authenticated"
        }
      }
    }
  ]
}
```

This policy allows federated users from `cognito-identity.amazonaws.com` (the issuer of the OpenID Connect token) to assume this role. Additionally, the policy restricts the `aud` of the token, in this case the identity pool ID, to match the identity pool. Finally, the policy specifies that the `amr` of the token contains the value `authenticated`.

Granting Pass Role Permission

To allow an IAM user to set roles with permissions in excess of the user's existing permissions on an identity pool, you grant that user `iam:PassRole` permission to pass the role to the `set-identity-pool-`

`roles` API. For example, if the user cannot write to Amazon S3, but the IAM role that the user sets on the identity pool grants write permission to Amazon S3, the user can only set this role if `iam:PassRole` permission is granted for the role. The following example policy shows how to allow `iam:PassRole` permission.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/myS3WriteAccessRole"
            ]
        }
    ]
}
```

In this policy example, the `iam:PassRole` permission is granted for the `myS3WriteAccessRole` role. The role is specified using the role's ARN. You must also attach this policy to your IAM user or role to which your user belongs. For more information, see Working with Managed Policies.

> **Note**
> Lambda functions use resource-based policy, where the policy is attached directly to the Lambda function itself. When creating a rule that invokes a Lambda function, you do not pass a role, so the user creating the rule does not need the `iam:PassRole` permission. For more information about Lambda function authorization, see Manage Permissions: Using a Lambda Function Policy.

Using Tokens to Assign Roles to Users

For users who log in via Amazon Cognito Your User Pools, roles can be passed in the ID token that was assigned by the user pool. The roles appear in the following claims in the ID token:

- The `cognito:preferred_role` claim is the role ARN.
- The `cognito:roles` claim is a comma-separated string containing a set of allowed role ARNs.

The claims are set as follows:

- The `cognito:preferred_role` claim is set to the role from the group with the best (lowest) `Precedence` value. If there is only one allowed role, `cognito:preferred_role` is set to that role. If there are multiple roles and no single role has the best precedence, this claim is not set.
- The `cognito:roles` claim is set if there is at least one role.

When using tokens to assign roles, if there are multiple roles that can be assigned to the user, Amazon Cognito Federated Identities chooses the role as follows:

- Use the GetCredentialsForIdentity `CustomRoleArn` parameter if it is set and it matches a role in the `cognito:roles` claim. If this parameter doesn't match a role in `cognito:roles`, deny access.
- If the `cognito:preferred_role` claim is set, use it.
- If the `cognito:preferred_role` claim is not set, the `cognito:roles` claim is set, and `CustomRoleArn` is not specified in the call to `GetCredentialsForIdentity`, then the **Role resolution** setting in the console or the `AmbiguousRoleResolution` field (in the `RoleMappings` parameter of the SetIdentityPoolRoles API) is used to determine the role to be assigned.

Using Rule-Based Mapping to Assign Roles to Users

Rules allow you to map claims from an identity provider token to IAM roles.

Each rule specifies a token claim (such as a user attribute in the ID token from an Amazon Cognito user pool), match type, a value, and an IAM role. The match type can be `Equals`, `NotEqual`, `StartsWith`, or `Contains`. If a user has a matching value for the claim, the user can assume that role when the user gets credentials. For example, you can create a rule that assigns a specific IAM role for users with a `custom:dept` custom attribute value of `Sales`.

> **Note**
> In the rule settings, custom attributes require the `custom:` prefix to distinguish them from standard attributes.

Rules are evaluated in order, and the IAM role for the first matching rule is used, unless `CustomRoleArn` is specified to override the order. For more information about user attributes in Amazon Cognito user pools, see Specifying User Pool Attribute Settings (p. 14).

You can set multiple rules for an authentication provider in the identity pool console. Rules are applied in order. You can drag the rules to change their order. The first matching rule takes precedence. If the match type is `NotEqual` and the claim doesn't exist, the rule is not evaluated. If no rules match, the **Role resolution** setting is applied to either **Use default Authenticated role** or **DENY**.

In the API and CLI, you can specify the role to be assigned when no rules match in the `AmbiguousRoleResolution` field of the RoleMapping type, which is specified in the `RoleMappings` parameter of the SetIdentityPoolRoles API.

For each user pool or other authentication provider configured for an identity pool, you can create up to 25 rules. If you need more than 25 rules for a provider, please open a Service Limit Increase support case.

Token Claims to Use in Rule-Based Mapping

Amazon Cognito

An Amazon Cognito ID token is represented as a JSON Web Key Token (JWT). The token contains claims about the identity of the authenticated user, such as `name`, `family_name`, and `phone_number`. For more information about standard claims, see the OpenID Connect specification. Apart from standard claims, the following are the additional claims specific to Amazon Cognito:

- `cognito:groups`
- `cognito:roles`
- `cognito:preferred_role`

Amazon

The following claims, along with possible values for those claims, can be used with Login with Amazon:

- `iss`: www.amazon.com
- `aud`: App Id
- `sub`: `sub` from the Login with Amazon token

Facebook

The following claims, along with possible values for those claims, can be used with Facebook:

- `iss`: graph.facebook.com

- `aud`: App Id
- `sub`: `sub` from the Facebook token

Twitter/Digits

The following claims, along with possible values for those claims, can be used with Twitter/Digits:

- `iss`: api.twitter.com or www.digits.com
- `aud`: App Id
- `sub`: Twitter/Digits user ID

Google

A Google token contains standard claims from the OpenID Connect specification. All of the claims in the OpenID token are available for rule-based mapping. See Google's OpenID Connect site to learn about the claims available from the Google token.

OpenID

All of the claims in the Open Id Token are available for rule-based mapping. For more information about standard claims, see the OpenID Connect specification. Refer to your OpenID provider documentation to learn about any additional claims that are available.

SAML

Claims are parsed from the received SAML assertion. All the claims that are available in the SAML assertion can be used in rule-based mapping.

Best Practices for Role-Based Access Control

Important
If the claim that you are mapping to a role can be modified by the end user, any end user can assume your role and set the policy accordingly. Only map claims that cannot be directly set by the end user to roles with elevated permissions. In an Amazon Cognito user pool, you can set per-app read and write permissions for each user attribute.

Important
If you set roles for groups in an Amazon Cognito user pool, those roles are passed through the user's ID token. To use these roles, you must also set **Choose role from token** for the authenticated role selection for the identity pool.
You can use the **Role resolution** setting in the console and the `RoleMappings` parameter of the SetIdentityPoolRoles API to specify what the default behavior is when the correct role cannot be determined from the token.

Getting Credentials

This section describes how to get credentials and how to retrieve an Amazon Cognito identity.

Android

You can use Amazon Cognito to deliver temporary, limited-privilege credentials to your application, so that your users can access AWS resources. Amazon Cognito supports both authenticated and unauthenticated identities. To provide AWS credentials to your app, follow the steps below.

1. In the Amazon Cognito console, create an identity pool and copy the starter code snippets.

2. If you haven't already done so, add the AWS Mobile SDK for Android to your project. For instructions, see Set Up the Mobile SDK for Android.
3. Include the following import statements:

```
import com.amazonaws.auth.CognitoCachingCredentialsProvider;
import com.amazonaws.regions.Regions;
```

4. Initialize the Amazon Cognito credentials provider using the code snippet generated by the Amazon Cognito console. The value for `IDENTITY_POOL_ID` will be specific to your account:

```
CognitoCachingCredentialsProvider credentialsProvider = new
 CognitoCachingCredentialsProvider(
  getApplicationContext(), // Context
  "IDENTITY_POOL_ID", // Identity Pool ID
  Regions.US_EAST_1 // Region
);
```

5. Pass the initialized Amazon Cognito credentials provider to the constructor of the AWS client to be used. The code required depends on the service to be initialized. The client will use this provider to get credentials with which it will access AWS resources.

   Note
   If you created your identity pool before February 2015, you will need to reassociate your roles with your identity pool in order to use this constructor without the roles as parameters. To do so, open the Amazon Cognito console, select your identity pool, choose **Edit Identity Pool**, specify your authenticated and unauthenticated roles, and save the changes.

Retrieving an Amazon Cognito Identity

If you're allowing unauthenticated users, you can retrieve a unique Amazon Cognito identifier (identity ID) for your end user immediately. If you're authenticating users, you can retrieve the identity ID after you've set the login tokens in the credentials provider:

```
String identityId = credentialsProvider.getIdentityId();
Log.d("LogTag", "my ID is " + identityId);
```

Note
Do not call `getIdentityId()`, `refresh()`, or `getCredentials()` in the main thread of your application. As of Android 3.0 (API Level 11), your app will automatically fail and throw a NetworkOnMainThreadException if you perform network I/O on the main application thread. You will need to move your code to a background thread using `AsyncTask`. For more information, consult the Android documentation. You can also call `getCachedIdentityId()` to retrieve an ID, but only if one is already cached locally. Otherwise, the method will return null.

iOS - Objective-C

You can use Amazon Cognito to deliver temporary, limited-privilege credentials to your application, so that your users can access AWS resources. Amazon Cognito supports both authenticated and unauthenticated identities. To provide AWS credentials to your app, follow the steps below.

1. In the Amazon Cognito console, create an identity pool and copy the starter code snippets.
2. If you haven't already done so, add the AWS Mobile SDK for iOS to your project. For instructions, see Set Up the Mobile SDK for iOS.
3. In your source code, include the AWSCore header:

```
#import <AWSCore/AWSCore.h>
```

4. Initialize the Amazon Cognito credentials provider using the code snippet generated by the Amazon Cognito console. The value for `IDENTITY_POOL_ID` will be specific to your account:

```
AWSCognitoCredentialsProvider *credentialsProvider = [[AWSCognitoCredentialsProvider
 alloc]
initWithRegionType:AWSRegionUSEast1 identityPoolId:@"IDENTITY_POOL_ID"];
AWSServiceConfiguration *configuration = [[AWSServiceConfiguration alloc]
initWithRegion:AWSRegionUSEast1 credentialsProvider:credentialsProvider];
AWSServiceManager.defaultServiceManager.defaultServiceConfiguration = configuration;
```

> **Note**
> If you created your identity pool before February 2015, you will need to reassociate your roles with your identity pool in order to use this constructor without the roles as parameters. To do so, open the Amazon Cognito console, select your identity pool, choose **Edit Identity Pool**, specify your authenticated and unauthenticated roles, and save the changes.

Retrieving an Amazon Cognito Identity

You can retrieve a unique Amazon Cognito identifier (identity ID) for your end user immediately if you're allowing unauthenticated users or after you've set the login tokens in the credentials provider if you're authenticating users:

```
// Retrieve your Amazon Cognito ID
[[credentialsProvider getIdentityId] continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"Error: %@", task.error);
    }
    else {
        // the task result will contain the identity id
        NSString *cognitoId = task.result;
    }
    return nil;
}];
```

> **Note**
> `getIdentityId` is an asynchronous call. If an identity ID is already set on your provider, you can call `credentialsProvider.identityId` to retrieve that identity, which is cached locally. However, if an identity ID is not set on your provider, calling `credentialsProvider.identityId` will return `nil`. For more information, consult the Mobile SDK for iOS API Reference.

iOS - Swift

You can use Amazon Cognito to deliver temporary, limited-privilege credentials to your application, so that your users can access AWS resources. Amazon Cognito supports both authenticated and unauthenticated identities. To provide AWS credentials to your app, follow the steps below.

1. In the Amazon Cognito console, create an identity pool and copy the starter code snippets.
2. If you haven't already done so, add the Mobile SDK for iOS to your project. For instructions, see Set Up the SDK for iOS.
3. In your source code, include the `AWSCore` header:

```
import AWSCore
```

4. Initialize the Amazon Cognito credentials provider using the code snippet generated by the Amazon Cognito console. The value for `IDENTITY_POOL_ID` will be specific to your account:

```
let credentialsProvider = AWSCognitoCredentialsProvider(regionType: .USEast1,
 identityPoolId: "IDENTITY_POOL_ID")
let configuration = AWSServiceConfiguration(region: .USEast1, credentialsProvider:
 credentialsProvider)
AWSServiceManager.default().defaultServiceConfiguration = configuration
```

Note

If you created your identity pool before February 2015, you will need to reassociate your roles with your identity pool in order to use this constructor without the roles as parameters. To do so, open the Amazon Cognito console, select your identity pool, choose **Edit Identity Pool**, specify your authenticated and unauthenticated roles, and save the changes.

Retrieving an Amazon Cognito Identity

You can retrieve a unique Amazon Cognito identifier (identity ID) for your end user immediately if you're allowing unauthenticated users or after you've set the login tokens in the credentials provider if you're authenticating users:

```
// Retrieve your Amazon Cognito ID
credentialsProvider.getIdentityId().continueWith(block: { (task) -> AnyObject? in
    if (task.error != nil) {
        print("Error: " + task.error!.localizedDescription)
    }
    else {
        // the task result will contain the identity id
        let cognitoId = task.result!
        print("Cognito id: \(cognitoId)")
    }
    return task;
})
```

Note

`getIdentityId` is an asynchronous call. If an identity ID is already set on your provider, you can call `credentialsProvider.identityId` to retrieve that identity, which is cached locally. However, if an identity ID is not set on your provider, calling `credentialsProvider.identityId` will return `nil`. For more information, consult the Mobile SDK for iOS API Reference.

JavaScript

You can use Amazon Cognito to deliver temporary, limited-privilege credentials to your application, so that your users can access AWS resources. Amazon Cognito supports both authenticated and unauthenticated identities. To provide AWS credentials to your app, follow the steps below.

```
// Set the region where your identity pool exists (us-east-1, eu-west-1)
AWS.config.region = 'us-east-1';

// Configure the credentials provider to use your identity pool
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
    IdentityPoolId: 'IDENTITY_POOL_ID',
});

// Make the call to obtain credentials
AWS.config.credentials.get(function(){

    // Credentials will be available when this function is called.
    var accessKeyId = AWS.config.credentials.accessKeyId;
    var secretAccessKey = AWS.config.credentials.secretAccessKey;
    var sessionToken = AWS.config.credentials.sessionToken;

});
```

Note
If you created your identity pool before February 2015, you will need to reassociate your roles with your identity pool in order to use this constructor without the roles as parameters. To do so, open the Amazon Cognito console, select your identity pool, choose **Edit Identity Pool**, specify your authenticated and unauthenticated roles, and save the changes.

Retrieving an Amazon Cognito Identity

You can retrieve a unique Amazon Cognito identifier (identity ID) for your end user immediately if you're allowing unauthenticated users or after you've set the login tokens in the credentials provider if you're authenticating users:

```
var identityId = AWS.config.credentials.identityId;
```

Unity

You can use Amazon Cognito to deliver temporary, limited-privilege credentials to your application, so that your users can access AWS resources. Amazon Cognito supports both authenticated and unauthenticated identities. To provide AWS credentials to your app, follow the steps below.

1. In the Amazon Cognito console, create an identity pool and copy the starter code snippets.
2. If you haven't already done so, download and import the AWS Mobile SDK for Unity package into your project. You can do so from the menu Assets > Import Package > Custom Package.
3. Paste the starter code snippet from the Console into the script you want to call Amazon Cognito from. The value for `IDENTITY_POOL_ID` will be specific to your account:

```
CognitoAWSCredentials credentials = new CognitoAWSCredentials (
      "IDENTITY_POOL_ID",    // Cognito Identity Pool ID
      RegionEndpoint.USEast1 // Region
    );
```

4. Pass the initialized Amazon Cognito credentials to the constructor of the AWS client to be used. The code required depends on the service to be initialized. The client will use this provider to get credentials with which it will access AWS resources.

   Note
   If you created your identity pool before February 2015, you will need to reassociate your roles with your identity pool in order to use this constructor without the roles as parameters. To do so, open the Amazon Cognito console, select your identity pool, choose **Edit Identity Pool**, specify your authenticated and unauthenticated roles, and save the changes.

Retrieving an Amazon Cognito Identity

You can retrieve a unique Amazon Cognito identifier (identity ID) for your end user immediately if you're allowing unauthenticated users or after you've set the login tokens in the credentials provider if you're authenticating users:

```
credentials.GetIdentityIdAsync(delegate(AmazonCognitoIdentityResult<string> result) {
    if (result.Exception != null) {
        //Exception!
    }
    string identityId = result.Response;
});
```

Xamarin

You can use Amazon Cognito to deliver temporary, limited-privilege credentials to your application, so that your users can access AWS resources. Amazon Cognito supports both authenticated and unauthenticated identities. To provide AWS credentials to your app, follow the steps below.

1. In the Amazon Cognito console, create an identity pool and copy the starter code snippets.
2. If you haven't already done so, add the AWS Mobile SDK for Xamarin to your project. For instructions, see Set Up the SDK for Xamarin.
3. Include the following using statements:

```
using Amazon.CognitoIdentity;
```

4. Paste the starter code snippet from the Console into the script you want to call Amazon Cognito from. The value for `IDENTITY_POOL_ID` will be specific to your account:

```
CognitoAWSCredentials credentials = new CognitoAWSCredentials (
  "IDENTITY_POOL_ID",    // Cognito Identity Pool ID
  RegionEndpoint.USEast1 // Region
);
```

5. Pass the initialized Amazon Cognito credentials to the constructor of the AWS client to be used. The code required depends on the service to be initialized. The client will use this provider to get credentials with which it will access AWS resources.

Note

Note: If you created your identity pool before February 2015, you will need to reassociate your roles with your identity pool in order to use this constructor without the roles as parameters. To do so, open the Amazon Cognito console, select your identity pool, choose **Edit Identity Pool**, specify your authenticated and unauthenticated roles, and save the changes.

Retrieving an Amazon Cognito Identity

You can retrieve a unique Amazon Cognito identifier (identity ID) for your end user immediately if you're allowing unauthenticated users or after you've set the login tokens in the credentials provider if you're authenticating users:

```
var identityId = await credentials.GetIdentityIdAsync();
```

Accessing AWS Services

Once the Amazon Cognito credentials provider is initialized and refreshed, you can pass it directly to the initializer for an AWS client. For example, the following snippet initializes an Amazon DynamoDB client:

Android

```
// Create a service client with the provider
AmazonDynamoDB client = new AmazonDynamoDBClient(credentialsProvider);
```

The credentials provider communicates with Amazon Cognito, retrieving both the unique identifier for authenticated and unauthenticated users as well as temporary, limited privilege AWS credentials for the AWS Mobile SDK. The retrieved credentials are valid for one hour, and the provider refreshes them when they expire.

iOS - Objective-C

```
// create a configuration that uses the provider
AWSServiceConfiguration *configuration = [AWSServiceConfiguration
 configurationWithRegion:AWSRegionUSEast1 provider:credentialsProvider];

// get a client with the default service configuration
AWSDynamoDB *dynamoDB = [AWSDynamoDB defaultDynamoDB];
```

The credentials provider communicates with Amazon Cognito, retrieving both the unique identifier for authenticated and unauthenticated users as well as temporary, limited privilege AWS credentials for the AWS Mobile SDK. The retrieved credentials are valid for one hour, and the provider refreshes them when they expire.

iOS - Swift

```
// get a client with the default service configuration
let dynamoDB = AWSDynamoDB.default()

// get a client with a custom configuration
AWSDynamoDB.register(with: configuration!, forKey: "USWest2DynamoDB");
let dynamoDBCustom = AWSDynamoDB(forKey: "USWest2DynamoDB")
```

The credentials provider communicates with Amazon Cognito, retrieving both the unique identifier for authenticated and unauthenticated users as well as temporary, limited privilege AWS credentials for the AWS Mobile SDK. The retrieved credentials are valid for one hour, and the provider refreshes them when they expire.

JavaScript

```
// Create a service client with the provider
var dynamodb = new AWS.DynamoDB({region: 'us-west-2'});
```

The credentials provider communicates with Amazon Cognito, retrieving both the unique identifier for authenticated and unauthenticated users as well as temporary, limited privilege AWS credentials for the AWS Mobile SDK. The retrieved credentials are valid for one hour, and the provider refreshes them when they expire.

Unity

```
// create a service client that uses credentials provided by Cognito
AmazonDynamoDBClient client = new AmazonDynamoDBClient(credentials, REGION);
```

The credentials provider communicates with Amazon Cognito, retrieving both the unique identifier for authenticated and unauthenticated users as well as temporary, limited privilege AWS credentials for the AWS Mobile SDK. The retrieved credentials are valid for one hour, and the provider refreshes them when they expire.

Xamarin

```
// create a service client that uses credentials provided by Cognito
```

```
var client = new AmazonDynamoDBClient(credentials, REGION)
```

The credentials provider communicates with Amazon Cognito, retrieving both the unique identifier for authenticated and unauthenticated users as well as temporary, limited privilege AWS credentials for the AWS Mobile SDK. The retrieved credentials are valid for one hour, and the provider refreshes them when they expire.

External Identity Providers

Using the `logins` property, you can set credentials received from an identity provider. Moreover, you can associate an Amazon Cognito identity with multiple identity providers. For example, you could set both the Facebook and Google tokens in the `logins` property, so that the unique Amazon Cognito identity would be associated with both identity provider logins. No matter which account the end user uses for authentication, Amazon Cognito returns the same user identifier.

The instructions below guide you through authentication with the identity providers supported by Amazon Cognito.

Topics

- Facebook (p. 155)
- Login with Amazon (p. 160)
- Google (p. 163)
- Twitter/Digits (p. 169)
- Open ID Connect Providers (p. 173)
- SAML Identity Provider (p. 175)

Facebook

Amazon Cognito integrates with Facebook to provide federated authentication for your mobile application users. This section explains how to register and set up your application with Facebook as an identity provider.

Set Up Facebook

You need to register your application with Facebook before you can start authenticating Facebook users and interacting with Facebook APIs.

The Facebook Developers portal takes you through the process of setting up your application. If you haven't gone through that process yet, you'll need to do so before you can integrate Facebook in your Amazon Cognito Identity Pool:

To set up Facebook

1. At the Facebook Developers portal, log in with your Facebook credentials.
2. From the **Apps** menu, select **Add a New App**.
3. Select a platform and complete the quick start process.

Android

The Facebook Getting Started Guide provides additional information on integrating with Facebook Login.

iOS - Objective-C

The Facebook Getting Started Guide provides additional information about integrating with Facebook Login.

iOS - Swift

The Facebook Getting Started Guide provides additional information about integrating with Facebook Login.

JavaScript

The Facebook Getting Started Guide provides additional information about integrating with Facebook Login.

Unity

The Facebook Getting Started Guide provides additional information about integrating with Facebook Login.

Xamarin

To provide Facebook authentication, first follow the appropriate flow below to include and set up the Facebook SDK in your application. Amazon Cognito uses the Facebook access token to generate a unique user identifier that is associated to a Cognito Identity.

