Amazon Cognito

Developer Guide

Amazon Cognito: Developer Guide

Table of Contents

What Is Amazon Cognito?

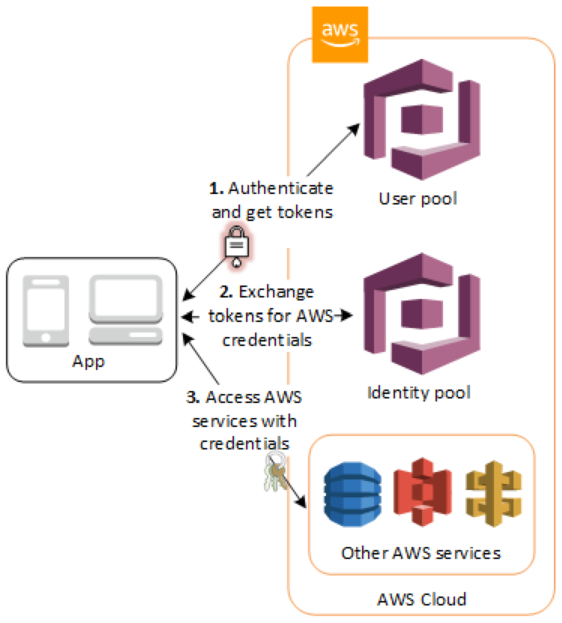
Amazon Cognito provides authentication, authorization, and user management for your web and mobile apps. Your users can sign in directly with a user name and password, or through a third party such as Facebook, Amazon, Google or Apple.

The two main components of Amazon Cognito are user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your app users. Identity pools enable you to grant your users access to other AWS services. You can use identity pools and user pools separately or together.

An Amazon Cognito user pool and identity pool used together

See the diagram for a common Amazon Cognito scenario. Here the goal is to authenticate your user, and then grant your user access to another AWS service.

1. In the first step your app user signs in through a user pool and receives user pool tokens after a successful authentication.
2. Next, your app exchanges the user pool tokens for AWS credentials through an identity pool.
3. Finally, your app user can then use those AWS credentials to access other AWS services such as Amazon S3 or DynamoDB.


aws
1. Authenticate and get tokens
User pool
2. Exchange tokens for AWS credentials
App
Identity pool
3. Access AWS services with credentials
Other AWS services
AWS Cloud

For more examples using identity pools and user pools, see Common Amazon Cognito Scenarios (p. 9).

Amazon Cognito is compliant with SOC 1-3, PCI DSS, ISO 27001, and is HIPAA-BAA eligible. For more information, see AWS Services in Scope. See also Regional Data Considerations (p. 322).

Topics

- Getting Started with Amazon Cognito (p. 2)
- Regional Availability (p. 3)
- Pricing for Amazon Cognito (p. 3)
- Using the Amazon Cognito Console (p. 3)

Features of Amazon Cognito

User pools

A user pool is a user directory in Amazon Cognito. With a user pool, your users can sign in to your web or mobile app through Amazon Cognito, or federate through a third-party identity provider (IdP). Whether your users sign in directly or through a third party, all members of the user pool have a directory profile that you can access through an SDK.

User pools provide:

- Sign-up and sign-in services.
- A built-in, customizable web UI to sign in users.
- Social sign-in with Facebook, Google, Login with Amazon, and Sign in with Apple, and through SAML and OIDC identity providers from your user pool.
- User directory management and user profiles.
- Security features such as multi-factor authentication (MFA), checks for compromised credentials, account takeover protection, and phone and email verification.
- Customized workflows and user migration through AWS Lambda triggers.

For more information about user pools, see Getting Started with User Pools (p. 20) and the Amazon Cognito user pools API Reference.

Identity pools

With an identity pool, your users can obtain temporary AWS credentials to access AWS services, such as Amazon S3 and DynamoDB. Identity pools support anonymous guest users, as well as the following identity providers that you can use to authenticate users for identity pools:

- Amazon Cognito user pools
- Social sign-in with Facebook, Google, Login with Amazon, and Sign in with Apple
- OpenID Connect (OIDC) providers
- SAML identity providers
- Developer authenticated identities

To save user profile information, your identity pool needs to be integrated with a user pool.

For more information about identity pools, see Getting Started with Amazon Cognito Identity Pools (Federated Identities) (p. 187) and the Amazon Cognito Identity Pools API Reference.

Getting Started with Amazon Cognito

For a guide to top tasks and where to start, see Getting Started with Amazon Cognito (p. 5).

For videos, articles, documentation, and sample apps, see Amazon Cognito Developer Resources.

To use Amazon Cognito, you need an AWS account. For more information, see Using the Amazon Cognito Console (p. 3).

Regional Availability

Amazon Cognito is available in multiple AWS Regions worldwide. In each Region, Amazon Cognito is distributed across multiple Availability Zones. These Availability Zones are physically isolated from each other, but are united by private, low-latency, high-throughput, and highly redundant network connections. These Availability Zones enable AWS to provide services, including Amazon Cognito, with very high levels of availability and redundancy, while also minimizing latency.

For a list of all the Regions where Amazon Cognito is currently available, see AWS Regions and Endpoints in the *Amazon Web Services General Reference*. To learn more about the number of Availability Zones that are available in each Region, see AWS Global Infrastructure.

Pricing for Amazon Cognito

For information about Amazon Cognito pricing, see Amazon Cognito Pricing.

Using the Amazon Cognito Console

You can use the Amazon Cognito console to create and manage user pools and identity pools.

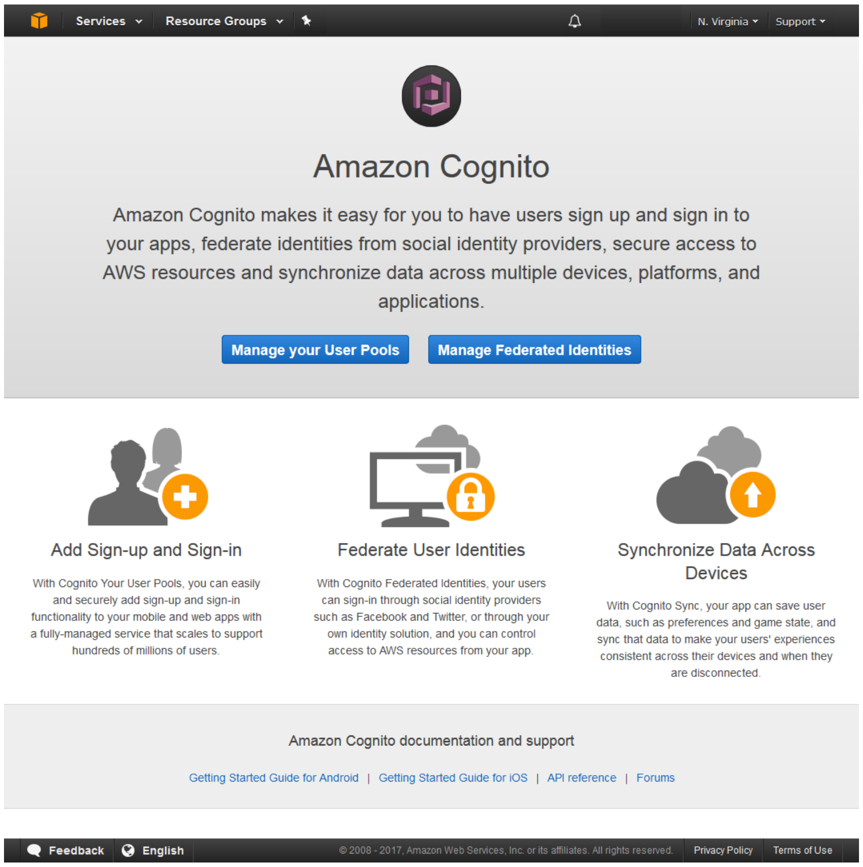
To use the Amazon Cognito console

1. To use Amazon Cognito, you need to sign up for an AWS account.
2. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
3. To create or edit a user pool, choose **Manage your User Pools**.

   For more information, see Getting Started with User Pools (p. 20).
4. To create or edit an identity pool, choose **Manage Identity Pools**.

   For more information, see Getting Started with Amazon Cognito Identity Pools (Federated Identities) (p. 187).

The Amazon Cognito console is a part of the AWS Management Console, which provides information about your account and billing. For more information, see Working with the AWS Management Console.



Amazon Cognito

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.

Add Sign-up and Sign-in

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.

Federate User Identities

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.

Synchronize Data Across Devices

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.

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

Getting Started with Amazon Cognito

This section describes the top Amazon Cognito tasks and where to start. For an overview of Amazon Cognito, see What Is Amazon Cognito? (p. 1).

The two main components of Amazon Cognito are user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your web and mobile app users. Identity pools provide AWS credentials to grant your users access to other AWS services. You can use user pools and identity pools separately or together.

Top Tasks and Where to Start
Add Sign-up and Sign-in with a User Pool
1. Create a user directory with a user pool. 2. Add an app to enable the hosted UI. 3. Add social sign-in to a user pool. 4. Add sign-in through SAML-based identity providers (IdPs) to a user pool. 5. Add sign-in through OpenID Connect (OIDC) IdPs to a user pool. 6. Install a user pool SDK. 7. Customize the built-in hosted web UI sign-in and sign-up pages. 8. Configure user pool security features. 9. Customize user pool workflows with Lambda triggers. 10. Gather data and target campaigns with Amazon Pinpoint analytics.
Manage Users in a User Pool
• Sign up and confirm user accounts. • Create user accounts as administrator. • Manage and search user accounts. • Add groups to a user pool. • Import users into a user pool.
Access Resources
Common Amazon Cognito scenarios: • Authenticate with a user pool. • Access backend resources through a user pool. • Access API Gateway and Lambda through a user pool. • Access AWS services with a user pool and an identity pool. • Access AWS services through a third party and an identity pool. • Access AWS AppSync resources through a user pool or an identity pool.

Get an AWS account and your root user credentials

To access AWS, you must sign up for an AWS account.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Creating an IAM user

If your account already includes an IAM user with full AWS administrative permissions, you can skip this section.

When you first create an Amazon Web Services (AWS) account, you begin with a single sign-in identity. That identity has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user*. When you sign in, enter the email address and password that you used to create the account.

> **Important**
> We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks. To view the tasks that require you to sign in as the root user, see Tasks that require root user credentials.

To create an administrator user for yourself and add the user to an administrators group (console)

1. Sign in to the IAM console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   > **Note**
   > We strongly recommend that you adhere to the best practice of using the `Administrator` IAM user that follows and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.
2. In the navigation pane, choose **Users** and then choose **Add user**.
3. For **User name**, enter `Administrator`.
4. Select the check box next to **AWS Management Console access**. Then select **Custom password**, and then enter your new password in the text box.
5. (Optional) By default, AWS requires the new user to create a new password when first signing in. You can clear the check box next to **User must create a new password at next sign-in** to allow the new user to reset their password after they sign in.
6. Choose **Next: Permissions**.
7. Under **Set permissions**, choose **Add user to group**.
8. Choose **Create group**.
9. In the **Create group** dialog box, for **Group name** enter `Administrators`.
10. Choose **Filter policies**, and then select **AWS managed - job function** to filter the table contents.
11. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.

> **Note**
> You must activate IAM user and role access to Billing before you can use the `AdministratorAccess` permissions to access the AWS Billing and Cost Management console. To do this, follow the instructions in step 1 of the tutorial about delegating access to the billing console.

12. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
13. Choose **Next: Tags**.
14. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM entities in the *IAM User Guide*.
15. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

You can use this same process to create more groups and users and to give your users access to your AWS account resources. To learn about using policies that restrict user permissions to specific AWS resources, see Access management and Example policies.

Signing in as an IAM user

Sign in to the IAM console by choosing **IAM user** and entering your AWS account ID or account alias. On the next page, enter your IAM user name and your password.

> **Note**
> For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose the sign-in link beneath the button to return to the main sign-in page. From there, you can enter your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

Creating IAM user access keys

Access keys consist of an access key ID and secret access key, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them from the AWS Management Console. As a best practice, do not use the AWS account root user access keys for any task where it's not required. Instead, create a new administrator IAM user with access keys for yourself.

The only time that you can view or download the secret access key is when you create the keys. You cannot recover them later. However, you can create new access keys at any time. You must also have permissions to perform the required IAM actions. For more information, see Permissions required to access IAM resources in the *IAM User Guide*.

To create access keys for an IAM user

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. Choose the name of the user whose access keys you want to create, and then choose the **Security credentials** tab.
4. In the **Access keys** section, choose **Create access key**.
5. To view the new access key pair, choose **Show**. You will not have access to the secret access key again after this dialog box closes. Your credentials will look something like this:

  - Access key ID: AKIAIOSFODNN7EXAMPLE
  - Secret access key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
6. To download the key pair, choose **Download .csv file**. Store the keys in a secure location. You will not have access to the secret access key again after this dialog box closes.

   Keep the keys confidential in order to protect your AWS account and never email them. Do not share them outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.
7. After you download the `.csv` file, choose **Close**. When you create an access key, the key pair is active by default, and you can use the pair right away.

Related topics

- What is IAM? in the *IAM User Guide*
- AWS security credentials in *AWS General Reference*

Common Amazon Cognito Scenarios

This topic describes six common scenarios for using Amazon Cognito.

The two main components of Amazon Cognito are user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your web and mobile app users. Identity pools provide AWS credentials to grant your users access to other AWS services.

A user pool is a user directory in Amazon Cognito. Your app users can sign in either directly through a user pool, or federate through a third-party identity provider (IdP). The user pool manages the overhead of handling the tokens that are returned from social sign-in through Facebook, Google, Amazon, and Apple, and from OpenID Connect (OIDC) and SAML IdPs. Whether your users sign in directly or through a third party, all members of the user pool have a directory profile that you can access through an SDK.

With an identity pool, your users can obtain temporary AWS credentials to access AWS services, such as Amazon S3 and DynamoDB. Identity pools support anonymous guest users, as well as federation through third-party IdPs.

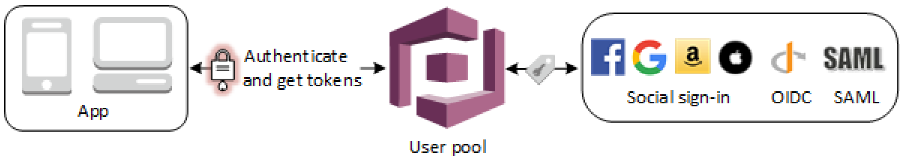
Topics

Authenticate with a User Pool

You can enable your users to authenticate with a user pool. Your app users can sign in either directly through a user pool, or federate through a third-party identity provider (IdP). The user pool manages the overhead of handling the tokens that are returned from social sign-in through Facebook, Google, Amazon, and Apple, and from OpenID Connect (OIDC) and SAML IdPs.

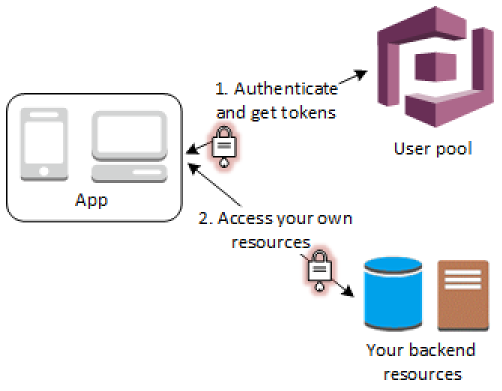
After a successful authentication, your web or mobile app will receive user pool tokens from Amazon Cognito. You can use those tokens to retrieve AWS credentials that allow your app to access other AWS services, or you might choose to use them to control access to your server-side resources, or to the Amazon API Gateway.

For more information, see User Pool Authentication Flow (p. 306) and Using Tokens with User Pools (p. 149).



Access Your Server-side Resources with a User Pool

After a successful user pool sign-in, your web or mobile app will receive user pool tokens from Amazon Cognito. You can use those tokens to control access to your server-side resources. You can also create user pool groups to manage permissions, and to represent different types of users. For more information on using groups to control access your resources see Adding Groups to a User Pool (p. 127).



Once you configure a domain for your user pool, Amazon Cognito provisions a hosted web UI that allows you to add sign-up and sign-in pages to your app. Using this OAuth 2.0 foundation you can create your own resource server to enable your users to access protected resources. For more information see Defining Resource Servers for Your User Pool (p. 44).

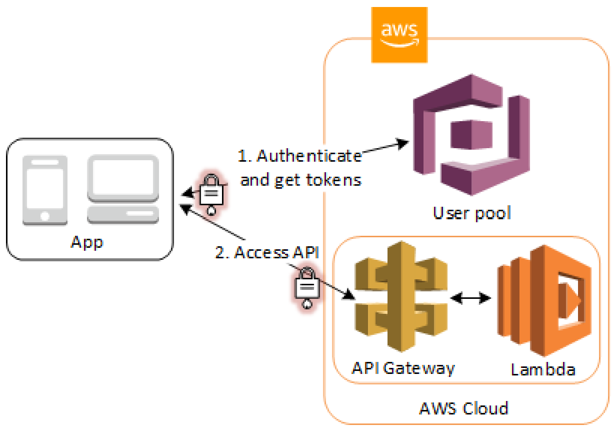
For more information about user pool authentication see User Pool Authentication Flow (p. 306) and Using Tokens with User Pools (p. 149).

Access Resources with API Gateway and Lambda with a User Pool

You can enable your users to access your API through API Gateway. API Gateway validates the tokens from a successful user pool authentication, and uses them to grant your users access to resources including Lambda functions, or your own API.

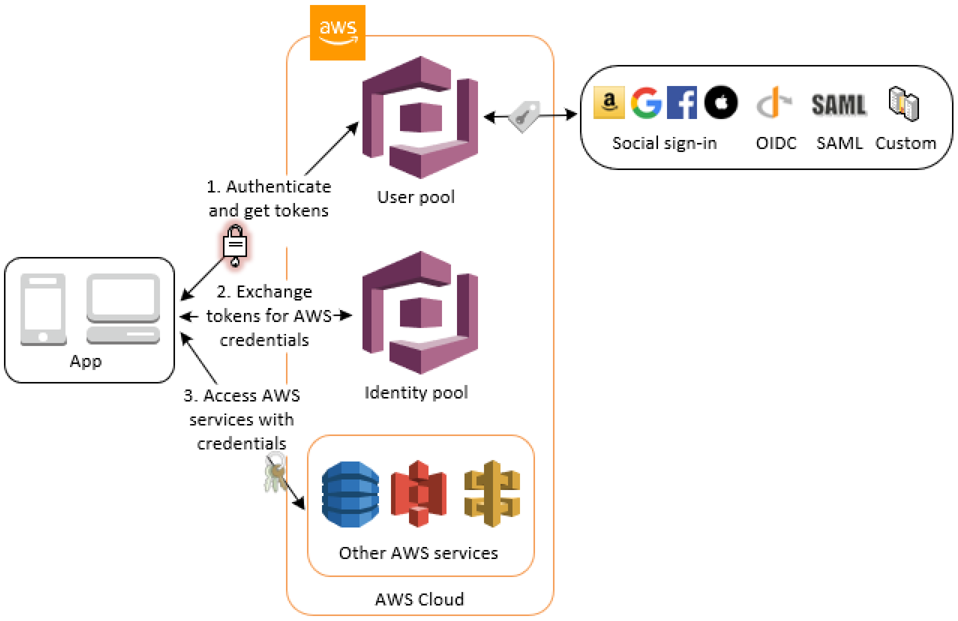
You can use groups in a user pool to control permissions with API Gateway by mapping group membership to IAM roles. The groups that a user is a member of are included in the ID token provided by a user pool when your app user signs in. For more information on user pool groups See Adding Groups to a User Pool (p. 127).

You can submit your user pool tokens with a request to API Gateway for verification by an Amazon Cognito authorizer Lambda function. For more information on API Gateway, see Using API Gateway with Amazon Cognito user pools.



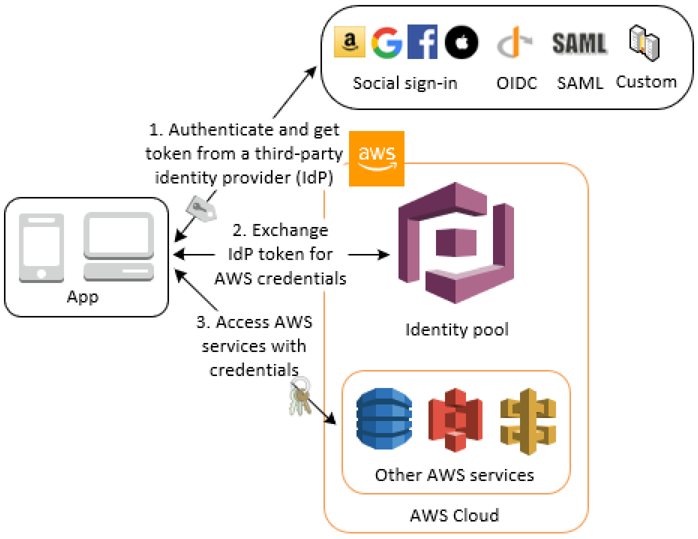
Access AWS Services with a User Pool and an Identity Pool

After a successful user pool authentication, your app will receive user pool tokens from Amazon Cognito. You can exchange them for temporary access to other AWS services with an identity pool. For more information, see Accessing AWS Services Using an Identity Pool After Sign-in (p. 160) and Getting Started with Amazon Cognito Identity Pools (Federated Identities) (p. 187).


aws
Social sign-in
OIDC
SAML
Custom
User pool
1. Authenticate and get tokens
2. Exchange tokens for AWS credentials
App
Identity pool
3. Access AWS services with credentials
Other AWS services
AWS Cloud

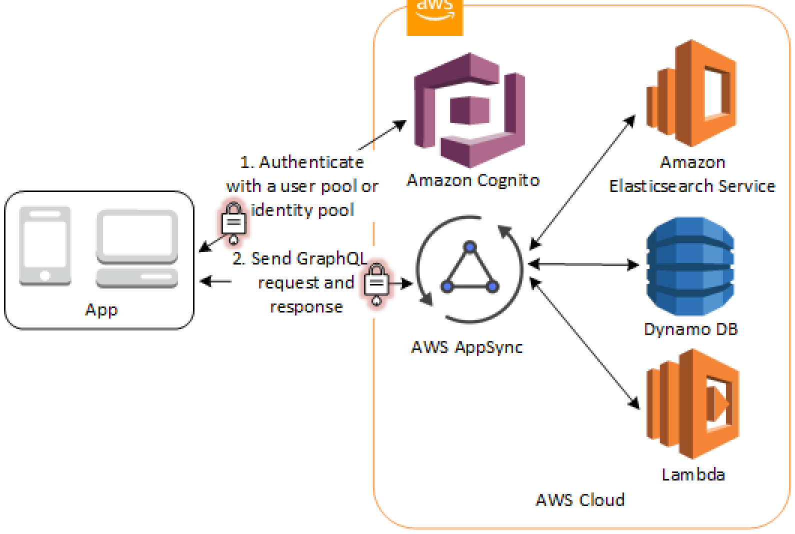
Authenticate with a Third Party and Access AWS Services with an Identity Pool

You can enable your users access to AWS services through an identity pool. An identity pool requires an IdP token from a user that's authenticated by a third-party identity provider (or nothing if it's an anonymous guest). In exchange, the identity pool grants temporary AWS credentials that you can use to access other AWS services. For more information, see Getting Started with Amazon Cognito Identity Pools (Federated Identities) (p. 187).


Social sign-in
OIDC
SAML
Custom
1. Authenticate and get token from a third-party identity provider (IdP)
App
2. Exchange IdP token for AWS credentials
Identity pool
3. Access AWS services with credentials
Other AWS services
AWS Cloud

Access AWS AppSync Resources with Amazon Cognito

You can grant your users access to AWS AppSync resources with tokens from a successful Amazon Cognito authentication (from a user pool or an identity pool). For more information, see Access AWS AppSync and Data Sources with User Pools or Federated Identities.


1. Authenticate with a user pool or identity pool
Amazon Cognito
Amazon Elasticsearch Service
App
2. Send GraphQL request and response
AWS AppSync
Dynamo DB
Lambda
AWS Cloud

Amazon Cognito Tutorials

The two main components of Amazon Cognito are user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your web and mobile app users. Identity pools provide AWS credentials to grant your users access to other AWS services.

Topics

Tutorial: Creating a User Pool

With a user pool, your users can sign in to your web or mobile app through Amazon Cognito.

To create a user pool

1. Go to the Amazon Cognito console. You may be prompted for your AWS credentials.
2. Choose **Manage User Pools**.
3. Choose **Create a user pool**.
4. Provide a name for your user pool and choose **Review defaults** to save the name.
5. On the **Review** page, choose **Create pool**.

Related Resources

For more information on user pools, see Amazon Cognito user pools (p. 19).

See also User Pool Authentication Flow (p. 306) and Using Tokens with User Pools (p. 149).

Tutorial: Creating an Identity Pool

With an identity pool, your users can obtain temporary AWS credentials to access AWS services, such as Amazon S3 and DynamoDB.

To create an identity pool

1. Go to the Amazon Cognito console. You may be prompted for your AWS credentials.
2. Choose **Manage Identity Pools**
3. Choose **Create new identity pool**.
4. Enter a name for your identity pool.
5. To enable unauthenticated identities select **Enable access to unauthenticated identities** from the **Unauthenticated identities** collapsible section.
6. Choose **Create Pool**.
7. You will be prompted for access to your AWS resources.

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.

8. Make a note of your identity pool Id number. You will use it to set up policies allowing your app users to access other AWS services such as Amazon Simple Storage Service or DynamoDB

Related Resources

For more information on identity pools, see Amazon Cognito Identity Pools (Federated Identities) (p. 187).

For an S3 example using an identity pool see Uploading Photos to Amazon S3 from a Browser.

Tutorial: Cleaning Up Your AWS Resources

To delete an identity pool

1. Go to the Amazon Cognito console. You may be prompted for your AWS credentials.
2. Choose **Manage Identity Pools**.
3. Choose the name of the identity pool that you want to delete. The **Dashboard page** for your identity pool appears.
4. In the top-right corner of the Dashboard page, choose **Edit identity pool**. The **Edit identity pool** page appears.
5. Scroll down and choose **Delete identity pool** to expand it.
6. Choose **Delete identity pool**.
7. Choose **Delete pool**.

To delete a user pool

1. Go to the Amazon Cognito console. You may be prompted for your AWS credentials.
2. **Manage User Pools**.
3. Choose the user pool you created in the previous step.
4. On the **Domain name** page under **App integration** select **Delete domain**.
5. Choose **Delete domain** when prompted to confirm.
6. Go to the **General Settings** page.
7. Select **Delete pool** in the upper right corner of the page.
8. Enter **delete** and choose **Delete pool** when prompted to confirm.

Integrating Amazon Cognito With Web and Mobile Apps

When new users discover your app, or when existing users return to it, their first tasks are to sign up or sign in. By integrating Amazon Cognito with your client code, you connect your app to backend AWS functionality that aids authentication and authorization workflows. Your app will use the Amazon Cognito API to, for example, create new users in your user pool, retrieve user pool tokens, and obtain temporary credentials from your identity pool. To integrate Amazon Cognito with your web or mobile app, use the SDKs and libraries that the AWS Amplify framework provides.

Amazon Cognito Authentication With the AWS Amplify Framework

AWS Amplify provides services and libraries for web and mobile developers. With AWS Amplify, you can build apps that integrate with backend environments that are composed of AWS services. To provision your backend environment, and to integrate AWS services with your client code, you use the AWS Amplify *framework*. The framework provides an interactive command line interface (CLI) that helps you configure AWS resources for features that are organized into categories, including analytics, storage, and authentication, among many others. The framework also provides high-level SDKs and libraries for web and mobile platforms, including iOS, Android, and JavaScript. Supported JavaScript frameworks include React, React Native, Angular, Ionic, and Vue. Each of the SDKs and libraries include authentication operations that you can use to implement the authentication workflows that Amazon Cognito drives.

To use the AWS Amplify framework to add authentication to your app, see the AWS Amplify authorization documentation for your platform:

- AWS Amplify authentication for JavaScript
- AWS Amplify authentication for iOS
- AWS Amplify authentication for Android

Multi-tenant application best practices

Amazon Cognito user pools can be used to secure small multi-tenant applications where the number of tenants and expected volume align with the related Amazon Cognito service quota. A common use case of multi-tenant design is running workloads to support testing multiple versions of an application. Multi-tenant design is also useful for testing a single application with different datasets, which allows full use of your cluster resources.

Note
Amazon Cognito Quotas are applied per AWS account and Region. These quotas are shared across all tenants in your application. Review the Amazon Cognito service quotas and make sure that the quota meets the expected volume and the expected number of tenants in your application.

You have four ways to secure multi-tenant applications: user pools, application clients, groups, or custom attributes.

Topics

User pool-based multi-tenancy

With this design, you can create a user pool for each tenant in your application. This approach provides maximum isolation for each tenant and allows you to implement different configurations for each tenant. Tenant isolation by user pool allows you flexibility in user-to-tenant mapping. It also allows multiple profiles for the same user. However, each user has to sign up individually for each tenant they have access to. Using this approach allows you to set up hosted UI for each tenant independently and redirect users to their tenant-specific instance of your application. This approach also allows easier integration with backend services like API Gateway. We recommend this approach in the following scenarios.

- Your application has different configurations for each tenant. For example, data residency requirements, password policy, and MFA configurations can be different for each tenant.
- Your application has complex user-to-tenant role mapping. For example, a single user could be a "Student" in tenant A and the same user could also be a "Teacher" in tenant B.
- Your application uses the default Amazon Cognito hosted UI as the primary authentication method for native users. (Native users are those that have been created in the user pool with user name and password).
- Your application has a silo multi-tenant application where each tenant gets a full instance of your application infrastructure for their usage.

Effort level

The development and operation effort to use this approach is high. You need to build tenant onboarding and administration components into your application that uses Amazon Cognito API operations and automation tools. These components are necessary to create the required resources for each tenant. You also need to implement a tenant-matching user interface. In addition, you must add logic to your application that allows users to sign up and sign in to their corresponding tenant's user pool.

Application client-based multi-tenancy

With application client-based multi-tenancy, you can map the same user to multiple tenants without the need to recreate a user's profile. You can create an application client for each tenant and enable the tenant external IdP as the only allowed identity provider for this application client. For more information see, Configuring a user pool app client.

Application client-based multi-tenancy requires additional considerations for user name, password, and more when you use hosted UI to authenticate users with native accounts. When the hosted UI is in use, a session cookie is created to maintain the session for the authenticated user. The session cookie also provides SSO between application clients in the same user pool. This approach can be used in the following scenarios:

- Your application has the same configurations across all tenants. For example, data residency and password policy are the same across all tenants.
- Your application has a one-to-many mapping between user and tenants. For example, a single user could have access to multiple tenants using the same profile.
- You have a federation-only multi-tenant application where tenants will always use an external IdP to sign in to your application.
- You have a B2B multi-tenant application and tenants backend services will use client-credentials grant to access your services. In this case, you can create application client for each tenant and share the client-id and secret with tenant backend service for machine-to-machine authentication.

Effort level

The development effort to use this approach is high. You need to implement tenant-matching logic and a user interface to match a user to the application client for their tenant.

Group-based multi-tenancy

With group-based multi-tenancy, you can associate an Amazon Cognito user pool group with a tenant. That way you can use additional functionality through role-based access control (RBAC). For more information see, Role-based access control.

Custom attribute-based multi-tenancy

With custom attribute-based multi-tenancy, you can store tenant identification data like `tenant_id` as a custom attribute in a user's profile. You then handle all multi-tenancy logic in your application and backend services. This approach allows you to use a unified sign-up and sign-in experience for all users. You can also identify the user's tenant in your application by checking this custom attribute.

Multi-tenancy security recommendations

The following recommendations can help make your application more secure.

- Avoid using an unverified email address to authorize user access to a tenant based on domain match. Email addresses and phone numbers shouldn’t be trusted unless they are verified by your application or a proof of verification is given by the external IdP. For more details on setting these permissions, see Attribute Permissions and Scopes.
- Make sure that user profile attributes used to identify tenants are immutable or mutable attributes that can be changed by administrators. Application clients should have read-only access to these attributes.
- Ensure you have 1:1 mapping between external IdP and application client to prevent unauthorized cross-tenant access. This could happen if a user has a valid Amazon Cognito session cookie and their external IdP is allowed on multiple application clients.
- When implementing tenant-matching and authorization logic in your application, ensure that the criteria used to authorize user access to the tenants can't be modified by users themselves. You should also ensure that user access can't be modified by the tenant identity provider administrators (if an external IdP is being used for federation).

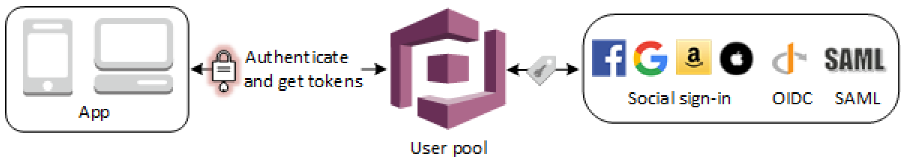
Amazon Cognito user pools

A user pool is a user directory in Amazon Cognito. With a user pool, your users can sign in to your web or mobile app through Amazon Cognito. Your users can also sign in through social identity providers like Google, Facebook, Amazon, or Apple, and through SAML identity providers. Whether your users sign in directly or through a third party, all members of the user pool have a directory profile that you can access through a Software Development Kit (SDK).

User pools provide:

- Sign-up and sign-in services.
- A built-in, customizable web UI to sign in users.
- Social sign-in with Facebook, Google, Login with Amazon, and Sign in with Apple, as well as sign-in with SAML identity providers from your user pool.
- User directory management and user profiles.
- Security features such as multi-factor authentication (MFA), checks for compromised credentials, account takeover protection, and phone and email verification.
- Customized workflows and user migration through AWS Lambda triggers.

After successfully authenticating a user, Amazon Cognito issues JSON web tokens (JWT) that you can use to secure and authorize access to your own APIs, or exchange for AWS credentials.



Amazon Cognito provides token handling through the Amazon Cognito user pools Identity SDKs for JavaScript, Android, and iOS. See Getting Started with User Pools (p. 20) and Using Tokens with User Pools (p. 149).

The two main components of Amazon Cognito are user pools and identity pools. Identity pools provide AWS credentials to grant your users access to other AWS services. To enable users in your user pool to access AWS resources, you can configure an identity pool to exchange user pool tokens for AWS credentials. For more information see Accessing AWS Services Using an Identity Pool After Sign-in (p. 160) and Getting Started with Amazon Cognito Identity Pools (Federated Identities) (p. 187).

Topics

Getting Started with User Pools

These steps describe setting up and configuring a user pool with the Amazon Cognito console. For a guide for where to start with Amazon Cognito, see Getting Started with Amazon Cognito (p. 5).

Topics

Prerequisite: Sign Up for an AWS Account

To use Amazon Cognito, 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://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

Next Step

Step 1. Create a User Pool (p. 20)

Step 1. Create a User Pool

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

To create a user pool

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. Choose **Manage User Pools**.
3. In the top-right corner of the page, choose **Create a user pool**.
4. Provide a name for your user pool, and choose **Review defaults** to save the name.
5. In the top-left corner of the page, choose **Attributes**, choose **Email address or phone number** and **Allow email addresses**, and then choose **Next step** to save.

   Note
   We recommend that you enable case insensitivity on the `username` attribute before you create your user pool. For example, when this option is selected, users will be able to sign in using either "username" or "Username". Enabling this option also enables both `preferred_username` and `email` alias to be case insensitive, in addition to the `username` attribute. For more information, see CreateUserPool in the *Amazon Cognito user pools API Reference*.
6. In the left navigation menu, choose **Review**.

7. Review the user pool information and make any necessary changes. When the information is correct, choose **Create pool**.

Next Step

Step 2. Add an App to Enable the Hosted Web UI (p. 21)

Step 2. Add an App to Enable the Hosted Web UI

After you create a user pool, you can create an app to use the built-in webpages for signing up and signing in your users.

To create an app in your user pool

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. Choose **Manage User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. On the navigation bar on the left-side of the page, choose **App clients** under **General settings**.
5. Choose **Add an app client**.
6. Give your app a name.
7. Clear the option **Generate client secret** for the purposes of this getting started exercise, as it would not be secure to send it on the URL using client-side JavaScript. The client secret is used by applications that have a server-side component that can secure the client secret.
8. Choose **Create app client**.
9. Note the **App client ID**.
10. Choose **Return to pool details**.
11. Choose **App client settings** from the navigation bar on the left-side of the console page.
12. Select **Cognito User Pool** as one of the **Enabled Identity Providers**.

    Note
    To sign in with external identity providers (IdPs) such as Facebook, Amazon, Google, and Apple, as well as through OpenID Connect (OIDC) or SAML IdPs, first configure them as described next, and then return to the **App client settings** page to enable them.
13. Enter a callback URL for the Amazon Cognito authorization server to call after users are authenticated. For a web app, the URL should start with `https://`, such as https://www.example.com.

    For an iOS or Android app, you can use a callback URL such as `myapp://`.
14. Enter a Sign out URL.
15. Select **Authorization code grant** to return an authorization code that is then exchanged for user pool tokens. Because the tokens are never exposed directly to an end user, they are less likely to become compromised. However, a custom application is required on the backend to exchange the authorization code for user pool tokens. For security reasons, we recommend that you use the authorization code grant flow, together with Proof Key for Code Exchange (PKCE), for mobile apps.
16. Under **Allowed OAuth Flows**, select **Implicit grant** to have user pool JSON web tokens (JWT) returned to you from Amazon Cognito. You can use this flow when there's no backend available to exchange an authorization code for tokens. It's also helpful for debugging tokens.

    Note
    You can enable both the **Authorization code grant** and the **Implicit code grant**, and then use each grant as needed.
17. Unless you specifically want to exclude one, select the check boxes for all of the **Allowed OAuth scopes**.

> **Note**
> Select **Client credentials** only if your app needs to request access tokens on its own behalf, not on behalf of a user.

18. Choose **Save changes**.
19. On the **Domain name** page, type a domain prefix that's available.
20. Make a note of the complete domain address.
21. Choose **Save changes**.

To view your sign-in page

You can view the hosted UI sign-in webpage with the following URL. Note the `response_type`. In this case, **response_type=code** for the authorization code grant.

```
https://your_domain/login?
response_type=code&client_id=your_app_client_id&redirect_uri=your_callback_url
```

You can view the hosted UI sign-in webpage with the following URL for the implicit code grant where **response_type=token**. After a successful sign-in, Amazon Cognito returns user pool tokens to your web browser's address bar.

```
https://your_domain/login?
response_type=token&client_id=your_app_client_id&redirect_uri=your_callback_url
```

You can find the JSON web token (JWT) identity token after the `#idtoken=` parameter in the response.

Here's a sample response from an implicit grant request. Your identity token string will be much longer.

```
https://www.example.com/
#id_token=123456789tokens123456789&expires_in=3600&token_type=Bearer
```

You can decode and verify user pool tokens using AWS Lambda, see Decode and verify Amazon Cognito JWT tokens on the AWS GitHub website.

Amazon Cognito user pools tokens are signed using an RS256 algorithm.

You might have to wait a minute to refresh your browser before changes you made in the console appear.

Your domain is shown on the **Domain name** page. Your app client ID and callback URL are shown on the **General settings** page.

Next Step

Step 3. Add Social Sign-in to a User Pool (Optional) (p. 22)

Step 3. Add Social Sign-in to a User Pool (Optional)

You can enable your app users to sign in through a social identity provider (IdP) such as Facebook, Google, Amazon, and Apple. Whether your users sign in directly or through a third party, all users have a profile in the user pool. Skip this step if you don't want to add sign in through a social sign-in identity provider.

Step 1: Register with a Social IdP

Before you create a social IdP with Amazon Cognito, you must register your application with the social IdP to receive a client ID and client secret.

To register an app with Facebook

1. Create a developer account with Facebook.
2. Sign in with your Facebook credentials.
3. From the **My Apps** menu, choose **Create New App**.
4. Give your Facebook app a name and choose **Create App ID**.
5. On the left navigation bar, choose **Settings** and then **Basic**.
6. Note the **App ID** and the **App Secret**. You will use them in the next section.
7. Choose **+ Add Platform** from the bottom of the page.
8. Choose **Website**.
9. Under **Website**, type your user pool domain with the /oauth2/idpresponse endpoint into **Site URL**.

   ```
   https://<your-user-pool-domain>/oauth2/idpresponse
   ```
10. Choose **Save changes**.
11. Type your user pool domain into **App Domains**.

    ```
    https://<your-user-pool-domain>
    ```
12. Choose **Save changes**.
13. From the navigation bar choose **Products** and then **Set up** from **Facebook Login**.
14. From the navigation bar choose **Facebook Login** and then **Settings**.

    Type your redirect URL into **Valid OAuth Redirect URIs**. It will consist of your user pool domain with the /oauth2/idpresponse endpoint.

    ```
    https://<your-user-pool-domain>/oauth2/idpresponse
    ```
15. Choose **Save changes**.

To register an app with Amazon

1. Create a developer account with Amazon.
2. Sign in with your Amazon credentials.
3. You need to create an Amazon security profile to receive the Amazon client ID and client secret.

   Choose **Apps and Services** from navigation bar at the top of the page and then choose **Login with Amazon**.
4. Choose **Create a Security Profile**.
5. Type in a **Security Profile Name**, a **Security Profile Description**, and a **Consent Privacy Notice URL**.
6. Choose **Save**.
7. Choose **Client ID** and **Client Secret** to show the client ID and secret. You will use them in the next section.

8. Hover over the gear and choose **Web Settings**, and then choose **Edit**.
9. Type your user pool domain into **Allowed Origins**.

```
https://<your-user-pool-domain>
```

10. Type your user pool domain with the **/oauth2/idpresponse** endpoint into **Allowed Return URLs**.

```
https://<your-user-pool-domain>/oauth2/idpresponse
```

11. Choose **Save**.

To register an app with Google

1. Create a developer account with Google.
2. Sign in with your Google credentials.
3. Choose **CONFIGURE A PROJECT**.
4. Type in a project name and choose **NEXT**.
5. Type in your product name and choose **NEXT**.
6. Choose **Web browser** from the **Where are you calling from?** drop-down list.
7. Type your user pool domain into **Authorized JavaScript origins**.

```
https://<your-user-pool-domain>
```

8. Choose **CREATE**. You will not use the **Client ID** and **Client Secret** from this step.
9. Choose **DONE**.
10. Sign in to the Google Console.
11. On the left navigation bar, choose **Credentials**.
12. Create your OAuth 2.0 credentials by choosing **OAuth client ID** from the **Create credentials** drop-down list.
13. Choose **Web application**.
14. Type your user pool domain into **Authorized JavaScript origins**.

```
https://<your-user-pool-domain>
```

15. Type your user pool domain with the **/oauth2/idpresponse** endpoint into **Authorized Redirect URIs**.

```
https://<your-user-pool-domain>/oauth2/idpresponse
```

16. Choose **Create** twice.
17. Note the **OAuth client ID** and **client secret**. You will need them for the next section.
18. Choose **OK**.

To register an app with Apple

1. Create a developer account with Apple.
2. Sign in with your Apple credentials.

3. On the left navigation bar, choose **Certificates, IDs & Profiles**.
4. On the left navigation bar, choose **Identifiers**.
5. On the **Identifiers** page, choose the **+** icon.
6. On the **Register a New Identifier** page, choose **App IDs**, and then choose **Continue**.
7. On the **Register an App ID** page, do the following:

    1. Under **Description**, type a description.
    2. Under **App ID Prefix**, type an identifier. Make a note of the value under **App ID Prefix** as you will need this value after you choose Apple as your identity provider in Step 2: Add a Social IdP to Your User Pool (p. 50).
    3. Under **Capabilities**, choose **Sign In with Apple**, and then choose **Edit**.
    4. On the **Sign in with Apple: App ID Configuration** page, select the appropriate setting for you app, and then choose **Save**.
    5. Choose **Continue**.
8. On the **Confirm your App ID** page, choose **Register**.
9. On the **Identifiers** page, hover over **App IDs** on the right side of the page, choose **Services IDs**, and then choose the **+** icon.
10. On the **Register a New Identifier** page, choose **Services IDs**, and then choose **Continue**.
11. On the **Register a Services ID** page, do the following:

    1. Under **Description**, type a description.
    2. Under **Identifier**, type an identifier. Make a note of this Services ID as you will need this value after you choose Apple as your identity provider in Step 2: Add a Social IdP to Your User Pool (p. 50).
    3. Select **Sign In with Apple**, and then choose **Configure**.
    4. On the **Web Authentication Configuration** page, choose a **Primary App ID**. Under **Web Domain**, type your user pool domain. Under **Return URLs**, type your user pool domain and include the /oauth2/idpresponse endpoint. For example:

    ```
    https://<your-user-pool-domain>/oauth2/idpresponse
    ```

    5. Choose **Add**, and then **Save**. You do not need to verify the domain.
    6. Choose **Continue**, and then choose **Register**.
12. On the left navigation bar, choose **Keys**.
13. On the **Keys** page, choose the **+** icon.
14. On the **Register a New Key** page, do the following:

    1. Under **Key Name**, type a key name.
    2. Choose **Sign In with Apple**, and then choose **Configure**.
    3. On the **Configure Key** page, choose a **Primary App ID**, and then choose **Save**.
    4. Choose **Continue**, and then choose **Register**.
15. On the **Download Your Key** page, choose **Download** to download the private key, and then choose **Done**. You will need this private key and the **Key ID** value shown on this page after you choose Apple as your identity provider in Step 2: Add a Social IdP to Your User Pool (p. 50).

Step 2: Add a Social IdP to Your User Pool

In this section, you configure a social IdP in your user pool using the client ID and client secret from the previous section.

To configure a user pool social identity provider with the AWS Management Console

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. Choose **Manage your User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. On the left navigation bar, choose **Identity providers**.
5. Choose a social identity provider: **Facebook**, **Google**, **Login with Amazon**, or **Apple**.
6. For Google and Login with Amazon, type the app client ID and app client secret that you received from the social identity provider in the previous section. For Facebook, type the app client ID, app client secret that you received from the social identity provider in the previous section, and choose an API version. We recommend choosing the highest available possible version as each Facebook API version has a lifecycle and a deprecation date for example, version 2.12. You can change the API version post creation if you encounter any issues. The Facebook scopes and attributes may vary with each API version, so we recommend testing your integration." For Sign in with Apple, provide the Services ID, Team ID, Key ID, and private key that you received in the previous section.
7. Type 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 Google and Login with Amazon, they should be separated by spaces. For Sign in with Apple, select the check boxes for the scopes you want access to.

Social identity provider	Example scopes
Facebook	`public_profile, email`
Google	`profile email openid`
Login with Amazon	`profile postal_code`
Sign in with Apple	`email name`

   Your app user is asked to consent to providing these attributes to your app. For more information about their scopes, see the documentation from Google, Facebook, Login with Amazon, or Sign in with Apple.

   In the case of Sign in with Apple, the following are user scenarios where scopes might not be returned:

   - An end user encounters failures after leaving Apple's sign in page (can be from Internal failures within Cognito or anything written by the developer)
   - The service id identifier is used across user pools and/or other authentication services
   - A developer adds additional scopes after the end user has signed in before (no new information is retrieved)
   - A developer deletes the user and then the user signs in again without removing the app from their Apple ID profile
8. Choose **Enable** for the social identity provider that you are configuring.
9. Choose **App client settings** from the navigation bar.
10. Select your social identity provider as one of the **Enabled Identity Providers** for your user pool app.
11. Type your callback URL into **Callback URL(s)** for your user pool app. This is the URL of the page where your user will be redirected after a successful authentication.

    ```
    https://www.example.com
    ```

12. Choose **Save changes**.
13. On the **Attribute mapping** tab, add mappings for at least the required attributes, typically `email`, as follows:

    a. Select the check box to choose the Facebook, Google, or Amazon attribute name. You can also type the names of additional attributes that are not listed in the Amazon Cognito console.
    b. Choose the destination user pool attribute from the drop-down list.
    c. Choose **Save changes**.
    d. Choose **Go to summary**.

Step 3: Test Your Social IdP Configuration

You can create a login URL by using the elements from the previous two sections. Use it to test your social IdP configuration.

```
https://<your_user_pool_domain>/login?
response_type=code&client_id=<your_client_id>&redirect_uri=https://www.example.com
```

You can find your domain on the user pool **Domain name** console page. The client_id is on the **App client settings** page. Use your callback URL for the **redirect_uri** parameter. This is the URL of the page where your user will be redirected after a successful authentication.

> **Note**
> Requests that are not completed within 5 minutes will be cancelled, redirected to the login page, and then display a `Something went wrong` error message.

Next Step

Step 4. Add Sign-in with a SAML Identity Provider to a User Pool (Optional) (p. 27)

Step 4. Add Sign-in with a SAML Identity Provider to a User Pool (Optional)

You can enable your app users to sign in through a SAML identity provider (IdP). Whether your users sign in directly or through a third party, all users have a profile in the user pool. Skip this step if you don't want to add sign in through a SAML identity provider.

You need to update your SAML identity provider and configure your user pool. See the documentation for your SAML identity provider for information about how to add your user pool as a relying party or application for your SAML 2.0 identity provider.

You also need to provide an assertion consumer endpoint to your SAML identity provider. Configure this endpoint for SAML 2.0 POST binding in your SAML identity provider:

```
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.

For some SAML identity providers, you also need to provide the SP `urn` / Audience URI / SP Entity ID, in the form:

```
urn:amazon:cognito:sp:<yourUserPoolID>
```

You can find your user pool ID on the **General settings** tab in the Amazon Cognito console.

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

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.

To configure a SAML 2.0 identity provider in your user pool

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. **Manage User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. On the left navigation bar, choose **Identity providers**.
5. Choose **SAML** to open the SAML dialog.
6. Under **Metadata document** upload a metadata document from your SAML IdP. You can also enter a URL that points to the metadata document. For more information, see Integrating Third-Party SAML Identity Providers with Amazon Cognito User Pools (p. 58).

    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.
7. Enter your SAML **Provider name**. For more information on SAML naming see Choosing SAML Identity Provider Names (p. 54).
8. Enter any optional SAML **Identifiers** you want to use.
9. Select **Enable IdP sign out flow** if you want your user to be logged out from the SAML IdP when logging out from Amazon Cognito.

    Enabling this flow sends a signed logout request to the SAML IdP when the Logout-endpointis called.

    Configure this endpoint for consuming logout responses from your IdP. This endpoint uses post binding.

    ```
    https://<yourDomainPrefix>.auth.<region>.amazoncognito.com/saml2/logout
    ```

    Note
    If this option is selected and your SAML identity provider expects a signed logout request, you will also need to configure the signing certificate provided by Amazon Cognito with your SAML IdP.
    The SAML IdP will process the signed logout request and logout your user from the Amazon Cognito session.
10. Choose **Create provider**.
11. On the **Attribute mapping** tab, add mappings for at least the required attributes, typically `email`, as follows:

    a. Type the SAML attribute name as it appears in the SAML assertion from your identity provider. If your identity provider offers sample SAML assertions, that might help you to find the name.

Some identity providers use simple names, such as `email`, while others use names similar to this:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress
```

   b. Choose the destination user pool attribute from the drop-down list.

12. Choose **Save changes**.
13. Choose **Go to summary**.

For more information, see Adding SAML Identity Providers to a User Pool (p. 52).

Next Steps

Now that you've created a user pool, here are some places to go next.

Dive deep into these user pool features:

- Customizing the Built-in Sign-in and Sign-up Webpages (p. 40)
- Adding Multi-Factor Authentication (MFA) to a User Pool (p. 323)
- Adding Advanced Security to a User Pool (p. 327)
- Customizing User Pool Workflows with Lambda Triggers (p. 67)
- Using Amazon Pinpoint Analytics with Amazon Cognito User Pools (p. 114)

For an overview of Amazon Cognito authentication and authorization cases, see Common Amazon Cognito Scenarios (p. 9).

To access other AWS services after a successful user pool authentication, see Accessing AWS Services Using an Identity Pool After Sign-in (p. 160).

In addition to the AWS Management Console and the user pool SDKs mentioned previously in this section, you can also interact with your user pool profiles by using the AWS Command Line Interface.

Using the Amazon Cognito Hosted UI for Sign-Up and Sign-In

Amazon Cognito Hosted UI provides you an OAuth 2.0 compliant authorization server. It provides default implementation of end user flows such as registration, authentication etc. You can customize the user flows such as addition of Multi Factor Authentication (MFA) by simply changing your user pool configuration. Your application will redirect to Hosted UI and it will handle the user flows. The user experience can be customized by providing brand specific logos and changing the look and feel. Amazon Cognito Hosted UI also allows you to easily add ability for end users to sign in with social providers (Facebook, LoginWithAmazon, Google and Apple), any OpenID Connect (OIDC) compliant and SAML providers.

Topics

- Defining Resource Servers for Your User Pool (p. 44)

Setting Up the Hosted UI with AWS Amplify

If you use AWS Amplify to add authentication to your web or mobile app, you can set up your hosted UI by using the command line interface (CLI) and libraries in the AWS Amplify framework. To add authentication to your app, you use the AWS Amplify CLI to add the Auth category to your project. Then, in your client code, you use the AWS Amplify libraries to authenticate users with your Amazon Cognito user pool.

You can display a pre-built hosted UI, or you can federate users through an OAuth 2.0 endpoint that redirects to a social sign-in provider, such as Facebook, Google, Amazon, or Apple. After a user successfully authenticates with the social provider, AWS Amplify creates a new user in your user pool if needed, and it provides the user's OIDC token to you app.

For more information, see the AWS Amplify framework documentation for your app platform:

- AWS Amplify authentication for JavaScript.
- AWS Amplify authentication for iOS.
- AWS Amplify authentication for Android.

Setting Up the Hosted UI with the Amazon Cognito Console

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. Choose **Manage User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. On the navigation bar on the left-side of the page, choose **App clients** under **General settings**.
5. Choose **Add an app client**.
6. Enter your app name.
7. Unless required by your authorization flow, clear the option **Generate client secret**. The client secret is used by applications that have a server-side component that can secure the client secret.
8. (Optional) Change the token expiration settings.
9. Select **Auth Flows Configuration** options.
10. Choose a **Security configuration**. We recommend you select **Enabled**.
11. (Optional) Choose **Set attribute read and write permissions**.
12. Choose **Create app client**.
13. Note the **App client id**.
14. Choose **Return to pool details**.

Configure the app

1. Choose **App client settings** from the navigation bar on the left-side of the console page.
2. Select **Cognito User Pool** as one of the **Enabled Identity Providers**.

   Note
   To sign in with external identity providers (IdPs), such as Facebook, Amazon, Google, or Apple as well as through OpenID Connect (OIDC) or SAML IdPs, first configure them as described next, and then return to the **App client settings** page to enable them.