- Facebook iOS SDK by Xamarin
- Facebook Android SDK by Xamarin

Configure the External Provider in the Amazon Cognito Console

From the Amazon Cognito Console home page:

1. Choose **Manage Federated Identities**.
2. Choose the name of the identity pool for which you want to enable Facebook as an external provider. The **Dashboard** page for your identity pool appears.
3. In the top-right corner of the **Dashboard** page, choose **Edit identity pool**. The **Edit identity pool** page appears.
4. Scroll down and choose **Authentication providers** to expand it.
5. Choose the **Facebook** tab.
6. Choose **Unlock**.
7. Enter the Facebook App ID you obtained from Facebook, and then choose **Save Changes**.

Using Facebook

Android

To provide Facebook authentication, first follow the Facebook guide to include their SDK in your application. Then add a "Login with Facebook" button to your Android user interface. The Facebook SDK uses a session object to track its state. Amazon Cognito uses the access token from this session object to authenticate the user, generate the unique identifier, and, if needed, grant the user access to other AWS resources.

Once you have authenticated your user with the Facebook SDK, add the session token to the Amazon Cognito credentials provider.

Facebook SDK 4.0 or later:

```
Map<String, String> logins = new HashMap<String, String>();
logins.put("graph.facebook.com", AccessToken.getCurrentAccessToken().getToken());
credentialsProvider.setLogins(logins);
```

Facebook SDK before 4.0:

```
Map<String, String> logins = new HashMap<String, String>();
logins.put("graph.facebook.com", Session.getActiveSession().getAccessToken());
credentialsProvider.setLogins(logins);
```

The Facebook login process initializes a singleton session in its SDK. The Facebook session object contains an OAuth token that Amazon Cognito uses to generate AWS credentials for your authenticated end user. Amazon Cognito also uses the token to check against your user database for the existence of a user matching this particular Facebook identity. If the user already exists, the API returns the existing identifier. Otherwise a new identifier is returned. Identifiers are automatically cached by the client SDK on the local device.

> **Note**
> After setting the logins map, you'll need to make a call to `refresh` or `get` to actually get the AWS credentials.

iOS - Objective-C

To add Facebook authentication, first follow the Facebook guide to integrate the Facebook SDK into your application. Then add a Login with Facebook button to your user interface. The Facebook SDK uses a session object to track its state. Amazon Cognito uses the access token from this session object to authenticate the user and bind them to a unique Amazon Cognito identity.

To provide the Facebook access token to Amazon Cognito, implement the AWSIdentityProviderManager protocol.

In the implementation of the `logins` method, return a dictionary containing `AWSIdentityProviderFacebook` as the key and the current access token from the authenticated Facebook user as the value, as shown in the following code example.

```
- (AWSTask<NSDictionary<NSString *, NSString *> *> *)logins {
    FBSDKAccessToken* fbToken = [FBSDKAccessToken currentAccessToken];
    if(fbToken){
        NSString *token = fbToken.tokenString;
        return [AWSTask taskWithResult: @{ AWSIdentityProviderFacebook : token }];
    }else{
            return [AWSTask taskWithError:[NSError errorWithDomain:@"Facebook Login"
                                                          code:-1
                                                      userInfo:@{@"error":@"No current
 Facebook access token"}]];
    }
}
```

When you instantiate the `AWSCognitoCredentialsProvider`, pass the class that implements `AWSIdentityProviderManager` as the value of `identityProviderManager` in the constructor. For more information, go to the AWSCognitoCredentialsProvider reference page and choose **initWithRegionType:identityPoolId:identityProviderManager**.

iOS - Swift

To add Facebook authentication, first follow the Facebook guide to integrate the Facebook SDK into your application. Then add a Login with Facebook button to your user interface. The Facebook SDK uses a session object to track its state. Amazon Cognito uses the access token from this session object to authenticate the user and bind them to a unique Amazon Cognito identity.

To provide the Facebook access token to Amazon Cognito, implement the AWSIdentityProviderManager protocol.

In the implementation of the `logins` method, return a dictionary containing `AWSIdentityProviderFacebook` as the key and the current access token from the authenticated Facebook user as the value, as shown in the following code example.

```
class FacebookProvider: NSObject, AWSIdentityProviderManager {
    func logins() -> AWSTask<NSDictionary> {
        if let token = AccessToken.current?.authenticationToken {
            return AWSTask(result: [AWSIdentityProviderFacebook:token])
        }
        return AWSTask(error:NSError(domain: "Facebook Login", code: -1 , userInfo:
 ["Facebook" : "No current Facebook access token"]))
    }
}
```

When you instantiate the `AWSCognitoCredentialsProvider`, pass the class that implements `AWSIdentityProviderManager` as the value of `identityProviderManager` in the constructor. For more information, go to the AWSCognitoCredentialsProvider reference page and choose **initWithRegionType:identityPoolId:identityProviderManager**.

JavaScript

To provide Facebook authentication, follow the Facebook Login for the Web to add the "Login with Facebook" button on your website. The Facebook SDK uses a session object to track its state. Amazon Cognito uses the access token from this session object to authenticate the user, generate the unique identifier, and, if needed, grant the user access to other AWS resources.

Once you have authenticated your user with the Facebook SDK, add the session token to the Amazon Cognito credentials provider.

```
FB.login(function (response) {

  // Check if the user logged in successfully.
  if (response.authResponse) {

    console.log('You are now logged in.');

    // Add the Facebook access token to the Cognito credentials login map.
    AWS.config.credentials = new AWS.CognitoIdentityCredentials({
      IdentityPoolId: 'IDENTITY_POOL_ID',
      Logins: {
        'graph.facebook.com': response.authResponse.accessToken
      }
    });

    // Obtain AWS credentials
    AWS.config.credentials.get(function(){
        // Access AWS resources here.
    });

  } else {
    console.log('There was a problem logging you in.');
  }

});
```

The Facebook SDK obtains an OAuth token that Amazon Cognito uses to generate AWS credentials for your authenticated end user. Amazon Cognito also uses the token to check against your user database for the existence of a user matching this particular Facebook identity. If the user already exists, the API returns the existing identifier. Otherwise a new identifier is returned. Identifiers are automatically cached by the client SDK on the local device.

> **Note**
> After setting the logins map, you need to make a call to `refresh` or `get` to get the AWS credentials. For a code example, see "Use Case 17, Integrating User Pools with Cognito Identity," in the JavaScript README file.

Unity

To provide Facebook authentication, first follow the Facebook guide to include and set up their SDK in your application. Amazon Cognito uses the Facebook access token from the 'FB' object to generate a unique user identifier that is associated to a Cognito Identity.

Once you have authenticated your user with the Facebook SDK, add the session token to the Amazon Cognito credentials provider:

```
void Start()
{
    FB.Init(delegate() {
        if (FB.IsLoggedIn) { //User already logged in from a previous session
            AddFacebookTokenToCognito();
        } else {
            FB.Login ("email", FacebookLoginCallback);
        }
    });
}

void FacebookLoginCallback(FBResult result)
{
    if (FB.IsLoggedIn)
    {
        AddFacebookTokenToCognito();
    }
    else
    {
        Debug.Log("FB Login error");
    }
}

void AddFacebookTokenToCognito()
{
    credentials.AddLogin ("graph.facebook.com",
 AccessToken.CurrentAccessToken.TokenString);
}
```

You should make sure to call `FB.Login()` and that `FB.IsLoggedIn` is true before using `FB.AccessToken`.

Xamarin

Xamarin for Android:

```
public void InitializeFacebook() {
    FacebookSdk.SdkInitialize(this.ApplicationContext);
    callbackManager = CallbackManagerFactory.Create();
    LoginManager.Instance.RegisterCallback(callbackManager, new FacebookCallback &lt;
 LoginResult &gt; () {
      HandleSuccess = loginResult = &gt; {
        var accessToken = loginResult.AccessToken;
        credentials.AddLogin("graph.facebook.com", accessToken.Token);
        //open new activity
      },
      HandleCancel = () = &gt; {
        //throw error message
      },
      HandleError = loginError = &gt; {
        //throw error message
      }
    });
    LoginManager.Instance.LogInWithReadPermissions(this, new List &lt; string &gt; {
      "public_profile"
    });
  }
```

Xamarin for iOS:

```
public void InitializeFacebook() {
  LoginManager login = new LoginManager();
  login.LogInWithReadPermissions(readPermissions.ToArray(),
 delegate(LoginManagerLoginResult result, NSError error) {
    if (error != null) {
      //throw error message
    } else if (result.IsCancelled) {
      //throw error message
    } else {
      var accessToken = loginResult.AccessToken;
      credentials.AddLogin("graph.facebook.com", accessToken.Token);
      //open new view controller
    }
  });
}
```

Login with Amazon

Amazon Cognito integrates with Login with Amazon to provide federated authentication for your mobile application and web application users. This section explains how to register and set up your application using Login with Amazon as an identity provider.

There are two ways to set up Login with Amazon to work with Amazon Cognito. If you're not sure which one to use, or if you need to use both, see "Setting Up Login with Amazon" in the Login with Amazon FAQ.

- Through the Amazon Developer Portal. Use this method if you want to let your end users authenticate with Login with Amazon, but you don’t have a Seller Central account.
- Through Seller Central using http://login.amazon.com/. Use this method if you are a retail merchant that uses Seller Central.

> **Note**
> For Xamarin, follow the Xamarin Getting Started Guide to integrate Login with Amazon into your Xamarin application.
>
> **Note**
> Login with Amazon integration is not natively supported on the Unity platform. Integration currently requires the use of a web view to go through the browser sign in flow.

Setting Up Login with Amazon

To implement Login with Amazon, do one of the following:

- Create a Security Profile ID for your application through the Amazon Developer Portal. Use this method if you want to let your end users authenticate with Amazon, but you don't have a Seller Central account. The Developer Portal Login with Amazon documentation takes you through the process of setting up Login with Amazon in your application, downloading the client SDK, and declaring your application on the Amazon developer platform. Make a note of the Security Profile ID, as you'll need to enter it as the Amazon App ID when you create an Amazon Cognito identity pool, as described in Getting Credentials.
- Create an Application ID for your application through Seller Central using http://login.amazon.com/. Use this method if you are a retail merchant that uses Seller Central. The Seller Central Login with Amazon documentation takes you through the process of setting up Login with Amazon in your application, downloading the client SDK, and declaring your application on the Amazon developer platform. Make a note of the Application ID, as you'll need to enter it as the Amazon App ID when you create an Amazon Cognito identity pool, as described in Getting Credentials.

Configure the External Provider in the Amazon Cognito Console

From the Amazon Cognito Console home page:

1. Click the name of the identity pool for which you want to enable Login with Amazon as an external provider. The **Dashboard** page for your identity pool appears.
2. In the top-right corner of the **Dashboard** page, click **Edit identity pool**. The Edit identity pool page appears.
3. Scroll down and click **Authentication providers** to expand it.
4. Choose the **Amazon** tab.
5. Choose **Unlock**.
6. Enter the Amazon App ID you obtained from Amazon, and then choose **Save Changes**.

Use Login with Amazon: Android

Once you've implemented Amazon login, you can pass the token to the Amazon Cognito credentials provider in the onSuccess method of the TokenListener interface. The code looks like this:

```
@Override
public void onSuccess(Bundle response) {
    String token = response.getString(AuthzConstants.BUNDLE_KEY.TOKEN.val);
    Map<String, String> logins = new HashMap<String, String>();
    logins.put("www.amazon.com", token);
    credentialsProvider.setLogins(logins);
}
```

Use Login with Amazon: iOS - Objective-C

Once you've implemented Amazon login, you can pass the token to the Amazon Cognito credentials provider in the requestDidSucceed method of the AMZNAccessTokenDelegate:

```
- (void)requestDidSucceed:(APIResult \*)apiResult {
    if (apiResult.api == kAPIAuthorizeUser) {
        [AIMobileLib getAccessTokenForScopes:[NSArray arrayWithObject:@"profile"]
 withOverrideParams:nil delegate:self];
    }
    else if (apiResult.api == kAPIGetAccessToken) {
        credentialsProvider.logins = @{ @(AWSCognitoLoginProviderKeyLoginWithAmazon):
 apiResult.result };
    }
}}
```

Use Login with Amazon: iOS - Swift

Once you've implemented Amazon login, you can pass the token to the Amazon Cognito credentials provider in the `requestDidSucceed` method of the `AMZNAccessTokenDelegate`:

```
func requestDidSucceed(apiResult: APIResult!) {
    if apiResult.api == API.AuthorizeUser {
        AIMobileLib.getAccessTokenForScopes(["profile"], withOverrideParams: nil, delegate:
 self)
    } else if apiResult.api == API.GetAccessToken {
        credentialsProvider.logins = [AWSCognitoLoginProviderKey.LoginWithAmazon.rawValue:
 apiResult.result]
    }
}
```

Use Login with Amazon: JavaScript

After the user authenticates with Login with Amazon and is redirected back to your website, the Login with Amazon access_token is provided in the query string. Pass that token into the credentials login map.

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
   IdentityPoolId: 'IDENTITY_POOL_ID',
   Logins: {
       'www.amazon.com': 'Amazon Access Token'
   }
});
```

Use Login with Amazon: Xamarin

Xamarin for Android

```
AmazonAuthorizationManager manager = new AmazonAuthorizationManager(this, Bundle.Empty);

var tokenListener = new APIListener {
  Success = response => {
    // Get the auth token
    var token = response.GetString(AuthzConstants.BUNDLE_KEY.Token.Val);
    credentials.AddLogin("www.amazon.com", token);
  }
};

// Try and get existing login
manager.GetToken(new[] {
  "profile"
}, tokenListener);
```

Xamarin for iOS

In AppDelegate.cs, insert the following:

```
public override bool OpenUrl (UIApplication application, NSUrl url, string
 sourceApplication, NSObject annotation)
{
    // Pass on the url to the SDK to parse authorization code from the url
    bool isValidRedirectSignInURL = AIMobileLib.HandleOpenUrl (url, sourceApplication);
    if(!isValidRedirectSignInURL)
        return false;

    // App may also want to handle url
    return true;
}
```

Then, in ViewController.cs, do the following:

```
public override void ViewDidLoad ()
{
    base.LoadView ();

    // Here we create the Amazon Login Button
    btnLogin = UIButton.FromType (UIButtonType.RoundedRect);
    btnLogin.Frame = new RectangleF (55, 206, 209, 48);
    btnLogin.SetTitle ("Login using Amazon", UIControlState.Normal);
    btnLogin.TouchUpInside += (sender, e) => {
        AIMobileLib.AuthorizeUser (new [] { "profile"}, new AMZNAuthorizationDelegate ());
    };
    View.AddSubview (btnLogin);
}

// Class that handles Authentication Success/Failure
public class AMZNAuthorizationDelegate : AIAuthenticationDelegate
{
  public override void RequestDidSucceed(ApiResult apiResult)
    {
      // Your code after the user authorizes application for requested scopes
      var token = apiResult["access_token"];
      credentials.AddLogin("www.amazon.com",token);
    }

    public override void RequestDidFail(ApiError errorResponse)
    {
      // Your code when the authorization fails
      InvokeOnMainThread(() => new UIAlertView("User Authorization Failed",
 errorResponse.Error.Message, null, "Ok", null).Show());
    }
}
```

Google

Amazon Cognito integrates with Google to provide federated authentication for your mobile application users. This section explains how to register and set up your application with Google as an identity provider.

Android

> **Note**
> If your app uses Google and will be available on multiple mobile platforms, you should configure it as an OpenID Connect Provider, adding all created client IDs as additional audience values to allow for better integration. To learn more about Google's cross-client identity model, see Cross-client Identity.

Set Up Google

To enable Google+ Sign-in for Android, you will need to create a Google Developers console project for your application.

1. Go to the Google Developers console and create a new project.

2. Under **APIs and auth > APIs > Social APIs**, enable the Google+ API.
3. Under **APIs and auth > Credentials > OAuth consent screen**, create the dialog that will be shown to users when your app requests access to their private data.
4. Under **Credentials > Add Credentials**, create an OAuth 2.0 client ID for Android. You will need a client ID for each platform you intend to develop for (e.g. web, iOS, Android).
5. Under Credentials > Add Credentials, create a Service Account. The console will alert you that a new public/private key has been created.

For additional instructions on using the Google Developers console, see Managing projects in the Developers Console.

For additional instructions on integrating Google+ into your Android app, see the Google documentation for Android.

Configure the External Provider in the Amazon Cognito Console

From the Amazon Cognito Console home page:

1. Click the name of the identity pool for which you want to enable Amazon as an external provider. The Dashboard page for your identity pool appears.
2. In the top-right corner of the Dashboard page, click Edit identity pool. The Edit identity pool page appears.
3. Scroll down and click Authentication providers to expand it.
4. Click the Google tab.
5. Click Unlock.
6. Enter the Google Client ID you obtained from Google, and then click Save Changes.

Use Google

To enable login with Google in your application, follow the Google+ documentation for Android. Successful authentication results in an OpenID Connect authentication token, which Amazon Cognito uses to authenticate the user and generate a unique identifier.

The following sample code shows how to retrieve the authentication token from the Google Play Service:

```
GooglePlayServicesUtil.isGooglePlayServicesAvailable(getApplicationContext());
AccountManager am = AccountManager.get(this);
Account[] accounts = am.getAccountsByType(GoogleAuthUtil.GOOGLE_ACCOUNT_TYPE);
String token = GoogleAuthUtil.getToken(getApplicationContext(), accounts[0].name,
        "audience:server:client_id:YOUR_GOOGLE_CLIENT_ID");
Map<String, String> logins = new HashMap<String, String>();
logins.put("accounts.google.com", token);
credentialsProvider.setLogins(logins);
```

iOS - Objective-C

> **Note**
> If your app uses Google and will be available on multiple mobile platforms, you should configure it as an OpenID Connect Provider, adding all created client IDs as additional audience values to allow for better integration. To learn more about Google's cross-client identity model, see Cross-client Identity.

To enable Google+ Sign-in for iOS, you will need to create a Google Developers console project for your application.

Set Up Google

1. Go to the Google Developers console and create a new project.
2. Under **APIs and auth > APIs > Social APIs**, enable the Google+ API.
3. Under **APIs and auth > Credentials > OAuth consent screen**, create the dialog that will be shown to users when your app requests access to their private data.
4. Under **Credentials > Add Credentials**, create an OAuth 2.0 client ID for iOS. You will need a client ID for each platform you intend to develop for (e.g. web, iOS, Android).
5. Under Credentials > Add Credentials, create a Service Account. The console will alert you that a new public/private key has been created.

For additional instructions on using the Google Developers console, see Managing projects in the Developers Console.

For additional instructions on integrating Google+ into your iOS app, see the Google documentation for iOS.

From the Amazon Cognito Console home page:

Configure the External Provider in the Amazon Cognito Console

1. Click the name of the identity pool for which you want to enable Amazon as an external provider. The Dashboard page for your identity pool appears.
2. In the top-right corner of the Dashboard page, click Edit identity pool. The Edit identity pool page appears.
3. Scroll down and click Authentication providers to expand it.
4. Click the Google tab.
5. Click Unlock.
6. Enter the Google Client ID you obtained from Google, and then click Save Changes.

Use Google

To enable login with Google in your application, follow the Google+ documentation for iOS. Successful authentication results in an OpenID Connect authentication token, which Amazon Cognito uses to authenticate the user and generate a unique identifier.

Successful authentication results in a GTMOAuth2Authentication object which contains an id_token, which Amazon Cognito uses to authenticate the user and generate a unique identifier:

```
- (void)finishedWithAuth: (GTMOAuth2Authentication *)auth error: (NSError *) error {
        NSString *idToken = [auth.parameters objectForKey:@"id_token"];
        credentialsProvider.logins = @{ @(AWSCognitoLoginProviderKeyGoogle): idToken };
    }
```

iOS - Swift

> **Note**
> If your app uses Google and will be available on multiple mobile platforms, you should configure it as an OpenID Connect Provider, adding all created client IDs as additional audience values to allow for better integration. To learn more about Google's cross-client identity model, see Cross-client Identity.

To enable Google+ Sign-in for iOS, you will need to create a Google Developers console project for your application.

Set Up Google

1. Go to the Google Developers console and create a new project.
2. Under **APIs and auth > APIs > Social APIs**, enable the Google+ API.
3. Under **APIs and auth > Credentials > OAuth consent screen**, create the dialog that will be shown to users when your app requests access to their private data.
4. Under **Credentials > Add Credentials**, create an OAuth 2.0 client ID for iOS. You will need a client ID for each platform you intend to develop for (e.g. web, iOS, Android).
5. Under Credentials > Add Credentials, create a Service Account. The console will alert you that a new public/private key has been created.

For additional instructions on using the Google Developers console, see Managing projects in the Developers Console.

For additional instructions on integrating Google+ into your iOS app, see the Google documentation for iOS.

From the Amazon Cognito Console home page:

Configure the External Provider in the Amazon Cognito Console

1. Click the name of the identity pool for which you want to enable Amazon as an external provider. The Dashboard page for your identity pool appears.
2. In the top-right corner of the Dashboard page, click Edit identity pool. The Edit identity pool page appears.
3. Scroll down and click Authentication providers to expand it.
4. Click the Google tab.
5. Click Unlock.
6. Enter the Google Client ID you obtained from Google, and then click Save Changes.

Use Google

To enable login with Google in your application, follow the Google+ documentation for iOS. Successful authentication results in an OpenID Connect authentication token, which Amazon Cognito uses to authenticate the user and generate a unique identifier.

Successful authentication results in a `GTMOAuth2Authentication` object which contains an `id_token`, which Amazon Cognito uses to authenticate the user and generate a unique identifier:

```
func finishedWithAuth(auth: GTMOAuth2Authentication!, error: NSError!) {
    if error != nil {
      print(error.localizedDescription)
    }
    else {
      let idToken = auth.parameters.objectForKey("id_token")
      credentialsProvider.logins = [AWSCognitoLoginProviderKey.Google.rawValue: idToken!]
    }
}
```

JavaScript

Note
If your app uses Google and will be available on multiple mobile platforms, you should configure it as an OpenID Connect Provider, adding all created client IDs as additional audience

values to allow for better integration. To learn more about Google's cross-client identity model, see Cross-client Identity.

Set Up Google

To enable Google+ Sign-in for your web application, you will need to create a Google Developers console project for your application.

1. Go to the Google Developers console and create a new project.
2. Under **APIs and auth > APIs > Social APIs**, enable the Google+ API.
3. Under **APIs and auth > Credentials > OAuth consent screen**, create the dialog that will be shown to users when your app requests access to their private data.
4. Under **Credentials > Add Credentials**, create an OAuth 2.0 client ID for your web application. You will need a client ID for each platform you intend to develop for (e.g. web, iOS, Android).
5. Under Credentials > Add Credentials, create a Service Account. The console will alert you that a new public/private key has been created.

For additional instructions on using the Google Developers console, see Managing projects in the Developers Console.

Configure the External Provider in the Amazon Cognito Console

From the Amazon Cognito Console home page:

1. Click the name of the identity pool for which you want to enable Amazon as an external provider. The Dashboard page for your identity pool appears.
2. In the top-right corner of the Dashboard page, click Edit identity pool. The Edit identity pool page appears.
3. Scroll down and click Authentication providers to expand it.
4. Click the Google tab.
5. Click Unlock.
6. Enter the Google Client ID you obtained from Google, and then click Save Changes.

Use Google

To enable login with Google in your application, follow the Google+ documentation for Web.

Successful authentication results in a response object which contains an `id_token`, which Amazon Cognito uses to authenticate the user and generate a unique identifier:

```
function signinCallback(authResult) {
  if (authResult['status']['signed_in']) {

     // Add the Google access token to the Cognito credentials login map.
     AWS.config.credentials = new AWS.CognitoIdentityCredentials({
        IdentityPoolId: 'IDENTITY_POOL_ID',
        Logins: {
           'accounts.google.com': authResult['id_token']
        }
     });

     // Obtain AWS credentials
     AWS.config.credentials.get(function(){
        // Access AWS resources here.
     });
  }
}
```

Unity

Set Up Google

To enable Google+ Sign-in for your web application, you will need to create a Google Developers console project for your application.

1. Go to the Google Developers console and create a new project.
2. Under **APIs and auth > APIs > Social APIs**, enable the Google+ API.
3. Under **APIs and auth > Credentials > OAuth consent screen**, create the dialog that will be shown to users when your app requests access to their private data.
4. For Unity, you need to create a total of three IDs: two for Android and one for iOS. Under **Credentials > Add Credentials**:
   - Android: Create an OAuth 2.0 client ID for Android and an OAuth 2.0 client ID for a web application.
   - iOS: Create an OAuth 2.0 client ID for iOS.
5. Under Credentials > Add Credentials, create a Service Account. The console will alert you that a new public/private key has been created.

Create an OpenID Provider in the IAM Console

1. Next, you will need to create an OpenID Provider in the IAM Console. For instructions on how to set up an OpenID Provider, see Using OpenID Connect Identity Providers.
2. When prompted for your Provider URL, enter `"https://accounts.google.com"`.
3. When prompted to enter a value in the **Audience** field, enter any one of the three client IDs your created in the previous steps.
4. After creating the provider, click on the provider name and add two more audiences, providing the two remaining client IDs.

Configure the External Provider in the Amazon Cognito Console

From the Amazon Cognito Console home page:

1. Click the name of the identity pool for which you want to enable Amazon as an external provider. The Dashboard page for your identity pool appears.
2. In the top-right corner of the Dashboard page, click Edit identity pool. The Edit identity pool page appears.
3. Scroll down and click Authentication providers to expand it.
4. Click the Google tab.
5. Click Unlock.
6. Enter the Google Client ID you obtained from Google, and then click Save Changes.

Install the Unity Google Plugin

1. Add the Google Play Games plugin for Unity to your Unity project.
2. In Unity, from the **Windows** menu, configure the plugin using the three IDs for the Android and iOS platforms.

Use Google

The following sample code shows how to retrieve the authentication token from the Google Play Service:

```
void Start()
{
  PlayGamesClientConfiguration config = new PlayGamesClientConfiguration.Builder().Build();
  PlayGamesPlatform.InitializeInstance(config);
  PlayGamesPlatform.DebugLogEnabled = true;
  PlayGamesPlatform.Activate();
  Social.localUser.Authenticate(GoogleLoginCallback);
}

void GoogleLoginCallback(bool success)
{
  if (success)
  {
    string token = PlayGamesPlatform.Instance.GetIdToken();
    credentials.AddLogin("accounts.google.com", token);
  }
  else
  {
    Debug.LogError("Google login failed. If you are not running in an actual Android/iOS
 device, this is expected.");
  }
}
```

Xamarin

Note
Google integration is not natively supported on the Xamarin platform. Integration currently requires the use of a web view to go through the browser sign in flow. To learn how Google integration works with other SDKs, please select another platform.

To enable login with Google in your application, you will need to authenticate your users and obtain an OpenID Connect token from them. Amazon Cognito uses this token to generate a unique user identifier that is associated to a Cognito Identity. Unfortunately, the Google SDK for Xamarin doesn't allow you to retrieve the OpenID Connect token, so you will need to use an alternative client or the web flow in a web view.

Once you have the token, you can set it in your `CognitoAWSCredentials`:

```
credentials.AddLogin("accounts.google.com", token);
```

Note
If your app uses Google and will be available on multiple mobile platforms, you should configure it as an OpenID Connect Provider, adding all created client IDs as additional audience values to allow for better integration. To learn more about Google's cross-client identity model, see Cross-client Identity.

Twitter/Digits

Android

Amazon Cognito integrates with Twitter and Digits to provide federated authentication for your mobile application users. This section explains how to register and set up your application with Twitter and Digits as an identity provider.

Set Up Twitter/Digits

You will need to integrate the TwitterKit SDK in your application. Twitter offers Fabric as a mechanism for enabling various functionality in your application.

Once you've integrated TwitterKit into your application, you get the following values from the Fabric console:

- Consumer Key
- Consumer Secret

You must enter these values in the Amazon Cognito Console to configure your identity pool for Twitter/Digits integration.

Using Twitter

Using the TwitterKit SDK login functionality, we need to simply capture the result of the completion handler and pass the appropriate fields to Amazon Cognito.

Twitter sessions contain two important values:

- User token
- User secret

Amazon Cognito expects these values to be stored in a single value in `logins` with the key `api.twitter.com`, concatenated with a single semicolon (`;`) delimiter.

```
loginButton = (TwitterLoginButton) findViewById(R.id.login_button);
loginButton.setCallback(new Callback<TwitterSession>() {
   @Override
   public void success(Result<TwitterSession> result) {
       TwitterSession session = result.data;
       TwitterAuthToken authToken = session.getAuthToken();
       String value = authToken.token + ";" + authToken.secret;
       Map<String, String> logins = new HashMap<String, String>();
       logins.put("api.twitter.com", value);
       // Note: This overrides any existing logins
       credentialsProvider.setLogins(logins);
   }

   @Override
   public void failure(TwitterException exception) {
       // Do something on failure
   }
});
```

Using Digits

Digits support is handled through a separate call in the Fabric SDK, but the session returned is essentially the same as with Twitter login. You just need to concatenate the token and secret from the session using a single semicolon (`;`) delimiter and store in `logins` with the key of `www.digits.com`.

```
DigitsAuthButton digitsButton = (DigitsAuthButton) findViewById(R.id.auth_button);
digitsButton.setCallback(new AuthCallback() {
   @Override
   public void success(DigitsSession session, String phoneNumber) {
       TwitterAuthToken authToken = (TwitterAuthToken) session.getAuthToken();
       String value = authToken.token + ";" + authToken.secret;
       Map<String, String> logins = new HashMap<String, String>();
       logins.put("www.digits.com", value);
       // Note: This overrides any existing logins
       credentialsProvider.setLogins(logins);
   }

   @Override
   public void failure(DigitsException exception) {
       // Do something on failure
   }
});
```

iOS

Amazon Cognito integrates with Twitter and Digits to provide federated authentication for your mobile application users. This section explains how to register and set up your application with Twitter and Digits as an identity provider.

Set Up Twitter/Digits

You will need to integrate the TwitterKit SDK in your application. Twitter offers Fabric as a mechanism for enabling various functionality in your application.

Once you've integrated TwitterKit into your application, you get the following values from the Fabric console:

- Consumer Key
- Consumer Secret

You must enter these values in the Amazon Cognito Console to configure your identity pool for Twitter/Digits integration.

Using Twitter

Using the TwitterKit SDK login functionality, we need to simply capture the result of the completion handler and pass the appropriate fields to Amazon Cognito.

Twitter sessions contain two important values:

- User token
- User secret

Amazon Cognito expects these values to be stored in a single value in `logins` with the key `api.twitter.com`, concatenated with a single semicolon (`;`) delimiter.

Objective-C

```
[[Twitter sharedInstance] logInWithCompletion:^
            (TWTRSession *session, NSError *error) {
  if (session) {
      NSString value = [NSString stringWithFormat:@"%@;%@", session.authToken,
 session.authTokenSecret];
      // Note: This overrides any existing logins
      credentialsProvider.logins = @{@"api.twitter.com", value};
  } else {
      NSLog(@"error: %@", [error localizedDescription]);
  }
}];
```

Swift

```
Twitter.sharedInstance().logInWithCompletion {
                (session, error) -> Void in
    if (session != nil) {
        var value = session.authToken + ";" + session.authTokenSecret
        // Note: This overrides any existing logins
        credentialsProvider.logins = ["api.twitter.com": value]
    } else {
        println("error: (error.localizedDescription)")
    }
}
```

Using Digits

Digits support is handled through a separate call in the Fabric SDK, but the session returned is essentially the same as with Twitter login. You just need to concatenate the token and secret from the session using a single semicolon (`;`) delimiter and store in `logins` with the key of `www.digits.com`.

Objective-C

```
[[Digits sharedInstance] authenticateWithCompletion:^
            (DGTSession* session, NSError *error) {
    if (session) {
        NSString value = [NSString stringWithFormat:@"%@;%@", session.authToken,
 session.authTokenSecret];
      // Note: This overrides any existing logins
      credentialsProvider.logins = @{@"www.digits.com", value};
    }
}];
```

Swift

```
let digits = Digits.sharedInstance()
digits.authenticateWithCompletion { (session, error) in
    if (session != nil) {
        var value = session.authToken + ";" + session.authTokenSecret
        // Note: This overrides any existing logins
        credentialsProvider.logins = ["www.digits.com": value]
    }
}
```

JavaScript

Amazon Cognito integrates with Twitter and Digits to provide federated authentication for your mobile application users. This section explains how to register and set up your application with Twitter and Digits as an identity provider.

> **Note**
> We do not yet have documentation for integrating Twitter/Digits with the JavaScript SDK. To learn how Twitter/Digits integration works with other SDKs, select a different platform.

External Resources

- Sign in with Twitter
- Sign in with Digits

Unity

Amazon Cognito integrates with Twitter and Digits to provide federated authentication for your mobile application users. This section explains how to register and set up your application with Twitter and Digits as an identity provider.

> **Note**
> We do not yet have documentation for integrating Twitter/Digits with the Unity SDK. To learn how Twitter/Digits integration works with other SDKs, select a different platform.

External Resources

- Twitter Kit Fabric plugin for Unity

Xamarin

Twitter integration is not natively supported on the Xamarin platform. Integration currently requires the use of a web view to go through the browser sign in flow. To learn how Twitter/Digits integration works with other SDKs, please select another platform.

External Resources

- Browser sign in flow for Twitter

Open ID Connect Providers

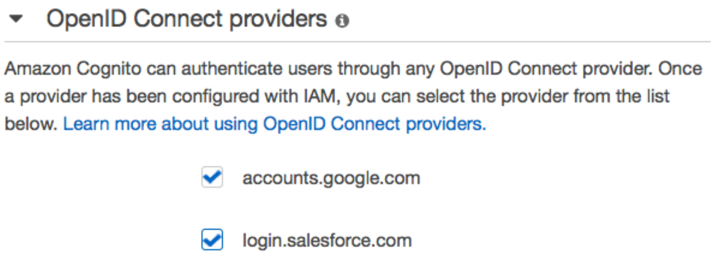
OpenID Connect is an open standard for authentication that is supported by a number of login providers. Amazon Cognito supports linking of identities with OpenID Connect providers that are configured through AWS Identity and Access Management.

Adding an OpenID Connect Provider

For information on how to create an OpenID Connect Provider, see the IAM documentation.

Associating a Provider to Amazon Cognito

Once you've created an OpenID Connect provider in the IAM Console, you can associate it to an identity pool. All configured providers will be visible in the Edit Identity Pool screen in the Amazon Cognito Console under the OpenID Connect Providers header.



OpenID Connect providers

Amazon Cognito can authenticate users through any OpenID Connect provider. Once a provider has been configured with IAM, you can select the provider from the list below. Learn more about using OpenID Connect providers.

☑ accounts.google.com

☑ login.salesforce.com

You can associate multiple OpenID Connect providers to a single identity pool.

Using OpenID Connect

Refer to your provider's documentation for how to login and receive an ID token.

Once you have a token, add the token to the logins map, using the URI of your provider as the key.

Validating an OpenID Connect Token

When first integrating with Amazon Cognito, you may receive an `InvalidToken` exception. It is important to understand how Amazon Cognito validates OpenID Connect tokens.

1. The **iss** parameter must match the key used in the logins map (e.g. login.provider.com).
2. The signature must be valid. The signature must be verifiable via an RSA public key.

3. The fingerprint of the certificate hosting the public key matches what's configured on your OpenId Connect Provider.
4. If the **azp** parameter is present, check this value against listed client IDs in your OpenId Connect provider.
5. If the **azp** parameter is not present, check the aud parameter against listed client IDs in your OpenId Connect provider.

The website jwt.io is a valuable resource for decoding tokens to verify these values.

Android

```
Map<String, String> logins = new HashMap<String, String>();
logins.put("login.provider.com", token);
credentialsProvider.setLogins(logins);
```

iOS - Objective-C

```
credentialsProvider.logins = @{ "login.provider.com": token }
```

iOS - Swift

To provide the OIDC id token to Amazon Cognito, implement the `AWSIdentityProviderManager` protocol.

In the implementation of the logins method, return a dictionary containing the OIDC provider name you configured as the key and the current id token from the authenticated user as the value, as shown in the following code example.

```
class OIDCProvider: NSObject, AWSIdentityProviderManager {
    func logins() -> AWSTask<NSDictionary> {
        let completion = AWSTaskCompletionSource<NSString>()
        getToken(tokenCompletion: completion)
        return completion.task.continueOnSuccessWith { (task) -> AWSTask<NSDictionary>? in
            //login.provider.name is the name of the OIDC provider as setup in the Cognito
 console
            return AWSTask(result:["login.provider.name":task.result!])
        } as! AWSTask<NSDictionary>

    }

    func getToken(tokenCompletion: AWSTaskCompletionSource<NSString>) -> Void {
        //get a valid oidc token from your server, or if you have one that hasn't expired
 cached, return it

        //TODO code to get token from your server
        //...

        //if error getting token, set error appropriately
        tokenCompletion.set(error:NSError(domain: "OIDC Login", code: -1 , userInfo:
 ["Unable to get OIDC token" : "Details about your error"]))
        //else
        tokenCompletion.set(result:"result from server id token")
    }
}
```

When you instantiate the `AWSCognitoCredentialsProvider`, pass the class that implements AWSIdentityProviderManager as the value of identityProviderManager in the constructor.

For more information, go to the AWSCognitoCredentialsProvider reference page and choose initWithRegionType:identityPoolId:identityProviderManager.

JavaScript

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
 IdentityPoolId: 'IDENTITY_POOL_ID',
 Logins: {
    'login.provider.com': token
 }
});
```

Unity

```
credentials.AddLogin("login.provider.com", token);
```

Xamarin

```
credentials.AddLogin("login.provider.com", token);
```

SAML Identity Provider

Amazon Cognito supports authentication with identity providers through Security Assertion Markup Language 2.0 (SAML 2.0). You can use an identity provider that supports SAML with Amazon Cognito to provide a simple onboarding flow for your users. Your SAML-supporting identity provider specifies the IAM roles that can be assumed by your users so that different users can be granted different sets of permissions.

Configuring Your Identity Pool for a SAML Provider

The following steps describe how to configure your identity pool to use a SAML-based provider.

> **Note**
> Before configuring your identity pool to support a SAML provider, you must first configure the SAML identity provider in the IAM console. For more information, see Integrating third-party SAML solution providers with AWS in the *IAM User Guide*.

To configure your identity pool to support a SAML provider

1. Sign in to the Amazon Cognito console, choose **Manage Federated Identities**, and choose **Create new identity pool**.
2. In the **Authentication providers** section, choose the **SAML** tab.
3. Choose the ARN of the SAML provider and then choose **Create Pool**.

Configuring Your SAML Identity Provider

After you create the SAML provider, configure your SAML identity provider to add relying party trust between your identity provider and AWS. Many identity providers allow you to specify a URL from which the identity provider can read an XML document that contains relying party information and certificates. For AWS, you can use https://signin.aws.amazon.com/static/saml-metadata.xml. The next step is to configure the SAML assertion response from your identity provider to populate the claims needed by AWS. For details on the claim configuration, see Configuring SAML assertions for authentication response.

Customizing Your User Role with SAML

Using SAML with Amazon Cognito Identity allows the role to be customized for the end user. Only the enhanced flow (p. 136) is supported with the SAML-based identity provider. You do not need to specify an authenticated or unauthenticated role for the identity pool to use a SAML-based identity provider. The `https://aws.amazon.com/SAML/Attributes/Role` claim attribute specifies one or more pairs of comma delimited role and provider ARN. These are the roles that the user is allowed to assume. The SAML identity provider can be configured to populate the role attributes based on the user attribute information available from the identity provider. If multiple roles are received in the SAML assertion, the optional `customRoleArn` parameter should be populated while calling `getCredentialsForIdentity`. The input role received in the parameter will be assumed by the user if it matches a role in the claim in the SAML assertion.

Authenticating Users with a SAML Identity Provider

To federate with the SAML-based identity provider, you must determine the URL that is being used to initiate the login. AWS federation uses IdP-initiated login. In AD FS 2.0 the URL takes the form of `https://<fqdn>/adfs/ls/IdpInitiatedSignOn.aspx?loginToRp=urn:amazon:webservices`.

To add support for your SAML identity provider in Amazon Cognito, you must first authenticate users with your SAML identity provider from your iOS or Android app. The code for integrating and authenticating with the SAML identity provider is specific to SAML providers. After your user is authenticated, you can provide the resulting SAML assertion to Amazon Cognito Identity using Amazon Cognito APIs.

Android

If you are using the Android SDK you can populate the logins map with the SAML assertion as follows.

```
Map logins = new HashMap();
logins.put("arn:aws:iam::aws account id:saml-provider/name", "base64 encoded assertion
 response");
// Now this should be set to CognitoCachingCredentialsProvider object.
CognitoCachingCredentialsProvider credentialsProvider = new
 CognitoCachingCredentialsProvider(context, identity pool id, region);
credentialsProvider.setLogins(logins);
// If SAML assertion contains multiple roles, resolve the role by setting the custom role
credentialsProvider.setCustomRoleArn("arn:aws:iam::aws account id:role/customRoleName");
// This should trigger a call to Cognito service to get the credentials.
credentialsProvider.getCredentials();
```

iOS

If you are using the iOS SDK you can provide the SAML assertion in `AWSIdentityProviderManager` as follows.

```
- (AWSTask<NSDictionary<NSString*,NSString*> *> *) logins {
    //this is hardcoded for simplicity, normally you would asynchronously go to your SAML
 provider
    //get the assertion and return the logins map using a AWSTaskCompletionSource
    return [AWSTask taskWithResult:@{@"arn:aws:iam::aws account id:saml-provider/
name":@"base64 encoded assertion response"}];
}

// If SAML assertion contains multiple roles, resolve the role by setting the custom role.
// Implementing this is optional if there is only one role.
- (NSString *)customRoleArn {
    return @"arn:aws:iam::accountId:role/customRoleName";
}
```

Developer Authenticated Identities

Amazon Cognito supports developer authenticated identities, in addition to web identity federation through Facebook (p. 155), Google (p. 163), and Login with Amazon (p. 160). With developer authenticated identities, you can register and authenticate users via your own existing authentication process, while still using Amazon Cognito to synchronize user data and access AWS resources. Using developer authenticated identities involves interaction between the end user device, your backend for authentication, and Amazon Cognito. For more details, please read our blog.

Understanding the Authentication Flow

For information on the developer authenticated identities authflow and how it differs from the external provider authflow, see Authentication Flow (p. 136).

Define a Developer Provider Name and Associate it with an Identity Pool

To use developer authenticated identities, you'll need an identity pool associated with your developer provider. To do so, follow these steps:

1. Log in to the Amazon Cognito console.
2. Create a new identity pool and, as part of the process, define a developer provider name in the **Custom** tab in **Authentication Providers**.
3. Alternatively, edit an existing identity pool and define a developer provider name in the **Custom** tab in **Authentication Providers**.

Note: Once the provider name has been set, it cannot be changed.

For additional instructions on working with the Amazon Cognito Console, see Using the Amazon Cognito Console (p. 230).

Implement an Identity Provider

Android

To use developer authenticated identities, implement your own identity provider class which extends `AWSAbstractCognitoIdentityProvider`. Your identity provider class should return a response object containing the token as an attribute.