3. Enter **Callback URL(s)**. A callback URL indicates where the user is to be redirected after a successful sign-in.
4. Enter **Sign out URL(s)**. A sign-out URL indicates where your user is to be redirected after signing out.
5. Select **Authorization code grant** to return an authorization code that is then exchanged for user pool tokens. Because the tokens are never exposed directly to an end user, they are less likely to become compromised. However, a custom application is required on the backend to exchange the authorization code for user pool tokens. For security reasons, we recommend that you use the authorization code grant flow, together with Proof Key for Code Exchange (PKCE), for mobile apps.
6. Select **Implicit grant** to have user pool JSON web tokens (JWT) returned to you from Amazon Cognito. You can use this flow when there's no backend available to exchange an authorization code for tokens. It's also helpful for debugging tokens.
7. You can enable both the **Authorization code grant** and the **Implicit code grant**, and then use each grant as needed.
8. Unless you specifically want to exclude one, select the check boxes for all of the **Allowed OAuth scopes**.
9. Select **Client credentials** only if your app needs to request access tokens on its own behalf, not on behalf of a user.
10. Choose **Save changes**.

Configure a domain

1. Select **Choose domain name**.
2. On the **Domain name** page, type a domain prefix and check availability or enter your own domain.
3. Make a note of the complete domain address.
4. Choose **Save changes**.

To view your sign-in page

You can view the hosted UI sign-in webpage with the following URL. Note the `response_type`. In this case, **response_type=code** for the authorization code grant.

```
https://<your_domain>/login?
response_type=code&client_id=<your_app_client_id>&redirect_uri=<your_callback_url>
```

You can view the hosted UI sign-in webpage with the following URL for the implicit code grant where **response_type=token**. After a successful sign-in, Amazon Cognito returns user pool tokens to your web browser's address bar.

```
https://<your_domain>/login?
response_type=token&client_id=<your_app_client_id>&redirect_uri=<your_callback_url>
```

You can find the JSON web token (JWT) identity token after the `#idtoken=` parameter in the response.

Here's a sample response from an implicit grant request. Your identity token string will be much longer.

```
https://www.example.com/
#id_token=123456789tokens123456789&expires_in=3600&token_type=Bearer
```

You can decode and verify user pool tokens using AWS Lambda, see Decode and verify Amazon Cognito JWT tokens on the AWS GitHub website.

Amazon Cognito user pools tokens are signed using an RS256 algorithm.

You might have to wait a minute to refresh your browser before changes you made in the console appear.

Your domain is shown on the **Domain name** page. Your app client ID and callback URL are shown on the **App client settings** page.

> **Note**
> The Amazon Cognito hosted sign-in web page does not support the custom authentication flow.

Configuring a User Pool App Client

After you create a user pool, you can configure an app client to use the built-in webpages for signing up and signing in your users. For terminology, see App Client Settings Terminology (p. 34).

To Configure an App Client (AWS Management Console)

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. Choose **Manage User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. On the navigation bar on the left-side of the page, choose **App clients** under **General settings**.
5. Choose **Add an app client**.
6. Enter your app name.
7. Unless required by your authorization flow, clear the option **Generate client secret**. The client secret is used by applications that have a server-side component that can secure the client secret.
8. (Optional) Change the token expiration settings.
9. Select **Auth Flows Configuration** options.
10. Choose a **Security configuration**. We recommend you select **Enabled**.
11. (Optional) Choose **Set attribute read and write permissions**.
12. Choose **Create app client**.
13. Note the **App client id**.
14. Choose **Return to pool details**.

To Configure an App Client (AWS CLI and AWS API)

You can use the AWS CLI to update, create, describe, and delete your user pool app client.

Replace "MyUserPoolID" and "MyAppClientID" with your user pool and app client ID values in these examples. Likewise, your parameter values might be different than those used in these examples.

> **Note**
> Use JSON format for callback and logout URLs to prevent the CLI from treating them as remote parameter files:
> --callback-urls "["https://example.com"]"
> --logout-urls "["https://example.com"]"

Updating a User Pool App Client (AWS CLI and AWS API)

```
aws cognito-idp update-user-pool-client --user-pool-id  "MyUserPoolID" --client-id
 "MyAppClientID" --allowed-o-auth-flows-user-pool-client --allowed-o-auth-flows "code"
 "implicit" --allowed-o-auth-scopes "openid" --callback-urls "["https://example.com"]" --
supported-identity-providers "["MySAMLIdP", "LoginWithAmazon"]"
```

If the command is successful, the AWS CLI returns a confirmation:

```
{
    "UserPoolClient": {
        "ClientId": "MyClientID",
        "SupportedIdentityProviders": [
            "LoginWithAmazon",
            "MySAMLIdP"
        ],
        "CallbackURLs": [
            "https://example.com"
        ],
        "AllowedOAuthScopes": [
            "openid"
        ],
        "ClientName": "Example",
        "AllowedOAuthFlows": [
            "implicit",
            "code"
        ],
        "RefreshTokenValidity": 30,
        "CreationDate": 1524628110.29,
        "AllowedOAuthFlowsUserPoolClient": true,
        "UserPoolId": "MyUserPoolID",
        "LastModifiedDate": 1530055177.553
    }
}
```

See the AWS CLI command reference for more information: update-user-pool-client.

AWS API: UpdateUserPoolClient

Creating a User Pool App Client (AWS CLI and AWS API)

```
aws cognito-idp create-user-pool-client --user-pool-id MyUserPoolID --client-name myApp
```

See the AWS CLI command reference for more information: create-user-pool-client

AWS API: CreateUserPoolClient

Getting Information about a User Pool App Client (AWS CLI and AWS API)

```
aws cognito-idp describe-user-pool-client --user-pool-id MyUserPoolID --client-id
 MyClientID
```

See the AWS CLI command reference for more information: describe-user-pool-client.

AWS API: DescribeUserPoolClient

Listing All App Client Information in a User Pool (AWS CLI and AWS API)

```
aws cognito-idp list-user-pool-clients --user-pool-id "MyUserPoolID" --max-results 3
```

See the AWS CLI command reference for more information: list-user-pool-clients.

AWS API: ListUserPoolClients

Deleting a User Pool App Client (AWS CLI and AWS API)

```
aws cognito-idp delete-user-pool-client --user-pool-id "MyUserPoolID" --client-id
 "MyAppClientID"
```

See the AWS CLI command reference for more information: delete-user-pool-client

AWS API: DeleteUserPoolClient

App Client Settings Terminology

The following terms and definitions can help you with configuring your app client.

Enabled Identity Providers

You can choose your identity provider (IDP) to authenticate your users. This service can be performed by your user pool, or by a third party such as Facebook. Before you can use an IdP, you need to enable it. You can enable multiple IdPs, but you must enable at least one. For more information on using external IdPs see Adding User Pool Sign-in Through a Third Party (p. 46).

Callback URL(s)

A callback URL indicates where the user is to be redirected after a successful sign-in. Choose at least one callback URL, and it should:

- Be an absolute URI.
- Be pre-registered with a client.
- Not include a fragment component.

See OAuth 2.0 - Redirection Endpoint.

Amazon Cognito requires `HTTPS` over `HTTP` except for `http://localhost` for testing purposes only.

App callback URLs such as `myapp://example` are also supported.

Sign out URL(s)

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

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. 358). Because the tokens are never exposed directly to an end user, they are less likely to become compromised. However, a custom application is required on the backend to exchange the authorization code for user pool tokens.

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. 353). Choose this flow if your app cannot initiate the **Authorization code grant** flow. For more information, see the OAuth 2.0 specification.

You can enable both the **Authorization code grant** and the **Implicit code grant**, and then use each grant as needed.

The **Client credentials** flow is used in machine-to-machine communications. With it you can request an access token to access your own resources. Use this flow when your app is requesting the token on its own behalf, not on behalf of a user.

Note
Since the client credentials flow is not used on behalf of a user, only custom scopes can be used with this flow. A custom scope is one that you define for your own resource server. See Defining Resource Servers for Your User Pool (p. 44).

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. This scope can only be requested with the `openid` scope.

Allowed Custom Scopes

A custom scope is one that you define for your own resource server in the **Resource Servers**. The format is *`resource-server-identifier`*/*`scope`*. See Defining Resource Servers for Your User Pool (p. 44).

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

Configuring a User Pool Domain

After setting up an app client, you can configure the address of your sign-up and sign-in webpages. You can use an Amazon Cognito hosted domain and choose an available domain prefix, or you can use your own web address as a custom domain.

To add an app client and an Amazon Cognito hosted domain with the AWS Management Console, see Adding an App to Enable the Hosted Web UI.

Note
You can't use the text `aws`, `amazon`, or `cognito`, in the domain prefix.

Topics

Using the Amazon Cognito Domain for the Hosted UI

After setting up an app client, you can configure the address of your sign-up and sign-in webpages. You can use the hosted Amazon Cognito domain with your own domain prefix.

To add an app client and an Amazon Cognito hosted domain with the AWS Management Console, see Adding an App to Enable the Hosted Web UI.

Topics

Prerequisites

Before you begin, you need:

- A user pool with an app client. For more information, see Getting Started with User Pools (p. 20).

Step 1: Configure a Hosted User Pool Domain

To configure a hosted user pool domain (AWS Management Console)

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

To configure a Amazon Cognito hosted 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. Type the domain prefix you want to use in the **Prefix domain name** box.
5. Choose **Check availability** to confirm the domain prefix is available.
6. Choose **Save changes**.

To configure a hosted user pool domain (AWS CLI and AWS API)

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

To configure a user pool 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

Step 2: Verify Your Sign-in Page

- Verify that the sign-in page is available from your Amazon Cognito hosted domain.

```
https://your_domain/login?
response_type=code&client_id=your_app_client_id&redirect_uri=your_callback_url
```

Your domain is shown on the **Domain name** page of the Amazon Cognito console. Your app client ID and callback URL are shown on the **App client settings** page.

Using Your Own Domain for the Hosted UI

After setting up an app client, you can configure your user pool with a custom domain for the Amazon Cognito hosted UI. With a custom domain, you enable your users to sign in to your application by using your own web address.

Topics

Adding a Custom Domain to a User Pool

To add a custom domain to your user pool, you specify the domain name in the Amazon Cognito console, and you provide a certificate you manage with AWS Certificate Manager (ACM). After you add your domain, Amazon Cognito provides an alias target, which you add to your DNS configuration.

Prerequisites

Before you begin, you need:

- A user pool with an app client. For more information, see Getting Started with User Pools (p. 20).
- A web domain that you own. Its root must have a valid **A record** in DNS. For more information see Domain Names.
- The ability to create a subdomain for your custom domain. We recommend using **auth** as the subdomain. For example: *`auth.example.com`*.

  Note
  You might need to obtain a new certificate for your custom domain's subdomain if you don't have a wildcard certificate.
- A Secure Sockets Layer (SSL) certificate managed by ACM.

  Note
  You must change the AWS region to US East (N. Virginia) in the ACM console before you request or import a certificate.
- To set up a custom domain name or to update its certificate, you must have permission to update Amazon CloudFront distributions. You can do so by attaching the following IAM policy statement to an IAM user, group, or role in your AWS account:

```
{
    "Version": "2012-10-17",
    "Statement": [
         {
            "Sid": "AllowCloudFrontUpdateDistribution",
            "Effect": "Allow",
            "Action": [
                "cloudfront:updateDistribution"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

See Using Identity-Based Policies (IAM Policies) for CloudFront.

Step 1: Enter Your Custom Domain Name

You can add your domain to your user pool by using the Amazon Cognito console or API.

To add your domain to your user pool (Amazon Cognito console)

1. Sign in to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. Sign in to the AWS Management Console and open the Amazon Cognito console at https://console.aws.amazon.com/cognito/home.
3. Choose **Manage User Pools**.
4. On the **Your User Pools** page, choose the user pool that you want to add your domain to.
5. On the navigation menu on the left, choose **Domain name**.
6. Under **Your own domain**, choose **Use your domain**.
7. For **Domain name**, enter your custom domain name. Your domain name can include only lowercase letters, numbers, and hyphens. Do not use a hyphen for the first or last character. Use periods to separate subdomain names.
8. For **AWS managed certificate**, choose the SSL certificate that you want to use for your domain. You can choose one of the certificates that you manage with ACM.

   If you don't have a certificate that is available to choose, you can use ACM to provision one. For more information, see Getting Started in the *AWS Certificate Manager User Guide*.
9. Choose **Save changes**.
10. Note the **Alias target**. Instead of an IP address or a domain name, the **Alias target** is an alias resource record set that points to an Amazon CloudFront distribution.

To add your domain to your user pool (Amazon Cognito API)

- Use the CreateUserPoolDomain action.

Step 2: Add an Alias Target and Subdomain

In this step, you set up an alias through your Domain Name Server (DNS) service provider that points back to the alias target from the previous step. If you are using Amazon Route 53 for DNS address resolution, choose the section **To add an alias target and subdomain using Route 53.**

To add an alias target and subdomain to your current DNS configuration

- If you aren't using Route 53 for DNS address resolution, then you need to have your DNS service provider add the alias target from the previous step as an alias for your user pool custom domain. Your DNS provider will also need to set up the subdomain for your custom domain.

To add an alias target and subdomain using Route 53

1. Sign in to the Route 53 console. You might be prompted for your AWS credentials.
2. If you don't have a hosted zone in Route 53, set one up. Otherwise, skip this step.

   a. Choose **Create Hosted Zone**.

   b. Choose your custom domain from the **Domain Name** list.
   c. For **Comment**, type an optional comment about the hosted zone.
   d. Choose **Create**.
3. On the **Hosted Zones** page, choose the name of your hosted zone.
4. Choose **Create Record Set**.
5. Select **Yes** for the **Alias** option.
6. Type the alias target name that you noted in a previous step into **Alias Target**.
7. Choose **Create**.

   Note
   Your new records take time to propagate to the Route 53 DNS servers. Currently, the only way to verify that changes have propagated is to use the Route 53 GetChange API method. Changes generally propagate to all Route 53 name servers within 60 seconds.
8. Add a subdomain in Route 53 by using the alias target.

   a. On the **Hosted Zones** page, choose the name of your hosted zone.
   b. Choose **Create Record Set** and enter the following values:

      i. For **Name**, type your preferred subdomain name. For example, if the subdomain you're attempting to create is *auth.example.com*, type *auth*.
      ii. For **Type**, choose **A - IPv4 address**.
      iii. Select **Yes** for the **Alias** option.
      iv. Type the alias target name that you noted in a previous step in **Alias Target**.
   c. Choose **Create**.

      Note
      Alternatively, you can create a new hosted zone to hold the records that are associated with your subdomain. You can also createa delegation set in the parent hosted zone that refers clients to the subdomain hosted zone. This method offers more flexibility when you're managing the hosted zones (for example, restricting who can edit the zones). You can only use this method for public hosted zones, because adding NS records to private hosted zones isn't currently supported. For more information, see Creating a subdomain for a domain hosted through Amazon Route 53.

Step 3: Verify Your Sign-in Page

- Verify that the sign-in page is available from your custom domain.

  Sign in with your custom domain and subdomain by entering this address into your browser. This is an example URL of a custom domain *example.com* with the subdomain *auth*:

```
https://auth.example.com/login?
response_type=code&client_id=<your_app_client_id>&redirect_uri=<your_callback_url>
```

Changing the SSL Certificate for Your Custom Domain

When necessary, you can use Amazon Cognito to change the certificate that you applied to your custom domain.

Usually, this is unnecessary following routine certificate renewal with ACM. When you renew your existing certificate in ACM, the ARN for your certificate remains the same, and your custom domain uses the new certificate automatically.

However, if you replace your existing certificate with a new one, ACM gives the new certificate a new ARN. To apply the new certificate to your custom domain, you must provide this ARN to Amazon Cognito.

After you provide your new certificate, Amazon Cognito requires up to 1 hour to distribute it to your custom domain.

Before you begin
Before you can change your certificate in Amazon Cognito, you must add your certificate to ACM. For more information, see Getting Started in the *AWS Certificate Manager User Guide*. When you add your certificate to ACM, you must choose US East (N. Virginia) as the AWS Region.

You can change your certificate by using the Amazon Cognito console or API.

To renew a certificate (Amazon Cognito console)

1. Sign in to the AWS Management Console and open the Amazon Cognito console at https://console.aws.amazon.com/cognito/home.
2. Choose **Manage User Pools**.
3. On the **Your User Pools** page, choose the user pool that you want to add your domain to.
4. On the navigation menu on the left, choose **Domain name**.
5. Under **Your own domain**, for **AWS managed certificate**, choose your new certificate.
6. Choose **Save changes**.

To renew a certificate (Amazon Cognito API)

- Use the UpdateUserPoolDomain action.

Customizing the Built-in Sign-in and Sign-up Webpages

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:
  - `background-customizable`

  - `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 a 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 **General settings** tab.

To specify app UI customization settings

1. Sign in to the Amazon Cognito console.
2. In the navigation pane, **Manage 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 a 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

Once you configure a domain for your user pool, the Amazon Cognito service automatically provisions a hosted web UI that allows you to add sign-up and sign-in pages to your app. For more information see Step 2. Add an App to Enable the Hosted Web UI (p. 21).

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 CRUD 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 Lambda authorizers, see Use API Gateway Lambda 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. 149).

> **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, **Manage 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

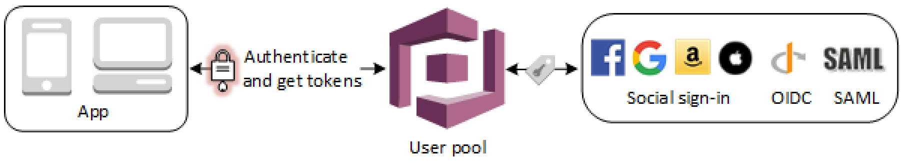
- AWS CLI: `aws cognito-idp delete-resource-server`
- AWS API: DeleteResourceServer

To update the settings for a resource server

- AWS CLI: `aws cognito-idp update-resource-server`
- AWS API: UpdateResourceServer

Adding User Pool Sign-in Through a Third Party

Your app users can sign in either directly through a user pool, or federate through a third-party identity provider (IdP). The user pool manages the overhead of handling the tokens that are returned from social sign-in through Facebook, Google, Amazon, and Apple, and from OpenID Connect (OIDC) and SAML IdPs. With the built-in hosted web UI, Amazon Cognito provides token handling and management for authenticated users from all identity providers, so your backend systems can standardize on one set of user pool tokens.



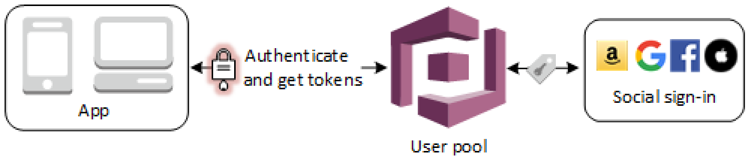
Note
Sign-in through a third party (federation) is available in Amazon Cognito user pools. This feature is independent of federation through Amazon Cognito identity pools (federated identities).

Topics

Adding Social Identity Providers to a User Pool

Your web and mobile app users can sign in through social identity providers (IdP) like Facebook, Google, Amazon, and Apple. With the built-in hosted web UI, Amazon Cognito provides token handling and management for all authenticated users, so your backend systems can standardize on one set of user pool tokens.

You can add a social identity provider in the AWS Management Console, with the AWS CLI, or using Amazon Cognito API calls.



Note
Sign-in through a third party (federation) is available in Amazon Cognito user pools. This feature is independent of federation through Amazon Cognito identity pools (federated identities).

Topics

Prerequisites

Before you begin, you need:

- A user pool with an application client and a user pool domain. For more information, see Create a user pool.
- A social identity provider.

Step 1: Register with a Social IdP

Before you create a social IdP with Amazon Cognito, you must register your application with the social IdP to receive a client ID and client secret.

To register an app with Facebook

1. Create a developer account with Facebook.
2. Sign in with your Facebook credentials.
3. From the **My Apps** menu, choose **Create New App**.
4. Give your Facebook app a name and choose **Create App ID**.
5. On the left navigation bar, choose **Settings** and then **Basic**.
6. Note the **App ID** and the **App Secret**. You will use them in the next section.
7. Choose **+ Add Platform** from the bottom of the page.
8. Choose **Website**.
9. Under **Website**, type your user pool domain with the /oauth2/idpresponse endpoint into **Site URL**.

```
https://<your_user_pool_domain>/login?
response_type=code&client_id=<your_client_id>&redirect_uri=https://www.example.com
```

10. Choose **Save changes**.
11. Type your user pool domain into **App Domains**.

```
https://<your-user-pool-domain>
```

12. Choose **Save changes**.
13. From the navigation bar choose **Products** and then **Set up** from **Facebook Login**.
14. From the navigation bar choose **Facebook Login** and then **Settings**.

    Type your redirect URL into **Valid OAuth Redirect URIs**. It will consist of your user pool domain with the /oauth2/idpresponse endpoint.

```
https://<your-user-pool-domain>/oauth2/idpresponse
```

15. Choose **Save changes**.

To register an app with Amazon

1. Create a developer account with Amazon.
2. Sign in with your Amazon credentials.
3. You need to create an Amazon security profile to receive the Amazon client ID and client secret.

   Choose **Apps and Services** from navigation bar at the top of the page and then choose **Login with Amazon**.
4. Choose **Create a Security Profile**.
5. Type in a **Security Profile Name**, a **Security Profile Description**, and a **Consent Privacy Notice URL**.
6. Choose **Save**.
7. Choose **Client ID** and **Client Secret** to show the client ID and secret. You will use them in the next section.
8. Hover over the gear and choose **Web Settings**, and then choose **Edit**.
9. Type your user pool domain into **Allowed Origins**.

```
https://<your-user-pool-domain>
```

10. Type your user pool domain with the **/oauth2/idpresponse** endpoint into **Allowed Return URLs**.

```
https://<your-user-pool-domain>/oauth2/idpresponse
```

11. Choose **Save**.

To register an app with Google

1. Create a developer account with Google.
2. Sign in with your Google credentials.
3. Choose **CONFIGURE A PROJECT**.
4. Type in a project name and choose **NEXT**.
5. Type in your product name and choose **NEXT**.
6. Choose **Web browser** from the **Where are you calling from?** drop-down list.
7. Type your user pool domain into **Authorized JavaScript origins**.

```
https://<your-user-pool-domain>
```

8. Choose **CREATE**. You will not use the **Client ID** and **Client Secret** from this step.
9. Choose **DONE**.
10. Sign in to the Google Console.
11. On the left navigation bar, choose **Credentials**.
12. Create your OAuth 2.0 credentials by choosing **OAuth client ID** from the **Create credentials** drop-down list.
13. Choose **Web application**.
14. Type your user pool domain into **Authorized JavaScript origins**.

```
https://<your-user-pool-domain>
```

15. Type your user pool domain with the **/oauth2/idpresponse** endpoint into **Authorized Redirect URIs**.

```
https://<your-user-pool-domain>/oauth2/idpresponse
```

16. Choose **Create** twice.
17. Note the **OAuth client ID** and **client secret**. You will need them for the next section.
18. Choose **OK**.

To register an app with Apple

1. Create a developer account with Apple.
2. Sign in with your Apple credentials.
3. On the left navigation bar, choose **Certificates, IDs & Profiles**.
4. On the left navigation bar, choose **Identifiers**.
5. On the **Identifiers** page, choose the **+** icon.
6. On the **Register a New Identifier** page, choose **App IDs**, and then choose **Continue**.
7. On the **Register an App ID** page, do the following:
    1. Under **Description**, type a description.
    2. Under **App ID Prefix**, type an identifier. Make a note of the value under **App ID Prefix** as you will need this value after you choose Apple as your identity provider in Step 2: Add a Social IdP to Your User Pool (p. 50).
    3. Under **Capabilities**, choose **Sign In with Apple**, and then choose **Edit**.
    4. On the **Sign in with Apple: App ID Configuration** page, select the appropriate setting for you app, and then choose **Save**.
    5. Choose **Continue**.
8. On the **Confirm your App ID** page, choose **Register**.
9. On the **Identifiers** page, hover over **App IDs** on the right side of the page, choose **Services IDs**, and then choose the **+** icon.
10. On the **Register a New Identifier** page, choose **Services IDs**, and then choose **Continue**.
11. On the **Register a Services ID** page, do the following:
    1. Under **Description**, type a description.
    2. Under **Identifier**, type an identifier. Make a note of this Services ID as you will need this value after you choose Apple as your identity provider in Step 2: Add a Social IdP to Your User Pool (p. 50).

3. Select **Sign In with Apple**, and then choose **Configure**.
4. On the **Web Authentication Configuration** page, choose a **Primary App ID**. Under **Web Domain**, type your user pool domain. Under **Return URLs**, type your user pool domain and include the /oauth2/idpresponse endpoint. For example:

```
https://<your-user-pool-domain>/oauth2/idpresponse
```

5. Choose **Add**, and then **Save**. You do not need to verify the domain.
6. Choose **Continue**, and then choose **Register**.

12. On the left navigation bar, choose **Keys**.
13. On the **Keys** page, choose the **+** icon.
14. On the **Register a New Key** page, do the following:

    1. Under **Key Name**, type a key name.
    2. Choose **Sign In with Apple**, and then choose **Configure**.
    3. On the **Configure Key** page, choose a **Primary App ID**, and then choose **Save**.
    4. Choose **Continue**, and then choose **Register**.

15. On the **Download Your Key** page, choose **Download** to download the private key, and then choose **Done**. You will need this private key and the **Key ID** value shown on this page after you choose Apple as your identity provider in Step 2: Add a Social IdP to Your User Pool (p. 50).

Step 2: Add a Social IdP to Your User Pool

In this section, you configure a social IdP in your user pool using the client ID and client secret from the previous section.

To configure a user pool social identity provider with the AWS Management Console

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. Choose **Manage your User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. On the left navigation bar, choose **Identity providers**.
5. Choose a social identity provider: **Facebook**, **Google**, **Login with Amazon**, or **Apple**.
6. For Google and Login with Amazon, type the app client ID and app client secret that you received from the social identity provider in the previous section. For Facebook, type the app client ID, app client secret that you received from the social identity provider in the previous section, and choose an API version. We recommend choosing the highest available possible version as each Facebook API version has a lifecycle and a deprecation date for example, version 2.12. You can change the API version post creation if you encounter any issues. The Facebook scopes and attributes may vary with each API version, so we recommend testing your integration." For Sign in with Apple, provide the Services ID, Team ID, Key ID, and private key that you received in the previous section.
7. Type 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 Google and Login with Amazon, they should be separated by spaces. For Sign in with Apple, select the check boxes for the scopes you want access to.

Social identity provider	Example scopes
Facebook	`public_profile, email`

Social identity provider	Example scopes
Google	`profile email openid`
Login with Amazon	`profile postal_code`
Sign in with Apple	`email name`

Your app user is asked to consent to providing these attributes to your app. For more information about their scopes, see the documentation from Google, Facebook, Login with Amazon, or Sign in with Apple.

In the case of Sign in with Apple, the following are user scenarios where scopes might not be returned:

- An end user encounters failures after leaving Apple's sign in page (can be from Internal failures within Cognito or anything written by the developer)
- The service id identifier is used across user pools and/or other authentication services
- A developer adds additional scopes after the end user has signed in before (no new information is retrieved)
- A developer deletes the user and then the user signs in again without removing the app from their Apple ID profile

8. Choose **Enable** for the social identity provider that you are configuring.
9. Choose **App client settings** from the navigation bar.
10. Select your social identity provider as one of the **Enabled Identity Providers** for your user pool app.
11. Type your callback URL into **Callback URL(s)** for your user pool app. This is the URL of the page where your user will be redirected after a successful authentication.

```
https://www.example.com
```

12. Choose **Save changes**.
13. On the **Attribute mapping** tab, add mappings for at least the required attributes, typically `email`, as follows:

    a. Select the check box to choose the Facebook, Google, or Amazon attribute name. You can also type the names of additional attributes that are not listed in the Amazon Cognito console.
    b. Choose the destination user pool attribute from the drop-down list.
    c. Choose **Save changes**.
    d. Choose **Go to summary**.

Step 3: Test Your Social IdP Configuration

You can create a login URL by using the elements from the previous two sections. Use it to test your social IdP configuration.

```
https://<your_user_pool_domain>/login?
response_type=code&client_id=<your_client_id>&redirect_uri=https://www.example.com
```

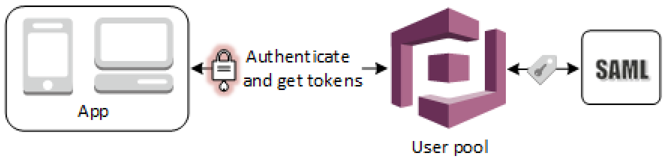
You can find your domain on the user pool **Domain name** console page. The client_id is on the **App client settings** page. Use your callback URL for the **redirect_uri** parameter. This is the URL of the page where your user will be redirected after a successful authentication.

Note
Requests that are not completed within 5 minutes will be cancelled, redirected to the login page, and then display a `Something went wrong` error message.

Adding SAML Identity Providers to a User Pool

You can enable your web and mobile app users to sign in through a SAML identity provider (IdP) such as Microsoft Active Directory Federation Services (ADFS), or Shibboleth. Choose a SAML identity provider that supports the SAML 2.0 standard.

With the built-in hosted web UI, Amazon Cognito provides token handling and management for all authenticated users, so your backend systems can standardize on one set of user pool tokens. You can create and manage a SAML IdP in the AWS Management Console, with the AWS CLI, or using Amazon Cognito API calls. To get started with the console see Adding sign-in through SAML-based identity providers to a user pool with the AWS Management Console.



Note
Sign-in through a third party (federation) is available in Amazon Cognito user pools. This feature is independent of federation through Amazon Cognito identity pools (federated identities).

You need to update your SAML identity provider and configure your user pool to support it. See the documentation for your SAML identity provider for information about how to add your user pool as a relying party or application for your SAML 2.0 identity provider.

Note
Cognito supports `relayState` values larger than 80 bytes. Whilst SAML specifications state that the `relayState` value "MUST NOT exceed 80 bytes in length", current industry practice often deviates from this behavior. As a consequence, rejecting `relayState` more than 80 bytes will break many standard SAML provider integrations.

You also need to provide an assertion consumer endpoint to your SAML identity provider. Configure this endpoint for SAML 2.0 POST binding in your SAML identity provider:

```
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.

For some SAML identity providers, you also need to provide the SP `urn` / Audience URI / SP Entity ID, in the form:

```
urn:amazon:cognito:sp:<yourUserPoolID>
```

You can find your user pool ID on the **General settings** tab in the Amazon Cognito console.

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

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. Your user pool 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.

Topics

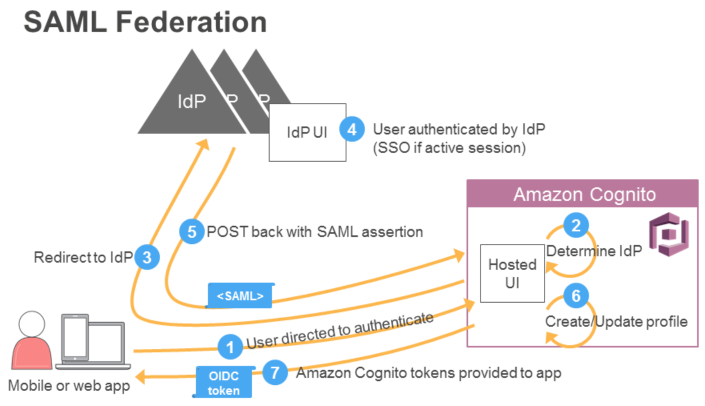
SAML User Pool IdP Authentication Flow

You can integrate SAML-based IdPs directly from your user pool.

1. The app starts the sign-up and sign-in process by directing your user to the UI hosted by AWS. A mobile app can use web view to show the pages hosted by AWS.
2. Typically your user pool determines the identity provider for your user from that user's email address.

   Alternatively, if your app gathered information before directing the user to your user pool, it can provide that information to Amazon Cognito through a query parameter.
3. Your 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 the Amazon Cognito service.
6. After verifying the SAML assertion and collecting the user attributes (claims) from the assertion, Amazon Cognito internally creates or updates the user's profile in the userpool. Amazon Cognito 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
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
7 Amazon Cognito tokens provided to app
OIDC token
Mobile or web app

Note
Requests that are not completed within 5 minutes will be cancelled, redirected to the login page, and then display a `Something went wrong` error message.

When a user authenticates, the user pool returns 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.

Choosing SAML Identity Provider Names

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 SAML 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 all 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 Configuring Identity Providers for Your User Pool (p. 181).

Creating and Managing a SAML Identity Provider for a User Pool (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. 58).

To configure a SAML 2.0 identity provider in your user pool

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. **Manage User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. On the left navigation bar, choose **Identity providers**.
5. Choose **SAML** to open the SAML dialog.
6. Under **Metadata document** upload a metadata document from your SAML IdP. You can also enter a URL that points to the metadata document. For more information, see Integrating Third-Party SAML Identity Providers with Amazon Cognito User Pools (p. 58).

   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.
7. Enter your SAML **Provider name**. For more information on SAML naming see Choosing SAML Identity Provider Names (p. 54).
8. Enter any optional SAML **Identifiers** you want to use.
9. Select **Enable IdP sign out flow** if you want your user to be logged out from the SAML IdP when logging out from Amazon Cognito.

   Enabling this flow sends a signed logout request to the SAML IdP when the LOGOUT Endpoint (p. 364) is called.

   Configure this endpoint for consuming logout responses from your IdP. This endpoint uses post binding.

   ```
   https://<yourDomainPrefix>.auth.<region>.amazoncognito.com/saml2/logout
   ```

   Note
   If this option is selected and your SAML identity provider expects a signed logout request, you will also need to configure the signing certificate provided by Amazon Cognito with your SAML IdP.
   The SAML IdP will process the signed logout request and logout your user from the Amazon Cognito session.
10. Choose **Create provider**.
11. On the **Attribute mapping** tab, add mappings for at least the required attributes, typically `email`, as follows:

    a. Type the SAML attribute name as it appears in the SAML assertion from your identity provider. If your identity provider offers sample SAML assertions, that might help you to find the name. Some identity providers use simple names, such as `email`, while others use names similar to this:

    ```
    http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress
    ```

    b. Choose the destination user pool attribute from the drop-down list.

12. Choose **Save changes**.
13. Choose **Go to summary**.

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 a user pool as a relying party

- The user pools service provider URN is: `urn:amazon:cognito:sp:<user_pool_id>`. Amazon Cognito 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`.
- Make sure 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. Use persistent SAML Name ID format.

To set up the SAML IdP to add a signing certificate

- To get the certificate containing the public key which will be used by the identity provider to verify the signed logout request, choose **Show signing certificate** under **Active SAML Providers** on the **SAML** dialog under **Identity providers** on the **Federation** console page.

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 **Identity providers** from the **Federation** console page.
4. Choose **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 for a User Pool (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** console page 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 **General 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.

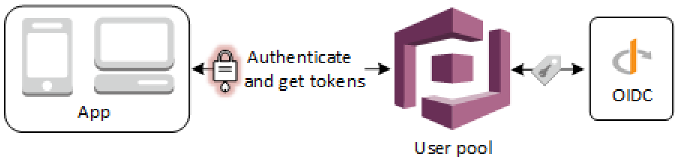
Note
Identity provider support is built in to Amazon Cognito, so you only need to go to the following provider sites to get the SAML metadata document. You may see further instructions on the provider website about integrating with AWS, but you won't need those.

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

Solution	More information
	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**, 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.

Adding OIDC Identity Providers to a User Pool

You can enable your users who already have accounts with OpenID Connect (OIDC) identity providers (IdPs) (like Salesforce or Ping Identity) to skip the sign-up step—and sign in to your application using an existing account. With the built-in hosted web UI, Amazon Cognito provides token handling and management for all authenticated users, so your backend systems can standardize on one set of user pool tokens.



Note
Sign-in through a third party (federation) is available in Amazon Cognito user pools. This feature is independent of federation through Amazon Cognito identity pools (federated identities).

You can add an OIDC IdP to your user pool in the AWS Management Console, with the AWS CLI, or by using the user pool API method CreateIdentityProvider.

Topics

Prerequisites

Before you begin, you need:

- A user pool with an application client and a user pool domain. For more information, see Create a user pool.
- An OIDC IdP.

Step 1: Register with an OIDC IdP

Before you create an OIDC IdP with Amazon Cognito, you must register your application with the OIDC IdP to receive a client ID and client secret.

To register with an OIDC IdP

1. Create a developer account with the OIDC IdP.

   Links to OIDC IdPs

OIDC IdP	How to Install	OIDC Discovery URL
Salesforce	Install a Salesforce identity provider	https://login.salesforce.com
Ping Identity	Install a Ping Identity identity provider	https://*Your Ping domain address*:9031/idp/userinfo.openid For example: `https://pf.company.com:9031/idp/userinfo.openid`
Okta	Install an Okta identity provider	https://*Your Okta subdomain*.oktapreview.com or `https://`*Your Okta subdomain*`.okta.com`
Microsoft Azure Active Directory (Azure AD)	Install a Microsoft Azure AD identity provider	https://login.windows.net/common
Google	Install a Google identity provider	https://accounts.google.com **Note** Amazon Cognito offers Google as an integrated social sign-in IdP. We recommend that you use the integrated IdP. See Adding Social Identity Providers to a User Pool (p. 46).

2. Register your user pool domain URL with the /oauth2/idpresponse endpoint with your OIDC IdP. This ensures that the OIDC IdP later accepts it from Amazon Cognito when it authenticates users.

```
https://<your-user-pool-domain>/oauth2/idpresponse
```

3. Register your callback URL with your Cognito User Pool. This is the URL of the page where your user will be redirected after a successful authentication.

```
https://www.example.com
```

4. Select your scopes. The scope **openid** is required. The **email** scope is needed to grant access to the **email** and **email_verified** claims.
5. The OIDC IdP provides you with a client ID and a client secret. You'll use them when you set up an OIDC IdP in your user pool.

Example: Use Salesforce as an OIDC IdP with your user pool

You use an OIDC identity provider when you want to establish trust between an OIDC-compatible IdP such as Salesforce and your user pool.

1. Create an account on the Salesforce Developers website.
2. Sign in through your developer account that you set up in the previous step.
3. Look at the top of your Salesforce page.
   - If you're using Lightning Experience, choose the Setup gear icon, then choose **Setup Home**.
   - If you're using Salesforce Classic and you see **Setup** in the user interface header, choose it.
   - If you're using Salesforce Classic and you don't see **Setup** in the header, choose your name from the top navigation bar, and choose **Setup** from the drop-down list.
4. On the left navigation bar, choose **Company Settings**.
5. On the navigation bar, choose **Domain**, type a domain, and choose **Create**.
6. On the left navigation bar, go to **Platform Tools** and choose **Apps**.
7. Choose **App Manager**.
8. a. Choose **new connected app**.

   b. Fill out the required fields.

      Under **Start URL**, type your user pool domain URL with the /oauth2/idpresponse endpoint.

      ```
      https://<your-user-pool-domain>/oauth2/idpresponse
      ```

   c. Enable **OAuth settings** and, type your callback URL into **Callback URL**. This is the URL of the page where your user will be redirected after a successful sign-in.

      ```
      https://www.example.com
      ```
9. Select your scopes. The scope **openid** is required. The **email** scope is needed to grant access to the **email** and **email_verified** claims.

   Scopes are separated by spaces.
10. Choose **Create**.

    In Salesforce, the client ID is called a **Consumer Key**, and the client secret is a **Consumer Secret**. Note your client ID and client secret. You will use them in the next section.

Step 2: Add an OIDC IdP to Your User Pool

In this section, you configure your user pool to process OIDC-based authentication requests from an OIDC IdP.

To add an OIDC IdP (Amazon Cognito console)

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. **Manage User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. On the left navigation bar, choose **Identity providers**.
5. Choose **OpenId Connect**.
6. Type a unique name into **Provider name**.
7. Type the OIDC IdP's client ID from the previous section into **Client ID**.
8. Type the client secret from the previous section into **Client secret**.
9. In the drop-down list, choose the HTTP method (either GET or POST) that's used to fetch the details of the user from the **userinfo** endpoint into **Attributes request method**.
10. Type the names of the scopes that you want to authorize. Scopes define which user attributes (such as `name` and `email`) that you want to access with your application. Scopes are separated by spaces, according to the OAuth 2.0 specification.

    Your app user is asked to consent to providing these attributes to your application.
11. Type the URL of your IdP and choose **Run discovery**.

    Note
    The URL should start with `https://`, and shouldn't end with a slash `/`. Only port numbers 443 and 80 can be used with this URL. For example, Salesforce uses this URL:
    `https://login.salesforce.com`

    - If **Run discovery** isn't successful, then you need to provide the **Authorization endpoint**, **Token endpoint**, **Userinfo endpoint**, and **Jwks uri** (the location of the JSON Web Key).

    Note
    If provider uses discovery for federated login, the discovery document must use HTTPS for the following values: authorization_endpoint, token_endpoint, userinfo_endpoint, and jwks_uri. Otherwise the login will fail.
12. Choose **Create provider**.
13. On the left navigation bar, choose **App client settings**.
14. Select the OIDC provider that you set up in the previous step as one of the **Enabled Identity Providers**.
15. Type a callback URL for the Amazon Cognito authorization server to call after users are authenticated. This is the URL of the page where your user will be redirected after a successful authentication.

    ```
    https://www.example.com
    ```
16. Under **Allowed OAuth Flows**, enable both the **Authorization code grant** and the **Implicit code grant**.

    Unless you specifically want to exclude one, select the check boxes for all of the **Allowed OAuth scopes**.
17. Choose **Save changes**.
18. On the **Attribute mapping** tab on the left navigation bar, add mappings of OIDC claims to user pool attributes.

    a. As a default, the OIDC claim **sub** is mapped to the user pool attribute **Username**. You can map other OIDC claims to user pool attributes. Type in the OIDC claim, and choose the corresponding user pool attribute from the drop-down list. For example, the claim **email** is often mapped to the user pool attribute **Email**.

b. In the drop-down list, choose the destination user pool attribute.

c. Choose **Save changes**.

d. Choose **Go to summary**.

To add an OIDC IdP (AWS CLI)

- See the parameter descriptions for the CreateIdentityProvider API method.

```
aws cognito-idp create-identity-provider
--user-pool-id string
--provider-name string
--provider-type OIDC
--provider-details map


--attribute-mapping string
--idp-identifiers (list)
--cli-input-json string
--generate-cli-skeleton string
```

Use this map of provider details:

```
{
  "client_id": "string",
  "client_secret": "string",
  "authorize_scopes": "string",
  "attributes_request_method": "string",
  "oidc_issuer": "string",

  "authorize_url": "string",
  "token_url": "string",
  "attributes_url": "string",
  "jwks_uri": "string"
}
```

Step 3: Test Your OIDC IdP Configuration

You can create the authorization URL by using the elements from the previous two sections, and using them to test your OIDC IdP configuration.

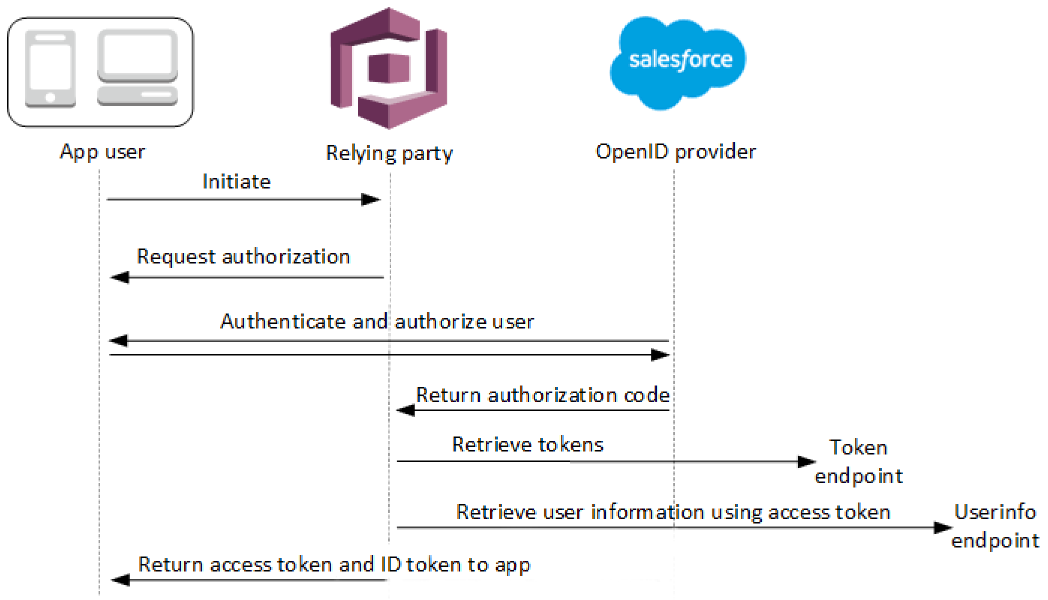
```
https://<your_user_pool_domain>/oauth2/authorize?
response_type=code&client_id=<your_client_id>&redirect_uri=https://www.example.com
```

You can find your domain on the user pool **Domain name** console page. The client_id is on the **General settings** page. Use your callback URL for the **redirect_uri** parameter. This is the URL of the page where your user will be redirected after a successful authentication.

OIDC User Pool IdP Authentication Flow

When your user signs in to your application using an OIDC IdP, this is the authentication flow.

1. Your user lands on the Amazon Cognito built-in sign-in page, and is offered the option to sign in through an OIDC IdP such as Salesforce.
2. Your user is redirected to the OIDC IdP's `authorization` endpoint.
3. After your user is authenticated, the OIDC IdP redirects to Amazon Cognito with an authorization code.
4. Amazon Cognito exchanges the authorization code with the OIDC IdP for an access token.
5. Amazon Cognito creates or updates the user account in your user pool.
6. Amazon Cognito issues your application bearer tokens, which might include identity, access, and refresh tokens.


App user
Relying party
salesforce
OpenID provider
Initiate
Request authorization
Authenticate and authorize user
Return authorization code
Retrieve tokens
Token endpoint
Retrieve user information using access token
Userinfo endpoint
Return access token and ID token to app

Note
Requests that are not completed within 5 minutes will be cancelled, redirected to the login page, and then display a `Something went wrong` error message.

OIDC is an identity layer on top of OAuth 2.0, which specifies JSON-formatted (JWT) identity tokens that are issued by IdPs to OIDC client apps (relying parties). See the documentation for your OIDC IdP for information about to add Amazon Cognito as an OIDC relying party.

When a user authenticates, the user pool returns 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.

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.

Things to know about mappings

Before using mappings, review the following important details:

- A mapping must be present for each user pool attribute that's required when a user signs in to your application. For example, if your user pool requires an email attribute for sign-in, map this attribute to its equivalent from the identity provider.
- By default, mapped email addresses are unverified. You can't verify a mapped email address using a one-time code. Instead, map an attribute from your identity provider to get the verification status. For example, Google and most OIDC providers include the `email_verified` attribute.
- For each mapped user pool attribute, the maximum value length (2048 characters) must be large enough for the value that Amazon Cognito obtains from the identity provider. Otherwise, Amazon Cognito throws an error when users sign in to your application. If you map a custom attribute to an identity provider token, set the length to 2048 characters.
- The `username` user pools attribute must be mapped only to specific attributes for the following identity providers:

Identity Provider	Attribute to Map to `username`
Facebook	`id`
Google	`sub`
Login with Amazon	`user_id`
OpenID Connect (OIDC) providers	`sub`
Sign in with Apple	`sub`

- Amazon Cognito must be able to update your mapped user pool attributes when users sign in to your application. When a user signs in through an identity provider, Amazon Cognito updates the mapped attributes with the latest information from the identity provider. Amazon Cognito updates each mapped attribute, even if its current value already matches the latest information. If Amazon Cognito can't update the attribute, it throws an error. To ensure that Amazon Cognito can update the attributes, check the following requirements:
  - The mapped user pool attributes must be *mutable*. Mutable attributes can be updated by app clients that have write permissions for those attributes. You can set user pool attributes as mutable when you define them in the Amazon Cognito console. Or, if you create your user pool by using the CreateUserPool API operation, you can set the `Mutable` parameter for each of these attributes to `true`.
  - In the app client settings for your application, the mapped attributes must be *writable*. You can set which attributes are writable in the **App clients** page in the Amazon Cognito console. Or, if you create the app client by using the `CreateUserPoolClient` API operation, you can add these attributes to the `WriteAttributes` array.
- In the case of identity provider attributes containing multiple values, Amazon Cognito will URL form encode the values containing non-alphanumeric characters (excluding the '`.`', '`-`', '`*`', and '`_`' characters). You should decode the values prior to use in your application.

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
Amazon Cognito will map incoming claims to user pool attributes only if the claims exist in the incoming token. If a previously mapped claim no longer exists in the incoming token, it won't be deleted or changed. If your application requires mapping of deleted claims, you can use the Pre-Authentication lambda trigger to delete the custom attribute during authentication and allow these attributes to re-populate from the incoming token.

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**, **Amazon** or **Apple** tab.
5. For each attribute you need to map, perform the following steps:

    a. Select the **Capture** check box.
    b. For **User pool attribute**, in the drop-down list, choose the user pool attribute you want to map to the social identity provider attribute.
    c. If you need more attributes, choose **Add Facebook attribute** (or **Add Google attribute** or **Add Amazon attribute** or **Add Apple attribute**) and perform the following steps:

        i. In the **Facebook attribute** (or **Google attribute** or **Amazon attribute** or **Apple attribute**) field, enter the name of the attribute to be mapped.
        ii. In the **User pool attribute** field, choose the user pool attribute to map the social identity provider attribute to from the drop-down list.
    d. 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:

    a. Choose **Add SAML attribute**.
    b. In the **SAML attribute** field, enter the name of the SAML attribute to be mapped.
    c. 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

Customizing User Pool Workflows with Lambda Triggers

You can create an AWS Lambda function and then trigger that function during user pool operations such as user sign-up, confirmation, and sign-in (authentication) with a Lambda trigger. You can add authentication challenges, migrate users, and customize verification messages.

The following table summarizes some of the customizations that can be made:

User Pool Flow	Operation	Description
Custom Authentication Flow	**Define Auth Challenge**	Determines the next challenge in a custom auth flow

User Pool Flow	Operation	Description
	Create Auth Challenge	Creates a challenge in a custom auth flow
	Verify Auth Challenge Response	Determines if a response is correct in a custom auth flow
Authentication Events	**the section called "Pre Authentication Lambda Trigger" (p. 81)**	Custom validation to accept or deny the sign-in request
	the section called "Post Authentication Lambda Trigger" (p. 84)	Event logging for custom analytics
	the section called "Pre Token Generation Lambda Trigger" (p. 96)	Augment or suppress token claims
Sign-Up	**the section called "Pre Sign-up Lambda Trigger" (p. 72)**	Custom validation to accept or deny the sign-up request
	the section called "Post Confirmation Lambda Trigger" (p. 78)	Custom welcome messages or event logging for custom analytics
	the section called "Migrate User Lambda Trigger" (p. 100)	Migrate a user from an existing user directory to user pools
Messages	**the section called "Custom Message Lambda Trigger" (p. 103)**	Advanced customization and localization of messages
Token Creation	**the section called "Pre Token Generation Lambda Trigger" (p. 96)**	Add or remove attributes in Id tokens
Email and SMS third-party providers	**the section called "Custom Sender Lambda Triggers" (p. 108)**	Use a third-party provider to send SMS and email messages

Topics

- Migrate User Lambda Trigger (p. 100)
- Custom Message Lambda Trigger (p. 103)
- Custom sender Lambda triggers (p. 108)

Important Considerations

The following information is important to consider before you start working with Lambda functions:

- Except for Custom Sender Lambda triggers, Amazon Cognito invokes Lambda functions synchronously. When called, your Lambda function must respond within 5 seconds. If it does not, Amazon Cognito retries the call. After 3 unsuccessful attempts, the function times out. This 5-second timeout value cannot be changed. For more information see the Lambda programming model.
- If you delete an AWS Lambda trigger, you must update the corresponding trigger in the user pool. For example, if you delete the post authentication trigger, you must set the **Post authentication** trigger in the corresponding user pool to **none**.
- Except for Custom Sender Lambda triggers, errors thrown by Lambda triggers will be visible directly to your end users if they are using Amazon Cognito Hosted UI as query parameters in the Callback URL. As a recommended best practice, end user facing errors should be thrown from the Lambda triggers and any sensitive or debugging information should be logged in the Lambda trigger itself.

Adding a User Pool Lambda Trigger

To add a user pool Lambda trigger with the console

1. Create a Lambda function using the Lambda console. For more information on Lambda functions, see the AWS Lambda Developer Guide.
2. Navigate to the Amazon Cognito console, choose **Manage User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. In your user pool, choose the **Triggers** tab from the navigation bar.
5. Choose a Lambda trigger such as **Pre sign-up** or **Pre authentication** and choose your Lambda function from the **Lambda function** drop-down list.
6. Choose **Save changes**.
7. You can log your Lambda function using CloudWatch in the Lambda console. For more information see Accessing CloudWatch Logs for Lambda.

User Pool Lambda Trigger Event

Amazon Cognito passes event information to your Lambda function which returns the same event object back to Amazon Cognito with any changes in the response. This event shows the Lambda trigger common parameters:

JSON

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

User Pool Lambda Trigger Common Parameters

version

The version number of your Lambda function.

triggerSource

The name of the event that triggered the Lambda function. For a description of each triggerSource see User Pool Lambda Trigger Sources (p. 70).

region

The AWS Region, as an `AWSRegion` instance.

userPoolId

The user pool ID for the user pool.

userName

The username of the current user.

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 in the response depend on the triggering event.

User Pool Lambda Trigger Sources

This section describes each Amazon Cognito Lambda triggerSource parameter, and its triggering event.

Sign-up, confirmation, and sign-in (authentication) triggers

Trigger	triggerSource value	Triggering event
Pre sign-up	`PreSignUp_SignUp`	Pre sign-up.
Pre sign-up	`PreSignUp_AdminCreateUser`	Pre sign-up when an admin creates a new user.
Pre sign-up	`PreSignUp_ExternalProvider`	Pre sign-up for external identity providers.
Pre sign-up	`PreSignUp_ExternalProvider`	Pre sign-up for external identity providers.
Post confirmation	`PostConfirmation_ConfirmSignUp`	Post sign-up confirmation.
Post confirmation	`PostConfirmation_ConfirmForgotPassword`	Post Forgot Password confirmation.
Pre authentication	`PreAuthentication_Authentication`	Pre authentication.
Post authentication	`PostAuthentication_Authentication`	Post authentication.

Custom authentication challenge triggers

Trigger	triggerSource value	Triggering event
Define auth challenge	`DefineAuthChallenge_Authentication`	Define Auth Challenge.
Create auth challenge	`CreateAuthChallenge_Authentication`	Create Auth Challenge.
Verify auth challenge	`VerifyAuthChallengeResponse_Authentication`	Verify Auth Challenge Response.