Below is a simple example of an identity provider which is used in our sample app:

```
public class DeveloperAuthenticationProvider extends
 AWSAbstractCognitoDeveloperIdentityProvider {

  private static final String developerProvider = "<Developer_provider_name>";

  public DeveloperAuthenticationProvider(String accountId, String identityPoolId, Regions
 region) {
    super(accountId, identityPoolId, region);
    // Initialize any other objects needed here.
  }

  // Return the developer provider name which you choose while setting up the
  // identity pool in the &COG; Console
  @Override
  public String getProviderName() {
    return developerProvider;
  }

  // Use the refresh method to communicate with your backend to get an
  // identityId and token.

  @Override
  public String refresh() {

    // Override the existing token
    setToken(null);

    // Get the identityId and token by making a call to your backend
    // (Call to your backend)

    // Call the update method with updated identityId and token to make sure
    // these are ready to be used from Credentials Provider.

    update(identityId, token);
    return token;

  }

  // If the app has a valid identityId return it, otherwise get a valid
  // identityId from your backend.

  @Override
  public String getIdentityId() {

    // Load the identityId from the cache
    identityId = cachedIdentityId;

    if (identityId == null) {
       // Call to your backend
    } else {
       return identityId;
    }

  }
}
```

To use this identity provider, you have to pass it into `CognitoCachingCredentialsProvider`. Here's an example:

```
DeveloperAuthenticationProvider developerProvider = new
 DeveloperAuthenticationProvider( null, "IDENTITYPOOLID", context, Regions.USEAST1);
CognitoCachingCredentialsProvider credentialsProvider = new
 CognitoCachingCredentialsProvider( context, developerProvider, Regions.USEAST1);
```

iOS - Objective-C

To use developer authenticated identities, implement your own identity provider class which extends AWSCognitoCredentialsProviderHelper. Your identity provider class should return a response object containing the token as an attribute.

```
@implementation DeveloperAuthenticatedIdentityProvider
/*
 * Use the token method to communicate with your backend to get an
 * identityId and token.
 */

- (AWSTask <NSString*>) token {
    //Write code to call your backend:
    //Pass username/password to backend or some sort of token to authenticate user
    //If successful, from backend call getOpenIdTokenForDeveloperIdentity with logins map
    //containing "your.provider.name":"enduser.username"
    //Return the identity id and token to client
    //You can use AWSTaskCompletionSource to do this asynchronously

    // Set the identity id and return the token
    self.identityId = response.identityId;
    return [AWSTask taskWithResult:response.token];
}

@end
```

To use this identity provider, pass it into `AWSCognitoCredentialsProvider` as shown in the following example:

```
DeveloperAuthenticatedIdentityProvider * devAuth = [[DeveloperAuthenticatedIdentityProvider
 alloc] initWithRegionType:AWSRegionYOUR_IDENTITY_POOL_REGION
                                         identityPoolId:@"YOUR_IDENTITY_POOL_ID"
                                        useEnhancedFlow:YES
                                identityProviderManager:nil];
AWSCognitoCredentialsProvider *credentialsProvider = [[AWSCognitoCredentialsProvider alloc]

 initWithRegionType:AWSRegionYOUR_IDENTITY_POOL_REGION
                                                 identityProvider:devAuth];
```

If you want to support both unauthenticated identities and developer authenticated identities, override the `logins` method in your `AWSCognitoCredentialsProviderHelper` implementation.

```
- (AWSTask<NSDictionary<NSString *, NSString *> *> *)logins {
    if(/*logic to determine if user is unauthenticated*/) {
        return [AWSTask taskWithResult:nil];
    }else{
        return [super logins];
    }
}
```

If you want to support developer authenticated identities and social providers, you must manage who the current provider is in your `logins` implementation of `AWSCognitoCredentialsProviderHelper`.

```
- (AWSTask<NSDictionary<NSString *, NSString *> *> *)logins {
    if(/*logic to determine if user is unauthenticated*/) {
        return [AWSTask taskWithResult:nil];
    }else if (/*logic to determine if user is Facebook*/){
        return [AWSTask taskWithResult: @{ AWSIdentityProviderFacebook : [FBSDKAccessToken
 currentAccessToken] }];
    }else {
        return [super logins];
    }
}
```

iOS - Swift

To use developer authenticated identities, implement your own identity provider class which extends AWSCognitoCredentialsProviderHelper. Your identity provider class should return a response object containing the token as an attribute.

```
import AWSCore
/*
 * Use the token method to communicate with your backend to get an
 * identityId and token.
 */
class DeveloperAuthenticatedIdentityProvider : AWSCognitoCredentialsProviderHelper {
    override func token() -> AWSTask<NSString> {
    //Write code to call your backend:
    //pass username/password to backend or some sort of token to authenticate user, if
 successful,
    //from backend call getOpenIdTokenForDeveloperIdentity with logins map containing
 "your.provider.name":"enduser.username"
    //return the identity id and token to client
    //You can use AWSTaskCompletionSource to do this asynchronously

    // Set the identity id and return the token
    self.identityId = resultFromAbove.identityId
    return AWSTask(result: resultFromAbove.token)
}
```

To use this identity provider, pass it into `AWSCognitoCredentialsProvider` as shown in the following example:

```
let devAuth =
 DeveloperAuthenticatedIdentityProvider(regionType: .YOUR_IDENTITY_POOL_REGION,
 identityPoolId: "YOUR_IDENTITY_POOL_ID", useEnhancedFlow: true,
 identityProviderManager:nil)
let credentialsProvider =
 AWSCognitoCredentialsProvider(regionType: .YOUR_IDENTITY_POOL_REGION,
 identityProvider:devAuth)
let configuration = AWSServiceConfiguration(region: .YOUR_IDENTITY_POOL_REGION,
 credentialsProvider:credentialsProvider)
AWSServiceManager.default().defaultServiceConfiguration = configuration
```

If you want to support both unauthenticated identities and developer authenticated identities, override the `logins` method in your `AWSCognitoCredentialsProviderHelper` implementation.

```
override func logins () -> AWSTask<NSDictionary> {
    if(/*logic to determine if user is unauthenticated*/) {
        return AWSTask(result:nil)
    }else {
        return super.logins()
    }
}
```

If you want to support developer authenticated identities and social providers, you must manage who the current provider is in your `logins` implementation of `AWSCognitoCredentialsProviderHelper`.

```
override func logins () -> AWSTask<NSDictionary> {
    if(/*logic to determine if user is unauthenticated*/) {
        return AWSTask(result:nil)
    }else if (/*logic to determine if user is Facebook*/){
        if let token = AccessToken.current?.authenticationToken {
            return AWSTask(result: [AWSIdentityProviderFacebook:token])
        }
        return AWSTask(error:NSError(domain: "Facebook Login", code: -1 , userInfo:
 ["Facebook" : "No current Facebook access token"]))
    }else {
        return super.logins()
    }
}
```

JavaScript

Once you obtain an identity ID and session token from your backend, you will to pass them into the `AWS.CognitoIdentityCredentials` provider. Here's an example:

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
   IdentityPoolId: 'IDENTITY_POOL_ID',
   IdentityId: 'IDENTITY_ID_RETURNED_FROM_YOUR_PROVIDER',
   Logins: {
      'cognito-identity.amazonaws.com': 'TOKEN_RETURNED_FROM_YOUR_PROVIDER'
   }
});
```

Unity

To use developer-authenticated identities you have to extend `CognitoAWSCredentials` and override the `RefreshIdentity` method to retrieve the user identity id and token from your backend and return them. Below is a simple example of an identity provider that would contact a hypothetical backend at 'example.com':

```
using UnityEngine;
using System.Collections;
using Amazon.CognitoIdentity;
using System.Collections.Generic;
using ThirdParty.Json.LitJson;
using System;
using System.Threading;

public class DeveloperAuthenticatedCredentials : CognitoAWSCredentials
{
    const string PROVIDER_NAME = "example.com";
    const string IDENTITY_POOL = "IDENTITY_POOL_ID";
    static readonly RegionEndpoint REGION = RegionEndpoint.USEast1;

    private string login = null;

    public DeveloperAuthenticatedCredentials(string loginAlias)
        : base(IDENTITY_POOL, REGION)
    {
        login = loginAlias;
    }

    protected override IdentityState RefreshIdentity()
    {
        IdentityState state = null;
        ManualResetEvent waitLock = new ManualResetEvent(false);
        MainThreadDispatcher.ExecuteCoroutineOnMainThread(ContactProvider((s) =>
        {
            state = s;
            waitLock.Set();
        }));
        waitLock.WaitOne();
        return state;
    }

    IEnumerator ContactProvider(Action<IdentityState> callback)
    {
        WWW www = new WWW("http://example.com/?username="+login);
        yield return www;
        string response = www.text;

        JsonData json = JsonMapper.ToObject(response);
        //The backend has to send us back an Identity and a OpenID token
        string identityId = json["IdentityId"].ToString();
        string token = json["Token"].ToString();

        IdentityState state = new IdentityState(identityId, PROVIDER_NAME, token, false);
        callback(state);
    }
}
```

The code above uses a thread dispatcher object to call a coroutine. If you don't have a way to do this in your project, you can use the following script in your scenes:

```
using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class MainThreadDispatcher : MonoBehaviour
{
    static Queue<IEnumerator> _coroutineQueue = new Queue<IEnumerator>();
    static object _lock = new object();

    public void Update()
    {
        while (_coroutineQueue.Count > 0)
        {
            StartCoroutine(_coroutineQueue.Dequeue());
        }
    }

    public static void ExecuteCoroutineOnMainThread(IEnumerator coroutine)
    {
        lock (_lock) {
            _coroutineQueue.Enqueue(coroutine);
        }
    }
}
```

Xamarin

To use developer-authenticated identities you have to extend `CognitoAWSCredentials` and override the `RefreshIdentity` method to retrieve the user identity id and token from your backend and return them. Below is a simple example of an identity provider that would contact a hypothetical backend at 'example.com':

```
public class DeveloperAuthenticatedCredentials : CognitoAWSCredentials
{
    const string PROVIDER_NAME = "example.com";
    const string IDENTITY_POOL = "IDENTITY_POOL_ID";
    static readonly RegionEndpoint REGION = RegionEndpoint.USEast1;
    private string login = null;

    public DeveloperAuthenticatedCredentials(string loginAlias)
        : base(IDENTITY_POOL, REGION)
    {
        login = loginAlias;
    }

    protected override async Task<IdentityState> RefreshIdentityAsync()
    {
        IdentityState state = null;
        //get your identity and set the state
        return state;
    }
}
```

Updating the Logins Map (Android and iOS only)

Android

After successfully authenticating the user with your authentication system, update the logins map with the developer provider name and a developer user identifier, which is an alphanumeric string that uniquely identifies a user in your authentication system. Be sure to call the `refresh` method after updating the logins map as the `identityId` might have changed:

```
HashMap<String, String> loginsMap = new HashMap<String, String>();
loginsMap.put(developerAuthenticationProvider.getProviderName(), developerUserIdentifier);

credentialsProvider.setLogins(loginsMap);
credentialsProvider.refresh();
```

iOS - Objective-C

The iOS SDK only calls your `logins` method to get the latest logins map if there are no credentials or they have expired. If you want to force the SDK to obtain new credentials (e.g., your end user went from unauthenticated to authenticated and you want credentials against the authenticated user), call `clearCredentials` on your `credentialsProvider`.

```
[credentialsProvider clearCredentials];
```

iOS - Swift

The iOS SDK only calls your `logins` method to get the latest logins map if there are no credentials or they have expired. If you want to force the SDK to obtain new credentials (e.g., your end user went from unauthenticated to authenticated and you want credentials against the authenticated user), call `clearCredentials` on your `credentialsProvider`.

```
credentialsProvider.clearCredentials()
```

Getting a Token (Server Side)

You obtain a token by calling GetOpenIdTokenForDeveloperIdentity. This API must be invoked from your backend using AWS developer credentials. It must not be invoked from the client SDK. The API receives the Cognito identity pool ID; a logins map containing your identity provider name as the key and identifier as the value; and optionally a Cognito identity ID (i.e., you are making an unauthenticated user authenticated). The identifier can be the username of your user, an email address, or a numerical value. The API responds to your call with a unique Cognito ID for your user and an OpenID Connect token for the end user.

A few things to keep in mind about the token returned by `GetOpenIdTokenForDeveloperIdentity`:

- You can specify a custom expiration time for the token so you can cache it. If you don't provide any custom expiration time, the token is valid for 15 minutes.
- The maximum token duration you can set is 24 hours.

- Be mindful of the security implications of increasing the token duration. If an attacker obtains this token, they can exchange it for AWS credentials for the end user for the token duration.

The following Java snippet shows how to initialize an Amazon Cognito client and retrieve a token for a developer authenticated identity.

```
// authenticate your end user as appropriate
// ....

// if authenticated, initialize a cognito client with your AWS developer credentials
AmazonCognitoIdentity identityClient = new AmazonCognitoIdentityClient(
  new BasicAWSCredentials("access_key_id", "secret_access_key")
);

// create a new request to retrieve the token for your end user
GetOpenIdTokenForDeveloperIdentityRequest request =
  new GetOpenIdTokenForDeveloperIdentityRequest();
request.setIdentityPoolId("YOUR_COGNITO_IDENTITY_POOL_ID");

request.setIdentityId("YOUR_COGNITO_IDENTITY_ID"); //optional, set this if your client has
 an
                                                   //identity ID that you want to link to
 this
                                                   //developer account

// set up your logins map with the username of your end user
HashMap<String,String> logins = new HashMap<>();
logins.add("YOUR_IDENTITY_PROVIDER_NAME","YOUR_END_USER_IDENTIFIER");
request.setLogins(logins);

// optionally set token duration (in seconds)
request.setTokenDuration(60 * 15l);
GetOpenIdTokenForDeveloperIdentityResult response =
  identityClient.getOpenIdTokenForDeveloperIdentity(request);

// obtain identity id and token to return to your client
String identityId = response.getIdentityId();
String token = response.getToken();

//code to return identity id and token to client
//...
```

Following the steps above, you should be able to integrate developer authenticated identities in your app. If you have any issues or questions please feel free to post in our forums.

Connect to an Existing Social Identity

All linking of providers when you are using developer authenticated identities must be done from your backend. To connect a custom identity to a user's social identity (Facebook, Google, or Amazon), add the identity provider token to the logins map when you call GetOpenIdTokenForDeveloperIdentity. To make this possible, when you call your backend from your client SDK to authenticate your end user, additionally pass the end user's social provider token.

For example, if you are trying to link a custom identity to Facebook, you would add the Facebook token in addition to your identity provider identifier to the logins map when you call `GetOpenIdTokenForDeveloperIdentity`.

```
logins.add("YOUR_IDENTITY_PROVIDER_NAME","YOUR_END_USER_IDENTIFIER");
logins.add("graph.facebook.com","END_USERS_FACEBOOK_ACCESSTOKEN");
```

Supporting Transition Between Providers

Android

Your application might require supporting unauthenticated identities or authenticated identities using public providers (Login with Amazon, Facebook or Google) along with developer authenticated identities. The essential difference between developer authenticated identities and other identities (unauthenticated identities and authenticated identities using public provider) is the way the identityId and token are obtained. For other identities the mobile application will interact directly with Amazon Cognito instead of contacting your authentication system. So the mobile application should be able to support two distinct flows depending on the choice made by the app user. For this you will have to make some changes to the custom identity provider.

The `refresh` method should check the logins map, if the map is not empty and has a key with developer provider name, then you should call your backend; otherwise just call the getIdentityId method and return null.

```
public String refresh() {

   setToken(null);

   // If the logins map is not empty make a call to your backend
   // to get the token and identityId
   if (getProviderName() != null &&
      !this.loginsMap.isEmpty() &&
      this.loginsMap.containsKey(getProviderName())) {

      /**
       * This is where you would call your backend
       **/

      // now set the returned identity id and token in the provider
      update(identityId, token);
      return token;

   } else {
      // Call getIdentityId method and return null
      this.getIdentityId();
      return null;
   }
}
```

Similarly the `getIdentityId` method will have two flows depending on the contents of the logins map:

```
public String getIdentityId() {

   // Load the identityId from the cache
   identityId = cachedIdentityId;

   if (identityId == null) {

      // If the logins map is not empty make a call to your backend
      // to get the token and identityId

      if (getProviderName() != null && !this.loginsMap.isEmpty()
         && this.loginsMap.containsKey(getProviderName())) {

         /**
          * This is where you would call your backend
          **/
         // now set the returned identity id and token in the provider
         update(identityId, token);
         return token;

      } else {
         // Otherwise call &COG; using getIdentityId of super class
         return super.getIdentityId();
      }

   } else {
      return identityId;
   }

}
```

iOS - Objective-C

Your application might require supporting unauthenticated identities or authenticated identities using public providers (Login with Amazon, Facebook or Google) along with developer authenticated identities. To do this, override the AWSCognitoCredentialsProviderHelper `logins` method to be able to return the correct logins map based on the current identity provider. This example shows you how you might pivot between unauthenticated, Facebook and developer authenticated.

```
- (AWSTask<NSDictionary<NSString *, NSString *> *> *)logins {
    if(/*logic to determine if user is unauthenticated*/) {
        return [AWSTask taskWithResult:nil];
    }else if (/*logic to determine if user is Facebook*/){
        return [AWSTask taskWithResult: @{ AWSIdentityProviderFacebook : [FBSDKAccessToken
 currentAccessToken] }];
    }else {
        return [super logins];
    }
}
```

When you transition from unauthenticated to authenticated, you should call `[credentialsProvider clearCredentials];` to force the SDK to get new authenticated credentials. When you switch between two authenticated providers and you aren't trying to link the two providers (i.e. you are not providing tokens for multiple providers in your logins dictionary), you should call `[credentialsProvider clearKeychain];`. This will clear both the credentials and identity and force the SDK to get new ones.

iOS - Swift

Your application might require supporting unauthenticated identities or authenticated identities using public providers (Login with Amazon, Facebook or Google) along with developer authenticated identities. To do this, override the AWSCognitoCredentialsProviderHelper `logins` method to be able to return the correct logins map based on the current identity provider. This example shows you how you might pivot between unauthenticated, Facebook and developer authenticated.

```
override func logins () -> AWSTask<NSDictionary> {
    if(/*logic to determine if user is unauthenticated*/) {
        return AWSTask(result:nil)
    }else if (/*logic to determine if user is Facebook*/){
        if let token = AccessToken.current?.authenticationToken {
            return AWSTask(result: [AWSIdentityProviderFacebook:token])
        }
        return AWSTask(error:NSError(domain: "Facebook Login", code: -1 , userInfo:
 ["Facebook" : "No current Facebook access token"]))
    }else {
        return super.logins()
    }
}
```

When you transition from unauthenticated to authenticated, you should call `credentialsProvider.clearCredentials()` to force the SDK to get new authenticated credentials. When you switch between two authenticated providers and you aren't trying to link the two providers (i.e. you are not providing tokens for multiple providers in your logins dictionary), you should call `credentialsProvider.clearKeychain()`. This will clear both the credentials and identity and force the SDK to get new ones.

Unity

Your application might require supporting unauthenticated identities or authenticated identities using public providers (Login with Amazon, Facebook or Google) along with developer authenticated identities. The essential difference between developer authenticated identities and other identities (unauthenticated identities and authenticated identities using public provider) is the way the identityId and token are obtained. For other identities the mobile application will interact directly with Amazon Cognito instead of contacting your authentication system. So the mobile application should be able to support two distinct flows depending on the choice made by the app user. For this you will have to make some changes to the custom identity provider.

The recommended way to do it in Unity is to extend your identity provider from AmazonCognitoEnhancedIdentityProvide instead of AbstractCognitoIdentityProvider, and call the parent RefreshAsync method instead of your own in case the user is not authenticated with your own backend. If the user is authenticated, you can use the same flow explained before.

Xamarin

Your application might require supporting unauthenticated identities or authenticated identities using public providers (Login with Amazon, Facebook or Google) along with developer authenticated identities. The essential difference between developer authenticated identities and other identities (unauthenticated identities and authenticated identities using public provider) is the way the identityId and token are obtained. For other identities the mobile application will interact directly with Amazon Cognito instead of contacting your authentication system. So the mobile application should be able to support two distinct flows depending on the choice made by the app user. For this you will have to make some changes to the custom identity provider.

Switching Identities

Android

Users can begin their life in an application as unauthenticated guests. Eventually they may decide to log in using one of the supported identity providers. Amazon Cognito will ensure that an old identity retains the same unique identifier as the new one, and the profile data will be merged automatically.

Your application is informed of a profile merge through the `IdentityChangedListener` interface. Implement the `identityChanged` method in the interface to receive these messages:

```
@override
public void identityChanged(String oldIdentityId, String newIdentityId) {
    // handle the change
}
```

iOS - Objective-C

Users can begin their life in an application as unauthenticated guests. Eventually they may decide to log in using one of the supported identity providers. Amazon Cognito will ensure that an old identity retains the same unique identifier as the new one, and the profile data will be merged automatically.

`NSNotificationCenter` informs your application of a profile merge:

```
[[NSNotificationCenter defaultCenter] addObserver:self
                                      selector:@selector(identityIdDidChange:)
                                      name:AWSCognitoIdentityIdChangedNotification
                                      object:nil];

-(void)identityDidChange:(NSNotification*)notification {
    NSDictionary *userInfo = notification.userInfo;
    NSLog(@"identity changed from %@ to %@",
        [userInfo objectForKey:AWSCognitoNotificationPreviousId],
        [userInfo objectForKey:AWSCognitoNotificationNewId]);
}
```

iOS - Swift

Users can begin their life in an application as unauthenticated guests. Eventually they may decide to log in using one of the supported identity providers. Amazon Cognito will ensure that an old identity retains the same unique identifier as the new one, and the profile data will be merged automatically.

`NSNotificationCenter` informs your application of a profile merge:

```
[NSNotificationCenter.defaultCenter().addObserver(observer: self
   selector:"identityDidChange"
   name:AWSCognitoIdentityIdChangedNotification
   object:nil)

func identityDidChange(notification: NSNotification!) {
  if let userInfo = notification.userInfo as? [String: AnyObject] {
    print("identity changed from: \(userInfo[AWSCognitoNotificationPreviousId])
    to: \(userInfo[AWSCognitoNotificationNewId])")
  }
}
```

JavaScript

Users can begin their life in an application as unauthenticated guests. Eventually they may decide to log in using one of the supported identity providers. Amazon Cognito will ensure that an old identity retains the same unique identifier as the new one, and the profile data will be merged automatically.

Unity

Users can begin their life in an application as unauthenticated guests. Eventually they may decide to log in using one of the supported identity providers. Amazon Cognito will ensure that an old identity retains the same unique identifier as the new one, and the profile data will be merged automatically.

You can subscribe to the `IdentityChangedEvent` to be notified of profile merges:

```
credentialsProvider.IdentityChangedEvent += delegate(object sender,
 CognitoAWSCredentials.IdentityChangedArgs e)
```

```
{
    // handle the change
    Debug.log("Identity changed from " + e.OldIdentityId + " to " + e.NewIdentityId);
};
```

Xamarin

Users can begin their life in an application as unauthenticated guests. Eventually they may decide to log in using one of the supported identity providers. Amazon Cognito will ensure that an old identity retains the same unique identifier as the new one, and the profile data will be merged automatically.