Pre token generation triggers

Trigger	triggerSource value	Triggering event
Pre token generation	`TokenGeneration_HostedAuth`	Called during authentication from the Amazon Cognito hosted UI sign-in page.
Pre token generation	`TokenGeneration_Authentication`	Called after user authentication flows have completed.
Pre token generation	`TokenGeneration_NewPasswordChallenge`	Called after the user is created by an admin. This flow is invoked when the user has to change a temporary password.
Pre token generation	`TokenGeneration_AuthenticateDevice`	Called at the end of the authentication of a user device.
Pre token generation	`TokenGeneration_RefreshTokens`	Called when a user tries to refresh the identity and access tokens.

Migrate user triggers

Trigger	triggerSource value	Triggering event
User migration	`UserMigration_Authentication`	User migration at the time of sign in.
User migration	`UserMigration_ForgotPassword`	User migration during the forgot-password flow.

Custom message triggers

Trigger	triggerSource value	Triggering event
Custom message	`CustomMessage_SignUp`	Custom message – To send the confirmation code post sign-up.
Custom message	`CustomMessage_AdminCreateUser`	Custom message – To send the temporary password to a new user.
Custom message	`CustomMessage_ResendCode`	Custom message – To resend the confirmation code to an existing user.
Custom message	`CustomMessage_ForgotPassword`	Custom message – To send the confirmation code for Forgot Password request.
Custom message	`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.
Custom message	`CustomMessage_VerifyUserAttribute`	Custom message – This trigger sends a verification code to the user when they manually request it for a new email or phone number.
Custom message	`CustomMessage_Authentication`	Custom message – To send MFA code during authentication.

Pre Sign-up Lambda Trigger

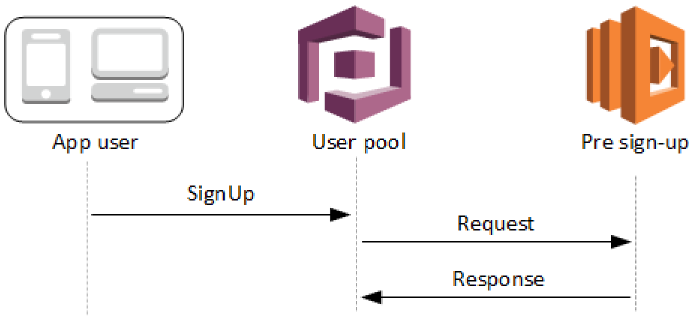
The pre sign-up Lambda function is triggered just before Amazon Cognito signs up a new user. It allows you to perform custom validation to accept or deny the registration request as part of the sign-up process.

Topics

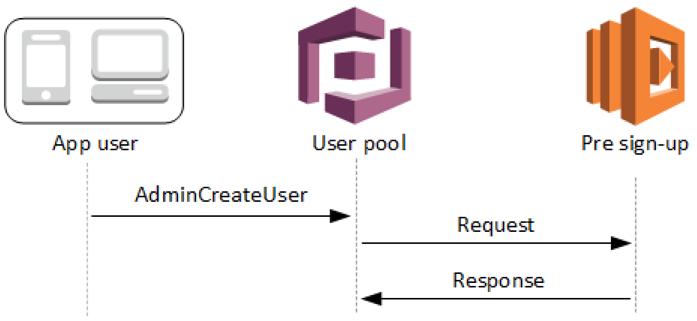
- Pre Sign-up Example: Auto-Confirm Users from a Registered Domain (p. 75)
- Pre Sign-up Example: Auto-Confirm and Auto-Verify all Users (p. 77)

Pre Sign-up Lambda Flows

Client Sign-up Flow



Server Sign-up Flow



The request includes validation data from the client which comes from the `ValidationData` values passed to the user pool SignUp and AdminCreateUser API methods.

Pre Sign-up Lambda Trigger Parameters

These are the parameters required by this Lambda function in addition to the common parameters.

JSON

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

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

Pre Sign-up Request Parameters

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. You can pass this data to your Lambda function by using the ClientMetadata parameter in the InitiateAuth and AdminInitiateAuth API actions.

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function that you specify for the pre sign-up trigger. You can pass this data to your Lambda function by using the ClientMetadata parameter in the following API actions: AdminCreateUser, AdminRespondToAuthChallenge, ForgotPassword, and SignUp.

Pre Sign-up Response Parameters

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.

Note
Response parameters `autoVerifyPhone`, `autoVerifyEmail` and `autoConfirmUser` are ignored by Amazon Cognito when the Pre Sign-up lambda is triggered by the `AdminCreateUser` API.

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. 166).

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 exists, 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. 166).

Sign-up Tutorials

The pre sign-up Lambda function is triggered before user sign-up. See these Amazon Cognito sign-up tutorials for JavaScript, Android, and iOS.

Platform	Tutorial
JavaScript Identity SDK	Sign up users with JavaScript
Android Identity SDK	Sign up users with Android
iOS Identity SDK	Sign up users with iOS

Pre Sign-up Example: Auto-Confirm Users from a Registered Domain

With the pre sign-up Lambda trigger, You can add custom logic to validate new users signing up for your user pool. This is a sample JavaScript program that demonstrates how to sign up a new user. It will invoke a pre sign-up Lambda trigger as part of the authentication.

JavaScript

```
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 dataEmailDomain = {
    Name: "custom:domain",
    Value: "example.com"
}
var attributeEmail =
new AmazonCognitoIdentity.CognitoUserAttribute(dataEmail);
var attributePhoneNumber =
new AmazonCognitoIdentity.CognitoUserAttribute(dataPhoneNumber);
var attributeEmailDomain =
new AmazonCognitoIdentity.CognitoUserAttribute(dataEmailDomain);

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

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());
});
```

This is a sample Lambda trigger called just before sign-up with the user pool pre sign-up Lambda trigger. It uses a custom attribute **custom:domain** to automatically confirm new users from a particular email domain. Any new users not in the custom domain will be added to the user pool, but not automatically confirmed.

Node.js

```
exports.handler = (event, context, callback) => {
    // Set the user pool autoConfirmUser flag after validating the email domain
    event.response.autoConfirmUser = false;

    // Split the email address so we can compare domains
    var address = event.request.userAttributes.email.split("@")

    // This example uses a custom attribute "custom:domain"
    if (event.request.userAttributes.hasOwnProperty("custom:domain")) {
        if ( event.request.userAttributes['custom:domain'] === address[1]) {
            event.response.autoConfirmUser = true;
        }
    }

    // Return to Amazon Cognito
    callback(null, event);
};
```

Python

```
def lambda_handler(event, context):
    # It sets the user pool autoConfirmUser flag after validating the email domain
    event['response']['autoConfirmUser'] = False

    # Split the email address so we can compare domains
    address = event['request']['userAttributes']['email'].split('@')

    # This example uses a custom attribute 'custom:domain'
    if 'custom:domain' in event['request']['userAttributes']:
        if event['request']['userAttributes']['custom:domain'] == address[1]:
            event['response']['autoConfirmUser'] = True

    # Return to Amazon Cognito
    return event
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object back to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that's relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
    "request": {
        "userAttributes": {
            "email": "testuser@example.com",
            "custom:domain": "example.com"
        }
    },
    "response": {}
}
```

Pre Sign-up Example: Auto-Confirm and Auto-Verify all Users

This example confirms all users and sets the user's `email` and `phone_number` attributes to verified if the attribute is present. Also, if aliasing is enabled, aliases will be created for `phone_number` and `email` when auto-verify is set.

Note
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. To prevent this from happening, you can use the user pools 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.

Node.js

```
exports.handler = (event, context, callback) => {

    // Confirm the user
        event.response.autoConfirmUser = true;

    // Set the email as verified if it is in the request
    if (event.request.userAttributes.hasOwnProperty("email")) {
        event.response.autoVerifyEmail = true;
    }

    // Set the phone number as verified if it is in the request
    if (event.request.userAttributes.hasOwnProperty("phone_number")) {
        event.response.autoVerifyPhone = true;
    }

    // Return to Amazon Cognito
    callback(null, event);
};
```

Python

```
def lambda_handler(event, context):
    # Confirm the user
    event['response']['autoConfirmUser'] = True

    # Set the email as verified if it is in the request
    if 'email' in event['request']['userAttributes']:
        event['response']['autoVerifyEmail'] = True

    # Set the phone number as verified if it is in the request
    if 'phone_number' in event['request']['userAttributes']:
        event['response']['autoVerifyPhone'] = True

    # Return to Amazon Cognito
    return event
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object back to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that’s relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
  "request": {
    "userAttributes": {
      "email": "user@example.com",
      "phone_number": "+12065550100"
    }
  },
  "response": {}
}
```

Post Confirmation Lambda Trigger

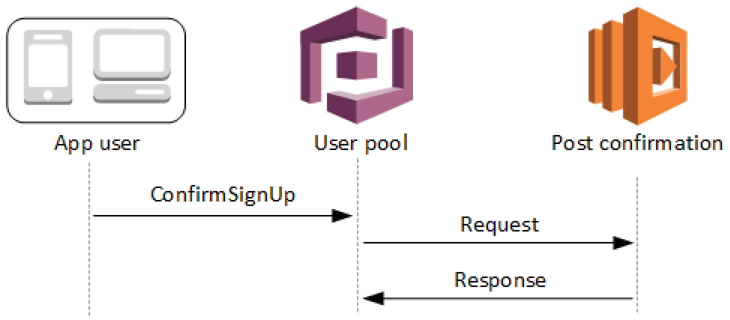
Amazon Cognito invokes this trigger after a new user is confirmed, allowing you to send custom messages or to add custom logic. For example, you could use this trigger to gather new user data.

The request contains the current attributes for the confirmed user.

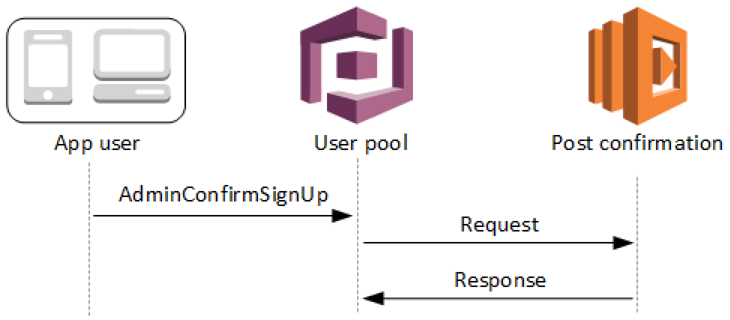
Topics

Post Confirmation Lambda Flows

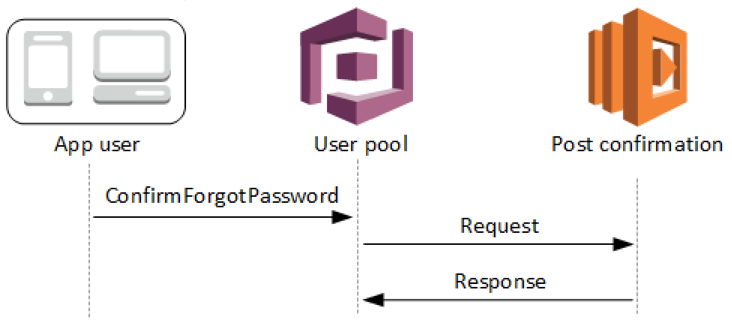
Client Confirm Sign-up Flow



Server Confirm Sign-up Flow



Confirm Forgot Password Flow



Post Confirmation Lambda Trigger Parameters

These are the parameters required by this Lambda function in addition to the common parameters.

JSON

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

Post Confirmation Request Parameters

userAttributes

One or more key-value pairs representing user attributes.

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function that you specify for the post confirmation trigger. You can pass this data to your Lambda function by using the ClientMetadata parameter in the following API actions: AdminConfirmSignUp, ConfirmForgotPassword, ConfirmSignUp, and SignUp.

Post Confirmation Response Parameters

No additional return information is expected in the response.

User Confirmation Tutorials

The post confirmation Lambda function is triggered just after Amazon Cognito confirms a new user. See these user confirmation tutorials for JavaScript, Android, and iOS.

Platform	Tutorial
JavaScript Identity SDK	Confirm users with JavaScript
Android Identity SDK	Confirm users with Android
iOS Identity SDK	Confirm users with iOS

Post Confirmation Example

This example Lambda function sends a confirmation email message to your user using Amazon SES. For more information see Amazon Simple Email Service Developer Guide.

Node.js

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

var ses = new aws.SES();

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

    if (event.request.userAttributes.email) {
            sendEmail(event.request.userAttributes.email, "Congratulations " +
 event.userName + ", you have been confirmed: ", function(status) {

            // Return to Amazon Cognito
            callback(null, event);
        });
    } else {
        // Nothing to do, the user's email ID is unknown
        callback(null, event);
    }
};

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

        // Replace source_email with your SES validated email address
        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");
};
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object back to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that's relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
    "request": {
        "userAttributes": {
            "email": "user@example.com",
            "email_verified": true
        }
    },
    "response": {}
}
```

Pre Authentication Lambda Trigger

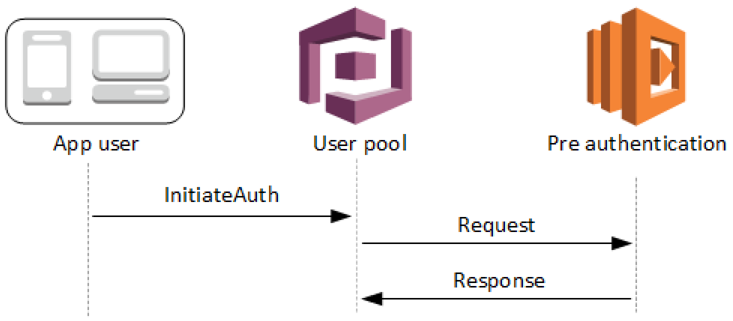
Amazon Cognito invokes this trigger when a user attempts to sign in, allowing custom validation to accept or deny the authentication request.

Note
Triggers are dependant on the user existing in the user pool before trigger activation.

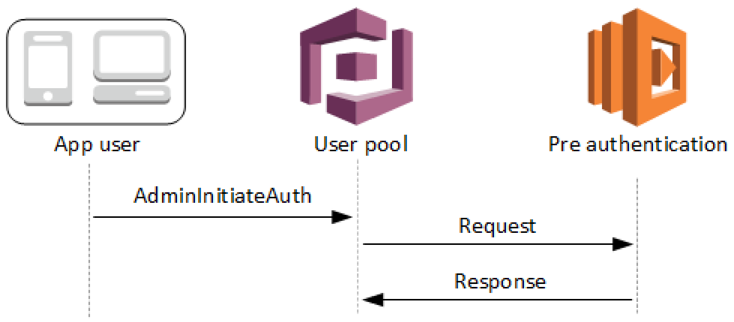
Topics

Pre Authentication Lambda Flows

Client Authentication Flow



Server Authentication Flow



The request includes validation data from the client which comes from the `ClientMetadata` values passed to the user pool InitiateAuth and AdminInitiateAuth API methods.

For more information, see User Pool Authentication Flow (p. 306).

Pre Authentication Lambda Trigger Parameters

These are the parameters required by this Lambda function in addition to the common parameters.

JSON

```
{
    "request": {
        "userAttributes": {
            "string": "string",
            . . .
        },
        "validationData": {
            "string": "string",
            . . .
        },
        "userNotFound": boolean
    },
    "response": {}
}
```

Pre Authentication Request Parameters

userAttributes

One or more name-value pairs representing user attributes.

userNotFound

This boolean is populated when `PreventUserExistenceErrors` is set to `ENABLED` for your User Pool client.

validationData

One or more key-value pairs containing the validation data in the user's sign-in request. You can pass this data to your Lambda function by using the ClientMetadata parameter in the InitiateAuth and AdminInitiateAuth API actions.

Pre Authentication Response Parameters

No additional return information is expected in the response.

Authentication Tutorials

The pre authentication Lambda function is triggered just before Amazon Cognito signs in a new user. See these sign-in tutorials for JavaScript, Android, and iOS.

Platform	Tutorial
JavaScript Identity SDK	Sign in users with JavaScript
Android Identity SDK	Sign in users with Android
iOS Identity SDK	Sign in users with iOS

Pre Authentication Example

This sample function prevents users from a specific user pool app client to sign-in to the user pool.

Node.js

```
exports.handler = (event, context, callback) => {
    if (event.callerContext.clientId === "user-pool-app-client-id-to-be-blocked") {
        var error = new Error("Cannot authenticate users from this user pool app
 client");

        // Return error to Amazon Cognito
        callback(error, event);
    }

    // Return to Amazon Cognito
    callback(null, event);
};
```

Python

```
def lambda_handler(event, context):
    if event['callerContext']['clientId'] == "<user pool app client id to be blocked>":
        raise Exception("Cannot authenticate users from this user pool app client")

    # Return to Amazon Cognito
    return event
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object back to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that's relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
    "callerContext": {
        "clientId": "<user pool app client id to be blocked>"
    },
    "response": {}
}
```

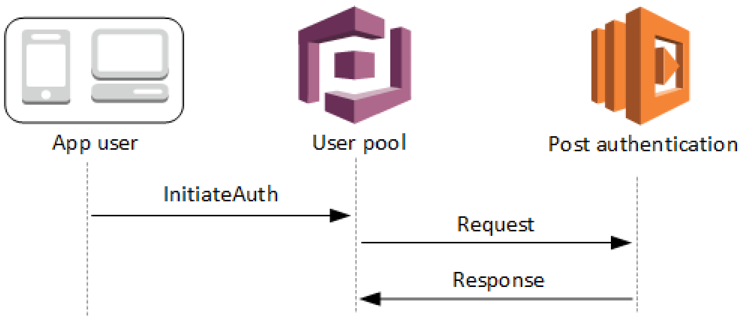
Post Authentication Lambda Trigger

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

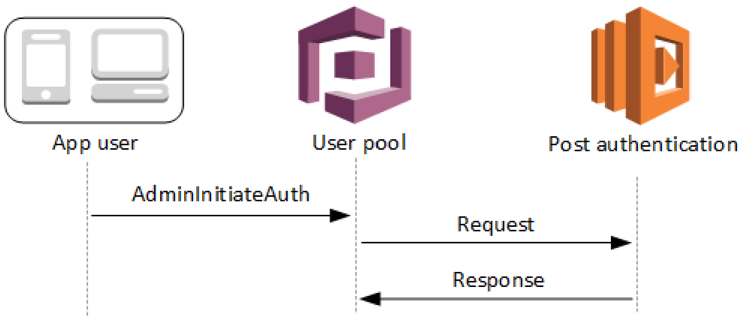
Topics

Post Authentication Lambda Flows

Client Authentication Flow



Server Authentication Flow



For more information, see User Pool Authentication Flow (p. 306).

Post Authentication Lambda Trigger Parameters

These are the parameters required by this Lambda function in addition to the common parameters.

JSON

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

Post Authentication Request Parameters

newDeviceUsed

This flag indicates if the user has signed in on a new device. It is set only if the remembered devices value of the user pool is set to `Always` or `User Opt-In`.

userAttributes

One or more name-value pairs representing user attributes.

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function that you specify for the post authentication trigger. You can pass this data to your Lambda function by using the ClientMetadata parameter in the AdminRespondToAuthChallenge and RespondToAuthChallenge API actions.

Post Authentication Response Parameters

No additional return information is expected in the response.

Authentication Tutorials

The post authentication Lambda function is triggered just after Amazon Cognito signs in a new user. See these sign-in tutorials for JavaScript, Android, and iOS.

Platform	Tutorial
JavaScript Identity SDK	Sign in users with JavaScript
Android Identity SDK	Sign in users with Android
iOS Identity SDK	Sign in users with iOS

Post Authentication Example

This post authentication sample Lambda function sends data from a successful sign-in to CloudWatch Logs.

Node.js

```
exports.handler = (event, context, callback) => {

    // Send post authentication data to Cloudwatch logs
    console.log ("Authentication successful");
    console.log ("Trigger function =", event.triggerSource);
    console.log ("User pool = ", event.userPoolId);
    console.log ("App client ID = ", event.callerContext.clientId);
    console.log ("User ID = ", event.userName);

    // Return to Amazon Cognito
    callback(null, event);
};
```

Python

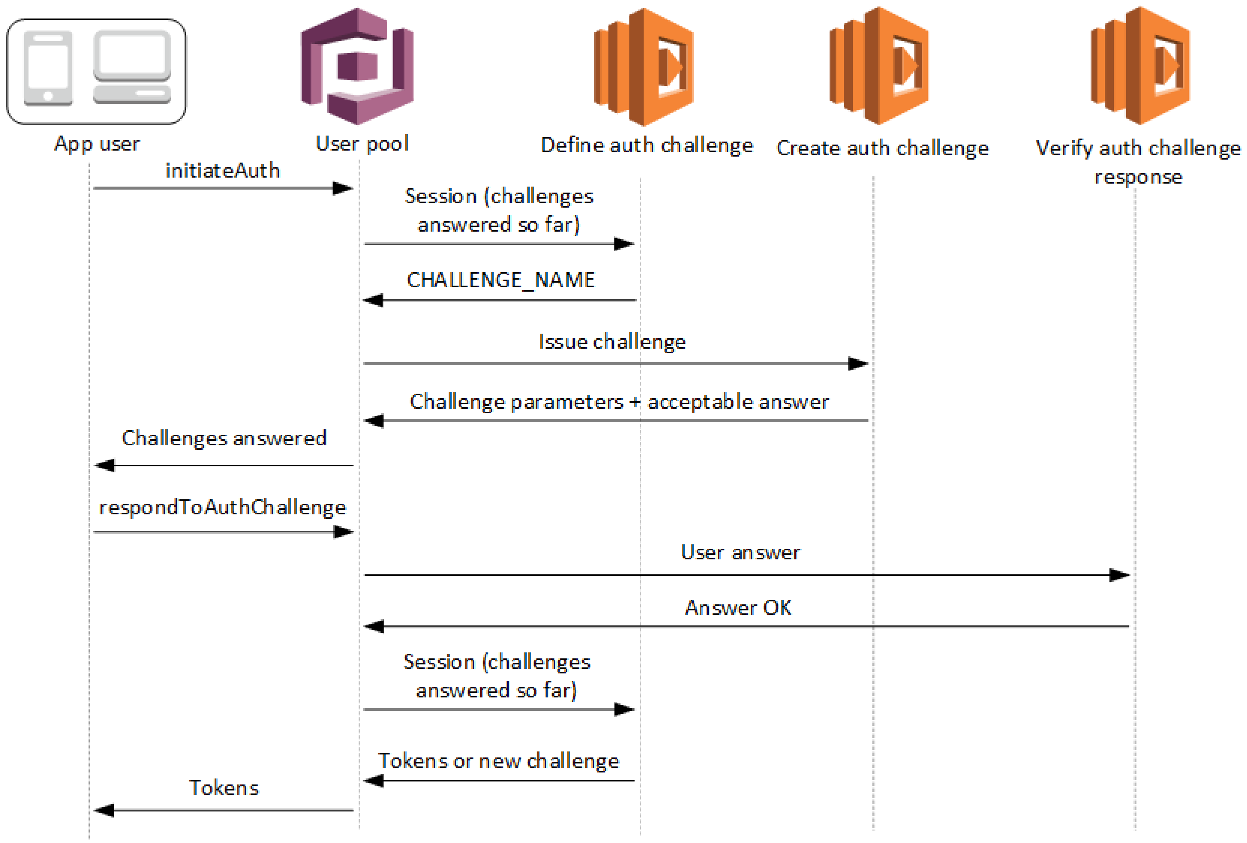
```
from __future__ import print_function
def lambda_handler(event, context):

    # Send post authentication data to Cloudwatch logs
    print ("Authentication successful")
    print ("Trigger function =", event['triggerSource'])
    print ("User pool = ", event['userPoolId'])
    print ("App client ID = ", event['callerContext']['clientId'])
    print ("User ID = ", event['userName'])

    # Return to Amazon Cognito
    return event
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object back to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that's relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
  "triggerSource": "testTrigger",
  "userPoolId": "testPool",
  "userName": "testName",
  "callerContext": {
      "clientId": "12345"
  },
  "response": {}
}
```

Custom Authentication Challenge Lambda Triggers



These Lambda triggers issue and verify their own challenges as part of a user pool custom authentication flow.

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** to create a custom challenge.

Verify auth challenge response

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

You can incorporate new challenge types with these challenge Lambda triggers. For example, these challenge types might include CAPTCHAs or dynamic challenge questions.

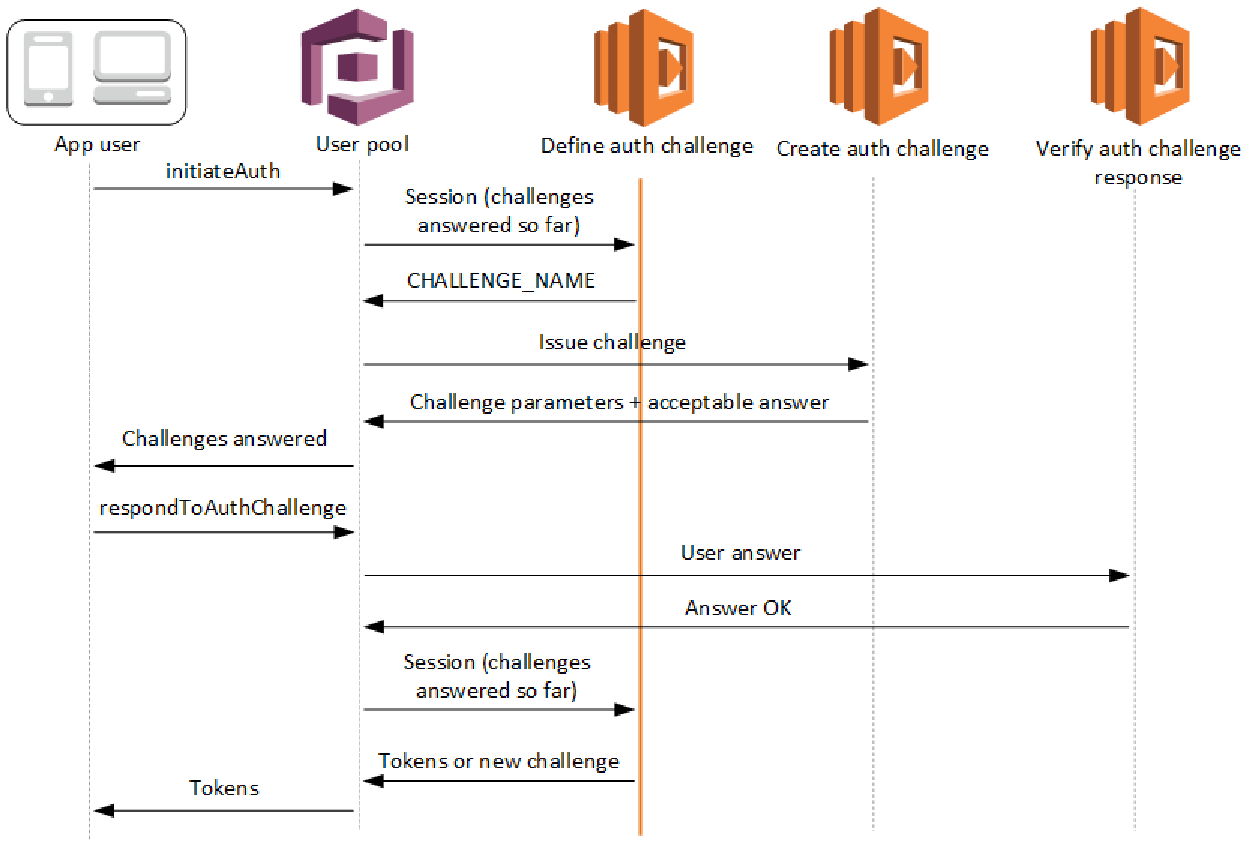
You can generalize authentication into two common steps with the user pool InitiateAuth and RespondToAuthChallenge API methods.

In this flow, a user authenticates by answering successive challenges until authentication either fails or the user is issued tokens. These two API calls can be repeated to include different challenges.

Note
The Amazon Cognito hosted sign-in web page does not support the custom authentication flow.

Topics

Define Auth Challenge Lambda Trigger



Define auth challenge

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

The request for this Lambda trigger contains `session`, which is an array that contains all of the challenges that are presented to the user in the current authentication process. The request also includes 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.

You can have Amazon Cognito verify user passwords before it issues your custom challenges. Here is an overview of the process:

1. To start, have your app initiate sign-in by calling `InitiateAuth` or `AdminInitiateAuth` with the `AuthParameters` map, including `CHALLENGE_NAME: SRP_A`, and values for `SRP_A` and `USERNAME`.
2. Your define auth challenge Lambda trigger is invoked with an initial session that contains `challengeName: SRP_A` and `challengeResult: true`.
3. After receiving those inputs, your Lambda function responds with `challengeName: PASSWORD_VERIFIER`, `issueTokens: false`, `failAuthentication: false`.
4. If the password verification succeeds, your Lambda function is invoked again with a new session containing `challengeName: PASSWORD_VERIFIER` and `challengeResult: true`.

5. Your Lambda function initiates your custom challenges by responding with `challengeName: CUSTOM_CHALLENGE`, `issueTokens: false`, and `failAuthentication: false`. If you don't want to start your custom auth flow with password verification, you can initiate sign-in with the `AuthParameters` map including `CHALLENGE_NAME: CUSTOM_CHALLENGE`.
6. The challenge loop repeats until all challenges are answered.

Topics

Define Auth Challenge Lambda Trigger Parameters

These are the parameters required by this Lambda function in addition to the common parameters.

JSON

```
{
    "request": {
        "userAttributes": {
            "string": "string",
                . . .
        },
        "session": [
            ChallengeResult,
            . . .
        ],
        "clientMetadata": {
            "string": "string",
            . . .
        },
        "userNotFound": boolean
    },
    "response": {
        "challengeName": "string",
        "issueTokens": boolean,
        "failAuthentication": boolean
}
```

Define Auth Challenge Request Parameters

These are the parameters provided to your Lambda function when it is invoked.

userAttributes

One or more name-value pairs representing user attributes.

userNotFound

A Boolean that is populated when `PreventUserExistenceErrors` is set to `ENABLED` for your user pool client. A value of `true` means that the user id (user name, email address, etc.) did not match any existing users. When `PreventUserExistenceErrors` is set to `ENABLED`, the service will not report back to the app that the user does not exist. The recommended best practice is for your Lambda functions to maintain the same user experience including latency so the caller cannot detect different behavior when the user exists or doesn't exist.

session

an array of `ChallengeResult` elements, each of which contains the following elements:

challengeName

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

Important
You should always check `challengeName` in your `DefineAuthChallenge` Lambda trigger to make sure it matches the expected value when determining if a user has successfully authenticated and should be issued tokens.

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`.

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function that you specify for the define auth challenge trigger. You can pass this data to your Lambda function by using the `ClientMetadata` parameter in the AdminRespondToAuthChallenge and RespondToAuthChallenge API operations.

Define Auth Challenge Response Parameters

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

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 been 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.

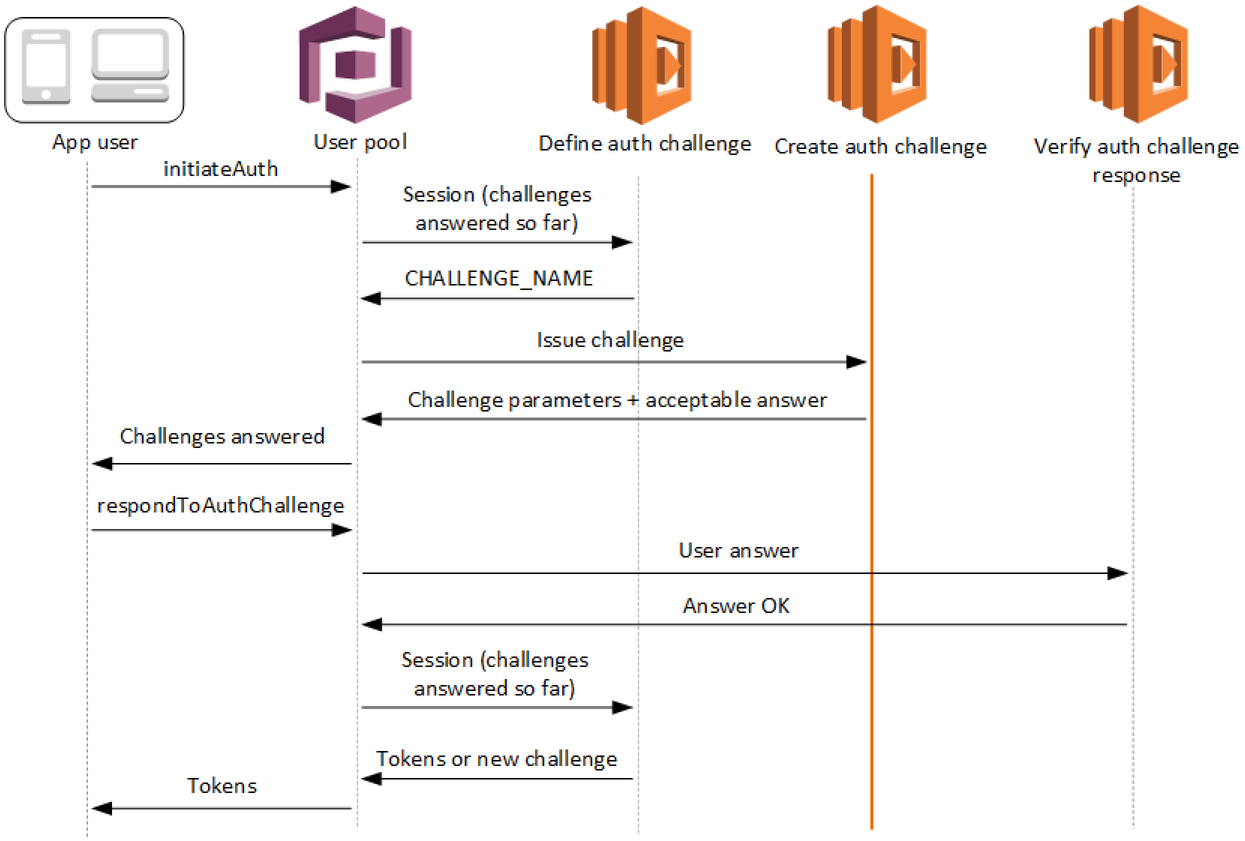
Define Auth Challenge Example

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

Node.js

```
exports.handler = (event, context, callback) => {
    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;
    }

    // Return to Amazon Cognito
    callback(null, event);
}
```

Create Auth Challenge Lambda Trigger



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. It creates a custom authentication flow.

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.

The challenge loop will repeat until all challenges are answered.

Topics

Create Auth Challenge Lambda Trigger Parameters

These are the parameters required by this Lambda function in addition to the common parameters.

JSON

```
{
    "request": {
        "userAttributes": {
            "string": "string",
            . . .
        },
        "challengeName": "string",
        "session": [
            ChallengeResult,
            . . .
        ],
        "clientMetadata": {
            "string": "string",
            . . .
        },
        "userNotFound": boolean
    },
    "response": {
        "publicChallengeParameters": {
            "string": "string",
            . . .
        },
        "privateChallengeParameters": {
            "string": "string",
            . . .
        },
        "challengeMetadata": "string"
    }
}
```

Create Auth Challenge Request Parameters

userAttributes

One or more name-value pairs representing user attributes.

userNotFound

This boolean is populated when `PreventUserExistenceErrors` is set to `ENABLED` for your User Pool client.

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"`.

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function that you specify for the create auth challenge trigger. You can pass this data to your Lambda function by using the ClientMetadata parameter in the AdminRespondToAuthChallenge and RespondToAuthChallenge API actions.

Create Auth Challenge Response Parameters

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.

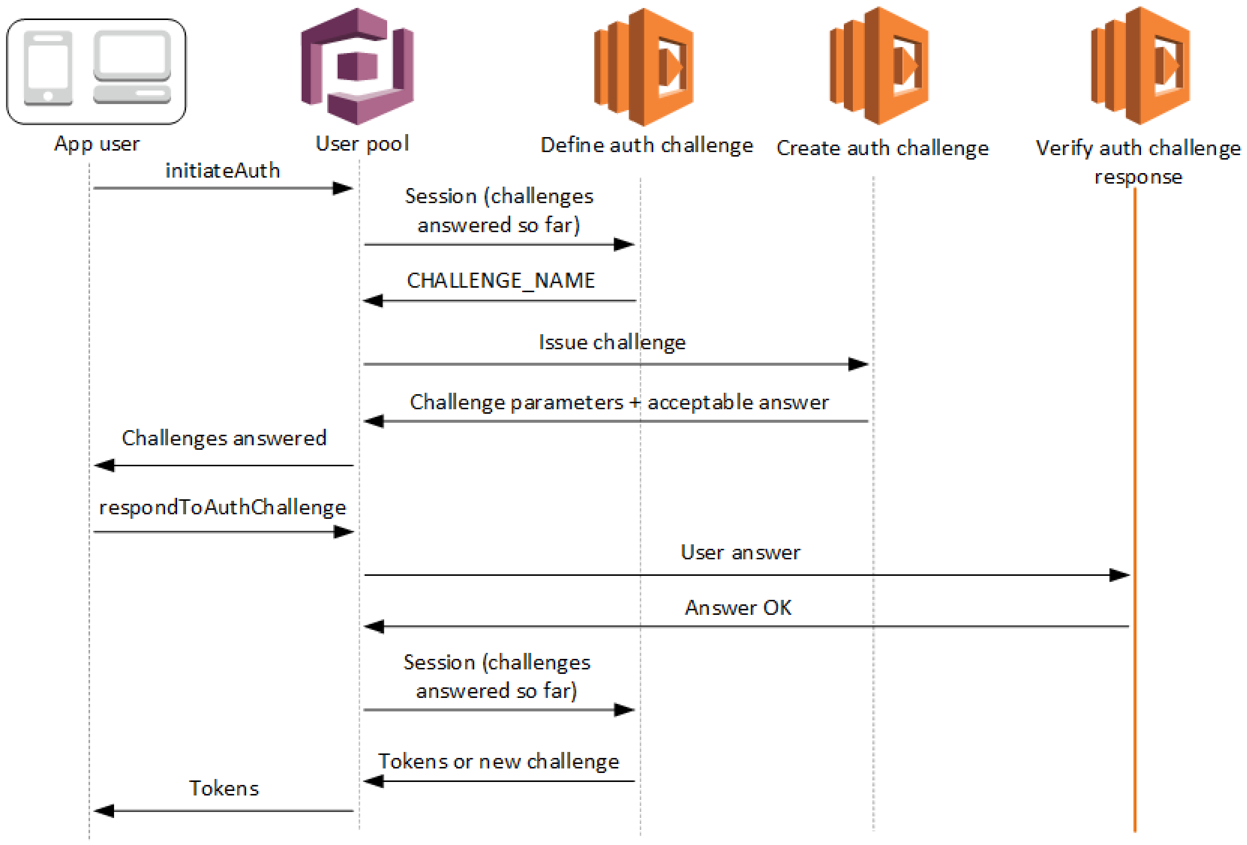
Create Auth Challenge Example

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.

Node.js

```
exports.handler = (event, context, callback) => {
    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';
    }

    Return to Amazon Cognito
    callback(null, event);
}
```

Verify Auth Challenge Response Lambda 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. It is part of a user pool custom authentication flow.

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.

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

The challenge loop will repeat until all challenges are answered.

Topics

Verify Auth Challenge Lambda Trigger Parameters

These are the parameters required by this Lambda function in addition to the common parameters.

JSON

```
{
    "request": {
        "userAttributes": {
            "string": "string",
            . . .
        },
        "privateChallengeParameters": {
            "string": "string",
            . . .
        },
        "challengeAnswer": {
            "string": "string",
            . . .
        },
        "clientMetadata": {
            "string": "string",
            . . .
        },
        "userNotFound": boolean
    },
    "response": {
        "answerCorrect": boolean
    }
}
```

Verify Auth Challenge Request Parameters

userAttributes

One or more name-value pairs representing user attributes.

userNotFound

This boolean is populated when `PreventUserExistenceErrors` is set to `ENABLED` for your User Pool client.

privateChallengeParameters

This parameter comes from the Create Auth Challenge trigger, and is compared against a user's **challengeAnswer** to determine whether the user passed the challenge.

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

challengeAnswer

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

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function that you specify for the verify auth challenge trigger. You can pass this data to your Lambda function by using the ClientMetadata parameter in the AdminRespondToAuthChallenge and RespondToAuthChallenge API actions.

Verify Auth Challenge Response Parameters

answerCorrect

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

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.

Node.js

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

    // Return to Amazon Cognito
    callback(null, event);
}
```

Pre Token Generation Lambda Trigger

Amazon Cognito invokes this trigger before token generation allowing you to customize identity token claims.

This Lambda trigger allows you to customize an identity token before it is generated. You can use this trigger to add new claims, update claims, or suppress claims in the identity token. To use this feature, you can associate a Lambda function from the Amazon Cognito user pools console or by updating your user pool through the AWS CLI.

There are some claims which cannot be modified. These include `acr`, `amr`, `aud`, `auth_time`, `azp`, `exp`, `iat`, `identities`, `iss`, `sub`, `token_use`, `nonce`, `at_hash`, and `cognito:username`.

Topics

Pre Token Generation Lambda Trigger Sources

triggerSource value	Triggering event
`TokenGeneration_HostedAuth`	Called during authentication from the Amazon Cognito hosted UI sign-in page.
`TokenGeneration_Authentication`	Called after user authentication flows have completed.

triggerSource value	Triggering event
`TokenGeneration_NewPasswordChallenge`	Called after the user is created by an admin. This flow is invoked when the user has to change a temporary password.
`TokenGeneration_AuthenticateDevice`	Called at the end of the authentication of a user device.
`TokenGeneration_RefreshTokens`	Called when a user tries to refresh the identity and access tokens.

Pre Token Generation Lambda Trigger Parameters

These are the parameters required by this Lambda function in addition to the common parameters.

JSON

```
{
    "request": {
        "userAttributes": {
            "string": "string",
            . . .
        },
        "groupConfiguration": {
        "groupsToOverride": ["string", . . .],
        "iamRolesToOverride": ["string", . . .],
        "preferredRole": "string",
        "clientMetadata": {
            "string": "string",
            . . .
        }
    },
    "response": {
        "claimsOverrideDetails": {
            "claimsToAddOrOverride": {
                "string": "string",
                . . .
            },
            "claimsToSuppress": ["string", . . .],

            "groupOverrideDetails": {
                "groupsToOverride": ["string", . . .],
                "iamRolesToOverride": ["string", . . .],
                "preferredRole": "string"
            }
        }
    }
}
```

Pre Token Generation Request Parameters

groupConfiguration

The input object containing the current group configuration. It includes `groupsToOverride`, `iamRolesToOverride`, and `preferredRole`.

groupsToOverride

A list of the group names that are associated with the user that the identity token is issued for.

iamRolesToOverride

A list of the current IAM roles associated with these groups.

preferredRole

A string indicating the preferred IAM role.

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function that you specify for the pre token generation trigger. You can pass this data to your Lambda function by using the ClientMetadata parameter in the AdminRespondToAuthChallenge and RespondToAuthChallenge API actions.

Pre Token Generation Response Parameters

claimsToAddOrOverride

A map of one or more key-value pairs of claims to add or override. For group related claims, use groupOverrideDetails instead.

claimsToSuppress

A list that contains claims to be suppressed from the identity token.

Note
If a value is both suppressed and replaced, then it will be suppressed.

groupOverrideDetails

The output object containing the current group configuration. It includes `groupsToOverride`, `iamRolesToOverride`, and `preferredRole`.

The groupOverrideDetails object is replaced with the one you provide. If you provide an empty or null object in the response, then the groups are suppressed. To leave the existing group configuration as is, copy the value of the request's groupConfiguration object to the groupOverrideDetails object in the response, and pass it back to the service.

Pre Token Generation Example: Add a New Claim and Suppress an Existing Claim

This example uses the Pre Token Generation Lambda to add a new claim and suppresses an existing one.

Node.js

```
exports.handler = (event, context, callback) => {
    event.response = {
        "claimsOverrideDetails": {
            "claimsToAddOrOverride": {
                "attribute_key2": "attribute_value2",
                "attribute_key": "attribute_value"
            },
            "claimsToSuppress": ["email"]
        }
    };

    // Return to Amazon Cognito
    callback(null, event);
};
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object back to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that's relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
  "request": {},
  "response": {}
}
```

Pre Token Generation Example: Modify the User's Group Membership

This example uses the Pre Token Generation Lambda to modify the user's group membership.

Node.js

```
exports.handler = (event, context, callback) => {
    event.response = {
        "claimsOverrideDetails": {
            "claimsToAddOrOverride": {
                "attribute_key2": "attribute_value2",
                "attribute_key": "attribute_value"
            },
            "claimsToSuppress": ["email"],
            "groupOverrideDetails": {
                "groupsToOverride": ["group-A", "group-B", "group-C"],
                "iamRolesToOverride": ["arn:aws:iam::XXXXXXXXXXXX:role/sns_callerA",
 "arn:aws:iam::XXXXXXXXX:role/sns_callerB", "arn:aws:iam::XXXXXXXXXX:role/
sns_callerC"],
                "preferredRole": "arn:aws:iam::XXXXXXXXXXX:role/sns_caller"
            }
        }
    };

    // Return to Amazon Cognito
    callback(null, event);
};
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object back to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that's relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
  "request": {},
  "response": {}
}
```

Migrate User Lambda Trigger

Amazon Cognito invokes this trigger when a user does not exist in the user pool at the time of sign-in with a password, or in the forgot-password flow. After the Lambda function returns successfully, Amazon Cognito creates the user in the user pool. For details on the authentication flow with the user migration Lambda trigger see Importing Users into User Pools With a User Migration Lambda Trigger (p. 133).

You can migrate users from your existing user directory into Amazon Cognito user pools at the time of sign-in, or during the forgot-password flow with this Lambda trigger.

Topics

Migrate User Lambda Trigger Sources

triggerSource value	Triggering event
`UserMigration_Authentication`	User migration at the time of sign in.
`UserMigration_ForgotPassword`	User migration during forgot-password flow.

Migrate User Lambda Trigger Parameters

These are the parameters required by this Lambda function in addition to the common parameters.

JSON

```
{
    "userName": "string",
    "request": {
        "password": "string",
        "validationData": {
            "string": "string",
            . . .
        },
        "clientMetadata": {
       "string": "string",
       . . .
        }
    },
    "response": {
        "userAttributes": {
            "string": "string",
            . . .
        },
        "finalUserStatus": "string",
        "messageAction": "string",
        "desiredDeliveryMediums": [ "string", . . .],
        "forceAliasCreation": boolean
    }
}
```

Migrate User Request Parameters

userName

The username entered by the user.

password

The password entered by the user for sign-in. It is not set in the forgot-password flow.

validationData

One or more key-value pairs containing the validation data in the user's sign-in request. You can pass this data to your Lambda function by using the ClientMetadata parameter in the InitiateAuth and AdminInitiateAuth API actions.

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function that you specify for the migrate user trigger. You can pass this data to your Lambda function by using the ClientMetadata parameter in the AdminRespondToAuthChallenge and ForgotPassword API actions.

Migrate User Response Parameters

userAttributes

This field is required.

It must contain one or more name-value pairs representing user attributes to be stored in the user profile in your user pool. You can include both standard and custom user attributes. Custom attributes require the `custom:` prefix to distinguish them from standard attributes. For more information see Custom attributes.

Note
In order for users to reset their passwords in the forgot-password flow, they must have either a verified email or a verified phone number. Amazon Cognito sends a message containing a reset password code to the email or phone number in the user attributes.

Attributes	Requirement
Any attributes marked as required when you created your user pool	If any required attributes are missing during the migration, default values will be used.
`username`	Required if you have configured your user pool with email and/or preferred_username aliases in addition to username for sign-in, and the user has entered an email or phone number to sign-in. Otherwise, it is optional and will be used as the username instead of the username entered by the user. **Note** username must be unique in the user pool.

finalUserStatus

During sign-in, this attribute can be set to `CONFIRMED`, or not set, to auto-confirm your users and allow them to sign-in with their previous passwords. This is the simplest experience for the user.

If this attribute is set to RESET_REQUIRED, the user is required to change his or her password immediately after migration at the time of sign-in, and your client app needs to handle the `PasswordResetRequiredException` during the authentication flow.

> **Note**
> The password strength policy that is configured for the user pool will not be enforced during migration using Lambda trigger. If the password doesn't meet the configured password policy, it will still be accepted to allow user migration to continue. To enforce password strength policy and reject passwords that don't meet the policy, validate the password strength in your code. Then set finalUserStatus to RESET_REQUIRED if the password doesn't meet the policy.

messageAction

This attribute can be set to "SUPPRESS" to suppress the welcome message usually sent by Amazon Cognito to new users. If this attribute is not returned, the welcome message will be sent.

desiredDeliveryMediums

This attribute can be set to "EMAIL" to send the welcome message by email, or "SMS" to send the welcome message by SMS. If this attribute is not returned, the welcome message will be sent by SMS.

forceAliasCreation

If this parameter is set to "true" and the phone number or email address specified in the UserAttributes parameter already exists as an alias with a different user, the API call will migrate the alias from the previous user to the newly created user. The previous user will no longer be able to log in using that alias.

If this attribute is set to "false" and the alias exists, the user will not be migrated, and an error is returned to the client app.

If this attribute is not returned, it is assumed to be "false".

Example: Migrate a User with an Existing Password

This example Lambda function migrates the user with an existing password and suppresses the welcome message from Amazon Cognito.

Node.js

```
exports.handler = (event, context, callback) => {

    var user;

    if ( event.triggerSource == "UserMigration_Authentication" ) {

        // authenticate the user with your existing user directory service
        user = authenticateUser(event.userName, event.request.password);
        if ( user ) {
            event.response.userAttributes = {
                "email": user.emailAddress,
                "email_verified": "true"
            };
            event.response.finalUserStatus = "CONFIRMED";
            event.response.messageAction = "SUPPRESS";
            context.succeed(event);
        }
        else {
            // Return error to Amazon Cognito
            callback("Bad password");
        }
    }
    else if ( event.triggerSource == "UserMigration_ForgotPassword" ) {

        // Lookup the user in your existing user directory service
        user = lookupUser(event.userName);
        if ( user ) {
            event.response.userAttributes = {
                "email": user.emailAddress,
                // required to enable password-reset code to be sent to user
                "email_verified": "true"
            };
            event.response.messageAction = "SUPPRESS";
            context.succeed(event);
        }
        else {
            // Return error to Amazon Cognito
            callback("Bad password");
        }
    }
    else {
        // Return error to Amazon Cognito
        callback("Bad triggerSource " + event.triggerSource);
    }
};
```

Custom Message Lambda Trigger

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.

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.

Note
A custom message Lambda function with the `CustomMessage_AdminCreateUser` trigger returns a user name and verification code and so the request must include both `request.usernameParameter` and `request.codeParameter`.

Topics

Custom Message Lambda Trigger Sources

triggerSource value	Triggering event
`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.

triggerSource value	Triggering event
`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 they manually request it for a new email or phone number.
`CustomMessage_Authentication`	Custom message – To send MFA code during authentication.

Custom Message Lambda Trigger Parameters

These are the parameters required by this Lambda function in addition to the common parameters.

JSON

```
{
    "request": {
        "userAttributes": {
            "string": "string",
            . . .
        }
        "codeParameter": "####",
        "usernameParameter": "string",
        "clientMetadata": {
            "string": "string",
            . . .
        }
    },
    "response": {
        "smsMessage": "string",
        "emailMessage": "string",
        "emailSubject": "string"
    }
}
```

Custom Message Request Parameters

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.

usernameParameter

The username parameter. It is a required request parameter for the admin create user flow.

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function that you specify for the custom message trigger. You can pass this data to your Lambda function by using the ClientMetadata parameter in the following API actions:

- AdminResetUserPassword
- AdminRespondToAuthChallenge
- AdminUpdateUserAttributes
- ForgotPassword
- GetUserAttributeVerificationCode
- ResendConfirmationCode
- SignUp
- UpdateUserAttributes

Custom Message Response Parameters

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

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. If EmailSendingAccount is not DEVELOPER and EmailMessage is returned, 400 error code `com.amazonaws.cognito.identity.idp.model.InvalidLambdaResponseException` will be thrown. emailMessage is allowed only if UserPool's EmailSendingAccount is DEVELOPER.

emailSubject

The subject line for the custom message. If EmailSendingAccount is not DEVELOPER and EmailMessage is returned, 400 error code `com.amazonaws.cognito.identity.idp.model.InvalidLambdaResponseException` will be thrown. emailSubject is allowed only if UserPool's EmailSendingAccount is DEVELOPER.

Custom Message for Sign Up Example

This Lambda function is invoked to customize an email or SMS message when the service requires an app to send a verification code to the user.

A Lambda trigger can be 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.

Node.js

```
exports.handler = (event, context, callback) => {
    //
    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 to Amazon Cognito
    callback(null, event);
};
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object back to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that's relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
  "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>"
  }
}
```

Custom Message for Admin Create User Example

A custom message Lambda function with the `CustomMessage_AdminCreateUser` trigger returns a user name and verification code and so include both `request.usernameParameter` and `request.codeParameter` in the message body.

The code parameter value `"####"` is a placeholder for the temporary password and "username" is a placeholder for the username delivered to your 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.

The response includes messages for both SMS and email.

Node.js

```
exports.handler = (event, context, callback) => {
    //
    if(event.userPoolId === "theSpecialUserPool") {
        // Identify why was this function invoked
        if(event.triggerSource === "CustomMessage_AdminCreateUser") {
            // Ensure that your message contains event.request.codeParameter
 event.request.usernameParameter. This is the placeholder for the code and username
 that will be sent to your user.
            event.response.smsMessage = "Welcome to the service. Your user
 name is " + event.request.usernameParameter + " Your temporary password is " +
 event.request.codeParameter;
            event.response.emailSubject = "Welcome to the service";
            event.response.emailMessage = "Welcome to the service. Your user
 name is " + event.request.usernameParameter + " Your temporary password is " +
 event.request.codeParameter;
        }
        // Create custom message for other events
    }
    // Customize messages for other user pools

    // Return to Amazon Cognito
    callback(null, event);
};
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object back to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that's relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
  "version": 1,
  "triggerSource": "CustomMessage_AdminCreateUser",
  "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": "####",
      "usernameParameter": "username"
  },
  "response": {
      "smsMessage": "<custom message to be sent in the message with code parameter and
 username parameter>"
```

```
      "emailMessage": "<custom message to be sent in the message with code parameter
and username parameter>"
      "emailSubject": "<custom email subject>"
  }
}
```

Custom sender Lambda triggers

Amazon Cognito user pools provide two Lambda triggers `CustomEmailSender` and `CustomSMSSender` to activate third-party email and SMS notifications. You can use your choice of SMS and email providers to send notifications to users from within your Lambda function code. These trigger your configured Lambda functions when notifications like confirmation codes, verification codes, or temporary passwords need to be sent to users. Amazon Cognito sends the code and temporary passwords (secrets) to your triggered Lambda functions. The secrets are encrypted using an AWS KMS customer managed key and the AWS Encryption SDK. The AWS Encryption SDK is a client-side encryption library that helps you to encrypt and decrypt generic data.

Note
You can use the AWS CLI or SDK to configure your user pools to use these Lambda triggers. These configurations aren't available from Amazon Cognito console.

CustomEmailSender (p. 108)

Amazon Cognito invokes this trigger to send email notifications to users.

CustomSMSSender (p. 111)

Amazon Cognito invokes this trigger to send SMS notifications to users.

Resources

The following resources can assist you with using the `CustomEmailSender` and `CustomSMSSender` triggers.

AWS KMS

AWS KMS is a managed service that makes it easy for you to create and control customer master keys (CMKs), which are the encryption keys that are used to encrypt your data. For more information see, What is AWS Key Management Service?.

Customer master key

A customer master key (CMK) is a logical representation of a master key. The CMK includes metadata, such as the key ID, creation date, description, and key state. The CMK also contains the key material used to encrypt and decrypt data. For more information see, Customer master keys.

Symmetric customer master key

A symmetric customer master key represents a 256-bit encryption key that never leaves AWS KMS unencrypted. To use a symmetric CMK, you must call AWS KMS. Symmetric keys are used in symmetric encryption, where the same key is used for encryption and decryption. For more information see, Symmetric customer master keys.

Custom Email Lambda Trigger

The `CustomEmailSender` trigger is invoked to enable a third-party provider to send email notifications to your users from within your Lambda function code. Using this trigger involves five main steps:

Note

The `CustomEmailSender` trigger isn't available in the Amazon Cognito console.

- Create a Lambda function for the `CustomEmailSender`. Amazon Cognito uses the AWS Encryption SDK to encrypt the secrets (temporary passwords or authorization codes).
- Create an encryption key in AWS KMS. This key will be used to encrypt temporary passwords and authorization codes that Amazon Cognito generates. You can then decrypt these secrets in the custom sender Lambda function to send them to the end-user in plain-text.
- Grant Amazon Cognito service principal cognito-idp.amazonaws.com access to invoke the Lambda function.
- Edit the code in your Lambda function to use customer senders or third-party providers.
- Update the existing user pool to add custom sender Lambda triggers.

Important

For additional security, you must configure a symmetric customer master key in AWS KMS (KMS), when `CustomEmailSender` or `CustomSMSSender` is configured with your user pool. Amazon Cognito uses your configured KMS key to encrypt codes or temporary passwords. Amazon Cognito sends the base64 encoded ciphertext to your Lambda functions. For more information see, Symmetric customer master keys

Enable the `CustomEmailSender` Lambda trigger

You can enable the `CustomEmailSender` Lambda trigger using a Lambda function.

Step 1: Create a Lambda function

Create a Lambda function for the `CustomEmailSender` trigger. Amazon Cognito uses the AWS Encryption SDK to encrypt the secrets (temporary passwords or authorization codes).

Step 2: Create an encryption key in AWS KMS

Create an encryption key in AWS KMS. This key will be used to encrypt temporary passwords and authorization codes that Amazon Cognito generates. You can then decrypt these secrets in the custom sender Lambda function to be able to send them to the end user in plaintext.

Step 3: Grant Amazon Cognito service principal cognito-idp.amazonaws.com access to invoke the Lambda function

Use the following command:

```
       aws lambda add-permission --function-name lambda_arn --statement-id
 "CognitoLambdaInvokeAccess" --action lambda:InvokeFunction --principal cognito-
idp.amazonaws.com
```

Step 4: Edit the code to use custom sender

Amazon Cognito uses AWS Encryption SDK to encrypt secrets (temporary passwords and authorization codes) before sending them to the custom sender Lambda function. You need to decrypt these secrets before sending them to end users using the custom provider of your choice. To use the AWS Encryption SDK with your Lambda function, you need to package the SDK with your function. For information, see Installing the AWS Encryption SDK for JavaScript. You can also update the Lambda package by completing the following steps.

1. Export the Lambda function package from the console.
2. Unzip the package.

3. Add the AWS Encryption SDK to the package. For example, if you are using Node.js, then add the `node_modules` directory and include the libraries from @aws-crypto/client-node.
4. Recreate the package.
5. Update the Lambda function code from the modified directory.

Step 5: Update user pool to add custom sender Lambda triggers

Update the user pool to add the `CustomEmailSender` trigger.

```
             #Send the parameter to update-user-pool along with any existing user pool
 configurations.

               --lambda-config "CustomEmailSender={LambdaVersion=V1_0,LambdaArn= lambda-
arn },KMSKeyID= key-id"
```

The following Node.js example shows how to use the `CustomEmailSender` Lambda function.

```
           const AWS = require('aws-sdk');
           const b64 = require('base64-js');
           const encryptionSdk = require('@aws-crypto/client-node');


           #Configure the encryption SDK client with the KMS key from the environment
 variables.

           const { encrypt, decrypt } =
 encryptionSdk.buildClient(encryptionSdk.CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT);
           const generatorKeyId = process.env.KEY_ALIAS;
           const keyIds = [ process.env.KEY_ID ];
           const keyring = new encryptionSdk.KmsKeyringNode({ generatorKeyId, keyIds })
           exports.handler = async (event) => {


           #Decrypt the secret code using encryption SDK.
           let plainTextCode;
           if(event.request.code){
           const { plaintext, messageHeader } = await decrypt(keyring,
 b64.toByteArray(event.request.code));
           plainTextCode = plaintext
           }

           #PlainTextCode now has the decrypted secret.

           if(event.triggerSource == 'CustomEmailSender_SignUp'){

           #Send email to end-user using custom or 3rd party provider.
           #Include temporary password in the email.

           }else if(event.triggerSource == 'CustomEmailSender_ResendCode'){

           }else if(event.triggerSource == 'CustomEmailSender_ForgotPassword'){

           }else if(event.triggerSource == 'CustomEmailSender_UpdateUserAttribute'){

           }else if(event.triggerSource == 'CustomEmailSender_VerifyUserAttribute'){

           }else if(event.triggerSource == 'CustomEmailSender_AdminCreateUser'){
           }else if(event.triggerSource ==
'CustomEmailSender_AccountTakeOverNotification'){

           }

           return;
           };
```

Custom Email sender Lambda trigger sources

The following table shows the triggering event for custom email trigger sources in your Lambda code.

`TriggerSource value`	Triggering event
`CustomEmailSender_SignUp`	To send the confirmation code after sign-up.
`CustomEmailSender_ResendCode`	To resend the temporary password to a new user.
`CustomEmailSender_ForgotPassword`	To resend the confirmation code to an existing user.
`CustomEmailSender_UpdateUserAttribute`	When a user's email is changed, this trigger sends a verification code automatically to the user. Cannot be used for other attributes.
`CustomEmailSender_VerifyUserAttribute`	This trigger sends a verification code to the user when they manually request it for a new email address.
`CustomEmailSender_AdminCreateUser`	To send the temporary password to a new user.
`CustomEmailSender_AccountTakeOverNotification`	This trigger sends customers a notification when an attempt to take over their account has been detected.

Custom SMS Sender Lambda Trigger

The `CustomSMSSender` trigger is invoked from within your Lambda function code to enable a third-party provider to send SMS notifications to your users. Using this trigger involves five main steps:

Note
The `CustomSMSSender` trigger isn't available in the Amazon Cognito console.

- Create a Lambda function for the `CustomSMSSender`.
- Create an encryption key in AWS KMS.
- Grant Amazon Cognito service principal cognito-idp.amazonaws.com access to invoke the Lambda function.
- Edit the code in your Lambda function to include third-party providers.
- Update your user pool to add custom triggers.

Important
For additional security you must configure a symmetric customer master key in AWS KMS, when `CustomEmailSender` or `CustomSMSSender` is configured with your user pool. Amazon Cognito uses your configured KMS key to encrypt codes or temporary passwords. Amazon

Cognito sends the base64 encoded ciphertext to your Lambda functions. For more information see, Symmetric customer master keys

Enable the `CustomSMSSender` Lambda trigger

You can enable the `CustomSMSSender` trigger using a Lambda function.

Step 1: Create a Lambda function

Create a Lambda function for the `CustomSMSSender` trigger. Amazon Cognito uses the AWS Encryption SDK to encrypt the secrets (temporary passwords or authorization codes).

Step 2: Create an encryption key in AWS KMS

Create an encryption key in AWS KMS. This key will be used to encrypt temporary passwords and authorization codes that Amazon Cognito generates. You can then decrypt these secrets in the custom sender Lambda function to be able to send them to the end user in plaintext.

Step 3: Grant Amazon Cognito service principal cognito-idp.amazonaws.com access to invoke the Lambda function

Use the following command to grant access to the Lambda function:

```
        aws lambda add-permission --function-name lambda_arn --statement-id
 "CognitoLambdaInvokeAccess" --action lambda:InvokeFunction --principal cognito-
idp.amazonaws.com
```

Step 4: Edit the code to use custom sender

Amazon Cognito uses AWS Encryption SDK to encrypt secrets (temporary passwords and authorization codes) before sending them to the custom sender Lambda function. You need to decrypt these secrets before sending them to end users using the custom provider of your choice. To use the AWS Encryption SDK with your Lambda function, you need to package the SDK with your function. For information, see Installing the AWS Encryption SDK for JavaScript. You can also update the Lambda package by completing the following steps.

1. Export the Lambda function package from the console
2. Unzip the package.
3. Add the AWS Encryption SDK to the package. For example, if you are using Node.js, then add the `node_modules` directory and include the libraries from @aws-crypto/client-node.
4. Recreate the package.
5. Update the Lambda function code from the modified directory.

Step 5: Update user pool to add custom sender Lambda triggers

Update the user pool to add the CustomSMSSender trigger.

Code examples

```
   #Send the parameter to update-user-pool along with any existing user pool
 configurations.

      --lambda-config "CustomSMSSender={LambdaVersion=V1_0,LambdaArn= lambda-arn
 },KMSKeyID= key-id"
```

The following Node.js example shows how to use the `CustomSMSSender` Lambda function.

```
            const AWS = require('aws-sdk');
            const b64 = require('base64-js');
            const encryptionSdk = require('@aws-crypto/client-node');

            #Configure the encryption SDK client with the KMS key from the environment
variables.

            const { encrypt, decrypt } =
encryptionSdk.buildClient(encryptionSdk.CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT);
            const generatorKeyId = process.env.KEY_ALIAS;
            const keyIds = [ process.env.KEY_ID ];
            const keyring = new encryptionSdk.KmsKeyringNode({ generatorKeyId, keyIds })
            exports.handler = async (event) => {

            #Decrypt the secret code using encryption SDK.

            let plainTextCode;
            if(event.request.code){
            const { plaintext, messageHeader } = await decrypt(keyring,
b64.toByteArray(event.request.code));
            plainTextCode = plaintext
            }

            #PlainTextCode now has the decrypted secret.

            if(event.triggerSource == 'CustomSMSSender_SignUp'){

           #Send sms to end-user using custom or 3rd party provider.
           #Include temporary password in the email.

            }else if(event.triggerSource == 'CustomSMSSender_ResendCode'){

            }else if(event.triggerSource == 'CustomSMSSender_ForgotPassword'){

            }else if(event.triggerSource == 'CustomSMSSender_UpdateUserAttribute'){

            }else if(event.triggerSource == 'CustomSMSSender_VerifyUserAttribute'){

            }else if(event.triggerSource == 'CustomSMSSender_AdminCreateUser'){

            }else if(event.triggerSource == 'CustomSMSSender_AccountTakeOverNotification'){

            }

            return;
            };
```

Topics

Custom SMS sender Lambda trigger sources

The following table shows the triggering event for custom SMS trigger sources in your Lambda code.

`TriggerSource value`	Triggering event
`CustomSMSSender_SignUp`	Custom message to send the confirmation code after sign-up.
`CustomSMSSender_ResendCode`	To send the temporary password to a new user.
`CustomSMSSender_ForgotPassword`	To resend the confirmation code to an existing user.
`CustomSMSSender_UpdateUserAttribute`	When a user's email or phone number is changed, this trigger sends a verification code automatically to the user. This can't be used for other attributes.
`CustomSMSSender_VerifyUserAttribute`	This trigger sends a verification code to the user when they manually request it for a new phone number.
`CustomSMSSender_Authentication`	To send MFA code during authentication.
`CustomSMSSender_AdminCreateUser`	To send the temporary password to a new user.

Using Amazon Pinpoint Analytics with Amazon Cognito User Pools

Amazon Cognito User Pools are integrated with Amazon Pinpoint to provide analytics for Amazon Cognito user pools and to enrich the user data for Amazon Pinpoint campaigns. Amazon Pinpoint provides analytics and targeted campaigns to drive user engagement in mobile apps using push notifications. With Amazon Pinpoint analytics support in Amazon Cognito user pools, you can track user pool sign-ups, sign-ins, failed authentications, daily active users (DAUs), and monthly active users (MAUs) in the Amazon Pinpoint console. You can drill into the data for different date ranges or attributes, such as device platform, device locale, and app version.

You can also set up user attributes that are specific to your app using the AWS Mobile SDK for Android or AWS Mobile SDK for iOS. Those can then be used to segment your users on Amazon Pinpoint and send them targeted push notifications. If you choose **Share user attribute data with Amazon Pinpoint** in the **Analytics** tab in the Amazon Cognito console, additional endpoints are created for user email addresses and phone numbers.

Find Amazon Cognito and Amazon Pinpoint Region mappings

The following table shows Region mappings between Amazon Cognito and Amazon Pinpoint. Use the table to find the Region where you built your Amazon Cognito user pool and the corresponding Amazon Pinpoint Region. Next, use these Regions to integrate Amazon Cognito and your Amazon Pinpoint project.

Amazon Cognito regions that support Amazon Pinpoint	Amazon Pinpoint project regions
ap-northeast-1	us-east-1
ap-northeast-2	us-east-1

Amazon Cognito regions that support Amazon Pinpoint	Amazon Pinpoint project regions
ap-south-1	us-east-1, ap-south-1
ap-southeast-1	us-east-1
ap-southeast-2	us-east-1, ap-southeast-2
ca-central-1	us-east-1
eu-central-1	us-east-1, eu-central-1
eu-west-1	us-east-1, eu-west-1
eu-west-2	us-east-1
us-east-1	us-east-1
us-east-2	us-east-1
us-west-2	us-east-1, us-west-2

Region mapping examples

- If you create a user pool in ap-northest-1, you have to create your Amazon Pinpoint project in us-east-1.
- If you create a user pool in ap-south-1, you have to create your Amazon Pinpoint project in either us-east-1 or ap-south-1.

Note
Amazon Pinpoint is available in several AWS Regions in North America, Europe, Asia, and Oceania. Besides the exceptions in the table, Amazon Cognito will only support in-Region Amazon Pinpoint integrations. If Amazon Pinpoint is available the same Region as Amazon Cognito, then Amazon Cognito sends events to Amazon Pinpoint projects within the same Region. If Amazon Pinpoint isn't available in the Region, then Amazon Cognito doesn't support Amazon Pinpoint integrations in that Region until Amazon Pinpoint becomes available. For Amazon Pinpoint detailed Region information, see Amazon Pinpoint endpoints and quotas.

Specifying Amazon Pinpoint Analytics Settings (AWS Management Console)

To specify analytics settings

1. Sign in to the Amazon Cognito console.
2. In the navigation pane, **Manage User Pools**, and choose the user pool you want to edit.
3. Choose the **Analytics** tab.
4. Choose **Add analytics and campaigns**.
5. Choose a **Cognito app client** from the list.
6. To map your Amazon Cognito app to an **Amazon Pinpoint project**, choose the Amazon Pinpoint project from the list.

   Note
   The Amazon Pinpoint project ID is a 32-character string that is unique to your Amazon Pinpoint project. It is listed in the Amazon Pinpoint console.

You can map multiple Amazon Cognito apps to a single Amazon Pinpoint project. However, each Amazon Cognito app can only be mapped to one Amazon Pinpoint project.
In Amazon Pinpoint, each project should be a single app. For example, if a game developer has two games, each game should be a separate Amazon Pinpoint project, even if both games use the same Amazon Cognito user pool. For more information on Amazon Pinpoint projects, see Create a project in Amazon Pinpoint.

7. Choose **Share user attribute data with Amazon Pinpoint** if you want Amazon Cognito to send email addresses and phone numbers to Amazon Pinpoint in order to create additional endpoints for users. After the account phone number and email address are verified, they are only shared with Amazon Pinpoint if they are available to the user account.

   Note
   An *endpoint* uniquely identifies a user device to which you can send push notifications with Amazon Pinpoint. For more information about endpoints, see Adding Endpoints in the *Amazon Pinpoint Developer Guide*.
8. Choose **Save changes**.
9. To specify additional app mappings, choose **Add another app mapping**.
10. Choose **Save changes**.

Specifying Amazon Pinpoint Analytics Settings (AWS CLI and AWS API)

Use the following commands to specify Amazon Pinpoint analytics settings for your user pool.

To specify the analytics settings for your user pool's existing client app at app creation time

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

To update the analytics settings for your user pool's existing client app

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

Note
Amazon Cognito supports in-Region integrations when you use `ApplicationArn`

Managing Users in User Pools

After you create a user pool, you can create, confirm, and manage users accounts. With Amazon Cognito user pools groups you can manage your users and their access to resources by mapping IAM roles to groups.

You can import your users into a user pool with a user migration Lambda trigger. This approach enables seamless migration of users from your existing user directory to user pools when they sign in to your user pool for the first time.

Topics

Signing Up and Confirming User Accounts

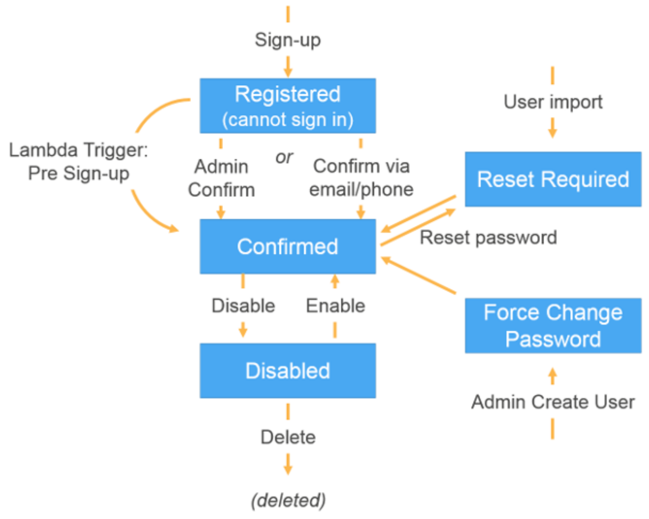
User accounts are added to your user pool in one of the following ways:

- 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 From a CSV File (p. 134).
- 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 (p. 124).

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:



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. The code or link is valid for 24 hours.

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.

Verifying Contact Information at Sign-Up

When new users sign up in your app, you probably want them to provide at least one contact method. For example, with your users' contact information, you might:

- Send a temporary password when a user chooses to reset his or her password.
- Notify users when their personal or financial information is updated.
- Send promotional messages, such as special offers or discounts.
- Send account summaries or billing reminders.

For use cases like these, it's important that you send your messages to a verified destination. Otherwise, you might send your messages to an invalid email address or phone number that was typed incorrectly. Or worse, you might send sensitive information to bad actors who pose as your users.

To help ensure that you send messages only to the right individuals, configure your Amazon Cognito user pool so that users must provide the following when they sign up:

a. An email address or phone number.
b. A verification code that Amazon Cognito sends to that email address or phone number.

By providing the verification code, a user proves that he or she has access to the mailbox or phone that received the code. After the user provides the code, Amazon Cognito updates the information about the user in your user pool by:

- Setting the user's status to `CONFIRMED`.
- Updating the user's attributes to indicate that the email address or phone number is verified.

To view this information, you can use the Amazon Cognito console. Or, you can use the `AdminGetUser` API action, the `admin-get-user` command with the AWS CLI, or a corresponding action in one of the AWS SDKs.

If a user has a verified contact method, Amazon Cognito automatically sends a message to the user when the user requests a password reset.

To Configure Your User Pool to Require Email or Phone Verification

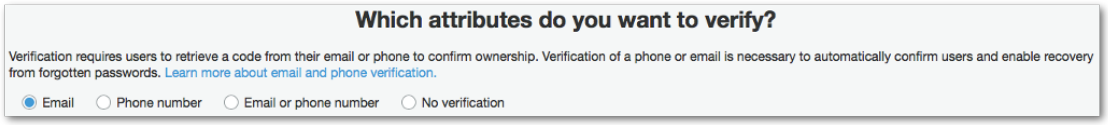
By requiring email or phone verification, you help ensure that you have a reliable way to contact your users. Complete the following steps to configure your user pool by using the Amazon Cognito console.

Before you begin
If you don't already, you need to have a user pool in your account. To create one, see Getting Started with User Pools (p. 20).

To configure your user pool

1. Sign in to the AWS Management Console and open the Amazon Cognito console at https://console.aws.amazon.com/cognito.
2. Choose **Manage User Pools**.
3. On the **Your User Pools** page, choose the user pool that you want to configure.
4. In the navigation menu on the left, choose **MFA and verifications**.

   The options for email or phone verification are shown under **Which attributes do you want to verify?**



Which attributes do you want to verify?

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. Learn more about email and phone verification.

(●) Email ( ) Phone number ( ) Email or phone number ( ) No verification

5. Choose one of the following options:

   Email

      If you choose this option, Amazon Cognito emails a verification code when the user signs up. Choose this option if you typically communicate with your users through email. For example, you will want to use verified email addresses if you send billing statements, order summaries, or special offers.

   Phone number

      If you choose this option, Amazon Cognito sends a verification code through SMS when the user signs up. Choose this option if you typically communicate with your users through SMS. For example, you will want to use verified phone numbers if you send delivery notifications, appointment confirmations, or alerts.

   Email or phone number

      Choose this option if you don't require all users to have the same verified contact method. In this case, the sign-up page in your app could ask users to verify only their preferred contact method. When Amazon Cognito sends a verification code, it sends the code to the contact method provided in the `SignUp` request from your app. If a user provides both an email address and a phone number, and your app provides both contact methods in the `SignUp` request, Amazon Cognito sends a verification code only to the phone number.

      If you require users to verify both an email address and a phone number, choose this option. Amazon Cognito verifies one contact method when the user signs up, and your app must verify the other contact method after the user signs in. For more information, see If You Require Users to Confirm Both Email Addresses and Phone Numbers (p. 120).

   None

      If you choose this option, Amazon Cognito doesn't send verification codes when users sign up. Choose this option if you are using a custom authentication flow that verifies at least one contact method without using verification codes from Amazon Cognito. For example, you might use a pre sign-up Lambda trigger that automatically verifies email addresses that belong to a specific domain.

If you don't verify your users' contact information, they might be unable to use your app in some cases. Remember that users require verified contact information to:

- Reset their passwords. When a user does an action in your app that calls the `ForgotPassword` API action, Amazon Cognito sends a temporary password to the user's email address or phone number. Amazon Cognito sends this password only if the user has at least one verified contact method.
- Sign in by using an email address or phone number as an alias. If you configure your user pool to allow these aliases, then a user can sign in with an alias only if the alias is verified. For more information, see Overview of Aliases (p. 166).

6. Choose **Save changes**.

Authentication Flow with Email or Phone Verification

If your user pool requires users to verify their contact information, your app must facilitate the following flow when a user signs up:

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. The code is valid for 24 hours
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.

If You Require Users to Confirm Both Email Addresses and Phone Numbers

Amazon Cognito verifies only one contact method when a user signs up. In cases where Amazon Cognito must choose between verifying an email address or phone number, it chooses to verify the phone number by sending a verification code through SMS. For example, if you configure your user pool to allow users to verify either email addresses or phone numbers, and if your app provides both of these attributes upon sign-up, Amazon Cognito verifies only the phone number. After a user verifies his or her phone number, Amazon Cognito sets the user's status to `CONFIRMED`, and the user is allowed to sign in to your app.

After the user signs in, your app can provide the option to verify the contact method that wasn't verified during sign-up. To verify this second method, your app calls the `VerifyUserAttribute` API action. Note that this action requires an `AccessToken` parameter, and Amazon Cognito only provides access tokens for authenticated users. Therefore, you can verify the second contact method only after the user signs in.

If you require your users to verify both email addresses and phone numbers, do the following:

1. Configure your user pool to allow users to verify email address or phone numbers.
2. In the sign-up flow for your app, require users to provide both an email address and a phone number. Call the `SignUp` API action, and provide the email address and phone number for the `UserAttributes` parameter. At this point, Amazon Cognito sends a verification code to the user's phone.
3. In your app interface, present a confirmation page where the user enters the verification code. Confirm the user by calling the `ConfirmSignUp` API action. At this point, the user's status is `CONFIRMED`, and the user's phone number is verified, but the email address is not verified.
4. Present the sign-in page, and authenticate the user by calling the `InitiateAuth` API action. After the user is authenticated, Amazon Cognito returns an access token to your app.
5. Call the `GetUserAttributeVerificationCode` API action. Specify the following parameters in the request:

   - `AccessToken` – The access token returned by Amazon Cognito when the user signed in.
   - `AttributeName` – Specify `"email"` as the attribute value.

   Amazon Cognito sends a verification code to the user's email address.
6. Present a confirmation page where the user enters the verification code. When the user submits the code, call the `VerifyUserAttribute` API action. Specify the following parameters in the request:

   - `AccessToken` – The access token returned by Amazon Cognito when the user signed in.
   - `AttributeName` – Specify `"email"` as the attribute value.
   - `Code` – The verification code that the user provided.

   At this point, the email address is verified.

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 with 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. 126) in Creating User Accounts as Administrator (p. 124) and the Custom Message trigger in Customizing User Pool Workflows with 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 From a CSV File (p. 134)) 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. 143).

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.

Sending Emails While Testing Your App

Amazon Cognito emails your users when they create and manage their accounts in the client app for your user pool. If you configure your user pool to require email verification, Amazon Cognito sends an email when:

- A user signs up.
- A user updates their email address.
- A user performs an action that calls the `ForgotPassword` API action.
- You create a user account as an administrator.

Depending on the action that initiates the email, the email contains a verification code or a temporary password. Your users must receive these emails and understand the message. Otherwise, they might be unable to sign in and use your app.

To ensure that emails send successfully and that the message looks correct, test the actions in your app that initiate email deliveries from Amazon Cognito. For example, by using the sign-up page in your app, or by using the `SignUp` API action, you can initiate an email by signing up with a test email address. When you test in this way, remember the following:

Important

When you use an email address to test actions that initiate emails from Amazon Cognito, don't use a fake email address (one that has no mailbox). Use a real email address that will receive the email from Amazon Cognito without creating a *hard bounce*.

A hard bounce occurs when Amazon Cognito fails to deliver the email to the recipient's mailbox, which always happens if the mailbox doesn't exist.

Amazon Cognito limits the number of emails that can be sent by AWS accounts that persistently incur hard bounces.

When you test actions that initiate emails, use one of the following email addresses to prevent hard bounces:

- An address for an email account that you own and use for testing. When you use your own email address, you receive the email that Amazon Cognito sends. With this email, you can use the verification code to test the sign-up experience in your app. If you customized the email message for your user pool, you can check that your customizations look correct.
- The mailbox simulator address, *success@simulator.amazonses.com*. If you use the simulator address, Amazon Cognito sends the email successfully, but you're not able to view it. This option is useful when you don't need to use the verification code and you don't need to check the email message.
- The mailbox simulator address with the addition of an arbitrary label, such as *success +user1@simulator.amazonses.com* or *success+user2@simulator.amazonses.com*. Amazon Cognito emails these addresses successfully, but you're not able to view the emails that it sends. This option is useful when you want to test the sign-up process by adding multiple test users to your user pool, and each test user has a unique email address.

Creating User Accounts as Administrator

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 with 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.

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. 149).

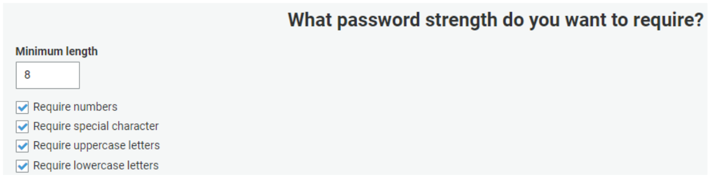
Creating a New User in the AWS Management Console

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 the required password strength.

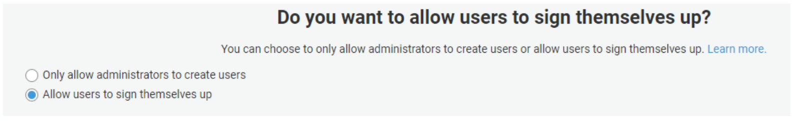


What password strength do you want to require?

Minimum length: 8

- [✓] Require numbers
- [✓] Require special character
- [✓] Require uppercase letters
- [✓] Require lowercase letters

- Specify whether to allow users to sign themselves up. This option is set by default.



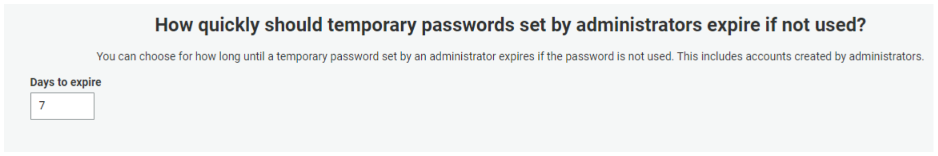
Do you want to allow users to sign themselves up?

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 temporary passwords set by administrators expire if not used?

You can choose for how long until a temporary password set by an administrator expires if the password is not used. This includes accounts created by administrators.

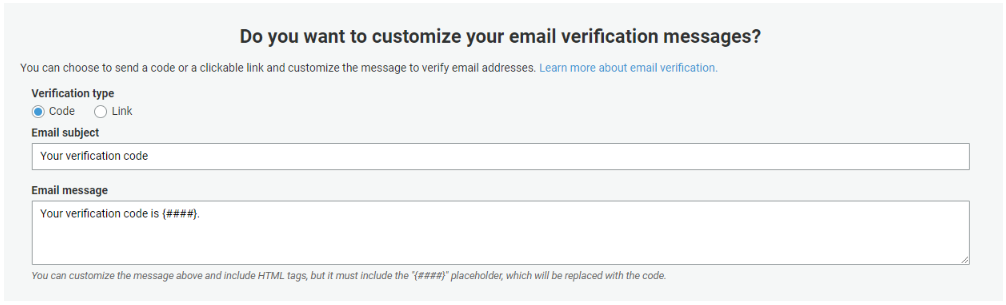
Days to expire: 7

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?

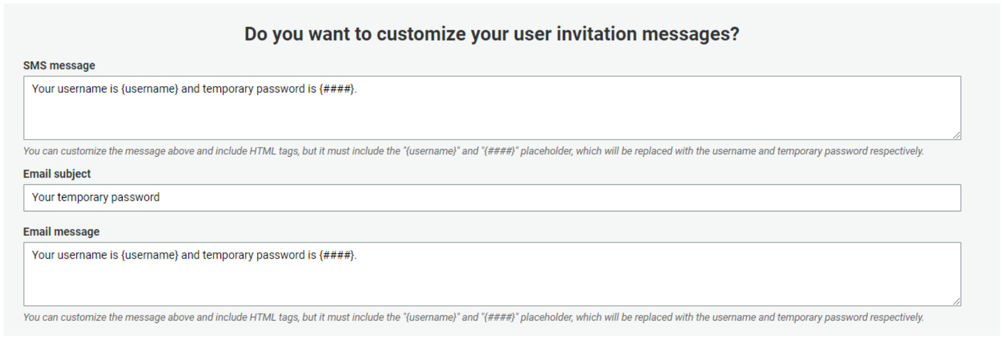
You can choose to send a code or a clickable link and customize the message to verify email addresses. Learn more about email verification.

Verification type: [✓ Code] Link

Email subject: Your verification code

Email message: Your verification code is {####}.

You can customize the message above and include HTML tags, 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 and include HTML tags, but it must include the "{username}" and "{####}" placeholder, which will be replaced with the username and temporary password respectively.

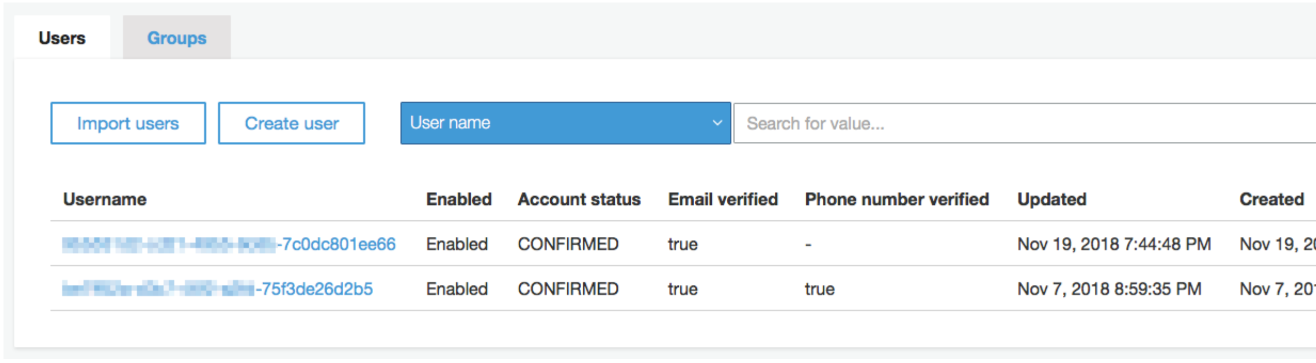
Email subject: Your temporary password

Email message: Your username is {username} and temporary password is {####}.

You can customize the message above and include HTML tags, but it must include the "{username}" and "{####}" placeholder, which will be replaced with the username and temporary password respectively.

Users Tab

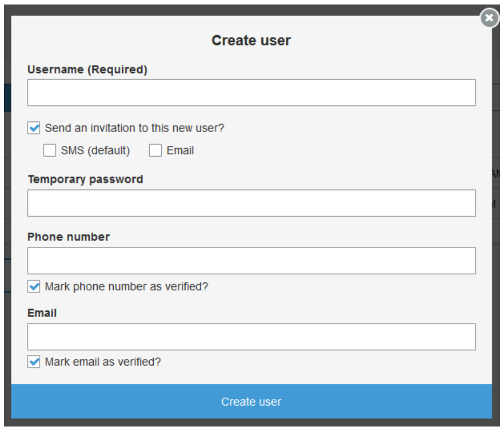
The **Users** tab in the **Users and groups** tab has a **Create user** button.



Username	Enabled	Account status	Email verified	Phone number verified	Updated	Created
[redacted]-7c0dc801ee66	Enabled	CONFIRMED	true	-	Nov 19, 2018 7:44:48 PM	Nov 19, 20
[redacted]-75f3de26d2b5	Enabled	CONFIRMED	true	true	Nov 7, 2018 8:59:35 PM	Nov 7, 20

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.



Create user

Username (Required)

✓ Send an invitation to this new user?

☐ SMS (default) ☐ Email

Temporary password

Phone number

✓ Mark phone number as verified?

Email

✓ Mark email as verified?

Create user

Adding Groups to a User Pool

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 your resources using an IAM role. IAM roles include trust policies and permission policies. The role trust policy specifies who can use the role. The permissions policies specify the actions and resources that your group members can access. When you create an IAM role, set up the role trust policy to allow your group users to assume the role. In the role permissions policies, specify the permissions that you want your group to have.

When you create a group in Amazon Cognito, you specify an IAM role by providing the role's ARN. When group members sign in using Amazon Cognito, they can receive temporary credentials from the identity pools. Their permissions are determined by the associated IAM role.

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.

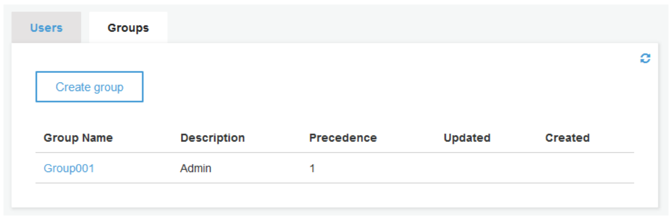
Limitations on Groups

User groups are subject to the following limitations:

- The number of groups you can create is limited by the Amazon Cognito service limits (p. 343).
- 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.

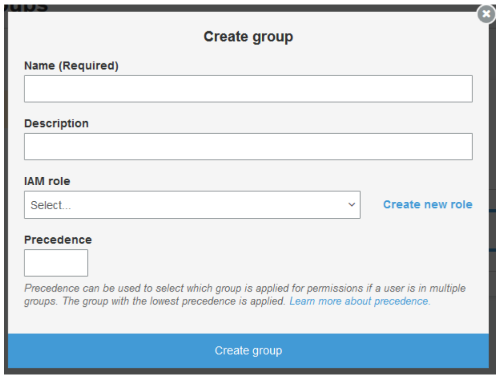
Creating a New Group in the AWS Management Console

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.

Managing and Searching for User Accounts

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 User Pools Reference (AWS Management Console) (p. 163).
- 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)
- sub

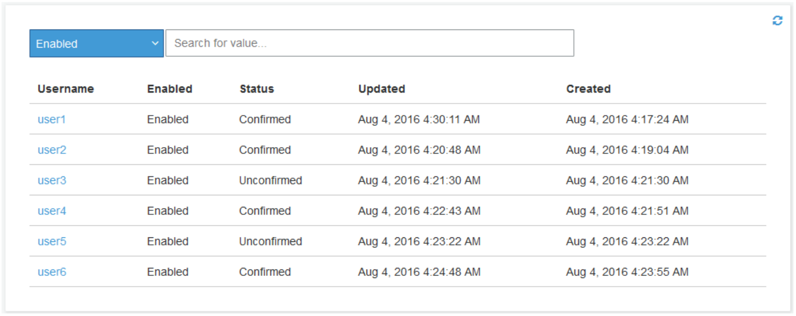
Searching for Users Using the AWS Management Console

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.

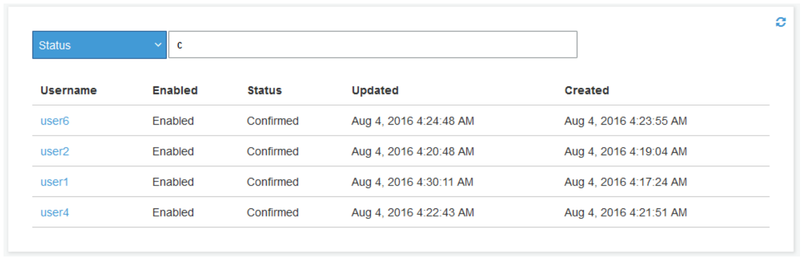
All of the following examples use the same user pool.

For example, if you want to list all users, leave the search box empty.



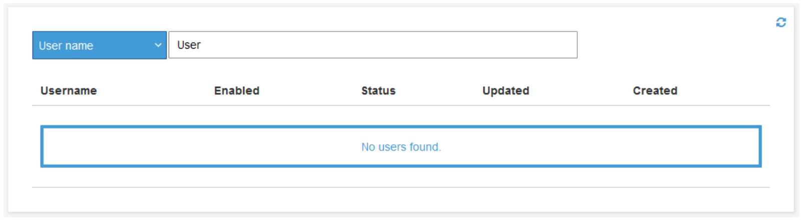
Username	Enabled	Status	Updated	Created
user1	Enabled	Confirmed	Aug 4, 2016 4:30:11 AM	Aug 4, 2016 4:17:24 AM
user2	Enabled	Confirmed	Aug 4, 2016 4:20:48 AM	Aug 4, 2016 4:19:04 AM
user3	Enabled	Unconfirmed	Aug 4, 2016 4:21:30 AM	Aug 4, 2016 4:21:30 AM
user4	Enabled	Confirmed	Aug 4, 2016 4:22:43 AM	Aug 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."



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**.



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"
}
```

Recovering User Accounts

The `AccountRecoverySetting` parameter enables you to customize which method a user can use to recover their password when they call the `ForgotPassword` API. `ForgotPassword` sends a recovery code to a verified email or a verified phone number. When you specify `AccountRecoverySetting`, the preferred setting is chosen from the priorities defined by `AccountRecoverySetting`.

When you define `AccountRecoverySetting` and a user has SMS MFA configured, SMS cannot be used as an account recovery mechanism. The priority for this setting is determined with 1 being of the highest priority. Cognito sends a verification to only one of the specified methods.

For example, `admin_only` is a value used when the administrator does not want the user to recover their account themselves, and would instead require them to contact the administrator to reset their account. You cannot use `admin_only` with any other account recovery mechanism.

If you do not specify `AccountRecoverySetting`, Amazon Cognito uses the legacy mechanism to determine the password recovery method. In this case, Cognito uses a verified phone first. If the verified phone is not found for the user, Cognito falls back and will use verified email next.

For more information about `AccountRecoverySetting`, see CreateUserPool and UpdateUserPool in the *Amazon Cognito Identity Provider API Reference*.

Forgot Password Behavior

In a given hour, we allow between 5 and 20 attempts for a user to request or enter a password reset code as part of forgot-password and confirm-forgot-password actions. The exact value depends on the risk parameters associated with the requests. Please note that this behavior is subject to change.

Importing Users into a User Pool

There are two ways you can import or migrate users from your existing user directory or user database into Amazon Cognito user pools. You can migrate users when they sign-in using Amazon Cognito for the first time with a user migration Lambda trigger. With this approach, users can continue using their existing passwords and will not have to reset them after the migration to your user pool. Alternatively, you can migrate users in bulk by uploading a CSV file containing the user profile attributes for all users. The following sections describe both these approaches.

Topics

Importing Users into User Pools With a User Migration Lambda Trigger

This approach enables seamless migration of users from your existing user directory to user pools when they use your new Amazon Cognito-enabled app for the first time, either during their first sign-in or during the forgot-password process. This migration is enabled by a user migration Lambda function which you need to configure in your user pool. For details on this Lambda trigger including request and response parameters, and example code see Migrate User Lambda Trigger Parameters (p. 100).

Before starting the user migration process, create a user migration Lambda function in your AWS account, and configure your user pool to use this Lambda function ARN for the user migration trigger. Add an authorization policy to your Lambda function to enable only the Amazon Cognito service account principal ("cognito-idp.amazonaws.com") and your user pool's SourceARN to invoke it, to prevent any other AWS customer's user pool from invoking your Lambda function. For more information, see Using Resource-Based Policies for AWS Lambda (Lambda Function Policies).

Steps during sign-in

1. The user opens your app and signs-in with the native sign-in UI screens from your app using the Cognito Identity Provider APIs from an AWS Mobile SDK, or using the hosted sign-in UI provided by Amazon Cognito that you leverage with the Amazon Cognito Auth SDK.
2. Your app sends the username and password to Amazon Cognito. If your app has a native sign-in UI and uses the Cognito Identity Provider SDK, your app must use the USER_PASSWORD_AUTH flow, in which the SDK sends the password to the server (your app must not use the default USER_SRP_AUTH flow since the SDK does not send the password to the server in the SRP authentication flow). The USER_PASSWORD_AUTH flow is enabled by setting AuthenticationDetails.authenticationType to "USER_PASSWORD".
3. Amazon Cognito checks if the username exists in the user pool, including as an alias for the user's email, phone number, or preferred_username. If the user does not exist, Amazon Cognito calls your

user migration Lambda function with parameters including the username and password, as detailed in Migrate User Lambda Trigger Parameters (p. 100).

4. Your user migration Lambda function should authenticate the user with your existing user directory or user database, by calling your existing sign-in service, and it should return user attributes to be stored in the user's profile in the user pool. If you would like users to continue to use their existing passwords, set the attribute `finalUserStatus` = "CONFIRMED" in the Lambda response. For the required attributes for the response see Migrate User Lambda Trigger Parameters (p. 100).

   Important
   Do not log the entire request event object in your user migration Lambda code (since that would include the password in the log record sent to the CloudWatch logs), and ensure you sanitize the logs so that passwords are not logged.

5. Amazon Cognito creates the user profile in your user pool, and returns tokens to your app client.
6. Your app client can now proceed with its normal functionality after sign-in.

After the user is migrated, we recommend that your native mobile apps use the `USER_SRP_AUTH` flow for sign-in. It authenticates the user using the Secure Remote Password (SRP) protocol without sending the password across the network, and provides security benefits over the `USER_PASSWORD_AUTH` flow used during migration.

In case of errors during migration, including client device or network issues, your app will get error responses from the Amazon Cognito user pools API, and the user account might or might not have been created in your user pool. The user should then attempt sign-in again, and if that fails repeatedly, then attempt to reset his or her password using the forgot-password flow in your app.

The forgot-password flow works in a similar way, except that no password is provided to your user migration Lambda function, and your function only looks up the user in your existing user directory, and returns attributes to be stored in your user pool. Then Amazon Cognito sends the user a reset password code by email or SMS, and the user can then set a new password in your app. If your app has a native UI, it needs to provide screens for the forgot-password flow. If your app uses the Amazon Cognito hosted UI, then we provide the UI screens.

Importing Users into User Pools From a CSV File

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. So, your users will be in a RESET_REQUIRED state when imported using this method.

Note
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.

Topics

Creating the CloudWatch Logs IAM Role (AWS CLI, API)

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 (AWS CLI, API)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**.
3. For a new role, choose **Create role**.
4. For **Select type of trusted entity**, choose **AWS service**.
5. In **Common use cases**, choose **EC2**.
6. Choose **Next: Permissions**.
7. In **Attach permissions policy**, choose **Create policy** to open a new browser tab and create a new policy from scratch.
8. On the **Create policy** page, choose the **JSON** tab.
9. In the **JSON** tab, copy and 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/*"
            ]
        }
    ]
}
```

10. Choose **Review policy**. Add a name and optional description and choose **Create policy**.

    After you create the policy, close that browser tab and return to your original tab.
11. In **Attach Policy**, choose **Next: Tags**.
12. (Optional) In **Tags**, add metadata to the role by entering tags as key–value pairs.

13. Choose **Next: Review**.
14. In **Review**, enter a **Role Name**.
15. (Optional) Enter a **Role Description**.
16. Choose **Create role**.
17. In the navigation pane of the IAM console, choose **Roles**.
18. In **Roles**, choose the role you created.
19. In **Summary**, choose the **Trust relationships** tab.
20. In the **Trust relationships** tab, choose **Edit trust relationship**.
21. Copy and 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"
            }
        ]
    }
```

22. Choose **Update Trust Policy**. You are now finished creating the role.
23. 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. 137).

Downloading the .csv File Header (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 (AWS 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 Configuring User Pool Attributes (p. 164).

- 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 (Console)

The following procedure describes how to import the users from the .csv file.

To import users from the .csv file (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 (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 (AWS CLI, API) (p. 135):

```
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 presigned 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 import job can be active at a time per account.

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

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 user name and password (via `InitiateAuth`).
2. Amazon Cognito returns `NotAuthorizedException` when `PreventUserExistenceErrors` is enabled. Otherwise, it returns `PasswordResetRequiredException`.
3. The app should direct the user into the `ForgotPassword` flow as outlined in the following procedure:

    a. The app calls `ForgotPassword(`*`user name`*`)`.

    b. 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.

    c. 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.

    d. The user enters the code and new password in the app.

    e. The app calls `ConfirmForgotPassword(`*`code`*`, `*`password`*`)`, which, if successful, sets the new password.

    f. The app should now direct the user to a sign-in page.

Email Settings for Amazon Cognito User Pools

Certain events in the client app for your user pool might cause Amazon Cognito to email your users. For example, if you configure your user pool to require email verification, Amazon Cognito sends an email when a user signs up for a new account in your app or resets their password. Depending on the action that initiates the email, the email contains a verification code or a temporary password.

To handle email delivery, you can use either of the following options:

- The default email functionality (p. 144) that is built into the Amazon Cognito service.
- Your Amazon SES configuration (p. 144).

These settings are reversible. When needed, you can update your user pool to switch between them.

Default Email Functionality

You can allow Amazon Cognito to handle email deliveries for you by using the default email functionality that comes with the service. When you use the default option, Amazon Cognito allows only a limited number of emails each day for your user pool. For specific limits information, see Quotas in Amazon Cognito (p. 343). For typical production environments, the default email limit is below the required delivery volume. To enable a higher delivery volume, you must use your Amazon SES email configuration.

With the default option, you can use either of the following email addresses as the FROM address:

- The default email address, which is no-reply@verificationemail.com.
- A custom email address that you own. Before you can use your own email address, you must verify it with Amazon SES, and you must grant Amazon Cognito permission to use it.

Amazon SES Email Configuration

Your application might require a higher delivery volume than what is available with the default option. To enable a higher delivery volume, configure your user pool to email your users by using your Amazon SES configuration. Using your Amazon SES configuration also provides a greater ability to monitor your email sending activity.

Before you can use your Amazon SES configuration, you must verify one or more email addresses with Amazon SES. You use a verified email addresses as the FROM email address that you assign to your user pool. Then, when Amazon Cognito sends an email, it uses your email address by calling Amazon SES on your behalf.

When you use your Amazon SES configuration, the email delivery limits for your user pool are the same limits that apply to your Amazon SES verified email address in your AWS account.

Note
Available regions for Amazon Cognito are us-east-1, us-east-2, us-west-2, eu-west-1, eu-west-2, eu-central-1, ap-northeast-1, ap-northeast-2, ap-south-1, ap-southeast-1, ap-southeast-2, and ca-central-1. Available Amazon SES regions are: us-east-1, us-west-2, eu-west-1.

Configuring Email for Your User Pool

Complete the following steps to configure the email settings for your user pool. Depending on settings that you want to use, you might need to complete steps with Amazon SES, AWS Identity and Access Management (IAM), and Amazon Cognito.

Note
The resources that you create in these steps can't be shared across AWS accounts. For example, you can't configure a user pool in one account with an Amazon SES email address that is in a different account. Therefore, if you use Amazon Cognito in multiple accounts, remember to repeat these steps in each of them.

Step 1: Verify Your Email Address with Amazon SES

Before you configure your user pool, you must verify one or more email addresses with Amazon SES if you want to do either of the following:

- Use your own email address as the FROM address
- Use your Amazon SES configuration to handle email delivery

By verifying your email address, you confirm that you own it, which helps prevent unauthorized use.

For the steps to this procedure, see Verifying an Email Address in the *Amazon Simple Email Service Developer Guide*.

Step 2: Move Your Account Out of the Amazon SES Sandbox

When you first begin using Amazon SES, your AWS account is placed in the Amazon SES sandbox. Amazon SES uses the sandbox to prevent fraud and abuse. If you are using your Amazon SES configuration to handle email delivery, you must move your AWS account out of the sandbox before Amazon Cognito can email your users.

You can skip this step if you are using the default Amazon Cognito email functionality.

In the sandbox, Amazon SES imposes restrictions on how many emails you can send and where you can send them. You can send emails only to addresses and domains that you have verified with Amazon SES, or you can send them to Amazon SES mailbox simulator addresses. While your AWS account remains in the sandbox, do not use your Amazon SES configuration for applications that are in production. In this situation, Amazon Cognito would be unable to send messages to your users' email addresses.

For the steps to move out of the sandbox, see Moving Out of the Amazon SES Sandbox in the *Amazon Simple Email Service Developer Guide.*

Step 3: Grant Email Permissions to Amazon Cognito

You might need to grant specific permissions to Amazon Cognito before it can email your users. The permissions that you grant, and the process that you use to grant them, depend on whether you are using the default email functionality or your Amazon SES configuration.

To Grant Permissions to Use the Default Email Functionality

If you configure your user pool to use the default email functionality, you use either of the following addresses as the FROM address from which Amazon Cognito emails your users:

- The default address
- A custom address, which must be a verified address in Amazon SES

If you are using the default email address, Amazon Cognito does not need additional permissions, and you can skip this step.

If you are using a custom address, Amazon Cognito needs additional permissions so that it can use this address to email your users. These permissions are granted by a *sending authorization policy*, which gets attached to the address in Amazon SES. If you use the Amazon Cognito console to add your custom address to your user pool, it automatically attaches the policy for you. But, if you configure your user pool outside of the console, for example by using the AWS CLI or Amazon Cognito API, you must attach the policy yourself.

For more information, see Using Sending Authorization with Amazon SES in the *Amazon Simple Email Service Developer Guide*.

Example Sending Authorization Policy

The following example is a sending authorization policy that allows Amazon Cognito to send email by using an Amazon SES verified email address.

```
{
    "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>"
        }
    ]
}
```

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*.

For more examples, see Amazon SES Sending Authorization Policy Examples in the *Amazon Simple Email Service Developer Guide*.

To Grant Permissions to Use Your Amazon SES Configuration

If you configure your user pool to use your Amazon SES configuration, Amazon Cognito needs additional permissions to call Amazon SES on your behalf when it emails your users. This authorization is granted with the IAM service.

When you configure your user pool with this option, Amazon Cognito creates a *service-linked role*, which is a type of IAM role, in your AWS account. This role contains the permissions that allow Amazon Cognito to access Amazon SES and send email with your address.

Before Amazon Cognito can create this role, the IAM permissions that you use to set up your user pool must include the `iam:CreateServiceLinkedRole` action. For more information about updating permissions in IAM, see Changing Permissions for an IAM User in the *IAM User Guide*.

For more information about the service-linked role that Amazon Cognito creates, see Using Service-Linked Roles for Amazon Cognito (p. 302).

Step 4: Configure Your User Pool

Complete the following steps if you want to configure your user pool with any of the following:

- A custom FROM address that appears as the email sender
- A custom REPLY-TO address that receives the messages that your users send to your FROM address
- Your Amazon SES configuration

You don't need to complete this procedure if you want to use the default Amazon Cognito email functionality and address.

To configure your email address

1. Sign in to the AWS Management Console and open the Amazon Cognito console at https://console.aws.amazon.com/cognito.

2. Choose **Manage User Pools**.
3. On the **Your User Pools** page, choose the user pool that you want to configure.
4. In the navigation menu on the left, choose **Message customizations**.
5. If you want to use a custom FROM address, choose **Add custom FROM address** and do the following:

   a. For **SES region**, choose the region that contains your verified email address.
   b. For **Source ARN**, choose your email address. The Amazon Cognito console allows you to choose only those email addresses that you have verified with Amazon SES in the selected region.
   c. For **FROM email address**, choose your email address. You can provide your email address or your email address with your name.
6. Under **Do you want to send emails through your Amazon SES Configuration?**, choose **Yes - Use Amazon SES** or **No - Use Cognito (Default)**.