```
credentialsProvider.IdentityChangedEvent += delegate(object sender,
 CognitoAWSCredentials.IdentityChangedArgs e){
    // handle the change
    Console.WriteLine("Identity changed from " + e.OldIdentityId + " to " +
 e.NewIdentityId);
};
```

Amazon Cognito Sync

Amazon Cognito Sync is an AWS service and client library that enables cross-device syncing of application-related user data. You can use it to synchronize user profile data across mobile devices and the web without requiring your own backend. The client libraries cache data locally so your app can read and write data regardless of device connectivity status. When the device is online, you can synchronize data, and if you set up push sync, notify other devices immediately that an update is available.

For information about Amazon Cognito Identity region availability, see AWS Service Region Availability.

To learn more about Amazon Cognito Sync, see the following topics.

Topics

- Getting Started with Amazon Cognito Sync (p. 190)
- Synchronizing Data (p. 191)
- Handling Callbacks (p. 198)
- Push Sync (p. 209)
- Amazon Cognito Streams (p. 215)
- Amazon Cognito Events (p. 217)

Getting Started with Amazon Cognito Sync

Amazon Cognito Sync is an AWS service and client library that enable cross-device syncing of application-related user data. You can use it to synchronize user profile data across mobile devices and web applications. The client libraries cache data locally so your app can read and write data regardless of device connectivity status. When the device is online, you can synchronize data, and if you set up push sync, notify other devices immediately that an update is available.

Sign Up for an AWS Account

To use Amazon Cognito Sync, you need an AWS account. If you don't already have one, use the following procedure to sign up:

To sign up for an AWS account

1. Open https://aws.amazon.com/, and then choose **Create an AWS Account**.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

Set Up an Identity Pool in Amazon Cognito

Amazon Cognito Sync requires an Amazon Cognito identity pool to provide user identities. Thus you need to first set up an identity pool before using Amazon Cognito Sync. Follow the Getting Started with Amazon Cognito Federated Identities (p. 134) guide to create an identity pool and install the SDK.

Store and Sync Data

Once you have set up your identity pool and installed the SDK, you can start storing and syncing data between devices. See Synchronizing Data (p. 191) for more information.

Synchronizing Data

Amazon Cognito lets you save end user data in datasets containing key-value pairs. This data is associated with an Amazon Cognito identity, so that it can be accessed across logins and devices. To sync this data between the Amazon Cognito service and an end user's devices, invoke the synchronize method. Each dataset can have a maximum size of 1 MB. You can associate up to 20 datasets with an identity.

The Amazon Cognito Sync client creates a local cache for the identity data. Your app talks to this local cache when it reads and writes keys. This guarantees that all of your changes made on the device are immediately available on the device, even when you are offline. When the synchronize method is called, changes from the service are pulled to the device, and any local changes are pushed to the service. At this point the changes are available to other devices to synchronize.

Initializing the Amazon Cognito Sync Client

To initialize the Amazon Cognito Sync client, you first need to create a credentials provider. The credentials provider acquires temporary AWS credentials to enable your app to access your AWS resources. You'll also need to import the required header files. Use the following steps to initialize the Amazon Cognito Sync client.

Android

1. Create a credentials provider, following the instructions in Getting Credentials (p. 148).
2. Import the Amazon Cognito package: `import com.amazonaws.mobileconnectors.cognito.*;`
3. Initialize Amazon Cognito Sync, passing in the Android app context, the identity pool ID, an AWS region, and an initialized Amazon Cognito credentials provider:

```
CognitoSyncManager client = new CognitoSyncManager(
    getApplicationContext(),
    Regions.YOUR_REGION,
    credentialsProvider);
```

iOS - Objective-C

1. Create a credentials provider, following the instructions in Getting Credentials (p. 148).

2. Import `AWSCore` and `Cognito`, and initialize `AWSCognito`:

```
#import <AWSiOSSDKv2/AWSCore.h>
#import <AWSCognitoSync/Cognito.h>

AWSCognito *syncClient = [AWSCognito defaultCognito];
```

3. If you're using CocoaPods, replace `<AWSiOSSDKv2/AWSCore.h>` with `AWSCore.h` and follow the same syntax for the Amazon Cognito import.

iOS - Swift

1. Create a credentials provider, following the instructions in Getting Credentials (p. 148).
2. Import and initialize `AWSCognito`:

```
import AWSCognito
let syncClient = AWSCognito.default()!
```

JavaScript

1. Download the Amazon Cognito Sync Manager for JavaScript.
2. Include the Sync Manager library in your project.
3. Create a credentials provider, following the instructions in Getting Credentials (p. 148).
4. Initialize the Sync Manager:

```
var syncManager = new AWS.CognitoSyncManager();
```

Unity

1. You will need to first create an instance of `CognitoAWSCredentials`, following the instructions in Getting Credentials (p. 148).
2. Create an instance of `CognitoSyncManager`, passing the `CognitoAwsCredentials` object and a `AmazonCognitoSyncConfig` with, at least, the region set:

```
AmazonCognitoSyncConfig clientConfig = new AmazonCognitoSyncConfig { RegionEndpoint =
 REGION };
CognitoSyncManager syncManager = new CognitoSyncManager(credentials, clientConfig);
```

Xamarin

1. You will need to first create an instance of `CognitoAWSCredentials`, following the instructions in Getting Credentials (p. 148).
2. Create an instance of `CognitoSyncManager`, passing the `CognitoAwsCredentials` object and a `AmazonCognitoSyncConfig` with, at least, the region set:

```
AmazonCognitoSyncConfig clientConfig = new AmazonCognitoSyncConfig { RegionEndpoint =
 REGION };
CognitoSyncManager syncManager = new CognitoSyncManager(credentials, clientConfig);
```

Understanding Datasets

With Amazon Cognito, end user profile data is organized into datasets. Each dataset can contain up to 1MB of data in the form of key-value pairs. A dataset is the most granular entity on which you can perform a sync operation. Read and write operations performed on a dataset only affect the local store until the synchronize method is invoked. A dataset is identified by a unique string. You can create a new dataset or open an existing one as shown in the following.

Android

```
Dataset dataset = client.openOrCreateDataset("datasetname");
```

To delete a dataset, first call the method to remove it from local storage, then call the `synchronize` method to delete the dataset from Amazon Cognito:

```
dataset.delete();
dataset.synchronize(syncCallback);
```

iOS - Objective-C

```
AWSCognitoDataset *dataset = [syncClient openOrCreateDataset:@"myDataSet"];
```

To delete a dataset, first call the method to remove it from local storage, then call the `synchronize` method to delete the dataset from Amazon Cognito:

```
[dataset clear];
[dataset synchronize];
```

iOS - Swift

```
let dataset = syncClient.openOrCreateDataset("myDataSet")!
```

To delete a dataset, first call the method to remove it from local storage, then call the `synchronize` method to delete the dataset from Amazon Cognito:

```
dataset.clear()
dataset.synchronize()
```

JavaScript

```
syncManager.openOrCreateDataset('myDatasetName', function(err, dataset) {
   // ...
});
```

Unity

```
string myValue = dataset.Get("myKey");
dataset.Put("myKey", "newValue");
```

You can use `Remove` to delete a key from a dataset:

```
dataset.Remove("myKey");
```

Xamarin

```
Dataset dataset = syncManager.OpenOrCreateDataset("myDatasetName");
```

To delete a dataset, first call the method to remove it from local storage, then call the `synchronize` method to delete the dataset from Amazon Cognito:

```
dataset.Delete();
dataset.SynchronizeAsync();
```

Reading and Writing Data in Datasets

Amazon Cognito datasets function as dictionaries, with values accessible by key. The keys and values of a dataset can be read, added, or modified just as if the dataset were a dictionary. The following shows an example.

Note that values written to a dataset only affect the local cached copy of the data until you call the synchronize method.

Android

```
String value = dataset.get("myKey");
dataset.put("myKey", "my value");
```

iOS - Objective-C

```
[dataset setString:@"my value" forKey:@"myKey"];
NSString *value = [dataset stringForKey:@"myKey"];
```

iOS - Swift

```
dataset.setString("my value", forKey:"myKey")
let value = dataset.stringForKey("myKey")
```

JavaScript

```
dataset.get('myKey', function(err, value) {
  console.log('myRecord: ' + value);
});

dataset.put('newKey', 'newValue', function(err, record) {
  console.log(record);
});

dataset.remove('oldKey', function(err, record) {
  console.log(success);
});
```

Unity

```
string myValue = dataset.Get("myKey");
dataset.Put("myKey", "newValue");
```

Xamarin

```
//obtain a value
string myValue = dataset.Get("myKey");

// Create a record in a dataset and synchronize with the server
dataset.OnSyncSuccess += SyncSuccessCallback;
dataset.Put("myKey", "myValue");
dataset.SynchronizeAsync();

void SyncSuccessCallback(object sender, SyncSuccessEventArgs e) {
  // Your handler code here
}
```

Android

You can use the `remove` method to remove keys from a dataset:

```
dataset.remove("myKey");
```

iOS - Objective-C

You can use `removeObjectForKey` to delete a key from a dataset:

```
[dataset removeObjectForKey:@"myKey"];
```

iOS - Swift

You can use `removeObjectForKey` to delete a key from a dataset:

```
dataset.removeObjectForKey("myKey")
```

Unity

You can use `Remove` to delete a key from a dataset:

```
dataset.Remove("myKey");
```

Xamarin

You can use `Remove` to delete a key from a dataset:

```
dataset.Remove("myKey");
```

Synchronizing Local Data with the Sync Store

Android

The `synchronize` method compares local cached data to the data stored in the Amazon Cognito Sync store. Remote changes are pulled from the Amazon Cognito Sync store; conflict resolution is invoked if any conflicts occur; and updated values on the device are pushed to the service. To synchronize a dataset, call its `synchronize` method:

```
dataset.synchronize(syncCallback);
```

The `synchronize` method receives an implementation of the `SyncCallback` interface, discussed below.

The `synchronizeOnConnectivity()` method attempts to synchronize when connectivity is available. If connectivity is immediately available, `synchronizeOnConnectivity()` behaves like `synchronize()`. Otherwise it monitors for connectivity changes and performs a sync once connectivity is available. If `synchronizeOnConnectivity()`is called multiple times, only the last synchronize request is kept, and only the last callback will fire. If either the dataset or the callback is garbage-collected, this method won't perform a sync, and the callback won't fire.

To learn more about dataset synchronization and the different callbacks, see Handling Callbacks (p. 198).

iOS - Objective-C

The `synchronize` method compares local cached data to the data stored in the Amazon Cognito Sync store. Remote changes are pulled from the Amazon Cognito Sync store; conflict resolution is invoked if any conflicts occur; and updated values on the device are pushed to the service. To synchronize a dataset, call its `synchronize` method:

The `synchronize` method is asynchronous and returns an `AWSTask` object to handle the response:

```
[[dataset synchronize] continueWithBlock:^id(AWSTask *task) {
    if (task.isCancelled) {
        // Task cancelled.
    } else if (task.error) {
        // Error while executing task.
    } else {
        // Task succeeded. The data was saved in the sync store.
    }
    return nil;
}];
```

The `synchronizeOnConnectivity` method attempts to synchronize when the device has connectivity. First, `synchronizeOnConnectivity` checks for connectivity and, if the device is online, immediately invokes synchronize and returns the `AWSTask` object associated with the attempt.

If the device is offline, `synchronizeOnConnectivity` 1) schedules a synchronize for the next time the device comes online and 2) returns an `AWSTask` with a nil result. The scheduled synchronize is only valid for the lifecycle of the dataset object. The data will not be synchronized if the app is exited before connectivity is regained. If you want to be notified when events occur during the scheduled synchronize, you must add observers of the notifications found in `AWSCognito`.

To learn more about dataset synchronization and the different callbacks, see Handling Callbacks (p. 198).

iOS - Swift

The `synchronize` method compares local cached data to the data stored in the Amazon Cognito Sync store. Remote changes are pulled from the Amazon Cognito Sync store; conflict resolution is invoked if any conflicts occur; and updated values on the device are pushed to the service. To synchronize a dataset, call its `synchronize` method:

The `synchronize` method is asynchronous and returns an `AWSTask` object to handle the response:

```
dataset.synchronize().continueWith(block: { (task) -> AnyObject? in

            if task.isCancelled {
                // Task cancelled.
            } else if task.error != nil {
                // Error while executing task
            } else {
                // Task succeeded. The data was saved in the sync store.
            }
            return task
})
```

The `synchronizeOnConnectivity` method attempts to synchronize when the device has connectivity. First, `synchronizeOnConnectivity` checks for connectivity and, if the device is online, immediately invokes `synchronize` and returns the `AWSTask` object associated with the attempt.

If the device is offline, `synchronizeOnConnectivity` 1) schedules a synchronize for the next time the device comes online and 2) returns an `AWSTask` object with a nil result. The scheduled synchronize is only valid for the lifecycle of the dataset object. The data will not be synchronized if the app is exited before connectivity is regained. If you want to be notified when events occur during the scheduled synchronize, you must add observers of the notifications found in `AWSCognito`.

To learn more about dataset synchronization and the different callbacks, see Handling Callbacks (p. 198).

JavaScript

The `synchronize` method compares local cached data to the data stored in the Amazon Cognito Sync store. Remote changes are pulled from the Amazon Cognito Sync store; conflict resolution is invoked if any conflicts occur; and updated values on the device are pushed to the service. To synchronize a dataset, call its `synchronize` method:

```
dataset.synchronize();
```

To learn more about dataset synchronization and the different callbacks, see Handling Callbacks (p. 198).

Unity

The synchronize method compares local cached data to the data stored in the Amazon Cognito Sync store. Remote changes are pulled from the Amazon Cognito Sync store; conflict resolution is invoked if any conflicts occur; and updated values on the device are pushed to the service. To synchronize a dataset, call its `synchronize` method:

```
dataset.Synchronize();
```

Synchronize will run asynchronously and will end up calling one of the several callbacks you can specify in the Dataset.

To learn more about dataset synchronization and the different callbacks, see Handling Callbacks (p. 198).

Xamarin

The `synchronize` method compares local cached data to the data stored in the Amazon Cognito Sync store. Remote changes are pulled from the Amazon Cognito Sync store; conflict resolution is invoked if any conflicts occur; and updated values on the device are pushed to the service. To synchronize a dataset, call its `synchronize` method:

```
dataset.SynchronizeAsync();
```

To learn more about dataset synchronization and the different callbacks, see Handling Callbacks (p. 198).

Handling Callbacks

This section describes how to handle callbacks.

Android

SyncCallback Interface

By implementing the `SyncCallback` interface, you can receive notifications on your app about dataset synchronization. Your app can then make active decisions about deleting local data, merging unauthenticated and authenticated profiles, and resolving sync conflicts. You should implement the following methods, which are required by the interface:

- `onSuccess()`
- `onFailure()`
- `onConflict()`
- `onDatasetDeleted()`
- `onDatasetsMerged()`

Note that, if you don't want to specify all the callbacks, you can also use the class `DefaultSyncCallback` which provides default, empty implementations for all of them.

onSuccess

The `onSuccess()` callback is triggered when a dataset is successfully downloaded from the sync store.

```
@Override
public void onSuccess(Dataset dataset, List<Record> newRecords) {
}
```

onFailure

onFailure() is called if an exception occurs during synchronization.

```
@Override
public void onFailure(DataStorageException dse) {
}
```

onConflict

Conflicts may arise if the same key has been modified on the local store and in the sync store. The `onConflict()` method handles conflict resolution. If you don't implement this method, the Amazon Cognito Sync client defaults to using the most recent change.

```
@Override
public boolean onConflict(Dataset dataset, final List<SyncConflict> conflicts) {
    List<Record> resolvedRecords = new ArrayList<Record>();
    for (SyncConflict conflict : conflicts) {
        /* resolved by taking remote records */
        resolvedRecords.add(conflict.resolveWithRemoteRecord());

        /* alternately take the local records */
        // resolvedRecords.add(conflict.resolveWithLocalRecord());

        /* or customer logic, say concatenate strings */
        // String newValue = conflict.getRemoteRecord().getValue()
        //     + conflict.getLocalRecord().getValue();
        // resolvedRecords.add(conflict.resolveWithValue(newValue);
    }
    dataset.resolve(resolvedRecords);

    // return true so that synchronize() is retried after conflicts are resolved
    return true;
}
```

onDatasetDeleted

When a dataset is deleted, the Amazon Cognito client uses the `SyncCallback` interface to confirm whether the local cached copy of the dataset should be deleted too. Implement the `onDatasetDeleted()` method to tell the client SDK what to do with the local data.

```
@Override
public boolean onDatasetDeleted(Dataset dataset, String datasetName) {
    // return true to delete the local copy of the dataset
    return true;
}
```

onDatasetMerged

When two previously unconnected identities are linked together, all of their datasets are merged. Applications are notified of the merge through the `onDatasetsMerged()` method:

```
@Override
public boolean onDatasetsMerged(Dataset dataset, List<String> datasetNames) {
    // return false to handle Dataset merge outside the synchronization callback
    return false;
}
```

iOS - Objective-C

Sync Notifications

The Amazon Cognito client will emit a number of `NSNotification` events during a synchronize call. You can register to monitor these notifications via the standard `NSNotificationCenter`:

```
[NSNotificationCenter defaultCenter]
  addObserver:self
  selector:@selector(myNotificationHandler:)
  name:NOTIFICATION_TYPE
  object:nil];
```

Amazon Cognito supports five notification types, listed below.

AWSCognitoDidStartSynchronizeNotification

Called when a synchronize operation is starting. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized.

AWSCognitoDidEndSynchronizeNotification

Called when a synchronize operation completes (successfully or otherwise). The `userInfo` will contain the key dataset which is the name of the dataset being synchronized.

AWSCognitoDidFailToSynchronizeNotification

Called when a synchronize operation fails. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized and the key error which will contain the error that caused the failure.

AWSCognitoDidChangeRemoteValueNotification

Called when local changes are successfully pushed to Amazon Cognito. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized and the key keys which will contain an NSArray of record keys that were pushed.

AWSCognitoDidChangeLocalValueFromRemoteNotification

Called when a local value changes due to a synchronize operation. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized and the key keys which will contain an NSArray of record keys that changed.

Conflict Resolution Handler

During a sync operation, conflicts may arise if the same key has been modified on the local store and in the sync store. If you haven't set a conflict resolution handler, Amazon Cognito defaults to choosing the most recent update.

By implementing and assigning an AWSCognitoRecordConflictHandler you can alter the default conflict resolution. The AWSCognitoConflict input parameter conflict contains an AWSCognitoRecord object for both the local cached data and for the conflicting record in the sync store. Using the AWSCognitoConflict you can resolve the conflict with the local record: [conflict resolveWithLocalRecord], the remote record: [conflict resolveWithRemoteRecord] or a brand new value: [conflict resolveWithValue:value]. Returning nil from this method prevents synchronization from continuing and the conflicts will be presented again the next time the sync process starts.

You can set the conflict resolution handler at the client level:

```
client.conflictHandler = ^AWSCognitoResolvedConflict* (NSString *datasetName,
 AWSCognitoConflict *conflict) {
    // always choose local changes
    return [conflict resolveWithLocalRecord];
};
```

Or at the dataset level:

```
dataset.conflictHandler = ^AWSCognitoResolvedConflict* (NSString *datasetName,
 AWSCognitoConflict *conflict) {
    // override and always choose remote changes
    return [conflict resolveWithRemoteRecord];
};
```

Dataset Deleted Handler

When a dataset is deleted, the Amazon Cognito client uses the `AWSCognitoDatasetDeletedHandler` to confirm whether the local cached copy of the dataset should be deleted too. If no `AWSCognitoDatasetDeletedHandler` is implemented, the local data will be purged automatically. Implement an `AWSCognitoDatasetDeletedHandler` if you wish to keep a copy of the local data before wiping, or to keep the local data.

You can set the dataset deleted handler at the client level:

```
client.datasetDeletedHandler = ^BOOL (NSString *datasetName) {
    // make a backup of the data if you choose
    ...
    // delete the local data (default behavior)
    return YES;
};
```

Or at the dataset level:

```
dataset.datasetDeletedHandler = ^BOOL (NSString *datasetName) {
    // override default and keep the local data
    return NO;
};
```

Dataset Merge Handler

When two previously unconnected identities are linked together, all of their datasets are merged. Applications are notified of the merge through the `DatasetMergeHandler`. The handler will receive the name of the root dataset as well as an array of dataset names that are marked as merges of the root dataset.

If no `DatasetMergeHandler` is implemented, these datasets will be ignored, but will continue to use up space in the identity's 20 maximum total datasets.

You can set the dataset merge handler at the client level:

```
client.datasetMergedHandler = ^(NSString *datasetName, NSArray *datasets) {
    // Blindly delete the datasets
    for (NSString *name in datasets) {
        AWSCognitoDataset *merged = [[AWSCognito defaultCognito] openOrCreateDataset:name];
       [merged clear];
       [merged synchronize];
    }
};
```

Or at the dataset level:

```
dataset.datasetMergedHandler = ^(NSString *datasetName, NSArray *datasets) {
    // Blindly delete the datasets
    for (NSString *name in datasets) {
        AWSCognitoDataset *merged = [[AWSCognito defaultCognito] openOrCreateDataset:name];
        // do something with the data if it differs from existing dataset
        ...
        // now delete it
        [merged clear];
        [merged synchronize];
    }
};
```

iOS - Swift

Sync Notifications

The Amazon Cognito client will emit a number of `NSNotification` events during a synchronize call. You can register to monitor these notifications via the standard `NSNotificationCenter`:

```
NSNotificationCenter.defaultCenter().addObserver(observer: self,
   selector: "myNotificationHandler",
   name:NOTIFICATION_TYPE,
   object:nil)
```

Amazon Cognito supports five notification types, listed below.

AWSCognitoDidStartSynchronizeNotification

Called when a synchronize operation is starting. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized.

AWSCognitoDidEndSynchronizeNotification

Called when a synchronize operation completes (successfully or otherwise). The `userInfo` will contain the key dataset which is the name of the dataset being synchronized.

AWSCognitoDidFailToSynchronizeNotification

Called when a synchronize operation fails. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized and the key error which will contain the error that caused the failure.

AWSCognitoDidChangeRemoteValueNotification

Called when local changes are successfully pushed to Amazon Cognito. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized and the key keys which will contain an NSArray of record keys that were pushed.

AWSCognitoDidChangeLocalValueFromRemoteNotification

Called when a local value changes due to a synchronize operation. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized and the key keys which will contain an NSArray of record keys that changed.

Conflict Resolution Handler

During a sync operation, conflicts may arise if the same key has been modified on the local store and in the sync store. If you haven't set a conflict resolution handler, Amazon Cognito defaults to choosing the most recent update.

By implementing and assigning an `AWSCognitoRecordConflictHandler` you can alter the default conflict resolution. The `AWSCognitoConflict` input parameter conflict contains an `AWSCognitoRecord` object for both the local cached data and for the conflicting record in the sync store. Using the `AWSCognitoConflict` you can resolve the conflict with the local record: [conflict resolveWithLocalRecord], the remote record: [conflict resolveWithRemoteRecord] or a brand new value: [conflict resolveWithValue:value]. Returning nil from this method prevents synchronization from continuing and the conflicts will be presented again the next time the sync process starts.

You can set the conflict resolution handler at the client level:

```
client.conflictHandler = {
    (datasetName: String?, conflict: AWSCognitoConflict?) -> AWSCognitoResolvedConflict? in
    return conflict.resolveWithLocalRecord()
}
```

Or at the dataset level:

```
dataset.conflictHandler = {
    (datasetName: String?, conflict: AWSCognitoConflict?) -> AWSCognitoResolvedConflict? in
    return conflict.resolveWithLocalRecord()
}
```

Dataset Deleted Handler

When a dataset is deleted, the Amazon Cognito client uses the `AWSCognitoDatasetDeletedHandler` to confirm whether the local cached copy of the dataset should be deleted too. If no `AWSCognitoDatasetDeletedHandler` is implemented, the local data will be purged automatically. Implement an `AWSCognitoDatasetDeletedHandler` if you wish to keep a copy of the local data before wiping, or to keep the local data.

You can set the dataset deleted handler at the client level:

```
client.datasetDeletedHandler = {
      (datasetName: String!) -> Bool in
      // make a backup of the data if you choose
      ...
      // delete the local data (default behaviour)
      return true
}
```

Or at the dataset level:

```
dataset.datasetDeletedHandler = {
      (datasetName: String!) -> Bool in
      // make a backup of the data if you choose
      ...
      // delete the local data (default behaviour)
      return true
}
```

Dataset Merge Handler

When two previously unconnected identities are linked together, all of their datasets are merged. Applications are notified of the merge through the `DatasetMergeHandler`. The handler will receive the name of the root dataset as well as an array of dataset names that are marked as merges of the root dataset.

If no `DatasetMergeHandler` is implemented, these datasets will be ignored, but will continue to use up space in the identity's 20 maximum total datasets.

You can set the dataset merge handler at the client level:

```
client.datasetMergedHandler = {
    (datasetName: String!, datasets: [AnyObject]!) -> Void in
    for nameObject in datasets {
        if let name = nameObject as? String {
            let merged = AWSCognito.defaultCognito().openOrCreateDataset(name)
            merged.clear()
            merged.synchronize()
        }
    }
}
```

Or at the dataset level:

```
dataset.datasetMergedHandler = {
    (datasetName: String!, datasets: [AnyObject]!) -> Void in
    for nameObject in datasets {
        if let name = nameObject as? String {
            let merged = AWSCognito.defaultCognito().openOrCreateDataset(name)
            // do something with the data if it differs from existing dataset
              ...
              // now delete it
            merged.clear()
            merged.synchronize()
        }
    }
}
```

JavaScript

Synchronization Callbacks

When performing a synchronize() on a dataset, you can optionally specify callbacks to handle each of the following states:

```
dataset.synchronize({

   onSuccess: function(dataset, newRecords) {
      //...
   },

   onFailure: function(err) {
      //...
   },

   onConflict: function(dataset, conflicts, callback) {
      //...
   },

   onDatasetDeleted: function(dataset, datasetName, callback) {
      //...
   },

  onDatasetMerged: function(dataset, datasetNames, callback) {
      //...
  }

});
```

onSuccess()

The `onSuccess()` callback is triggered when a dataset is successfully updated from the sync store. If you do not define a callback, the synchronization will succeed silently.