   If you choose to use a Amazon SES, Amazon Cognito will create a service-linked role after you save your changes.
7. If you want to use a custom REPLY-TO address, choose **Add custom REPLY-TO address**. Then, specify the email address where you want to receive that messages that your users send to your FROM address.
8. When you finish setting your email account options, choose **Save changes**.

The **Message customizations** page also provides the options to customize your verification messages (p. 173) and customize your invitation messages (p. 174).

SMS message settings for Amazon Cognito user pools

Some Amazon Cognito events for your user pool might cause Amazon Cognito to send SMS text messages to your users. For example, if you configure your user pool to require phone verification, Amazon Cognito sends an SMS text message when a user signs up for a new account in your app or resets their password. Depending on the action that initiates the SMS text message, the message contains a verification code, a temporary password, or a welcome message.

Amazon Cognito uses Amazon Simple Notification Service (SNS) for delivery of SMS text messages. If this is the first time that you are sending a text message through Amazon Cognito or Amazon SNS, you will be placed in a sandbox environment in Amazon SNS. This will allow you to test your applications for SMS text messages. In the sandbox, messages can be sent only to verified phone numbers.

Setting up SMS messages for the first time in Amazon Cognito user pools

Amazon Cognito uses Amazon SNS to send SMS messages to your user pools. The first time that you set up Amazon SNS to send SMS text messages, your AWS account is placed in the Amazon SNS sandbox. Amazon SNS uses the sandbox to prevent fraud and abuse and to meet compliance requirements. In the sandbox, Amazon SNS imposes some restrictions. For example, you can send text messages to up to 10 phone numbers that you have verified with Amazon SNS. While your AWS account remains in the sandbox, do not use your Amazon SNS configuration for applications that are in production. When you're in the sandbox, Amazon Cognito can't send messages to your users' phone numbers.

To send SMS text messages to user pool users in production for the first time, you must complete the following tasks:

1. Confirm that you are in the SMS sandbox

2. Verify phone numbers for Amazon Cognito in Amazon SNS

3. Obtain an origination identity for sending SMS messages to U.S. phone numbers

4. Move your account out of Amazon SNS sandbox

5. Finish setting up the user pool in Amazon Cognito

STEP 1: Confirm that you are in the SMS sandbox

1. Sign in to the AWS Management Console and open the Amazon Cognito console at https://console.aws.amazon.com/cognito.
2. Create a new user pool (p. 20) or edit an existing user pool (p. 119).
3. If your account is in the SMS sandbox, you will see the following message in Amazon Cognito.

   ```
   You are currently in a Sandbox environment in Amazon SNS.
   ```

   If you don't see this message, then someone already performed the necessary steps to set up SMS messages for the first time in your account. Skip to STEP 5: Complete user pool setup in Amazon Cognito (p. 149).
4. Choose the Amazon SNS link in the message to open the Amazon SNS console in a new tab.
5. Verify that you are in the sandbox environment. The console message will indicate your sandbox status and AWS Region. For example:

   ```
   This account is in the SMS sandbox in US East (N. Virginia).
   ```

In most AWS Regions, SMS messages from your user pool are routed through Amazon SNS in the same region. SMS messages in the following Amazon Cognito Regions are rerouted through the corresponding supported Amazon SNS Regions.

Amazon Cognito Regions	Supported Amazon SNS Regions
US East (Ohio)	US East (N. Virginia)
Asia Pacific (Mumbai)	Asia Pacific (Sydney)
Asia Pacific (Seoul)	Asia Pacific (Tokyo)
Canada (Central)	US East (N. Virginia)
Europe (Frankfurt)	Europe (Ireland)
Europe (London)	Europe (Ireland)

STEP 2: Verify phone numbers for Amazon Cognito in Amazon SNS

To verify SMS destination phone numbers for testing with your application, you must add destination phone numbers to Amazon SNS and then verify the numbers. For detailed instructions, see Adding and verifying phone numbers in the SMS sandbox in the *Amazon Simple Notification Service Developer Guide*.

> **Note**
> There is a limit to the number of destination phone numbers you can add to the sandbox. For details, see SMS sandbox in the *Amazon Simple Notification Service Developer Guide*.

STEP 3: Obtain an origination identity for sending SMS messages to US phone numbers

If you plan to send SMS text messages to U.S. phone numbers, you must obtain an origination identity.

Starting June 1, 2021, U.S. carriers require an origination identity to send messages to U.S. phone numbers. If you do not already have an origination identity, you must get one. To learn how to obtain an origination identity, see Requesting a number in the *Amazon Pinpoint User Guide*.

If you operate in the following AWS Regions, you must open an AWS Support ticket to obtain an origination identity. For instructions, see Requesting support for SMS messaging in the *Amazon Simple Notification Service Developer Guide*.

- Europe (Stockholm)
- Middle East (Bahrain)
- Europe (Paris)
- South America (São Paulo)
- US West (N. California)

STEP 4: Move your account out of Amazon SNS sandbox

When your account is in the SMS sandbox in Amazon SNS, Amazon Cognito can send SMS text messages to only verified phone numbers and not to your end users.

To send SMS messages to end users, you must move your account out of the sandbox and into production. For detailed instructions, see Moving Out of the Amazon SNS Sandbox in the *Amazon Simple Notification Service Developer Guide*.

STEP 5: Complete user pool setup in Amazon Cognito

Return to the browser tab where you were creating (p. 20) or editing (p. 119) your user pool. Complete the procedure.

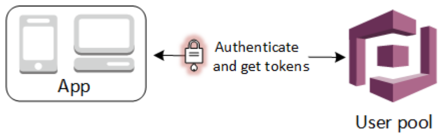
Using Tokens with User Pools

Authenticate users and grant access to resources with tokens. Tokens have claims, which are pieces of information about the user. The ID token contains claims about the identity of the authenticated user, such as name and email. The Access token contains claims about the authenticated user, a list of the user's groups, and a list of scopes.

Amazon Cognito also has tokens that you can use to get new tokens or revoke existing tokens. Refresh a token (p. 153) to retrieve a new ID and access tokens. Revoke a token (p. 154) to revoke user access that is allowed by refresh tokens.

Authenticating with tokens

When a user signs into your app, Amazon Cognito verifies the login information. If the login is successful, Amazon Cognito creates a session and returns an ID, access, and refresh token for the authenticated user. You can use the tokens to grant your users access to your own server-side resources or to the Amazon API Gateway. Or you can exchange them for temporary AWS credentials to access other AWS services.



Important
We strongly recommended that you secure all tokens in transit and storage in the context of your application.

Topics

Using the ID Token

The ID token is a JSON Web Token (JWT) that contains claims about the identity of the authenticated user such as `name`, `email`, and `phone_number`. You can use this identity information inside your application. The ID token can also be used to authenticate users to your resource servers or server applications. You can also use an ID token outside of the application with your web API operations. In those cases, you must verify the signature of the ID token before you can trust any claims inside the ID token. See Verifying a JSON Web Token (p. 156).

You can set the ID token expiration to any value between 5 minutes and 1 day. This value can be set per app client.

Important
For access and ID tokens, don't specify a minimum less than an hour. Amazon Cognito HostedUI uses cookies that are valid for an hour, if you enter a minimum less than an hour, you won't get a lower expiry time.

ID Token Header

The header contains two pieces of information: the key ID (`kid`), and the algorithm (`alg`).

```
{
"kid" : "1234example="
"alg" : "RS256",
}
```

Key ID (`kid`)

The `kid` parameter is a hint that indicates which key was used to secure the JSON Web Signature (JWS) of the token.

For more information about the kid parameter, see the Key Identifier (kid) Header Parameter.

Algorithm (`alg`)

The `alg` parameter represents the cryptographic algorithm that is used to secure the ID token. User pools use an RS256 cryptographic algorithm, which is an RSA signature with SHA-256.

For more information about the `alg` parameter, see Algorithm (alg) Header Parameter.

ID Token Payload

This is a sample payload from an ID token. It contains claims about the authenticated user. For more information about OIDC standard claims, see the OIDC Standard Claims.

```
{
"sub": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
"aud": "xxxxxxxxxxxxexample",
"email_verified": true,
"token_use": "id",
"auth_time": 1500009400,
"iss": "https://cognito-idp.us-east-1.amazonaws.com/us-east-1_example",
"cognito:username": "janedoe",
"exp": 1500013000,
"given_name": "Jane",
"iat": 1500009400,
"email": "janedoe@example.com".
"jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
"origin_jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee"

}
```

Subject (`sub`)

The `sub` claim is a unique identifier (UUID) for the authenticated user. It is not the same as the user name which might not be unique.

Issuer (`iss`)

The `iss` claim has the following format:

```
https://cognito-idp.{region}.amazonaws.com/{userPoolId}
```

For example, suppose you created a user pool in the `us-east-1` Region and its user pool ID is `u123456`. In that case, the ID token that is issued for users of your user pool have the following `iss` claim value:

```
https://cognito-idp.us-east-1.amazonaws.com/u123456
```

Audience (`aud`)

The `aud` claim contains the `client_id` that is used in the user authentication.

Token use (`token_use`)

`The token_use` claim describes the intended purpose of this token. Its value is always `id` in the case of the ID token.

Authentication time (`auth_time`)

The `auth_time` claim contains the time when the authentication occurred. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC format. On refreshes, it represents the time when the original authentication occurred, not the time when the token was issued.

Origin jti (`origin_jti`)

The originating JWT identifier, from when the original authentication occurred.

jti (`jti`)

The `jti` claim is a unique identifier of the JWT.

The ID token can contain OpenID Connect (OIDC) standard claims that are defined in OIDC Standard Claims. It can also contain custom attributes that you define in your user pool.

Note
User pool custom attributes are always prefixed with a custom: prefix.

ID Token Signature

The signature of the ID token is calculated based on the header and payload of the JWT token. When used outside of an application in your web APIs, you must always verify this signature before accepting the token. See Verifying a JSON Web Token. Verifying a JSON Web Token (p. 156).

Using the Access Token

The user pool access token contains claims about the authenticated user, a list of the user's groups, and a list of scopes. The purpose of the access token is to authorize API operations in the context of the user in the user pool. For example, you can use the access token to grant your user access to add, change, or delete user attributes.

The access token is represented as a JSON Web Token (JWT). The header for the access token has the same structure as the ID token. However, the key ID (`kid`) is different because different keys are used to sign ID tokens and access tokens. As with the ID token, you must first verify the signature of the access token in your web APIs before you can trust any of its claims. See Verifying a JSON Web Token (p. 156) You can set the access token expiration to any value between 5 minutes and 1 day. This value can be set per app client.

Important
For access and ID tokens, don't specify a minimum less than an hour. Amazon Cognito HostedUI uses cookies that are valid for an hour, if you enter a minimum less than an hour, you won't get a lower expiry time.

Access Token Header

The header contains two pieces of information: the key ID (`kid`), and the algorithm (`alg`).

```
{
"kid" : "1234example="
"alg" : "RS256",
}
```

Key ID (`kid`)

The `kid` parameter is a hint indicating which key was used to secure the JSON Web Signature (JWS) of the token.

For more information about the `kid` parameter, see the Key Identifier (kid) Header Parameter.

Algorithm (`alg`)

The `alg` parameter represents the cryptographic algorithm that was used to secure the access token. User pools use an RS256 cryptographic algorithm, which is an RSA signature with SHA-256.

For more information about the `alg` parameter, see Algorithm (alg) Header Parameter.

Access Token Payload

This is a sample payload from an access token. For more information, see JWT Claims.

```
{
  "sub": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "device_key": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "cognito:groups": [
    "admin"
  ],
  "token_use": "access",
  "scope": "aws.cognito.signin.user.admin",
  "auth_time": 1562190524,
  "iss": "https://cognito-idp.us-west-2.amazonaws.com/us-west-2_example",
  "exp": 1562194124,
  "iat": 1562190524,
  "origin_jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "client_id": "57cbishk4j24pabc1234567890",
  "username": "janedoe@example.com"
}
```

Subject (`sub`)

The `sub` claim is a unique identifier (UUID) for the authenticated user. It is not the same as the user name, which may not be unique.

Amazon Cognito groups (`cognito:groups`)

The `cognito:groups` claim is a list of groups the user belongs to.

Token use (`token_use`)

The `token_use` claim describes the intended purpose of this token. Its value is always `access` in the case of the access token.

Scope (`scope`)

The scope claim is a list of Oauth 2.0 scopes that define what access the token provides.

Authentication time (`auth_time`)

The `auth_time` claim contains the time when the authentication occurred. Its value is a JSON number that represents the number of seconds from 1970-01-01T0:0:0Z as measured in UTC format. On refreshes, it represents the time when the original authentication occurred, not the time when the token was issued.

Issuer (`iss`)

The `iss` claim has the following format:

https://cognito-idp.{region}.amazonaws.com/{userPoolId}.

Origin jti (`origin_jti`)

The originating JWT identifier, from when the original authentication occurred.

jti (`jti`)

The `jti` claim is the unique identifier of the JWT.

Access Token Signature

The signature of the access token is calculated based on the header and payload of the JWT token. When used outside of an application in your web APIs, you must always verify this signature before accepting the token. For more information, see JWT tokens.

Using the Refresh Token

You can use the refresh token to retrieve new ID and access tokens. By default, the refresh token expires 30 days after your application user signs into your user pool. When you create an application for your user pool, you can set the application's refresh token expiration to any value between 60 minutes and 10 years.

The Mobile SDK for iOS. Mobile SDK for Android, Amplify for iOS, Android, and Flutter, automatically refresh your ID and access tokens if a valid (unexpired) refresh token is present. The ID and access tokens have a minimum remaining validity of 2 minutes. If the refresh token is expired, your app user must reauthenticate by signing in again to your user pool. If the minimum for the access token and ID token is set to 5 minutes, and you are using the SDK, the refresh token will continually refresh. To see the expected behavior, set a minimum of 7 minutes instead of 5 minutes.

> **Note**
> The Mobile SDK for Android offers the option to change the minimum validity period of the ID and access tokens to a value between 0 and 30 minutes. See the `setRefreshThreshold()` method of CognitoIdentityProviderClientConfig in the AWS Mobile SDK for Android API Reference.

Your user's 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 ID and access tokens with the user pool API, use the AdminInitiateAuth or InitiateAuth methods. Pass `REFRESH_TOKEN_AUTH` for the `AuthFlow` parameter. The authorization parameter, `AuthParameters`, is a key-value map where the key is `"REFRESH_TOKEN"` and the value is the actual refresh token. Amazon Cognito responds with new ID and access tokens.

> **Note**
> To change tokens for users signed in using hostedUI, use the `InitiateAuth` API operation.

Revoking Refresh Tokens

You can revoke refresh tokens that belong to a user. For more information about revoking tokens, see Revoking Tokens (p. 154).

> **Note**
> Revoking the refresh token will revoke all tokens which are issued with the refresh token.

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` API operations. After the user is signed out, the following things occur:

- The user's refresh token cannot be used to get new tokens for the user.
- The user's access token cannot be used for the user pools service.
- The user must re-authenticate to get new tokens. The session cookies do not expire automatically. As a best practice, applications should redirect users to the logout endpoint to force the browser to clear session cookies.

An application can use the `GlobalSignOut` API to allow individual users to sign themselves out from all devices. Typically an application would present this option as a choice, such as **Sign out from all devices**. The application must call this method with the user's valid, non-expired, non-revoked access token. This method cannot be used to allow a user to sign out another user.

An application can use the `AdminUserGlobalSignOut` API to allow administrators to sign out a user from all devices. The administrator application must call this method with AWS developer credentials and pass the user pool ID and the user's user name as parameters. The `AdminUserGlobalSignOut` API can sign out any user in the user pool.

Revoking Tokens

You can revoke a refresh token for a user using the AWS API. When you revoke a refresh token, all access tokens that were previously issued by that refresh token become invalid. The other refresh tokens issued to the user are not affected.

> **Note**
> JWT tokens are self-contained with a signature and expiration time that was assigned when the token was created. Revoked tokens can't be used with any Cognito API calls that require a token.

However, Revoked tokens will still be valid if they are verified using any JWT library that verifies the signature and expiration of the token.

You can revoke a refresh token for user pool client with token revocation enabled. When you create a new user pool client, token revocation is enabled by default.

Enable token revocation

Before you can revoke a token for an existing user pool client, you must enable token revocation. You can enable token revocation for existing user pool clients using the AWS CLI or the AWS API. To do this, call the `aws cognito update-user-pool-client` CLI command or the `UpdateUserPoolClient` API operation. When you do, set the `EnableTokenRevocation` parameter to `true`.

When you create a new user pool client using the AWS Management Console, the AWS CLI, or the AWS API, token revocation is enabled by default.

After you enable token revocation, new claims are added in the Amazon Cognito JSON web tokens. The `origin_jti` and `jti` claims are added to access and ID tokens. These claims increase the size of the application client access and ID tokens.

The following JSON example shows a request to enable token revocation using the `CreateUserPoolClient` API.

```
{
    "AccessTokenValidity": 123,
    "AllowedOAuthFlows": [
        "string"
    ],
    "AllowedOAuthFlowsUserPoolClient": true,
    "AllowedOAuthScopes": [
        "string"
    ],
    "AnalyticsConfiguration": {
        "ApplicationArn": "string",
        "ApplicationId": "string",
        "ExternalId": "string",
        "RoleArn": "string",
        "UserDataShared": false
    },
    "CallbackURLs": [
        "string"
    ],
    "ClientName": "string",
    "DefaultRedirectURI": "string",
    "ExplicitAuthFlows": [
        "string"
    ],
    "GenerateSecret": true,
    "IdTokenValidity": 123,
    "LogoutURLs": [
        "string"
    ],
    "PreventUserExistenceErrors": "string",
    "ReadAttributes": [
        "string"
    ],
    "RefreshTokenValidity": 456,
    "SupportedIdentityProviders": [
        "string"
    ],
    "TokenValidityUnits": {
        "AccessToken": "string",
        "IdToken": "string",
        "RefreshToken": "string"
    },
    "UserPoolId": "string",
    "WriteAttributes": [
        "string"
    ],
    "EnableTokenRevocation": true
}
```

Revoke a token

You can revoke a refresh token using the `RevokeToken` API operation. You can also use the `aws cognito-idp revoke-token` CLI command to revoke tokens. You can also revoke tokens using the revocation endpoint. This endpoint is available after you add a domain to your user pool. You can use the revocation endpoint on either a Amazon Cognito hosted domain or your own custom domain.

> **Note**
> Refresh tokens must be revoked using the same client ID that was used to obtain the token.

Verifying a JSON Web Token

These steps describe verifying a user pool JSON Web Token (JWT).

Topics

Prerequisites

The tasks in this section might be already handled by your library, SDK, or software framework. For example, user pool token handling and management are provided on the client side through the Amazon Cognito SDKs. Likewise, the Mobile SDK for iOS and the Mobile SDK for Android automatically refresh your ID and access tokens if two conditions are met: A valid (unexpired) refresh token must present, and the ID and access tokens must have a minimum remaining validity of 5 minutes. For information on the SDKs, and sample code for JavaScript, Android, and iOS see Amazon Cognito User Pool SDKs.

Many good libraries are available for decoding and verifying a JSON Web Token (JWT). Such libraries can help if you need to manually process tokens for server-side API processing or if you are using other programming languages. See the OpenID Foundation list of libraries for working with JWT tokens.

Step 1: Confirm the Structure of the JWT

A JSON Web Token (JWT) includes three sections:

1. Header
2. Payload
3. Signature

```
11111111111.22222222222.33333333333
```

These sections are encoded as base64url strings and are separated by dot (.) characters. If your JWT does not conform to this structure, consider it invalid and do not accept it.

Step 2: Validate the JWT Signature

The JWT signature is a hashed combination of the header and the payload. Amazon Cognito generates two pairs of RSA cryptographic keys for each user pool. One of the private keys is used to sign the token.

To verify the signature of a JWT token

1. Decode the ID token.

   You can use AWS Lambda to decode user pool JWTs. For more information see Decode and verify Amazon Cognito JWT tokens using Lambda.

   The OpenID Foundation also maintains a list of libraries for working with JWT tokens.
2. Compare the local key ID (`kid`) to the public kid.

   a. Download and store the corresponding public JSON Web Key (JWK) for your user pool. It is available as part of a JSON Web Key Set (JWKS). You can locate it at https://cognito-idp.{region}.amazonaws.com/{userPoolId}/.well-known/jwks.json.

      For more information on JWK and JWK sets, see JSON Web Key (JWK).

      Note
      This is a one-time step before your web API operations can process tokens. Now you can perform the following steps each time the ID token or the access token are used with your web API operations.

      This is a sample `jwks.json` file:

```
{
 "keys": [{
  "kid": "1234example=",
  "alg": "RS256",
  "kty": "RSA",
  "e": "AQAB",
  "n": "1234567890",
  "use": "sig"
 }, {
  "kid": "5678example=",
  "alg": "RS256",
  "kty": "RSA",
  "e": "AQAB",
  "n": "987654321",
  "use": "sig"
 }]
}
```

      Key ID (`kid`)

      The `kid` is a hint that indicates which key was used to secure the JSON web signature (JWS) of the token.

      Algorithm (`alg`)

      The `alg` header parameter represents the cryptographic algorithm that is used to secure the ID token. User pools use an RS256 cryptographic algorithm, which is an RSA signature with SHA-256. For more information on RSA, see RSA Cryptography.

      Key type (`kty`)

      The `kty` parameter identifies the cryptographic algorithm family that is used with the key, such as "RSA" in this example.

RSA exponent (e)

The e parameter contains the exponent value for the RSA public key. It is represented as a Base64urlUInt-encoded value.

RSA modulus (n)

The n parameter contains the modulus value for the RSA public key. It is represented as a Base64urlUInt-encoded value.

Use (use)

The use parameter describes the intended use of the public key. For this example, the use value sig represents signature.

b. Search the public JSON web key for a kid that matches the kid of your JWT.

3. Use the public key to verify the signature using your JWT library. You might need to convert the JWK to PEM format first. This example takes the JWT and JWK and uses the Node.js library, jsonwebtoken, to verify the JWT signature:

   Node.js

```
var jwt = require('jsonwebtoken');
var jwkToPem = require('jwk-to-pem');
var pem = jwkToPem(jwk);
jwt.verify(token, pem, { algorithms: ['RS256'] }, function(err, decodedToken) {
});
```

Step 3: Verify the Claims

To verify JWT claims

1. Verify that the token is not expired.
2. The audience (aud) claim should match the app client ID that was created in the Amazon Cognito user pool.
3. The issuer (iss) claim should match your user pool. For example, a user pool created in the us-east-1 Region will have the following iss value:

   `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 API operations, its value must be access.
   - If you are only using the ID token, its value must be id.
   - If you are using both ID and access tokens, the token_use claim must be either id or access.

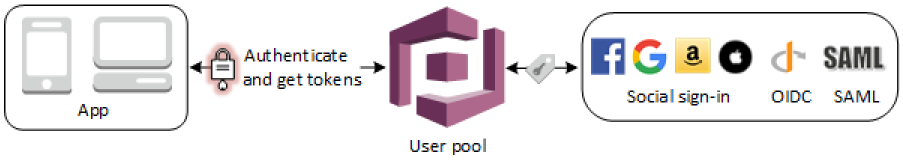
You can now trust the claims inside the token.

Accessing Resources After a Successful User Pool Authentication

Your app users can sign in either directly through a user pool, or federate through a third-party identity provider (IdP). The user pool manages the overhead of handling the tokens that are returned from social sign-in through Facebook, Google, Amazon, and Apple, and from OpenID Connect (OIDC) and SAML IdPs. For more information see Using Tokens with User Pools (p. 149).

After a successful authentication, your app will receive user pool tokens from Amazon Cognito. You can use those tokens to retrieve AWS credentials that allow your app to access other AWS services, or you might choose to use them to control access to your own server-side resources, or to the Amazon API Gateway.

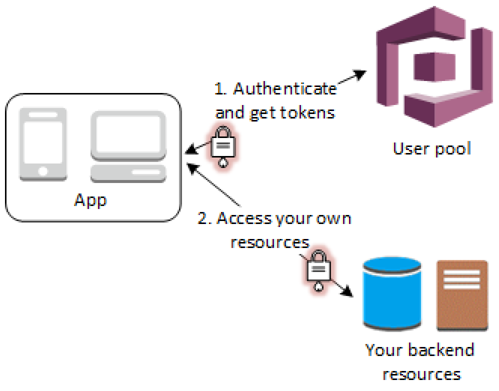
For more information, see User Pool Authentication Flow (p. 306) and Using Tokens with User Pools (p. 149).



Topics

Accessing Server-side Resources after Sign-in

After a successful authentication, your web or mobile app will receive user pool tokens from Amazon Cognito. You can use those tokens to control access to your server-side resources. You can also create user pool groups to manage permissions, and to represent different types of users. For more information on using groups to control access your resources see Adding Groups to a User Pool (p. 127).

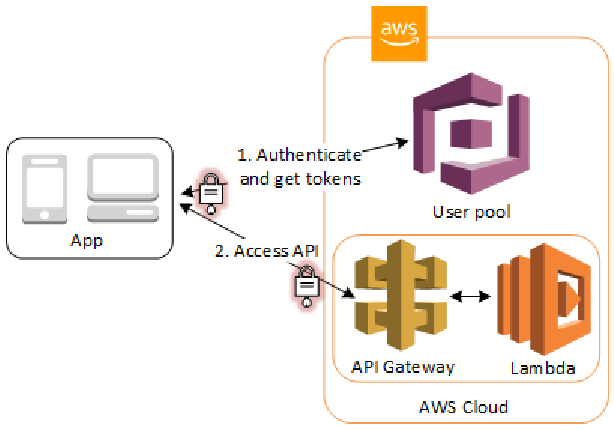


Once you configure a domain for your user pool, Amazon Cognito provisions a hosted web UI that allows you to add sign-up and sign-in pages to your app. Using this OAuth 2.0 foundation you can create your own resource server to enable your users to access protected resources. For more information see Defining Resource Servers for Your User Pool (p. 44).

For more information about user pool authentication see User Pool Authentication Flow (p. 306) and Using Tokens with User Pools (p. 149).

Accessing Resources with API Gateway and Lambda After Sign-in

You can enable your users to access your API through API Gateway. API Gateway validates the tokens from a successful user pool authentication, and uses them to grant your users access to resources including Lambda functions, or your own API.



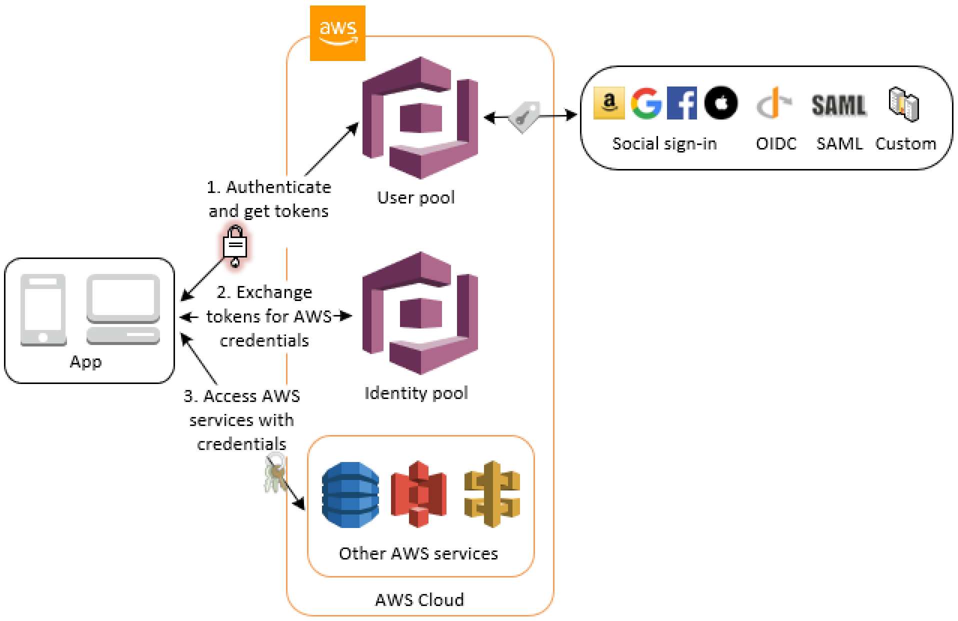
You can use groups in a user pool to control permissions with API Gateway by mapping group membership to IAM roles. The groups that a user is a member of are included in the ID token provided by a user pool when your web or mobile app user signs in. For more information on user pool groups See Adding Groups to a User Pool (p. 127).

You can submit your user pool tokens with a request to API Gateway for verification by an Amazon Cognito authorizer Lambda function. For more information on API Gateway, see Using API Gateway with Amazon Cognito user pools.

Accessing AWS Services Using an Identity Pool After Sign-in

You can enable your users to sign-in with a user pool, and then access AWS services using an identity pool.

After a successful authentication, your web or mobile app will receive user pool tokens from Amazon Cognito. You can use those tokens to retrieve AWS credentials that allow your app to access other AWS services. For more information, see Getting Started with Amazon Cognito Identity Pools (Federated Identities) (p. 187).


aws
User pool
Social sign-in
OIDC
SAML
Custom
1. Authenticate and get tokens
2. Exchange tokens for AWS credentials
3. Access AWS services with credentials
App
Identity pool
Other AWS services
AWS Cloud

For more information about using identity pools together with user pool groups to control access your AWS resources see Adding Groups to a User Pool (p. 127) and Role-Based Access Control (p. 208). See also Identity Pools Concepts (Federated Identities) (p. 193) for more information about identity pools and AWS Identity and Access Management.

Setting Up a User Pool with the AWS Management Console

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 Getting Started with User Pools (p. 20).

Setting Up an Identity Pool with 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 Identity Pools**.
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.

Integrating a User Pool with an Identity Pool

After your app 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`.

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()
    }
   });
  }
 });
}
```

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);
 }

});
```

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)
```

User Pools Reference (AWS Management Console)

You can customize the user pool settings to the needs of your app. This section 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

Adding 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 From a CSV File (p. 134)
- Adding Groups to a User Pool (p. 127)
- Creating User Accounts as Administrator (p. 124)

Configuring User Pool Attributes

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`
- `updated_at`
- `website`
- `zoneinfo`

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.

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 SMS multi-factor authentication (MFA) is enabled. For more information, see Adding Multi-Factor Authentication (MFA) to a User Pool (p. 323).

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_number_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.

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. 166).

3. Move on to create Custom Attributes (p. 168).

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. 166).

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. 166).

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.

> **Note**
> You can choose to sign in or sign up using either lowercase or uppercase letters in an alias when you configure your user pool for username case insensitivity. For more information, see CreateUserPool in the *Amazon Cognito user pools API Reference*.

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.

Note
This setting cannot be changed once the user pool is created.

Topics

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 to your user pool. You can specify a minimum and/or maximum length for custom attributes. However, the maximum length for any custom attribute can be no more than 2048 characters.

Each custom attribute:

- Can be defined as a string or a number.
- Cannot be required.

- Cannot be removed or changed once added to the user pool.
- Can have a name with a character length that is within the limit that is accepted by Amazon Cognito. For more information, see Quotas in Amazon Cognito (p. 343).

> **Note**
> In your code and in rules settings for Role-Based Access Control (p. 208), custom attributes require the `custom:` prefix to distinguish them from standard attributes.

To add a custom attribute using the console

1. From the navigation bar on the left choose **Attributes**.
2. For each new attribute:

   a. Choose **Add another attribute** under **Do you want to add custom attributes?**.

   b. Choose the properties for each custom attribute, such as the data **Type** (string or number), the **Name**, **Min length**, and **Max length**.

   c. If you want to allow the user to change the value of a custom attribute after the value has been provided by the user, select **Mutable**.

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. By default any new custom attributes will not be available until you set read and write permissions for them.

To set or change attribute permissions using the console

1. From the navigation bar on the left choose **App clients**.
2. Choose **Show Details** for the app client you want update.
3. Choose **Set the attribute read and write permissions for each attribute.**
4. Choose **Save app client changes**.

Repeat these steps for each app client using the custom attribute.

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.

Adding User Pool Password Requirements

Specifying a minimum password length of at least 8 characters, as well as requiring uppercase, numeric, and special characters, creates strong passwords for your app users. Complex passwords are harder to guess, and we recommend them as a security best practice.

These characters are allowed in passwords:

- uppercase and lowercase letters
- numbers
- Special characters listed in the next section.

Creating a Password Policy

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 a special character** from this set:

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

Configuring an 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.

Configuring Email or Phone Verification

You can choose settings for email or phone verification in the **MFA and verifications** tab. For more information on MFA, see SMS Text Message MFA (p. 325).

Amazon Cognito uses Amazon SNS to send SMS text messages. If you have never sent an SMS message from Amazon Cognito or any other AWS service, Amazon SNS might place your account in the SMS sandbox. AWS recommends that you test sending a message to a verified phone number before removing your account from the sandbox to production. Additionally, if you plan to send SMS messages to U.S. destination phone numbers, you must obtain an origination or sender ID from Amazon Pinpoint. To configure your Amazon Cognito user pool for SMS, see SMS message settings for Amazon Cognito user pools (p. 147).

Amazon Cognito can automatically verify email addresses or mobile phone numbers by sending a verification code—or, for 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.

Verification of a phone or email is necessary to automatically confirm users and enable recovery from forgotten passwords. Alternatively, you can automatically confirm users with the pre-sign up Lambda trigger or by using the AdminConfirmSignUp API. For more information, see Signing Up and Confirming User Accounts (p. 117).

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.

Notes

- Use of SMS text messaging for verifying phone numbers is charged separately by Amazon SNS. (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.
- When you test actions in your app that initiate emails from Amazon Cognito, use a real email address that Amazon Cognito can send to without incurring hard bounces. For more information, see the section called “Sending Emails While Testing Your App” (p. 123).
- The forgotten password flow requires either the user's email or the user's phone number to be verified.

Important
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.

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. For more information, see Amazon SNS FAQs.

SMS messages from Amazon Cognito user pools are routed through Amazon SNS in the same region unless noted in the following table.

Amazon Cognito Region	Supported SNS Regions
US East (Ohio) us-east-2	us-east-1
Asia Pacific (Mumbai) ap-south-1	ap-southeast-1
Asia Pacific (Seoul) ap-northeast-2	ap-notheast-1
Canada(Central) ca-central-1	us-east-1
Europe (Frankfurt) eu-central-1	eu-west-1
Europe (London) eu-west-2	eu-west-1

Example: If your Cognito user pool is in us-east-1 region, you can update the Amazon SNS limit in us-east-1 region.

Example: If your Cognito user pool is in ap-south-1 region, you can update the Amazon SNS limit in ap-southeast-1 region.

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 can create an AWS Identity and Access Management (IAM) role in the **MFA and verifications** tab of the Amazon Cognito console by choosing **Create role**.

Configuring SMS and Email Verification Messages and User Invitation Messages

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

- Your SMS text message 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.

Topics

Message Templates

Message templates allow you to insert a field into your message using a placeholder that will be replaced with the corresponding value.

Template placeholders

Description	Token
Verification code	{####}
Temporary password	{####}
User name	{username}

Note
{username} can't be used in verification emails. {username} is available for invitation emails sent after AdminCreateUser call. These invitation emails provide two place holders: username {username} and temporary password {####}

You can use advanced security template placeholders to:

- Include specific details about an event such as IP address, city, country, login time, and device name, for analysis by Amazon Cognito advanced security features.
- Verify whether a one-click link is valid.

- Build your own one-click link using event ID, feedback token, and username.

Advanced security template placeholders

Description	Token
IP address	{ip-address}
City	{city}
Country	{country}
Login time	{login-time}
Device name	{device-name}
One click link is valid	{one-click-link-valid}
One click link is invalid	{one-click-link-invalid}
Event ID	{event-id}
Feedback token	{feedback-token}

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, in the console, choose a userpool. Next, choose **Message customizations.** In **Message customizations,** choose a **SES Region** . In the **FROM email address ARN** field enter your email address. Verify the Amazon Simple Email Service identity by choosing the link after the **FROM email address ARN field.** 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, enter a valid email address in the **REPLY-TO email address** field.

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 pools 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 Adding Multi-Factor Authentication (MFA) to a User Pool (p. 323).

> **Note**
> If the device remembering feature is set to **Always** and **Do you want to use a remembered device to suppress the second factor during multi-factor authentication (MFA)?** is set to **Yes**, then the MFA settings for medium/high risks in risk-based MFA are ignored.

Configuring a User Pool App Client

An app is an entity within a user pool that has permission to call unauthenticated API operations (operations that do not have an authenticated user). Examples include operations to register, sign in, and handle forgotten passwords. To call these API operations, 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 operations.

You can create multiple apps for a user pool. Generally an app corresponds to the platform of an app. For example, you might 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 might 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 client (console)

1. On the user pool dashboard, select **Create a user pool**.
2. Enter a **Pool name**.
3. Choose **Review defaults**.
4. Choose **Add app client**.
5. Choose **Add an app client**.
6. Enter a **App client name**.
7. Specify the app's **Refresh token expiration**. The default value is 30. You can change it to any value between 1 hour and 10 years.
8. Specify the app's **Access token expiration**. The default value is 1 hour. You can change it to any value between 5 minutes and 24 hours.
9. Specify the app's **ID token expiration**. The default value is 1 hour. You can change it to any value between 5 minutes and 24 hours.

Important
If you use Hosted UI and setup tokens less than an hour, the end user will be able to get new tokens based on their session cookie which is currently fixed at one hour.

10. By default, user pools generate a client secret for your app. If you don't want that to happen, clear **Generate client secret**.
11. If your server app requires developer credentials (using Signature Version 4) and doesn't use Secure Remote Password (SRP) authentication select **Enable username password auth for admin APIs for authentication (ALLOW_ADMIN_USER_PASSWORD_AUTH)** to enable server-side authentication. For more information, see Admin authentication flow (p. 310).
12. Under **Prevent User Existence Errors**, choose **Legacy** or **Enabled**. For more information, see Managing error response.
13. 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 or choose **Create app client** to finish.

    a. Choose **Set attribute read and write permissions**.
    b. Do either of the following to set read and write permissions:

       - Choose one or more scopes. Each scope is a set of standard attributes. For more information, see the list of standard OIDC scopes.
       - Choose individual standard or custom attributes.

    Note
    You cannot remove required attributes from write permissions in any app.
14. Choose **Create app client**.
15. If you want to create another app, choose **Add an app**.
16. Once you've created all the apps you want, choose **Return to pool details**, update any other fields, and then choose **Create pool**.

To create and update app clients in a user pool (API, AWS CLI)

Do one of the following:

- **API** – Use the CreateUserPoolClient and UpdateUserPoolClient operations.
- **AWS CLI** – At the command line, run the `create-user-pool-client` and `update-user-pool-client` commands.

Configuring User Pool Lambda Triggers

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**, **Verify Auth Challenge Response**, and **User Migration**.

A user migration Lambda trigger allows easy migration of users from your existing user management system into your user pool.

For examples of each Lambda trigger, see Customizing User Pool Workflows with 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 with 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**.

Configuring User Pool Analytics

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

Using Amazon Pinpoint Analytics, you can track Amazon Cognito user pools sign-ups, sign-ins, failed authentications, daily active users (DAUs), and monthly active users (MAUs). You can also set up user attributes specific to your app using the AWS Mobile SDK for Android or AWS Mobile SDK for iOS. Those can then be used to segment your users in Amazon Pinpoint and send them targeted push notifications.

In the **Analytics** tab, you can specify an Amazon Pinpoint project for your Amazon Cognito app client. For more information, see Using Amazon Pinpoint Analytics with Amazon Cognito user pools (p. 114).

> **Note**
> Amazon Pinpoint is available in several AWS Regions in North America, Europe, Asia, and Oceania. Amazon Pinpoint regions include the Amazon Pinpoint API. If a Amazon Pinpoint region is supported by Amazon Cognito, then Amazon Cognito will send events to Amazon Pinpoint projects within the *same* Amazon Pinpoint region. If a region *isn't* supported by Amazon Pinpoint, then Amazon Cognito will *only* support sending events in us-east-1. For Amazon Pinpoint detailed region information, see Amazon Pinpoint endpoints and quotas and Using Amazon Pinpoint Analytics with Amazon Cognito User Pools.

To add analytics and campaigns

1. Choose **Add analytics and campaigns**.
2. Choose a **Cognito app client** from the list.
3. To map your Amazon Cognito app to an **Amazon Pinpoint project**, choose the Amazon Pinpoint project from the list.

   > **Note**
   > The Amazon Pinpoint project ID is a 32-character string that is unique to your Amazon Pinpoint project. It is listed the Amazon Pinpoint console.
   > You can map multiple Amazon Cognito apps to a single Amazon Pinpoint project. However, each Amazon Cognito app can only be mapped to one Amazon Pinpoint project.
   > In Amazon Pinpoint, each project should be a single app. For example, if a game developer has two games, each game should be a separate Amazon Pinpoint project, even if both games use the same Amazon Cognito user pools.

4. Choose **Share user attribute data with Amazon Pinpoint** if you want Amazon Cognito to send email addresses and phone numbers to Amazon Pinpoint in order to create additional endpoints for users.

   > **Note**
   > An *endpoint* uniquely identifies a user device to which you can send push notifications with Amazon Pinpoint. For more information about endpoints, see Adding Endpointsin the *Amazon Pinpoint Developer Guide*.

5. Enter an **IAM role** that you already created or choose **Create new role** to create a new role in the IAM console.
6. Choose **Save changes**.
7. To specify additional app mappings, choose **Add app mapping**.
8. Choose **Save changes**.

Configuring App Client Settings

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

On the **General 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 Configuring a User Pool App Client (p. 32).

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 App Client Settings Terminology (p. 34) 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. 44).

Adding 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 Configuring a User Pool Domain (p. 35).

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 Configuring App Client Settings (p. 179).

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 Customizing the Built-in Sign-in and Sign-up Webpages (p. 40).

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 **General 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**.

Adding 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. 44).

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`.

Configuring 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 Adding User Pool Sign-in Through a Third Party (p. 46).

Topics

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 pools.

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

- Google Identity Platform
- Facebook for Developers
- Login with Amazon
- Sign in with Apple>

To allow users to sign in using a social identity provider

1. Choose a social identity provider such as **Facebook**, **Google**, **Login with Amazon**, or **SignInWithApple**.
2. For the Facebook, Google or Amazon app ID and app secret, enter the app ID and app secret that you received when you created your Facebook, Google, or Login with Amazon client app. For the Sign in with Apple services ID, team ID, key ID, and private key, enter the service ID you provided to Apple, and the team ID, key ID, and private key that you received when you created your Sign in with Apple 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, Login with Amazon, and Sign in with Apple (CLI), they should be separated by spaces (Google example: `profile email openid`. Login with Amazon example: `profile postal_code`. Sign in with Apple example: `name email`). For Sign in with Apple (console), use the check boxes to select them.

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, Login with Amazon, or Sign in with Apple.

5. Choose **Enable Facebook**, **Enable Google**, **Enable Login with Amazon**, or **Enable Sign in with Apple**.

For more information on Social IdPs see Adding Social Identity Providers to a User Pool (p. 46).

Allowing Users to Sign in Using an OpenID Connect (OIDC) Identity Provider

You can enable your users to sign in through an OIDC identity provider (IdP) such as Salesforce or Ping Identity.

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. **Manage User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. On the left navigation bar, choose **Identity providers**.
5. Choose **OpenId Connect**.
6. Type a unique name into **Provider name**.
7. Type the OIDC IdP's client ID into **Client ID**.
8. Type the OIDC IdP's client secret into **Client secret**.
9. In the drop-down list, choose the HTTP method (either GET or POST) that's used to fetch the details of the user from the **userinfo** endpoint into **Attributes request method**.
10. Type the names of the scopes that you want to authorize. Scopes define which user attributes (such as `name` and `email`) that you want to access with your application. Scopes are separated by spaces, according to the OAuth 2.0 specification.

    Your app user is asked to consent to providing these attributes to your application.
11. Type the URL of your IdP and choose **Run discovery**.

    For example, Salesforce uses this URL:

    ```
    https://login.salesforce.com
    ```

    Note
    The URL should start with `https://`, and shouldn't end with a slash `/`.

    - If **Run discovery** isn't successful, then you need to provide the **Authorization endpoint**, **Token endpoint**, **Userinfo endpoint**, and **Jwks uri** (the location of the JSON Web Key).
12. Choose **Create provider**.
13. On the left navigation bar, choose **App client settings**.
14. Select your OIDC provider as one of the **Enabled Identity Providers**.
15. Type a callback URL for the Amazon Cognito authorization server to call after users are authenticated. This is the URL of the page where your user will be redirected after a successful sign-in.

```
https://www.example.com
```

16. Under **Allowed OAuth Flows**, enable both the **Authorization code grant** and the **Implicit code grant**.

    Unless you specifically want to exclude one, select the check boxes for all of the **Allowed OAuth scopes**.
17. Choose **Save changes**.
18. On the **Attribute mapping** tab on the left navigation bar, add mappings of OIDC claims to user pool attributes.

    a. As a default, the OIDC claim **sub** is mapped to the user pool attribute **Username**. You can map other OIDC claims to user pool attributes. Type in the OIDC claim, and choose the corresponding user pool attribute from the drop-down list. For example, the claim **email** is often mapped to the user pool attribute **Email**.
    b. In the drop-down list, choose the destination user pool attribute.
    c. Choose **Save changes**.
    d. Choose **Go to summary**.

For more information on OIDC IdPs see Adding OIDC Identity Providers to a User Pool (p. 59).

Allowing Users to Sign in Using 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. 58).

To allow users to sign in using SAML

1. Choose **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 Adding SAML Identity Providers to a User Pool (p. 52).
4. Select **Enable IdP sign out flow** when you want your user to be logged out from a SAML IdP when logging out from Amazon Cognito.

   Enabling this flow sends a signed logout request to the SAML IdP when the LOGOUT Endpoint (p. 364) is called.

   Note
   If this option is selected and your SAML identity provider expects a signed logout request, you will also need to configure the signing certificate provided by Amazon Cognito with your SAML IdP.
   The SAML IdP will process the signed logout request and logout your user from the Amazon Cognito session.
5. Choose **Create provider**.
6. 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.

To set up the SAML IdP to add a signing certificate

- To get the certificate containing the public key which will be used by the identity provider to verify the signed logout request, choose **Show signing certificate** under **Active SAML Providers** on the **SAML** dialog under **Identity providers** on the **Federation** console page.

For more information on SAML IdPs see Adding SAML Identity Providers to a User Pool (p. 52).

Configuring 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. 64).

> **Note**
> Currently, only the Facebook `id`, Google `sub`, Login with Amazon `user_id`, and Sign in with Apple `sub` 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**, **Amazon**, or **Apple** tab.
2. For each attribute you need to map, perform the following steps:

   a. Choose the **Capture** check box.
   b. In the **User pool attribute** field, choose the user pool attribute to map the social identity provider attribute to from the drop-down list.
   c. For Facebook, Google, and Login with Amazon, if you need more attributes, choose either **Add Facebook attribute**, **Add Google attribute** or **Add Amazon attribute**, and then perform the following steps:

      > **Note**
      > Sign in with Apple does not provide additional attributes at this time.

      i. In the **Facebook attribute**, **Google attribute** or **Amazon attribute** field, enter the name of the attribute to be mapped.
      ii. 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.
   d. 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:

   a. Choose **Add SAML attribute**.

b. In the **SAML attribute** field, enter the name of the SAML attribute to be mapped.

c. For **User pool attribute**, choose the user pool attribute you want to map to the SAML attribute from the drop-down list.

d. Choose **Save changes**.

Managing error responses

Amazon Cognito supports customizing error responses returned by User Pools. Custom error responses are available for authentication, confirmation, and password recovery-related operations. Use the `PreventUserExistenceErrors` setting of a user pool app client to enable or disable user existence related errors.

When you enable custom error responses, Amazon Cognito authentication APIs return a generic authentication failure response. The error response tells you the user name or password is incorrect. Amazon Cognito account confirmation and password recovery APIs return a response indicating a code was sent to a simulated delivery medium. The error response works when the status is `ENABLED` and the user doesn't exist. Below are the detailed behaviors for the Amazon Cognito operations when `PreventUserExistenceErrors` is set to `ENABLED`.

User authentication operations

You can use either authentication flow method with the following operations.

- `AdminInitiateAuth`
- `AdminRespondToAuthChallenge`
- `InitiateAuth`
- `RespondToAuthChallenge`

User name password based authentication

In the authentication flows for `ADMIN_USER_PASSWORD_AUTH` and `USER_PASSWORD_AUTH` the user name and password returns with a single call of `InitiateAuth`. Amazon Cognito returns a generic `NotAuthorizedException` error indicating either the user name or password is incorrect.

Secure Remote Password (SRP) based authentication

In the USER_SRP_AUTH authentication flow Amazon Cognito receives a user name and SRP parameter 'A' in the first step. In response, Amazon Cognito returns SRP parameter 'B' and 'salt' for the user as per SRP protocol. When a user isn't found, Amazon Cognito returns a simulated response in the first step as described in RFC 5054. Amazon Cognito returns the same 'salt' and internal user id in Universally Unique IDentifier (UUID) format for the same user name and user pool combination. When the next operation of `RespondToAuthChallenge` proof of password runs, Amazon Cognito returns a generic `NotAuthorizedException` error indicating either user name or password was incorrect.

Note
You can use `UsernamePassword` to simulate a generic response if you are using verification based aliases and the format of immutable user name isn't a UUID.

ForgotPassword

When a user isn't found, is disabled, or doesn't have a mechanism to recover their password, Amazon Cognito returns `CodeDeliveryDetails` with a simulated delivery medium for a user. The simulated delivery medium is determined by the input user name format and verification settings of the user pool.

ConfirmForgotPassword

Amazon Cognito returns the `CodeMismatchException` error for users that don't exist or are disabled. If a code isn't requested when using `ForgotPassword`, Amazon Cognito returns the `ExpiredCodeException` error.

ResendConfirmationCode

Amazon Cognito returns `CodeDeliveryDetails` for a disabled user or a user that doesn't exist. Amazon Cognito sends a confirmation code to the existing user's email or phone number.

ConfirmSignUp

`ExpiredCodeException` returns if a code has expired. Amazon Cognito returns `NotAuthorizedException` when a user isn't authorized. If the code doesn't match what the server expects Amazon Cognito returns `CodeMismatchException`.

SignUp

The `SignUp` operation returns `UsernameExistsException` when a user name is already taken. To prevent the `UsernameExistsException` error for email or phone number during `SignUp`, you can use verification based aliases. For more information, see AliasAttributes Amazon Cognito API Reference guide. For more information about aliases see Overview of Aliases.

Imported users

If `PreventUserExistenceErrors` is enabled, during authentication of imported users a generic `NotAuthorizedException` error is returned indicating either the user name or password was incorrect instead of returning `PasswordResetRequiredException`. See Requiring imported users to reset their passwords for more information.

Migrate user Lambda trigger

Amazon Cognito returns a simulated response for users that don't exist when an empty response was set in the original event context by the Lambda trigger. For more information, see Migrate User Lambda Trigger.

Custom Authentication Challenge Lambda trigger

If you use Custom Authentication Challenge Lambda Trigger and you enable error responses, then `LambdaChallenge` returns a boolean parameter named `UserNotFound`. Then it's passed in the request of `DefineAuthChallenge`, `VerifyAuthChallenge`, and `CreateAuthChallenge` Lambda triggers. You can use this trigger to simulate custom authorization challenges for a user that doesn't exist. If you call the Pre-Authentication Lambda trigger for a user that doesn't exist, then Amazon Cognito returns `UserNotFound`.

Amazon Cognito Identity Pools (Federated Identities)

Amazon Cognito identity pools (federated identities) enable you to create unique identities for your users and federate them with identity providers. With an identity pool, you can obtain temporary, limited-privilege AWS credentials to access other AWS services. Amazon Cognito identity pools support the following identity providers:

- Public providers: Login with Amazon (Identity Pools) (p. 224), Facebook (Identity Pools) (p. 219), Google (Identity Pools) (p. 227), Sign in with Apple (Identity Pools) (p. 234).
- Amazon Cognito user pools (p. 19)
- Open ID Connect Providers (Identity Pools) (p. 238)
- SAML Identity Providers (Identity Pools) (p. 240)
- Developer Authenticated Identities (Identity Pools) (p. 241)

For information about Amazon Cognito identity pools region availability, see AWS Service Region Availability.

For more information about Amazon Cognito identity pools, see the following topics.

Topics

Getting Started with Amazon Cognito Identity Pools (Federated Identities)

Amazon Cognito identity pools 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 external identity providers such as Facebook, Google, Apple, 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

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://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

Create an Identity Pool in Amazon Cognito

You can 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 Identity Pools**, and then choose **Create new identity pool**.
2. Type a name for your identity pool.
3. To enable unauthenticated identities select **Enable access to unauthenticated identities** from the **Unauthenticated identities** collapsible section.
4. If desired, configure an authentication provider in the **Authentication providers** section.
5. Choose **Create Pool**.

   Note
   At least one identity is required for a valid identity pool.
6. You will be prompted for access to your AWS resources.

   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.

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 identity pools (federated identities) support user authentication through Amazon Cognito user pools, federated identity providers—including Amazon, Facebook, Google, Apple, and SAML identity providers—as well as unauthenticated identities. This feature also supports Developer Authenticated Identities (Identity Pools) (p. 241), 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. 19) and Accessing AWS Services Using an Identity Pool After Sign-in (p. 160).

To learn more about using external identity providers, see Identity Pools (Federated Identities) External Identity Providers (p. 219).

To learn more about integrating your own back-end authentication process, see Developer Authenticated Identities (Identity Pools) (p. 241).

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. 211).

Using Identity Pools (Federated Identities)

Amazon Cognito identity pools provide temporary AWS credentials for users who are guests (unauthenticated) and for users who have been authenticated and received a token. An identity pool is a store of user identity data specific to your account.

To create a new identity pool in the console

1. Sign in to the Amazon Cognito console, choose **Manage Identity Pools**, and then choose **Create new identity pool**.
2. Type a name for your identity pool.
3. To enable unauthenticated identities select **Enable access to unauthenticated identities** from the **Unauthenticated identities** collapsible section.
4. If desired, configure an authentication provider in the **Authentication providers** section.
5. Choose **Create Pool**.

   Note
   At least one identity is required for a valid identity pool.
6. You will be prompted for access to your AWS resources.

   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. 3).

User IAM Roles

An IAM role defines the permissions for your users to access AWS resources, like Amazon Cognito Sync (p. 255). 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. 198).

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 Identity Pools (Federated Identities) External Identity Providers (p. 219).
- To configure your own backend authentication process, see Developer Authenticated Identities (Identity Pools) (p. 241).

Enable or disable unauthenticated identities

Amazon Cognito Identity Pools 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 Getting Started with Amazon Cognito Identity Pools (Federated Identities) (p. 187).

Choose **Manage Identity Pools** from the Amazon Cognito console:

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**.

Change the role associated with an identity type

Identity pools define two types of identities: authenticated and unauthenticated. Every identity in your identity pool is either authenticated or unauthenticated. Authenticated identities belong to users who are authenticated by a public login provider (Amazon Cognito user pools, Login with Amazon, Sign in with Apple, Facebook, Google, 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 change which role is associated with an identity type using the Amazon Cognito identity pool (federated identities) console. Choose **Manage Identity Pools** from the Amazon Cognito console:

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 edit authentication providers

If you allow your users to authenticate using public identity providers (e.g. Amazon Cognito user pools, Login with Amazon, Sign in with Apple, Facebook, Google), you can specify your application identifiers in the Amazon Cognito identity pools (federated identities) 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. 208).

Warning
Changing the application ID to which an identity pool is linked will disable existing users from authenticating with that identity pool. Learn more about Identity Pools (Federated Identities) External Identity Providers (p. 219).

Choose **Manage Identity Pools** from the Amazon Cognito console:

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.

Delete an Identity Pool

Choose **Manage Identity Pools** from the Amazon Cognito console:

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

Choose **Manage Identity Pools** from the Amazon Cognito console:

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**.

Managing Datasets

If you have implemented Amazon Cognito Sync functionality in your application, the Amazon Cognito Identity Pools 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 Identity Pools 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

Choose **Manage Identity Pools** from the Amazon Cognito Identity Pools Amazon Cognito console :

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

Choose **Manage Identity Pools** from the Amazon Cognito console:

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**.

Bulk Publish Data

Bulk publish can be used to export data already stored in your Amazon Cognito Sync store to a Kinesis stream. For instructions on how to bulk publish all of your streams, see Amazon Cognito Streams (p. 281).

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. Choose **Manage Identity Pools** from the Amazon Cognito console:

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 Streams

Amazon Cognito Streams gives developers control and insight into their data stored in Amazon Cognito Sync. Developers can now configure a Kinesis stream to receive events as data. Amazon Cognito can push each dataset change to a 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. 281).

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. 282).

To learn more about AWS Lambda, see AWS Lambda.

Identity Pools Concepts (Federated Identities)

Amazon Cognito identity pools enable 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 access other AWS services. Amazon Cognito identity pools support public identity providers—Amazon, Apple, 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 pools Region availability, see AWS Service Region Availability. For more information about Amazon Cognito identity pools concepts, see the following topics.

Topics

Identity Pools (Federated Identities) 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.

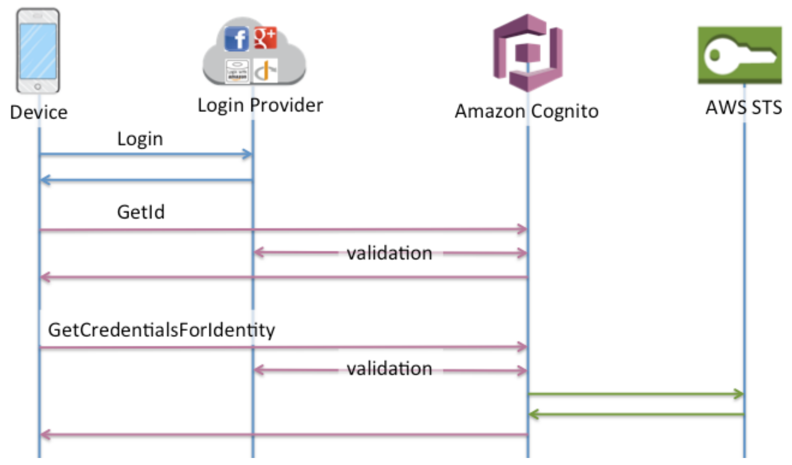
External Provider Authflow

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. 198).

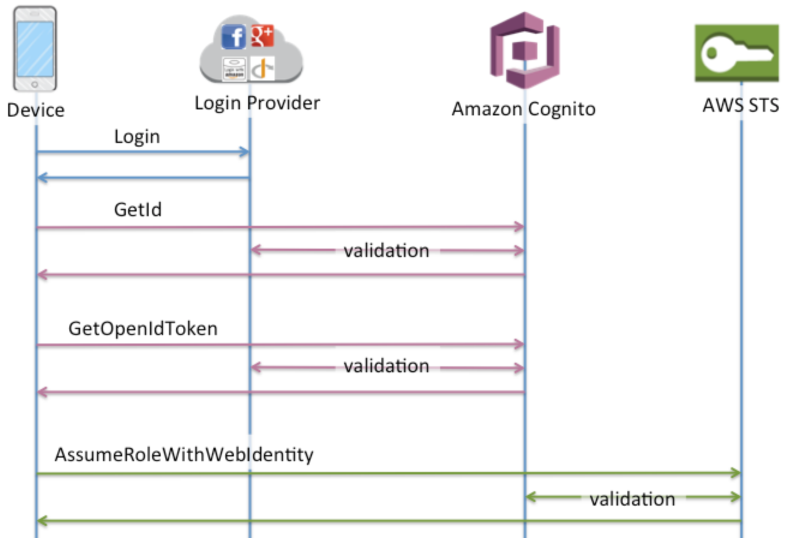
Enhanced (Simplified) Authflow

1. `GetId`
2. `GetCredentialsForIdentity`



Basic (Classic) Authflow

1. `GetId`
2. `GetOpenIdToken`
3. `AssumeRoleWithWebIdentity`

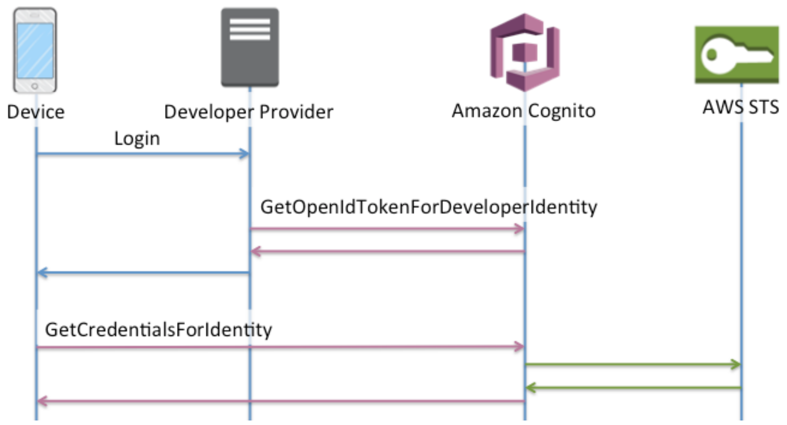


Developer Authenticated Identities Authflow

When using Developer Authenticated Identities (Identity Pools) (p. 241), 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.

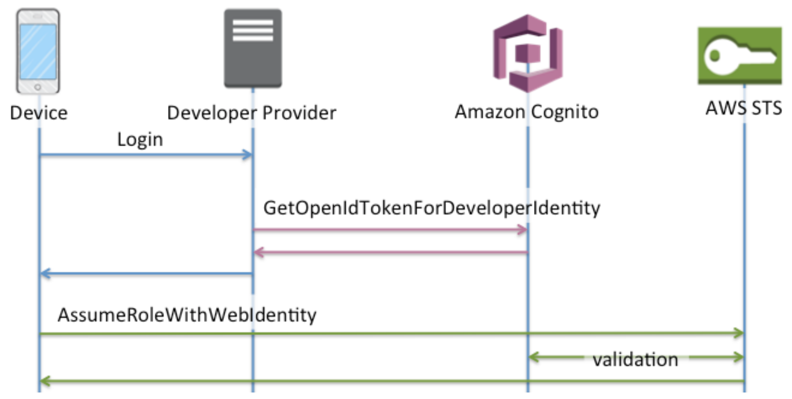
Enhanced Authflow

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



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. AssumeRoleWithWebIdentity



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, or Sign in with Apple), 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+, Login with Amazon, or Sign in with Apple 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. 203).

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. 208).

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.

> **Note**
> As a best practice, define policies that follow the principle of granting *least privilege*. In other words, the policies include only the permissions that users require to perform their tasks. For more information, see Grant Least Privilege in the *IAM User Guide*.
> Remember that unauthenticated identities are assumed by users who do not log in to your app. Typically, the permissions that you assign for unauthenticated identities should be more restrictive than those for authenticated identities.

Topics

Set Up a Trust Policy

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. 203).

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 the ones listed below for your unauthenticated users. In other words, this policy allows an identity using these credentials with access to only the following services:

AWS AppSync	Amazon Comprehend	GameLift	Amazon Lex	Amazon Polly	Amazon SimpleDB	Amazon Transcribe
CloudWatch	DynamoDB	AWS IoT	Amazon Machine Learning	Amazon Rekognition	Amazon SES	Amazon Translate
Amazon Cognito Identity	Amazon Kinesis Data Firehose	AWS KMS	Amazon Mobile Analytics	Amazon Sumerian	Amazon SNS	
Amazon Cognito Sync	Amazon Kinesis Data Streams	AWS Lambda	Amazon Pinpoint	Amazon S3	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.

Access Policy Examples

In this section, you can find example Amazon Cognito access policies that grant only the permissions your identities need to complete a specific operation. You can further limit the permissions for a given identity ID by using policy variables where possible. For example, using ${cognito-identity.amazonaws.com:sub}. For more information, see Understanding Amazon Cognito Authentication Part 3: Roles and Policies on the *AWS Mobile Blog*.

Note
As a security best practice, policies should include only the permissions that users require to perform their tasks. This means that you should try to always scope access to an individual identity for objects whenever possible.

Example 1: Allow an Identity to Have Read Access to a Single Object in S3

The following access policy grants read permissions to an identity to retrieve a single object from a given S3 bucket.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket/assets/my_picture.jpg"]
    }
  ]
}
```

Example 2: Allow an Identity to Have Both Read and Write Access to Identity Specific Paths in S3

The following access policy grants read and write permissions to access a specific prefix "folder" in an S3 bucket by mapping the prefix to the `${cognito-identity.amazonaws.com:sub}` variable.

With this policy, an identity such as `us-east-1:12345678-1234-1234-1234-123456790ab` inserted via `${cognito-identity.amazonaws.com:sub}` will be able to get, put and list objects into `arn:aws:s3:::mybucket/us-east-1:12345678-1234-1234-1234-123456790ab`. However, the identity would not be granted access to other objects in `arn:aws:s3:::mybucket`.

```
{
  "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}/*"]
    }
  ]
}
```

Example 3: Assign Identities Fine-Grained Access to Amazon DynamoDB

The following access policy provides fine-grained access control to Amazon DynamoDB resources using Amazon Cognito variables that grant access to items in DynamoDB by identity ID. For more information, see Using IAM Policy Conditions for Fine-Grained Access Control in the *Amazon DynamoDB Developer Guide*.

```
{
  "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}"]
        }
      }
    }
  ]
}
```

Example 4: Allow an Identity to Execute an AWS Lambda Function Invocation

The following access policy grants an identity permissions to execute an AWS Lambda function.

```
{
    "Version": "2012-10-17",
    "Statement": [
         {
             "Effect": "Allow",
             "Action": "lambda:InvokeFunction",
             "Resource": [
                 "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
             ]
         }
     ]
 }
```

Example 5: Allow an Identity to Publish Records to an Amazon Kinesis Data Stream

The following access policy allows an identity to use the `PutRecord` operation with any of the Kinesis Data Streams. It can be applied to users that need to add data records to all streams in an account. For more information, see Controlling Access to Amazon Kinesis Data Streams Resources Using IAM in the *Amazon Kinesis Data Streams Developer Guide*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kinesis:PutRecord",
            "Resource": [
                "arn:aws:kinesis:us-east-1:111122223333:stream/stream1"
            ]
        }
    ]
}
```

Example 6: Allow an Identity Access to Their Data in the Amazon Cognito Sync store

The following access policy grants an identity permissions to only their data in the Amazon Cognito Sync store.

```
{
  "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}/*"]
      }]
  }
```

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-corner-
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.

Using attributes for access control as a form of attribute-based access control

You can use IAM policies to control access to AWS resources through Amazon Cognito identity pools based on user attributes. These attributes can be drawn from social and corporate identity providers. You can map attributes within providers' access and ID tokens or SAML assertions to tags that can be referenced in the IAM permissions policies.

You can choose default mappings or create your own custom mappings in Amazon Cognito identity pools. The default mappings allow you to write IAM policies based on a fixed set of user attributes. Custom mappings allow you to select a custom set of user attributes that are referenced in the IAM permissions policies. The **Attribute names** in the Amazon Cognito console are mapped to **Tag key for principal**, which are the tags that are referenced in the IAM permissions policy.

For example, let's say that you own a media streaming service with a free and paid membership. You store the media files in Amazon S3 and tag them with free or premium tags. You can use attributes for access control to allow access to free and paid content based on user membership level, which is part of user's profile. You can map the membership attribute to a tag key for principal to be passed on to the IAM permissions policy. This way you can create a single permissions policy and conditionally allow access to premium content based on the value of membership level and tag on the content files.

Topics

Using attributes to control access has several benefits:

- Permissions management is easier when you use attributes for access control. You can create a basic permissions policy that uses user attributes instead of creating multiple policies for different job functions.
- You don't need to update your policies whenever you add or remove resources or users for your application. The permissions policy will only grant the access to users with the matching user attributes. For example, you might need to control the access to certain S3 buckets based on the job title of users. In that case, you can create a permissions policy to allow access to these files only for users within the defined job title. For more information, see IAM Tutorial: Use SAML session tags for ABAC.
- Attributes can be passed as principal tags to a policy that allows or denies permissions based on the values of those attributes.

Using attributes for access control with Amazon Cognito identity pools

Before you can use attributes for access control, ensure that you meet the following prerequisites:

- An AWS account
- User pool
- Identity pool
- The mobile or JavaScript SDK

- Integrated identity providers
- Credentials

To use attributes for access control you have to configure **Tag Key for Principal** and **Attribute name**. In **Tag Key for Principal** the value is used to match the **PrincipalTag** condition in the permissions policies. The value in **Attribute name** is the name of the attribute whose value will be evaluated in the policy.

To use attributes for access control with identity pools

1. Open the Amazon Cognito console.
2. Choose **Manage Identity Pools**.
3. On the dashboard, choose the name of the identity pool that you want to use attributes for access control on.
4. Choose **Edit identity pool**.
5. Expand the **Authentication providers** section.
6. In the **Authentication providers** section, choose the provider tab you want to use.
7. In **Attributes for access control**, choose either **Default attribute mappings** or **Custom attribute mappings**. Default mappings are different for each provider. For more information, see Default provider mappings (p. 207) for attributes for access control.
8. If you choose **Custom attribute mappings**, complete the following steps.

   1. In **Tag Key for Principal**, enter your custom text. There is a maximum length of 128 characters.
   2. In **Attribute name**, enter the attribute names from your providers tokens or SAML assertions. You can get your attribute names for your IdPs from your providers developer guide. Attribute names have a maximum of 256 characters. In addition, the aggregated character limit for all attributes is 460 bytes.
   3. (Optional) **Add another provider**. You can add multiple providers for Amazon Cognito user pools, OIDC, and SAML providers in the console. For example, you can add two Amazon Cognito user pools as two separate identity providers. Amazon Cognito treats each tab as different IdPs. You can configure attributes for access control separately for each IdP.
   4. To finish, use the IAM console to create a permissions policy that includes the default mappings or the custom text mappings that you provided in **Tag Key for Principal**. For a tutorial on creating a permissions policy in IAM, see IAM tutorial: Define permissions to access AWS resources based on tags in the *IAM User Guide*.

Using attributes for access control policy example

Consider a scenario where an employee from the legal department of a company needs to list all files in buckets that belong to their department and are classified with their security level. Assume the token that this employee gets from the identity provider contains the following claims.

Claims

```
{ .
  .
"sub" : "57e7b692-4f66-480d-98b8-45a6729b4c88",
"department" : "legal",
"clearance" : "confidential",
 .
 .
}
```

These attributes can be mapped to tags and referenced in IAM permissions policies as principal tags. You can now manage access by changing the user profile on the identity provider's end. Alternatively, you can change attributes on the resource side by using names or tags without changing the policy itself.

The following permissions policy does two things:

- Allows list access to all s3 buckets that end with a prefix that matches the user’s department name.
- Allows read access on files in these buckets as long as the clearance tag on the file matches user’s clearance attribute.

Permissions policy

```
   {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:List*",
            "Resource": "arn:aws:s3:::*-${aws:PrincipalTag/department}"
        },
        {
            "Effect": "Allow",
            "Action": "s3:GetObject*",
            "Resource": "arn:aws:s3:::*-${aws:PrincipalTag/department}/*",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/clearance": "${aws:PrincipalTag/clearance}"
                }
            }
        }
    ]
}
```

The trust policy determines who can assume this role. The trust relationship policy allows the use of `sts:AssumeRoleWithWebIdentity` and `sts:TagSession` to allow access. It adds conditions to restrict the policy to the identity pool that you created and it ensures that it’s for an authenticated role.

Trust policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "cognito-identity.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRoleWithWebIdentity",
        "sts:TagSession"
      ],
      "Condition": {
        "StringEquals": {
          "cognito-identity.amazonaws.com:aud": "IDENTITY-POOL-ID"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "authenticated"
        }
```

```
            }
        }
    ]
}
```

Disable attributes for access control (console)

Follow this procedure to disable attributes for access control.

To disable attributes for access control

1. Open the Amazon Cognito console.
2. Choose **Manage Identity Pools**.
3. On the dashboard, choose the name of the identity pool whose attributes for access control you want to disable.
4. Choose **Edit identity pool** in the upper-right corner of the page.
5. Expand the **Authentication providers** section.
6. Under **Authenticated role selection**, choose **Disable**.
7. To finish, scroll to the bottom of the page and choose **Save Changes**.

Default provider mappings

The following table has the default mapping information for the authentication providers that Amazon Cognito supports.

Provider	Token type	Principal tag values	Example
Amazon Cognito user pool	ID token	aud(client ID) and sub(user ID)	"6jk8ltokc7ac9es6jrtg9q572f" , "57e7b692-4f66-480d-98b8-45a6729
Facebook	Access token	aud(app_id), sub(user_id)	"492844718097981", "112177216992379"
Google	ID token	aud(client ID) and sub(user ID)	"620493171733-eebk7c0hcp5lj3e1tlqp1gntt3k0rncv.ap "109220063452404746097"
SAML	Assertions	"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier" , "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"	"auth0|5e28d196f8f55a0eaaa95de3" , "user123@gmail.com"
Apple	ID token	aud(client ID) and sub (user ID)	"com.amazonaws.ec2-54-80-172-243. "001968.a6ca34e9c1e742458a26cf80
Amazon	Access token	aud (Client ID on Amzn Dev Ac), user_id(user ID)	"amzn1.application-oa2-client.9d70d9382d3446108aaee3dd7 "amzn1.account.AGHNIFJQMFSBG3G6

Provider	Token type	Principal tag values	Example
Standard OIDC providers	ID and access tokens	aud (as client_id), sub (as user ID)	"620493171733-eebk7c0hcp5lj3e1tlqp1gntt3k0rncv.ap "109220063452404746097"
Twitter	Access token	aud (app ID; app Secret), sub (user ID)	"DfwifTtKEX1FiIBRnOTlR0CFK;Xgj5xb8 "1269003884292222976"
DevAuth	Map	Not applicable	"tag1", "tag2"

Note
The default attribute mappings option is automatically populated for the **Tag Key for Principal** and **Attribute** names. You can't change default mappings.

Role-Based Access Control

Amazon Cognito identity pools assign 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 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-corner-
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 through Amazon Cognito 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 identity pools (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 Configuring User Pool Attributes (p. 164).

You can set multiple rules for an authentication provider in the identity pool (federated identities) 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 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

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.

Apple

An Apple token contains standard claims from the OpenID Connect specification. See Authenticating Users with Sign in with Apple in Apple's documentation to learn more about the claim available from the Apple token. Apple's token doesn't always contain `email`.

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

You can use Amazon Cognito to deliver temporary, limited-privilege credentials to your application, so that your users can access AWS resources. This section describes how to get credentials and how to retrieve an Amazon Cognito identity from an identity pool.

Amazon Cognito supports both authenticated and unauthenticated identities. Unauthenticated users do not have their identity verified, making this role appropriate for guest users of your app or in cases when it doesn't matter if users have their identities verified. Authenticated users log in to your application through a third-party identity provider, or a user pool, that verifies their identities. Make sure you scope the permissions of resources appropriately so you don't grant access to them from unauthenticated users.

Amazon Cognito identities are not credentials. They are exchanged for credentials using web identity federation support in the AWS Security Token Service (AWS STS). The recommended way to obtain AWS credentials for your app users is to use `AWS.CognitoIdentityCredentials`. The identity in the credentials object is then exchanged for credentials using AWS STS.

> **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 the `AWS.CognitoIdentityCredentials` constructor without the roles as parameters. To do so, open the Amazon Cognito console, choose **Manage Identity Pools**, select your identity pool, choose **Edit Identity Pool**, specify your authenticated and unauthenticated roles, and save the changes.

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. Choose **Manage Identity Pools** from 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, choose **Manage Federated Identies**, select your identity pool, and 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 identity pools support both authenticated and unauthenticated identities. To provide AWS credentials to your app, follow the steps below.

1. Choose **Manage Identity Pools** from 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, choose **Manage Identity Pools**, 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. Choose **Manage Identity Pools** from 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, choose **Manage Identity Pools**, 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

If you have not yet created one, create an identity pool in the Amazon Cognito console before using `AWS.CognitoIdentityCredentials`.

After you configure an identity pool with your identity providers, you can use `AWS.CognitoIdentityCredentials` to authenticate users. To configure your application credentials to use `AWS.CognitoIdentityCredentials`, set the `credentials` property of either `AWS.Config` or a per-service configuration. The following example uses `AWS.Config`:

```
// 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',
    Logins: { // optional tokens, used for authenticated login
        'graph.facebook.com': 'FBTOKEN',
        'www.amazon.com': 'AMAZONTOKEN',
        'accounts.google.com': 'GOOGLETOKEN',
        'appleid.apple.com': 'APPLETOKEN'
    }
});

// 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;

});
```

The optional `Logins` property is a map of identity provider names to the identity tokens for those providers. How you get the token from your identity provider depends on the provider you use. For example, if Facebook is one of your identity providers, you might use the `FB.login` function from the Facebook SDK to get an identity provider token:

```
FB.login(function (response) {
    if (response.authResponse) { // logged in
        AWS.config.credentials = new AWS.CognitoIdentityCredentials({
          IdentityPoolId: 'us-east-1:1699ebc0-7900-4099-b910-2df94f52a030',
          Logins: {
            'graph.facebook.com': response.authResponse.accessToken
          }
        });

        console.log('You are now logged in.');
    } else {
        console.log('There was a problem logging you in.');
    }
});
```

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. Choose **Manage Identity Pools**, from 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, choose **Manage Identity Pools**, 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. Choose **Manage Identity Pools**, from 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, choose **Manage Identity Pools**, 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.

Identity Pools (Federated Identities) External Identity Providers

Using the `logins` property, you can set credentials received from an identity provider. Moreover, you can associate an identity pool 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 identity pools.

Topics

Facebook (Identity Pools)

Amazon Cognito identity pools integrate 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 must 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 identity pools use the Facebook access token to generate a unique user identifier that is associated with an Amazon Cognito identity.

- Facebook iOS SDK by Xamarin
- Facebook Android SDK by Xamarin

Configure the External Provider in the Amazon Cognito Federated Identities Console

Use the following procedure to configure your external provider.

1. Choose **Manage Identity Pools** from the Amazon Cognito console home page.
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 must 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 pools (federated identities).

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`. This dictionary acts 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 pools (federated identities).

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`. This dictionary acts 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 Amazon 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 must make a call to `refresh` or `get` to get the 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 with an Amazon 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 (Identity Pools)

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 (p. 211).
- 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

Choose **Manage Identity Pools** from the Amazon Cognito console home page:

1. Choose 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, choose **Edit identity pool**. The Edit identity pool page appears.
3. Scroll down and choose **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 (Identity Pools)

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 (p. 238). 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 must 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 more information on integrating Google into your Android app, see the Google documentation for Android.

Configure the External Provider in the Amazon Cognito Console

Choose **Manage Identity Pools** from the Amazon Cognito Console home page:

1. Choose the name of the identity pool for which you want to enable Google as an external provider. The **Dashboard** page for your identity pool appears.
2. In the top-right corner of the **Dashboard** page, choose **Edit identity pool**. The Edit identity pool page appears.
3. Scroll down and choose **Authentication providers** to expand it.
4. Choose the **Google** tab.
5. Choose **Unlock**.
6. Enter the Google Client ID you obtained from Google, and then choose **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 (p. 238). Add 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 must 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 more information on integrating Google into your iOS app, see the Google documentation for iOS.

Choose **Manage Identity Pools** from the Amazon Cognito Console home page:

Configure the External Provider in the Amazon Cognito Console

1. Choose the name of the identity pool for which you want to enable Google as an external provider. The **Dashboard** page for your identity pool appears.
2. In the top-right corner of the **Dashboard** page, choose **Edit identity pool**. The Edit identity pool page appears.
3. Scroll down and choose **Authentication providers** to expand it.
4. Choose the **Google** tab.
5. Choose **Unlock**.
6. Enter the Google Client ID you obtained from Google, and then choose **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 (p. 238). Add 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 more information on integrating Google into your iOS app, see the Google documentation for iOS.

Choose **Manage Identity Pools** from the Amazon Cognito Console home page:

Configure the External Provider in the Amazon Cognito Console

1. Choose the name of the identity pool for which you want to enable Google as an external provider. The **Dashboard** page for your identity pool appears.
2. In the top-right corner of the **Dashboard** page, choose **Edit identity pool**. The Edit identity pool page appears.
3. Scroll down and choose **Authentication providers** to expand it.
4. Choose the **Google** tab.
5. Choose **Unlock**.
6. Enter the Google Client ID you obtained from Google, and then choose **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 (p. 238). Add 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 (such as web, iOS, Android).
5. Under **Credentials > Add Credentials**, create a Service Account. The console 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

Choose **Manage Identity Pools** from the Amazon Cognito Console home page:

1. Choose the name of the identity pool for which you want to enable Google as an external provider. The **Dashboard** page for your identity pool appears.
2. In the top-right corner of the **Dashboard** page, choose **Edit identity pool**. The Edit identity pool page appears.
3. Scroll down and choose **Authentication providers** to expand it.
4. Choose the **Google** tab.
5. Choose **Unlock**.
6. Enter the Google Client ID you obtained from Google, and then choose **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 that 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 Amazon 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 (p. 238).
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 that you created in the previous steps.
4. After creating the provider, choose the provider name and add two more audiences, providing the two remaining client IDs.

Configure the External Provider in the Amazon Cognito Console

Choose **Manage Identity Pools** from the Amazon Cognito Console home page:

1. Choose the name of the identity pool for which you want to enable Google as an external provider. The **Dashboard** page for your identity pool appears.
2. In the top-right corner of the **Dashboard** page, choose **Edit identity pool**. The Edit identity pool page appears.
3. Scroll down and choose **Authentication providers** to expand it.
4. Choose the **Google** tab.

5. Choose **Unlock**.
6. Enter the Google Client ID you obtained from Google, and then choose **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 with an Amazon Cognito identity. Unfortunately, the Google SDK for Xamarin doesn't allow you to retrieve the OpenID Connect token, so you must 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 (p. 238). Add 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.

Sign in with Apple (Identity Pools)

Amazon Cognito integrates with Sign in with Apple to provide federated authentication for your mobile application and web application users. This section explains how to register and set up your application using Sign in with Apple as an identity provider.

Adding Sign in with Apple as an authentication provider to an identity pool is a two-step process. First you integrate Sign in with Apple in an application, and then you configure Sign in with Apple in identity pools.

Set Up Sign in with Apple

To configure Sign in with Apple as an identity provider, you must register your application with the Apple to receive client ID.

1. Create a developer account with Apple.
2. Sign in with your Apple credentials.
3. In the left navigation pane, choose **Certificates, IDs & Profiles**.
4. In the left navigation pane, choose **Identifiers**.
5. On the **Identifiers** page, choose the **+**icon.
6. On the **Register a New Identifier** page, choose **App IDs** and then choose **Continue**.
7. On the **Register an App ID** page, do the following:

    a. Under **Description**, type a description.
    b. Under **Bundle ID,** type an identifier. Make a note of this bundle ID as you will need this value to configure Apple as a provider in the identity pool.
    c. Under **Capabilities**, choose **Sign In with Apple** and then choose **Edit**.
    d. On the **Sign in with Apple: App ID Configuration** page, select the appropriate setting for you app, Then choose **Save**.
    e. Choose **Continue**.
8. On the **Confirm your App ID** page, choose **Register**.
9. Proceed to step 10 if you want to integrate Sign in with Apple with a native iOS application. Step 11 is for applications that you want to integrate with Sign in with Apple JS.
10. On the **Identifiers** page, pause on **App IDs** on the right side of the page. Choose **Services IDs** and then choose the **plus +** icon.
11. On the **Register a New Identifier** page, choose **Services IDs** and then choose **Continue**.
12. On the **Register a Services ID** page, do the following:

    a. Under **Description**, type a description.
    b. Under **Identifier**, type an identifier. Make a note of the services ID as you will need this value to configure Apple as a provider in your identity pool.
    c. Select **Sign In with Apple** and then choose **Configure**.
    d. On the **Web Authentication Configuration** page, choose a **Primary App ID**. Under **Website URL's**, choose the **+** icon. For **Domains and Subdomains**, enter the domain name of your app. In **Return URLs,** enter the callback URL that the authorization redirects to after Sign in with Apple authentication.
    e. Choose **Next**.
    f. Choose **Continue** and then choose **Register**.
13. In the left navigation pane, choose **Keys**.
14. On the **Keys** page, choose the **+** icon.
15. On the **Register a New Key** page, do the following:

a. Under **Key Name**, type a key name.
b. Choose **Sign In with Apple** and then choose **Configure**.
c. On the **Configure Key** page, choose a **Primary App ID** and then choose **Save**.
d. Choose **Continue** and then choose **Register**.

Note
To integrate Sign in with Apple with a native iOS application, see Implementing User Authentication with Sign in with Apple.
To integrate Sign in with Apple in a platform other than native iOS, see Sign in with Apple JS.

Configure the External provider in the Amazon Cognito Federated Identities Console

Use the following procedure to configure your external provider.

1. Choose **Manage Identity Pools** from the Amazon Cognito console home page.
2. Choose the name of the identity pool for which you want to enable Apple as an external provider.
3. In the top-right corner of the dashboard, choose **Edit identity pool**.
4. Scroll down and choose **Authentication providers** to expand it.
5. Choose the **Apple** tab.
6. Enter the Bundle ID that you obtained from https://developer.apple.com. Then choose **Save Changes.**
7. If you use Sign in with Apple with native iOS applications, enter the `BundleID` you obtained from developer.apple.com. Or if you use Sign in with Apple with web or other applications, enter the service ID. Then choose **Save Changes**.

Sign in with Apple as a Provider in the Amazon Cognito Federated Identities CLI Examples

This example creates an identity pool named `MyIdentityPool` with Sign in with Apple as an identity provider.

```
aws cognito-identity create-identity-pool --identity-pool-name MyIdentityPool
--supported-login-providers appleid.apple.com="sameple.apple.clientid"
```

For more information, see Create identity pool

Generate an Amazon Cognito identity ID

This example generates (or retrieves) an Amazon Cognito ID. This is a public API so you don't need any credentials to call this API.

```
aws cognito-identity get-id --identity-pool-id SampleIdentityPoolId --logins
appleid.apple.com="SignInWithAppleIdToken"
```

For more information, see get-id.

Get credentials for an Amazon Cognito identity ID

This example returns credentials for the provided identity ID and Sign in with Apple login. This is a public API so you don't need any credentials to call this API.

```
aws cognito-identity get-credentials-for-identity --identity-id
SampleIdentityId --logins appleid.apple.com="SignInWithAppleIdToken"
```

For more information, see get-credentials-for-identity

Use Sign in with Apple: Android

Apple doesn't provide an SDK that supports Sign in with Apple for Android. You can use the web flow in a web view instead.

- To configure Sign in with Apple in your application, follow Configuring Your Web page for Sign In with Apple in the Apple documentation.
- To add a **Sign in with Apple** button to your Android user interface, follow Displaying and Configuring Sign In with Apple Buttons in the Apple documentation.
- To securely authenticate users using Sign in with Apple, follow Configuring Your webpage for Sign In with Apple in the Apple documentation.

Sign in with Apple uses a session object to track its state. Amazon Cognito uses the ID token from this session object to authenticate the user, generate the unique identifier, and, if needed, grant the user access to other AWS resources.

```
@Override
public void onSuccess(Bundle response) {
    String token = response.getString("id_token");
    Map<String, String> logins = new HashMap<String, String>();
    logins.put("appleid.apple.com", token);
    credentialsProvider.setLogins(logins);
}
```

Use Sign in with Apple: iOS - Objective-C

Apple provided SDK support for Sign in with Apple in native iOS applications. To implement user authentication with Sign in with Apple in native iOS devices, follow Implementing User Authentication with Sign in with Apple in the Apple documentation.

Amazon Cognito uses the ID token to authenticate the user, generate the unique identifier, and, if needed, grant the user access to other AWS resources.

```
(void)finishedWithAuth: (ASAuthorizationAppleIDCredential *)auth error: (NSError *) error {
        NSString *idToken = [ASAuthorizationAppleIDCredential
 objectForKey:@"identityToken"];
        credentialsProvider.logins = @{ "appleid.apple.com": idToken };
    }
```

Use Sign in with Apple: iOS - Swift

Apple provided SDK support for Sign in with Apple in native iOS applications. To implement user authentication with Sign in with Apple in native iOS devices, follow Implementing User Authentication with Sign in with Apple in the Apple documentation.

Amazon Cognito uses the ID token to authenticate the user, generate the unique identifier, and, if needed, grant the user access to other AWS resources.

For more information on setting up Sign in with Apple in iOS, see Set up Sign in with Apple

```
func finishedWithAuth(auth: ASAuthorizationAppleIDCredential!, error: NSError!) {
    if error != nil {
      print(error.localizedDescription)
    }
    else {
      let idToken = auth.identityToken,
      credentialsProvider.logins = ["appleid.apple.com": idToken!]
    }
}
```

Use Sign in with Apple: JavaScript

Apple doesn’t provide an SDK that supports Sign in with Apple for JavaScript. You can use the web flow in a web view instead.

- To configure Sign in with Apple in your application, follow Configuring Your Web page for Sign In with Apple in the Apple documentation.
- To add a **Sign in with Apple** button to your JavaScript user interface, follow Displaying and Configuring Sign In with Apple Buttons in the Apple documentation.
- To securely authenticate users using Sign in with Apple, follow Configuring Your Web page for Sign In with Apple in the Apple documentation.

Sign in with Apple uses a session object to track its state. Amazon Cognito uses the ID token from this session object to authenticate the user, generate the unique identifier, and, if needed, grant the user access to other AWS resources.

```
function signinCallback(authResult) {
     // Add the apple's id token to the Amazon Cognito credentials login map.
     AWS.config.credentials = new AWS.CognitoIdentityCredentials({
        IdentityPoolId: 'IDENTITY_POOL_ID',
        Logins: {
           'appleid.apple.com': authResult['id_token']
        }
     });

     // Obtain AWS credentials
     AWS.config.credentials.get(function(){
        // Access AWS resources here.
     });
}
```

Use Sign in with Apple: Xamarin

We don’t have an SDK that supports Sign in with Apple for Xamarin. You can use the web flow in a web view instead.

- To configure Sign in with Apple in your application, follow Configuring Your Web page for Sign In with Apple in the Apple documentation.
- To add a **Sign in with Apple** button to your Xamarin user interface, follow Displaying and Configuring Sign In with Apple Buttons in the Apple documentation.
- To securely authenticate users using Sign in with Apple, follow Configuring Your Web page for Sign In with Apple in the Apple documentation.

Sign in with Apple uses a session object to track its state. Amazon Cognito uses the ID 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 the token, you can set it in your `CognitoAWSCredentials`:

```
credentials.AddLogin("appleid.apple.com", token);
```

Open ID Connect Providers (Identity Pools)

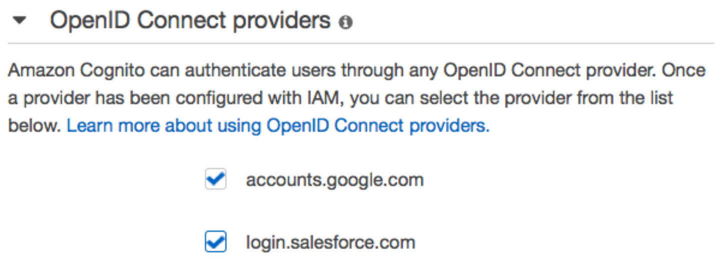
OpenID Connect is an open standard for authentication that is supported by a number of login providers. Amazon Cognito supports linking of identities with OpenID Connect providers that are configured through AWS Identity and Access Management.

Adding an OpenID Connect Provider

For information on how to create an OpenID Connect Provider, see the IAM documentation.

Associating a Provider with Amazon Cognito

Once you've created an OpenID Connect provider in the IAM Console, you can associate it with 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 with 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.

> **Note**
> As specified here (https://tools.ietf.org/html/rfc7523), Amazon Cognito allows for a grace period of 5 minutes to handle any clock skew between systems.

1. The `iss` parameter must match the key used in the logins map (such as 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 that you configured. This dictionary acts 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 Amazon
 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 Providers (Identity Pools)

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 Identity Pools**, 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. 194) 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 application. 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 the Amazon 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 (Identity Pools)

Amazon Cognito supports developer authenticated identities, in addition to web identity federation through Facebook (Identity Pools) (p. 219), Google (Identity Pools) (p. 227), Login with Amazon

(Identity Pools) (p. 224), and Sign in with Apple (Identity Pools) (p. 234). 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 Identity Pools (Federated Identities) Authentication Flow (p. 194).

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. 3).

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.

```
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.put("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 (Login with Amazon, Sign in with Apple, Facebook, or Google), 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.put("YOUR_IDENTITY_PROVIDER_NAME","YOUR_END_USER_IDENTIFIER");
logins.put("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, Sign in with Apple, 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, Sign in with Apple, 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, Sign in with Apple, 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, Sign in with Apple, 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, Sign in with Apple, 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 Unauthenticated Users to Authenticated Users (Identity Pools)

Amazon Cognito identity pools support both authenticated and unauthenticated users. Unauthenticated users receive access to your AWS resources even if they aren't logged in with any of your identity providers (IdPs). This degree of access is useful to display content to users before they log in. Each unauthenticated user has a unique identity in the identity pool, even though they haven't been individually logged in and authenticated.

This section describes the case where your user chooses to switch from logging in with an unauthenticated identity to using an authenticated identity.

Android

Users can log in to your application as unauthenticated guests. Eventually they might decide to log in using one of the supported IdPs. Amazon Cognito makes sure that an old identity retains the same unique identifier as the new one, and that the profile data is 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 log in to your application as unauthenticated guests. Eventually they might decide to log in using one of the supported IdPs. Amazon Cognito makes sure that an old identity retains the same unique identifier as the new one, and that the profile data is 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 log in to your application as unauthenticated guests. Eventually they might decide to log in using one of the supported IdPs. Amazon Cognito makes sure that an old identity retains the same unique identifier as the new one, and that the profile data is 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

Initially Unauthenticated User

Users typically start with the unauthenticated role. For this role, you set the credentials property of your configuration object without a Logins property. In this case, your default configuration might look like the following:

```
// set the default config object
var creds = new AWS.CognitoIdentityCredentials({
    IdentityPoolId: 'us-east-1:1699ebc0-7900-4099-b910-2df94f52a030'
});
AWS.config.credentials = creds;
```

Switch to Authenticated User

When an unauthenticated user logs in to an IdP and you have a token, you can switch the user from unauthenticated to authenticated by calling a custom function that updates the credentials object and adds the Logins token:

```

// Called when an identity provider has a token for a logged in user
function userLoggedIn(providerName, token) {
    creds.params.Logins = creds.params.Logins || {};
    creds.params.Logins[providerName] = token;

    // Expire credentials to refresh them on the next request
    creds.expired = true;
}
```

You can also create a `CognitoIdentityCredentials` object. If you do, you must reset the credentials properties of any existing service objects to reflect the updated credentials configuration information. See Using the Global Configuration Object.

For more information about the `CognitoIdentityCredentials` object, see AWS.CognitoIdentityCredentials in the AWS SDK for JavaScript API Reference.

Unity

Users can log in to your application as unauthenticated guests. Eventually they might decide to log in using one of the supported IdPs. Amazon Cognito makes sure that an old identity retains the same unique identifier as the new one, and that the profile data is 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 log in to your application as unauthenticated guests. Eventually they might decide to log in using one of the supported IdPs. Amazon Cognito makes sure that an old identity retains the same unique identifier as the new one, and that the profile data is 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

If you're new to Amazon Cognito Sync, use AWS AppSync. Like Amazon Cognito Sync, AWS AppSync is a service for synchronizing application data across devices.
It enables user data like app preferences or game state to be synchronized. It also extends these capabilities by allowing multiple users to synchronize and collaborate in real time on shared data.

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

If you're new to Amazon Cognito Sync, use AWS AppSync. Like Amazon Cognito Sync, AWS AppSync is a service for synchronizing application data across devices.
It enables user data like app preferences or game state to be synchronized. It also extends these capabilities by allowing multiple users to synchronize and collaborate in real time on shared data.

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://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on 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 Identity Pools (Federated Identities) (p. 187) 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. 256) for more information.

Synchronizing Data

If you're new to Amazon Cognito Sync, use AWS AppSync. Like Amazon Cognito Sync, AWS AppSync is a service for synchronizing application data across devices.
It enables user data like app preferences or game state to be synchronized. It also extends these capabilities by allowing multiple users to synchronize and collaborate in real time on shared 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. 211).
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. 211).
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. 211).
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. 211).
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. 211).
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. 211).

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. 263).

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. 263).

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. 263).

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. 263).

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. 263).

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. 263).

Handling Callbacks

If you're new to Amazon Cognito Sync, use AWS AppSync. Like Amazon Cognito Sync, AWS AppSync is a service for synchronizing application data across devices.
It enables user data like app preferences or game state to be synchronized. It also extends these capabilities by allowing multiple users to synchronize and collaborate in real time on shared data.

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

If you're new to Amazon Cognito Sync, use AWS AppSync. Like Amazon Cognito Sync, AWS AppSync is a service for synchronizing application data across devices.
It enables user data like app preferences or game state to be synchronized. It also extends these capabilities by allowing multiple users to synchronize and collaborate in real time on shared data.

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 **Manage Identity Pools**. The **Federated Identities** 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 Android Implementation 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

If you're new to Amazon Cognito Sync, use AWS AppSync. Like Amazon Cognito Sync, AWS AppSync is a service for synchronizing application data across devices.
It enables user data like app preferences or game state to be synchronized. It also extends these capabilities by allowing multiple users to synchronize and collaborate in real time on shared data.

Amazon Cognito Streams gives developers control and insight into their data stored in Amazon Cognito. Developers can now configure a Kinesis stream to receive events as data is updated and synchronized. Amazon Cognito can push each dataset change to a 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 **Manage Identity Pools**. The Manage Federated Identities 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

If you're new to Amazon Cognito Sync, use AWS AppSync. Like Amazon Cognito Sync, AWS AppSync is a service for synchronizing application data across devices.
It enables user data like app preferences or game state to be synchronized. It also extends these capabilities by allowing multiple users to synchronize and collaborate in real time on shared data.

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 Manage Federated Identities. The Manage Federated Identities 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);
};
```

Security in Amazon Cognito

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Amazon Cognito, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations

This documentation helps you understand how to apply the shared responsibility model when using Amazon Cognito. It shows you how to configure Amazon Cognito to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon Cognito resources.

Contents

Data Protection in Amazon Cognito

The AWS shared responsibility model applies to data protection in Amazon Cognito (Amazon Cognito). As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.

- Use SSL/TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with Amazon Cognito or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into Amazon Cognito or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

Data-Encryption

Data encryption typically falls into two categories: encryption at rest and encryption in transit.

Encryption at Rest

Data within Amazon Cognito is encrypted at rest in accordance with industry standards.

Encryption in Transit

All requests to Amazon Cognito must be made over the Transport Layer Security protocol (TLS). Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

> **Note**
> Amazon Cognito encrypts customer content internally and doesn't support customer provided keys.

Identity and Access Management for Amazon Cognito

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Cognito resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Authentication with a User Pool (p. 305)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Cognito.

Service user – If you use the Cognito service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Cognito features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Cognito, see Troubleshooting Amazon Cognito identity and access (p. 300).

Service administrator – If you're in charge of Cognito resources at your company, you probably have full access to Cognito. It's your job to determine which Cognito features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Cognito, see How Amazon Cognito works with IAM (p. 291).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Cognito. To view example Cognito identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon Cognito (p. 298).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, resources, and condition keys for Amazon Cognito in the *Service Authorization Reference*.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

    - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon Cognito works with IAM

Before you use IAM to manage access to Cognito, learn what IAM features are available to use with Cognito.

IAM features you can use with Amazon Cognito

IAM feature	Cognito support
Identity-based policies (p. 292)	Yes
Resource-based policies (p. 292)	No
Policy actions (p. 293)	Yes
Policy resources (p. 294)	Yes
Policy condition keys (p. 295)	Yes
ACLs (p. 296)	No
ABAC (tags in policies) (p. 296)	Partial
Temporary credentials (p. 296)	Yes
Principal permissions (p. 297)	No
Service roles (p. 297)	Yes
Service-linked roles (p. 297)	Yes

To get a high-level view of how Cognito and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Cognito

Supports identity-based policies	Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Cognito

To view examples of Cognito identity-based policies, see Identity-based policy examples for Amazon Cognito (p. 298).

Resource-based policies within Cognito

Supports resource-based policies	No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Policy actions for Cognito

Supports policy actions	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Cognito actions, see Actions defined by Amazon Cognito in the *Service Authorization Reference*.

Policy actions in Cognito use the following prefix before the action:

```
cognito-identity
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "cognito-identity:action1",
      "cognito-identity:action2"
         ]
```

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`

To view examples of Cognito identity-based policies, see Identity-based policy examples for Amazon Cognito (p. 298).

Policy resources for Cognito

Supports policy resources	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

Amazon Resource Names (ARNs)

ARNs for Amazon Cognito Federated Identities

In Amazon Cognito identity pools (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
```

To see a list of Cognito resource types and their ARNs, see Resources defined by Amazon Cognito in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon Cognito.

To view examples of Cognito identity-based policies, see Identity-based policy examples for Amazon Cognito (p. 298).

Policy condition keys for Cognito

Supports policy condition keys	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of Cognito condition keys, see Condition keys for Amazon Cognito in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Amazon Cognito.

To view examples of Cognito identity-based policies, see Identity-based policy examples for Amazon Cognito (p. 298).

Access control lists (ACLs) in Cognito

Supports ACLs	No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Attribute-based access control (ABAC) with Cognito

Supports ABAC (tags in policies)	Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

For more information about ABAC, see What is ABAC? in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using Temporary credentials with Cognito

Supports temporary credentials	Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for Cognito

Supports principal permissions	No

When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, resources, and condition keys for Amazon Cognito in the *Service Authorization Reference*.

Service roles for Cognito

Supports service roles	Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

For details about Cognito service roles, see Enable Push Synchronization (p. 193) and Push Sync (p. 275).

Warning
Changing the permissions for a service role might break Cognito functionality. Edit service roles only when Cognito provides guidance to do so.

Service-linked roles for Cognito

Supports service-linked roles	Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing Cognito service-linked roles, see Using Service-Linked Roles for Amazon Cognito (p. 302).

Identity-based policy examples for Amazon Cognito

By default, IAM users and roles don't have permission to create or modify Cognito resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform actions on the resources that they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating policies on the JSON tab in the *IAM User Guide*.

Topics

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Cognito resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started using AWS managed policies** – To start using Cognito quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Using the Cognito console

To access the Amazon Cognito console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Cognito resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

To ensure that users and roles can still use the Cognito console, also attach the Cognito `ConsoleAccess` or `ReadOnly` AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

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"
    }
  ]
}
```

Troubleshooting Amazon Cognito identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Cognito and IAM.

Topics

I am not authorized to perform an action in Cognito

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but does not have the fictional `cognito-identity:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: cognito-
identity:GetWidget on resource: my-example-widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *my-example-widget* resource using the cognito-identity:*GetWidget* action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the iam:PassRole action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Cognito.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named marymajor tries to use the console to perform an action in Cognito. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the iam:PassRole action.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, AKIAIOSFODNN7EXAMPLE) and a secret access key (for example, wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

Important
Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access Cognito

To allow others to access Cognito, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Cognito.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my Cognito resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Cognito supports these features, see How Amazon Cognito works with IAM (p. 291).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Using Service-Linked Roles for Amazon Cognito

Amazon Cognito uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon Cognito. Service-linked roles are predefined by Amazon Cognito and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon Cognito easier because you don't have to manually add the necessary permissions. Amazon Cognito defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon Cognito can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon Cognito resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-Linked Role Permissions for Amazon Cognito

Amazon Cognito uses the following service-linked roles:

- **AmazonCognitoIdpEmailService** – Allows Amazon Cognito user pools service to use your Amazon SES identities for sending email.

- **AmazonCognitoIdp** – Allows Amazon Cognito user pools to publish events and configure endpoints for your Amazon Pinpoint projects.

The `AmazonCognitoIdpEmailService` service-linked role trusts the following services to assume the role:

- `email.cognito-idp.amazonaws.com`

The role permissions policy allows Amazon Cognito to complete the following actions on the specified resources:

Allowed Actions for AmazonCognitoidpEmailService

- Action: `ses:SendEmail` and `ses:SendRawEmail`
- Resource: `*`

The policy denies Amazon Cognito the ability to complete the following actions on the specified resources:

Denied Actions

- Action: `ses:List*`
- Resource: `*`

With these permissions, Amazon Cognito can use your verified email addresses in Amazon SES only to email your users. Amazon Cognito emails your users when they perform certain actions in the client app for a user pool, such as signing up or resetting a password.

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

AmazonCognitoIdp

The AmazonCognitoIdp service-linked role trusts the following services to assume the role:

- `email.cognito-idp.amazonaws.com`

The role permissions policy allows Amazon Cognito to complete the following actions on the specified resources:

Allowed Actions for AmazonCognitoIdp

- Action: `cognito-idp:Describe`
- Resource: `*`

With this permission, Amazon Cognito can call `Describe` Amazon Cognito API operations for you.

Note

When you integrate Amazon Cognito with Amazon Pinpoint using `createUserPoolClient` and `updateUserPoolClient`, resource permissions will be added to the SLR as an inline policy. The inline policy will provide `mobiletargeting:UpdateEndpoint` and `mobiletargeting:PutEvents` permissions. These permissions allow Amazon Cognito to publish events and configure endpoints for Pinpoint projects you integrate with Cognito.

Creating a Service-Linked Role for Amazon Cognito

You don't need to manually create a service-linked role. When you configure a user pool to use your Amazon SES configuration to handle email delivery in the AWS Management Console, the AWS CLI, or the Amazon Cognito API, Amazon Cognito creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you configure a user pool to use your Amazon SES configuration to handle email delivery, Amazon Cognito creates the service-linked role for you again.

Before Amazon Cognito can create this role, the IAM permissions that you use to set up your user pool must include the `iam:CreateServiceLinkedRole` action. For more information about updating permissions in IAM, see Changing Permissions for an IAM User in the *IAM User Guide*.

Editing a Service-Linked Role for Amazon Cognito

Amazon Cognito doesn't allow you to edit the AmazonCognitoIdpEmailService or AmazonCognitoIdpEmailService service-linked roles. After you create a service-linked role, you can't change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide*.

Deleting a Service-Linked Role for Amazon Cognito

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. Before you can delete AmazonCognitoIdpEmailService or AmazonCognitoIdpEmailService service-linked roles, you must do either of the following for each user pool that uses the role:

- Delete the user pool.
- Update the email settings in the user pool to use the default email functionality. The default setting does not use the service-linked role.

Remember to do these actions in each AWS region that contains a user pool that uses the role.

> **Note**
> If the Amazon Cognito service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete an Amazon Cognito user pool

1. Sign in to the AWS Management Console and open the Amazon Cognito console at https://console.aws.amazon.com/cognito.
2. Choose **Manage User Pools**.
3. On the **Your User Pools** page, choose the user pool that you want to delete.
4. Choose **Delete pool**.
5. In the **Delete user pool** window, type **`delete`**, and choose **Delete pool**.

To update an Amazon Cognito user pool to use the default email functionality

1. Sign in to the AWS Management Console and open the Amazon Cognito console at https://console.aws.amazon.com/cognito.
2. Choose **Manage User Pools**.
3. On the **Your User Pools** page, choose the user pool that you want to update.
4. In the navigation menu on the left, choose **Message customizations**.
5. Under **Do you want to send emails through your Amazon SES Configuration?**, choose **No - Use Cognito (Default)**.
6. When you finish setting your email account options, choose **Save changes**.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete AmazonCognitoIdpEmailService or AmazonCognitoIdpEmailService service-linked roles. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

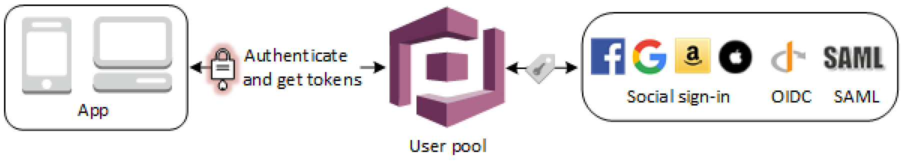
Supported Regions for Amazon Cognito Service-Linked Roles

Amazon Cognito supports using service-linked roles in all of the regions where the service is available. For more information, see AWS Regions and Endpoints.

Authentication with a User Pool

Your app users can sign in either directly through a user pool, or federate through a third-party identity provider (IdP). The user pool manages the overhead of handling the tokens that are returned from social sign-in through Facebook, Google, Amazon, and Apple, and from OpenID Connect (OIDC) and SAML IdPs.

After successful authentication, Amazon Cognito returns user pool tokens to your app. You can use the tokens to grant your users access to your own server-side resources, or to the Amazon API Gateway. Or, you can exchange them for AWS credentials to access other AWS services.



User pool token handling and management for your web or mobile app is provided on the client side through Amazon Cognito SDKs. Likewise, the Mobile SDK for iOS and the Mobile SDK for Android automatically refresh your ID and access tokens if there is a valid (non-expired) refresh token present, and the ID and access tokens have a minimum remaining validity of 5 minutes. For information on the SDKs, and sample code for JavaScript, Android, and iOS see Amazon Cognito User Pool SDKs.

After your app user successfully signs in, Amazon Cognito creates a session and returns an ID, access, and refresh token for the authenticated user.