```
onSuccess: function(dataset, newRecords) {
   console.log('Successfully synchronized ' + newRecords.length + ' new records.');
}
```

onFailure()

`onFailure()` is called if an exception occurs during synchronization. If you do not define a callback, the synchronization will fail silently.

```
onFailure: function(err) {
   console.log('Synchronization failed.');
   console.log(err);
}
```

onConflict()

Conflicts may arise if the same key has been modified on the local store and in the sync store. The `onConflict()` method handles conflict resolution. If you don't implement this method, the synchronization will be aborted when there is a conflict.

```
onConflict: function(dataset, conflicts, callback) {

   var resolved = [];

   for (var i=0; i<conflicts.length; i++) {

      // Take remote version.
      resolved.push(conflicts[i].resolveWithRemoteRecord());

      // Or... take local version.
      // resolved.push(conflicts[i].resolveWithLocalRecord());

      // Or... use custom logic.
      // var newValue = conflicts[i].getRemoteRecord().getValue() +
 conflicts[i].getLocalRecord().getValue();
      // resolved.push(conflicts[i].resovleWithValue(newValue);

   }

   dataset.resolve(resolved, function() {
      return callback(true);
   });

   // Or... callback false to stop the synchronization process.
   // return callback(false);

}
```

onDatasetDeleted()

When a dataset is deleted, the Amazon Cognito client uses the `onDatasetDeleted()` callback to decide whether the local cached copy of the dataset should be deleted too. By default, the dataset will not be deleted.

```
onDatasetDeleted: function(dataset, datasetName, callback) {

   // Return true to delete the local copy of the dataset.
   // Return false to handle deleted datasets outside the synchronization callback.

   return callback(true);

}
```

onDatasetMerged()

When two previously unconnected identities are linked together, all of their datasets are merged. Applications are notified of the merge through the `onDatasetsMerged()` callback.

```
onDatasetMerged: function(dataset, datasetNames, callback) {

   // Return true to continue the synchronization process.
   // Return false to handle dataset merges outside the synchronization callback.

   return callback(false);

}
```

Unity

After you open or create a dataset, you can set different callbacks to it that will be triggered when you use the Synchronize method. This is the way to register your callbacks to them:

```
dataset.OnSyncSuccess += this.HandleSyncSuccess;
dataset.OnSyncFailure += this.HandleSyncFailure;
dataset.OnSyncConflict = this.HandleSyncConflict;
dataset.OnDatasetMerged = this.HandleDatasetMerged;
dataset.OnDatasetDeleted = this.HandleDatasetDeleted;
```

Note that `SyncSuccess` and `SyncFailure` use += instead of = so you can subscribe more than one callback to them.

OnSyncSuccess

The `OnSyncSuccess` callback is triggered when a dataset is successfully updated from the cloud. If you do not define a callback, the synchronization will succeed silently.

```
private void HandleSyncSuccess(object sender, SyncSuccessEvent e)
{
    // Continue with your game flow, display the loaded data, etc.
}
```

OnSyncFailure

`OnSyncFailure` is called if an exception occurs during synchronization. If you do not define a callback, the synchronization will fail silently.

```
private void HandleSyncFailure(object sender, SyncFailureEvent e)
{
    Dataset dataset = sender as Dataset;
    if (dataset.Metadata != null) {
        Debug.Log("Sync failed for dataset : " + dataset.Metadata.DatasetName);
    } else {
        Debug.Log("Sync failed");
    }
    // Handle the error
    Debug.LogException(e.Exception);
}
```

OnSyncConflict

Conflicts may arise if the same key has been modified on the local store and in the sync store. The `OnSyncConflict` callback handles conflict resolution. If you don't implement this method, the synchronization will be aborted when there is a conflict.

```
private bool HandleSyncConflict(Dataset dataset, List < SyncConflict > conflicts)
{
  if (dataset.Metadata != null) {
    Debug.LogWarning("Sync conflict " + dataset.Metadata.DatasetName);
  } else {
    Debug.LogWarning("Sync conflict");
  }
  List < Amazon.CognitoSync.SyncManager.Record > resolvedRecords = new List <
 Amazon.CognitoSync.SyncManager.Record > ();
  foreach(SyncConflict conflictRecord in conflicts) {
    // SyncManager provides the following default conflict resolution methods:
    //      ResolveWithRemoteRecord - overwrites the local with remote records
    //      ResolveWithLocalRecord - overwrites the remote with local records
    //      ResolveWithValue - to implement your own logic
    resolvedRecords.Add(conflictRecord.ResolveWithRemoteRecord());
  }
  // resolves the conflicts in local storage
  dataset.Resolve(resolvedRecords);
  // on return true the synchronize operation continues where it left,
  //      returning false cancels the synchronize operation
  return true;
}
```

OnDatasetDeleted

When a dataset is deleted, the Amazon Cognito client uses the `OnDatasetDeleted` callback to decide whether the local cached copy of the dataset should be deleted too. By default, the dataset will not be deleted.

```
private bool HandleDatasetDeleted(Dataset dataset)
  {
      Debug.Log(dataset.Metadata.DatasetName + " Dataset has been deleted");
      // Do clean up if necessary
      // returning true informs the corresponding dataset can be purged in the local
 storage and return false retains the local dataset
      return true;
  }
```

OnDatasetMerged

When two previously unconnected identities are linked together, all of their datasets are merged. Applications are notified of the merge through the `OnDatasetsMerged` callback.

```
public bool HandleDatasetMerged(Dataset localDataset, List<string> mergedDatasetNames)
{
    foreach (string name in mergedDatasetNames)
    {
        Dataset mergedDataset = syncManager.OpenOrCreateDataset(name);
        //Lambda function to delete the dataset after fetching it
        EventHandler<SyncSuccessEvent> lambda;
        lambda = (object sender, SyncSuccessEvent e) => {
            ICollection<string> existingValues = localDataset.GetAll().Values;
            ICollection<string> newValues = mergedDataset.GetAll().Values;

            //Implement your merge logic here

            mergedDataset.Delete(); //Delete the dataset locally
            mergedDataset.OnSyncSuccess -= lambda; //We don't want this callback to be
 fired again
            mergedDataset.OnSyncSuccess += (object s2, SyncSuccessEvent e2) => {
                localDataset.Synchronize(); //Continue the sync operation that was
 interrupted by the merge
            };
            mergedDataset.Synchronize(); //Synchronize it as deleted, failing to do so will
 leave us in an inconsistent state
```

```
        };
        mergedDataset.OnSyncSuccess += lambda;
        mergedDataset.Synchronize(); //Asnchronously fetch the dataset
    }

    // returning true allows the Synchronize to continue and false stops it
    return false;
}
```

Xamarin

After you open or create a dataset, you can set different callbacks to it that will be triggered when you use the Synchronize method. This is the way to register your callbacks to them:

```
dataset.OnSyncSuccess += this.HandleSyncSuccess;
dataset.OnSyncFailure += this.HandleSyncFailure;
dataset.OnSyncConflict = this.HandleSyncConflict;
dataset.OnDatasetMerged = this.HandleDatasetMerged;
dataset.OnDatasetDeleted = this.HandleDatasetDeleted;
```

Note that `SyncSuccess` and `SyncFailure` use += instead of = so you can subscribe more than one callback to them.

OnSyncSuccess

The `OnSyncSuccess` callback is triggered when a dataset is successfully updated from the cloud. If you do not define a callback, the synchronization will succeed silently.

```
private void HandleSyncSuccess(object sender, SyncSuccessEventArgs e)
{
    // Continue with your game flow, display the loaded data, etc.
}
```

OnSyncFailure

`OnSyncFailure` is called if an exception occurs during synchronization. If you do not define a callback, the synchronization will fail silently.

```
private void HandleSyncFailure(object sender, SyncFailureEventArgs e)
{
    Dataset dataset = sender as Dataset;
    if (dataset.Metadata != null) {
        Console.WriteLine("Sync failed for dataset : " + dataset.Metadata.DatasetName);
    } else {
        Console.WriteLine("Sync failed");
    }
}
```

OnSyncConflict

Conflicts may arise if the same key has been modified on the local store and in the sync store. The `OnSyncConflict` callback handles conflict resolution. If you don't implement this method, the synchronization will be aborted when there is a conflict.

```
private bool HandleSyncConflict(Dataset dataset, List < SyncConflict > conflicts)
{
  if (dataset.Metadata != null) {
    Console.WriteLine("Sync conflict " + dataset.Metadata.DatasetName);
  } else {
    Console.WriteLine("Sync conflict");
  }
  List < Amazon.CognitoSync.SyncManager.Record > resolvedRecords = new List <
 Amazon.CognitoSync.SyncManager.Record > ();
  foreach(SyncConflict conflictRecord in conflicts) {
    // SyncManager provides the following default conflict resolution methods:
    //      ResolveWithRemoteRecord - overwrites the local with remote records
    //      ResolveWithLocalRecord - overwrites the remote with local records
    //      ResolveWithValue - to implement your own logic
    resolvedRecords.Add(conflictRecord.ResolveWithRemoteRecord());
  }
  // resolves the conflicts in local storage
  dataset.Resolve(resolvedRecords);
  // on return true the synchronize operation continues where it left,
  //      returning false cancels the synchronize operation
  return true;
}
```

OnDatasetDeleted

When a dataset is deleted, the Amazon Cognito client uses the `OnDatasetDeleted` callback to decide whether the local cached copy of the dataset should be deleted too. By default, the dataset will not be deleted.

```
private bool HandleDatasetDeleted(Dataset dataset)
{
    Console.WriteLine(dataset.Metadata.DatasetName + " Dataset has been deleted");
    // Do clean up if necessary
    // returning true informs the corresponding dataset can be purged in the local storage
 and return false retains the local dataset
    return true;
}
```

OnDatasetMerged

When two previously unconnected identities are linked together, all of their datasets are merged. Applications are notified of the merge through the `OnDatasetsMerged` callback.

```
public bool HandleDatasetMerged(Dataset localDataset, List<string> mergedDatasetNames)
{
    foreach (string name in mergedDatasetNames)
    {
        Dataset mergedDataset = syncManager.OpenOrCreateDataset(name);

            //Implement your merge logic here

        mergedDataset.OnSyncSuccess += lambda;
        mergedDataset.SynchronizeAsync(); //Asnchronously fetch the dataset
    }

    // returning true allows the Synchronize to continue and false stops it
    return false;
}
```

Push Sync

Amazon Cognito automatically tracks the association between identity and devices. Using the push synchronization, or push sync, feature, you can ensure that every instance of a given identity is notified when identity data changes. Push sync ensures that, whenever the sync store data changes for a particular identity, all devices associated with that identity receive a silent push notification informing them of the change.

> **Note**
> Push sync is not supported for JavaScript, Unity, or Xamarin.

Before you can use push sync, you must first set up your account for push sync and enable push sync in the Amazon Cognito console.

Create an Amazon Simple Notification Service (Amazon SNS) App

Create and configure an Amazon SNS app for your supported platforms, as described in the SNS Developer Guide.

Enable Push Sync in the Amazon Cognito console

You can enable push sync via the Amazon Cognito console. From the console home page:

1. Click the name of the identity pool for which you want to enable push sync. The **Dashboard** page for your identity pool appears.
2. In the top-right corner of the **Dashboard** page, click **Edit identity pool**. The **Edit identity pool** page appears.
3. Scroll down and click **Push synchronization** to expand it.
4. In the **Service role** dropdown menu, select the IAM role that grants Cognito permission to send an SNS notification. Click **Create role** to create or modify the roles associated with your identity pool in the AWS IAM Console.
5. Select a platform application, and then click **Save Changes**.
6. Grant SNS Access to Your Application

In the IAM console, configure your IAM roles to have full SNS access, or create a new role that trusts cognito-sync and has full SNS access. To learn more about IAM roles, see Roles (Delegation and Federation).

Use Push Sync in Your App: Android

Your application will need to import the Google Play services. You can download the latest version of the Google Play SDK via the Android SDK manager. Follow the Android documentation on implementing a GCM client to register your app and receive a registration ID from GCM. Once you have the registration ID, you need to register the device with Amazon Cognito, as shown in the snippet below:

```
String registrationId = "MY_GCM_REGISTRATION_ID";
try {
    client.registerDevice("GCM", registrationId);
} catch (RegistrationFailedException rfe) {
    Log.e(TAG, "Failed to register device for silent sync", rfe);
} catch (AmazonClientException ace) {
    Log.e(TAG, "An unknown error caused registration for silent sync to fail", ace);
}
```

You can now subscribe a device to receive updates from a particular dataset:

```
Dataset trackedDataset = client.openOrCreateDataset("myDataset");
if (client.isDeviceRegistered()) {
    try {
        trackedDataset.subscribe();
    } catch (SubscribeFailedException sfe) {
        Log.e(TAG, "Failed to subscribe to datasets", sfe);
    } catch (AmazonClientException ace) {
        Log.e(TAG, "An unknown error caused the subscription to fail", ace);
    }
}
```

To stop receiving push notifications from a dataset, simply call the unsubscribe method. To subscribe to all datasets (or a specific subset) in the `CognitoSyncManager` object, use `subscribeAll()`:

```
if (client.isDeviceRegistered()) {
    try {
        client.subscribeAll();
    } catch (SubscribeFailedException sfe) {
        Log.e(TAG, "Failed to subscribe to datasets", sfe);
    } catch (AmazonClientException ace) {
        Log.e(TAG, "An unknown error caused the subscription to fail", ace);
    }
}
```

In your implementation of the Android BroadcastReceiver object, you can check the latest version of the modified dataset and decide if your app needs to synchronize again:

```
@Override
public void onReceive(Context context, Intent intent) {

    PushSyncUpdate update = client.getPushSyncUpdate(intent);

    // The update has the source (cognito-sync here), identityId of the
    // user, identityPoolId in question, the non-local sync count of the
    // data set and the name of the dataset. All are accessible through
    // relevant getters.

    String source = update.getSource();
    String identityPoolId = update.getIdentityPoolId();
    String identityId = update.getIdentityId();
    String datasetName = update.getDatasetName;
    long syncCount = update.getSyncCount;

    Dataset dataset = client.openOrCreateDataset(datasetName);

    // need to access last sync count. If sync count is less or equal to
    // last sync count of the dataset, no sync is required.

    long lastSyncCount = dataset.getLastSyncCount();
    if (lastSyncCount < syncCount) {
        dataset.synchronize(new SyncCallback() {
            // ...
        });
    }

}
```

The following keys are available in the push notification payload:

- `source`: cognito-sync. This can serve as a differentiating factor between notifications.
- `identityPoolId`: The identity pool ID. This can be used for validation or additional information, though it's not integral from the receiver's point of view.
- `identityId`: The identity ID within the pool.

- `datasetName`: The name of the dataset that was updated. This is available for the sake of the openOrCreateDataset call.
- `syncCount`: The sync count for the remote dataset. You can use this as a way to make sure that the local dataset is out of date, and that the incoming synchronization is new.

Use Push Sync in Your App: iOS - Objective-C

To obtain a device token for your app, follow the Apple documentation on Registering for Remote Notifications. Once you've received the device token as an NSData object from APNs, you'll need to register the device with Amazon Cognito using the `registerDevice:` method of the sync client, as shown below:

```
AWSCognito *syncClient = [AWSCognito defaultCognito];
    [[syncClient registerDevice: devToken] continueWithBlock:^id(AWSTask *task) {
        if(task.error){
            NSLog(@"Unable to registerDevice: %@", task.error);
        } else {
            NSLog(@"Successfully registered device with id: %@", task.result);
        }
        return nil;
      }
    ];
```

In debug mode, your device will register with the APNs sandbox; in release mode, it will register with APNs. To receive updates from a particular dataset, use the `subscribe` method:

```
[[[syncClient openOrCreateDataset:@"MyDataset"] subscribe] continueWithBlock:^id(AWSTask
 *task) {
        if(task.error){
            NSLog(@"Unable to subscribe to dataset: %@", task.error);
        } else {
            NSLog(@"Successfully subscribed to dataset: %@", task.result);
        }
        return nil;
      }
    ];
```

To stop receiving push notifications from a dataset, simply call the `unsubscribe` method:

```
[[[syncClient openOrCreateDataset:@”MyDataset”] unsubscribe] continueWithBlock:^id(AWSTask
 *task) {
        if(task.error){
            NSLog(@"Unable to unsubscribe from dataset: %@", task.error);
        } else {
            NSLog(@"Successfully unsubscribed from dataset: %@", task.result);
        }
        return nil;
      }
    ];
```

To subscribe to all datasets in the `AWSCognito` object, call `subscribeAll`:

```
[[syncClient subscribeAll] continueWithBlock:^id(AWSTask *task) {
        if(task.error){
            NSLog(@"Unable to subscribe to all datasets: %@", task.error);
        } else {
            NSLog(@"Successfully subscribed to all datasets: %@", task.result);
        }
        return nil;
      }
    ];
```

Before calling `subscribeAll`, be sure to synchronize at least once on each dataset, so that the datasets exist on the server.

To react to push notifications, you need to implement the `didReceiveRemoteNotification` method in your app delegate:

```
- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary
 *)userInfo
    {
        [[NSNotificationCenter defaultCenter]
 postNotificationName:@"CognitoPushNotification" object:userInfo];
    }
```

If you post a notification using notification handler, you can then respond to the notification elsewhere in the application where you have a handle to your dataset. If you subscribe to the notification like this ...

```
[[NSNotificationCenter defaultCenter] addObserver:self
 selector:@selector(didReceivePushSync:)
    name: :@"CognitoPushNotification" object:nil];
```

...you can act on the notification like this:

```
- (void)didReceivePushSync:(NSNotification*)notification
    {
        NSDictionary * data = [(NSDictionary *)[notification object] objectForKey:@"data"];
        NSString * identityId = [data objectForKey:@"identityId"];
        NSString * datasetName = [data objectForKey:@"datasetName"];
        if([self.dataset.name isEqualToString:datasetName] && [self.identityId
 isEqualToString:identityId]){
            [[self.dataset synchronize] continueWithBlock:^id(AWSTask *task) {
                if(!task.error){
                    NSLog(@"Successfully synced dataset");
                }
                return nil;
            }];
        }
    }
```

The following keys are available in the push notification payload:

- `source`: cognito-sync. This can serve as a differentiating factor between notifications.
- `identityPoolId`: The identity pool ID. This can be used for validation or additional information, though it's not integral from the receiver's point of view.
- `identityId`: The identity ID within the pool.
- `datasetName`: The name of the dataset that was updated. This is available for the sake of the `openOrCreateDataset` call.
- `syncCount`: The sync count for the remote dataset. You can use this as a way to make sure that the local dataset is out of date, and that the incoming synchronization is new.

Use Push Sync in Your App: iOS - Swift

To obtain a device token for your app, follow the Apple documentation on Registering for Remote Notifications. Once you've received the device token as an NSData object from APNs, you'll need to register the device with Amazon Cognito using the registerDevice: method of the sync client, as shown below:

```
let syncClient = AWSCognito.default()
syncClient.registerDevice(devToken).continueWith(block: { (task: AWSTask!) -> AnyObject! in
    if (task.error != nil) {
        print("Unable to register device: " + task.error.localizedDescription)

    } else {
        print("Successfully registered device with id: \(task.result)")
    }
    return task
})
```

In debug mode, your device will register with the APNs sandbox; in release mode, it will register with APNs. To receive updates from a particular dataset, use the `subscribe` method:

```
syncClient.openOrCreateDataset("MyDataset").subscribe().continueWith(block: { (task:
 AWSTask!) -> AnyObject! in
  if (task.error != nil) {
      print("Unable to subscribe to dataset: " + task.error.localizedDescription)

  } else {
      print("Successfully subscribed to dataset: \(task.result)")
  }
  return task
})
```

To stop receiving push notifications from a dataset, call the `unsubscribe` method:

```
syncClient.openOrCreateDataset("MyDataset").unsubscribe().continueWith(block: { (task:
 AWSTask!) -> AnyObject! in
  if (task.error != nil) {
      print("Unable to unsubscribe to dataset: " + task.error.localizedDescription)

  } else {
      print("Successfully unsubscribed to dataset: \(task.result)")
  }
  return task
})
```

To subscribe to all datasets in the `AWSCognito` object, call `subscribeAll`:

```
syncClient.openOrCreateDataset("MyDataset").subscribeAll().continueWith(block: { (task:
 AWSTask!) -> AnyObject! in
  if (task.error != nil) {
      print("Unable to subscribe to all datasets: " + task.error.localizedDescription)

  } else {
      print("Successfully subscribed to all datasets: \(task.result)")
  }
  return task
})
```

Before calling `subscribeAll`, be sure to synchronize at least once on each dataset, so that the datasets exist on the server.

To react to push notifications, you need to implement the `didReceiveRemoteNotification` method in your app delegate:

```
func application(application: UIApplication, didReceiveRemoteNotification userInfo:
 [NSObject : AnyObject],
  fetchCompletionHandler completionHandler: (UIBackgroundFetchResult) -> Void) {

 NSNotificationCenter.defaultCenter().postNotificationName("CognitoPushNotification",
 object: userInfo)
})
```

If you post a notification using notification handler, you can then respond to the notification elsewhere in the application where you have a handle to your dataset. If you subscribe to the notification like this ...