JavaScript

```
// Amazon Cognito creates a session which includes the id, access, and refresh tokens
 of an authenticated user.

var authenticationData = {
        Username : 'username',
        Password : 'password',
    };
    var authenticationDetails = new
 AmazonCognitoIdentity.AuthenticationDetails(authenticationData);
    var poolData = { UserPoolId : 'us-east-1_ExaMPle',
        ClientId : '1example23456789'
    };
    var userPool = new AmazonCognitoIdentity.CognitoUserPool(poolData);
    var userData = {
        Username : 'username',
        Pool : userPool
    };
    var cognitoUser = new AmazonCognitoIdentity.CognitoUser(userData);
    cognitoUser.authenticateUser(authenticationDetails, {
        onSuccess: function (result) {
            var accessToken = result.getAccessToken().getJwtToken();

            /* Use the idToken for Logins Map when Federating User Pools with identity
 pools or when passing through an Authorization Header to an API Gateway Authorizer */
            var idToken = result.idToken.jwtToken;
        },

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

});
```

Android

```
// Session is an object of the type CognitoUserSession, and includes the id, access,
 and refresh tokens for a user.

String idToken = session.getIdToken().getJWTToken();
String accessToken = session.getAccessToken().getJWT();
```

iOS - Swift

```
// AWSCognitoIdentityUserSession includes id, access, and refresh tokens for a user.

- (AWSTask<AWSCognitoIdentityUserSession *> *)getSession;
```

iOS - Objective-C

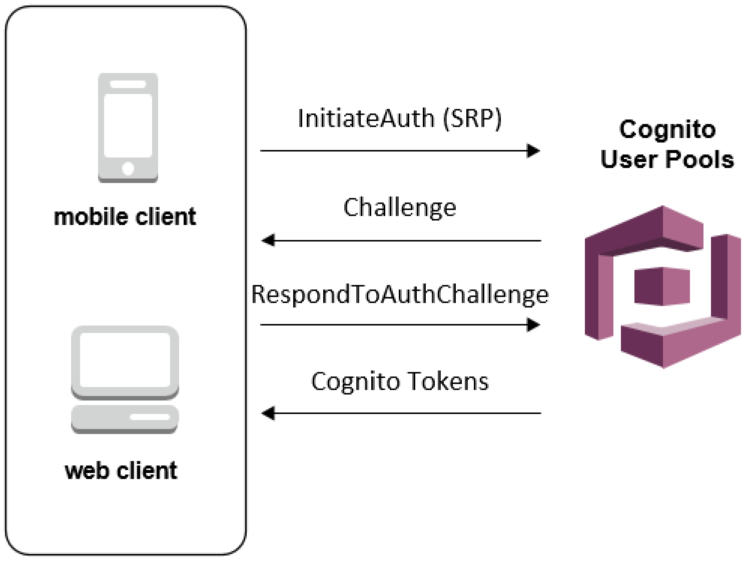
```
// AWSCognitoIdentityUserSession includes the id, access, and refresh tokens for a
 user.

[[user getSession:@"username" password:@"password" validationData:nil scopes:nil]
 continueWithSuccessBlock:^id _Nullable(AWSTask<AWSCognitoIdentityUserSession *> *
 _Nonnull task) {
    // success, task.result has user session
    return nil;
}];
```

Topics

User Pool Authentication Flow

Modern authentication flows incorporate new challenge types, in addition to a password, to verify the identity of users. We generalize authentication into two common steps, which are implemented through two API operations: `InitiateAuth` and `RespondToAuthChallenge`.

A user authenticates by answering successive challenges until authentication either fails or the user is issued tokens. With these two steps, which can be repeated to include different challenges, we can support any custom authentication flow.



You can customize your authentication flow with AWS Lambda triggers. These triggers issue and verify their own challenges as part of the authentication flow.

You can also use admin authentication flow for use on secure backend servers. You can use the user migration authentication flow to allow user migration without requiring your users to reset their passwords.

Note
We allow five failed sign-in attempts. After that we start temporary lockouts with exponentially increasing times starting at 1 second and doubling after each failed attempt up to about 15 minutes. Attempts during a temporary lockout period are ignored. After the temporary lockout period, if the next attempt fails, a new temporary lockout starts with twice the duration as the last. Waiting about 15 minutes without any attempts will also reset the temporary lockout. Please note that this behavior is subject to change.

Topics

Client-Side Authentication Flow

Here's how user pool authentication works for end-user client-side apps created with the AWS Mobile SDK for Android, AWS Mobile SDK for iOS, or AWS SDK for JavaScript:

1. The user enters their user name and password into the app.
2. The app calls the `InitiateAuth` operation with the user's user name and SRP details.

   This method returns the authentication parameters.

Note
The app generates the SRP details by using the Amazon Cognito SRP support in the Android, iOS, and JavaScript SDKs.

3. The app calls the `RespondToAuthChallenge` operation. If the call succeeds, it returns the user's tokens, and the authentication flow is complete.

   If another challenge is required, no tokens are returned. Instead, the call to `RespondToAuthChallenge` returns a session.
4. If `RespondToAuthChallenge` returns a session, the app calls `RespondToAuthChallenge` again, this time with the session and the challenge response (for example, MFA code).

Server-Side Authentication Flow

If you don't have an end-user app, but instead you're using a Java, Ruby, or Node.js secure backend or server-side app, you can use the authenticated server-side API for Amazon Cognito user pools.

For server-side apps, user pool authentication is similar to that for client-side apps, except for the following:

- The server-side app calls the `AdminInitiateAuth` API operation (instead of `InitiateAuth`). This operation requires AWS admin credentials. This operation returns the authentication parameters.
- Once it has the authentication parameters, the app calls the `AdminRespondToAuthChallenge` API operation (instead of `RespondToAuthChallenge`), which also requires AWS admin credentials.

The `AdminInitiateAuth` and `AdminRespondToAuthChallenge` operations can't accept user-name-and-password user credentials for admin sign-in, unless you explicitly enable them to do so by doing one of the following:

- Pass `ADMIN_USER_PASSWORD_AUTH` (formerly known as `ADMIN_NO_SRP_AUTH`) for the `ExplicitAuthFlow` parameter in your server-side app's call to `CreateUserPoolClient` or `UpdateUserPoolClient`.
- Choose `Enable sign-in API for server-based authentication (ADMIN_USER_PASSWORD_AUTH)` in the **App clients** tab in **Create a user pool**. For more information, see Configuring a User Pool App Client (p. 176).

Custom Authentication Flow

Amazon Cognito user pools also enable custom authentication flows, which can help you create a challenge/response-based authentication model using AWS Lambda triggers.

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 operation 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 one of the following:

- A challenge for the user along with a session and parameters.
- An error if the user fails to authenticate.
- ID, access, and refresh tokens, if the supplied parameters in the `InitiateAuth` call are sufficient to sign the user in. (Typically a challenge needs to be answered first, but your custom code needs to decide this.)

If Amazon Cognito responds to the `InitiateAuth` call with a challenge, the app gathers more input and calls the `RespondToAuthChallenge` operation, 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` API operations.

Built-in Authentication Flow and Challenges

Amazon Cognito has some built-in `AuthFlow` and `ChallengeName` values for a standard authentication flow to validate user name 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 included `PASSWORD_VERIFIER` as the `ChallengeName` and `SRP_B` in the challenge parameters. The app then calls `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 are returned. If multi-factor authentication (MFA) is enabled for the user, a `ChallengeName` of `SMS_MFA` is returned, and the app can provide the necessary code through another call to `RespondToAuthChallenge`.

Custom Authentication Flow and Challenges

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. It then outputs the next challenge name and Booleans that indicate whether the user is authenticated (and should be granted tokens) or not. 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 through SMS. It can use custom challenges such as CAPTCHA or secret questions.

Use SRP Password Verification in Custom Authentication Flow

If you want to include SRP in a custom authentication flow, you need to start with it.

- To initiate SRP password verification in a custom flow, the app calls `InitiateAuth` with `CUSTOM_AUTH` as the `Authflow`. It includes in the `AuthParameters` map `SRP_A:` (the SRP A value) and `CHALLENGE_NAME: SRP_A`.
- The `DefineAuthChallenge` Lambda trigger is invoked with an initial session of `challengeName: SRP_A` and `challengeResult: true`, and should respond with `challengeName: PASSWORD_VERIFIER`, `issueTokens: false`, and `failAuthentication: false`.
- The app next needs to call `RespondToAuthChallenge` with `challengeName: PASSWORD_VERIFIER` and the other parameters required for SRP in the `challengeResponses` map.
- If the password is verified, the `DefineAuthChallenge` Lambda trigger is invoked with a second session of `challengeName: PASSWORD_VERIFIER` and `challengeResult: true`. At that point the `DefineAuthChallenge` Lambda trigger can respond with `challengeName: CUSTOM_CHALLENGE` to start the custom challenge.

- If MFA is enabled for a user, it is handled automatically after the password is verified.

> **Note**
> The Amazon Cognito hosted sign-in webpage does not support the custom authentication flow.

For more information about the Lambda triggers, including sample code, see Customizing User Pool Workflows with Lambda Triggers (p. 67).

> **Note**
> The Amazon Cognito hosted sign-in web page does not support the custom authentication flow.

Admin authentication flow

The API operations described Custom Authentication Flow (p. 308) 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 API operations designed for use on secure backend servers if you want to avoid the SRP calculations. For these backend admin implementations, `AdminInitiateAuth` is used in place of `InitiateAuth`, and `AdminRespondToAuthChallenge` is used in place of `RespondToAuthChallenge`. When using these operations, the password can be submitted as plaintext so the SRP calculations are not needed. Here is an example:

```
AdminInitiateAuth Request {
    "AuthFlow":"ADMIN_USER_PASSWORD_AUTH",
    "AuthParameters":{
        "USERNAME":"<username>",
        "PASSWORD":"<password>"
        },
    "ClientId":"<clientId>",
    "UserPoolId":"<userPoolId>"
}
```

These admin authentication operations require developer credentials and use the AWS Signature Version 4 (SigV4) signing process. These operations are available in standard AWS SDKs including Node.js, which is convenient for use in Lambda functions. In order to use these operations and have them accept passwords in plaintext, you must enable them for the app in the console. Alternatively, you can pass `ADMIN_USER_PASSWORD_AUTH` for the `ExplicitAuthFlow` parameter in calls to `CreateUserPoolClient` or `UpdateUserPoolClient`. The `ADMIN_USER_PASSWORD_AUTH` `AuthFlow` is not accepted for the `InitiateAuth` and `RespondToAuthChallenge` operations.

In the `AdminInitiateAuth` response `ChallengeParameters`, the `USER_ID_FOR_SRP` attribute, if present, will contain the user's actual user name, not an alias (such as email address or phone number). In your call to `AdminRespondToAuthChallenge`, in the `ChallengeResponses`, you must pass this user name in the `USERNAME` parameter.

> **Note**
> Because it's designed for backend 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.

User migration authentication flow

A user migration Lambda trigger allows easy migration of users from a legacy user management system into your user pool. To avoid making your users reset their passwords during user migration, choose the `USER_PASSWORD_AUTH` authentication flow. This flow sends your users' passwords to the service over an encrypted SSL connection during authentication.

When you have completed migrating all your users, we recommend switching flows to the more secure SRP flow. The SRP flow does not send any passwords over the network.

To learn more about Lambda triggers see Customizing User Pool Workflows with Lambda Triggers (p. 67).

For more information about migrating users with a Lambda trigger see Importing Users into User Pools With a User Migration Lambda Trigger (p. 133).

Logging and Monitoring in Amazon Cognito

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Cognito and your other AWS solutions. Amazon Cognito currently supports the following two AWS services so that you can monitor your organization and the activity that happens within it.

- **AWS CloudTrail –** With CloudTrail you can capture API calls from the Amazon Cognito console and from code calls to the Amazon Cognito API operations. For example, when a user authenticates, CloudTrail can record details such as the IP address in the request, who made the request, and when it was made.
- **Amazon CloudWatch Metrics –** With CloudWatch metrics you can monitor, report and take automatic actions in case of an event in near real time. For example, you can create CloudWatch dashboards on the provided metrics to monitor your Amazon Cognito user pools, or you can create CloudWatch alarms on the provided metrics to notify you on breach of a set threshold.
- **Amazon CloudWatch Logs Insights** - With CloudWatch Logs Insights, you can configure CloudTrail to send events to CloudWatch for monitoring Amazon Cognito CloudTrail log files.

Topics

Tracking quotas and usage in CloudWatch and Service Quotas

You can monitor Amazon Cognito user pools using Amazon CloudWatch or using Service Quotas. CloudWatch collects raw data and processes it into readable, near-real-time metrics. In CloudWatch, you can set alarms that watch for certain thresholds and send notifications or take actions when those thresholds are met. To create a CloudWatch alarm for a service quota, see Create a CloudWatch alarm. Amazon Cognito metrics are available at five minute intervals. For more information about retention periods in CloudWatch, visit the Amazon CloudWatch FAQ page.

You can use Service Quotas to view and manage your Amazon Cognito user pools quota usage. The Service Quotas console has three features, view service quotas, request a service quota increase, and view current utilization. You can use the first feature to view quotas and see whether the quota is adjustable. You can use the second feature to request a Service Quotas increase. You can use the last feature to view quota utilization. This feature is only available after your account has been active for awhile. For more information on viewing quotas in the Service Quotas console, see Viewing Service Quotas.

Note
Amazon Cognito metrics are available at 5 minute intervals. For more information about retention periods in CloudWatch, visit the Amazon CloudWatch FAQ page.

Metrics for Amazon Cognito user pools

The following table lists the metrics available for Amazon Cognito user pools.

Note
Metrics that haven't had any new data points in the past two weeks don't appear in the console. They also don't appear when you enter their metric name or dimension names in the search box in the **All metrics** tab in the console. In addition, they are not returned in the results of a list-metrics command. The best way to retrieve these metrics is with the `get-metric-data` or `get-metric-statistics` commands in the AWS CLI.

Metric	Description
`SignUpSuccesses`	Provides the total number of successful user registration requests made to the Amazon Cognito user pool. A successful user registration request produces a value of 1, whereas an unsuccessful request produces a value of 0. A throttled request is also considered as an unsuccessful request, and hence a throttled request will also produce a count of 0. To find the percentage of successful user registration requests, use the `Average` statistic on this metric. To count the total number of user registration requests, use the `Sample Count` statistic on this metric. To count the total number of successful user registration requests, use the `Sum` statistic on this metric. To count the total number of failed user registration requests, use the CloudWatch `Math` expression and subtract the `Sum` statistic from the `Sample Count` statistic. This metric is published for each user pool for each user pool client. In case when the user registration is performed by an admin, the metric is published with the user pool client as `Admin`. Note that this metric is not emitted for User Import and User Migration cases. Metric dimension: `UserPool`, `UserPoolClient` Units: Count
`SignUpThrottles`	Provides the total number of throttled user registration requests made to the Amazon Cognito user pool. A count of 1 is published whenever a user registration request is throttled. To count the total number of throttled user registration requests, use the `Sum` statistic for this metric.

Metric	Description
	This metric is published for each user pool for each client. In case when the request that was throttled was made by an administrator, the metric is published with user pool client as `Admin`. Metric dimension: `UserPool`, `UserPoolClient` Units: Count
`SignInSuccesses`	Provides the total number of successful user authentication requests made to the Amazon Cognito user pool. A user authentication is considered successful when authentication token is issued to the user. A successful authentication produces a value of 1, whereas an unsuccessful request produces a value of 0. A throttled request is also considered as an unsuccessful request, and hence a throttled request will also produce a count of 0. To find the percentage of successful user authentication requests, use the `Average` statistic on this metric. To count the total number of user authentication requests, use the `Sample Count` statistic on this metric. To count the total number of successful user authentication requests, use the `Sum` statistic on this metric. To count the total number of failed user authentication requests, use the CloudWatch `Math` expression and subtract the `Sum` statistic from the `Sample Count` statistic. This metric is published for each user pool for each client. In case an invalid user pool client is provided with a request, the corresponding user pool client value in the metric contains a fixed value `Invalid` instead of the actual invalid value sent in the request. Note that requests to refresh the Amazon Cognito token is not included in this metric. There is a separate metric for providing `Refresh` token statistics. Metric dimension: `UserPool`, `UserPoolClient` Units: Count

Metric	Description
`SignInThrottles`	Provides the total number of throttled user authentication requests made to the Amazon Cognito user pool. A count of 1 is published whenever an authentication request is throttled. To count the total number of throttled user authentication requests, use the `Sum` statistic for this metric. This metric is published for each user pool for each client. In case an invalid user pool client is provided with a request, the corresponding user pool client value in the metric contains a fixed value `Invalid` instead of the actual invalid value sent in the request. Requests to refresh Amazon Cognito token is not included in this metric. There is a separate metric for providing `Refresh` token statistics. Metric dimension: `UserPool`, `UserPoolClient` Units: Count
`TokenRefreshSuccesses`	Provides the total number of successful requests to refresh an Amazon Cognito token that were made to the Amazon Cognito user pool. A successful refresh Amazon Cognito token request produces a value of 1, whereas an unsuccessful request produces a value of 0. A throttled request is also considered as an unsuccessful request, and hence a throttled request will also produce a count of 0. To find the percentage of successful requests to refresh an Amazon Cognito token, use the `Average` statistic on this metric. To count the total number of requests to refresh an Amazon Cognito token, use the `Sample Count` statistic on this metric. To count the total number of successful requests to refresh an Amazon Cognito token, use the `Sum` statistic on this metric. To count the total number of failed requests to refresh an Amazon Cognito token, use the CloudWatch `Math` expression and subtract the `Sum` statistic from the `Sample Count` statistic. This metric is published per each user pool client. If an invalid user pool client is in a request, the user pool client value contains a fixed value of `Invalid`. Metric dimension: `UserPool`, `UserPoolClient` Units: Count

Metric	Description
`TokenRefreshThrottles`	Provides the total number of throttled requests to refresh an Amazon Cognito token that were made to the Amazon Cognito user pool. A count of 1 is published whenever a refresh Amazon Cognito token request is throttled. To count the total number of throttled requests to refresh an Amazon Cognito token, use the `Sum` statistic for this metric. This metric is published for each user pool for each client. In case an invalid user pool client is provided with a request, corresponding user pool client value in the metric contains a fixed value `Invalid` instead of the actual invalid value sent in the request. Metric dimension: `UserPool`, `UserPoolClient` Units: Count
`FederationSuccesses`	Provides the total number of successful identity federation requests to the Amazon Cognito user pool. A successful identity federation request produces a value of 1, whereas an unsuccessful request produces a value of 0. A throttled request is also considered as an unsuccessful request, and hence a throttled request will also produce a count of 0. To find the percentage of successful identity federation requests, use the `Average` statistic on this metric. To count the total number of identity federation requests, use the `Sample Count` statistic on this metric. To count the total number of successful identity federation requests, use the `Sum` statistic on this metric. To count the total number of failed identity federation requests, use the CloudWatch `Math` expression and subtract the `Sum` statistic from the `Sample Count` statistic. Metric dimension: `UserPool`, `UserPoolClient`, `IdentityProvider` Units: Count

Metric	Description
`FederationThrottles`	Provides the total number of throttled identity federation requests to the Amazon Cognito user pool. A count of 1 is published whenever an identity federation request is throttled. To count the total number of throttled identity federation requests, use the `Sum` statistic for this metric. Metric dimension: `UserPool`, `UserPoolClient`, `IdentityProvider` Units: Count
`CallCount`	Provides the total number of calls customers made related to a category. This metric includes all the calls, such as throttled calls, failed calls, and successful calls. This metric is available in the **Usage** `nameSpace`. The category quota is enforced for each AWS account across all user pools in an account and Region. You can count the total number of calls in a category using the `Sum` statistic for this metric. Metric dimension: Service, Type, Resource, Class Units: Count
`ThrottleCount`	Provides the total number of throttled calls related to a category. This metric is available in the **Usage** `nameSpace`. This metric is published at the account level. You can count the total number of calls in a category, using the `Sum` statistic for this metric. Metric dimension: Service, Type, Resource, Class Units: Count

Dimensions for Amazon Cognito user pools

The following dimensions are used to refine the usage metrics that are published by Amazon Cognito. The dimensions only apply to `CallCount` and `ThrottleCount` metrics.

Dimension	Description
Service	The name of the AWS service containing the resource. For Amazon Cognito usage metrics, the value for this dimension is `Cognito user pool`.
Type	The type of entity that is being reported. The only valid value for Amazon Cognito usage metrics is API.
Resource	The type of resource that is running. The only valid value is category name.
Class	The class of resource being tracked. Amazon Cognito doesn't use the class dimension.

Use the Service Quotas console to track metrics

Amazon Cognito user pools is integrated with Service Quotas, which is a service that enables you to view and manage your quotas from a central location. You can use the Service Quotas console to view details about a specific quota, monitor quota utilization, request a quota increase, and create a CloudWatch alarm to track your quota utilization.

To view Amazon Cognito user pools service quotas utilization, complete the following steps.

1. Open the Service Quotas console.
2. In the navigation pane, choose **AWS services**.
3. From the **AWS services** list, enter Amazon Cognito user pools in the search field. The service quota page appears.
4. Scroll down to **Monitoring**.
5. In **Monitoring** you can view current service quota utilization in the graph.
6. In **Monitoring** select either one hour, three hours, twelve hours, one day, three days, or one week.
7. Select any area inside of the graph to view the service quota utilization percentage. From here, you can add the graph to your dashboard or use the action menu to select **View in metrics**, which will take you to the related metrics in the CloudWatch console.

Use the CloudWatch console to track metrics

You can track and collect Amazon Cognito user pools metrics using CloudWatch. The CloudWatch dashboard will display metrics about every AWS service you use. You can use CloudWatch to create metric alarms. The alarms can be setup to send you notifications or make a change to a specific resource that you are monitoring. To view service quota metrics in CloudWatch, complete the following steps.

1. Open the CloudWatch console.
2. In the navigation pane, choose **Metrics**.
3. In **All metrics** select a metric and a dimension.
4. Select the check box next to a metric. The metrics will appear in the graph.

   Note
   Metrics that haven't had any new data points in the past two weeks don't appear in the console. They also don't appear when you enter their metric name or dimension names in the search box

in the All metrics tab in the console, and they are not returned in the results of a list-metrics command. The best way to retrieve these metrics is with the `get-metric-data` or `get-metric-statistics` commands in the AWS CLI.

Create a CloudWatch alarm for a quota

Amazon Cognito provides CloudWatch usage metrics that correspond to the AWS service quotas for `CallCount` and `ThrottleCount` APIs. In the Service Quotas console, you can create alarms that alert you when your usage approaches a service quota. Use the following steps to set up a CloudWatch alarm using the Service Quotas console.

1. Open the Service Quota https://docs.aws.amazon.com/servicequotas/latest/userguide/ console.
2. In the navigation pane, choose **AWS services**.
3. Select **Amazon Cognito user pools**.
4. Select the quota you want to setup an alarm on.
5. Scroll down to **CloudWatch alarms**.
6. In **CloudWatch alarms**, choose**Create**,
7. In **Alarm threshold**, choose a threshold.
8. For **Alarm name**, enter a unique name.
9. Choose **Create**.

Logging Amazon Cognito API Calls with AWS CloudTrail

Amazon Cognito is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon Cognito. CloudTrail captures a subset of API calls for Amazon Cognito as events, including calls from the Amazon Cognito console and from code calls to the Amazon Cognito API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon Cognito. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Cognito, the IP address from which the request was made, who made the request, when it was made, and additional details.

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

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in Amazon Cognito, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon Cognito, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

Amazon Cognito User Pools

Amazon Cognito supports logging for all of the actions listed on the User Pool Actions page as events in CloudTrail log files. Amazon Cognito records `UserSub` but not `UserName` in CloudTrail logs for requests that are specific to a user. You can find a user for a given `UserSub` by calling the `ListUsers` API, and using a filter for sub.

> **Note**
> Hosted UI and federation calls are currently not included in CloudTrail.

Amazon Cognito Federated Identities

- CreateIdentityPool
- DeleteIdentityPool
- DescribeIdentityPool
- GetIdentityPoolRoles
- ListIdentityPools
- SetIdentityPoolRoles
- UpdateIdentityPool

Amazon Cognito Sync

Amazon Cognito supports logging for all of the actions listed on the Amazon Cognito Sync Actions page as events in CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Example: Amazon Cognito Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files 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"
       }
```

Analyzing Amazon Cognito CloudTrail events with Amazon CloudWatch Logs Insights

Amazon CloudWatch Logs Insights enables you to interactively search and analyze your Amazon Cognito CloudTrail events data. When you configure your trail to send events to CloudWatch Logs, CloudTrail sends only the events that match your trail settings.

To query or research your Amazon Cognito CloudTrail events, in the CloudTrail console, make sure that you select the **Management events** option in your trail settings so that you can monitor the management operations performed on your AWS resources. You can optionally select the **Insights events** option in your trail settings when you want to identify errors, unusual activity or unusual user behavior in your account.

Sample Amazon Cognito queries

You can use the following queries in the Amazon CloudWatch console.

General queries

Find the 25 most recently added log events.

```
fields @timestamp, @message | sort @timestamp desc | limit 25
| filter eventSource = "cognito-idp.amazonaws.com"
```

Get a list of the 25 most recently added log events that include exceptions.

```
fields @timestamp, @message | sort @timestamp desc | limit 25
| filter eventSource = "cognito-idp.amazonaws.com" and @message like /Exception/
```

Exception and Error Queries

Find the 25 most recently added log events with error code `NotAuthorizedException` along with Amazon Cognito user pool `sub`.

```
fields @timestamp, additionalEventData.sub as user | sort @timestamp desc | limit 25
| filter eventSource = "cognito-idp.amazonaws.com" and errorCode= "NotAuthorizedException"
```

Find the number of records with `sourceIPAddress` and corresponding `eventName`.

```
filter eventSource = "cognito-idp.amazonaws.com"
| stats count(*) by sourceIPAddress, eventName
```

Find the top 25 IP addresses that triggered a `NotAuthorizedException` error.

```
filter eventSource = "cognito-idp.amazonaws.com" and errorCode= "NotAuthorizedException"
| stats count(*) as count by sourceIPAddress, eventName
| sort count desc | limit 25
```

Find the top 25 IP addresses that called the `ForgotPassword` API.

```
filter eventSource = "cognito-idp.amazonaws.com" and eventName = 'ForgotPassword'
| stats count(*) as count by sourceIPAddress
| sort count desc | limit 25
```

Compliance Validation for Amazon Cognito

Third-party auditors assess the security and compliance of Amazon Cognito as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using Amazon Cognito is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – AWS Config; assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.

- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in Amazon Cognito

The AWS global infrastructure is built around AWS Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Topics

Regional Data Considerations

Amazon Cognito user pools are each created in one AWS Region, and they store the user profile data only in that region. User pools can send user data to a different AWS Region, depending on how optional features are configured.

- If the default no-reply@verificationemail.com email address setting is used for routing verification of emails addresses with Amazon Cognito user pools, emails are routed through the US West (Oregon) Region.
- If a different email address is used to configure Amazon Simple Email Service (Amazon SES) with Amazon Cognito user pools, that email address is routed through the AWS Region associated with the email address in Amazon SES.
- SMS messages from Amazon Cognito user pools are routed through the same region Amazon SNS unless noted otherwise on Configuring Email or Phone Verification.
- If Amazon Pinpoint analytics are used with Amazon Cognito user pools, the event data is routed to the US East (N. Virginia) Region.

Note
Amazon Pinpoint is available in several AWS Regions in North America, Europe, Asia, and Oceania. Amazon Pinpoint regions include the Amazon Pinpoint API. If a Amazon Pinpoint region is supported by Amazon Cognito, then Amazon Cognito will send events to Amazon Pinpoint projects within the *same* Amazon Pinpoint region. If a region *isn't* supported by Amazon Pinpoint, then Amazon Cognito will *only* support sending events in us-east-1. For Amazon Pinpoint detailed region information, see Amazon Pinpoint endpoints and quotas and Using Amazon Pinpoint Analytics with Amazon Cognito User Pools.

Infrastructure Security in Amazon Cognito

As a managed service, Amazon Cognito is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Amazon Cognito through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Configuration and Vulnerability Analysis in Amazon Cognito user pools

AWS handles basic security tasks like guest operating system (OS) and database patching, firewall configuration, and disaster recovery. These procedures have been reviewed and certified by the appropriate third parties. For more details, see the following resources:

- Compliance Validation for Amazon Cognito (p. 321)
- Shared Responsibility Model

Security Best Practices for Amazon Cognito user pools

You can add multi-factor authentication (MFA) to a user pool to protect the identity of your users. MFA adds a second authentication method that doesn't rely solely on user name and password. You can choose to use SMS text messages, or time-based one-time (TOTP) passwords as second factors in signing in your users. You can also use adaptive authentication with its risk-based model to predict when you might need another authentication factor. It's part of the user pool advanced security features, which also include protections against compromised credentials.

Topics

Adding Multi-Factor Authentication (MFA) to a User Pool

Multi-factor authentication (MFA) increases security for your app by adding another authentication method and not relying solely on user name and password. You can choose to use SMS text messages or time-based one-time (TOTP) passwords as second factors in signing in your users.

With adaptive authentication, you can configure your user pool to require second factor authentication in response to an increased risk level. To add adaptive authentication to your user pool, see Adding Advanced Security to a User Pool (p. 327).

When multi-factor authentication (MFA) is set to `required`for a user pool, all users must complete MFA to sign in. Each user needs at least one MFA factor such as SMS or TOTP setup to sign in. To avoid having users blocked from signing in when MFA is `required`, you must include the MFA setup in user onboarding.

If you enable SMS as an MFA factor, you can require phone numbers and have them verified during sign-up. If you have MFA set to `required` and only support SMS as a factor, most users will need to have a phone number. Users without phone numbers will need your support to add a phone number to their profile before they can sign in. You can use unverified phone numbers for SMS MFA. These numbers will have a verified status after multi-factor authentication succeeds.

Setting up users in user pools for TOTP tokens occurs during initial sign-in. The setting to enable or disable TOTP as an MFA factor for a user pool controls whether users can set up TOTP for themselves. If your users have set up TOTP, they can use it for MFA even if TOTP is later disabled for the user pool.

Topics

Prerequisites

Before you begin, you need the following:

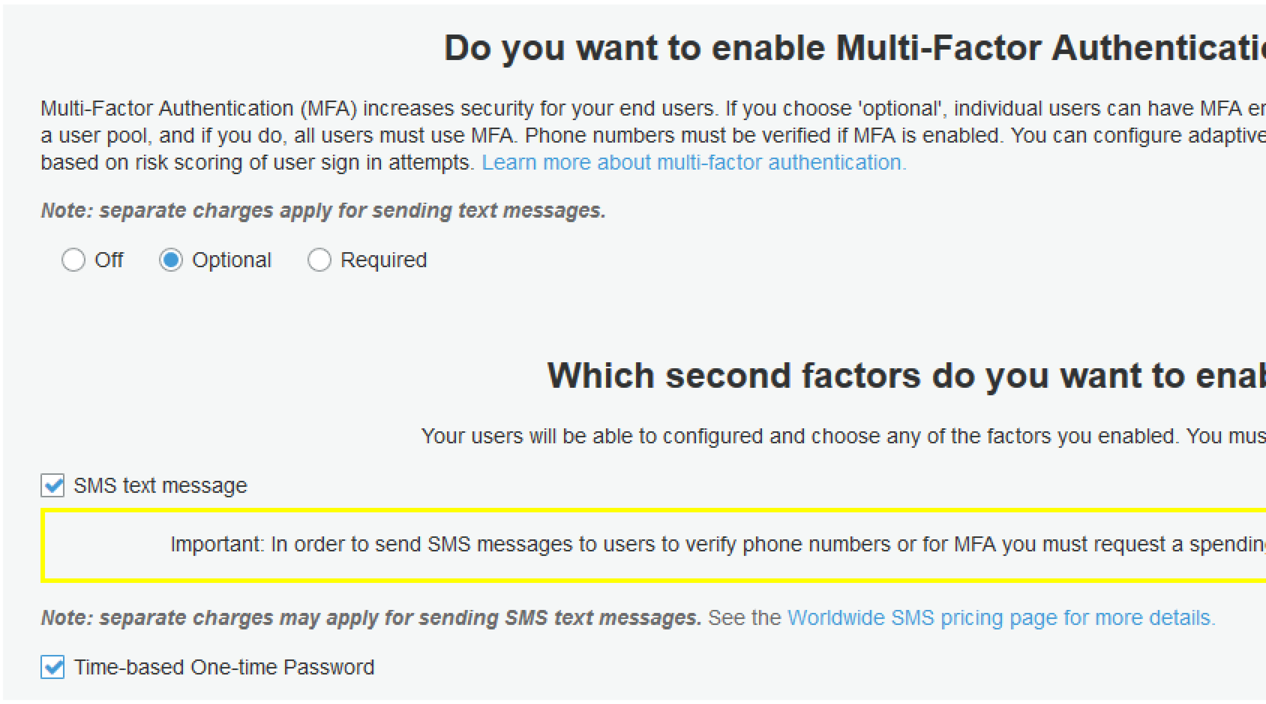
- You can only choose MFA as **Required** in the Amazon Cognito user pool console when you initially create a user pool. The SetUserPoolMfaConfig API operation can be used to set MFA to `required` for existing user pools.
- Phone numbers must be verified if MFA is enabled and **SMS text message** is chosen as a second factor.
- Advanced security features require that MFA is enabled, and set as optional in the Amazon Cognito user pool console. For more information, see Adding Advanced Security to a User Pool (p. 327).

Configuring Multi-Factor Authentication

You can configure MFA in the Amazon Cognito console.

To configure MFA in the Amazon Cognito console

1. From the left navigation bar, choose **MFA and verifications**.
2. Choose whether MFA is **Off**, **Optional**, or **Required**.



Do you want to enable Multi-Factor Authenticati

Multi-Factor Authentication (MFA) increases security for your end users. If you choose 'optional', individual users can have MFA er
a user pool, and if you do, all users must use MFA. Phone numbers must be verified if MFA is enabled. You can configure adaptive
based on risk scoring of user sign in attempts. Learn more about multi-factor authentication.

Note: separate charges apply for sending text messages.

Off [✓ Optional] Required

Which second factors do you want to enal

Your users will be able to configured and choose any of the factors you enabled. You mus

[✓ SMS text message]

Important: In order to send SMS messages to users to verify phone numbers or for MFA you must request a spendin

Note: separate charges may apply for sending SMS text messages. See the Worldwide SMS pricing page for more details.

[✓ Time-based One-time Password]

3. Choose **Optional** to enable MFA on a per-user basis or if you are using the risk-based adaptive authentication. For more information on adaptive authentication, see Adding Advanced Security to a User Pool (p. 327).
4. Choose which second factors to support in your app. Your users can use **SMS text message** or **Time-based One-time Password** as a second factor. We recommend using TOTP, which allows SMS to be used as a password recovery mechanism rather than as an authentication factor.
5. If you're using SMS text messages as a second factor and you don't have an IAM role defined with this permission, then you can create one in the console. Choose **Create role** to create an IAM role that allows Amazon Cognito to send SMS messages to your users on your behalf. For more information, see IAM Roles.
6. Choose **Save changes**.

SMS Text Message MFA

When a user signs in with MFA enabled, they first enter and submit their user name and password. The client app receives a `getMFA` response that indicates where the authorization code was sent. The client app should indicate to the user where to look for the code (such as which phone number the code was sent to). Next, it provides a form for entering the code. Finally, the client app submits the code to complete the sign-in process. The destination is masked, which hides all but the last 4 digits of the phone number. If an app is using the Amazon Cognito hosted UI, the ap shows a page for the user to enter the MFA code.

The SMS text message authorization code is valid for 3 minutes.

If a user no longer has access to their device where the SMS text message MFA codes are sent, they must request help from your customer service office. An administrator with necessary AWS account permissions can change the user's phone number, but only through the AWS CLI or the API.

When a user successfully goes through the SMS text message MFA flow, their phone number is also marked as verified.

Note
SMS for MFA 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.

Important
To ensure that SMS messages are sent to verify phone numbers and for SMS text message 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're requesting an SMS monthly spend limit increase.

To add MFA to your user pool, see Adding Multi-Factor Authentication (MFA) to a User Pool (p. 323).

TOTP Software Token MFA

Your user is challenged to complete authentication using a time-based one-time (TOTP) password. This option is enabled after the user name and password have been verified during activation of the TOTP software token for MFA. If your app is using the Amazon Cognito hosted UI to sign in users, the UI shows a second page. This page asks your user to submit their user name and password and then enter the TOTP password.

You can enable TOTP MFA for your user pool in the Amazon Cognito console, through the Amazon Cognito hosted UI, or using Amazon Cognito API operations. At the user pool level, you can configure MFA and enable TOTP MFA by calling SetUserPoolMfaConfig.

> **Note**
> If TOTP software token MFA isn't enabled for the user pool, users can't be associated or verified with the token. They receive this `SoftwareTokenMFANotFoundException` exception: "Software Token MFA has not been enabled by the userPool." Users who have previously associated and verified a TOTP token can continue to use it for MFA if the software token MFA is later disabled for the user pool.

Configuring TOTP for your user is a multi-step process where your user receives a secret code that they validate by entering a one-time password. Next, you can enable TOTP MFA for your user or set TOTP as the preferred MFA method for your user.

To add MFA to your user pool, see Adding Multi-Factor Authentication (MFA) to a User Pool (p. 323).

Associate the TOTP Token

Associating the TOTP token involves sending your user a secret code that they must validate with a one-time password. This part has three steps.

1. When your user chooses TOTP software token MFA, call AssociateSoftwareToken to return a unique generated shared secret key code for the user account. The request for this API method takes an access token or a session string, but not both. As a convenience, you can distribute the secret key as a quick response (QR) code.

   > **Note**
   > Calling AssociateSoftwareToken immediately disassociates the existing software token from the user account. If the user doesn't subsequently verify the software token, their account is essentially set up to authenticate without MFA. If MFA config is set to Optional at the user pool level, the user can then login without MFA. However, if MFA is set to Required for the user pool, the user will be asked to setup a new software token MFA during sign in.
2. The key code or QR code appears on your app. Your user needs to enter it into a TOTP-generating app such as Google Authenticator.
3. Your user enters the key code into the TOTP-generating app to associate a new account with your client app.

Verify the TOTP Token

The next step is to verify the TOTP token. Here is an overview of the process.

1. After a new TOTP account is associated with your app, it generates a temporary password.
2. Your user enters the temporary password into your app, which responds with a call to VerifySoftwareToken. On the Amazon Cognito service server, a TOTP code is generated and compared with your user's temporary password. If they match, then the service marks it as verified.
3. If the code is correct, check that the time used is in the range and within the maximum number of retries. Amazon Cognito also accepts TOTP tokens that are one 30-second window early or late to account for clock skew. If your user passes all of the steps, the verification is complete.

   If the code is wrong, the verification cannot be finished and your user can either try again or cancel. We recommend that your user sync the time of their TOTP-generating app.

Sign in with TOTP MFA

At this point, your user signs in with the time-based one-time password. The process is as follows.

1. Your user enters their user name and password to sign in to your client app.
2. The TOTP MFA challenge is invoked, and your user is prompted by your app to enter a temporary password.
3. Your user gets the temporary password from an associated TOTP-generating app.
4. Your user enters the TOTP code into your client app. Your app notifies the Amazon Cognito service to verify it. For each sign-in, RespondToAuthChallenge should be called to get a response to the new TOTP authentication challenge.
5. If the token is verified by Amazon Cognito, the sign-in is successful and your user continues with the authentication flow.

Remove the TOTP Token

Finally, your app should allow your user to remove the TOTP token:

1. Your client app should ask your user to enter their password.
2. If the password is correct, remove the TOTP token.

   Note
   A delete TOTP software token operation is not currently available in the API. This functionality is planned for a future release. Use SetUserMFAPreference to disable TOTP MFA for an individual user.

Adding Advanced Security to a User Pool

After you create your user pool, you have access to **Advanced security** on the navigation bar in the Amazon Cognito console. You can turn the user pool advanced security features on, and customize the actions that are taken in response to different risks. Or you can use audit mode to gather metrics on detected risks without taking action. In audit mode, the advanced security features publish metrics to Amazon CloudWatch. See Viewing Advanced Security Metrics (p. 333).

Note
Additional pricing applies for Amazon Cognito advanced security features. See the Amazon Cognito pricing page.

Topics

Prerequisites

Before you begin, you need:

- A user pool with an app client. For more information, see Getting Started with User Pools (p. 20).
- Set multi-factor authentication (MFA) to **Optional** in the Amazon Cognito console to use the risk-based adaptive authentication feature. For more information, see Adding Multi-Factor Authentication (MFA) to a User Pool (p. 323).

- If you're using email for notifications, go to the Amazon SES console to configure and verify an email address or domain to use with your notification emails. For more information about Amazon SES, see Verifying Identities in Amazon SES.

Configuring Advanced Security Features

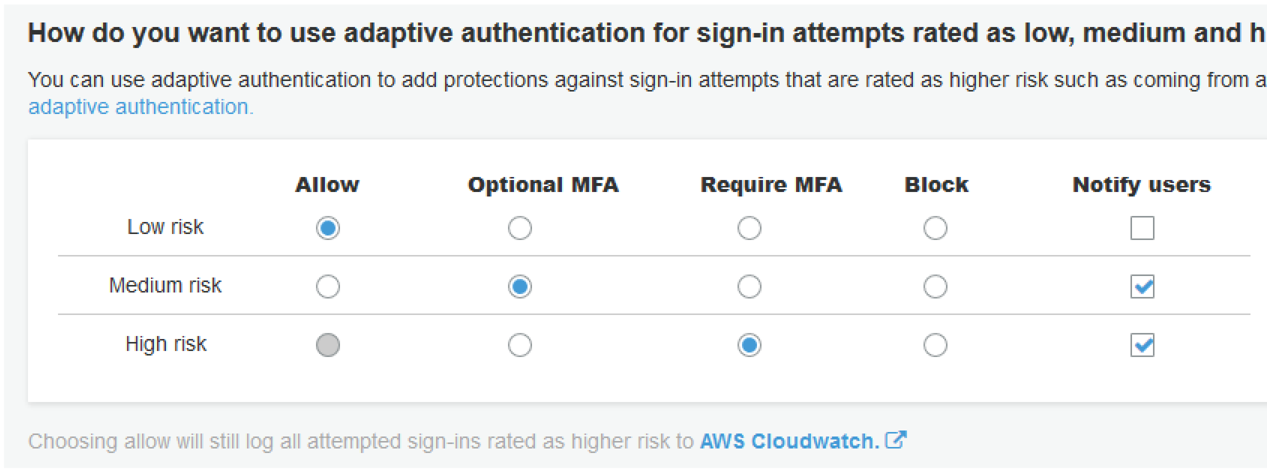
To configure advanced security for a user pool

1. From the left navigation bar, choose **Advanced security**.
2. For **Do you want to enable advanced security features for this user pool?**, choose **Yes** to enable advanced security. Or choose **Audit only** to gather information, and send user pool data to Amazon CloudWatch.

   We recommend keeping the advanced security features in audit mode for two weeks before enabling actions. This allows Amazon Cognito to learn the usage patterns of your app users.
3. From the drop-down list, choose **What app client do you want to customize settings for?**. The default is to leave your settings as global for all app clients.
4. For **Which action do you want to take with the compromised credentials?**, choose **Allow** or **Block use**.
5. Choose **Customize when compromised credentials are blocked** to select which events should trigger compromised credentials checks:

   - Sign-in
   - Sign-up
   - Password change
6. Choose how to respond to malicious sign-in attempts under **How do you want to use adaptive authentication for sign-in attempts rated as low, medium and high risk?**. You can allow or block the sign-in attempt, or require additional challenges before allowing the sign-in.

   To send email notifications when anomalous sign-in attempts are detected, choose **Notify users** .



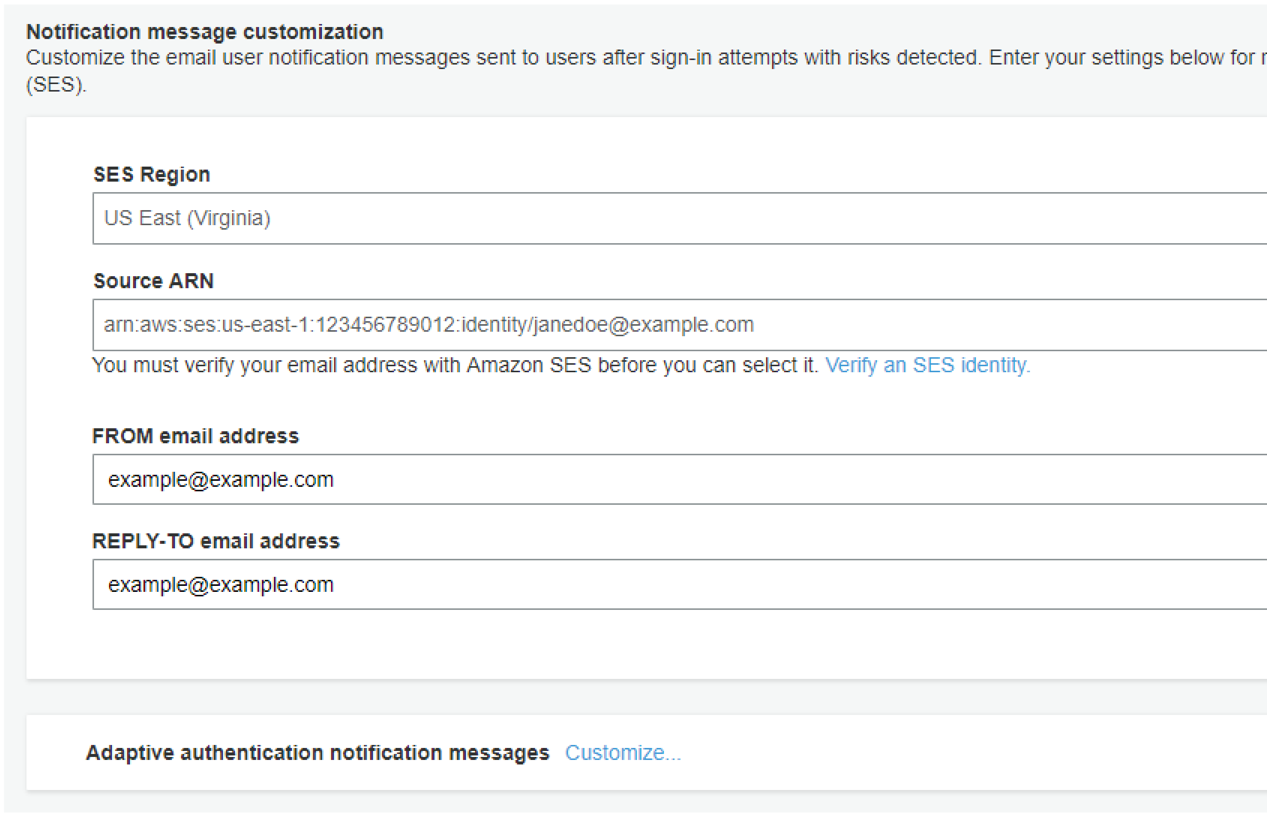
How do you want to use adaptive authentication for sign-in attempts rated as low, medium and h

You can use adaptive authentication to add protections against sign-in attempts that are rated as higher risk such as coming from a
adaptive authentication.

	Allow	Optional MFA	Require MFA	Block	Notify users
Low risk	●	○	○	○	☐
Medium risk	○	●	○	○	✓
High risk	○	○	●	○	✓

Choosing allow will still log all attempted sign-ins rated as higher risk to **AWS Cloudwatch.**

7. If you chose **Notify users** in the previous step, then you can customize the email notification messages by using the **Notification message customization** form:



Notification message customization
Customize the email user notification messages sent to users after sign-in attempts with risks detected. Enter your settings below for r
(SES).

SES Region
US East (Virginia)

Source ARN
arn:aws:ses:us-east-1:123456789012:identity/janedoe@example.com
You must verify your email address with Amazon SES before you can select it. Verify an SES identity.

FROM email address
example@example.com

REPLY-TO email address
example@example.com

Adaptive authentication notification messages Customize...

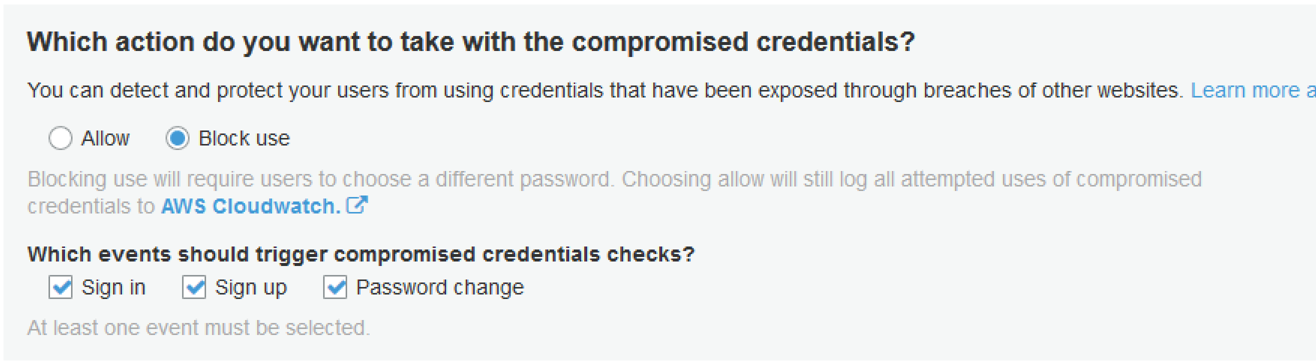
8. Choose **Customize** to customize adaptive authentication notifications with both HTML and plaintext email versions. To learn more about email templates, see Message Templates (p. 172).
9. Type any IP addresses that you want to **Always allow**, or **Always block**, regardless of the advanced security risk assessment. Specify the IP address ranges in CIDR notation (such as 192.168.100.0/24).
10. Choose **Save changes**.

Checking for Compromised Credentials

Amazon Cognito can detect if a user's credentials (user name and password) have been compromised elsewhere. This can happen when users reuse credentials at more than one site, or when they use passwords that are easy to guess.

From the **Advanced security** page in the Amazon Cognito console, you can choose whether to allow, or block the user if compromised credentials are detected. Blocking requires users to choose another password. Choosing **Allow** publishes all attempted uses of compromised credentials to Amazon CloudWatch. For more information, see Viewing Advanced Security Metrics (p. 333).

You can also choose whether Amazon Cognito checks for compromised credentials during sign-in, sign-up, and password changes.



Which action do you want to take with the compromised credentials?

You can detect and protect your users from using credentials that have been exposed through breaches of other websites. Learn more a

Allow [✓ Block use]

Blocking use will require users to choose a different password. Choosing allow will still log all attempted uses of compromised credentials to **AWS Cloudwatch.**

Which events should trigger compromised credentials checks?

✓ Sign in ✓ Sign up ✓ Password change

At least one event must be selected.

> **Note**
> Currently, Amazon Cognito doesn't check for compromised credentials for sign-in operations with Secure Remote Password (SRP) flow, which doesn't send the password during sign-in. Sign-ins that use the AdminInitiateAuth API with ADMIN_NO_SRP_AUTH flow and the InitiateAuth API with USER_PASSWORD_AUTH flow are checked for compromised credentials.

To add compromised credentials protections to your user pool, see Adding Advanced Security to a User Pool (p. 327).

Using Adaptive Authentication

With adaptive authentication you can configure your user pool to block suspicious sign-ins or add second factor authentication in response to an increased risk level. For each sign-in attempt, Amazon Cognito generates a risk score for how likely the sign-in request is to be from a compromised source. This risk score is based on many factors, including whether it detects a new device, user location, or IP address. Adaptive Authentication adds MFA based on risk level for users who don't have an MFA type enabled at the user level. When an MFA type is enabled at the user level, those users will always receive the second factor challenge during authentication regardless of how you configured adaptive authentication.

Amazon Cognito publishes sign-in attempts, their risk levels, and failed challenges to Amazon CloudWatch. For more information, see Viewing Advanced Security Metrics (p. 333).

To add adaptive authentication to your user pool, see Adding Advanced Security to a User Pool (p. 327).

Topics

Adaptive Authentication Overview

From the **Advanced security** page in the Amazon Cognito console, you can choose settings for adaptive authentication, including what actions to take at different risk levels and customization of notification messages to users.

For each risk level, you can choose from the following options:

Option	Action
Allow	The sign-in attempt is allowed without an additional factor.

Option	Action
Optional MFA	Users who have a second factor such as phone number* for SMS or a TOTP software token configured are required to complete a second factor challenge to sign in. Users who don't have a second factor configured are allowed to sign in without an additional factor.
Require MFA	Users who have a second factor such as phone number* for SMS or a TOTP software token configured are required to complete a second factor challenge to sign in. Users who don't have a second factor configured are blocked from signing in.
Block	All sign-in attempts at that risk level are blocked.

Note
*Phone numbers don't need to be verified to be used for SMS as a second authentication factor.

Creating a Device Fingerprint

When you call the Amazon Cognito authentication APIs, such as AdminInitiateAuth or AdminRespondToAuthChallenge from your server, you need to pass the source IP of the user in the ContextData parameter. This is in addition to your server name, server path, and encoded-device fingerprinting data, which are collected by using the Amazon Cognito context data collection library.

For information on enabling advanced security from your web or mobile app, see Enabling User Pool Advanced Security from Your App (p. 335).

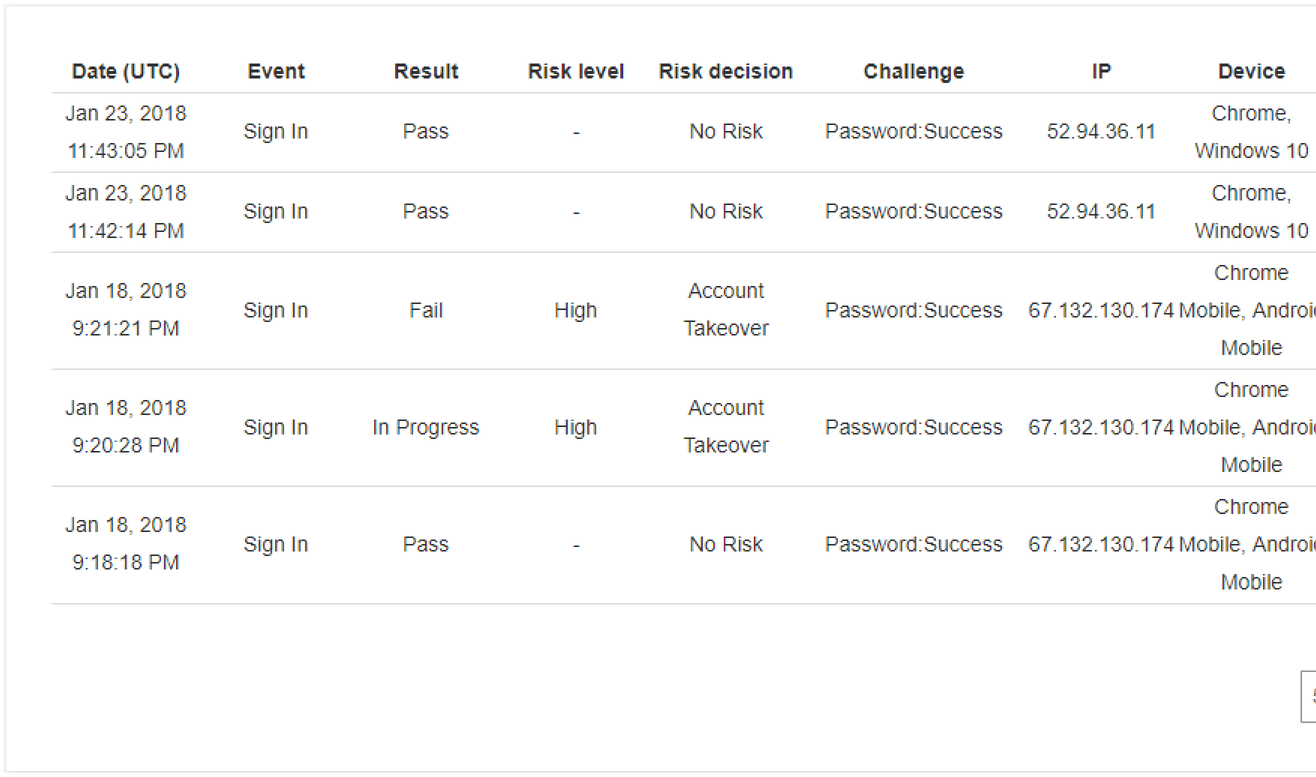
Collect user context data from the client side when your app is calling Amazon Cognito from your server. The following is an example that uses the JavaScript SDK method `getData`.