```
NSNotificationCenter.defaultCenter().addObserver(observer:self,
   selector:"didReceivePushSync:",
   name:"CognitoPushNotification",
   object:nil)
```

...you can act on the notification like this:

```
func didReceivePushSync(notification: NSNotification) {
    if let data = (notification.object as! [String: AnyObject])["data"] as? [String:
 AnyObject] {
        let identityId = data["identityId"] as! String
        let datasetName = data["datasetName"] as! String

        if self.dataset.name == datasetName && self.identityId == identityId {
          dataset.synchronize().continueWithBlock {(task) -> AnyObject! in
              if task.error == nil {
                print("Successfully synced dataset")
              }
              return nil
          }
        }
    }
}
```

The following keys are available in the push notification payload:

- `source`: cognito-sync. This can serve as a differentiating factor between notifications.
- `identityPoolId`: The identity pool ID. This can be used for validation or additional information, though it's not integral from the receiver's point of view.
- `identityId`: The identity ID within the pool.
- `datasetName`: The name of the dataset that was updated. This is available for the sake of the `openOrCreateDataset` call.
- `syncCount`: The sync count for the remote dataset. You can use this as a way to make sure that the local dataset is out of date, and that the incoming synchronization is new.

Amazon Cognito Streams

Amazon Cognito Streams gives developers control and insight into their data stored in Amazon Cognito. Developers can now configure an Kinesis stream to receive events as data is updated and synchronized. Amazon Cognito can push each dataset change to an Kinesis stream you own in real time.

Using Amazon Cognito Streams, you can move all of your Sync data to Kinesis, which can then be streamed to a data warehouse tool such as Amazon Redshift for further analysis. To learn more about Kinesis, see Getting Started Using Amazon Kinesis.

Configuring Streams

You can set up Amazon Cognito Streams in the Amazon Cognito console. To enable Amazon Cognito Streams in the Amazon Cognito console, you need to select the Kinesis stream to publish to and an IAM role that grants Amazon Cognito permission to put events in the selected stream.

From the console home page:

1. Click the name of the identity pool for which you want to set up Amazon Cognito Streams. The **Dashboard** page for your identity pool appears.
2. In the top-right corner of the **Dashboard** page, click **Edit identity pool**. The Edit identity pool page appears.
3. Scroll down and click **Cognito Streams** to expand it.
4. In the **Stream name** dropdown menu, select the name of an existing Kinesis stream. Alternatively, click **Create stream** to create one, entering a stream name and the number of shards. To learn about shards and for help on estimating the number of shards needed for your stream, see the Kinesis Developer Guide.
5. In the **Publish role** dropdown menu, select the IAM role that grants Amazon Cognito permission to publish your stream. Click **Create role** to create or modify the roles associated with your identity pool in the AWS IAM Console.
6. In the **Stream status** dropdown menu, select **Enabled** to enable the stream updates. Click **Save Changes**.

After you've successfully configured Amazon Cognito streams, all subsequent updates to datasets in this identity pool will be sent to the stream.

Stream Contents

Each record sent to the stream represents a single synchronization. Here is an example of a record sent to the stream:

```
{
    "identityPoolId": "Pool Id",
    "identityId": "Identity Id",
    "dataSetName": "Dataset Name",
    "operation": "(replace|remove)",
    "kinesisSyncRecords": [
        {
            "key": "Key",
            "value": "Value",
            "syncCount": 1,
            "lastModifiedDate": 1424801824343,
            "deviceLastModifiedDate": 1424801824343,
            "op": "(replace|remove)"
        },
        ...
    ],
    "lastModifiedDate": 1424801824343,
    "kinesisSyncRecordsURL": "S3Url",
    "payloadType": "(S3Url|Inline)",
    "syncCount": 1
}
```

For updates that are larger than the Kinesis maximum payload size of 50 KB, a presigned Amazon S3 URL will be included that contains the full contents of the update.

After you have configured Amazon Cognito streams, if you delete the Kinesis stream or change the role trust permission so that it can no longer be assumed by Amazon Cognito Sync, Amazon Cognito streams will be disabled. You will need to either recreate the Kinesis stream or fix the role, and then you will need to reenable the stream.

Bulk Publishing

Once you have configured Amazon Cognito streams, you will be able to execute a bulk publish operation for the existing data in your identity pool. After you initiate a bulk publish operation, either via the console or directly via the API, Amazon Cognito will start publishing this data to the same stream that is receiving your updates.

Amazon Cognito does not guarantee uniqueness of data sent to the stream when using the bulk publish operation. You may receive the same update both as an update as well as part of a bulk publish. Keep this in mind when processing the records from your stream.

To bulk publish all of your streams, follow steps 1-6 under Configuring Streams and then click Start bulk publish. You are limited to one ongoing bulk publish operation at any given time and to one successful bulk publish request every 24 hours.

Amazon Cognito Events

Amazon Cognito Events allows you to execute an AWS Lambda function in response to important events in Amazon Cognito. Amazon Cognito raises the Sync Trigger event when a dataset is synchronized. You can use the Sync Trigger event to take an action when a user updates data. The function can evaluate and optionally manipulate the data before it is stored in the cloud and synchronized to the user's other devices. This is useful to validate data coming from the device before it is synchronized to the user's other devices, or to update other values in the dataset based on incoming data such as issuing an award when a player reaches a new level.

The steps below will guide you through setting up a Lambda function that executes each time a Amazon Cognito Dataset is synchronized.

> **Note**
> When using Amazon Cognito events, you can only use the credentials obtained from Amazon Cognito Identity. If you have an associated Lambda function, but you call `UpdateRecords` with AWS account credentials (developer credentials), your Lambda function will not be invoked.

Creating a Function in AWS Lambda

To integrate Lambda with Amazon Cognito, you first need to create a function in Lambda. To do so:

Selecting the Lambda Function in Amazon Cognito

1. Open the Lambda console.
2. Click Create a Lambda function.
3. On the Select blueprint screen, search for and select "cognito-sync-trigger."
4. On the Configure event sources screen, leave the Event source type set to "Cognito Sync Trigger" and select your identity pool. Click Next.
5. On the Configure function screen, enter a name and description for your function. Leave Runtime set to "Node.js." Leave the code unchanged for our example. The default example makes no changes to the data being synced. It only logs the fact that the Amazon Cognito Sync Trigger event occurred. Leave Handler name set to "index.handler." For Role, select an IAM role that grants your code permission to access AWS Lambda. To modify roles, see the IAM console. Leave Advanced settings unchanged. Click Next.
6. On the Review screen, review the details and click Create function. The next page displays your new Lambda function.

Now that you have an appropriate function written in Lambda, you need to choose that function as the handler for the Amazon Cognito Sync Trigger event. The steps below walk you through this process.

From the console home page:

1. Click the name of the identity pool for which you want to set up Amazon Cognito Events. The Dashboard page for your identity pool appears.
2. In the top-right corner of the Dashboard page, click Edit identity pool. The Edit identity pool page appears.
3. Scroll down and click Cognito Events to expand it.
4. In the Sync Trigger dropdown menu, select the Lambda function that you want to trigger when a Sync event occurs.
5. Click Save Changes.

Now, your Lambda function will be executed each time a dataset is synchronized. The next section explains how you can read and modify the data in your function as it is being synchronized.

Writing a Lambda Function for Sync Triggers

Sync triggers follow the service provider interface programming paradigm. Amazon Cognito will provide input in the following JSON format to your Lambda function.

```
{
  "version": 2,
  "eventType": "SyncTrigger",
  "region": "us-east-1",
  "identityPoolId": "identityPoolId",
  "identityId": "identityId",
  "datasetName": "datasetName",
  "datasetRecords": {
    "SampleKey1": {
      "oldValue": "oldValue1",
      "newValue": "newValue1",
      "op": "replace"
    },
    "SampleKey2": {
      "oldValue": "oldValue2",
      "newValue": "newValue2",
      "op": "replace"
    },..
  }
}
```

Amazon Cognito expects the return value of the function in the same format as the input. A complete example is provided below.

Some key points to keep in mind when writing functions for the Sync Trigger event:

- When your Lambda function is invoked during UpdateRecords, it must respond within 5 seconds. If it does not, the Amazon Cognito Sync service throws a `LambdaSocketTimeoutException` exception. It is not possible to increase this timeout value.
- If you get a `LambdaThrottledException` exception, you should retry the sync operation (update records).
- Amazon Cognito will provide all the records present in the dataset as input to the function.
- Records updated by the app user will have the 'op' field set as “replace” and the records deleted will have 'op' field as "remove".
- You can modify any record, even if it is not updated by the app user.

- All the fields except the datasetRecords are read only and should not be changed. Changing these fields will result in a failure to update the records.
- To modify the value of a record, simply update the value and set the 'op' to "replace".
- To remove a record, either set the ‘op’ to remove or set the value to null.
- To add a record, simply add a new record to the datasetRecords array.
- Any omitted record in the response will be ignored for the update.

Sample Lambda Function

Here is a sample Lambda function showing how to access, modify and remove the data.

```
console.log('Loading function');

exports.handler = function(event, context) {
    console.log(JSON.stringify(event, null, 2));

    //Check for the event type
    if (event.eventType === 'SyncTrigger') {

        //Modify value for a key
        if('SampleKey1' in event.datasetRecords){
            event.datasetRecords.SampleKey1.newValue = 'ModifyValue1';
            event.datasetRecords.SampleKey1.op = 'replace';
        }

        //Remove a key
        if('SampleKey2' in event.datasetRecords){
            event.datasetRecords.SampleKey2.op = 'remove';
        }

        //Add a key
        if(!('SampleKey3' in event.datasetRecords)){
            event.datasetRecords.SampleKey3={'newValue':'ModifyValue3', 'op' : 'replace'};
        }

    }
    context.done(null, event);
};
```

Logging Amazon Cognito API Calls with AWS CloudTrail

Amazon Cognito is integrated with AWS CloudTrail, a service that captures specific API calls and delivers log files of the calls to an S3 bucket that you specify. CloudTrail captures API calls made from the Amazon Cognito console or from your code to the Amazon Cognito APIs. With the information collected by CloudTrail, you can determine which request was made to Amazon Cognito, the IP address from which the request was made, who made the request, when it was made, and so on.

To learn more about CloudTrail, including how to configure and enable it, see the *AWS CloudTrail User Guide*.

You can also create Amazon CloudWatch alarms for specific CloudTrail events. For example, you can set up CloudWatch to trigger an alarm if an identity pool configuration is changed. For more information, see Creating CloudWatch Alarms for CloudTrail Events: Examples.

Amazon Cognito Information in CloudTrail

When CloudTrail logging is enabled in your AWS account, API calls made to specific Amazon Cognito actions are tracked in CloudTrail log files, where they are written with other AWS service records. CloudTrail determines when to create and write to a new file based on a time period and file size.

The following actions are supported:

Amazon Cognito Your User Pools

- AddCustomAttributes
- CreateUserImportJob
- CreateUserPool
- CreateUserPoolClient
- DeleteUserPool
- DeleteUserPoolClient
- DescribeUserImportJob
- DescribeUserPool
- DescribeUserPoolClient
- GetCSVHeader

- ListUserImportJobs
- ListUserPoolClients
- ListUserPools
- StartUserImportJob
- StopUserImportJob
- UpdateUserPool
- UpdateUserPoolClient

Amazon Cognito Federated Identities

- CreateIdentityPool
- DeleteIdentityPool
- DescribeIdentityPool
- GetIdentityPoolRoles
- ListIdentityPools
- SetIdentityPoolRoles
- UpdateIdentityPool

Amazon Cognito Sync

- BulkPublish
- DescribeIdentityPoolUsage
- GetBulkPublishDetails
- GetCognitoEvents
- GetIdentityPoolConfiguration
- ListIdentityPoolUsage
- SetCognitoEvents
- SetIdentityPoolConfiguration

Every log entry contains information about who generated the request. The user identity information in the log helps you determine whether the request was made with root or AWS Identity and Access Management (IAM) user credentials, with temporary security credentials for a role or federated user, or by another AWS service. For more information, see the CloudTrail userIdentity Element.

You can store log files in your S3 bucket for as long as you want, but you can also define Amazon Simple Storage Service (Amazon S3) lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted by using Amazon S3 server-side encryption (SSE).

If you want to take quick action upon log file delivery, you can have CloudTrail publish Amazon Simple Notification Service (Amazon SNS) notifications when new log files are delivered. For more information, see Configuring Amazon SNS Notifications.

You can also aggregate Amazon Cognito log files from multiple AWS regions and multiple AWS accounts into a single S3 bucket. For more information, see Receiving CloudTrail Log Files From Multiple Regions.

Understanding Amazon Cognito Log File Entries

CloudTrail log files contain one or more log entries, where each entry lists multiple JSON-formatted events. A log entry represents a single request from any source and includes information about the requested operation, including the date and time of the operation, request parameters, and so on. Log entries are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following example is a log entry for a request for the `CreateIdentityPool` action. The request was made by an IAM user named Alice.

```
{
         "eventVersion":"1.03",
         "userIdentity":{
            "type":"IAMUser",
            "principalId":"PRINCIPAL_ID",
            "arn":"arn:aws:iam::123456789012:user/Alice",
            "accountId":"123456789012",
            "accessKeyId":"['EXAMPLE_KEY_ID']",
            "userName":"Alice"
         },
         "eventTime":"2016-01-07T02:04:30Z",
         "eventSource":"cognito-identity.amazonaws.com",
         "eventName":"CreateIdentityPool",
         "awsRegion":"us-east-1",
         "sourceIPAddress":"127.0.0.1",
         "userAgent":"USER_AGENT",
         "requestParameters":{
            "identityPoolName":"TestPool",
            "allowUnauthenticatedIdentities":true,
            "supportedLoginProviders":{
               "graph.facebook.com":"000000000000000"
            }
         },
         "responseElements":{
            "identityPoolName":"TestPool",
            "identityPoolId":"us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE",
            "allowUnauthenticatedIdentities":true,
            "supportedLoginProviders":{
               "graph.facebook.com":"000000000000000"
            }
         },
         "requestID":"15cc73a1-0780-460c-91e8-e12ef034e116",
         "eventID":"f1d47f93-c708-495b-bff1-cb935a6064b2",
         "eventType":"AwsApiCall",
         "recipientAccountId":"123456789012"
      }
```

Limits in Amazon Cognito

The following tables provide the soft (default) limits for Amazon Cognito, which are limits that can be changed. For information about these limits and how to them, see AWS Service Limits.

Soft Limits in Amazon Cognito User Pools

Resource	Default Limit
Maximum number of apps per user pool	25
Maximum number of user pools per account	60
Maximum number of user import jobs per user pool	50
Maximum number of groups per user pool	25
Maximum number of identity providers per user pool	25
Maximum number of resource servers per user pool	20
Maximum number of scopes per resource server	20

Soft Limits in Amazon Cognito Federated Identities

Resource	Default Limit
Maximum number of identity pools per account	60

Soft Limits in Amazon Cognito Sync

Resource	Default Limit
Maximum number of datasets per identity	20

Resource	Default Limit
Maximum number of records per dataset	1024
Maximum size of a single dataset	1 MB

The following tables describe Amazon Cognito hard limits, which are limits that cannot be changed.

Hard Limits in Amazon Cognito User Pools

Resource	Limit
Maximum number of custom attributes per user pool	25
Maximum characters per attribute	2048 bytes
Maximum character length for custom attribute name	20
Min/max password policy length	Between 6 and 99, inclusive
Maximum characters in email subject	140
Maximum character in email message	20,000
Maximum characters in SMS verification message	140
Maximum characters in password	256
Maximum character length for identity provider name	40
Maximum identifiers per identity provider	50
Maximum callback URLs per identity provider	100
Maximum logout URLs per identity provider	100

Hard Limits on Token Validity in Amazon Cognito User Pools

Resource	Limit
ID token	1 hour
Refresh token	Between 1 day and 3650 days, inclusive

Hard Limits on Code Validity in Amazon Cognito User Pools

Resource	Limit
Sign-up confirmation code	24 hours
User attribute verification code validity	24 hours
Multi-factor authentication code	3 minutes

Resource	Limit
Forgot password code	1 hour

Hard Limits in Amazon Cognito Federated Identities

Resource	Limit
Maximum number of identities per identity pool	Unlimited
Maximum character length for identity pool name	128 bytes
Maximum character length for login provider name	2048 bytes
Maximum number of results from a single List/ Lookup API call	60
Maximum Amazon Cognito user pool providers per identity pool	10

Hard Limits in Amazon Cognito Sync

Resource	Limit
Maximum character length for dataset name	128 bytes
Minimum waiting time for a bulk publish after a successful request	24 hours

Resource Permissions

This article covers restricting access to Amazon Cognito resources via IAM. If you are trying to define access permissions for your application's users, see Federated Identities Concepts (p. 136) for further details.

Amazon Resource Names (ARNs)

ARNs for Amazon Cognito Federated Identities

In Amazon Cognito Federated Identities, it is possible to restrict an IAM user's access to a specific identity pool, using the Amazon Resource Name (ARN) format, as in the following example. For more information about ARNs, see IAM Identifiers.

```
arn:aws:cognito-identity:REGION:ACCOUNT_ID:identitypool/IDENTITY_POOL_ID
```

ARNs for Amazon Cognito Sync

In Amazon Cognito Sync, customers can also restrict access by the identity pool ID, identity ID, and dataset name.

For APIs that operate on an identity pool, the identity pool ARN format is the same as for Amazon Cognito Federated Identities, except that the service name is `cognito-sync` instead of `cognito-identity`:

```
arn:aws:cognito-sync:REGION:ACCOUNT_ID:identitypool/IDENTITY_POOL_ID
```

For APIs that operate on a single identity, such as `RegisterDevice`, you can refer to the individual identity by the following ARN format:

```
arn:aws:cognito-sync:REGION:ACCOUNT_ID:identitypool/IDENTITY_POOL_ID/identity/IDENTITY_ID
```

For APIs that operate on datasets, such as `UpdateRecords` and `ListRecords`, you can refer to the individual dataset using the following ARN format:

```
arn:aws:cognito-sync:REGION:ACCOUNT_ID:identitypool/IDENTITY_POOL_ID/identity/IDENTITY_ID/
dataset/DATASET_NAME
```

ARNs for Amazon Cognito Your User Pools

For Amazon Cognito Your User Pools, it is possible to restrict an IAM user's access to a specific user pool, using the following ARN format:

```
arn:aws:cognito-idp:REGION:ACCOUNT_ID:userpool/USER_POOL_ID
```

Example Policies

Restricting Console Access to a Specific Identity Pool

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cognito-identity:ListIdentityPools"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "cognito-identity:*"
      ],
      "Resource": "arn:aws:cognito-identity:us-east-1:0123456789:identitypool/us-
east-1:1a1a1a1a-ffff-1111-9999-12345678"
    },
    {
      "Effect": "Allow",
      "Action": [
        "cognito-sync:*"
      ],
      "Resource": "arn:aws:cognito-sync:us-east-1:0123456789:identitypool/us-
east-1:1a1a1a1a-ffff-1111-9999-12345678"
    }
  ]
}
```

Allowing Access to Specific Dataset for All Identities in a Pool

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cognito-sync:ListRecords",
        "cognito-sync:UpdateRecords"
      ],
      "Resource": "arn:aws:cognito-sync:us-east-1:0123456789:identitypool/us-
east-1:1a1a1a1a-ffff-1111-9999-12345678/identity/*/dataset/UserProfile"
    }
  ]
}
```

Managed Policies

A number of policies are available via the IAM Console that customers can use to grant access to Amazon Cognito:

- AmazonCognitoPowerUser - Permissions for accessing and managing all aspects of your identity pools.
- AmazonCognitoReadOnly - Permissions for read only access to your identity pools.
- AmazonCognitoDeveloperAuthenticatedIdentities - Permissions for your authentication system to integrate with Amazon Cognito.

These policies are maintained by the Amazon Cognito team, so even as new APIs are added your IAM users will continue to have the same level of access.

> **Note**
> Because creating a new identity pool also requires creating IAM roles, any IAM user you want to be able to create new identity pools with must have the admin policy applied as well.

Signed versus Unsigned APIs

APIs that are signed with AWS credentials are capable of being restricted via an IAM policy. The following Cognito APIs are unsigned, and therefore cannot be restricted via an IAM policy:

Amazon Cognito Federated Identities

- `GetId`
- `GetOpenIdToken`
- `GetCredentialsForIdentity`
- `UnlinkIdentity`

Amazon Cognito Your User Pools

- `ChangePassword`
- `ConfirmDevice`
- `ConfirmForgotPassword`
- `ConfirmSignUp`
- `DeleteUser`
- `DeleteUserAttributes`
- `ForgetDevice`
- `ForgotPassword`
- `GetDevice`
- `GetUser`
- `GetUserAttributeVerificationCode`
- `GlobalSignOut`
- `InitiateAuth`
- `ListDevices`
- `ResendConfirmationCode`

- `RespondToAuthChallenge`
- `SetUserSettings`
- `SignUp`
- `UpdateDeviceStatus`
- `UpdateUserAttributes`
- `VerifyUserAttribute`

Using the Amazon Cognito Console

This guide provides a short introduction to working with the Amazon Cognito console.

What is the Amazon Cognito Console?

You can use the Amazon Cognito console to manage the resources for your applications that interact with Amazon Cognito. The console provides an intuitive user interface for performing many Amazon Cognito tasks, such as creating and managing identity pools and user pools, browsing the identities of your users, managing users in your user pools, viewing the number of data syncs for your application, and so on.

The Amazon Cognito console is a part of the AWS Management Console, which provides information about your account and billing. For more information on using the AWS Management Console, see Working with the AWS Management Console.

Create an Identity Pool

1. Sign in to the Amazon Cognito console, choose **Manage Federated Identities**, and then choose **Create new identity pool**.
2. Type a name for your identity pool, select **Enable access to unauthenticated identities**, and then choose **Create Pool**.
3. Choose **Allow** to create the two default roles associated with your identity pool–one for unauthenticated users and one for authenticated users. These default roles provide your identity pool access to Amazon Cognito Sync. You can modify the roles associated with your identity pool in the IAM console. For additional instructions on working with the Amazon Cognito console, see Using the Amazon Cognito Console (p. 230).

Delete an Identity Pool

From the Console home page:

1. Click the name of the identity pool that you want to delete. The **Dashboard page** for your identity pool appears.
2. In the top-right corner of the Dashboard page, click **Edit identity pool**. The **Edit identity pool** page appears.

3. Scroll down and click **Delete identity pool** to expand it.
4. Click **Delete identity pool**.
5. Click **Delete pool**.

> **Warning**
> When you click the delete button, you will permanently delete your identity pool and all the user data it contains. Deleting an identity pool will cause applications and other services utilizing the identity pool to stop working.

Delete an Identity from an Identity Pool

From the Console home page:

1. Click the name of the identity pool that contains the identity that you want to delete. The **Dashboard page** for your identity pool appears.
2. In the left-hand navigation on the Dashboard page, click **Identity browser**. The **Identities** page appears.
3. On the **Identities** page, enter the identity ID that you want to delete and then click **Search**.
4. On the **Identity details** page, click the **Delete identity** button, and then click **Delete**.

Enable or edit authentication providers

If you allow your users to authenticate using public identity providers (e.g. Amazon Cognito user pools, Facebook, Twitter, Amazon), you can specify your application identifiers in the Amazon Cognito Console. This associates the application ID (provided by the public login provider) with your identity pool.

You can also configure authentication rules for each provider from this page. Each provider allows up to 25 rules. The rules are applied in the order you save for each provider. For more information, see Role-Based Access Control (p. 145).

> **Warning**
> Changing the application ID to which your identity pool is linked will disable existing users from authenticating with Amazon Cognito. Learn more about External Identity Providers (p. 155).

From the Console home page:

1. Click the name of the identity pool for which you want to enable the external provider. The **Dashboard page** for your identity pool appears.
2. In the top-right corner of the Dashboard page, click **Edit identity pool**. The **Edit identity pool** page appears.
3. Scroll down and click **Authentication providers** to expand it.
4. Click the tab for the appropriate provider and enter the required information associated with that authentication provider.

Change the role associated with an identity type

Amazon Cognito defines two types of identities: authenticated and unauthenticated. Every identity in your identity pool is either authenticated and unauthenticated. Authenticated identities belong to users who are authenticated by a public login provider (Amazon Cognito user pools, Facebook, Amazon, Google, Twitter/Digits, SAML, or any OpenID Connect Providers) or a developer provider (your own backend authentication process). Unauthenticated identities typically belong to guest users.

For each identity type, there is an assigned role. This role has a policy attached to it which dictates which AWS services that role can access. When Amazon Cognito receives a request, the service will determine the identity type, determine the role assigned to that identity type, and use the policy attached to that role to respond. By modifying a policy or assigning a different role to an identity type, you can control which AWS services an identity type can access. To view or modify the policies associated with the roles in your identity pool, see the AWS IAM Console.

You can easily change which role is associated with an identity type using the Amazon Cognito Console. From the Console home page:

1. Click the name of the identity pool for which you want to modify roles. The **Dashboard page** for your identity pool appears.
2. In the top-right corner of the Dashboard page, click **Edit identity pool**. The **Edit identity pool** page appears.
3. Use the dropdown menus next to **Unauthenticated role** and **Authenticated role** to change roles. Click **Create new role** to create or modify the roles associated with each identity type in the AWS IAM console. For more information, see IAM Roles.

Enable or disable unauthenticated identities

Amazon Cognito can support unauthenticated identities by providing a unique identifier and AWS credentials for users who do not authenticate with an identity provider. If your application allows users who do not log in, you can enable access for unauthenticated identities. To learn more, see Identity Pools (p. 135).

From the Console home page:

1. Click the name of the identity pool for which you want to enable or disable unauthenticated identities. The **Dashboard page** for your identity pool appears.
2. In the top-right corner of the Dashboard page, click **Edit identity pool**. The **Edit identity pool** page appears.
3. Scroll down and click **Unauthenticated identities** to expand it.
4. Select the checkbox to enable or disable access to unauthenticated identities.
5. Click **Save Changes**.

Managing Datasets in the Amazon Cognito Console

If you have implemented Amazon Cognito Sync functionality in your application, the Amazon Cognito console enables you to manually create and delete datasets and records for individual identities. Any change you make to an identity's dataset or records in the Amazon Cognito console will not be saved until you click Synchronize in the console and will not be visible to the end user until the identity calls synchronize. The data being synchronized from other devices for individual identities will be visible once you refresh the list datasets page for a particular identity.

Create a Dataset for an Identity

From the Amazon Cognito console home page:

1. Click the name of the identity pool that contains the identity for which you want to create a dataset. The **Dashboard page** for your identity pool appears.
2. In the left-hand navigation on the Dashboard page, click **Identity browser**. The **Identities** page appears.
3. On the **Identities** page, enter the identity ID for which you want to create a dataset, and then click **Search**.
4. On the **Identity details** page for that identity, click the **Create dataset** button, enter a dataset name, and then click **Create and edit dataset**.
5. On the **Current dataset** page, click **Create record** to create a record to store in that dataset.
6. Enter a key for that dataset, the valid JSON value or values to store, and then click **Format as JSON** to prettify the value you entered and to confirm that it is well-formed JSON. When finished, click **Save Changes**.
7. Click **Synchronize** to synchronize the dataset. Your changes will not be saved until you click Synchronize and will not be visible to the user until the identity calls synchronize. To discard unsynchronized changes, select the change you wish to discard, and then click **Discard changes**.

Delete a Dataset Associated with an Identity

From the Amazon Cognito console home page:

1. Click the name of the identity pool that contains the identity for which you want to delete a dataset. The **Dashboard page** for your identity pool appears.
2. In the left-hand navigation on the Dashboard page, click **Identity browser**. The **Identities** page appears.
3. On the **Identities** page, enter the identity ID containing the dataset which you want to delete, and then click **Search**.
4. On the **Identity details** page, select the checkbox next to the dataset or datasets that you want to delete, click **Delete selected**, and then click **Delete**.

Set Up Amazon Cognito Streams

Amazon Cognito Streams gives developers control and insight into their data stored in Amazon Cognito Sync. Developers can now configure an Kinesis stream to receive events as data. Amazon Cognito can push each dataset change to an Kinesis stream you own in real time. For instructions on how to set up Amazon Cognito Streams in the Amazon Cognito console, see Amazon Cognito Streams (p. 215).

Bulk Publish Data

Bulk publish can be used to export data already stored in your Amazon Cognito Sync store to an Kinesis stream. For instructions on how to bulk publish all of your streams, see Amazon Cognito Streams (p. 215).

Enable Push Synchronization

Amazon Cognito automatically tracks the association between identity and devices. Using the push sync feature, you can ensure that every instance of a given identity is notified when identity data changes.

Push sync ensures that, whenever the sync store data changes for a particular identity, all devices associated with that identity receive a silent push notification informing them of the change.

You can enable Push Sync via the Amazon Cognito console. From the console home page:

1. Click the name of the identity pool for which you want to enable Push Sync. The **Dashboard page** for your identity pool appears.
2. In the top-right corner of the Dashboard page, click **Edit identity pool**. The **Edit identity pool** page appears.
3. Scroll down and click **Push synchronization** to expand it.
4. In the **Service role** dropdown menu, select the IAM role that grants Amazon Cognito permission to send an SNS notification. Click **Create role** to create or modify the roles associated with your identity pool in the AWS IAM console.
5. Select a platform application, and then click **Save Changes**.

Set Up Amazon Cognito Events

Amazon Cognito Events allows you to execute an AWS Lambda function in response to important events in Amazon Cognito Sync. Amazon Cognito Sync raises the Sync Trigger event when a dataset is synchronized. You can use the Sync Trigger event to take an action when a user updates data. For instructions on setting up Amazon Cognito Events from the console, see Amazon Cognito Events (p. 217).

To learn more about AWS Lambda, see AWS Lambda.

Amazon Cognito API Reference

For information about Amazon Cognito reference, see the following topics:

Topics

- Amazon Cognito Auth API Reference (p. 235)
- Amazon Cognito User Pools API Reference (p. 243)
- Amazon Cognito Federated Identities API Reference (p. 243)
- Amazon Cognito Sync API Reference (p. 243)

Amazon Cognito Auth API Reference

This section contains the HTTPS contract to the Amazon Cognito authentication server from a user pool client, including sample requests and responses. It describes the expected behavior from the authentication server for positive and negative conditions.

In addition to the server contract REST API, Amazon Cognito also provides SDKs for Android, iOS, and JavaScript that make it easier to form requests and interact with the server.

Topics

- AUTHORIZATION Endpoint (p. 235)
- TOKEN Endpoint (p. 239)
- LOGOUT Endpoint (p. 242)

AUTHORIZATION Endpoint

The `/oauth2/authorize` endpoint signs the user in.

GET /oauth2/authorize

The `/oauth2/authorize` endpoint only supports `HTTPS GET`. The user pool client typically makes this request through the system browser, which would typically be Custom Chrome Tab in Android and Safari View Control in iOS.

Request Parameters

response_type

The response type. Must be `code` or `token`. Indicates whether the client wants an authorization code (authorization code grant flow) for the end user or directly issues tokens for end user (implicit flow).

Required

client_id

The Client ID.

Must be a pre-registered client in the user pool and must be enabled for federation.

Required

redirect_uri

The URL to which the authentication server redirects the browser after authorization has been granted by the user.

Must have been pre-registered with a client.

Required

state

An opaque value the clients adds to the initial request. The authorization server includes this value when redirecting the back to the client.

This value must be used by the client to prevent CSRF attacks.

Optional but strongly recommended.

identity_provider

Used by the developer to directly authenticate with a specific provider.

Optional

idp_identifier

Used by the developer to map to a provider name without exposing the provider name.

Optional

scope

Can be a combination of any system-reserved scopes or custom scopes associated with a client. Scopes must be separated by spaces. System reserved scopes are `openid`, `email`, `phone`, `profile`, and `aws.cognito.signin.user.admin`. Any scope used must be preassociated with the client or it will be ignored at runtime.

If the client doesn't request any scopes, the authentication server uses all scopes associated with the client.

An ID token is only returned if `openid` scope is requested. The access token can be only used against Amazon Cognito User Pools if `aws.cognito.signin.user.admin` scope is requested. The `phone`, `email`, and `profile` scopes can only be requested if `openid` scope is also requested. These scopes dictate the claims that go inside the ID token.

Optional

code_challenge_method

The method used to generate the challenge. The PKCE RFC defines two methods, S256 and plain; however, Amazon Cognito authentication server supports only S256.

Optional

code_challenge

The generated challenge from the `code_verifier`.

Required only when the `code_challenge_method` is specified.

Examples of Positive Requests

Sample Request: Authorization code grant request

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&
client_id=ad398u21ijw3s9w3939&
redirect_uri=https://YOUR_APP/redirect_uri&
state=STATE&
scope=openid+profile+aws.cognito.signin.user.admin
```

Sample Response

The Amazon Cognito authentication server redirects back to your app with the authorization code and state. The code and state must be returned in the query string parameters and not in the fragment. A query string is the part of a web request that appears after a '?' character; the string can contain one or more parameters separated by '&' characters. A fragment is the part of a web request that appears after a '#' character to specify a subsection of a document.

```
HTTP/1.1 302 Found
Location: https://YOUR_APP/redirect_uri?code=AUTHORIZATION_CODE&state=STATE
```

Sample Request: Authorization code grant request with PKCE

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&
client_id=ad398u21ijw3s9w3939&
redirect_uri=https://YOUR_APP/redirect_uri&
state=STATE&
scope=aws.cognito.signin.user.admin&
code_challenge_method=S256&
code_challenge=CODE_CHALLENGE
```

Sample Response

The authentication server redirects back to your app with the authorization code and state. The code and state must be returned in the query string parameters and not in the fragment.

```
HTTP/1.1 302 Found
Location: https://YOUR_APP/redirect_uri?code=AUTHORIZATION_CODE&state=STATE
```

Sample Request: Token grant request without `openid` scope

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=token&
client_id=ad398u21ijw3s9w3939&
```

```
redirect_uri=https://YOUR_APP/redirect_uri&
state=STATE&
scope=aws.cognito.signin.user.admin
```

Sample Response

The Amazon Cognito authorization server redirects back to your app with access token. Since `openid` scope was not requested, an ID token is not returned. A refresh token is never returned in this flow. Token and state are returned in the fragment and not in the query string.

```
HTTP/1.1 302 Found
Location: https://YOUR_APP/
redirect_uri#access_token=ACCESS_TOKEN&token_type=bearer&expires_in=3600&state=STATE
```

Sample Request: Token Grant request with `openid` scope

```
GET
                            https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/
authorize?
response_type=token&
client_id=ad398u21ijw3s9w3939&
redirect_uri=https://YOUR_APP/redirect_uri&
state=STATE&
scope aws.cognito.signin.user.admin+openid+profile
```

Sample Response

The authorization server redirects back to your app with access token and ID token (because `openid` scope was included).

```
HTTP/1.1 302 Found
Location: https://YOUR_APP/
redirect_ur#id_token=ID_TOKEN&access_token=ACCESS_TOKEN&token_type=bearer&expires_in=3600&state=STATE
```

Examples of Negative Requests

The following are examples of negative requests:

- If `client_id` and `redirect_uri` are valid but there are other problems with the request parameters (for example, if `response_type` is not included; if `code_challenge` is supplied but `code_challenge_method` is not supplied; or if `code_challenge_method` is not 'S256'), the authentication server redirects the error to client's `redirect_uri`.

  ```
  HTTP 1.1 302 Found Location: https://client_redirect_uri?error=invalid_request
  ```
- If the client requests 'code' or 'token' in `response_type` but does not have permission for these requests, the Amazon Cognito authorization server should return `unauthorized_client` to client's `redirect_uri`, as follows:

  ```
  HTTP 1.1 302 Found Location: https://client_redirect_uri?error=unauthorized_client
  ```
- If the client requests invalid, unknown, malformed scope, the Amazon Cognito authorization server should return `invalid_scope` to the client's `redirect_uri`, as follows:

  ```
  HTTP 1.1 302 Found Location: https://client_redirect_uri?error=invalid_scope
  ```
- If there is any unexpected error in the server, the authentication server should return `server_error` to client's `redirect_uri`. It should not be the HTTP 500 error displayed to the end user in the browser, because this error doesn't get sent to the client. The following error should return:

  ```
  HTTP 1.1 302 Found Location: https://client_redirect_uri?error=server_error
  ```

TOKEN Endpoint

The `/oauth2/token` endpoint gets the user's tokens.

POST /oauth2/token

The /oauth2/token endpoint only supports `HTTPS POST`. The user pool client makes requests to this endpoint directly and not through the system browser.

Request Parameters

Authorization

If the client was issued a secret, the client must pass its `client_id` and `client_secret` in the authorization header through Basic HTTP authorization. The secret is Basic `Base64Encode(client_id:client_secret)`.

code

Required if `grant_type` is `authorization_code`.

Content-Type

Must always be `'application/x-www-form-urlencoded'`.

Request Parameters in Body

grant_type

Grant type.

Must be `authorization_code` or `refresh_token` or `client_credentials`.

Required

client_id

Client ID.

Must be a preregistered client in the user pool. The client must be enabled for Amazon Cognito federation.

Required if the client is public and does not have a secret.

scope

Can be a combination of any custom scopes associated with a client. Any scope requested must be preassociated with the client or it will be ignored at runtime. If the client doesn't request any scopes, the authentication server uses all custom scopes associated with the client.

Optional. Only used if the `grant_type` is `client_credentials`.

redirect_uri

Must be the same `redirect_uri` that was used to get `authorization_code` in /oauth2/authorize.

Required only if `grant_type` is `authorization_code`.

refresh_token

The refresh token.

Required if `grant_type` is `refresh_token`.

code_verifier

The proof key.

Required if `grant_type` is `authorization_code` and the authorization code was requested with PKCE.

Examples of Positive Requests

Sample Request: Exchanging authorization code for tokens

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token&
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic aSdxd892iujendek328uedj

grant_type=authorization_code&
client_id=djc98u3jiedmi283eu928&
code=AUTHORIZATION_CODE&
redirect_uri=com.myclientapp://myclient/redirect
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
 "access_token":"eyJz9sdfsdfsdfsd",
 "refresh_token":"dn43ud8uj32nk2je",
 "id_token":"dmcxd329ujdmkemkd349r",
 "token_type":"Bearer",
 "expires_in":3600
}
```

Sample Request: Exchanging client credentials for access token

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token >
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic aSdxd892iujendek328uedj

grant_type=client_credentials&
scope={resourceServerIdentifier1}/{scope1} {resourceServerIdentifier2}/{scope2}
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
 "access_token":"eyJz9sdfsdfsdfsd",
 "token_type":"Bearer",
 "expires_in":3600
}
```

Sample Request: Exchanging authorization code grant with PKCE for tokens

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic aSdxd892iujendek328uedj

grant_type=authorization_code&
```

```
client_id=djc98u3jiedmi283eu928&
code=AUTHORIZATION_CODE&
code_verifier=CODE_VERIFIER&
redirect_uri=com.myclientapp://myclient/redirect
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
 "access_token":"eyJz9sdfsdfsdfsd",
 "refresh_token":"dn43ud8uj32nk2je",
 "id_token":"dmcxd329ujdmkemkd349r",
 "token_type":"Bearer",
 "expires_in":3600
}
```

Sample Request: Exchanging refresh token for tokens

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token >
Content-Type='application/x-www-form-urlencoded'
Authorization=Basic aSdxd892iujendek328uedj

grant_type=refresh_token&
client_id=djc98u3jiedmi283eu928&
refresh_token=REFRESH_TOKEN
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
 "access_token":"eyJz9sdfsdfsdfsd",
 "refresh_token":"dn43ud8uj32nk2je",
 "id_token":"dmcxd329ujdmkemkd349r",
 "token_type":"Bearer",
 "expires_in":3600
}
```

Examples of Negative Requests

Sample Error Response

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8

{
"error":"invalid_request|invalid_client|invalid_grant|unauthorized_client|
unsupported_grant_type|"
}
```

invalid_request

The request is missing a required parameter, includes an unsupported parameter value (other than `unsupported_grant_type`), or is otherwise malformed. For example, `grant_type` is `refresh_token` but `refresh_token` is not included.

invalid_client

Client authentication failed. For example, when the client includes `client_id` and `client_secret` in the authorization header, but there's no such client with that `client_id` and `client_secret`.

invalid_grant

Refresh token has been revoked.

Authorization code has been consumed already or does not exist.

unauthorized_client

Client is not allowed for code grant flow or for refreshing tokens.

unsupported_grant_type

Returned if `grant_type` is anything other than `authorization_code` or `refresh_token`.

LOGOUT Endpoint

The `/logout` endpoint signs the user out.

GET /logout

The `/logout` endpoint only supports `HTTPS GET`. The user pool client typically makes this request through the system browser, which would typically be Custom Chrome Tab in Android and Safari View Control in iOS.

Request Parameters

client_id

The app client ID for your app. To obtain an app client ID, you must register the app in the user pool. For more information, see Specifying User Pool App Settings (p. 24).

Optional

logout_uri

A sign-out URL that you registered for your client app. For more information, see Specifying Identity Provider Settings for Your User Pool App (p. 83).

Optional

Sample Requests

Example #1: Logout and Redirect Back to Client

This example clears out the existing session and redirects back to the client. Both parameters are required.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/logout?
client_id=ad398u21ijw3s9w3939&
logout_uri=com.myclientapp://myclient/logout
```

Example #2: Logout and Prompt the User to Sign In As Another User

This example clears out the existing session and shows the login screen, using the same parameters as for `GET /oauth2/authorize`.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/logout?
response_type=code&
client_id=ad398u21ijw3s9w3939&
redirect_uri=https://YOUR_APP/redirect_uri&
state=STATE&
scope=openid+profile+aws.cognito.signin.user.admin
```

Amazon Cognito User Pools API Reference

Amazon Cognito Your User Pools API Reference

Amazon Cognito Federated Identities API Reference

Amazon Cognito API Reference

Amazon Cognito Sync API Reference

Amazon Cognito Sync API Reference

Document History for Amazon Cognito

The following table describes the documentation for this release of Amazon Cognito.

- **Original API versions:**

  Amazon Cognito User Pools: 2016-04-18

  Amazon Cognito Federated Identities: 2014-06-30

  Amazon Cognito Sync: 2014-06-30
- **Latest documentation update:** August 10, 2017

Change	Description	Date
Federation and built-in app UI features of Amazon Cognito User Pools	Added the ability to allow your users to sign in to your user pool through Facebook, Google, Login with Amazon, or a SAML identity provider. Added a customizable built-in app UI and OAuth 2.0 support with custom claims.	August 10, 2017
HIPAA and PCI compliance-related feature changes	Added the ability to allow your users to use a phone number or email address as their user name.	July 6, 2017
User groups and role-based access control features	Added administrative capability to create and manage user groups. Administrators can assign IAM roles to users based on group membership and administrator-created rules. For more information, see User Groups (p. 106) and Role-Based Access Control (p. 145).	December 15, 2016

Change	Description	Date
Documentation update	Updated iOS code examples in Developer Authenticated Identities (p. 177).	November 18, 2016
Documentation update	Added information about confirmation flow for user accounts. For more information, see Signing Up and Confirming User Accounts (p. 117).	November 9, 2016
Create user accounts feature	Added administrative capability to create user accounts through the Amazon Cognito console and the API. For more information, see Creating User Accounts as Administrator in the AWS Management Console and with the Amazon Cognito User Pools API (p. 103).	October 6, 2016
Documentation update	Updated examples that show how to use AWS Lambda triggers with user pools. For more information, see Customizing User Pool Workflows by Using AWS Lambda Triggers (p. 67).	September 27, 2016
User import feature	Added bulk import capability for Cognito User Pools. Use this feature to migrate users from your existing identity provider to an Amazon Cognito user pool. For more information, see Importing Users into User Pools (p. 108).	September 1, 2016
General availability of Cognito User Pools	Added the Cognito User Pools feature. Use this feature to create and maintain a user directory and add sign-up and sign-in to your mobile app or web application using user pools. For more information, see Amazon Cognito User Pools (p. 9).	July 28, 2016
SAML support	Added support for authentication with identity providers through Security Assertion Markup Language 2.0 (SAML 2.0). For more information, see SAML Identity Provider (p. 175).	June 23, 2016

Change	Description	Date
CloudTrail integration	Added integration with AWS CloudTrail. For more information, see Logging Amazon Cognito API Calls with AWS CloudTrail (p. 220).	February 18, 2016
Twitter support	Added support for Twitter as an external identity provider. For more information, see External Identity Providers (p. 155).	April 30, 2015
Integration of events with Lambda	Enables you to execute an AWS Lambda function in response to important events in Amazon Cognito. For more information, see Amazon Cognito Events (p. 217).	April 9, 2015
Data stream to Amazon Kinesis	Provides control and insight into your data streams. For more information, see Amazon Cognito Streams (p. 215).	March 4, 2015
Push synchronization	Enables support for silent push synchronization. For more information, see Amazon Cognito Sync (p. 190).	November 6, 2014
OpenID Connect support	Enables support for OpenID Connect providers. For more information, see External Identity Providers (p. 155).	October 23, 2014
Developer-authenticated identities support added	Enables developers who own their own authentication and identity management systems to be treated as an identity provider in Amazon Cognito. For more information, see Developer Authenticated Identities (p. 177).	September 29, 2014
Amazon Cognito general availability		July 10, 2014

AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.