```
var encodedData = AmazonCognitoAdvancedSecurityData.getData(username, userPoolId,
 clientId);
```

We recommend incorporating the latest Amazon Cognito SDK into your app. This enables adaptive authentication to collect device fingerprinting information—such as device ID, model, and time zone. For more information about Amazon Cognito SDKs, see Install a user pool SDK.

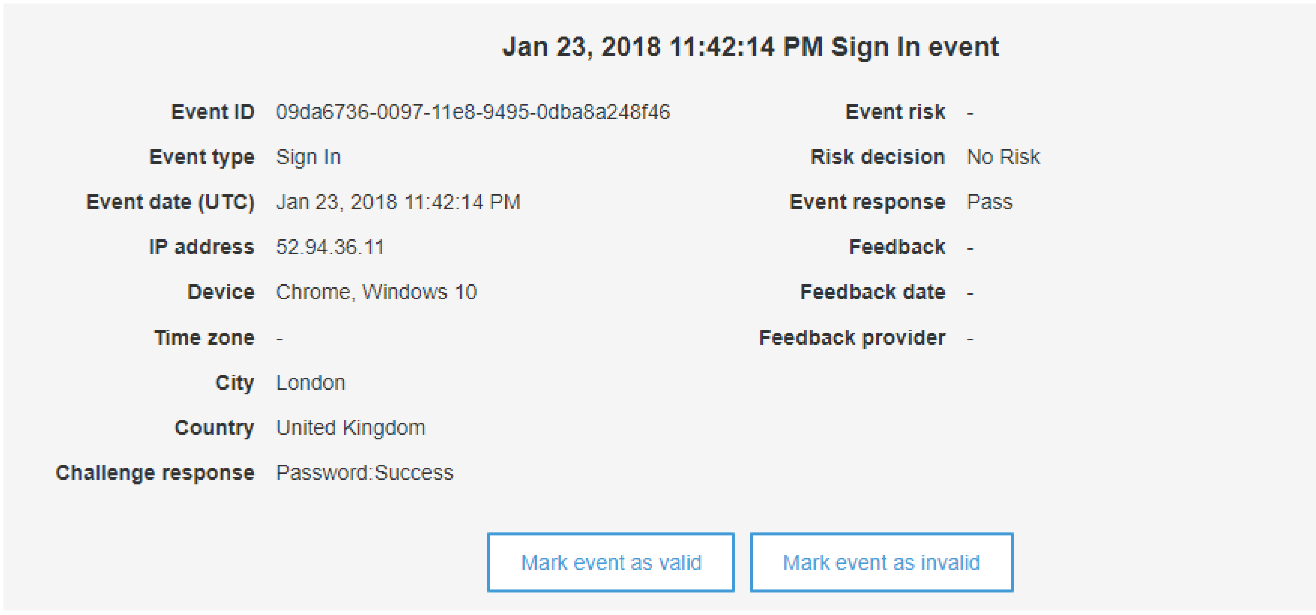
Viewing User Event History

You can choose a user in the Amazon Cognito console from **Users and groups** to see the sign-in history for that user. User event history is retained for two years by Amazon Cognito.



Date (UTC)	Event	Result	Risk level	Risk decision	Challenge	IP	Device
Jan 23, 2018 11:43:05 PM	Sign In	Pass	-	No Risk	Password:Success	52.94.36.11	Chrome, Windows 10
Jan 23, 2018 11:42:14 PM	Sign In	Pass	-	No Risk	Password:Success	52.94.36.11	Chrome, Windows 10
Jan 18, 2018 9:21:21 PM	Sign In	Fail	High	Account Takeover	Password:Success	67.132.130.174	Chrome Mobile, Androi Mobile
Jan 18, 2018 9:20:28 PM	Sign In	In Progress	High	Account Takeover	Password:Success	67.132.130.174	Chrome Mobile, Androi Mobile
Jan 18, 2018 9:18:18 PM	Sign In	Pass	-	No Risk	Password:Success	67.132.130.174	Chrome Mobile, Androi Mobile

Each sign-in event has an event ID, context data such as location, device details, and risk detection results associated with it.



Jan 23, 2018 11:42:14 PM Sign In event

Event ID	09da6736-0097-11e8-9495-0dba8a248f46	**Event risk**	-
Event type	Sign In	**Risk decision**	No Risk
Event date (UTC)	Jan 23, 2018 11:42:14 PM	**Event response**	Pass
IP address	52.94.36.11	**Feedback**	-
Device	Chrome, Windows 10	**Feedback date**	-
Time zone	-	**Feedback provider**	-
City	London		
Country	United Kingdom		
Challenge response	Password:Success		

You can correlate the event id to the token issued as well. The issued token, such as the ID token and access token, carries this event ID in its payload. Even using the refresh token persists the original event ID. The original event ID can be traced back to the event ID of the sign-in event that resulted in issuing the Amazon Cognito tokens. This enables you to trace usage of a token within your system to a particular authentication event.

Providing Event Feedback

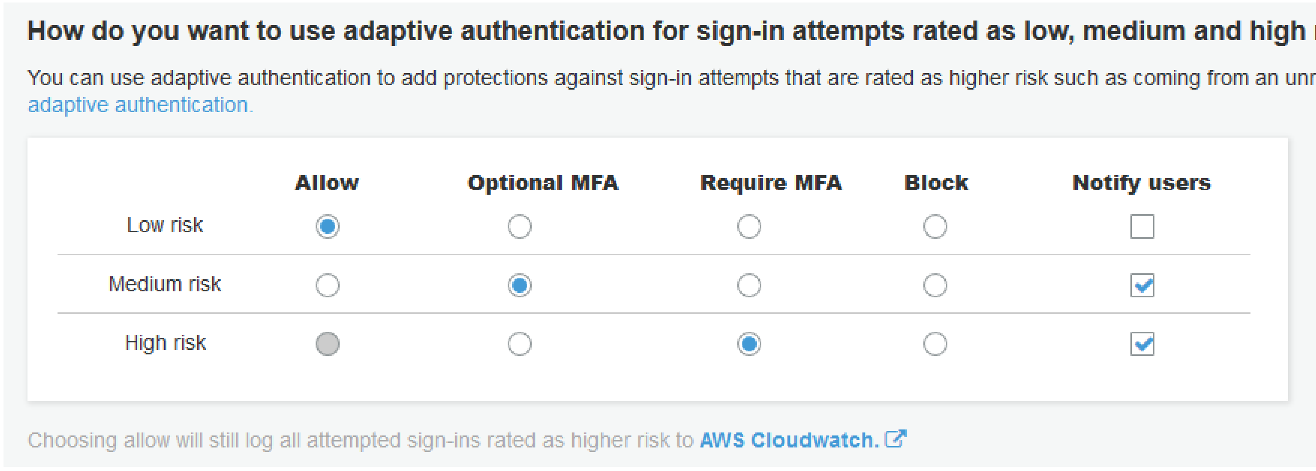
Event feedback affects risk evaluation in real time, and improves the risk evaluation algorithm over time. You can provide feedback on the validity of sign-in attempts through the Amazon Cognito console and API operations.

In the console, on the **Users and groups** tab, the sign-in history is listed, and if you click an entry, you can mark the event as valid or invalid. You can also provide feedback through the user pool API method AdminUpdateAuthEventFeedback, and through the AWS CLI command admin-update-auth-event-feedback.

Sending Notification Messages

With advanced security protections, Amazon Cognito can notify your users of sign-in attempts, prompt them to click links to indicate if the sign-in was valid or invalid, and use their feedback to improve the risk detection accuracy for your user pool.

In the **How do you want to use adaptive authentication for sign-in attempts rated as low, medium and high risk?** section choose **Notify Users** for the low, medium, and high-risk cases.



How do you want to use adaptive authentication for sign-in attempts rated as low, medium and high

You can use adaptive authentication to add protections against sign-in attempts that are rated as higher risk such as coming from an unr
adaptive authentication.

	Allow	Optional MFA	Require MFA	Block	Notify users
Low risk	●	○	○	○	☐
Medium risk	○	●	○	○	✓
High risk	○	○	●	○	✓

Choosing allow will still log all attempted sign-ins rated as higher risk to **AWS Cloudwatch.**

You can customize notification emails, and provide both plaintext and HTML versions. Choose **Customize** from **Adaptive authentication notification messages** to customize your email notifications. To learn more about email templates, see Message Templates (p. 172).

Viewing Advanced Security Metrics

Amazon Cognito publishes metrics for advanced security features to your account in Amazon CloudWatch. The advanced security metrics are grouped together by risk level and also by request level.

To view metrics by using the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. Choose Amazon Cognito.
4. Choose a group of aggregated metrics, such as **By Risk Classification**.
5. The **All metrics** tab displays all metrics for that choice. You can do the following:
   - To sort the table, use the column heading.
   - To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.

- To filter by resource, choose the resource ID, and then choose **Add to search**.
- To filter by metric, choose the metric name, and then choose **Add to search**.

Metric	**Description**	**Metric Dimensions**
CompromisedCredentialRisk	Requests where Amazon Cognito detected compromised credentials.	Operation: The type of operation `PasswordChange`, `SignIn`, or `SignUp` are the only dimensions. UserPoolId: The identifier of the user pool. RiskLevel: high (default), medium, or low.
AccountTakeOverRisk	Requests where Amazon Cognito detected account take-over risk.	Operation: The type of operation `PasswordChange`, `SignIn`, or `SignUp` are the only dimensions. UserPoolId: The identifier of the user pool. RiskLevel: high, medium, or low.
OverrideBlock	Requests that Amazon Cognito blocked because of the configuration provided by the developer.	Operation: The type of operation `PasswordChange`, `SignIn`, or `SignUp` are the only dimensions. UserPoolId: The identifier of the user pool. RiskLevel: high, medium, or low.
Risk	Requests that Amazon Cognito marked as risky.	Operation: The type of operation such as `PasswordChange`, `SignIn`, or `SignUp`. UserPoolId: The identifier of the user pool.
NoRisk	Requests where Amazon Cognito did not identify any risk.	Operation: The type of operation such as `PasswordChange`, `SignIn`, or `SignUp`. UserPoolId: The identifier of the user pool.

Amazon Cognito offers you two predefined groups of metrics for ready analysis in CloudWatch. **By Risk Classification** identifies the granularity of the risk level for requests that Amazon Cognito identified as risky, and the **By Request Classification** reflects metrics aggregated by request level.

Aggregated Metrics Group	Description
By Risk Classification	Requests that Amazon Cognito identified as risky.
By Request Classification	Metrics aggregated by request.

Enabling User Pool Advanced Security from Your App

After you configure the advanced security features for your user pool, you need to enable them in your web or mobile app.

To use advanced security with JavaScript

1. You might need to update your Amazon Cognito SDK to the latest version. For more information about Amazon Cognito SDKs, see Install a user pool SDK.
2. To use the Auth SDK to enable the hosted UI, see the CognitoAuth JavaScript sample app.
3. Set `AdvancedSecurityDataCollectionFlag` to true. Also, set `UserPoolId` to your user pool ID.
4. In your application replace <region> with your AWS Region such as us-east-1, and add this source reference to your JavaScript file:

   ```
   <script src="https://amazon-cognito-assets.<region>.amazoncognito.com/amazon-cognito-
   advanced-security-data.min.js"></script>
   ```

   For more information, see Sample for Amazon Cognito Auth SDK for JavaScript.

To use advanced security with Android

1. You might need to update your Amazon Cognito SDK to the latest version. For more information about Amazon Cognito SDKs, see Install a user pool SDK.
2. To use the Auth SDK to enable the hosted UI, see the CognitoAuth Android sample app.
3. Use `{ transitive = true; }` while importing `aws-android-sdk-cognitoauth` through Maven in Gradle.

   Include this as a dependency in your build.gradle file:

   ```
   compile "com.amazonaws:aws-android-sdk-cognitoidentityprovider-asf:1.0.0"
   ```

   For more information, see AWS SDK for Android - Amazon Cognito Identity Provider ASF.

To use advanced security with iOS

1. You might need to update your Amazon Cognito SDK to the latest version. For more information about Amazon Cognito SDKs, see Install a user pool SDK.
2. To use the Auth SDK to enable the hosted UI, see the CognitoAuth iOS sample app.
3. To configure the Auth SDK by using Info.plist, add the PoolIdForEnablingASF key to your Amazon Cognito user pool configuration, and set it to your user pool ID.

   To configure the Auth SDK using AWSCognitoAuthConfiguration, use this initializer and specify your user pool ID as userPoolIdForEnablingASF.

   For more information, see AWSCognitoIdentityProviderASF.

AWS managed policies for Amazon Cognito

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

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 and user pools. To view the permissions for this policy, see AmazonCognitoPowerUser.
- `AmazonCognitoReadOnly` - Permissions for read-only access to your identity pools and user pools. To view the permissions for this policy, see AmazonCognitoReadOnly.
- `AmazonCognitoDeveloperAuthenticatedIdentities` - Permissions for your authentication system to integrate with Amazon Cognito. To view the permissions for this policy, see AmazonCognitoDeveloperAuthenticatedIdentities.

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.

Cognito updates to AWS managed policies

View details about updates to AWS managed policies for Cognito since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Cognito Document history page.

Change	Description	Date
`AmazonCognitoPowerUser` – Update to an existing policy	Added a new permission to allow Amazon Cognito to call Amazon Simple Notification Service's `GetSMSSandboxAccountStatus` operation. This change allows Amazon Cognito user pools user pools to decide if you need to graduate out of the Amazon Simple Notification Service sandbox in order to send messages to all end users through user pools.	June 1, 2021
Cognito started tracking changes	Cognito started tracking changes for its AWS managed policies.	March 1, 2021

Tagging Amazon Cognito Resources

A *tag* is a metadata label that you assign or that AWS assigns to an AWS resource. Each tag consists of a *key* and a *value*. For tags that you assign, you define the key and value. For example, you might define the key as `stage` and the value for one resource as `test`.

Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you could assign the same tag to an Amazon Cognito user pool that you assign to an Amazon DynamoDB table.
- Track your AWS costs. You activate these tags on the AWS Billing and Cost Management dashboard. AWS uses the tags to categorize your costs and deliver a monthly cost allocation report to you. For more information, see Use Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.
- Control access to your resources based on the tags that are assigned to them. You control access by specifying tag keys and values in the conditions for an AWS Identity and Access Management (IAM) policy. For example, you could allow an IAM user to update a user pool, but only if the user pool has an `owner` tag with a value of that user's name. For more information, see Controlling Access Using Tags in the *IAM User Guide*.

You can use the AWS CLI or the Amazon Cognito API to add, edit, or delete tags for user pools and identity pools. For user pools specifically, you can also manage tags by using the Amazon Cognito console.

For tips on using tags, see the AWS Tagging Strategies post on the *AWS Answers* blog.

The following sections provide more information about tags for Amazon Cognito.

Supported Resources in Amazon Cognito

The following resources in Amazon Cognito support tagging:

- User pools
- Identity pools

Tag Restrictions

The following basic restrictions apply to tags on Amazon Cognito resources:

- Maximum number of tags that you can assign to a resource – 50
- Maximum key length – 128 Unicode characters
- Maximum value length – 256 Unicode characters
- Valid characters for key and value – a-z, A-Z, 0-9, space, and the following characters: _ . : / = + - and @
- Keys and values are case sensitive
- Don't use `aws:` as a prefix for keys; it's reserved for AWS use

Managing Tags by Using the Amazon Cognito Console

You can use the Amazon Cognito console to manage the tags that are assigned to your user pools.

For identity pools, the console does not include tagging features, so you must manage tags programmatically. For example, you can use the AWS CLI.

To add tags to a user pool

1. Sign in to the AWS Management Console and open the Amazon Cognito console at https://console.aws.amazon.com/cognito.
2. Choose **Manage User Pools**.
3. On the **Your User Pools** page, choose the user pool that you want to add tags to.
4. In the navigation menu on the left, choose **Tags**.
5. Unless your user pool already has tags, choose **Add tag** to add the first one.
6. Specify values for **Tag Key** and **Tag Value**.
7. For each additional tag that you want to add, choose **Add another tag**.
8. When you are finished adding tags, choose **Save changes.**

On the **Tags** page, you can also edit the keys and values of any existing tags. To remove a tag, choose the *x* in the top-right corner of the tag.

AWS CLI Examples

The AWS CLI provides commands that you can use to manage the tags that you assign to your Amazon Cognito user pools and identity pools.

Assigning tags

Use the following commands to assign tags to your existing user pools and identity pools.

Example `tag-resource` command for user pools

Assign tags to a user pool by using `tag-resource` within the `cognito-idp` set of commands:

```
$ aws cognito-idp tag-resource \
> --resource-arn user-pool-arn \
> --tags Stage=Test
```

This command includes the following parameters:

- `resource-arn` – The Amazon Resource Name (ARN) of the user pool that you are applying tags to. To look up the ARN, choose the user pool in the Amazon Cognito console, and view the **Pool ARN** value on the **General settings** tab.
- `tags` – The key-value pairs of the tags.

To assign multiple tags at once, specify them in a comma-separated list:

```
$ aws cognito-idp tag-resource \
> --resource-arn user-pool-arn \
> --tags Stage=Test,CostCenter=80432,Owner=SysEng
```

Example `tag-resource` command for identity pools

Assign tags to an identity pool by using `tag-resource` within the `cognito-identity` set of commands:

```
$ aws cognito-identity tag-resource \
> --resource-arn identity-pool-arn \
> --tags Stage=Test
```

This command includes the following parameters:

- `resource-arn` – The Amazon Resource Name (ARN) of the identity pool that you are applying tags to. To look up the ARN, choose the identity pool in the Amazon Cognito console, and choose **Edit identity pool**. Then, at **Identity pool ID**, choose **Show ARN**.
- `tags` – The key-value pairs of the tags.

To assign multiple tags at once, specify them in a comma-separated list:

```
$ aws cognito-identity tag-resource \
> --resource-arn identity-pool-arn \
> --tags Stage=Test,CostCenter=80432,Owner=SysEng
```

Viewing tags

Use the following commands to view the tags that you have assigned to your user pools and identity pools.

Example `list-tags-for-resource` command for user pools

View the tags that are assigned to a user pool by using `list-tags-for-resource` within the `cognito-idp` set of commands:

```
$ aws cognito-idp list-tags-for-resource --resource-arn user-pool-arn
```

Example `list-tags-for-resource` command for identity pools

View the tags that are assigned to an identity pool by using `list-tags-for-resource` within the `cognito-identity` set of commands:

```
$ aws cognito-identity list-tags-for-resource --resource-arn identity-pool-arn
```

Removing tags

Use the following commands to remove tags from your user pools and identity pools.

Example `untag-resource` command for user pools

Remove tags from a user pool by using `untag-resource` within the `cognito-idp` set of commands:

```
$ aws cognito-idp untag-resource \
> --resource-arn user-pool-arn \
> --tag-keys Stage CostCenter Owner
```

For the `--tag-keys` parameter, specify one or more tag keys, and do not include the tag values.

Example `untag-resource` command for identity pools

Remove tags from an identity pool by using `untag-resource` within the `cognito-identity` set of commands:

```
$ aws cognito-identity untag-resource \
> --resource-arn identity-pool-arn \
> --tag-keys Stage CostCenter Owner
```

For the `--tag-keys` parameter, specify one or more tag keys, and do not include the tag values.

> **Important**
> After you delete a user or identity pool, tags related to the deleted pool can still appear in the console or API calls up to thirty days after the deletion.

Applying tags when you create resources

Use the following commands to assign tags at the moment you create a user pool or identity pool.

Example `create-user-pool` command with tags

When you create a user pool by using the `create-user-pool` command, you can specify tags with the `--user-pool-tags` parameter:

```
$ aws cognito-idp create-user-pool \
> --pool-name user-pool-name \
> --user-pool-tags Stage=Test,CostCenter=80432,Owner=SysEng
```

Example `create-identity-pool` command with tags

When you create an identity pool by using the `create-identity-pool` command, you can specify tags with the `--identity-pool-tags` parameter:

```
$ aws cognito-identity create-identity-pool \
> --identity-pool-name identity-pool-name \
> --allow-unauthenticated-identities \
> --identity-pool-tags Stage=Test,CostCenter=80432,Owner=SysEng
```

Managing Tags by Using the Amazon Cognito API

You can use the following actions in the Amazon Cognito API to manage the tags for your user pools and identity pools.

API Actions for User Pool Tags

Use the following API actions to assign, view, and remove tags for user pools.

- TagResource

- ListTagsForResource
- UntagResource
- CreateUserPool

API Actions for Identity Pool Tags

Use the following API actions to assign, view, and remove tags for identity pools.

- TagResource
- ListTagsForResource
- UntagResource
- CreateIdentityPool

Quotas in Amazon Cognito

Amazon Cognito limits the number of operations that you can perform in your account. Amazon Cognito also limits the number and size of Amazon Cognito resources.

Operation Quotas

Quota categorization

Amazon Cognito limits the number of operations, such as `InitiateAuth` or `RespondToAuthChallenge`, that you can use to perform certain user actions in your applications. These operations are grouped into categories of common use cases, such as `UserAuthentication` or `UserCreation`. For a list of categorized operations, see Category operations (p. 344). The categories make it easier to monitor usage and request quota increases.

Operation quotas are defined as the number of allowed requests per second (RPS) for all operations within a category. The Amazon Cognito user pools service applies quotas to all operations in each category. For example, the category `UserCreation`, includes four operations, `SignUp`, `ConfirmSignUp`, `AdminCreateUser`, and `AdminConfirmSignUp`. It's allocated with a combined quota of 30 RPS. Each operation within this category can call up to 30 RPS separately or combined if multiple operations take place at the same time.

Note
The category quota is enforced for each AWS account across all user pools in an account and region.

Operation special handling

The operation quotas are measured and enforced for the combined total requests at the category level, with the exception of the `AdminRespondToAuthChallenge` and the `RespondToAuthChallenge` operations, where special handling rules are applied.

The `UserAuthentication` category include four operations, `AdminInitiateAuth`, `InitiateAuth`, `AdminRespondToAuthChallenge`, and `RespondToAuthChallenge`. The `InitiateAuth` and `AdminInitiateAuth` operations are measured and enforced per category quota. The matching `RespondToAuthChallenge` and `AdminRespondToAuthChallenge` operations are subject to a separate quota that is three times the `UserAuthentication` category limit, to accommodate multiple authentication challenges set up in developers' apps. This elevated quota is sufficient to cover the large majority of use cases. Beyond the three-time per authentication call threshold, the excess rate will be counted towards the `UserAuthentication` category limit.

For example, if your quota for the `UserAuthentication` category is 80 RPS, you can call `RespondToAuthChallenge` or `AdminRespondToAuthChallenge` up to 240 RPS (80 RPS x 3). If the app is set up to have four rounds of challenge per authentication call and you have 70 sign-ins per second, then total `RespondToAuthChallenge` will be 280 RPS (70 x 4), which is 40 RPS above the quota. The extra 40 RPS will be added to 70 `InitiateAuth` calls. The total usage of `UserAuthentication` category is 110 RPS (40 + 70). This exceeds the category quota set at 80 RPS so this app would get throttled because it's 30 RPS above the category quota of 80 RPS.

Category operations

You can find the mappings between operations and their respective categories in the following table. All categories listed in this table have adjustable quotas, which can be increased upon customer requests. For more information, see API request rate quotas (p. 349). Adjustable quotas are applied at the account level. There are some category operations that are subject to more constrained quota rules with a lower limit threshold at the user pool level. You can find these categories with an asterisk in the table and their quotas in the note below the table.

> **Note**
> The rate limit for each category depends on Monthly Active Users (MAUs). The default limits support up to two million MAUs. It's not recommended to submit a limit increase request if you have less than one million MAUs.

Category operation quotas, are applied across all users in a user pool. There are also per-user quotas that apply to each user. For each user, you can make up to ten requests per second that are "read" operations including signing in or getting profile or device information. You can also make up to ten requests per second that are "write" operations, including updating profile information or MFA settings. You can't change per-user quotas.

Category name	**Operations**	**Description**	**Default quota** **(in requests per second)**	
`UserAuthentication`	• InitiateAuth • RespondToAuthChallenge • AdminInitiateAuth • AdminRespondToAuthChallenge	Operations that authenticate (sign-in) a user. These operations are subject to Operation special handling (p. 343).	120	
`UserCreation`	• SignUp • ConfirmSignUp • AdminCreateUser • AdminConfirmSignUp	Operations that create or confirm a native Amazon Cognito user. Native users are users who are created and verified directly by your Cognito user pools.	50	

Category name	Operations	Description	Default quota (in requests per second)	
`UserFederation`	Amazon Cognito managed operations to federate a user via a third party IdP into Cognito.	Operations that federate (authenticate) users with a third-party identity provider into your Cognito user pools.	25	
`UserAccountRecovery`	• ChangePassword • ConfirmForgotPassword • ForgotPassword • AdminResetUserPassword • AdminSetUserPassword	Operations that recover a user's account or change or update a user's password.	30	
`UserRead`	• AdminGetUser • GetUser	Operations that retrieve a user from your user pools.	120	
`UserUpdate`	• AdminAddUserToGroup • AdminDeleteUserAttributes • AdminUpdateUserAttributes • AdminDeleteUser • AdminDisableUser • AdminEnableUser • AdminLinkProviderForUser • AdminDisableProviderForUser • VerifyUserAttribute • DeleteUser • DeleteUserAttributes • UpdateUserAttributes • AdminUserGlobalSignOut • GlobalSignOut • AdminRemoveUserFromGroup	The operations that customers use to manage users and user attributes,	25	
`UserResourceRead`	• AdminGetDevice • AdminListGroupsForUser • AdminListDevices • GetDevice • ListDevices • GetUserAttributeVerificationCode • ResendConfirmationCode • AdminListUserAuthEvents	Operations that retrieve user resource information from Amazon Cognito such as device or group.	50	

Category name	**Operations**	**Description**	**Default quota (in requests per second)**	
`UserResourceUpdate`	• AdminForgetDevice • AdminUpdateAuthEventFeedback • AdminSetUserMFAPreference • AdminSetUserSettings • AdminUpdateDeviceStatus • UpdateDeviceStatus • UpdateAuthEventFeedback • ConfirmDevice • SetUserMFAPreference • SetUserSettings • VerifySoftwareToken • AssociateSoftwareToken • ForgetDevice	Operations that update user resource information, such as group.	25	
`UserList`	• ListUsers • ListUsersInGroup	Operations that return a list of users.	30	
`UserPoolRead`	• DescribeUserPool • ListUserPools	Operations that read your user pools.	15	
`UserPoolUpdate`	• CreateUserPool • UpdateUserPool • DeleteUserPool	Operations that create, update, and delete your user pools.	15	
`UserPoolResourceRead`	• DescribeIdentityProvider • DescribeResourceServer • DescribeUserImportJob • DescribeUserPoolDomain • GetCSVHeader • GetGroup • GetSigningCertificate • GetIdentityProviderByIdentifier • GetUserPoolMfaConfig • ListGroups • ListIdentityProviders • ListResourceServers • ListTagsForResource • ListUserImportJobs • DescribeRiskConfiguration • GetUICustomization	Operations that retrieve a resource from a user pool, such as group.*	20	

Category name	Operations	Description	Default quota (in requests per second)	
`UserPoolResourceUpdate`	• AddCustomAttributes • CreateGroup • CreateIdentityProvider • CreateResourceServer • CreateUserImportJob • CreateUserPoolDomain • DeleteGroup • DeleteIdentityProvider • DeleteResourceServer • DeleteUserPoolDomain • SetUserPoolMfaConfig • StartUserImportJob • StopUserImportJob • UpdateGroup • UpdateIdentityProvider • UpdateResourceServer • SetRiskConfiguration • SetUICustomization • TagResource • UntagResource	Operations that update resource information for a user pool, such as group.*	15	
`UserPoolClientRead`	• DescribeUserPoolClient • ListUserPoolClients	Operations that list your user pool clients.*	15	
`UserPoolClientUpdate`	• CreateUserPoolClient • DeleteUserPoolClient • UpdateUserPoolClient	Operations that create, update, and delete your user pool clients.*	15	
`UserToken`	• RevokeToken	Operations that create, update, and delete your tokens.	120	

*Any individual operation in this category has a constraint that prevents the operation from being called at a rate higher than 5 RPS for a single user pool.

Track quota usage

Each API category has both `CallCount` and `ThrottleCount` CloudWatch metrics which are applied at the account level. You can use `CallCount` to track the total number of calls customers made related to a category. You can use `ThrottleCount` to track the total number of throttled calls related to a category. You can use the `CallCount` and `ThottleCount` metrics with the `Sum` statistic to count the total number of calls in a category. For more information on the CloudWatch usage metrics, see CloudWatch usage metrics.

Utilization and usage are two terms that may help you understand service quota monitoring. Utilization is the percentage of a service quota in use. For example, if the quota value is 200 resources and 150 resources are in use, the utilization is 75%. Usage is the number of resources or operations in use for a service quota.

Tracking usage through CloudWatch metrics

You can track and collect Amazon Cognito user pools utilization metrics using CloudWatch. The CloudWatch dashboard will display metrics about every AWS service you use. You can use CloudWatch to create metric alarms. The alarms can be setup to send you notifications or make a change to a specific resource that you are monitoring. For more information on CloudWatch metrics, For more information on Service Quotas, see Track your CloudWatch usage metrics (p. 311).

Tracking utilization through Service Quotas metrics

Amazon Cognito user pools is integrated with Service Quotas, which is a browser-based interface that you can use to view and manage your service quota usage. In the Service Quotas console, you can look up the value of a specific quota, view monitoring information, request a quota increase, and setup CloudWatch alarms. After your account has been active a while, you can view a graph of your resource utilization.

For more information on viewing quotas in the Service Quotas console, see Viewing Service Quotas.

Identify quota requirements

To calculate quota requirements, determine how many active users will interact with your application in a specific time period. For example, if your application expects an average of 1 million active users to sign-in within an 8-hour period, then you would need to be able to authenticate an average of 35 users per second.

In addition, if you assume the average user session is two hours, and tokens are configured to expire after an hour, each user has to refresh their tokens once during this session. Then the required average quota for the UserAuthentication category to support this load is 70 RPS.

If you assume a peak-to-average ratio of 3:1 by accounting the variance of user sign-in frequency during the eight hour period, then you need the desired UserAuthentication quota of 200 RPS.

> **Note**
> If you call multiple operations for each user action, you need to sum up the individual operation call rates at the category level.

Optimize quotas

Follow one of the approached below to handle a peak call rate.

Retry the attempt after a back-off waiting period

You can catch the error at every API call, and then re-try the attempt after a back-off period. You can adjust the back-off algorithm according to business needs and load. Amazon SDKs have built-in retry logic. For more information on Amazon Tools and SDKs, see Tools to Build on AWS.

Use a external database for frequently updated attributes

If your application requires several calls to a user pool to read or write custom attributes, use external storage. You can use your preferred database to store custom attributes or use a cache layer to load a user profile during sign-in. You can reference this profile from the cache when needed instead of reloading the user profile from a user pool.

Validate JWT tokens on the client-side

Applications have to validate JWT tokens before trusting them. You can verify the signature and validity of tokens on the client side without sending API calls to a user pool. After the token is validated, you can trust claims in the token and use the claims instead of making more `getUser` API calls. For more information on JSON token verification, see Verifying a JSON Web Token.

API request rate quotas

Amazon Cognito limits the number of user pool operations that you can perform in your account. You can request an increase to the adjustable API request rate quotas in Amazon Cognito. To request a quota increase, you can use the Service Quotas console, use the Service limit increase form, use the `RequestServiceQuotaIncrease` API or the `ListAWSDefaultServiceQuotas` API.

- To request a quota increase using the Service Quotas console, see Requesting a API quota increase in the *Service Quotas User Guide*.
- If the quota is not yet available in Service Quotas, use the Service limit increase form.

Note
Only adjustable quotas can be increased.

Resource quotas

Resource quotas are quotas that limit the number and size of your resources. You can request an increase to the adjustable resource quotas in Amazon Cognito. To request a quota increase, you can use the Service Quotas console or use the Service limit increase form. To request a quota from the Service Quotas console, see Requesting a quota increase in the *Service Quotas User Guide*. If the quota is not yet available in Service Quotas, use the Service limit increase form.

Adjustable resource quotas

The following tables provide the adjustable resource quotas for Amazon Cognito. These quotas can be changed.

User Pools resource quotas

Resource	Quota
Maximum number of app clients per user pool	1,000
Maximum number of user pools per account	1,000
Maximum number of user import jobs per user pool	1,000
Maximum number of identity providers per user pool	300

Resource	Quota
Maximum number of resource servers per user pool	25
Maximum number of users per user pool	40,000,000

Identity pools (federated users) resource quotas

Resource	Quota
Maximum number of identity pools per account	1,000
Maximum Amazon Cognito user pool providers per identity pool	50

Sync resource quotas

Resource	Quota
Maximum number of datasets per identity	20
Maximum number of records per dataset	1,024
Maximum size of a single dataset	1 MB

Non-adjustable resource quotas

The following tables describe Amazon Cognito non-adjustable quotas. These quotas cannot be changed.

User Pools token validity quotas

Token	Quota
ID token	5 minutes – 1 day
Refresh token	1 hour – 3,650 days
Access token	5 minutes – 1 day

User Pools resource quotas

Resource	Quota
Maximum number of custom attributes per user pool	50
Maximum characters per attribute	2,048 bytes
Maximum characters in custom attribute name	20
Minimum and maximum characters in password	6 – 99

Resource	Quota
Maximum number of email messages sent daily per AWS account [1]	50
Maximum characters in email subject	140
Maximum characters in email message	20,000
Maximum characters in SMS verification message	140
Maximum characters in password	256
Maximum characters in identity provider name	40
Maximum identifiers per identity provider	50
Maximum number of identities linked to a user	5
Maximum callback URLs per app client	100
Maximum logout URLs per app client	100
Maximum number of scopes per resource server	100
Maximum number of scopes per app client	50
Maximum number of custom domains per account	4
Maximum number of groups that each user can belong to	100
Maximum number of groups per user pool	10,000

[1]This quota applies only if you are using the default email feature for an Amazon Cognito User Pool. To enable a higher email delivery volume, configure your user pool to use your Amazon SES email configuration. For more information, see Email Settings for Amazon Cognito User Pools (p. 143).

User Pools code validity resource quotas

Resource	Quota
Sign-up confirmation code	24 hours
User attribute verification code validity	24 hours
Multi-factor authentication code	3 minutes
Forgot password code	1 hour

Hosted UI quotas

Workflow name	Quota
Client credentials	150 requests per second

Identity pools (federated users) resource quotas

Resource	Quota
Maximum character length for identity pool name	128 bytes
Maximum character length for login provider name	2,048 bytes
Maximum number of identities per identity pool	Unlimited
Maximum number of identity providers for which role mappings can be specified	10
Maximum number of results from a single list or lookup call	60
Maximum number of role-based access control (RBAC) rules	25

Sync resource quotas

Resource	Quota
Maximum characters in dataset name	128 bytes
Minimum waiting time for a bulk publish after a successful request	24 hours

Amazon Cognito API References

Use the following links to go to the related API reference pages.

- Amazon Cognito User Pools API Reference
- Amazon Cognito Identity Pools (Federated Identities) API Reference
- Amazon Cognito Sync API Reference
- Amazon Cognito user pools Auth API Reference

Amazon Cognito User Pools API Reference

With Amazon Cognito user pools, you can enable your web and mobile app users to sign up and sign in. You can change passwords for authenticated users and initiate forgotten password flows for unauthenticated users. For more information, see User Pool Authentication Flow (p. 306) and Using Tokens with User Pools (p. 149).

For a complete user pool API reference see Amazon Cognito user pools API Reference.

Amazon Cognito user pools Auth API Reference

Once a domain has been configured for your user pool, Amazon Cognito hosts an authentication server where you can add sign-up and sign-in webpages to your app. For more information see Add an app to enable the hosted UI.

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 Auth SDKs for Android, iOS, and JavaScript that make it easier to form requests and interact with the server. Learn more about the Amazon Cognito SDKs.

For more information on the specification see OpenID Connect 1.0 and OAuth 2.0.

Topics

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 a browser. Web browsers include Chrome or Firefox. Android browsers include Custom Chrome Tab. iOS browsers include Safari View Control.

The authorization server requires HTTPS instead of HTTP as the protocol when accessing the authorization endpoint. For more information on the specification see Authorization Endpoint.

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 preregistered 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.

A redirect URI must have the following attributes:

- It must be an absolute URI.
- It must be pre-registered with a client.
- It can not include a fragment component.

See OAuth 2.0 - Redirection Endpoint.

Amazon Cognito requires `HTTPS` over `HTTP` except for `http://localhost` for testing purposes only.

App callback URLs such as `myapp://example` are also supported.

Required.

state

An opaque value that the clients adds to the initial request. The authorization server includes this value when redirecting 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.

- For social sign-in the valid values are **Facebook**, **Google**, **LoginWithAmazon**, and **SignInWithApple**.
- For Amazon Cognito user pools, the value is **COGNITO**.

- For other identity providers this would be the name you assigned to the IdP in your user pool.

Optional.

idp_identifier

The developer this parameter 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 that are 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 associated with the client, or it will be ignored at runtime.

If the client doesn't request any scopes, the authentication server uses all scopes that are 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 Requests with Positive Responses

Authorization Code Grant

Sample 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.

Note
The response returns a one time use code that is valid for five minutes.

```
HTTP/1.1 302 Found
L                           ocation: https://YOUR_APP/redirect_uri?
code=AUTHORIZATION_CODE&state=STATE
```

Authorization Code Grant with PKCE

Sample 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=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
```

Token grant without `openid` scope

Sample Request

```
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
```

Token grant with `openid` scope

Sample Request

```
                        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 Responses

The following are examples of negative responses:

- If `client_id` and `redirect_uri` are valid, but the request parameters have other problems (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, or malformed scope, the Amazon Cognito authorization server returns `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 returns `server_error` to client's `redirect_uri`. Do not display the HTTP 500 error 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
  ```
- When authenticating by federating to third-party identity providers, Cognito might experience connection issues such as the following:
  - If a connection timeout occurs while requesting token from the identity provider, the authentication server redirects the error to the client’s `redirect_uri` as follows:

    ```
    HTTP 1.1 302 Found Location: https://client_redirect_uri?
    error=invalid_request&error_description=Timeout+occurred+in+calling+IdP
    +token+endpoint
    ```
  - If a connection timeout occurs while calling jwks endpoint for `id_token` validation, the authentication server redirects the error to the client’s `redirect_uri` as follows:

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?
error=invalid_request&error_description=error_description=Timeout+in+calling
+jwks+uri
```

- When authenticating by federating to third-party identity providers, the providers may return error responses due to configuration errors or otherwise such as the following:
  - If an error response is received from other providers, the authentication server redirects the error to the client's `redirect_uri` as follows:

    ```
    HTTP 1.1 302 Found Location: https://client_redirect_uri?
    error=invalid_request&error_description=[IdP name]+Error+-+[status
    code]+error getting token
    ```
  - If an error response is received from Google, the authentication server redirects the error to the client's `redirect_uri` as follows: `HTTP 1.1 302 Found Location: https://client_redirect_uri?error=invalid_request&error_description=Google+Error+-+[status code]+[Google provided error code]`
- In the rare case where Cognito encounters an exception in the communication protocol while making any connection to an external identity provider, the authentication server redirects the error to the client's `redirect_uri` with either of the following messages:
  - `HTTP 1.1 302 Found Location: https://client_redirect_uri?error=invalid_request&error_description=Connection+reset`
  - `HTTP 1.1 302 Found Location: https://client_redirect_uri?error=invalid_request&error_description=Read+timed+out`

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.

For more information on the specification see Token Endpoint.

Request Parameters in Header

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)`.

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.

Note
The token endpoint returns `refresh_token` only when the `grant_type` is `authorization_code`.

code

Required if `grant_type` is `authorization_code`.

code_verifier

The proof key.

Required if `grant_type` is `authorization_code` and the authorization code was requested with PKCE.

Examples Requests with Positive Responses

Exchanging an Authorization Code for Tokens

Sample Request

```
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
                           }
```

Note
The token endpoint returns `refresh_token` only when the `grant_type` is `authorization_code`.

Exchanging Client Credentials for an Access Token

Sample Request

```
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
                        }
```

Exchanging an Authorization Code Grant with PKCE for Tokens

Sample Request

```
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
                                     }
```

Note
The token endpoint returns `refresh_token` only when the `grant_type` is `authorization_code`.

Exchanging a Refresh Token for Tokens

Sample Request

```
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
                                 }
```

Note
The token endpoint returns `refresh_token` only when the `grant_type` is `authorization_code`.

Examples of Negative Responses

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`.

USERINFO Endpoint

The `/oauth2/userInfo` endpoint returns information about the authenticated user.

GET /oauth2/userInfo

The user pool client makes requests to this endpoint directly and not through a browser.

For more information see UserInfo Endpoint in the OpenID Connect (OIDC) specification.

Topics

Request Parameters in Header

Authorization: Bearer `<access_token>`

Pass the access token using the authorization header field.

Required.

Sample Request

```
GET https://<your-user-pool-domain>/oauth2/userInfo
                Authorization: Bearer <access_token>
```

Sample Positive Response

```
HTTP/1.1 200 OK
                       Content-Type: application/json;charset=UTF-8
                       {
                          "sub": "248289761001",
                          "name": "Jane Doe",
                          "given_name": "Jane",
                          "family_name": "Doe",
                          "preferred_username": "j.doe",
                          "email": "janedoe@example.com"
                       }
```

For a list of OIDC claims see Standard Claims.

Sample Negative Responses

Invalid Request

```
HTTP/1.1 400 Bad Request
                WWW-Authenticate: error="invalid_request",
                  error_description="Bad OAuth2 request at UserInfo Endpoint"
```

invalid_request

The request is missing a required parameter, includes an unsupported parameter value, or is otherwise malformed.

Invalid Token

```
HTTP/1.1 401 Unauthorized
                WWW-Authenticate: error="invalid_token",
                  error_description="Access token is expired, disabled, or deleted, or the
 user has globally signed out."
```

invalid_token

The access token is expired, revoked, malformed, or invalid.

LOGIN Endpoint

The `/login` endpoint signs the user in. It loads the login page and presents the authentication options configured for the client to the user.

GET /login

The `/login` endpoint only supports `HTTPS GET`. The user pool client makes this request through a system browser. System browsers for JavaScript include Chrome or Firefox. Android browsers include Custom Chrome Tab. iOS browsers include Safari View Control.

Request Parameters

client_id

The app client ID for your app. To obtain an app client ID, register the app in the user pool. For more information, see Configuring a User Pool App Client (p. 176).

Required.

redirect_uri

The URI where the user is redirected after a successful authentication. It should be configured on `response_type` of the specified `client_id`.

Required.

response_type

The OAuth response type, which can be `code` for code grant flow and `token` for implicit flow.

Required.

state

An opaque value the client adds to the initial request. The value is then returned back to the client upon redirect.

This value must be used by the client to prevent CSRF attacks.

Optional but strongly recommended.

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 is 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 an `openid` scope is requested. The access token can only be used against Amazon Cognito user pools if an `aws.cognito.signin.user.admin` scope is requested. The `phone`, `email`, and `profile` scopes can only be requested if an `openid` scope is also requested. These scopes dictate the claims that go inside the ID token.

Optional.

Sample Request: Prompt the User to Sign in

This example displays the login screen.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/login?
                response_type=code&
                client_id=ad398u21ijw3s9w3939&
                redirect_uri=https://YOUR_APP/redirect_uri&
                state=STATE&
                scope=openid+profile+aws.cognito.signin.user.admin
```

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 Configuring a User Pool App Client (p. 176).

Required.

logout_uri

A sign-out URL that you registered for your client app. For more information, see Configuring a User Pool App Client (p. 32).

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=https://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
```

REVOCATION Endpoint

The `revocation` endpoint invalidates the provided token.

POST/oauth2/revoke

The `revocation` endpoint only supports `HTTPS POST`. The user pool client makes requests to this endpoint directly and not through the system browser.

Request Parameters in Header

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)`.

Content-Type

Must always be `'application/x-www-form-urlencoded'`.

Request Parameters in Body

token

The refresh token that the client wants to revoke. Access tokens issued from the refresh token are also revoked.

Required

Revocation request example

```
                                POST /revoke HTTP/1.1
                                         Host: https://auth-domain.auth.us-east-1.amazoncognito.com
                                         Accept: application/json
                                         Content-Type: application/x-www-form-urlencoded
                                         Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
                                         token=2YotnFZFEjr1zCsicMWpAA
```

Revocation error response

A successful response contains an empty body. The error response is a JSON object with an error field and possibly `error_description` field.

Endpoint errors

- HTTP 400 and error `invalid_request` is returned if the token is not present in the request or if the feature is disabled for the app client.
- HTTP 400 and error `unsupported_token_type` is returned if the token sent in the revocation request is not a refresh token
- HTTP 401 and error `invalid_client` is returned if the client credentials are invalid.
- HTTP 200 if the token has been revoked or if the client submitted an invalid token.

Amazon Cognito Identity Pools (Federated Identities) API Reference

With an Amazon Cognito identity pool, your web and mobile app users can obtain temporary, limited-privilege AWS credentials enabling them to access other AWS services.

For a complete identity pools (federated identities) API reference see Amazon Cognito API Reference.

Amazon Cognito Sync API Reference

Amazon Cognito Sync is an AWS service and client library that enables cross-device syncing of application-related user data.

For more information on Amazon Cognito Sync API Reference, see 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: 2021-1-15

  Amazon Cognito Federated Identities: 2014-06-30

  Amazon Cognito Sync: 2014-06-30

Change	Description	Date
RevokeToken API and Revocation Endpoint	You can use the RevokeToken operation to revoke a refresh token (p. 154) for a user.	June 10, 2021
Publication of guide markdown to GitHub	As with all of AWS documentation, this guide now has markdown available to review and comment on in at https://github.com/awsdocs/amazon-cognito-developer-guide.	March 23, 2021
Multi-tenant best practices	Best practices for multi-tenant applications were added to the documentation.	March 4, 2021
Attributes for access control	Amazon Cognito Identity Pools provide attributes for access control (AFAC) as a way for customers to grant users access to AWS resources. Authorization can be done based on users' attributes from the identity provider which they used to federate with Amazon Cognito.	January 15, 2021
Custom SMS Sender Lambda Trigger and Custom Email Sender Lambda Trigger	The Custom SMS Sender Lambda Trigger and Custom Email Sender Lambda Trigger allow you to enable a third-party provider to send email and SMS notifications to your users from within your Lambda function code.	November 2020
Amazon Cognito token updates	Updated expiration information was added to Access, ID, and Refresh tokens.	October 29, 2020

Change	Description	Date
Amazon Cognito Service Quotas	Service Quotas are available for Amazon Cognito category quotas. You can use the Service Quotas console to view quota usage, request a quota increase, and create CloudWatch alarms to monitor your quota usage. As part of this change the Available CloudWatch Metrics for Amazon Cognito User Pools section was updated to reflect the new information. The new section name is: Tracking quotas and usage in CloudWatch and Service Quotas	October 29, 2020
Amazon Cognito quota categorization	Quota categories are available to help you monitor quota usage and request an increase. The quotas are grouped into categories based on common use cases.	August 17, 2020
Amazon Cognito Pinpoint document updates	New service-linked role was added. Instructions were updated on "Using Amazon Pinpoint Analytics with Amazon Cognito User Pools".	May 13, 2020
Amazon Cognito supported in US AWS GovCLoud	Amazon Cognito is now supported in the AWS GovCloud (US) Region.	May 13, 2020
New Amazon Cognito dedicated security chapter	The Security chapter can help your organization get in-depth information about both the built-in and the configurable security of AWS services. Our new chapters provide information about the security of the cloud and in the cloud.	April 30, 2020
Amazon Cognito Identity Pools now supports Sign in with Apple	Sign in with Apple is available in all regions where Amazon Cognito operates, except cn-north-1 region.	April 7, 2020
New Facebook API Versioning	Added version selection to Facebook API.	April 3, 2020
Username case insensitivity update	Added recommendation about enabling username case insensitivity (p. 20) before creating a user pool.	February 11, 2020

Change	Description	Date
New information about AWS Amplify	Added information about integrating Amazon Cognito (p. 15) with your web or mobile app by using AWS Amplify SDKs and libraries. Removed information about using the Amazon Cognito SDKs that preceded AWS Amplify.	November 22, 2019
New attribute for user pool triggers	Amazon Cognito now includes a `clientMetadata` parameter in the event information that it passes to the AWS Lambda functions for most user pool triggers. You can use this parameter to enhance your custom authentication workflow with additional data.	October 4, 2019
Updated limit	The throttling limit for the ListUsers API action is updated. For more information, see Quotas in Amazon Cognito (p. 343).	June 25, 2019
New limit	The soft limits for user pools now include a limit for the number of users. For more information, see Quotas in Amazon Cognito (p. 343).	June 17, 2019
Amazon SES email settings for Amazon Cognito user pools	You can configure a user pool so that Amazon Cognito emails your users by using your Amazon SES configuration. This setting allows Amazon Cognito to send email with a higher delivery volume than is otherwise possible. For more information, see Email Settings for Amazon Cognito User Pools (p. 143).	April 8, 2019
Tagging support	Added information about tagging Amazon Cognito resources (p. 338).	March 26, 2019
Change the certificate for a custom domain	If you use a custom domain to host the Amazon Cognito hosted UI, you can change the SSL certificate for this domain as needed. For more information, see Changing the SSL Certificate for Your Custom Domain (p. 39).	December 19, 2018

Change	Description	Date
New limit	A new limit is added for the maximum number of groups that each user can belong to. For more information, see Quotas in Amazon Cognito (p. 343).	December 14, 2018
Updated limits	The soft limits for user pools are updated. For more information, see Quotas in Amazon Cognito (p. 343).	December 11, 2018
Documentation update for verifying email addresses and phone numbers	Added information about configuring your user pool to require email or phone verification when a user signs up in your app. For more information, see Verifying Contact Information at Sign-Up (p. 118).	November 20, 2018
Documentation update for testing emails	Added guidance for initiating emails from Amazon Cognito while you test your app. For more information, see Sending Emails While Testing Your App (p. 123).	November 13, 2018
Amazon Cognito Advanced Security	Added new security features to enable developers to protect their apps and users from malicious bots, secure user accounts against compromised credentials, and automatically adjust the challenges required to sign in based on the calculated risk of the sign in attempt.	June 14, 2018
Custom Domains for Amazon Cognito Hosted UI	Allow developers to use their own fully custom domain for the hosted UI in Amazon Cognito User Pools.	June 4, 2018
Amazon Cognito User Pools OIDC Identity Provider	Added user pool sign-in through an OpenID Connect (OIDC) identity provider such as Salesforce or Ping Identity.	May 17, 2018
Amazon Cognito Developer Guide Update	Added top level "What is Amazon Cognito" and "Getting Started with Amazon Cognito". Also added common scenarios and reorganized the user pools TOC. Added a new "Getting Started with Amazon Cognito user pools" section.	April 6, 2018

Change	Description	Date
Amazon Cognito Lambda Migration Trigger	Added pages covering the Lambda Migration Trigger feature	February 8, 2018
Amazon Cognito Advanced Security Beta	Added new security features to enable developers to protect their apps and users from malicious bots, secure user accounts against credentials in the wild that have been compromised elsewhere on the internet, and automatically adjust the challenges required to sign in based on the calculated risk of the sign in attempt.	November 28, 2017
Amazon Pinpoint integration	Added the ability to use Amazon Pinpoint to provide analytics for your Amazon Cognito User Pools apps and to enrich the user data for Amazon Pinpoint campaigns. For more information, see Using Amazon Pinpoint Analytics with Amazon Cognito User Pools (p. 114).	September 26, 2017
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 Adding Groups to a User Pool (p. 127) and Role-Based Access Control (p. 208).	December 15, 2016
Documentation update	Updated iOS code examples in Developer Authenticated Identities (Identity Pools) (p. 241).	November 18, 2016

Change	Description	Date
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 (p. 124).	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 with 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 From a CSV File (p. 134).	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. 19).	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 Providers (Identity Pools) (p. 240).	June 23, 2016
CloudTrail integration	Added integration with AWS CloudTrail. For more information, see Logging Amazon Cognito API Calls with AWS CloudTrail (p. 318).	February 18, 2016

Change	Description	Date
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. 282).	April 9, 2015
Data stream to Amazon Kinesis	Provides control and insight into your data streams. For more information, see Amazon Cognito Streams (p. 281).	March 4, 2015
Push synchronization	Enables support for silent push synchronization. For more information, see Amazon Cognito Sync (p. 255).	November 6, 2014
OpenID Connect support	Enables support for OpenID Connect providers. For more information, see Identity Pools (Federated Identities) External Identity Providers (p. 219).	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 (Identity Pools) (p. 241).	September 29, 2014
Amazon Cognito general availability		July 10, 2014